Refonte et industrialisation logicielle de la solution ExpoTouch

Antoine Viau

le cnam

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

PARIS

Mémoire présenté en vue d'obtenir le

DIPLOME D'INGENIEUR CNAM

Spécialité : INFORMATIQUE

Option : Architecture et Ingénierie des Systèmes et des Logiciels

Par

Viau Antoine

Refonte et industrialisation logicielle de la solution ExpoTouch

Soutenu le 3 novembre 2014

JURY

PRESIDENT :

Yann Pollet - Professeur Titulaire de la Chaire d'Intégration des systèmes au CNAM

MEMBRES :

Jean-Michel Douin - Maitre de conférence au CNAM)

Serge Romerduc - Enseignant-chercheur au CNAM

Pascal Graffion - Ingénieur en développement d'applications client/serveur Web au CNAM

Vincent Delannoy - Chef de projet, Tactilia

Bruno Bouchahoua - Ingénieur logiciel, Cap Gemini

Remerciements

Je remercie,

Aurore, pour m'avoir inspiré et supporté (dans tous les sens du terme) durant ces trois années de reprise d'études au CNAM ;

Ma mère et Jean-Pierre, pour leur soutien tant affectif que logistique ;

Mon père, pour m'avoir laissé utiliser son ZX Spectrum+ et me faire ainsi découvrir ma passion de toujours ;

Tous ceux qui ont fait et font le CNAM, permettant à chacun d'apprendre et avancer quelque soit sa situation ;

La société Tactilia qui m'a permis de réaliser ce mémoire, en particulier Benoît Chanclou et Vincent Delannoy ;

Jean-Michel Douin, Pascal Graffion, Eric Lee pour leurs cours et leurs encouragements ;

Je dédie ce mémoire à Mamie Régine et Mamie Simone.

Abréviations

AJAX	Asynchonous Javascript And XML
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
DAO	Data Access Object
DOM	Document Object Model
EDI	Environnement de développement Intégré
HTML	Hyper-Text Markup Language
HTTP	Hyper-Text Transfer Protocol
IHM	Interface Homme-Machine
IMM	Interface Machine-Machine
JSON	JavaScript Object Notation
MVC	Modèle-Vue-Contrôleur
ORM	Object Relationnal Mapper
SGBD	Système de Gestion de Bases de Données
SPA	Single Page Application
SQL	Standard Query Language
XML	eXtensible Markup Language

Sommaire

1. Présentation

1.1.La société Tactilia

1.1.1. Historique

Tactilia est une société rennaise fondée en 2010 par Benoît Chanclou, Fabrice Grange et Yann Jehanneuf, tous trois issus de l'INRIA. C'est une « Touch Agency » : elle est spécialisée dans les solutions pour matériels tactiles tels que les tablettes, téléphones, bornes, etc. Aujourd'hui son principal produit est FidBe, une solution logicielle et matérielle destinée à la récolte de données et la distribution de contenu marketing.

Parallèlement et historiquement, Tactilia fournit un service d'ingénierie pour toute entité désirant s'équiper en matériels et logiciels tactiles spécifiquement adaptés à leurs besoins. Son expertise s'est construite à partir de ces missions, notamment au travers d'applications sur mesure installées sur tablettes Apple (iPad 2 à 4) ou des tables tactiles.

On notera en particulier :

- Rue des Sciences;
- Cap 3000 ;
- Coca-Cola ;
- La solution ExpoTouch dont la transformation sera le sujet de ce mémoire.

1.1.2. Organisation

Tactilia est une start-up. A ce titre, sa structure est de taille réduite et son fonctionnement repose sur un fort dynamisme afin d'être le plus réactif sur un marché en pleine croissance. De fait, l'équipe est limitée aujourd'hui à neuf personnes aux compétences ciblées répartis sur trois pôles :

- Le pôle commercial, constitué d'une téléprospectrice et une attachée commerciale. Il gère avec la direction le marketing, l'offre commerciale et la relation client ;
- Le pôle développement est dirigé par le directeur technique, accompagné d'un chef de projets, d'une équipe de développeurs et d'un designer.
- Le pôle matériel est à la charge d'une seule personne qui s'occupe de trouver et gérer les meilleures solutions physiques de tablettes, de bornes, etc.

Le pôle de développement évolue en fonction des missions et fait régulièrement appel à des prestataires extérieurs. Le chef de projets gère les demandes des clients sur l'ensemble du cycle de développement ainsi que l'après-vente. Il est l'interlocuteur principal des développeurs et sert d'interface entre ceux-ci et les clients.

1.2.La solution ExpoTouch

1.2.1. Description fonctionnelle

ExpoTouch est une solution marketing à destination de coopératives d'artisans. Elle a été démarrée en février 2012 à l'initiative de la Coopérative des Professionnels Artisans du Bois (COPAB), elle-même membre de l'Organisation des Coopératives d'Achats pour les Artisans du Bâtiment (ORCAB). ExpoTouch permet à des conseillers, équipés de tablettes tactiles, de proposer les produits et services des adhérents des coopératives, de collecter des informations clients/prospects et de réaliser des devis. On trouve notamment :

- Une gestion de la rencontre entre un conseiller et un prospect (heure de début, de fin…) ;
- Un système de questions-réponses pour collecter des informations sur les prospects ;
- L'annuaire des adhérents de la coopérative ;
- Le catalogue des produits et services - catégorisés et éventuellement hiérarchisés - proposés par les adhérents ;
- Une interface de conception de devis qui permet d'intégrer des produits, des documents, informations libres, tarifs, etc.

L'ensemble des données entrantes et sortantes des tablettes sont stockées sur un serveur distant. L'essentiel du contenu (catalogue, documents…) est issu d'une synchronisation du serveur vers les tablettes. Pour limiter la quantité transférée, certaines données sont envoyées à la demande. Enfin, les informations collectées par les conseillers sont renvoyées au serveur en temps réel, puis consolidées et exploitées à des fins commerciales et marketing.

La COPAB a été la première demandeuse. Mais avant même la mise en production elle a été suivie par Vendée Sani Therm (VST) et l'Union des Artisans du Bois (UAB). ExpoTouch est donc actuellement en production pour ces trois coopératives.

Il est important de noter que la solution n'est pas une licence : chaque coopérative s'est vue construire un ExpoTouch sur mesure. Toutes les solutions ont une base technologique,

tant côté serveur que côté client, mais elles sont trop distinctes pour permettre une maintenance commune.

1.2.2. Approche technique

Chaque coopérative dispose de sa propre Direction des Systèmes d'Informations (DSI). Les DSI récoltent les données actualisées du catalogue et les éléments marketing qu'ils préparent à l'intégration dans ExpoTouch. Ce processus se déroule en de nombreuses étapes :

- Les données sont converties au format JSON selon la nomenclature exigée par ExpoTouch ;
- Les différents fichiers médias (images, vidéos, PDF) sont rassemblés dans une arborescence qui respecte elle aussi une nomenclature ;
- L'ensemble est déposé via FTP sur un serveur ;
- Une interface Web sécurisée permet d'enclencher l'importation des données au sein de la solution ;
- Cette même interface permet de consulter et manipuler les données si nécessaire ;
- Toujours dans l'interface, on procède à l'exportation des données et fichiers vers les tablettes. Il s'agit en fait d'une transformation des données et d'un transfert de dossiers pour une mise à disposition auprès des tablettes ;
- Les tablettes récupérent les données et les fichiers sur le serveur en passant par une synchronisation différentielle afin de limiter la quantité à transférer.

Les conseillers disposent alors d'un catalogue de produits et d'outils de prospection à jour. Ils s'en servent comme conseil de vente mais surtout pour collecter des informations. Ces dernières sont envoyées et récupérées en temps réel sur un serveur Web. Par exemple, un conseiller peut consulter les précédents devis d'un client et lui en proposer un nouveau. Le traitement et l'exploitation de ces données passent essentiellement par la consultation directe de la base de données sous-jacente. Une interface minimale est disponible afin de « décorer » cette consultation, mais elle n'est pratiquement pas utilisée car elle ne permet pas la visualisation des données autrement que sous forme de listes.

1.3. Projet : ExpoTouch 2

La solution a prouvé son intérêt et son efficacité : elle peut donc passer à une nouvelle étape. Plusieurs facteurs techniques et économiques ont déclenché le lancement d'une nouvelle version plutôt qu'une simple évolution. Techniquement, l'accroissement de la quantité de données et d'utilisateurs commence à montrer les limites de la version actuelle du côté serveur :

- Le processus d'importation est devenu trop lent : ce qui prenait auparavant quelques minutes commence à se compter en heures ;
- Il n'y a aucune sécurité au niveau de l'intégrité des données. Une mauvaise manipulation, un problème de transmission ou perte de cohérence des données (plantage) impliquent un processus de recouvrement complexe et non garanti ;
- Les données ne sont manipulables que par des requêtes manuelles dans la base de données. L'interface actuelle est complexe car inadaptée, et ne fournit pas suffisamment de fonctionnalités pour accéder aux données.

La partie cliente est satisfaisante en l'état. Il n'est pas donc pas prévu de modifications des logiciels de tablettes. C'est un allègement de la charge de travail mais aussi une contrainte. La future version devra être compatible avec le protocole d'échange existant tout en améliorant les performances, la sécurité et l'intégrité des données.

L'organisation des coopératives (ORCAB) est satisfaite de la solution et veut l'étendre à de nouvelles coopératives. Mais le coût d'une version spécifique pour chacune d'elle est un frein majeur à cette extension. L'idée est donc de transformer ExpoTouch en un produit unique, configurable et distribuable sous forme de licence d'utilisation. Les coopératives seront alors en mesure de mutualiser les coûts de cette transformation ainsi que de sa maintenance et son évolution.

On va donc s'engager dans une refonte technique et fonctionnelle. Ce mémoire va en décrire les étapes et justifier les choix qui ont été faits :

- Tout d'abord le descriptif des objectifs : une nouvelle version est à la fois la fabrication d'un nouveau logiciel et le maintien de fonctionnalités existantes. Il va donc falloir définir le périmètre de transformations et celui des nouveautés, sachant que l'on cherche aussi à améliorer le processus de conception et de maintenance ;

- Ensuite, nous verrons la conduite du projet : à partir d'outils connus et reconnus nous verrons l'analyse fonctionnelle et l'architecture ;
- La conception sera alors passée en revue : cette phase implique les choix technologiques et méthodologiques ;
- Enfin, la réalisation nous plongera au cœur de l'industrialisation de l'ingénierie logicielle de ce projet.

2. Evaluation de l'existant et objectifs

2.1. Remise à plat du projet

2.1.1. Maintenance et évolutivité logicielle

Le développement de la première version d'ExpoTouch est intimement lié à la société : il a été un des fers de lance en termes de produits et d'exposition des capacités de Tactilia. Il s'est construit par additions successives de divers intervenants, en particulier un prestataire indépendant en charge de la partie serveur. Son projet est basée sur une solution mature de l'époque : Drupal 7.

Drupal est un système de gestion de contenu écrit en PHP (Drupal s.d.). Il se destine autant aux intégrateurs et designers qu'aux développeurs. Cette casquette multiple permet de construire rapidement des sites web, puis de les faire évoluer ou adapter aux besoins de façon précise. Il repose sur un modèle très visuel : notion de blocs, polices de caractères, thèmes graphiques, etc. Le contenu est stocké dans une base de données dont le modèle est opaque. L'accès aux données et l'extension pour des besoins spécifiques se font via une API qui recense environ 4000 fonctions.

Drupal a été bâti sur PHP 4, c'est-à-dire sur un langage purement procédural. En conséquence, son exploitation nécessite une rigueur propre au projet afin de palier à l'absence de l'orienté-objet : respect d'une nomenclature, hiérarchie de fonctions, conteneurs de données… De fait, la maintenance devient vite complexe : il faut être un spécialiste Drupal pour en comprendre la mécanique interne, appréhender sa manipulation et surtout la façon de l'exploiter de façon cohérente.

D'autre part, Drupal est une base monolithique pour un projet tel qu'ExpoTouch. On ne peut pas le scinder en plusieurs modules et n'utiliser que ce dont on a besoin. En conséquence, les « points faibles » du système sont démultipliés et moins contrôlables. De même, la mise à jour de Drupal (correction de bug, sécurité) peut avoir des conséquences lourdes et révéler de mauvaises surprises.

Enfin, la gestion des données par Drupal est totalement opaque : les bases de données sous-jacentes ne sont pas faites pour être manipulées autrement que par Drupal lui-même ou par son API. En conséquence, tout modèle de données qui sort de l'approche proposée par le canevas est très complexe à mettre en œuvre.

Drupal était conforme au développement pour un client unique dans un périmètre délimité. Mais le succès d'ExpoTouch auprès de nouvelles coopératives a rendu la solution de plus en plus exigeante et rendu Drupal de moins en moins adapté. De fait, l'évolution d'ExpoTouch s'est vue ralentie tant au niveau technique que fonctionnel. Chaque opération sur la solution devenait de plus en plus hasardeuse, et les nouvelles fonctionnalités de plus en plus complexes à mettre en œuvre.

L'objectif du point de vue de la maintenance et de l'évolutivité est de quitter le canevas Drupal. Il s'agit désormais de mettre en place une solution plus légère et plus adaptative. Comme nous le verrons, cela passera par une délégation technique moins laxiste, et une approche plus fine de la conception détaillée dans le cycle en V du projet. L'objectif d'évolutivité découle directement de celui de maintenance. Si le projet est suffisamment souple, simple à appréhender et à manipuler, son évolution en sera plus facile et plus performante. Pour ce faire, on cherchera des solutions éprouvées, notamment du côté des patrons de conceptions et la standardisation.

2.1.2. Performances

Le modèle de données de Drupal repose sur une granularité très fine et des spécifications orientées vers la gestion de contenu :

- Le modèle de données est en fait un méta-modèle : on ne dispose pas d'entités, mais de descriptions d'entités. Associé à la description, il y a bien évidemment le contenu ;
- Le méta-modèle suit un schéma orienté vers la publication de contenu pour le Web. Il y a notamment la notion de nœuds hiérarchiques (les *nodes*) à laquelle s'attache des notions de droits, de types, etc. ;
- Par ailleurs, il intègre une notion d'historique pour chaque donnée. Tout le contenu est dupliqué afin de maintenir des révisions.

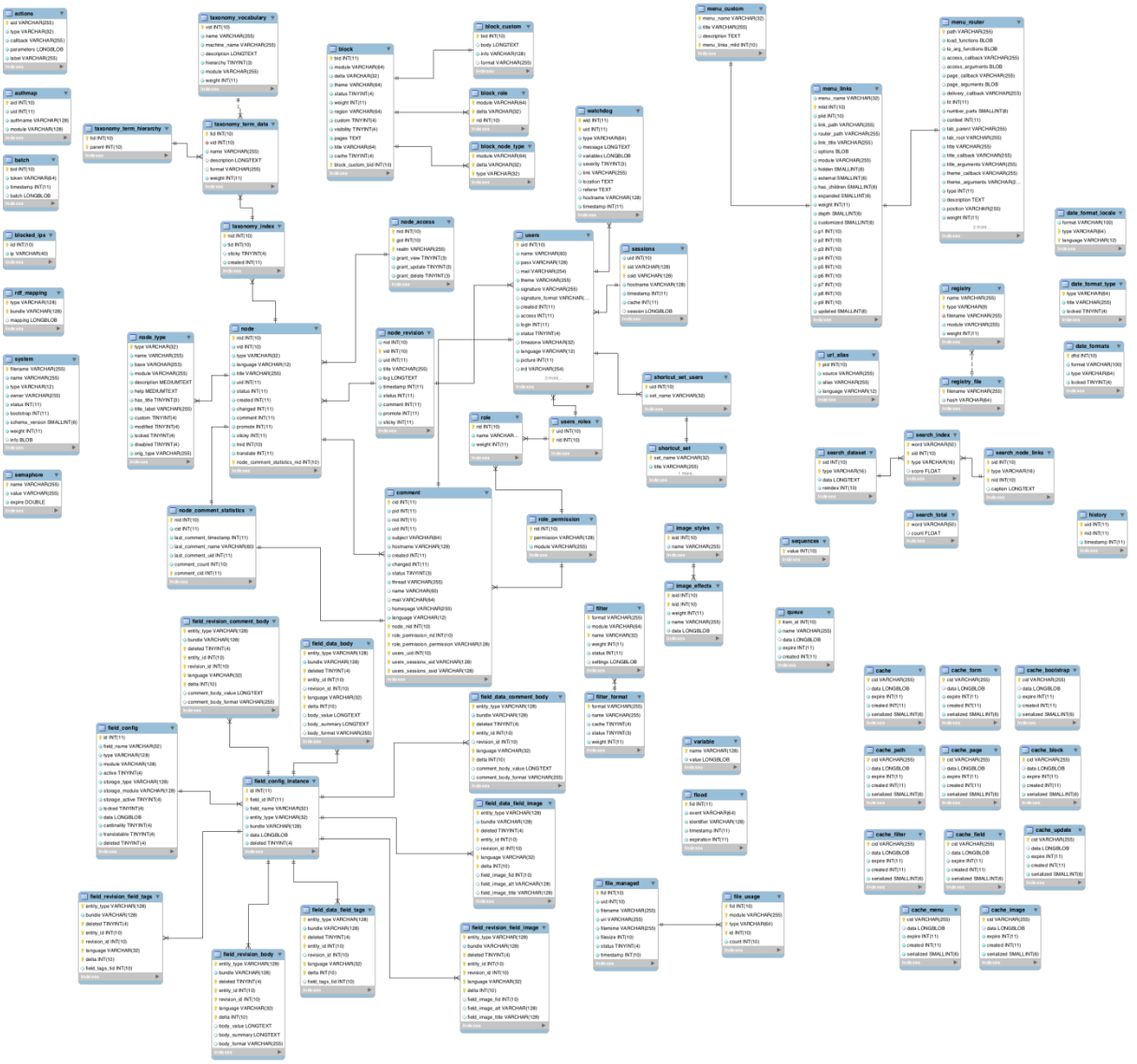


Figure 2-1 : représentation du schéma de Drupal 7 (Drupal Database Schema s.d.)

Comme on le voit dans la figure 2-1, il en résulte un schéma extrêmement chargé et totalement abstrait que seul Drupal et son API peuvent manipuler. La généricité du modèle, doublé de ses capacités d'historiques, provoquent un goulet d'étranglement majeur tant pour les opérations d'écriture que de lecture. Le problème s'est fait sentir avec l'accumulation de données au fil du temps : de quelques minutes, le processus est passé à plus d'une heure.

Coopérative	Nombre d'enregistrements	Stockage	Temps d'importation
UAB	Environ 15000	470 Mo	De 30 minutes à 1 heure
COPAB	Environ 5000	950 Mo	De 10 à 30 minutes

Tableau 1 : tailles et performances selon deux coopératives

La performance devient donc un objectif important dans la nouvelle version d'ExpoTouch. On ne pourra pas se contenter d'augmenter les capacités matérielles : les coopératives ont chacune leur propre serveur et on ne peut pas leur demander de nouveaux investissements dans ce domaine. Toutefois, comme nous le verrons, l'emploi de meilleures ressources matérielles n'est pas une solution à exclure, notamment par l'exploitation de la virtualisation qui permet une optimisation maximale des coûts et de la maintenance. Cette approche sera une fonctionnalité supplémentaire dans la proposition commerciale de la solution issue de sa transformation en licence.

L'objectif de performance passera donc par une remise à plat du modèle de données et de la chaîne de traitement. On utilisera des métriques et des tests afin de s'assurer des gains obtenus et de la pérennité des choix.

2.1.3. Intégrité des données

Comme nous l'avons vu, le processus d'intégration des données à destination des tablettes est particulièrement lourd. Chaque étape est un point de faiblesse dans la chaîne de traitement et rien n'a été prévu pour pallier à un éventuel défaut de fonctionnement. L'opacité de la solution actuelle ne permet pas de déterminer facilement et rapidement les sources de problèmes. Il va donc falloir mettre en place une nouvelle chaîne qui garantisse une reprise après accident avec un suivi précis des événements.

2.1.4. Compatibilité descendante

La partie cliente est indépendante car gérée par un prestataire externe. Il n'est pas directement concerné par la refonte et la nouvelle version doit être compatible avec le client actuel. De même, les coopératives ne veulent pas changer pour l'heure de procédure. On gardera donc une « façade » extérieure identique à l'existant, autant du côté fonctionnel (procédures d'importation/exportation) que technique (accès aux bases de données).

2.2.Factorisation

Les trois coopératives ont reçu chacun un produit issu d'une même technologie, mais avec des développements spécifiques. Ces spécificités se retrouvent – en théorie – dans des extensions de Drupal. En pratique, ces extensions sont un mélange de code dédié avec des appels à l'API Drupal, le tout sans nomenclature particulière. La maintenance en devient redondante : un problème trouvé sur l'un doit être reporté sur les autres. De même pour l'évolutivité, les coopératives faisant partie de la même organisation, l'une peut proposer de nouvelles

fonctionnalités que les autres pourront vouloir intégrer dans leur version de la solution. On se retrouve alors à dupliquer du code, des processus, documentations, etc.

L'objectif est donc de construire une solution unique qui peut être déployée au cas par cas avec des spécificités cadrées et configurables. Il s'agit de l'application du principe d'ouverture pour extensions et fermeture pour modifications (*open-close principle*), habituellement mis en œuvre sur les classes en programmation orientée objet (Meyer 1988) :

- On ne doit pas avoir à modifier le code de la classe ;
- La classe doit être en mesure d'être étendue (héritage, injection de dépendances, etc.).

Dans notre cas, la mise en œuvre de ce principe portera sur l'ensemble du code, mais aussi sur l'architecture. Cela impliquera bien évidemment des règles et des contraintes : on ne peut pas transformer une solution marketing en un traitement de texte !

2.3.Industrialisation de l'ingénierie logicielle

Le développement d'ExpoTouch s'est fait par couches successives avec des intervenants divers, extérieurs et/ou intérieurs à la société. Chaque intervenant a apporté ses méthodes et ses approches. Il en résulte des strates hétérogènes sur tous les aspects du projet : code, intégration, gestion des données, etc.

L'objectif est de mettre en place une chaîne complète de conception-réalisation-production-maintenance qui respecte un canevas clairement défini et justifié. Le projet doit pouvoir être transmis et maintenu via une industrialisation de l'ingénierie logicielle. On travaillera sur tous les aspects : écriture du code, environnement de développement, gestion des sources, processus d'intégration, indicateurs de performances, indicateurs de fiabilité, etc.

Cette démarche doit servir au reste de la société afin d'améliorer la production sur les différents projets.

3. Conduite de projet et analyse

3.1. Recherche et analyse des solutions technologiques

L'un des objectifs étant l'industrialisation logicielle, on commencera par une phase de recherche et d'évaluation des technologies. L'idée n'est pas seulement de trouver celles qui sont le plus adaptées au projet, mais aussi de maîtriser celles à même d'être réutilisées sur d'autres projets, actuels ou futurs. Il ne s'agit donc pas de simplement choisir des technologies sur un catalogue de fonctionnalités : la compréhension de leur mécanique interne et l'analyse de leur champ d'action sont indispensables. Ainsi, on sera à même de les employer de façon optimale, et de les transmettre au sein de l'entreprise avec un bagage suffisant pour gérer leur exploitation dans la plupart des cas de figures.

La recherche de technologies passe par deux critères :

- Privilégier les solutions *open-source* : outre une affaire de coût initial (pas de licence à payer), l'accès au fonctionnement interne est possible et, surtout, est documenté ;
- La validation par des « preuves de concept » *(proof of concept* – POC). Il s'agit de mettre les technologies choisies à l'épreuve par la création de briques logicielles relativement simples. Cela peut se faire par tests unitaires, des mini-sites, scripts, etc.

En procédant de la sorte, on dispose d'une certaine visibilité sur les technologies et on peut évaluer leur adéquation avec les besoins immédiats et futurs. De plus, la volonté d'une connaissance approfondie peut mener à une expertise dans un domaine et des technologies choisis. L'entreprise est alors en mesure de vendre des prestations avec un haut degré de spécialisation.

Le principal inconvénient est évidemment l'aspect chronophage et non directement productif de cette phase de recherche. D'où l'importance de mettre en place des POC itératifs qui permettent une évaluation progressive et rapide : on établit le degré d'adéquation par rapport aux objectifs, la complexité de la mise en œuvre en fonction des étapes, la quantité et qualité de la documentation.

3.2.Méthodologie projet

Etant donné que la solution existe déjà, la mise en place des spécifications est surtout une question de formalisation (inexistante au départ). Il y a très peu d'interactions avec les clients dans un premier temps : de leur point de vue on cherche à obtenir une solution identique à l'existant. Pour cette raison, le cycle en V est adapté pour la conduite du projet.

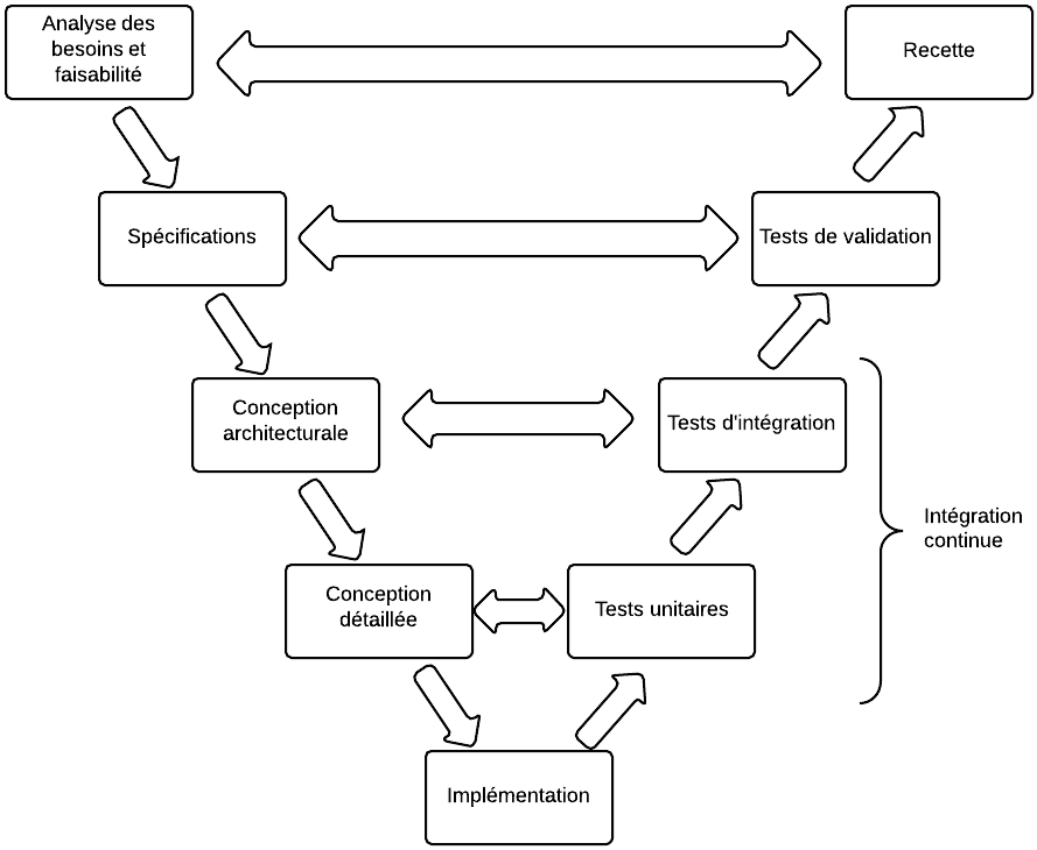


Figure 3-1: cycle en V

- **Analyse des besoins de faisabilité** : les besoins sont les mêmes que la version 1 d'ExpoTouch et l'environnement de production ainsi que les acteurs sont connus. La définition du périmètre est donc assez claire.
- **Spécifications** : il s'agit de ne pas changer fondamentalement l'expérience utilisateur des clients. En revanche, on va chercher à l'optimiser et l'améliorer.
- **Conception architecturale** : l'exploitation de la virtualisation jouera un rôle important dans cette étape. Toujours dans la logique d'optimisation, il va falloir établir une architecture qui restera compatible avec celles de toutes les solutions déjà en production.

- **Conception détaillée** : c'est la section critique de ce projet. Le cœur de la solution va être totalement revu, avec un objectif de maintenance et d'évolution sur le long terme.
- **Codage** : cette étape doit être particulièrement soignée car elle consiste en une mise en œuvre des étapes précédentes, mais surtout elle est celle qui se transmet le plus directement d'un intervenant à l'autre. Il est donc indispensable d'établir un ensemble de règles et de méthodes que chacun sera à même de comprendre et réemployer.

En remontant le cycle en V, on valide chaque étape précédente :

- **Tests unitaires** : ils jouent un rôle fondamental pour garantir la validité du codage et de la conception détaillée. Ils serviront aussi à assurer une couverture de code maximale.
- **Tests d'intégration** : on parlera plutôt d'intégration continue. Les capacités actuelles de virtualisation permettent de créer des environnements de tests rapidement et à moindre de coût. On les exploitera pour établir une chaîne qui va des tests unitaires à une procédure de mise en production, suivis par des processus de validation automatisés et manuels.
- **Tests de validation** (ou tests systèmes) : il s'agit de la validation de la mise en production.
- **Recette** (ou tests d'acceptation) : cela consiste en la validation des spécifications.

3.3.Expression des besoins

Les processus sont connus mais imparfaits. En conséquence, refaire l'existant ne signifie pas le copier mais l'améliorer. Et cela commence par une formalisation. Dès lors, cette partie se basera sur les outils d'Arrington (CNAM s.d.), en particulier les cas d'utilisation. Ceux-ci décrivent les fonctionnalités existantes et attendues. Ils sont réunis dans des sous-systèmes indépendants et montrent les relations entre des acteurs (personnes, machines, logiciels…) et les fonctionnalités. Un diagramme permettra de comprendre en un coup d'œil tout ou partie des fonctionnalités, et pour chaque cas d'utilisation on trouvera un descriptif organisé dans un tableau qui précise :

- Le nom du cas d'utilisation ;
- Un résumé ;
- Les acteurs en jeu ;
- Les pré-conditions nécessaires à la bonne marche du cas d'utilisation ;
- Une description ;
- Les exceptions possibles qui peuvent survenir (typiquement : des erreurs) ;
- Les post-conditions qui définissent ce qui est attendu à la fin du cas d'utilisation.

3.3.1. Acteurs

- Administrateur Tactilia : représente la personne qui gérera les clients utilisateurs (administrateurs des coopératives) de la solution ExpoTouch 2 ;
- Administrateur : personnel de la coopérative en charge de l'ensemble des tablettes en fonctionnement. Il s'occupera notamment de « pousser » les nouvelles données vers les tablettes. Par la suite, il sera en mesure de consulter les informations retournées par les tablettes après exploitation par les conseillers.
- Tablette : c'est l'appareil mobile employé par les conseillers pour présenter le catalogue aux clients. C'est aussi le périphérique d'entrée de nouvelles données qui seront remontées vers le serveur. On considère qu'à un conseiller correspond une tablette.

3.3.2. Système et sous-systèmes

Le système représente l'ensemble de la solution ExpoTouch. Nous le décomposons d'abord en sous-systèmes aux fonctionnalités distinctes.

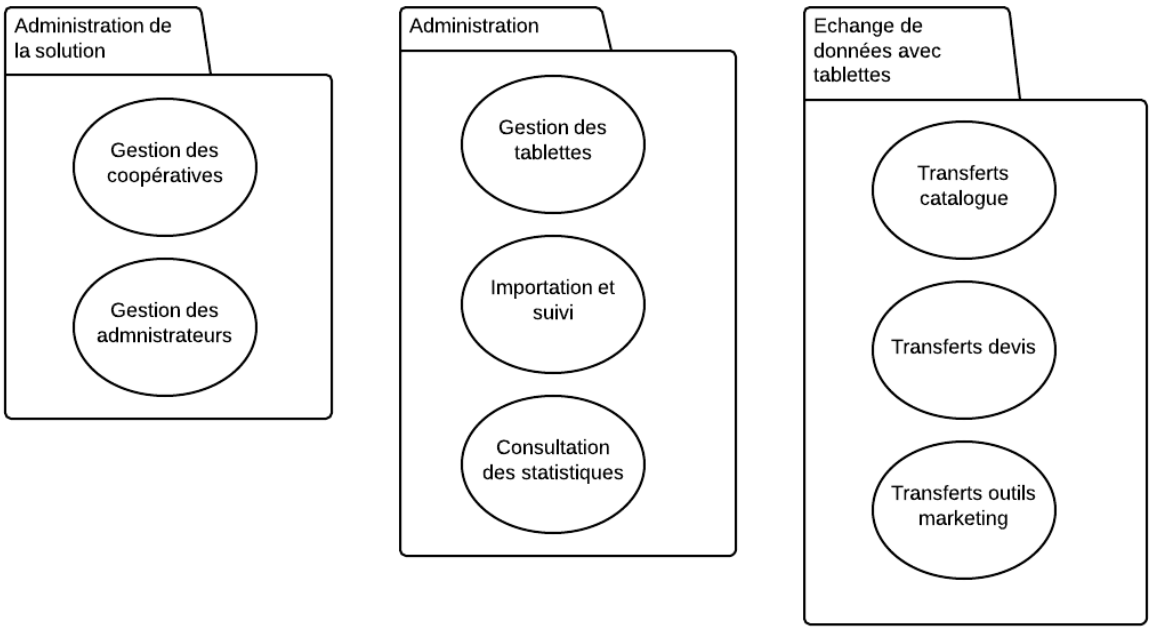


Figure 3-2 : sous-systèmes

3.3.3. Sous-système : administration de la solution

Dans l'optique d'une transformation en licence de la solution, Tactilia doit être en mesure de gérer plusieurs coopératives clientes. Chacune d'entre elles doit pouvoir gérer des administrateurs qui auront accès aux fonctionnalités d'importation, gestion, consultation, etc.

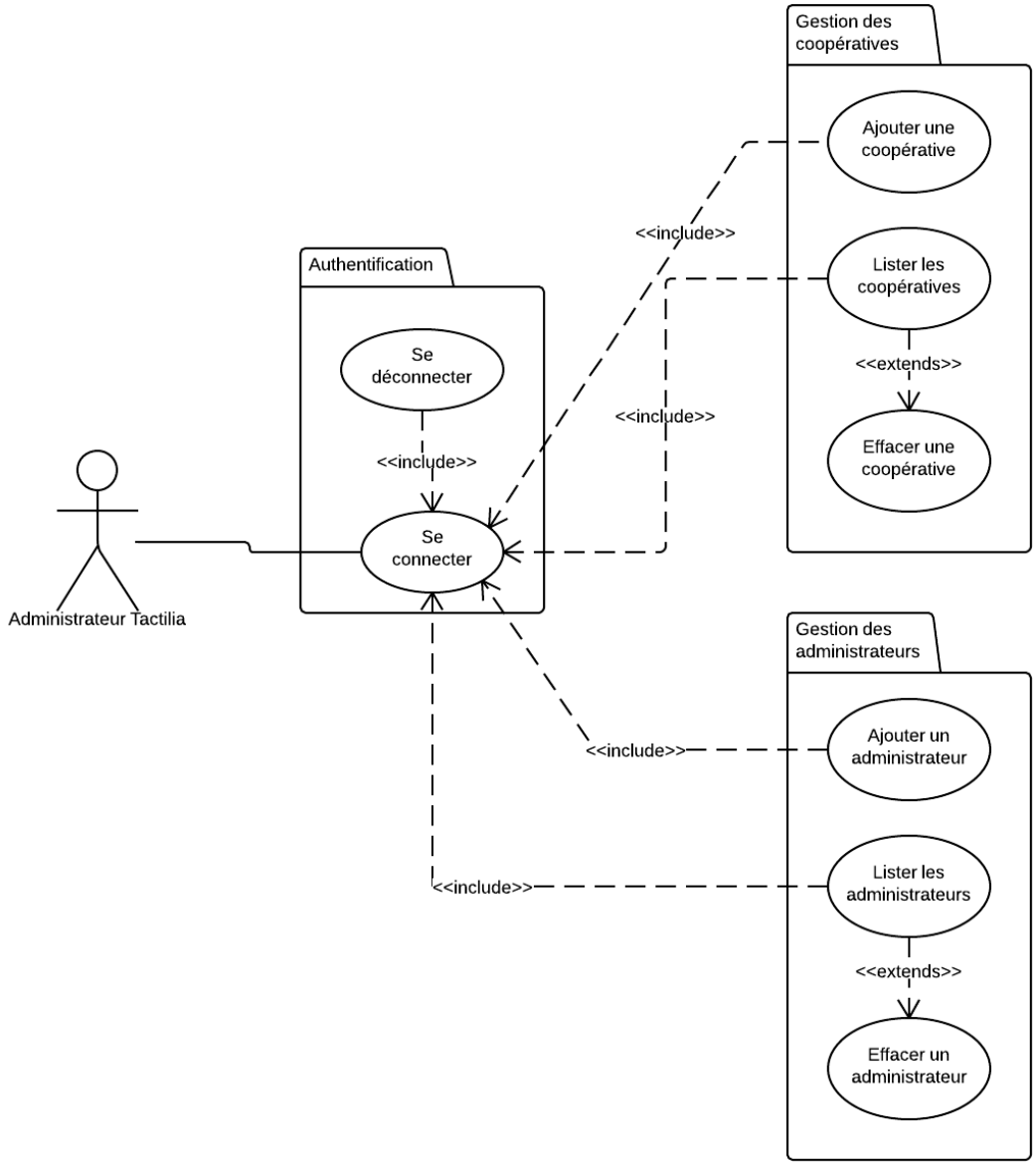


Figure 3-3 : cas d'utilisations de l'administration de la solution

3.3.3.1. Ajouter une coopérative

Nom	**Ajouter une coopérative**
Résumé	Permet à un administrateur Tactilia d'ajouter une coopérative dans ExpoTouch
Acteurs	Administrateur Tactilia
Pré-conditions	L'administrateur Tactilia doit être authentifié et connecté
Description	1 L'administrateur Tactilia indique le nom d'une coopérative. 2 Le système ajoute la nouvelle coopérative.
Exceptions	1a Les informations entrées par l'administrateur Tactilia sont invalides (taille du nom, caractères autorisés) : le système affiche le message correspondant à l'erreur. 1b La coopérative est déjà présente dans le système.
Post-conditions	La coopérative est créée dans le système, l'administrateur Tactilia est informé.

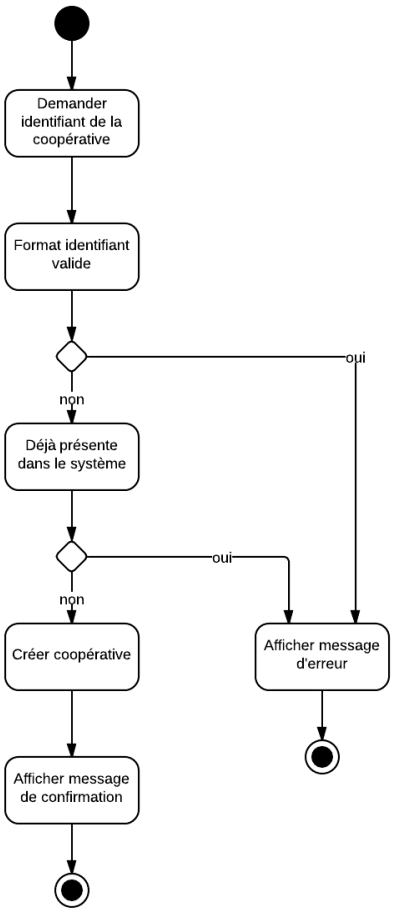


Figure 3-4 : diagramme d'activité « Ajouter une coopérative »

3.3.3.2. Lister les coopératives

Nom	**Lister les coopératives**
Résumé	Permet à un administrateur Tactilia de voir la liste des coopératives dans le système.
Acteurs	Administrateur Tactilia
Pré-conditions	L'administrateur Tactilia doit être authentifié et connecté
Description	Le système affiche la liste des coopératives.
Exceptions	Aucune
Post-conditions	La coopérative est créée dans le système, l'administrateur Tactilia est informé.

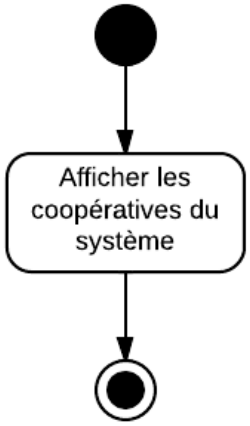


Figure 3-5 : diagramme d'activité « Lister les coopératives »

3.3.3.3. Effacer une coopérative

Nom	**Effacer une coopérative**
Résumé	Permet à un administrateur Tactilia de supprimer une coopérative d'ExpoTouch
Acteurs	Administrateur Tactilia
Pré-conditions	L'administrateur Tactilia doit être authentifié et connecté, la coopérative doit être présente dans le système.
Description	L'administrateur Tactilia efface la coopérative qu'il a sélectionnée.
Exceptions	Aucune
Post-conditions	La coopérative est effacée du système.

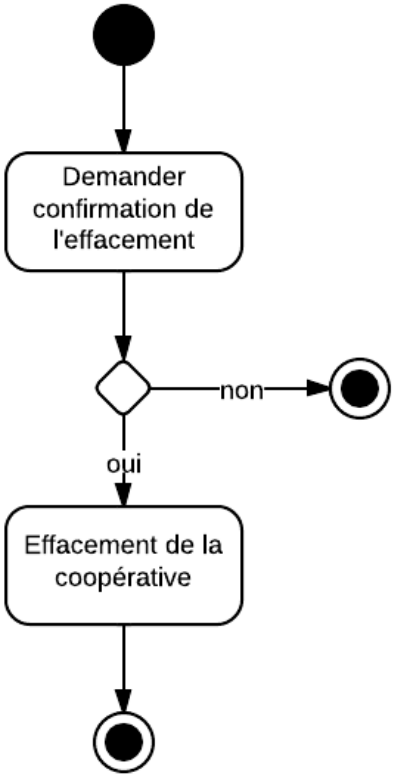


Figure 3-6 : diagramme d'activité « Effacer une coopérative »

3.3.3.4. Ajouter un administrateur

Nom	Ajouter un administrateur
Résumé	Permet à un administrateur Tactilia d'ajouter un administrateur ExpoTouch
Acteurs	Administrateur Tactilia
Pré-conditions	L'administrateur Tactilia doit être authentifié et connecté
Description	1 L'administrateur Tactilia indique l'e-mail de l'administrateur et un mot de passe 2 Le système ajoute le nouvel administrateur
Exceptions	1a Les informations entrées par l'administrateur sont invalides (format d'e-mail invalide, utilisateur inconnu du système) : le système affiche le message correspondant à l'erreur et permet de rentrer à nouveau les informations. 1b L'administrateur est déjà présent dans le système.
Post-conditions	L'administrateur est connecté. Il va sur la page tableau de bord de l'administration.

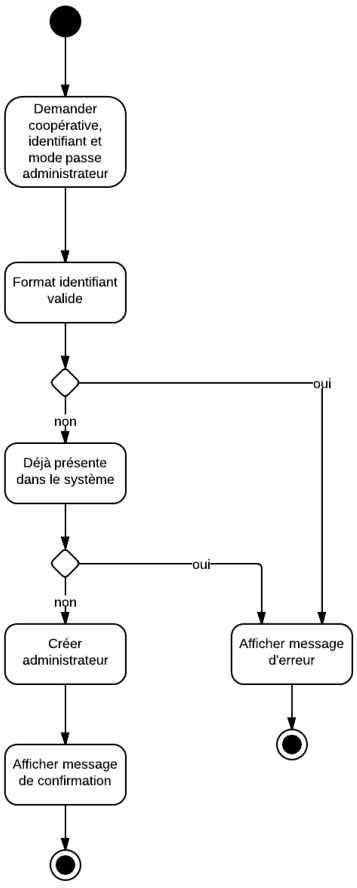


Figure 3-7 : diagramme d'activité « Ajouter un administrateur »

3.3.4. Sous-système : administration

Un administrateur doit être en mesure de pousser les données vers les tablettes et de consulter celles qui en sont remontées. Le sous-système dédié est bien évidemment sécurisé et implique donc une connexion authentifiée. La décomposition fonctionnelle de l'administration commence donc par l'authentification, qui permet de lancer l'importation des données, gérer les tablettes, et finalement consulter un jeu de statistiques complet.

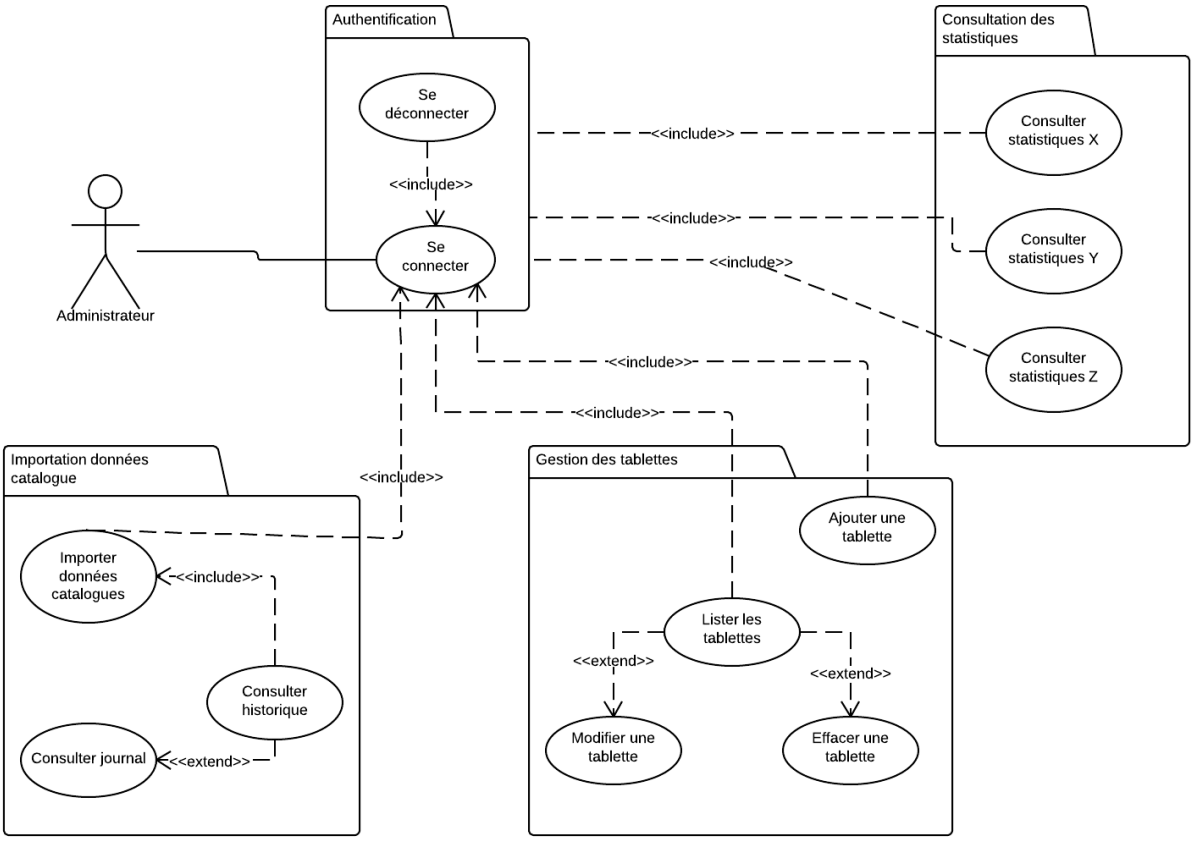


Figure 3-8 : cas d'utilisation de l'administration

3.3.4.1. Se connecter

Nom	Se connecter
Résumé	Permet à un administrateur de se connecter
Acteurs	Administrateur
Pré-conditions	L'administrateur doit exister dans le système (voir « Gestion des administrateurs »)
Description	1 L'administrateur indique un e-mail et un mot de passe correspondant. 2 Le système connecte l'administrateur.
Exceptions	Les informations entrées par l'administrateur sont invalides (format d'e-mail invalide, utilisateur inconnu du système) : le système affiche le message correspondant à l'erreur et permet de rentrer à nouveau les informations.
Post-conditions	L'administrateur est connecté. Il va sur la page tableau de bord de l'administration.

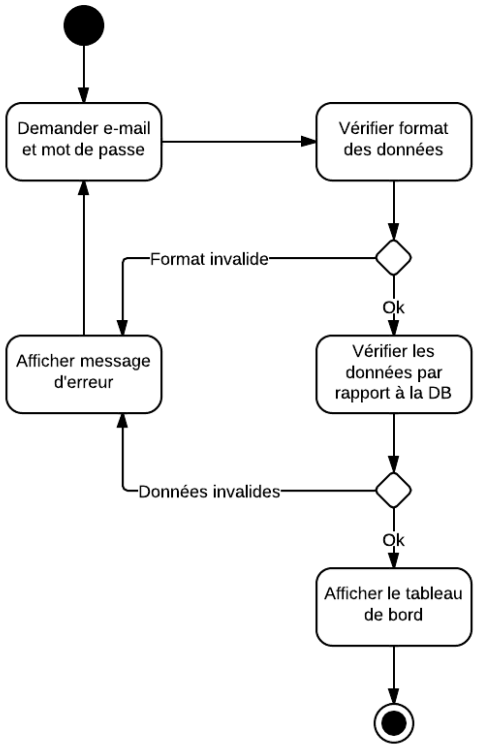


Figure 3-9 : diagramme d'activité « Se connecter »

3.3.4.2. Se déconnecter

Nom	Se déconnecter
Résumé	Permet à un administrateur de se déconnecter du système.
Acteurs	Administrateur
Pré-conditions	L'administrateur doit exister dans le système et être connecté.
Description	Le client indique au système qu'il veut se déconnecter. Le système applique la déconnexion.
Exceptions	Aucune
Post-conditions	L'administrateur est déconnecté.

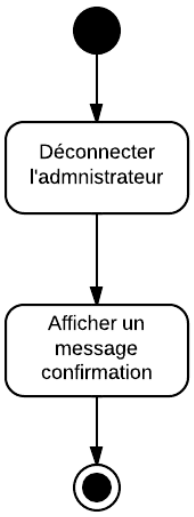


Figure 3-10 : diagramme d'activité « Se déconnecter »

3.3.4.3. Importer les données

Nom	Importer les données
Résumé	Permet à un administrateur de déclencher et suivre le processus d'importation des données (catalogue, outils marketing) dans le système.
Acteurs	Administrateur
Pré-conditions	L'administrateur doit être connecté, les données doivent être disponibles.
Description	Les données sont importées dans le système et viennent remplacer les données actuelles. Le déroulement est décrit à l'administrateur, puis un rapport est émis.
Exceptions	Des erreurs d'importations se produisent, le rapport d'erreur les précise, les données précédentes ne sont pas affectées.
Post-conditions	L'importation est faite, l'administrateur obtient un rapport.

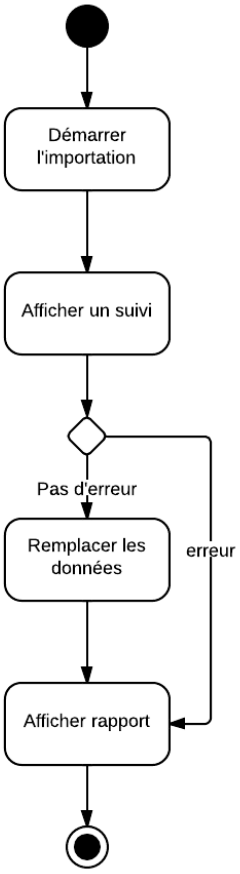


Figure 3-11 : diagramme d'activité « Importer les données »

3.3.4.4. Consulter l'historique des importations

Nom	Consulter l'historique des importations
Résumé	Permet à un administrateur de voir la liste des importations précédemment effectuées.
Acteurs	Administrateur
Pré-conditions	L'administrateur doit être connecté.
Description	Le système affiche la liste des précédentes importations.
Exceptions	Aucune
Post-conditions	La liste des précédentes importations est affichée.

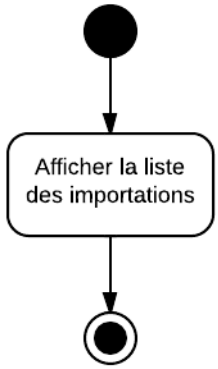


Figure 3-12 : diagramme d'activité « Consulter l'historique des importations »

3.3.4.5. Consulter le journal d'une importation

Nom	Consulter l'historique des importations
Résumé	Permet à un administrateur de consulter le journal d'exécution d'une importation
Acteurs	Administrateur
Pré-conditions	L'administrateur doit être connecté.
Description	Le système affiche le journal de l'importation
Exceptions	Aucune
Post-conditions	Le journal est affiché.

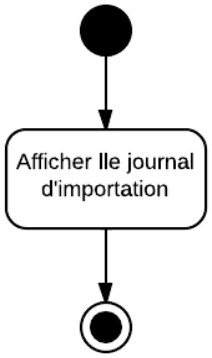


Figure 3-13 : diagramme d'activité « Consulter le journal d'une importation »

3.3.4.6. Ajouter une tablette

Nom	Ajouter une tablette
Résumé	Permet à un administrateur d'ajouter une tablette dans le système
Acteurs	Administrateur
Pré-conditions	L'administrateur doit être connecté.
Description	Le système ajoute la tablette avec les informations données par l'administrateur (login et mot de passe)
Exceptions	Les informations données par l'administrateur ne sont pas au bon format : un message d'erreur est affiché.
Post-conditions	La tablette est ajoutée.

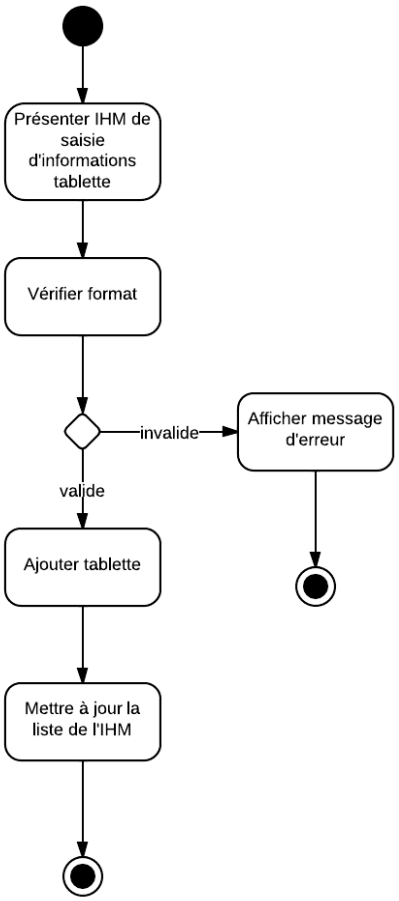


Figure 3-14 : diagramme d'activité « Ajouter une tablette »

3.3.4.7. Lister les tablettes

Nom	**Ajouter une tablette**
Résumé	Permet à un administrateur de consulter la liste des tablettes autorisées.
Acteurs	Administrateur
Pré-conditions	L'administrateur doit être connecté.
Description	Le système affiche la liste des tablettes.
Exceptions	Aucune
Post-conditions	Les tablettes sont affichées.

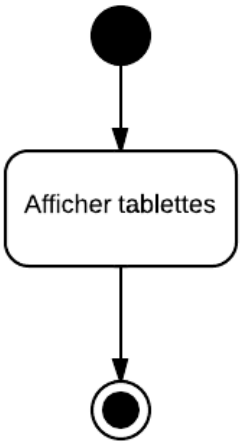


Figure 3-15 : diagramme d'activité « Lister les tablettes »

3.3.4.8. Modifier une tablette

Nom	Modifier une tablette
Résumé	Permet à un administrateur de modifier les caractéristiques d'une tablette dans le système
Acteurs	Administrateur
Pré-conditions	L'administrateur doit être connecté.
Description	Le système modifie la tablette avec les informations données par l'administrateur (login et mot de passe)
Exceptions	Les informations données par l'administrateur ne sont pas au bon format : un message d'erreur est affiché.
Post-conditions	La tablette est modifiée.

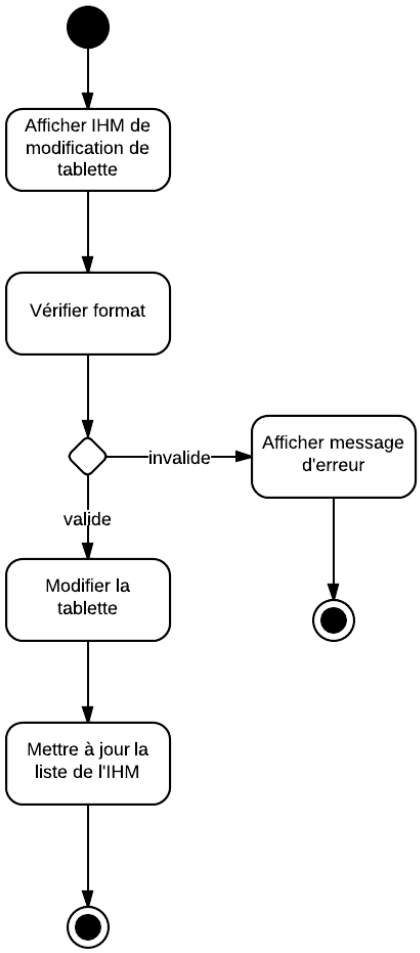


Figure 3-16 : diagramme d'activité « Modifier une tablette »

3.3.4.9. Effacer une tablette

Nom	Ajouter une tablette
Résumé	Permet à un administrateur de supprimer une tablette du système.
Acteurs	Administrateur
Pré-conditions	L'administrateur doit être connecté.
Description	Après demande de confirmation auprès de l'administrateur, le système supprime la tablette.
Exceptions	Aucune
Post-condition	La tablette est supprimée.

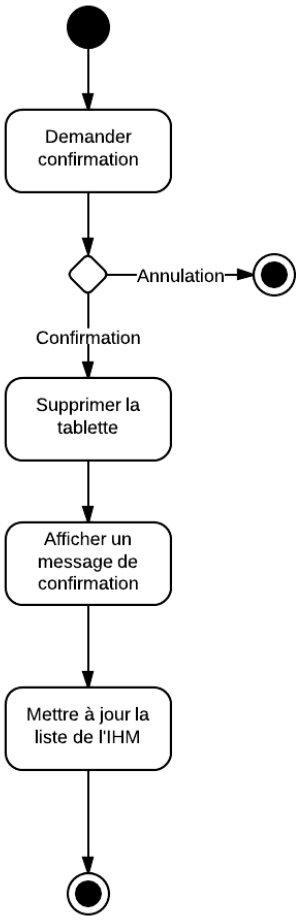


Figure 3-17 : diagramme d'activité « Effacer une tablette »

3.3.4.10. Consulter les statistiques

Nom	Consulter une statistique
Résumé	Permet à un administrateur de consulter une statistique
Acteurs	Administrateur
Pré-conditions	L'administrateur doit être connecté.
Description	L'administrateur choisit une statistique et les paramètres (période…), le système calcule et donne une vision graphique de la statistique demandée.
Exceptions	Aucune
Post-conditions	La statistique est affichée.

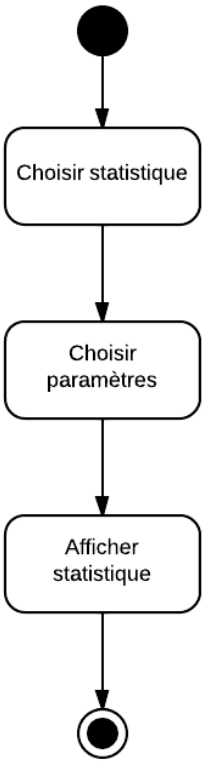


Figure 3-18 : diagramme d'activité « Consulter une statistique »

3.3.5. Sous-système : échange de données avec la tablette

Les tablettes reçoivent le catalogue des produits et services, ainsi que les outils marketing et remontent les données entrées par les conseillers ou les prospects. Il s'agit donc d'offrir un accès aux tablettes à la base de données du serveur autant en lecture qu'en écriture.

Cet accès est encapsulé dans une session authentifiée dont l'ouverture est le préalable nécessaire à toute opération.

On notera que l'acteur est bien la tablette et non pas le conseiller qui la manipule. En effet, les diverses opérations ne sont pas obligatoirement à l'initiative d'un conseiller. Elles peuvent l'être – et elles le sont actuellement - mais on préférera des opérations asynchrones découplées de tout critère de déclenchement unique.

Etant donné que les tablettes se servent du système comme d'un SGBD simplifié, les opérations sont de type CRUD et reproduisent donc un mode opératoire classique. Les diagrammes d'état sont donc les mêmes que l'on peut retrouver dans les cas d'utilisation précédents.

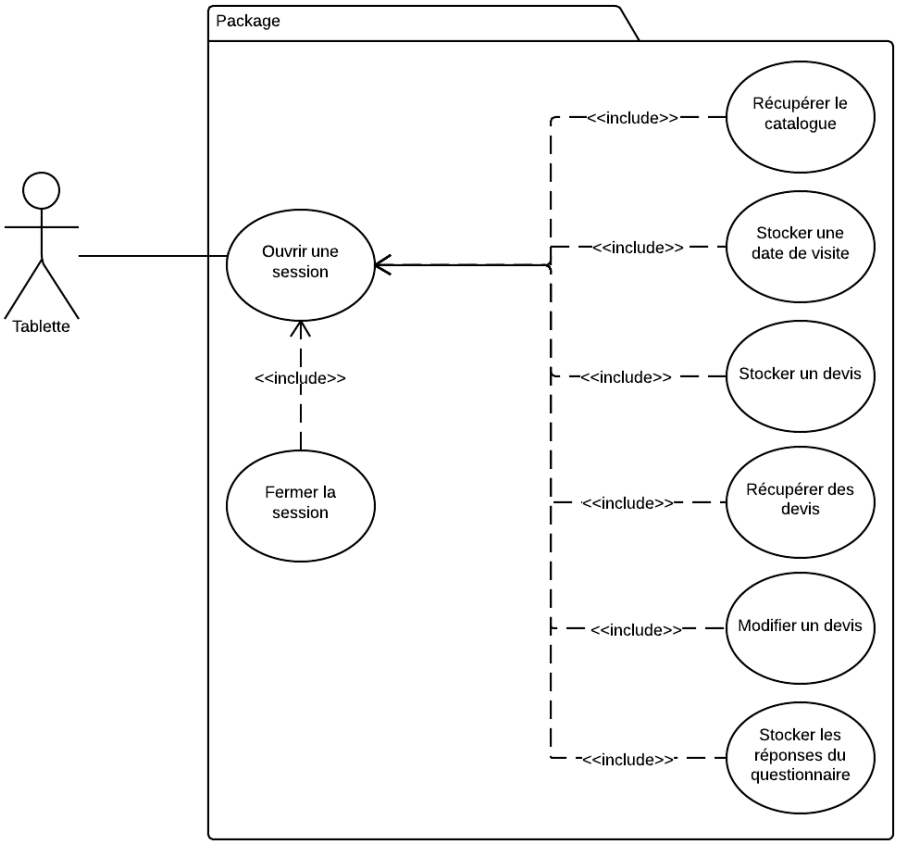


Figure 3-19 : cas d'utilisation des échanges de données avec la tablette

3.3.5.1. Ouvrir une session

Nom	Authentifier la tablette et ouvrir une session
Résumé	Permet à une tablette d'ouvrir une session authentifiée.
Acteurs	Tablette
Pré-conditions	La tablette doit exister dans le système.
Description	La tablette indique son login et mot de passe au système. Ce dernier ouvre une session.
Exceptions	Le login ou le mot de passe sont invalides, le système n'ouvre pas de session, la tablette est informée.
Post-conditions	La session est ouverte pour la tablette qui en est informé.

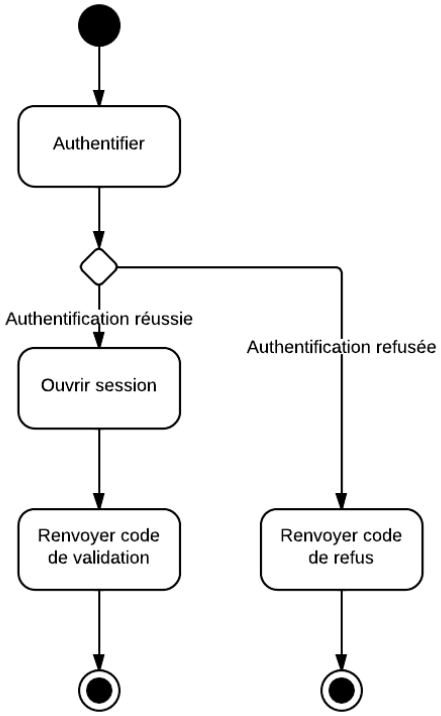


Figure 3-20 : diagramme d'activité « Authentifier la tablette et ouvrir une session »

3.3.5.2. Fermer la session

Nom	Fermer la session
Résumé	Permet à une tablette de fermer sa session
Acteurs	Tablette
Pré-conditions	La tablette doit avoir ouvert une session
Description	La tablette indique qu'elle veut clore la session, le système confirme.
Exceptions	Aucune
Post-conditions	La session est fermée.

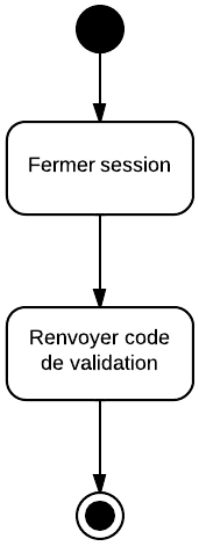


Figure 3-21 : diagramme d'activité « Fermer la session »

3.3.5.3. Récupérer le catalogue

Nom	**Importer les données dans la tablette**
Résumé	La tablette récupère le catalogue de produits et services ainsi que les outils marketing.
Acteurs	Tablette
Pré-conditions	La tablette doit s'être authentifiée, les données doivent être nouvelles et disponibles.
Description	La tablette télécharge les données.
Exceptions	
Post-conditions	La tablette dispose du catalogue, des outils marketing et les médias associés.

3.3.5.4. Stocker une date de visite

Nom	**Créer une date de visite**
Résumé	Permet de stocker la date de visite d'un client/prospect dans le système.
Acteurs	Tablette
Pré-conditions	La tablette doit avoir ouvert une session.
Description	Le conseiller indique sur la tablette la date et l'heure de visite d'un prospect/client. En validant l'information est stockée dans le système.
Exceptions	Le format des données est invalide. Le système renvoie une erreur.
Post-conditions	Les date et heure de la visite sont stockées dans le système.

3.3.5.5. Stocker un devis

Nom	Créer un devis
Résumé	Permet de stocker un devis dans le système.
Acteurs	Tablette
Pré-conditions	La tablette doit avoir ouvert une session et un devis doit avoir été créé.
Description	Le conseiller crée le devis et en le validant le stocke dans le système.
Exceptions	Le format des données est invalide. Le système renvoie une erreur.
Post-conditions	Le devis est stocké dans le système.

3.3.5.6. Récupérer les devis

Nom	Récupérer les devis
Résumé	Permet de récupérer les devis stockés dans le système.
Acteurs	Tablette
Pré-conditions	La tablette doit avoir ouvert une session.
Description	Le conseiller obtient la liste des devis stockés dans le système.
Exceptions	Aucune
Post-conditions	La liste des devis correspondants aux critères demandés est renvoyée à la tablette.

3.3.5.7. Modifier un devis

Nom	Modifier un devis
Résumé	Permet de modifier un devis déjà stocké dans le système.
Acteurs	Tablette
Pré-conditions	La tablette doit avoir ouvert une session, le devis doit exister.
Description	Le devis est modifié par le conseiller qui en le validant le stocke dans le système.
Exceptions	Si le devis n'existe pas ou que des données sont invalides, un message d'erreur est retourné par le système.
Post-conditions	Le devis modifié est stocké dans le système.

3.3.5.8. Stocker les réponses du questionnaire

Nom	Modifier un devis
Résumé	Permet de stocker les réponses du questionnaire proposé au client/prospect par le conseiller.
Acteurs	Tablette
Pré-conditions	La tablette doit avoir ouvert une session.
Description	Le conseiller utilise le questionnaire dans la tablette (qui lui est fourni par le système) auprès du client/prospect. En le validant il le stocke dans le système.
Exceptions	Si les données sont invalides, le système renvoie une erreur.
Post-conditions	Les réponses du questionnaire sont stockées dans le système.

3.4.Analyse orientée objets

3.4.1. Stéréotypes

Toujours en suivant la méthodologie Arrington, nous allons à présent voir la décomposition en objets de certains cas d'utilisation et le découpage en séquence des interactions entre ces objets. Dans l'analyse objets selon Arrington on distingue quatre types d'objets, dénommés les « stéréotypes » :

<<entity>> : ce sont les objets métiers, information durable et persistante ;

<<boundary>> : les objets à la frontière entre le système et un acteur ;

<<control>> : objets assurant la coordination d'autres objets ;

<<lifecycle>> : objets en charge des <<entity>>.

Les objets métiers du stéréotype <<entity>>, plus communément appelés « entités », sont destinés à représenter des objets concrets, transporter de l'information et être manipulés par les autres stéréotypes. Ils sont donc la première approche analytique des cas d'utilisation. ExpoTouch étant assez monolithique dans ses besoins, les entités ont toutes été définies au départ. On retrouvera donc dans la figure 3.22 la représentation d'un devis (Proposal), d'un conseiller (Interlocutor), d'un produit (Product), d'une famille de produits (Group), etc. Les liens entre entités précisent celles qui en agrègent (losange blanc) ou en composent d'autres (losange noir).

Les objets <<boundary>> représentent le plus souvent une interface homme-machine (IHM) ou machine-machine (IMM). Nous en avons plusieurs (application web, ligne de commande…), mais elles reposent sur le même fonctionnement, basé sur le patron de conception modèle-vue-contrôleur (MVC). Nous verrons lors de la phase de conception qu'un double usage du MVC permettra d'obtenir une IHM et une IMM exploitant la même architecture.

Les objets <<control>> contiennent la logique métier. Ils manipulent les entités et répondent au cœur des exigences des cas d'utilisation. Pour cela, ils communiquent avec les objets <<boundary>>, effectuent des traitements avec/sur les entités et éventuellement font appel aux objets <<lifecycle>>.

Les <<lifecycle>> ont en charge l'existence des entités dans le système et leur persistance. Ils doivent donc être en mesure de réaliser des opérations de type Création-Récupération-Mise à jour-Effacement (*Create-Read-Update-Delete* – CRUD).

Nous allons voir en premier un diagramme de classes décrivant les entités, puis trois diagrammes de séquences correspondants aux trois sous-systèmes encapsulant les cas d'utilisation.

3.4.2. Entités

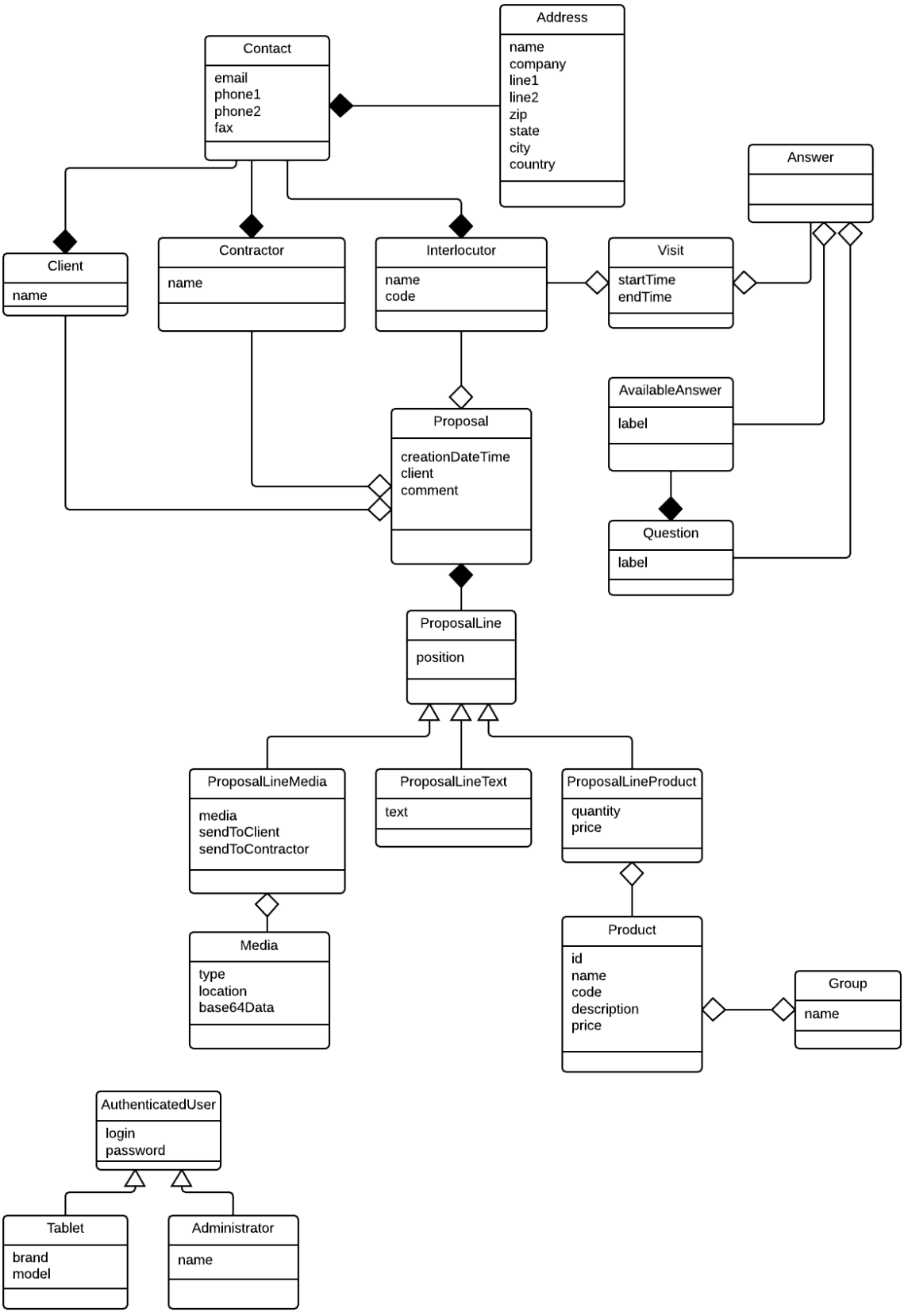


Figure 3-22 : diagramme de classes

3.4.3. Diagramme de séquence : ajout d'un administrateur

L'ajout d'un administrateur se fait depuis un outil en ligne de commande. L'objet <<boundary>> SuperUserCommand est en charge de gérer les entrées de la ligne de commande et d'afficher les messages d'informations. Le diagramme ci-dessous n'indique pas les exceptions, et on notera qu'un ajout réussi ne déclenche aucun affichage.

L'outil en ligne de commande sert autant à lister, ajouter ou supprimer des administrateurs. L'objet <<control>> SuperUserDispatcher s'occupe d'instancier et utiliser l'objet <<control>> correspondant à l'action demandée. En l'occurrence, il s'agit de SuperUserAdder.

Une fois la validité des données entrantes vérifiée, l'objet <<lifecycle> EntityManager cherche l'administrateur afin que SuperUserAdder ne l'ajoute pas deux fois dans le système. S'il n'est pas déjà présent, SuperUserAdder crée l'entité AdminUser et demande à l'EntityManager de le sauvegarder.

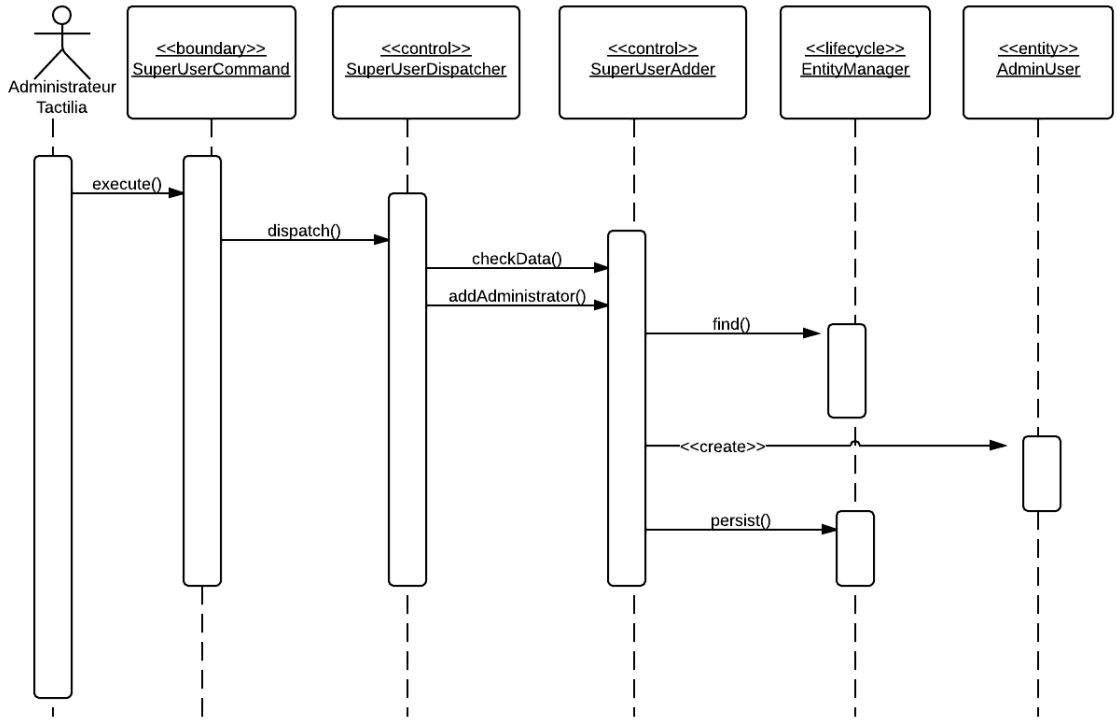


Figure 3-23 : diagramme de séquence « Ajout d'un administrateur »

3.4.4. Diagramme de séquence : consultation des tablettes

La consultation des tablettes se situe dans l'administration qui est une application Web. Comme il sera précisé dans l'architecture et la conception, c'est un double modèle de dérivés du MVC qui gère l'ensemble des interactions entre objets. Plus précisément, le navigateur dispose de son propre MVC, où le modèle se situe dans le serveur, qui lui-même à son tour utilise le MVC pour gérer l'accès aux données et les renvoyer selon la présentation (vue) demandée (en l'occurrence du JSON sur HTTP).

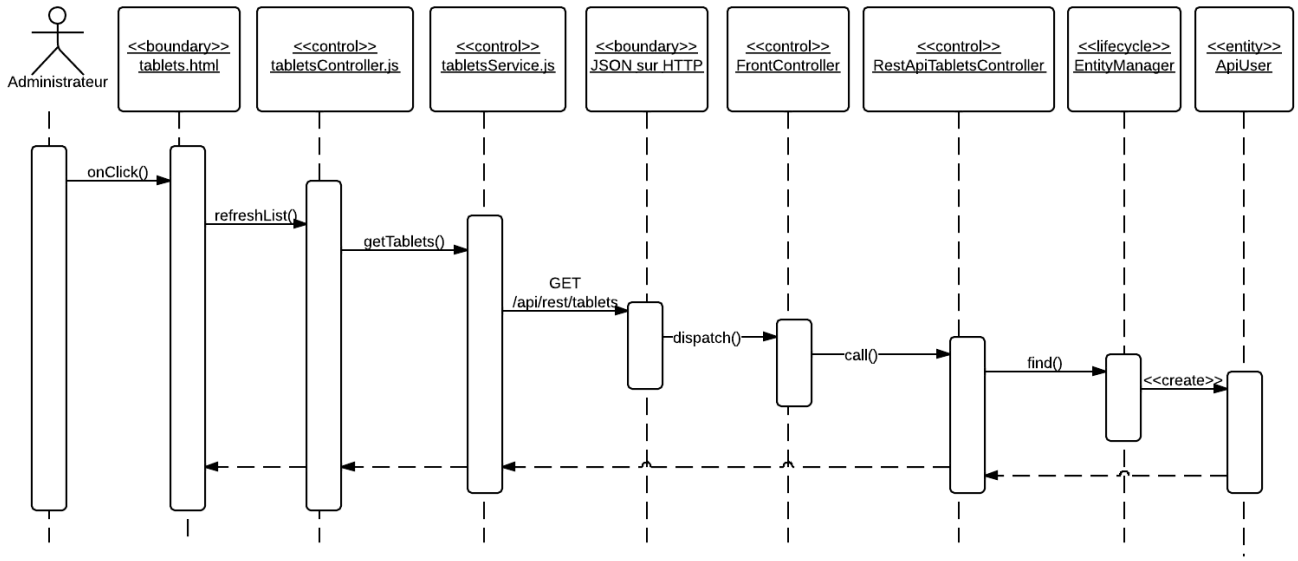


Figure 3-24 : diagramme de séquence « Consultation des tablettes »

3.4.5. Diagramme de séquence : modifier un devis

Cette opération est enclenchée par la tablette. Le stéréotype <<boundary>> correspond à du JSON porté par le protocole HTTP. Comme la réponse n'est pas porteuse de contenu, le code de retour HTTP suffit à « présenter » l'information sur le bon déroulement de l'opération.

On distingue aussi le découplage des objets métier : un contrôleur frontal pour gérer la requête qui mène à un contrôleur local propre à la requête, qui lui-même utilise un service métier spécifique aux devis. Ce dernier, enfin, utilise le gestionnaire d'entité (stéréotype <<lifecycle>>) pour créer l'entité correspondante à un devis. Le service la modifie et use une nouvelle fois le gestionnaire pour mettre à jour l'entité.

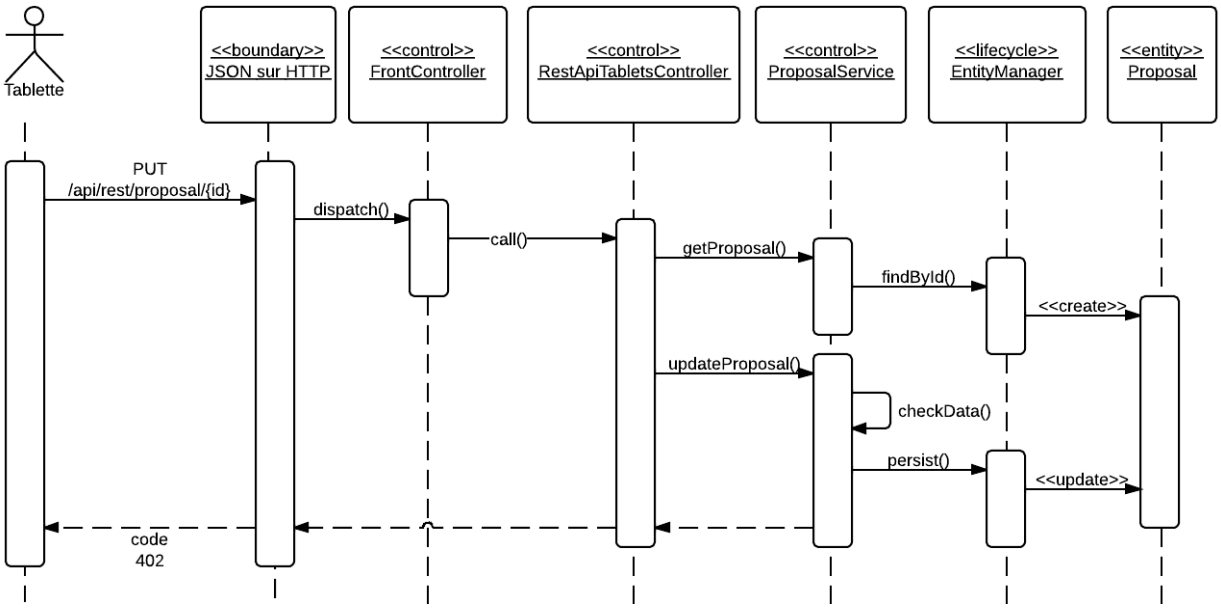


Figure 3-25 : diagramme de séquence « Modifier un devis »

3.5.Architecture

La définition d'une architecture va déterminer les principaux composants de la solution ExpoTouch, leur utilisation et interactions. Elle correspond à l'approche abstraite des choix technologiques et peut éventuellement découler de ces derniers. Nous allons d'abord voir l'architecture « logique », puis son éventuelle intégration dans un environnement orienté virtualisation.

3.5.1. Architecture logique

Pour ExpoTouch, l'idée va être de séparer les responsabilités et de découpler les composants dans l'objectif de rendre l'architecture aussi souple que possible. Le point de départ est une architecture 3-tiers classique où chaque couche communique avec ses voisines immédiates (la couche N parle aux couches N+1 et N-1 uniquement). Chacune d'entre elles a un rôle bien défini : présentation, métier et accès aux données :

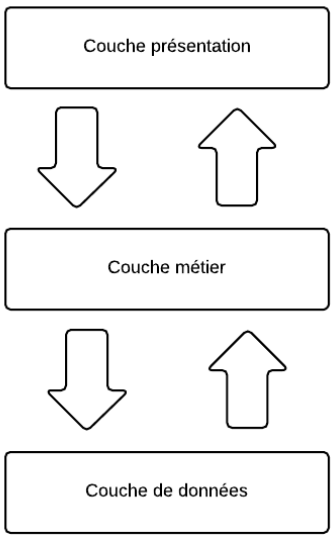


Figure 3-26 : architecture 3-tiers

3.5.1.1. Couche de présentation

Chaque sous-système dispose d'une forme de présentation qui lui est propre. Dans le cas de l'administration de la solution et de l'administration client, elle correspond à une interface homme-machine (IHM) qui présente et recueille les données de façon interactive. Dans le cas de la communication entre les tablettes et le système, il s'agira d'une représentation des données par le format JSON (contenant) accompagnée du protocole HTTP (conteneur).

Administration de la solution :

Elle ne nécessite pas une IHM complexe : elle est destinée à être utilisée en interne au sein de Tactilia et uniquement pour intégrer un nouveau client ou un nouvel administrateur, c'est-à-dire de façon occasionnelle. En revanche, son rôle de gestion de l'authentification lui confère une obligation de sécurité toute particulière. On va donc choisir une IHM en ligne de commande « enfouie ». Pour cela, on la rendra uniquement accessible via une interface SSH (*Secure Shell*) elle-même protégée par plusieurs niveaux : clef privée, encryptage de la clef, plusieurs niveaux d'utilisateurs.

Administration ExpoTouch :

L'administration par un client ExpoTouch est une application Web de dernière génération. Il s'agit d'obtenir un résultat proche d'une application dite « lourde ». Ce point est important car c'est, entre autres choses, ce qui fait particulièrement défaut dans la version 1 d'ExpoTouch : l'interface générique proposée par Drupal est touffue et complexe. Elle est notamment alourdie et ralentie par l'emploi du modèle traditionnel du Web : l'application est découpée en pages qui réclame chacune une requête et un rafraîchissement total de l'interface.

Le *World Wide Web* a désormais plus de vingt ans et son usage a considérablement évolué depuis les premières spécifications de Tim Berners-Lee. Les protocoles (HTTP, URL, HTML) ont peu changé, mais côté client les avancées et standardisations ont apporté de nouvelles approches. Le plus significatif a été l'explosion du Javascript, passé du mépris à la reconnaissance à grande échelle, aussi bien dans le navigateur qu'à l'extérieur du navigateur.

Ce retournement est notamment dû aux possibilités offertes par l'objet natif XmlHttpRequest, permettant l'envoi et la réception de données de façon asynchrone à partir d'un document XHTML chargé dans le navigateur (XMLHttpRequest Level 1 s.d.). Associée aux capacités de manipulation du Document Object Model (DOM) – représentation interne de la page XHTML – cette fonction a donné naissance aux technologies AJAX : *Asynchronous Javascript And XML*.

Le principal apport d'AJAX est la réduction de la distance entre une application Web et une application traditionnelle reposant directement sur le système d'exploitation. Cette réduction est notamment due à celle du nombre de pages effectivement chargées dans le navigateur. Le

modèle initial du Web, basé sur la notion de documents, évolue alors vers une granularité plus fine où chaque document peut être modifié partiellement et à la demande.

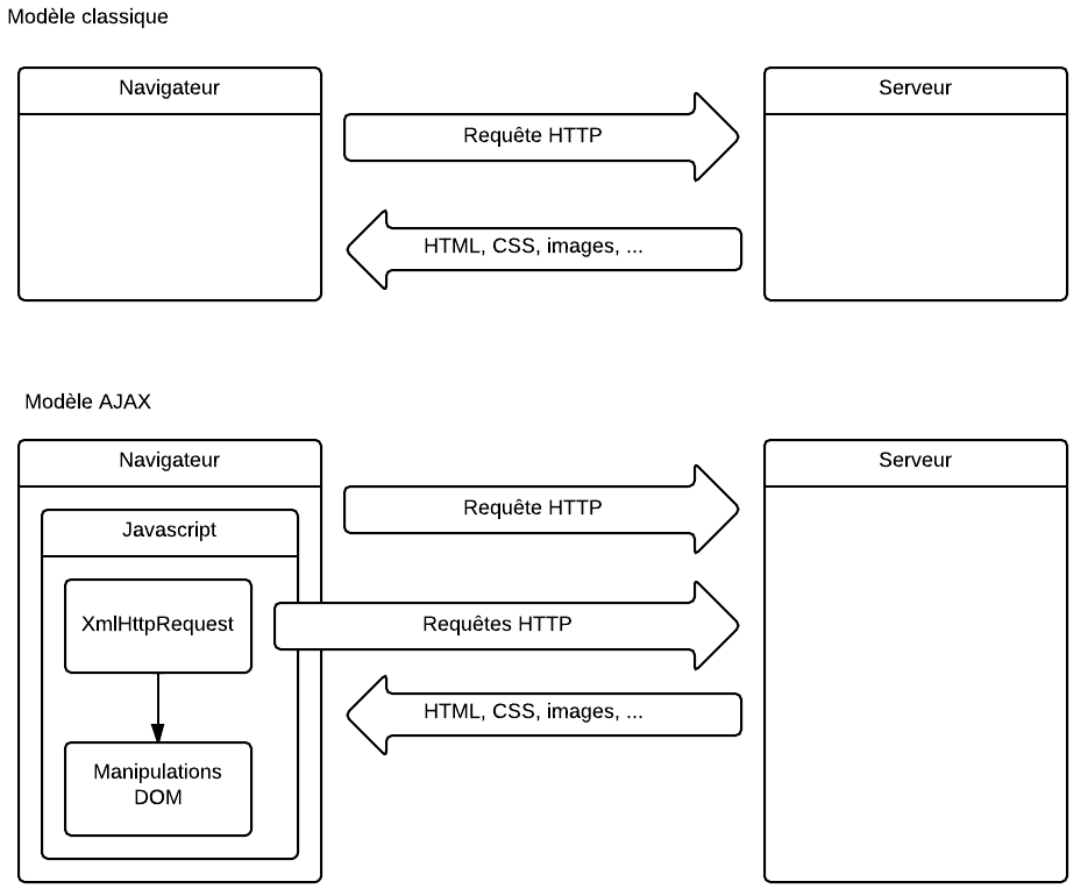


Figure 3-27 : modèle classique et modèle AJAX du Web

Le stade ultime de cette évolution se décrit dans l'acronyme SPA : *Single Page Application*. Le navigateur ne vient charger qu'une unique page HTML présentée en permanence à l'utilisateur, mais modifiée en temps réel afin d'offrir une interface cohérente, dynamique et réactive. Les modifications sont essentiellement déclenchées par la récupération de données via JavaScript en mode asynchrone. Le serveur ne retourne plus une vue HTML (comme le fait le modèle traditionnel) mais des données brutes, qui sont exploitées directement par le JavaScript au sein du navigateur afin de les mettre en forme et les afficher. Cela revient à déporter le traitement de la vue totalement côté client. On obtient alors une interface utilisateur qui approche au plus près des interfaces d'applications lourdes qui utilisent directement les ressources fournies par le système d'exploitation.

Outre la qualité de l'expérience utilisateur, une SPA offre aussi une ouverture vers le multiplateforme. En effet, que ce soit sur téléphone mobile, tablette, consoles de jeu ou

machines dédiée à l'industrie, le système d'exploitation sous-jacent est de plus en plus à même de fournir un composant Web. De même, les performances et les fonctionnalités des appareils actuels apportent une fluidité proche, si ce n'est identique, à celle que l'on peut trouver sur un poste de travail classique. Et plus encore, le composant Web est capable d'accéder à d'autres composants du système pour étendre les capacités de l'application (GPS, stockage local, audio…). Cela signifie que l'on peut inclure de façon transparente une application Web locale ou distante en tant qu'application intégrée au système, hors navigateur. On obtient alors un seul code, totalement portable, évolutif aisément, avec tous les avantages des technologies Web d'aujourd'hui, dont notamment une capacité de déploiement optimale.

Toutefois, ce paradigme d'application est encore jeune et se voit freiner par plusieurs inconvénients :

- Il faut un matériel et un navigateur récents pour obtenir une qualité d'expérience utilisateur totalement satisfaisante. Les PC d'aujourd'hui répondent correctement à cette exigence, mais les tablettes et téléphones nécessitent d'être tout récents ;
- Ce modèle de conception d'application est encore jeune et manque de maturité. Les outils et librairies évoluent très vite mais impliquent beaucoup de formation et un développement parfois en « terre inconnue ».

Couche présentation pour les tablettes :

La communication entre les tablettes et le système (serveur) fait aussi partie de la couche présentation même si celui qui « voit » n'est pas un humain. En l'occurrence, il s'agit du serveur et de la tablette dont on doit s'assurer qu'ils disposent d'un langage commun : la tablette « présente » les données qu'elle envoie, le serveur « présente » les données qu'il renvoie. La solution typique serait XML (*eXtended Markup Language*) dont l'essence même est l'interopérabilité des dialogues machine à machine. Toutefois, XML se voit aujourd'hui en concurrence avec un autre format : *Javascript Object Notation* (JSON). Celui-ci permet de porter des objets au format texte de façon très simple : il est non verbeux (contrairement à XML) donc plus léger et plus lisible, et s'intègre parfaitement dans nombre de technologies (Javascript, PHP…). Son mode de fonctionnement est très simple : il permet de décrire des ensembles clef/valeur et des listes ordonnées de valeurs (tableau). Une valeur peut être un objet, un nombre (entier, double), une chaîne de caractères, un booléen ou un tableau.

XML	JSON

<?xml version="1.0" encoding="UTF-8"?>
<product id= »1234 »>
<intitule>Chambranle</intitule>
<prix>500</prix>
<images>
<image>chambranle001.jpg</image>
<image>chambranle002.jpg</image>
<image>chambranle003.jpg</image>
</images>
</product>
	"product": {
	"id":"1234",
	"intitule": "Chambranle",
	"prix": 500
	"images": [
	"image001.jpg",
	"image002.jpg",
	"image003.jpg"
	]
	}

Tableau 2 : comparaison XML et JSON

Sa simplicité et sa facilité d'intégration ont donc décidé de son adoption pour ExpoTouch. Par ailleurs, le JSON n'est pas le seul élément « présenté ». On y ajoute les caractéristiques intrinsèques au protocole HTTP qui fait à la fois office de transporteur, conteneur et informateur. Les informations présentées par HTTP sont notamment les verbes (GET, POST, PUT, DELETE) et les codes de retour. C'est ce qui caractérise le paradigme RESTful qui sera détaillé dans la partie 5.1.4.

Standardisation des échanges pour la couche présentation :

Nous venons de voir que les couches présentation de l'administration et de la communication avec les tablettes fonctionnent avec des échanges de données. Plutôt que d'avoir deux systèmes distincts côté serveur et deux protocoles différents, on peut envisager un paradigme commun. Cela revient à dire que le serveur ne propose qu'un seul protocole qui présente les données de façon uniforme quelque soit le client.

Etant donnée la nature légère des échanges, tant entre pour le client d'administration (navigateur) que les tablettes, on peut choisir JSON et HTTP pour les deux. Il s'agira donc JSON échangé via REST. Ainsi, la couche présentation de l'administration émettra et recevra des données JSON via REST et les affichera après mise en forme grâce au JavaScript du navigateur. Il procède alors comme la couche présentation de la communication tablettes/serveur.

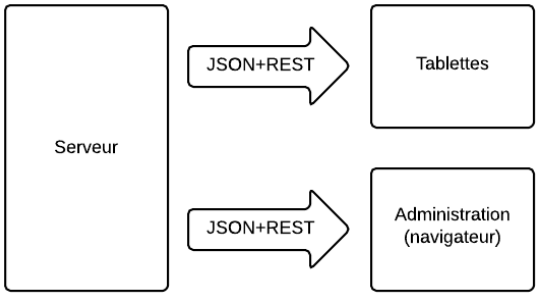


Figure 3-28 : couche présentation pour les tablettes et l'administration

3.5.1.2. Couche métier

Elle concentre les traitements des données issues de la couche présentation et de la couche données. A ce titre, dans le cadre d'une application Web, elle a de multiples rôles :

- Réceptionner les requêtes ;
- Réaliser la demande la requête (le cœur du métier) ;
- Renvoyer une réponse.

La réception des requêtes et le renvoi des réponses ne sont pas à proprement parler des objectifs métiers. On a affaire à un rôle de coordination où la logique métier est concentrée dans des unités de traitement : les services. Leur rôle est de répondre à une demande métier précise, indépendamment de l'origine de la demande et de la destination de la réponse. Ainsi, un service sera tout autant valide dans le cadre d'une requête Web que dans celui d'une application en ligne de commandes. Dès lors, on peut revoir notre architecture en couches en décomposant la couche métier en couche de coordination et couche services.

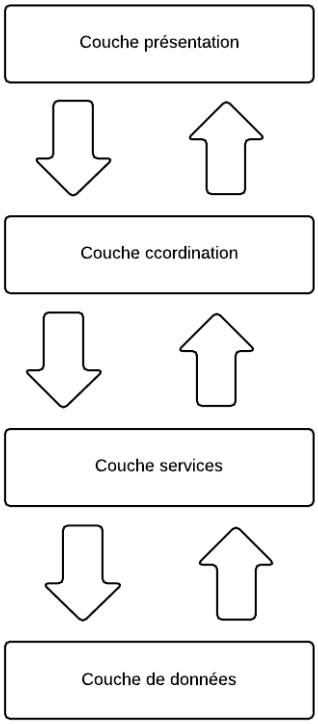


Figure 3-29 : architecture 3-tiers étendue (n-tiers)

Nous verrons que cette architecture est en fait le patron de conception Modèle-Vue-Contrôleur de seconde génération, notamment utilisé dans le canevas Symfony choisi pour la conception d'ExpoTouch.

Couche d'accès aux données :

Cette couche rassemble l'accès et la persistance des données. L'accès aux données est généralement pris en charge par une couche intermédiaire qui communique avec une couche uniquement dédiée à la persistance. Le rôle de la première est de préparer des objets utilisables par les couches supérieures : ce sont les entités. Le rôle de la seconde est de stocker les données – et notamment les entités – dans un format qui lui est propre. C'est dans cette couche que l'on trouvera un gestionnaire de bases de données (SGBD) et les librairies nécessaires pour l'exploiter (JBDC en Java, PDO en PHP, etc.).

Dans le cadre d'ExpoTouch, la technologie choisie (Doctrine - exposée dans la partie conception) est un *Object Relationnal Mapper* (ORM). Elle permet de combiner les deux couches (accès et persistance) en une seule. On reste alors dans un cadre strictement objet avec des possibilités nouvelles telles que l'héritage relationnel.

3.5.2. Virtualisation

La compatibilité descendante implique la contrainte de l'environnement de production de chaque coopérative. Mais dans le cadre de la transformation de la solution vers une licence, on doit être en mesure d'offrir une architecture adaptée à la solution et à son usage.

L'architecture choisie est assez classique (n-tiers en clients-serveurs) mais exploite les possibilités technologiques d'aujourd'hui, en particulier la virtualisation. Ce domaine connaît une croissance fulgurante depuis presque 10 ans, notamment du fait de sa démocratisation par le biais d'Amazon Web Services. Cette filiale du célèbre libraire en ligne fournit de nombreux services tels que : Elastic Computer Cloud, Simple Storage Service, Elastic Block Store, Glacier, etc. Son aboutissement et sa fiabilité en font le leader du marché avec pour références des clients célèbres : Reddit, Shazam, DropBox, Zynga… Outre mon expérience avec ce fournisseur de services virtualisés, ce sont aussi cette prédominance dans le secteur et le spectre de services qui ont guidé mon choix. Amazon Web Service permet tout aussi bien de fabriquer une architecture complexe de façon chirurgicale, que de simplement reposer sur des solutions prêtes à l'emploi. Dans le cadre du projet ExpoTouch, c'est le versant « manuel » qui a été choisi afin de bâtir un environnement de production parfaitement adapté au meilleur coût.

Le principe initial de la virtualisation consiste à faire fonctionner plusieurs systèmes d'exploitation sur une même machine physique. De ce principe, on peut extrapoler la virtualisation à tous types de ressources : stockage, base de données, réseau, etc. On obtient alors un système souple et disposant d'une certaine dynamique :

- Possibilité d'ajouter ou retirer des ressources ;
- Possibilité d'augmenter ou réduire les capacités des ressources ;
- Coût des ressources selon leur usage : temps d'utilisation, quantité de données stockées ou transférées.

Cela signifie que l'on dispose d'une liberté architecturale infiniment plus importante par rapport à une solution traditionnelle d'hébergement. Si des capacités viennent à se révéler trop justes, il suffit de les augmenter. A l'inverse, si l'on a surdimensionné au départ, on n'est plus prisonnier d'une solution achetée ou louée : on ajuste au nécessaire. Cet échelonnage des capacités (*scalability*) peut se faire au niveau de chaque ressource et/ou au niveau architectural. Dans le premier cas, on est dans un modèle vertical : c'est le modèle traditionnel

où l'on remplace ou étend du matériel obsolète par du plus récent (augmentation de la mémoire, changement de processeur…). La virtualisation permet de transformer le remplacement physique du matériel par une opération purement logicielle. Le second cas correspond au modèle horizontal : l'architecture accepte un nombre variable de ressources que l'on fait évoluer au gré des besoins.

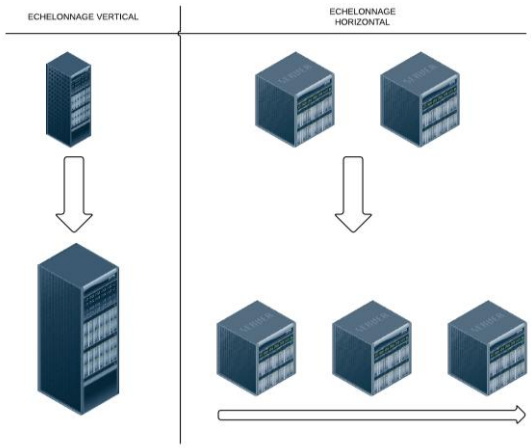


Figure 3-30 : comparaison de d'évolution d'une architecture échelonnée verticalement ou horizontalement

En considérant les possibilités du modèle horizontal, il faut envisager une architecture totalement découplée où chaque ressource est indépendante : serveur d'application, serveur de base de données, serveur de cache, stockage, etc. Des outils de métriques adaptés permettront alors d'analyser chaque composant de l'architecture afin d'en faire évaluer les capacités précisément, selon le modèle vertical. On cherche donc à mixer les deux modèles dans une optique d'optimisation des coûts et des performances.

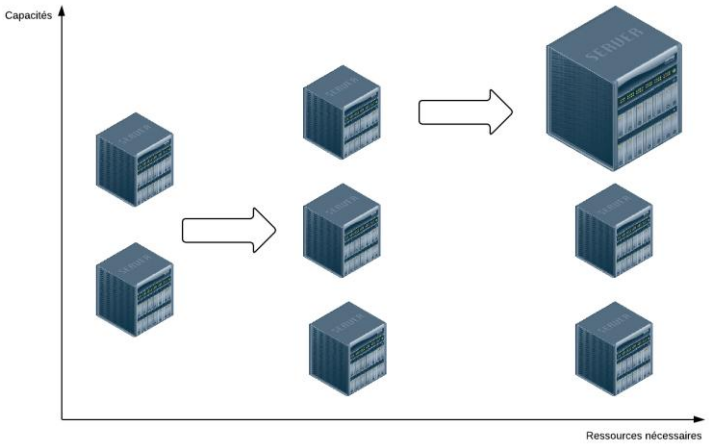


Figure 3-31 : application d'un échelonnage horizontal et vertical

La solution ExpoTouch sera distribuée sous deux formes :

- Installée dans l'environnement matériel du client ;
- Hébergée et maintenue par Tactilia.

Dans ce dernier cas, l'hébergement se fera chez Amazon Web Services (AWS). Ce choix est notamment lié au fait que AWS offre à la fois une prestation d'Infrastructure-en-tant-que-Service (*Infrastructure as a Service* – IaaS) et une prestation Plateforme-en-tant-que-Service (*Platform as a Service* – PaaS).

L'IaaS correspond à l'offre matérielle : on peut obtenir rapidement et facilement des machines virtuelles avec des caractéristiques de capacités et performances précises. Et surtout, on peut automatiser la gestion des machines (lancement, destruction, mise en veille…) ainsi que leurs capacités (augmentation des ressources). Concrètement, cela signifie que la solution ExpoTouch peut suivre l'évolution de ses utilisateurs sans avoir à opérer une migration complexe.

Le PaaS correspond aux offres logicielles du prestataire AWS. On retiendra en particulier les possibilités de gérer le stockage quelque soit la durée :

- Sur des disques rattachés aux machines virtuelles (on est encore dans le cas de l'IaaS) ;
- Dans un espace de stockage à performances moyennes, via une API et des librairies pour différents langages et systèmes d'exploitation ;
- Dans un espace de stockage à performances réduites mais très large.

Ces différents espaces sont exploités pour stocker et sauvegarder aussi bien des données brutes (les bases de données), que des machines virtuelles complètes sous la forme d'instantanés (*snapshots*).

4. Conception

4.1. Choix technologiques

Comme expliqué précédemment, les choix technologiques sont issus d'une phase de recherche et d'évaluation approfondie. Une première sélection s'est faite via des recherches sur Internet, notamment au travers des sites d'emplois qui listent les technologies demandées ou les sites spécialisés dans le développement. Citons notamment le site StackOverflow qui permet de poser et répondre à des questions de développement avec un système de recherche basé sur des mots-clefs (*tags*) offrant ainsi une vision temps réel des technologies actuelles (Tags s.d.). Bien évidemment, ma sélection s'est aussi faite sur mon expérience personnelle : l'idée n'étant pas de tout révolutionner, mais surtout rationnaliser et optimiser.

Une première liste étant établie, la quantité et qualité de documentation ont permis d'établir les candidats les plus aptes à être intégrés dans le projet ExpoTouch et l'entreprise Tactilia. Il est capital de pouvoir rapidement être en mesure d'appréhender une technologie et de suivre une courbe d'apprentissage aussi régulière et finie que possible. Il est à noter que mon travail sera aussi de lisser cette courbe lors de la transmission aux autres membres de l'équipe.

Enfin, pour valider les choix en cours, une série de preuves de concepts (POC) ont été mis en œuvre avec un triple objectif :

- Avoir un premier aperçu concret de la technologie ;
- La mettre à l'épreuve de quelques situations réelles (debugging, performances…) ;
- Construire une base de connaissance servant de départ à l'apprentissage.

Les parties suivantes sont le résultat de ce travail.

4.1.1. Côté serveur

4.1.1.1. Le langage PHP5, désormais un langage objet

Créé en 1994 par Rasmu Lerdorf, PHP a d'abord été une suite d'outils destinée à aider son créateur à gérer sa page personnelle. Les fonctionnalités offertes donnaient essentiellement de quoi traiter des données HTTP, de communiquer avec une base de données et de générer du HTML. Par la suite, il est devenu un langage, notamment parce qu'il exploitait (et exploite toujours) des éléments simplifiés du langage de script Perl. Cet héritage en a fait un langage interprété et procédural, du moins jusqu'à la version 4.

Ses fonctionnalités et sa simplicité ont permis son adoption rapide au cours du temps, notamment via des architectures Linux-Apache-MySql-PHP (LAMP). De nombreux hébergeurs ont proposé ce type d'architecture dans des solutions clefs-en-main, permettant au plus grand nombre de développer des sites dynamiques à moindre coût.

Toutefois, son évolution « organique » - et non planifiée - lui a fait porter une assez mauvaise réputation dans le monde professionnel. Contrairement à un langage comme Java qui a été conçu comme tel dès le départ, PHP a d'abord été un Perl pour le Web agrémenté de fonctions qui s'empilaient de façon plutôt anarchique. Entre son fonctionnement procédural, son typage faible, l'absence d'espace de nom, les incohérences de nommages et d'organisation des paramètres, etc. il a fallu attendre le changement de paradigme de la version 5 pour que PHP acquière ses premières lettres de noblesse.

Ce nouveau paradigme apparaît en 2004 et va permettre progressivement la légitimité du PHP auprès des professionnels : il devient un langage orienté objet. C'est le modèle de classes qui est choisi (par opposition au modèle de prototype), porté par une très forte similitude avec le Java. On retrouve de l'héritage simple, des interfaces, de l'encapsulation et protection des propriétés, de la surcharge. La compatibilité avec la version 4 est totalement maintenue et on en retrouve les principaux aspects : typage faible par défaut (mais qui peut être partiellement renforcé), fonctions procédurales, interprétation (et donc introspection), etc.

Les années suivantes ont vu l'évolution de la version 5 vers une professionnalisation accrue de par l'engouement qu'elle a suscité. Le premier porteur étant l'un des plus gros sites du Web : Facebook. Avec un tel « exploitant », PHP a continué de croître en usage mais aussi en capacités via l'apparition de bibliothèques et d'outils. Ces derniers ont totalement sorti PHP de son ornière d'amateurisme. Dorénavant on dispose de quoi faire des tests unitaires

(PHPUnit), de la liaison objet-relationnel (Doctrine, Propel), du débogage (XDebug), de la gestion de paquets (Packagist et PHAR), etc.

PHP a été choisi pour Expotouch 2 assez logiquement parce qu'il était le langage de la version 1. Toutefois, ce sont aussi les qualités intrinsèques de la version 5 du langage qui ont dicté ce choix. PHP 4 aurait été hors de question parce qu'il ne répond pas aux exigences minimum d'un langage moderne, notamment par l'absence d'orienté objet. Dorénavant au même niveau qu'un langage mature tel que Java ou C#, le PHP garantit la mise à disposition des qualités nécessaires à la production d'un code industriel. Ajoutons que cette maturité se traduit aussi par de nombreuses librairies, canevas, documentations, etc. offrant une meilleure productivité.

On retiendra pour la suite du développement quelques écueils à contourner :

- PHP est interprété, ce qui peut provoquer une dégradation des performances en cas d'opérations trop intensives. Heureusement, l'interprétation est en fait une compilation « juste-à-temps » (« *just-in-time* », JIT) qui produit du code intermédiaire (*opcodes*) lui-même interprété par une machine virtuelle. On obtient un gain substantiel de performances en intégrant un cache de compilation qui maintiendra le code intermédiaire en mémoire et évitera une phase compilation à chaque exécution ;

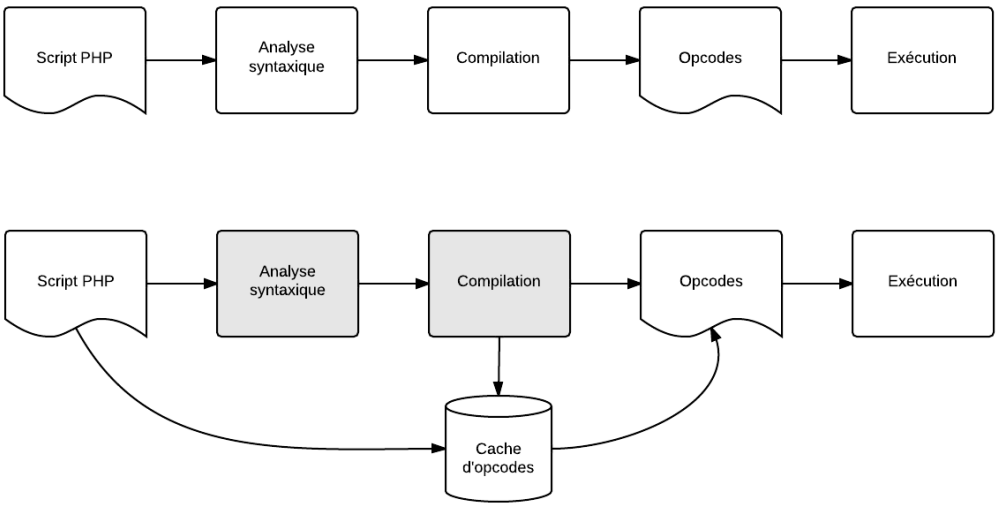


Figure 4-1 : processus d'interprétation sans et avec cache de PHP

- PHP dispose d'un typage faible : il est même totalement optionnel. On veillera à l'employer partout où cela est possible. Le contrôle qualité du code, notamment par l'emploi des tests unitaires et l'analyse de la couverture de code, permettront de limiter les risques d'erreurs ;

4.1.1.2. Symfony 2, le canevas

Un canevas (*framework*) est une proposition d'architecture logicielle sous la forme d'un ensemble cohérent de composants. Son rôle est de fournir une standardisation de conception ainsi que des outils pour faciliter le développement. Il en existe dans presque tous les langages, et avec des objectifs plus ou moins ciblés, notamment :

- Canevas pour application Web ;
- Canevas pour le mapping objet-relationnel ;
- Canevas de logging ;

Les faiblesses inhérentes aux premières versions de PHP ont très vite donné naissance à des canevas (*framework*) de développement. Leurs rôles étaient autant de pallier à ces faiblesses que de fournir un modèle standardisé de production logicielle. Dorénavant, avec la version 5, les canevas ont atteint une nouvelle dimension et sont tournés autant vers une productivité optimum que vers un confort maximum de développement.

Dans le cadre d'ExpoTouch, le canevas choisi est Symfony (Symfony s.d.), notamment pour les raisons suivantes :

- Il s'agit d'un projet open-source dont la mise en œuvre et la maintenance sont assurées par la section R&D de l'agence Sensio Lab. Cette section est dirigée par Fabien Potencier, créateur et développeur en chef de Symfony. De fait, on a une garantie de cohérence, stabilité et pérennité ;
- Le canevas est mature et moderne : lancé en 2005, il en est désormais à la version 2 et exploite les dernières évolutions aussi bien méthodologiques (modularité, injection de dépendances, annotations…) que logicielles (PHP version 5, extensions…) ;
- Il repose sur d'autres projets open-source et ne se présente pas comme une « boîte noire » qui prend tout en charge ;
- Bien au contraire, son architecture accepte et encourage les extensions (*plugins*) ;
- Il est fourni avec un environnement et des outils qui simplifient considérablement nombre de tâches, et améliorent la courbe d'apprentissage ;

- Il est désormais largement reconnu et exploité. On notera notamment que la prochaine version de Drupal reposera sur Symfony !
- Il repose des patrons de conception, notamment le Modèle-Vue-Contrôleur (MVC), gage d'une certaine qualité d'architecture et de code.
- La documentation est exhaustive, proposant aussi bien des références de classes, que des mises en œuvre pratiques et des exposés de cas courants. A noter, elle est en anglais et en français ;
- S'il est majoritairement dédié à la conception d'applications serveurs, il intègre la possibilité de concevoir des outils en ligne de commande.

Choisir un canevas tel Symfony revient à poser un certain nombre d'exigences, tant au niveau architectural qu'au niveau implémentation. De fait, il cadre le développement au sein d'une équipe, et dans le cas d'ExpoTouch il offre une garantie de pérennité. En effet, avec son adoption reconnue et sa documentation de qualité, il faudra peu de temps pour que n'importe quel intervenant puisse prendre en main le projet. Et les interventions futures garderont la même logique, offrant une maintenance solide.

L'architecture de Symfony repose sur des modules (appelés « bundles ») qui hébergent un patron de conception modèle-vue-contrôleur de seconde génération (MVC 2). Le patron MVC de première génération est très largement adopté dans nombres d'applications (Web ou non) et offre une séparation entre les données, leur traitement et leur affichage. La seconde génération implémentée par Symfony fait usage d'un contrôleur frontal qui s'occupe de diriger les requêtes au contrôleur adapté. Ce dernier va alors traiter la requête, notamment en accédant au modèle, puis il renverra le résultat sous la forme d'une vue. La vue peut être un fichier HTML, des données XML, JSON, etc.

Il est à noter que le MVC choisi par Symfony est plus strict que le MVC original qui autorise une interaction directe en le modèle et la vue. Cette contrainte retire de la souplesse, voire alourdit l'usage du patron en déresponsabilisant la vue au profit du contrôleur. On prend alors le risque d'alourdir considérablement le contrôleur.

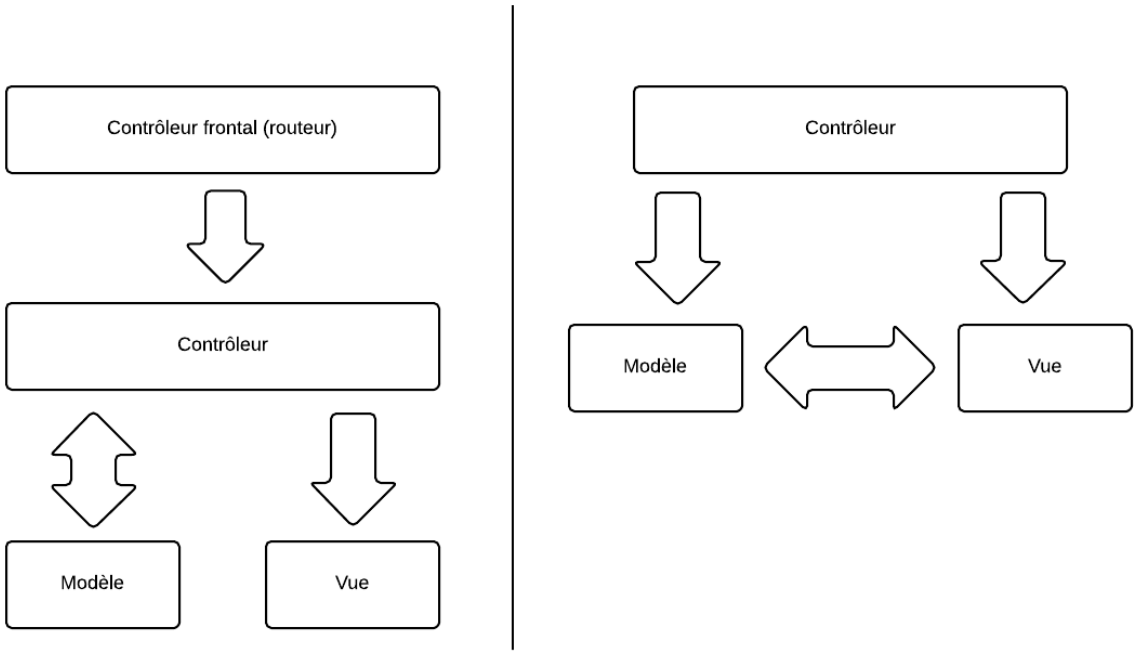


Figure 4-2 : comparaison des modèles MVC2 et MVC

Pour répondre à la problématique du contrôleur surchargé (« *fat controller* »), Symfony propose une décomposition supplémentaire sous la forme de « services ».

Dans une logique architecturale moderne, les concepteurs de Symfony ont choisi d'avoir un couplage aussi lâche que possible entre les composants du canevas. Pour ce faire, les traitements peuvent être rangés dans des « services » qui sont gérés par un « conteneur de services » (Bacco 2013). Le contrôleur utilisera ce dernier pour orchestrer les traitements et préparer la vue. L'objectif est alors d'obtenir des contrôleurs aussi légers que possible et qui exploitent une architecture modulaire et découplée. Le conteneur prend en charge les dépendances entre services, notamment par l'ordonnancement de leur chargement. Cette méthodologie n'est toutefois pas obligatoire : Symfony propose des solutions, impose une architecture… mais ne peut pas tout imposer !

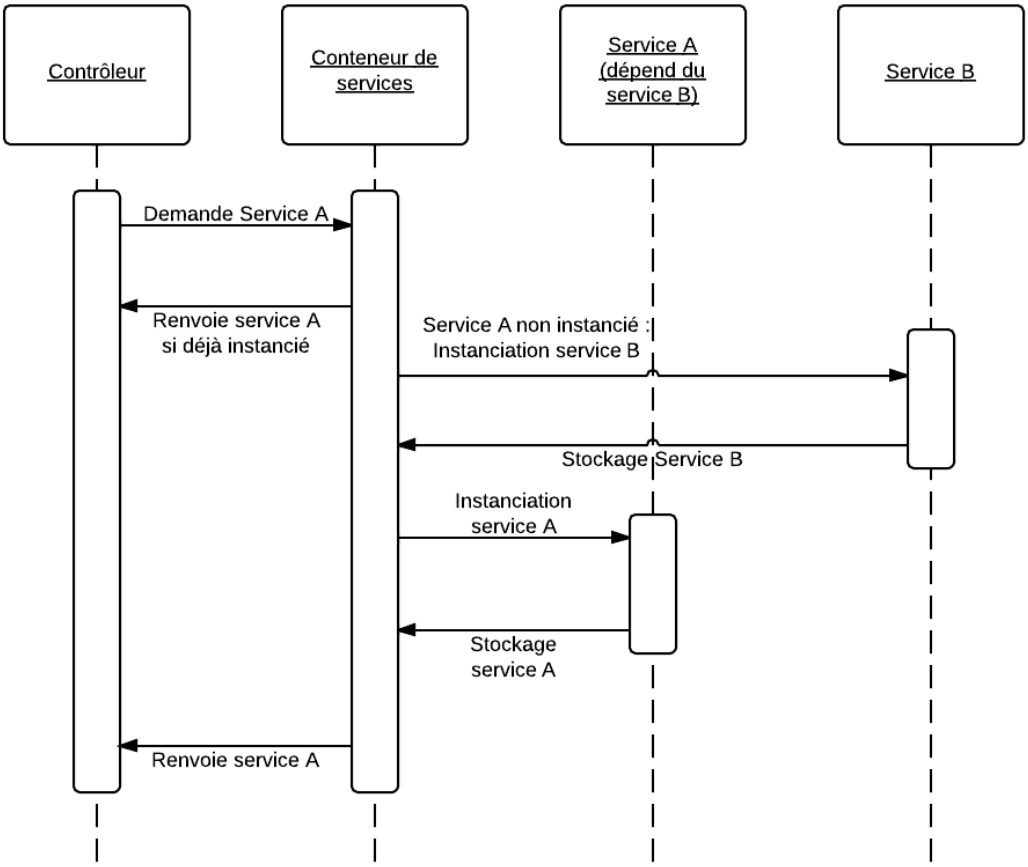


Figure 4-3 : fonctionnement du conteneur de services de Symfony

4.1.1.3. Doctrine pour l'accès aux données relationnelles

Doctrine (Doctrine Project s.d.) est un *Object Relational Mapper* (ORM) intégré à Symfony (mais dont l'utilisation est optionnelle). A l'instar de Java Persistence API (JPA) - son pendant Java - il permet par le biais d'annotations dans le code de faire le lien entre les classes et le modèle entités/relations de la base de données. Les ORM ont considérablement évolué ces dernières années, passant de systèmes assez lourds (multiples fichiers de définitions et configurations) à des environnements très confortables (annotations et outils de conception semi-automatisés). Un ORM facilite considérablement l'écriture du code en intégrant la couche d'accès aux données (*Data Access Layer* - DAL) et celle d'accès aux objets persistants (*Data Access Objects* – DAO) et gère leurs dépendances.

Plutôt que d'avoir à écrire le code du DAL, qui sera appelé par les DAO pour être utilisés par le code métier, les DAO sont simplement associés à des métas-data qui définissent la relation au gestionnaire de base de données. Ces méta-data étaient initialement décrites dans un fichier XML (cas de JPA), mais aujourd'hui les annotations ont pris le pas. C'est notamment le cas de Doctrine qui exploite ce mode de description bien qu'il ne soit pas natif dans PHP (contrairement à Java). Les annotations sont simplement indiquées dans des commentaires standardisés, analysés puis cachés par une librairie de Doctrine.

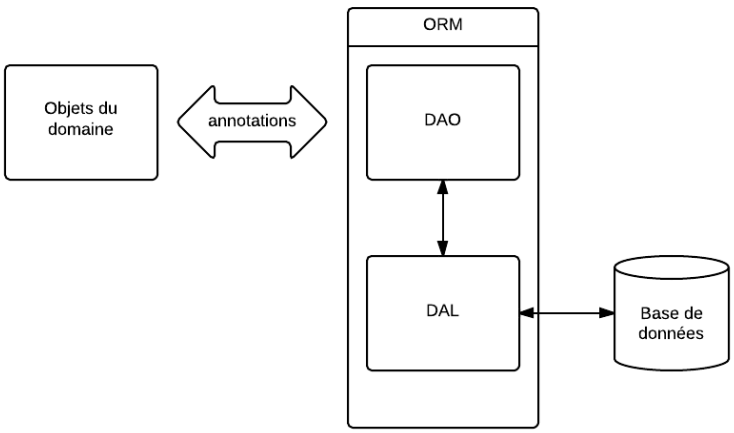


Figure 4-4 : fonctionnement d'un ORM

Toutefois, c'est un outil à double tranchant : il ne dispense pas de la maîtrise des concepts et usages des bases de données, autant pour obtenir des modèles cohérents que pour garantir des performances optimales. En effet, le confort offert par un ORM a un coût : si l'on se cantonne à penser uniquement « objets » sans regarder le schéma relationnel généré, on peut se retrouver très vite avec des tables démultipliées, des jointures et index inutiles. Pour

compléter cette opacité, Doctrine privilégie son propre langage d'accès aux données par-dessus SQL. Si cela permet de totalement s'abstraire des couches basses de la pile technologique (le SGBD sous-jacent), cela implique aussi les contraintes liés à un langage qui ne peut reproduire toutes les spécificités de tous les SGBD. On citera par exemple la gestion des dates qui oblige à des injections de code lourdes et complexes dans Doctrine.

Etant pourvu d'une certaine expérience en SQL, j'ai envisagé dans un premier temps de mettre en place ma propre couche d'accès aux données via un DAL en direction de MySql, puis un DAO pour faire la liaison avec mes classes métier. C'était la solution la plus évidente d'un point de vue expérience et assurance, le terrain des DAL et DAO étant déjà connu. De plus, c'est aussi la possibilité d'avoir un contrôle parfait sur les données, sans aucune dépendance que son propre code et les librairies bas niveau. Toutefois, c'est aussi la solution la plus chronophage et porteuse d'erreurs puisque l'on prend toute en charge.

Avec la lecture de la documentation de Doctrine (référencée par Symfony, gage d'une bonne intégration), j'ai très vite été séduit par les possibilités et le confort offert. Par ailleurs, Doctrine, tout comme Symfony, est actuellement à la version 2. Cette maturité se constate par le champ particulièrement étendu des situations que l'ORM est en mesure de gérer. Outre un accès assez fin au modèle du SGBD (tables, types des données, indexation), la liaison vers le modèle objet est toute aussi complète, notamment par la prise en charge de l'héritage. C'est donc considérant mon niveau de connaissance SQL et les avantages offerts par Doctrine que j'ai opté pour la solution ORM.

4.1.2. Côté client

4.1.2.1. AngularJS, le framework de Google pour HTML5/Javascript

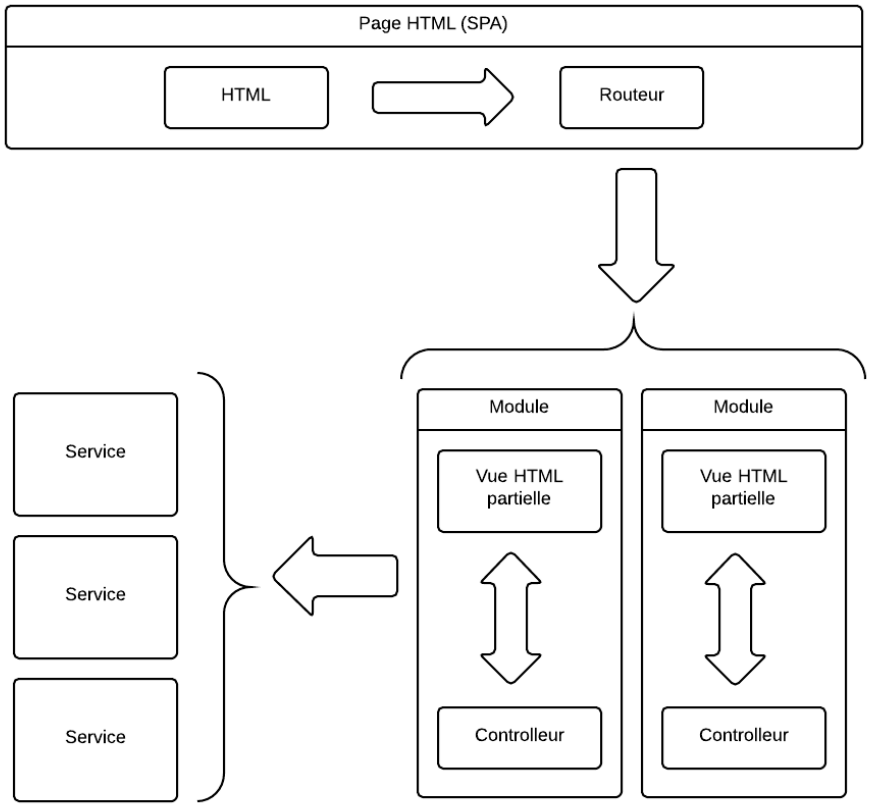
La création d'une *Single Page Application* (SPA) repose sur des technologies encore récentes et en recherche de maturité. Dans un premier temps, de nombreuses librairies et canevas sont apparus pour fournir des outils simples ou méthodologiques. Citons notamment Backbone.js, CoffeeScript ou JQuery. Toutefois, certains de ces outils cherchent à résoudre des problèmes trop précis et n'offrent pas une approche globale au développement d'une application dans le navigateur. Inversement, d'autres proposent des solutions en « boîte noire » et retirent toute granularité (par exemple, Backbone.js n'accepte que les échanges en REST, et pas en simple HTTP).

C'est de ce constat qu'en 2009, au sein de Google, est né Angular. Il s'agit d'un canevas particulièrement évolué qui permet une réelle industrialisation de la partie cliente sous HTML/CSS/JavaScript. Il repose sur plusieurs fonctionnalités et concepts :

- Une extension du patron de conception Modèle-Vue-VueModèle (MVVM), lui-même extension du classique Modèle-Vue-Contrôleur (MCV) (Minar s.d.);
- En résultante de l'emploi du MVVM : la liaison bidirectionnelle entre le modèle et la vue (*two ways data binding*) ;
- L'injection de dépendance, en particulier pour la gestion des services ;
- Une approche totalement asynchrone au travers des Promises.

L'ensemble offre une architecture (canevas) particulièrement efficace pour la mise en place d'applications Web en SPA. Outre de nombreuses problématiques simplifiées, on dispose avant tout d'un cadre de développement très structurant qui permet une industrialisation logicielle.

L'architecture SPA/Angular typique est constituée d'une page HTML destinée à encapsuler le contenu dynamique. On y trouvera tout ce qui est statique dans l'interface utilisateur : un bandeau, pied de page, menu, etc. Un routeur JavaScript est associé à la page : selon l'URL, il s'occupera de charger le contenu dynamique.



URL	Contenu dynamique chargé
http://www.expotouch.com/admin/#users	/partials/users.html
http://www.expotouch.com/admin/#/stats/sales	/partials/sales.html

Figure 4-5 : gestion du contenu en SPA avec AngularJS

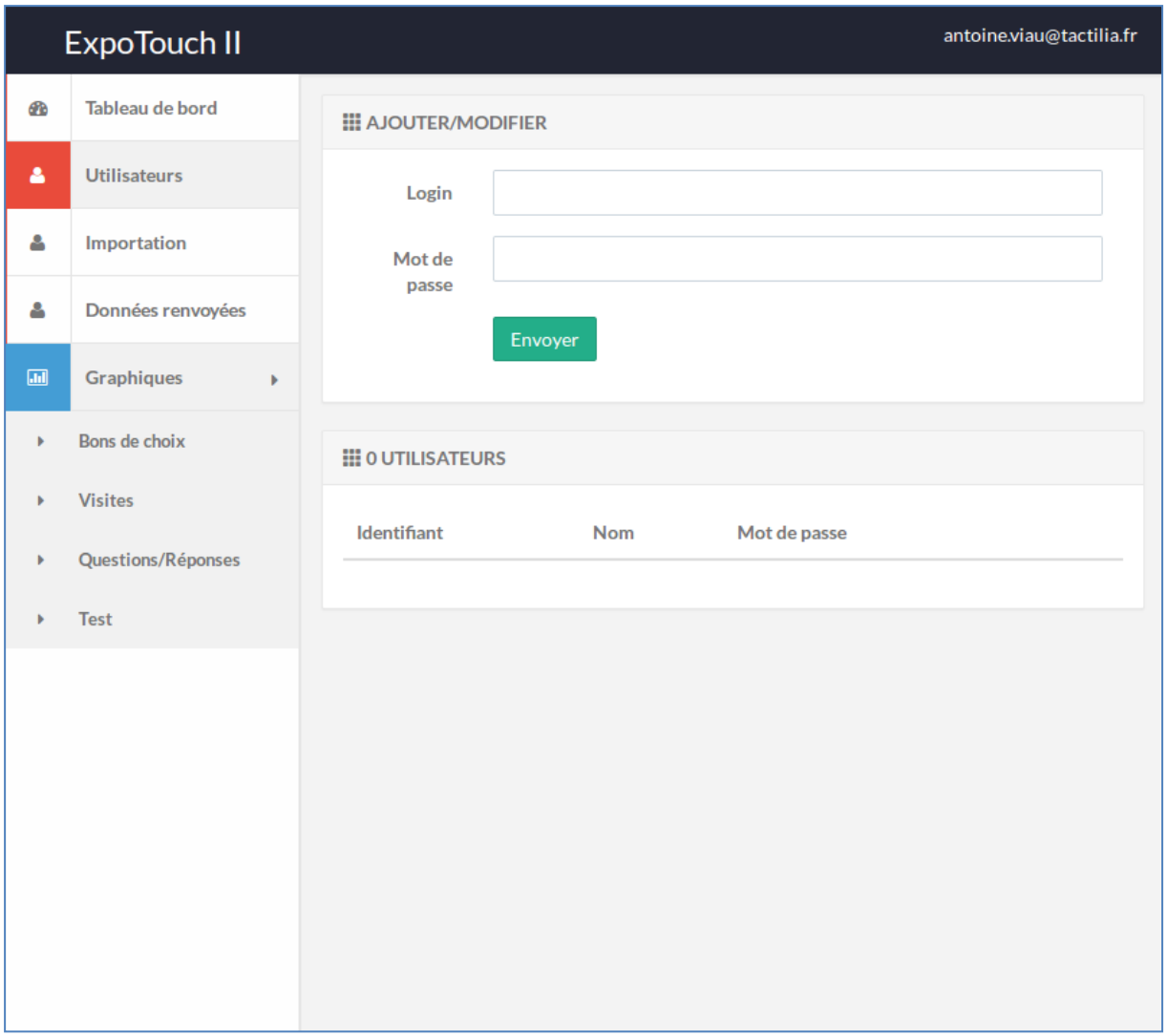


Figure 4-6 : écran de gestion des utilisateurs (tablettes) de l'administration ExpoTouch 2 : le menu de droite et la barre du haut sont statiques. Le contenu au centre est chargé dynamiquement, rendant l'interface beaucoup plus réactive

Dans chaque contenu dynamique, Angular associe un élément HTML et ses descendants à un contrôleur JavaScript. Ce contrôleur prend en entrée des services qu'il peut appeler pour tout type d'opérations. Typiquement, un service ira chercher des données sur un serveur et les mettra à disposition du contrôleur. Angular fournit certains services spécifiques destinés aux opérations de plus bas niveau. Citons notamment :

- $scope : service en charge de la liaison bidirectionnelle entre le code JavaScript et le contenu HTML via un langage d'expression spécifique. Toute modification de l'un sera répercutée sur l'autre ;
- $http : service qui encapsule l'objet XmlHttpRequest dans des Promises.

Ces deux services associés aux contrôleurs forment une extension au patron de conception Modèle-Vue-Modèle (MVVM). Ce dernier est dérivé du patron Modèle-Vue-Contrôleur (MVC) où une interaction directe – une liaison bidirectionnelle appelée Vue-Modèle – est installée entre le modèle et la vue dans une logique événementielle automatisée : la vue ne va pas chercher les données dans le modèle, elles lui sont automatiquement fournies à chaque mise à jour des données. Angular étend le MVVM en décorant la partie Vue-Modèle (le service $scope) avec le contrôleur.

L'architecture ainsi proposée sépare les rôles tout en offrant un certain confort dans la manipulation du modèle vers la vue. Le chemin typique d'un module serait :

- Le contrôleur va demander des données à un service ;
- Le service va chercher les données, par exemple sur un serveur distant ;
- Le contrôleur insère les données dans le service $scope ;
- Les modifications sur les données hébergées par le service $scope seront automatiquement répercutées dans la vue HTML ;
- La vue HTML formatera l'affichage des données via un langage propre à AngularJS ;
- Inversement, des données modifiées dans la vue (saisies par l'utilisateur) seront immédiatement répercutées dans le service $scope et mises à disposition du contrôleur.

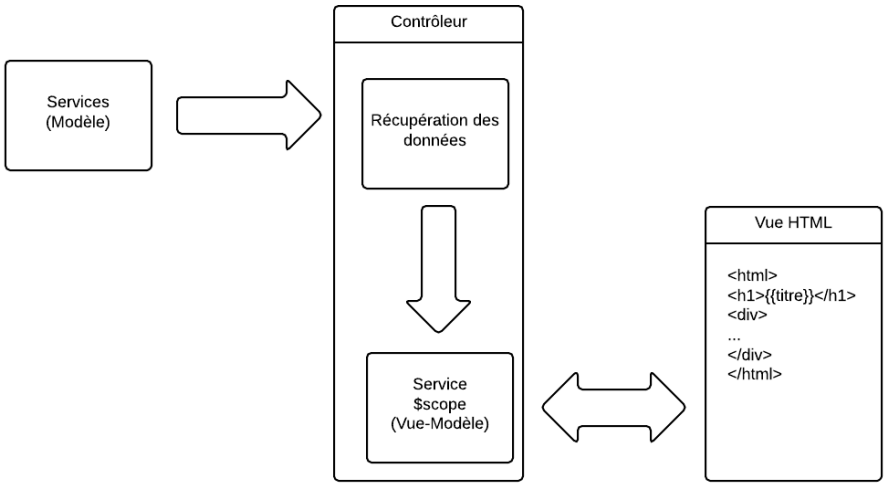


Figure 4-7 : parcours des données entre services et vues via le contrôleur dans AngularJS

Les services n'ont pas de rôle spécifique. Toutefois, ils sont essentiellement utilisés pour récupérer et traiter des données distantes. Ce processus distingue particulièrement une

application SPA d'une application classique : il prend du temps. On doit donc considérer deux problématiques :

- Informer l'utilisateur qu'une opération est en cours ;
- Concevoir l'architecture logicielle et le code dans une optique totalement asynchrone.

Si le premier cas est assez aisé à résoudre (on affiche un message ou une animation d'attente), le second est dépendant du Javascript et de l'objet XmlHttpRequest. Celui-ci fonctionne par fonctions de retour : on donne pour chaque opération des fonctions qui seront exécutées à leur terme. Dès que des opérations enclenchent de nouvelles opérations, la maintenance du code devient un réel problème du fait de l'imbrication des fonctions de retour. C'est ce qu'on appelle le « *callback hell* » (l'enfer des fonctions de retour).

```
userService.op1({
	success: function(err, user) {
		if (err) return {error: err};
		userService.op2( function(err) {
			if (err) return {error: err};
			userService.op3(callback);
		});
	},
	failure: function(err) {
		return {error: err}
	}
});
```

Le concept de Promises existe depuis la fin des années 1970 pour répondre aux problèmes de synchronisation dans certains langages concurrents. Au niveau JavaScript, l'implémentation a d'abord été élaborée dans la librairie Q de Kris Kowal (kriskowal/q s.d.), elle-même reprise par d'autres librairies et canevas. Le succès des Promises est tel, que la prochaine version des spécifications d'EcmaScript – les spécifications du langage Javascript – les intégrera (Draft Specification for ES.next (Ecma-262 Edition 6) s.d.).

Le rôle des Promises est de supprimer « l'enfer des fonctions de retour » afin de permettre une écriture de code asynchrone sous une forme synchrone. La lecture est beaucoup naturelle, la lisibilité considérablement améliorée. Bien plus qu'une conception cosmétique, les Promises offrent avec l'architecture d'Angular un réel paradigme d'écriture du code :

```
userService.op1()
.then( function() { return userService.op2(); }
       ,function(err) { ... }
).then( function() { return userService.op3(); }
       ,function(err) { ... }
).then( function() { ... }
       ,function(err) { ... }
);
```

Comme nous l'avons vu, l'emploi d'AngularJS correspond bien à la volonté d'industrialisation logicielle. Ce canevas offre autant une structure que des fonctionnalités à même d'améliorer considérablement la qualité d'architecture et d'écriture de la partie visible d'ExpoTouch. Et cela d'autant plus que le monde du développement Web côté client est encore très jeune et que l'on risque facilement d'utiliser plusieurs librairies plus ou moins intégrables les unes par rapport aux autres.

4.1.2.2. Affichage des statistiques via une librairie graphique : D3

L'interface d'administration doit permettre l'affichage de graphiques offrant une visualisation aisée de statistiques plus ou moins complexes. D'autre part, les statistiques en question ne sont pas arrêtées et l'on doit pouvoir en intégrer de nouvelles à la demande des clients de la solution. Pour ce faire, il est indispensable de reposer sur une librairie qui ne soit pas fermée et limitée dans ses possibilités de conception de graphiques.

Traditionnellement, les solutions employées reposaient sur Flash, seule extension réellement dédiée à l'affichage de graphiques complexes (au sens large du terme). Mais Flash tend à devenir obsolète, surtout depuis que les standards qui régissent les technologies Web – gérés par le W3C (*World Wide Web Consortium*) – intègrent les *Scalable Vector Graphics* (SVG). Ce format de données décrit en XML des graphiques au format vectoriel. Du fait qu'il soit en XML, il s'insère dans du XHTML (et donc dans le DOM), offrant une évolution naturelle à la conception de page Web (SCALABLE VECTOR GRAPHICS (SVG) s.d.).

Toutefois, le SVG reste « bas niveau ». Son rôle est de permettre l'affichage et l'animation de primitives graphiques (parfois assez évoluées) mais il n'est pas dédié à la conception de types de graphiques particuliers. C'est pour cela que la librairie Javascript D3 a été conçue. Plus qu'une librairie graphique, c'est un outil de conception de documents par les données, d'où son nom : *Data Driven Documents* (D3). Les données sont rattachées à l'arbre de représentation interne du HTML – le DOM – puis des transformations y sont appliquées afin

de générer le document attendu. On peut ainsi, à partir d'un même ensemble de données, générer une table, une courbe, un camembert, et même des représentations personnalisées et complexes.

Cette souplesse implique une phase d'apprentissage des concepts sous-jacents qui restent assez éloignés d'une simple application des données à un modèle de graphique spécifique. On doit d'abord définir les données via des sections d'entrée et de sortie : elles correspondent aux nouvelles données, et à celles obsolètes. C'est ce qui permettra de rendre leur affichage totalement dynamique et donc de rester dans un cadre strictement SPA : le graphique n'est pas détruit puis recréé à chaque évolution des données, mais simplement mis à jour. Il est à noter que cela permet aussi l'animation via des transitions.

Le graphique lui-même est alors construit par le rattachement des sections au DOM. D3 fournit l'API JavaScript nécessaire avec une formulation assez intuitive. On n'est pas dispensé de connaître les rudiments de SVG, mais de nombreuses opérations sont prises en charge par D3.

4.1.3. Le liant entre client et serveur : REST

REST est l'acronyme de *REpresentational State Transfer*. Il s'agit d'un style d'architecture décrit en 2000 par Roy Fielding. Ce style est basé sur l'architecture du Web :

> *"The design rationale behind the Web architecture can be described by an architectural style consisting of the set of constraints applied to elements within the architecture"*[1] (Fielding 2000)

REST peut se voir comme une approche de standardisation des architectures par composants dans le même esprit que les Web Services basés sur le protocole SOAP (*Simple Object Access Protocol*). Toutefois, REST est un style d'architecture et non pas un protocole. Il définit donc seulement des contraintes qui offrent une voie vers l'interopérabilité :

- Client-Serveur : séparation des rôles entre consommateurs (clients) et producteurs (serveurs) ;
- Sans état : chaque requête est indépendante de la précédente ou de la suivante, et doit contenir toute les informations nécessaires à son traitement ;
- Cachable : les réponses du client peuvent être mises en cache (côté client par exemple) ;

[1] *« Le principe de conception de l'architecture du Web peut être décrit par un modèle comprenant l'ensemble des contraintes appliquées aux éléments de cette architecture. »*

- Système en couches : le chemin entre le client et le serveur peut passer par divers intermédiaires (proxy, pare-feux…) ;
- Interface uniforme : les intervenants adoptent un modèle commun de communication ;
- Code à la demande (contrainte optionnelle) : possibilité d'exécuter du code côté client à la demande sur serveur pour décharger partiellement celui-ci.

REST repose uniquement sur le Web et sa philosophie. Les composants fournissent des ressources qui sont toutes identifiables via leurs URI (*Uniform Resource Identifier*) et manipulables via le protocole HTTP. Ce dernier fournit un nombre limité de commandes pour interagir : GET, POST, DELETE et PUT.

On peut donc dessiner une architecture où tout est ressource (données, algorithmes, procédures), identifiable et manipulable en CRUD dans des relations client à serveur, ou serveur à serveur. L'usage d'un service REST limite les nécessités de compréhension à celles de son comportement. Tout ce qui touche à la technique est maintenu dans un carcan délimité par des URI, des verbes, dans une pile protocolaire totalement connue (HTTP). Etant un style d'architecture, et non pas un modèle, ni une architecture en soi, il n'y a pas de spécifications strictement formelles. On est donc dans une approche assez libre où l'on peut faire varier la complexité selon les contextes. Cette double approche se veut la solution à la lourdeur des services Web traditionnels basés sur SOAP (Richardson 2007).

ExpoTouch respecte au mieux l'approche REST pour le support de l'API entre les tablettes et le serveur. La création se fait par des requêtes POST, les modifications par PUT, l'effacement par DELETE et la récupération par GET. Les requêtes portent dans leurs URI les informations ayant trait à la ressource demandée, tandis que le corps contient la ressource elle-même.

4.2.Développement

4.2.1. Développement et tests

La première version d'ExpoTouch n'a pas été développée avec des tests unitaires. Dans le cadre de la version 2, ils ont été d'une importance toute particulière. En premier lieu parce que le langage PHP est semi-interprété et faiblement typé. L'interprétation permet notamment de faire facilement des opérations introspectives : instanciation dynamique, génération d'attributs à la volée, etc. Ces opérations ne sont pas contrôlables par un compilateur et peuvent révéler des erreurs à l'exécution. Les tests unitaires sont des garde-fous qui assurent autant que possible contre ce genre d'erreurs. Le faible typage de PHP est à la fois un confort et une grande source de problèmes. Là encore, les tests unitaires permettent de valider les types de données attendus.

Mais les tests unitaires ont aussi été employés lors de la recherche de technologies adaptées au projet. Ils ont été des preuves de concept, mais aussi des outils d'apprentissage et outils de transmission. En effet, un test repose sur des entrées et un attendu en ce qui concerne les sorties. Les tests peuvent être rassemblés et ordonnés, permettant l'écriture d'un manuel de faisabilité et d'usage. Ajoutons à cela qu'ils permettent aussi, dans une certaine mesure, d'évaluer les performances d'une technologie ou d'une façon de l'utiliser. Ce qui peut être étendu par la suite au rôle réel d'un test unitaire, en testant si telle ou telle opération se déroule bien en dessous d'un temps donné.

Les tests unitaires ont été aussi employés pour la mise en place des tests d'intégration de l'API REST. En effet, celle-ci a été conçue dans un souci de responsabilité unique, essentiellement de par son approche CRUD : un appel (requête) correspond à une opération de type ajout, récupération, mise à jour ou effacement d'une entité donnée. Cette pseudo atomicité s'accorde bien avec le principe des test unitaire, et ce d'autant plus que le triptyque Symfony-Doctrine-PHPUnit est parfaitement adapté :

- Symfony intègre dans ses extensions un pseudo-client Web en mesure de simuler des appels réels ;
- Doctrine permet la connexion à un SGBD maintenu uniquement en mémoire (*in-memory*), dégageant tous les problèmes de configuration, stockage et performances ;
- PHPUnit est le canevas standard de PHP pour les tests unitaires ((PHPUnit s.d.).

Chaque appel de l'API REST est donc testé en profondeur, autant pour les scénarios nominaux (tout se passe bien) que les scénarios alternatifs (gestion des exceptions).

4.2.2. Couverture de code

En parfaite complémentarité avec les tests unitaires, on trouve les outils de couverture de code. Ceux-ci analysent un programme en cours d'exécution afin d'établir quelle proportion de code sur l'ensemble des sources est effectivement exécutée.

La couverture de code est une option intégrée dans PHPUnit (Code Coverage Analysis s.d.). Le résultat est un rapport au format HTML qui permet de voir autant globalement que localement le code qui a été exécuté ou ignoré.

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total	75.02%	1817 / 2422		59.50%	288 / 484		39.25%	42 / 107	
Entities	71.80%	303 / 422		63.18%	163 / 258		37.93%	11 / 29	
ImporterBundle	88.89%	8 / 9		66.67%	2 / 3		75.00%	3 / 4	
Importers	90.48%	722 / 798		70.41%	69 / 98		55.56%	15 / 27	
RestApiBundle	81.14%	727 / 896		57.14%	48 / 84		28.00%	7 / 25	
SuperUserBundle	88.89%	8 / 9		66.67%	2 / 3		75.00%	3 / 4	
UpgraderBundle	3.88%	9 / 232		6.45%	2 / 31		18.75%	3 / 16	
EntityManagerProvider.php	71.43%	40 / 56		28.57%	2 / 7		0.00%	0 / 2	

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total	82.44%	587 / 712		56.52%	39 / 69		15.79%	3 / 19	
AdminController.php	96.88%	31 / 32		0.00%	0 / 1		0.00%	0 / 1	
AuthenticatedController.php									
AuthenticatedRestApiController.php							100.00%	1 / 1	
DefaultController.php	0.00%	0 / 18		0.00%	0 / 2		0.00%	0 / 1	
ExportDeltaController.php	100.00%	31 / 31		100.00%	2 / 2		100.00%	1 / 1	
MailController.php	0.00%	0 / 12		0.00%	0 / 1		0.00%	0 / 1	
ProposalCreateUpdateDeleteController.php	95.45%	21 / 22		66.67%	2 / 3		0.00%	0 / 1	
ProposalDuplicateController.php	71.43%	30 / 42		0.00%	0 / 3		0.00%	0 / 1	
ProposalFindController.php	72.31%	47 / 65		50.00%	2 / 4		0.00%	0 / 1	
ProposalLineMediaController.php	88.31%	68 / 77		16.67%	1 / 6		0.00%	0 / 1	
ProposalLineProductController.php	96.97%	32 / 33		50.00%	1 / 2		0.00%	0 / 1	
ProposalLineTextController.php	95.65%	44 / 46		81.82%	9 / 11		0.00%	0 / 1	
ProposalRetrieveController.php	88.19%	112 / 127		50.00%	3 / 6		0.00%	0 / 1	
ProposalStatsController.php	60.42%	29 / 48		33.33%	1 / 3		0.00%	0 / 1	
QuestionController.php	100.00%	26 / 26		100.00%	3 / 3		100.00%	1 / 1	
RestApiController.php	93.75%	15 / 16		87.50%	7 / 8		0.00%	0 / 1	
TestsController.php	23.08%	3 / 13		60.00%	3 / 5		0.00%	0 / 1	
UserController.php	95.83%	23 / 24		50.00%	1 / 2		0.00%	0 / 1	
UsersManagementController.php	95.12%	39 / 41		50.00%	2 / 4		0.00%	0 / 1	
VisitController.php	92.31%	36 / 39		66.67%	2 / 3		0.00%	0 / 1	

Figure 4-8 : deux exemples de rapports de couverture de code fournis par PHPUnit

ExpoTouch reste un projet d'envergure raisonnable qui permet de fixer un objectif de couverture de code assez élevé dans une fourchette de 80% à 100%.

L'analyse de la couverture de code est particulièrement adaptée au développement d'une API REST. En effet, une API est avant tout une liste de fonctions et procédures rassemblées dans une interface. Au fur et à mesure de l'avancée des spécifications fonctionnelles, puis du développement, leur nombre croît de façon conséquente. Il devient alors nécessaire de s'assurer que toutes ont été testées, et surtout que tous leurs chemins d'exécution ont été couverts. Sans les rapports de couverture, on peut vite oublier certaines parties du projet et garantir une fiabilité logicielle trop partielle.

Par ailleurs, dans le cadre du suivi de projet, l'évolution de la couverture permet de détecter aisément les sous-parties qui sont en difficultés. Cela contribue à maintenir la cohérence du développement et à rediriger les priorités si nécessaire. Le cas plus caractéristique rencontré sur ExpoTouch a été la gestion des erreurs. La couverture de code a très vite mis en évidence un oubli récurrent des tests des exceptions. De fait, les erreurs (exceptions, événements imprévus) mal traitées étaient sources de mauvaises analyse et de perte de temps. Avec la couverture, la décision a été prise de favoriser prioritairement le chemin d'exécution des exceptions avant d'aller plus avant dans le développement l'API.

4.3.Intégration continue

Afin de garantir la fiabilité du logiciel en production, un système d'intégration continue et de validation est mis en place. Le principe use encore et toujours des possibilités offertes par la virtualisation : on peut à moindre coût (en temps, énergie et argent) mettre en place une machine virtuelle faisant office de serveur d'intégration.

Ses responsabilités sont multiples :

- Servir de base de stockage des sources et ressources versionnées ;
- Etre le plus semblable à la machine de production ;
- Recevoir les modifications destinées à la production (sources, configurations, données…) ;
- Tester et valider des modifications de façon automatique et manuelle et émettre un rapport ;
- Permettre ou non le passage en production des modifications (automatisation du déploiement).

La virtualisation joue encore à plein : le serveur d'intégration peut tout aussi bien être local que sur un réseau. Dans le premier cas, chaque développeur peut simuler une mise en production au plus près de la réalité (voir 5.1.1). Dans le second cas, le serveur est l'antichambre de la production et fait office de barrière en cas de soucis. L'un et l'autre peuvent s'additionner : le développeur valide son intégration en local, puis l'exécute sur la machine partagée pour le projet.

4.3.1. Outils de gestions de sources

La base de fonctionnement d'un processus d'intégration continue repose sur les sources. Il s'agit en premier lieu de les recevoir, les stocker et les organiser. Pour cela, un outil de gestion de sources est nécessaire.

Dès lors que l'on commence à manipuler une certaine quantité de fichiers – sources ou ressources au sein d'un projet – il devient indispensable de suivre leur évolution et de permettre leur partage. C'est de ce constat que sont nés les outils de suivi de versions. Les plus connus sont Concurrent Versioning System (CVS), Apache Subversion (SVN), Microsoft Visual SourceSafe (VSS). Ils ont la particularité de reposer sur une architecture client-serveur non distribuée. Le serveur stocke et met à disposition les évolutions des fichiers envoyés par les intervenants du projet. Un système de verrou et de comparaisons (*diff*) permet

éventuellement de résoudre les problèmes d'accès concurrents, voire d'automatiser la fusion de travaux partagés. Par ailleurs, ils peuvent aussi avoir un rôle organisationnel via les branches (dérivation d'une version) et les tags (marquage d'une version).

Depuis quelques années, une nouvelle solution est apparue et a définitivement pris le pas sur les autres. Il s'agit de Git (Git s.d.), développé par certains responsables du noyau Linux, avec une forte implication du créateur de Linux en personne, Linus Torvald. Git se distingue par une volonté de revoir l'emploi des outils de versions pour s'orienter vers une approche plus globale. A ce titre, il est à la fois décrit comme un système de fichiers et un système de gestion de sources (SCM – *Source Code Management*). On n'observe plus le projet sous une forme purement temporelle (les versions) et centralisée (le serveur) mais plutôt comme un arbre dont la topologie est totalement libre (des instantanés) stockée en local et/ou sur un ou plusieurs serveurs (voir figure 4-9).

Si une solution « traditionnelle » de type SVN aurait très bien pu suffire dans le cadre d'ExpoTouch, j'ai porté mon choix sur Git pour son approche plus moderne et plus souple ainsi que pour ses possibilités d'exploitation comme outil d'intégration continue.

La souplesse de Git se situe à plusieurs niveaux. Tout d'abord, c'est une solution décentralisée : on n'est plus dépendant d'un serveur. Le stockage des données est en premier lieu local sur la machine de développement, et peut éventuellement être mis à disposition sur un ou plusieurs serveurs. Le fait de pouvoir travailler localement retire une contrainte majeure des solutions d'ancienne génération. L'intégralité de l'historique du projet est disponible et évolutif à tout instant, avec comme seul point faible de la chaîne le système de stockage en local. C'est nettement plus sécurisant et plus fiable que l'accès via un réseau à une unique machine distante que l'on doit maintenir.

Git se comporte comme un enregistreur d'instantanés (*snapshots*) du projet : à la demande de l'utilisateur, tous les fichiers d'une arborescence locale (sauf ceux qui sont explicitement exclus) sont sauvegardés dans un « paquet », appelé *commit,* identifié de façon unique par hachage. Ce fonctionnement est ce qui fait la distinction entre Git et les autres solutions. Ces dernières voient l'évolution individuelle des fichiers, laissant à la charge des intervenants de gérer ce qui compose une étape du projet. Git offre une vision d'ensemble où chaque instantané est indépendant et manipulable.

Les manipulations possibles sont notamment le regroupement des instantanés sous forme de branches qui peuvent se scinder et se fusionner à volonté. Ce concept donne à Git son aspect topologique : un projet se répartit en branches indépendantes qui évoluent, et parfois se rejoignent afin de constituer un état stable et fini d'une étape du projet. Ce double système d'instantanés et de branches permet alors de construire un flux de travail (*workflow*) totalement adapté au projet.

Celui choisit pour ExpoTouch est assez classique :

- Une branche *Master* qui contient les étapes exploitables en production ;
- Une branche de correctifs (*hotfixes*);
- Une branche *Development* qui sert de tronc à l'état d'avancement du développement ;
- Des branches de fonctionnalités ;
- Des branches de tests et de concepts.

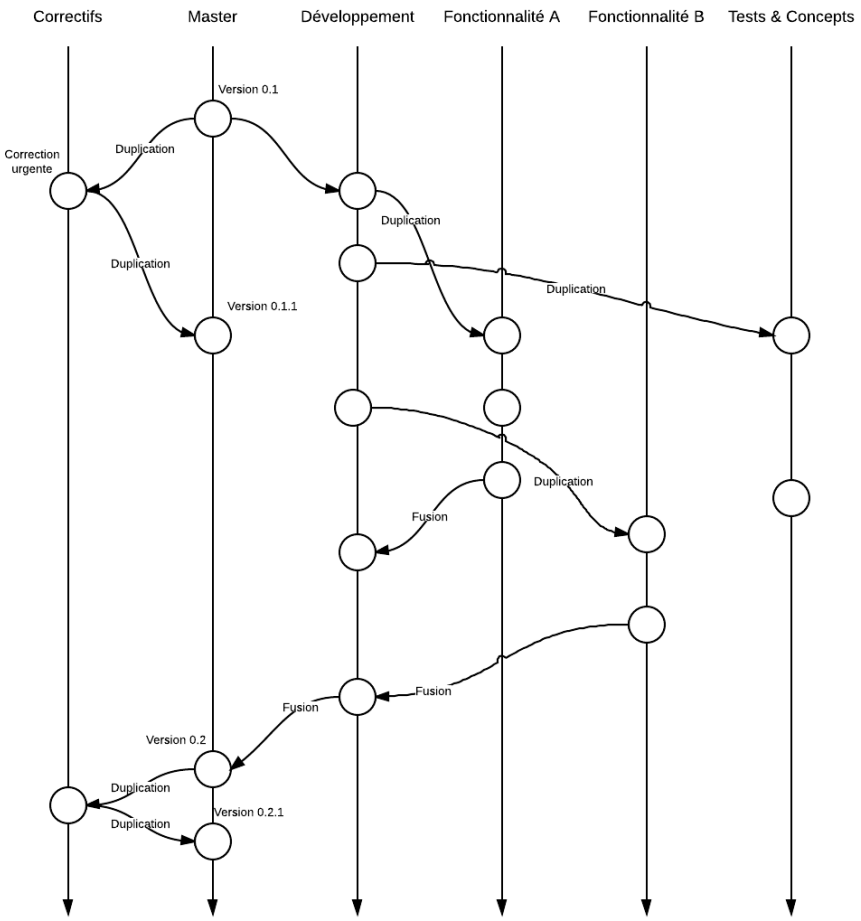


Figure 4-9 : évolution du projet sous Git : chaque barre verticale représente une branche au cours du temps. Les cercles représentent les instantanés (commits)

4.3.2. Processus d'intégration continue

L'intégration continue rassemble tous les processus nécessaires - mais aussi accessoires – à la validation du logiciel. Cela peut servir avant le déploiement, mais aussi simplement comme indicateur de bonne santé du projet. Il ne s'agit donc pas uniquement de s'assurer que l'on est prêt pour un déploiement, mais aussi de se garantir que l'architecture et le code produit respectent des standards établis au cours du temps.

Par ailleurs, autant dans une optique de déploiement que dans une recherche de maîtrise architecturale, l'intégration continue devrait idéalement pouvoir se faire par l'installation de l'environnement sur une machine vierge et le lancement des processus en une simple commande (Fowler s.d.). Cela implique l'automatisation de tous les composants du système : compilation, construction de bases de données, serveurs locaux, etc.

La toute première étape consiste à récupérer les sources et données du système. Comme pour tous les processus de l'intégration continue, on doit être en mesure de l'automatiser. Cette récupération sera le déclencheur des processus suivants.

Le processus le plus élémentaire sera la validation syntaxique du code. PHP étant interprété à l'exécution, il n'y a pas de phase de compilation, et donc de validation statique. Une coquille peut facilement être cachée au milieu des milliers de lignes des sources et provoquer un plantage irrécupérable.

Vient ensuite le processus des tests. Il y a bien évidemment les tests unitaires, mais aussi ceux d'intégration. Les premiers valident chaque composant indépendemment, les seconds assurent le bon fonctionnement des interactions entre composants. Cette partie implique une reproduction au plus près de la machine de production.

On pourra alors enclencher des processus accessoires qui exposent des métriques permettant d'évaluer la qualité du code : nombre de lignes, taille moyenne des classes et méthodes, couverture, complexité cyclomatique, répétitions, etc. On décide alors de valeurs paliers pour déterminer si le code actuel est valide ou non.

5. Réalisation

5.1.Mise en place d'une chaîne de développement

5.1.1. Machines virtuelles

Comme nous l'avons vu en 3.5, la virtualisation est l'une des technologies qui a le plus pris de l'ampleur ces dernières années, notamment au travers du *Cloud Computing*. Toutefois, elle ne se cantonne pas aux serveurs distribués et peut parfaitement s'intégrer dans un environnement de travail. Concrètement, cela se traduit par la création et l'exploitation de machines virtuelles tant pour des environnements de tests que de développement. Les avantages sont nombreux :

- On n'est plus prisonnier d'un environnement de travail : le système d'exploitation de la machine virtuelle peut-être un Linux, Windows ou même OSX. Ce choix peut résulter d'une obligation technologique (développement pour les produits Apple par exemple), mais aussi d'une préférence ou d'une normalisation au sein de la société, d'une équipe, un projet, etc.
- La multiplicité possible des systèmes d'exploitation permet de développer pour différentes plateformes beaucoup plus facilement qu'avec une machine physique pour chacune d'entre elles ;
- Le cloisonnement retire tous les soucis de compatibilités et dépendances entre les composants logiciels qui viennent s'accumuler dans un système d'exploitation unique ;
- Les machines virtuelles sont configurables en capacités, permettant ainsi d'éprouver le logiciel dans différentes conditions ;
- On peut récréer une architecture multi-machines complète et approcher au plus près de la réalité de production ;
- Parallèlement, on peut intégrer dans cette architecture virtualisée des ressources dédiées uniquement à l'intégration continue ;
- On peut prendre un instantané d'une machine virtuelle, le stocker, puis le restaurer si nécessaire. C'est donc la possibilité de procéder à des tests de non-régression au niveau même d'une architecture.

Le principe d'architecture virtualisée localement sur un poste de développement se justifie notamment par la lourdeur du processus d'intégration, puis de mise en production. Le développeur peut préparer son intégration localement, sans forcément perdre beaucoup de

temps dans toutes les étapes, puis effectuer le processus final et complet sur la machine partagée et spécialement dédiée à cela.

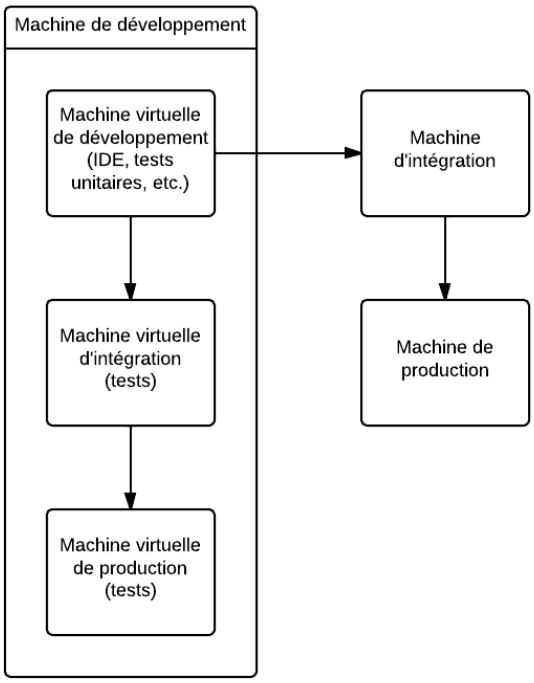


Figure 5-1 : utilisation de machines virtuelles dans la chaîne d'intégration/production

Dans le cadre d'une chaîne de développement, tous ces avantages dépassent le cadre du « confort ». Les intervenants sur le projet sont – eux aussi – découplés et réduisent leurs interdépendances. Un développeur peut manipuler une architecture complète et complexe à son niveau sans avoir à monopoliser des ressources partagées. Il peut aussi transmettre son travail sous une forme monolithique et directement exploitable. Par exemple, si de nouveaux membres rejoignent l'équipe, une simple copie des fichiers de machines virtuelles leur permet d'être presque immédiatement en situation de travailler sur le projet.

Pour ExpoTouch 2 – et dans l'optique de le généraliser à l'ensemble des projets de la société - le logiciel de virtualisation choisi est Oracle VirtualBox. C'est une solution open-source et multiplateformes, initialement développée par Innoteck. Elle a pris ses lettres de noblesse lors de son rachat par Sun Microsystems (ensuite acquis par Oracle). Son installation et sa configuration sont très simples sans poser de limites particulières. On peut notamment concevoir des machines virtuelles en réseau privé, avec un double écran, plusieurs cœurs pour le processeur, etc. Chaque machine consiste en quelques fichiers qui ne nécessitent aucune manipulation particulière pour le transfert, le clonage ou la copie. On peut même lancer une machine virtuelle hébergée sur un disque réseau.

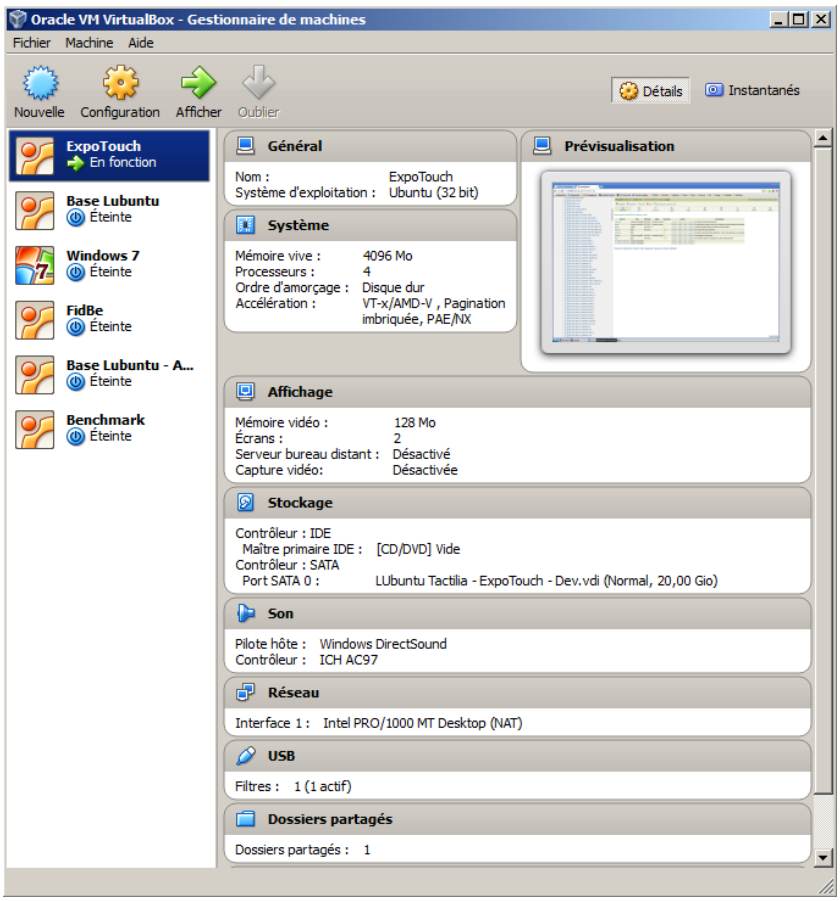


Figure 5-2 : capture d'écran de la fenêtre principale de VirtualBox. On y distingue notamment plusieurs machines virtuelles (à gauche) et les paramètres de l'une d'elle

5.1.2. Environnement de développement

Pour des raisons de coûts et de performances, la machine virtuelle de développement abrite le système d'exploitation Lubuntu. Il s'agit d'un Linux dérivé de la distribution Ubuntu, elle-même basée sur la distribution Debian. Lubuntu se distingue de son « parent » par sa légèreté : elle est en mesure de fonctionner correctement sur des petites configurations et requiert peu d'espace disque. Cela permet de garantir de bonnes performances – et donc un bon confort de développement – dans des machines virtuelles tournant sur une large gamme d'hôtes physiques, et non pas uniquement sur des PC de dernière génération. C'est aussi la possibilité de transférer l'environnement complet via une simple clef USB de 16 à 32 Go.

Le confort de développement est aussi recherché au travers de l'EDI (Environnement de Développement Intégré). Le choix s'est porté sur Netbeans : open-source et officiellement supporté par Oracle, il se veut le pendant « *user-friendly* » d'Eclipse. On y trouve toutes les fonctionnalités d'un EDI moderne, avec une bonne intégration du PHP : reformatage automatique configurable (voir 5.1.3.2), colorisation des sources adaptée, détection d'erreurs

syntaxiques et avertissements à la volée, gestion des espaces de noms, navigation intelligente, etc.

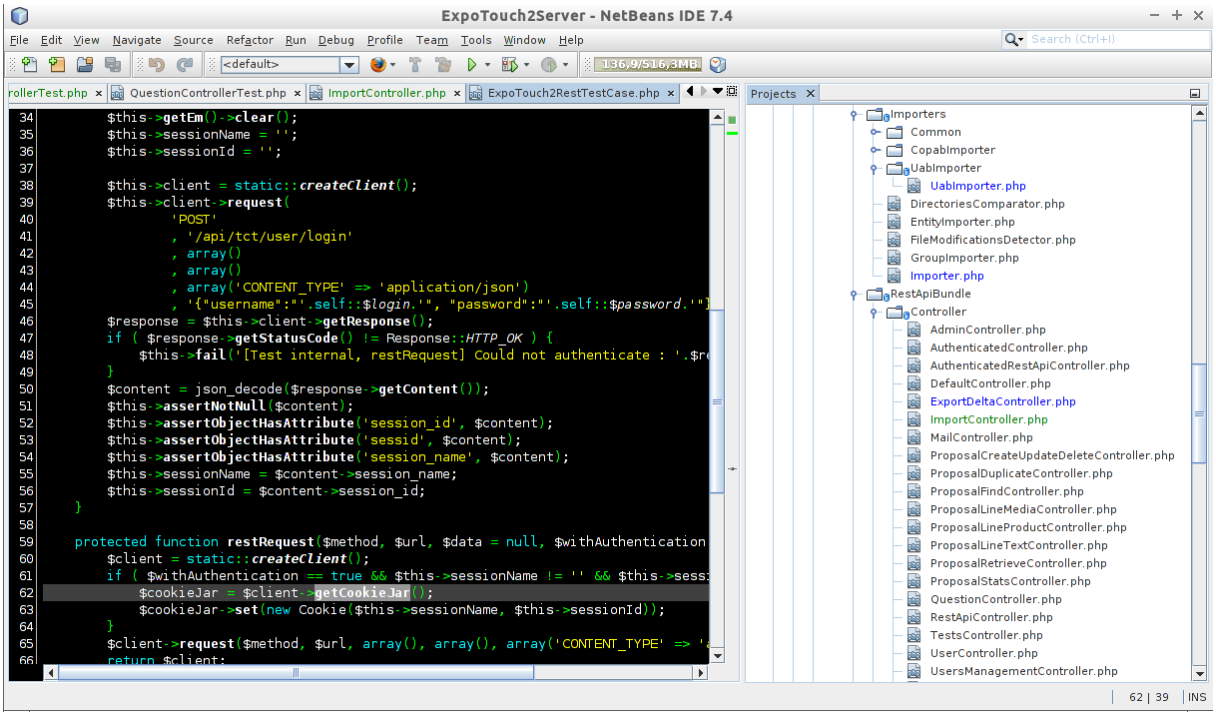


Figure 5-3 : environnement de développement intégré Netbeans 7.4

5.1.3. Bonnes pratiques et conventions d'écriture

La phase d'écriture du code est un nœud critique du développement d'un projet. Son importance réside notamment dans son exposition : lors de la transmission ou de l'évolution, c'est ce qui sera essentiellement vu et manipulé. Il convient alors d'être exigeant et rigoureux sur la qualité du code source produit, mais aussi sa segmentation, sa documentation et ses choix méthodologiques.

On peut distinguer trois principaux axes concernant l'écriture :

- La structure d'arborescence des fichiers sources : il s'agit de pouvoir se repérer aisément dans l'ensemble du projet ExpoTouch, c'est-à-dire retrouver rapidement qui fait quoi ;
- Les conventions d'écriture : le code est l'expression des développeurs. Il faut maintenir une cohérence dans la façon d'écrire afin que chacun puisse relire le travail des autres sans avoir à le décrypter ;
- La méthodologie : des règles à suivre au plus près pour maintenir une cohérence sémantique : chacun doit pouvoir comprendre le travail de l'autre parce qu'il respecte les mêmes approches.

Pour conduire la prise de décision, on trouve dans l'écosystème PHP le *Framework Interoperability Group* (FIG) qui fournit des *PHP Specification Request* (PSR). Il s'agit de recommandations discutées par les représentants des principaux canevas PHP du marché, dont Symfony (PHP Framework Interop Group s.d.). ExpoTouch exploite quatre d'entre elles (sur cinq), toutes étant utilisées par Symfony :

- PSR-0 : qui décrit une convention pour l'organisation des fichiers, des classes et des espaces de noms, utilisée par Symfony ;
- PSR-1 : qui décrit les éléments standards de codage pour une interopérabilité maximale. On y trouve par exemple l'encodage des fichiers sources (UTF-8) ;
- PSR-2 : qui décrit le style d'écriture ;
- PSR-3 : qui prend en charge l'interfaçage des outils de journalisation.

5.1.3.1. Structure d'arborescence :

ExpoTouch profite déjà de l'emploi de Symfony et qui suit PSR-0 pour cadrer l'écriture et l'organisation des sources. L'arborescence des fichiers sources respecte le même fonctionnement que Java, à savoir un dossier par sous-domaine de l'espace de nom. Par-dessus, Symfony organise le projet en composants (*bundles*) qui peuvent être une application Web ou une de ses sous-parties, mais aussi un outil en ligne de commande.

C'est aussi ce qui fait la force d'un canevas : une fois l'approche imposée maîtrisée et partagée, la productivité et le confort sont décuplés. Toutefois, cela reste aussi un ensemble de contraintes et il convient de bien comprendre les intentions des concepteurs de Symfony afin d'organiser de façon cohérente les fichiers. Sinon, l'effet est inversé et la maintenance devient très vite infernale, voire impossible.

5.1.3.2. Conventions d'écriture

Un style d'écriture commun et continu va au-delà d'une simple question de confort. On n'écrit pas seulement du code pour soi, mais aussi pour les autres (Martin 2009). Etant donné que le PHP ne définit pas formellement de style, il faut décider au lancement d'un projet de l'ensemble des règles qui régiront l'écriture. On distingue trois aspects :

- Tout ce qui a trait à la spatialité du code : placement des espaces, longueur et type des tabulations, interlignage et délimitation des blocs ;
- Les modalités d'identification : façon d'écrire une classe, ses attributs, des variables locales, etc.

- Les commentaires : intégration d'une documentation automatique, formats, placements, etc.

Ce sont généralement des choix personnels, mais j'ai opté très logiquement pour ceux de Symfony. Ils sont documentés précisément (voir annexe 2) et ont un niveau de contrainte assez standard :

- Nommage en camelCase (première lettre en minuscule, mots suivants avec une capitale) ;
- Espacement autour des opérateurs, après les virgules (comme en typographie française) ;
- Style de blocs double : aéré pour les classes et méthodes, condensé pour les boucles et les conditions.

5.1.3.3. Méthodologie et sémantique

N'hésitons pas à le répéter : on n'écrit pas du code pour soi, mais aussi pour les autres. Il faut donc avoir la volonté d'écrire un code qui s'exprime « vers l'extérieur ». Cela implique non seulement le respect de certaines règles, mais surtout un effort de style qui cherche le plus grand dénominateur commun.

Le premier d'entre eux à déterminer est celui du sens de lecture. Longtemps on a « empilé » les fonctions avant leurs appelants à cause des exigences des compilateurs qui réclamaient une déclaration avant tout usage. C'est notamment le cas du langage C, ancêtre lointain de nombreux langages actuels. Dorénavant, la contrainte de pré-déclaration étant levée, on privilégie le sens de lecture naturel du haut vers le bas (Martin 2009).

Pour continuer dans cette volonté de « lecture naturelle », on cherchera à employer des noms de classes, attributs, méthodes, etc. aussi explicites que possible. Cela implique l'effort de passer un minimum de temps à choisir chaque nom, et ainsi de donner du sens à ce que l'on écrit. On gagne alors non seulement en clarté pour les autres, mais aussi pour soi-même : à bien identifier ce que l'on manipule, on le maîtrise d'autant plus.

Enfin, pour exploiter au mieux le sens de lecture et le nommage recherché, on écrira des classes et méthodes courtes (Martin 2009). Associées à un nommage exigeant, elles imposent un découpage précis du code. Chaque « brique » doit faire une et une seule chose (et le faire bien) : c'est le principe de responsabilité unique (*Single Responsability Principle* – SRP). Les avantages sont nombreux :

- Chaque élément est susceptible d'être testé unitairement, offrant ainsi une meilleure garantie de fiabilité ;
- Le rôle confié à une classe ou une méthode lui donne implicitement un sens, donc de la compréhension ;
- Il est plus facile de mémoriser une fonctionnalité unique que plusieurs situées dans un même endroit.

5.2.Architecture détaillée

5.2.1. Modèle de données

Le modèle de données a une importance toute particulière au sein d'ExpoTouch 2. En effet, on cherche à passer d'une solution spécialisée pour chaque client à une solution générique. Le modèle initial (spécialisé) repose sur celui de Drupal. Ce dernier est « universel » dans le sens où son modèle relationnel sert à décrire et stocker un modèle relationnel adapté aux objectifs fonctionnels (que nous appellerons le « modèle fonctionnel »). De plus, il intègre un système de sauvegarde qui permet de restaurer des données obsolètes. Pour ce faire, Drupal décompose tous les éléments (entités, attributs) du modèle fonctionnel sous la forme d'entités relationnelles. Pour les sauvegardes, il duplique les entités et les attributs.

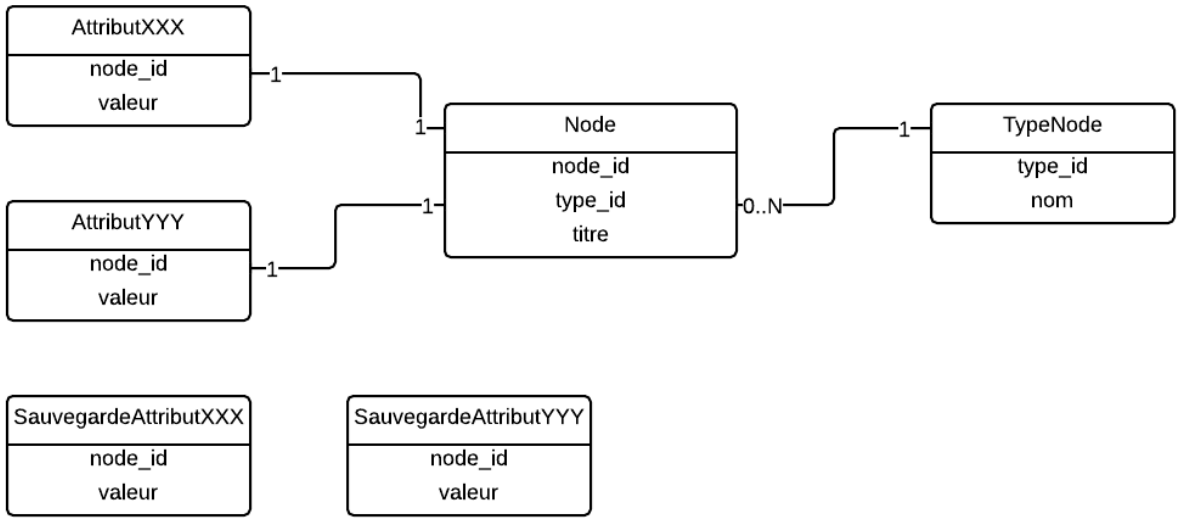


Figure 5-4 : modèle de données de Drupal

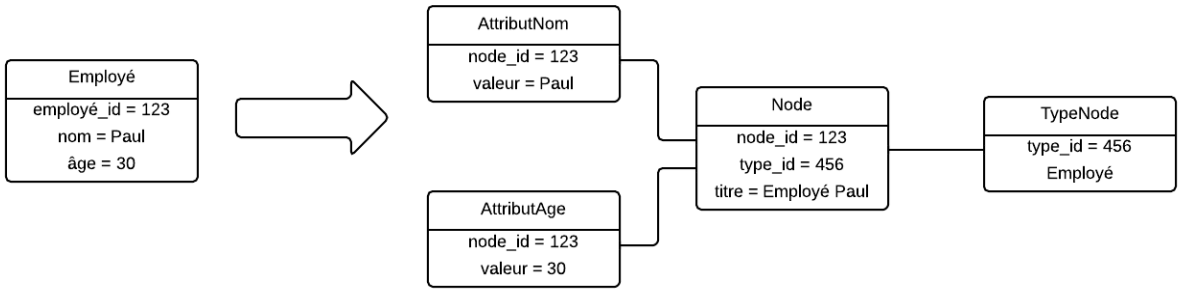


Figure 5-5 : application du modèle de données de Drupal à un exemple simple

Le résultat est un modèle relationnel qui permet de créer n'importe quel modèle fonctionnel et de le maintenir dans le temps. Si ces fonctionnalités sont séduisantes au premier abord, elles

posent un certain nombre de contraintes qui peuvent se révéler inadaptées aux exigences d'un projet.

En premier lieu, le modèle Drupal faisant office d'interlocuteur entre le modèle fonctionnel et le SGBD, l'opacité qui en résulte rend pratiquement impossible des manipulations au sein même du SGBD. Il faut alors s'en remettre totalement à Drupal, soit par son interface Web, soit par ses API. Dans les deux cas, la moindre opération, aussi simple soit-elle, devient un processus complexe et chronophage si l'on n'est pas un spécialiste Drupal.

Par ailleurs, les performances en termes de temps d'accès et d'espace disque se dégradent d'autant plus vite que le modèle fonctionnel est complexe. Chaque attribut de chaque entité du modèle fonctionnel se voit stocké, sauvegardé et indexé. Ce qui provoque :

- Des jointures multiples qui complexifient les accès aux données, même les plus basiques. Une liste de produits avec trois attributs provoquera trois jointures ;
- Des index multiples pour chaque attribut (pour limiter les conséquences des jointures multiples) qui seront stockés en plus des données. L'espace disque consommé est alors massivement plus important.

Ce modèle « universel » a toutefois l'avantage d'intégrer tous les modèles fonctionnels que l'on veut s'ils respectent la nomenclature des SGBD relationnels (entités, attributs et clefs étrangères). ExpoTouch étant destiné à devenir une solution sous forme de licence, on doit pouvoir manipuler le modèle de données de manière à ce que le code n'ait pas à être modifié pour chaque client. On écartera donc d'office une représentation qui colle strictement aux fonctionnalités d'une coopérative en particulier. Mais comme on l'a vu, il est hors de question de reprendre le modèle universel de Drupal, bien trop consommateur de ressources, et source de complexités.

J'ai donc choisi un intermédiaire qui cherche à se cantonner au périmètre fonctionnel, tout en permettant l'évolutivité. Pour cela, deux approches ont été utilisées en parallèle :

- Celle du « qui peut le plus, peut le moins », où l'on acceptera une certaine quantité de données mortes : il y aura des attributs inutilisés par certaines coopératives ;
- L'application du modèle universel sur certains aspects fonctionnels.

Pour la définition, plutôt qu'une approche entité-relation, c'est une conception orientée objets qui a servi de base de travail. Cela est dû au fait que le projet sera utilisera un ORM

(Doctrine) capable d'intégrer parfaitement dans le modèle relationnel les spécificités objets tels que l'héritage ou la composition.

Concrètement, des entités ont été conçues pour répondre aux spécifications fonctionnelles communes à toutes les versions d'ExpoTouch 1. On retrouve donc les entités du diagramme de classes (voir figure 3.22)

Les entités restantes peuvent s'intégrer dans un modèle universel. Celui de Drupal a été modifié afin de le simplifier. Les révisions ne sont pas nécessaires, et la décomposition typée des attributs beaucoup trop lourde. On ne conserve alors qu'une représentation minimum où l'entité de base agrège des attributs, et peut s'insérer dans des groupes. Comme on le voit dans la figure 5-6, un groupe est une entité, un attribut est une entité. Il s'agit de l'extension du patron de conception Composite appliqué à un modèle de données (Gamma, et al. s.d.).

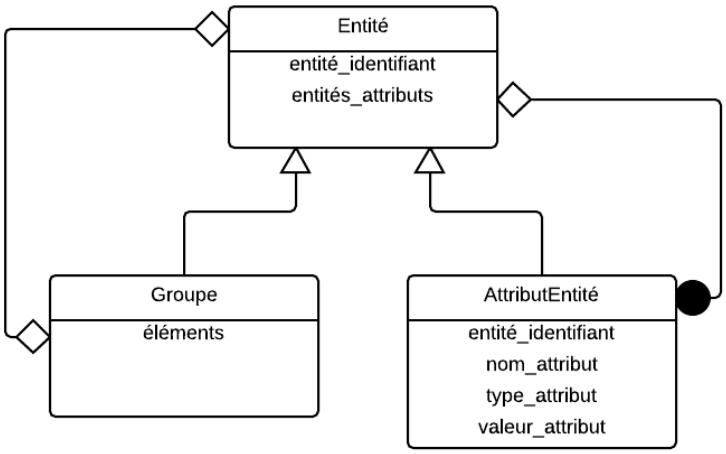
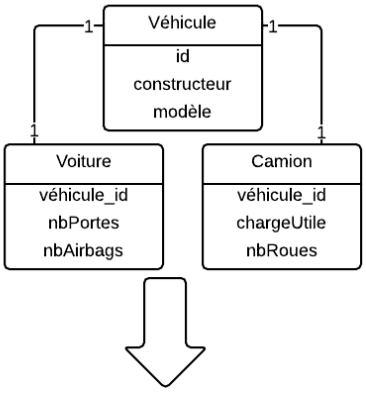


Figure 5-6 : diagramme de classe du schéma universel de représentation des données

Cette approche confie donc à l'ORM Doctrine le soin de créer les entités et leurs relations au sein d'un gestionnaire de bases de données. Doctrine est notamment capable d'intégrer la notion d'héritage dans un schéma relationnel. Pour ce faire, il propose deux solutions :

- Ajouter à chaque entité fille une colonne contenant une clef vers l'entité parente. L'accès aux données se fait alors par jointures, ce qui implique un coût en performances, voire en espace via les index ;
- Fusionner les entités filles et parentes dans une seule table qui contient donc tous les attributs de toute la hiérarchie. Cette approche monolithique s'éloigne du cadre relationnel

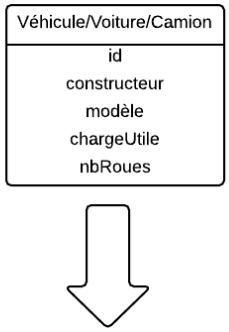
et accumule les données mortes : un rang de la table contiendra des champs utiles que pour une entité donnée dans la hiérarchie. Les autres seront vides.



Véhicule		
id	contructeur	modèle
1	Peugeot	206
2	Iveco	IrisBus
3	Renault	Twingo
4	Scania	V8

Voiture		
véhicule_id	nbPortes	nbAirbags
1	4	2
3	2	2

Camion		
véhicule_id	chargeUtile	nbRoues
2	5	6
4	8	8



Véhicule/Voiture/Camion						
id	constructeur	modèle	nbPortes	nbAirbags	chargeUtile	nbRoues
1	Peugeot	206	4	2	NULL	NULL
2	Iveco	IrisBus	NULL	NULL	5	6
3	Renault	Twingo	2	2	NULL	NULL
4	Scania	V8	NULL	NULL	8	8

Figure 5-7 : comparaison des deux types de schémas possibles générés par Doctrine pour représenter un héritage

Pour ExpoTouch, le choix a été le système de jointures pour deux raisons :

- Il est plus proche de l'esprit objet et moins « brut » qu'une table unique avec des données mortes. C'est une caractéristique non négligeable lorsqu'il s'agit de déboguer les manipulations sur les données ;
- Il est plus accessible et permettra d'ouvrir la base de données sous-jacente aux DSI des coopératives qui voudront requêter directement en plus d'utiliser les interfaces fournies par ExpoTouch.

Le modèle ainsi obtenu repose sur une base fixe issue des spécifications fonctionnelles, à laquelle s'intègre des éléments variables reposant sur le modèle universel. Pour ce dernier, l'usage principal se retrouve dans des familles et sous-familles où vont se ranger les produits. Dans la première version d'ExpoTouch, les différentes coopératives utilisaient chacune une forme de classement. Dans l'une, il n'y avait que deux familles non hiérarchiques, dans l'autre trois familles hiérarchisées. Avec le nouveau modèle, toutes les coopératives actuelles et futures reposent uniquement sur celui-ci.

La description du modèle de données dans Doctrine se fait par le biais d'annotations. Il ne s'agit pas de réelles annotations, puisque celles-ci n'existent pas dans le langage PHP, mais de commentaires écrits comme des annotations. Un analyseur propre à Doctrine profite du mode interprété du langage (les sources sont obligatoirement lisibles) pour déterminer le schéma relationnel à créer. L'entité Item précédemment décrite se définit donc (partiellement) ainsi :

```
/**
 * @ORM\Entity, @ORM\InheritanceType("JOINED")
 * @ORM\DiscriminatorColumn(name="EntityType", type="string")
 * @ORM\DiscriminatorMap({
 *      "Business" = "Business"
 *      ,"`Group`" = "Group"
 *      ,"ItemProperty" = "ItemProperty"
 *      ,"Item" = "Item" })
 */
class Item {
    /**
     * @ORM\Id @ORM\Column(type="integer", length=64, unique=true)
     * @ORM\GeneratedValue(strategy="AUTO")
     * @ORM\OneToOne(targetEntity="Item")
     */
    private $id;
    /**
     * @ORM\ManyToOne(targetEntity="Business")
     */
    private $business;
    /**
     * @ORM\OneToMany(targetEntity="ItemProperty", mappedBy="item", cascade={"all"})
     */
    private $properties;
    /**
     * @ORM\ManyToMany(targetEntity="Group", inversedBy="elements")
     */
    private $groups;
```

Il s'agit donc d'une classe simple (*Plain Old PHP Object* – POPO) à laquelle on ajoute des méta-informations. Chaque attribut de la classe trouve une correspondance dans le modèle relationnel avec des caractéristiques qui lui sont propres : type, taille, unicité, etc.

Toute la puissance de l'ORM se retrouve dans la possibilité de lier les entités entre elles – de créer des relations – sous différentes formes. On peut alors manipuler des structures relationnelles complexes sans accéder aux sous-couches inférieures de la couche de persistance (typiquement : Mysql).

Par exemple, une entité Contact comprend zéro ou plusieurs adresses. Cela se traduit par une relation un-à-plusieurs (*One-To-Many*) avec des méta-informations de référencements mutuels :

```
class Contact extends Item {
    /**
     * @ORM\OneToMany(targetEntity="Address", mappedBy="contact", cascade={"all"},
orphanRemoval=true)
     */
    private $addresses;
```

```
class Address extends Item {
    /**
     * @ORM\ManyToOne(targetEntity="Contact", inversedBy="addresses")
     * @ORM\JoinColumn(name="contact_id", referencedColumnName="id", onDelete="CASCADE")
     */
    private $contact;
```

5.2.2. Architecture Symfony

Le canevas Symfony est en mesure de contenir intégralement la solution ExpoTouch, tant au niveau de l'application Web, que les outils en ligne de commandes. Chaque sous-système fonctionnel fait l'objet d'un *bundle* : il s'agit d'une sous-partie indépendante dans l'architecture Symfony. Nous trouverons donc :

- SuperUserBundle : gestion des administrateurs ;
- ImporterBundle : importation du catalogue et des outils marketing ;
- UpgraderBundle : transfert des données de la version 1 vers la version 2 (mise à niveau) ;
- RestApiBundle : API REST ;
- AdminBundle : module d'administration et consultation des statistiques.

Les trois premiers *bundles* sont des outils en ligne de commande. Symfony fournit les classes et la méthodologie pour concevoir et manipuler facilement ce type de logiciel. On dispose notamment d'un analyseur de paramètres, d'un afficheur de textes colorés, d'un lecteur de fichiers de configurations standardisés, etc.

Toujours dans le souci du principe d'ouvert pour extension, fermé pour modifications, les outils en ligne de commande sont de simples conteneurs qui appellent des services externes. Ainsi, l'importateur est déporté dans une librairie à part, afin d'être utilisé aussi bien dans l'application Web, que par l'outil d'importation ou bien celui de mise à niveau.

Les autres *bundles* suivent la philosophie de Symfony : ils sont décomposés en contrôleurs, services et vues. Cela se concrétise par une répartition des rôles dans des dossiers spécialement dédiés.

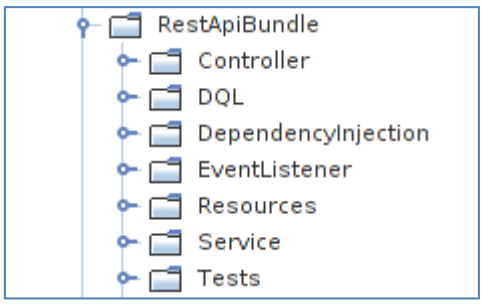


Figure 5-8 : disposition des dossiers Symfony

A l'instar de Doctrine, Symfony permet l'usage des annotations via un formatage spécifique des commentaires. Cela permet d'intégrer directement dans le code des informations de configurations que l'on retrouve généralement dans des fichiers séparés.

```
/**
 * @Route("/api/tct/stats/proposals/nb/day/{startDate}/{endDate}")
 * @Method({"GET"})
 */
public function getNbProposalsPerDay(Request $request, $startDate, $endDate) {
    . . .
    return $this->responseFromObject($response);
}
```

Il y a toutefois une divergence par rapport à l'utilisation « classique » que l'on fait de Symfony : généralement la vue est décrite par du HTML combiné à un langage d'interprétation de données dynamiques (Twig). Comme expliqué dans les parties précédentes, le serveur n'aura (presque) plus le rôle de génération de la vue. Les contrôleurs dans Symfony ne renverront donc pas du HTML/Twig, mais directement les données formatées en JSON.

5.2.3. Architecture AngularJS

Afin de produire une administration Web rapidement fonctionnelle, il a été décidé d'utiliser un gabarit (*template*) graphique basé sur AngularJS. De cette façon, on dispose d'une IHM moderne, prête à l'emploi et exploitable dans la technologie choisie. Ce gabarit, nommé Flatify (Flatify s.d.), se présente sous la forme d'un ensemble de composants ré-utilisables encapsulés dans une disposition unique (deux menus, un contenu). On dispose alors d'une IHM standardisée et cohérente avec de nombreuses possibilités en termes d'expériences utilisateur. Outre les composants classiques du HTML (boutons, champs d'entrée texte, sélecteurs, etc.) à l'apparence graphique très améliorée, on trouve aussi des composants spécifiques tels que des sliders, interrupteurs, sélecteurs de dates, bibliothèques d'icônes, etc. L'ensemble repose sur AngularJS et il suffit donc de s'insérer dedans en respectant la philosophie du canevas.

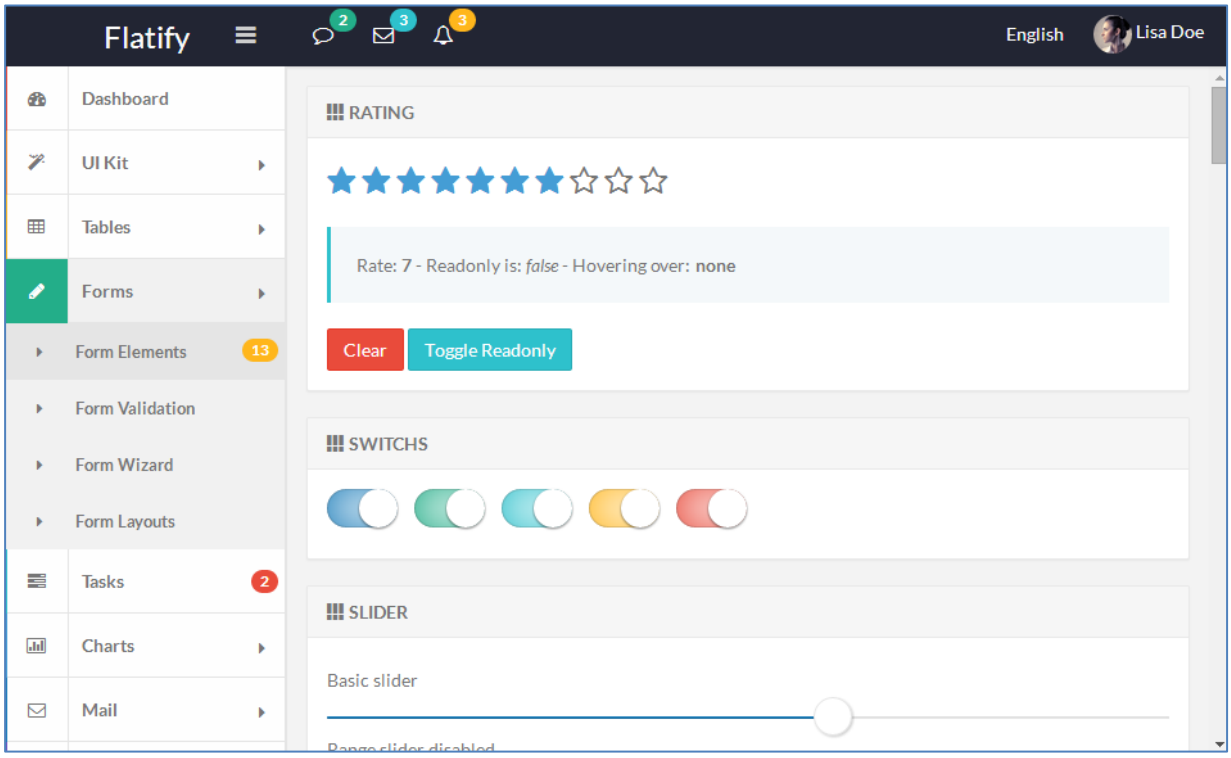


Figure 5-9 : quelques composants du gabarit Flatify

Comme vu précédemment, AngularJS repose sur des routeurs, des contrôleurs, des services et des vues. Les différentes fonctionnalités de l'administration d'ExpoTouch sont séparées dans des dossiers où l'on retrouve ces différents éléments logiciels. On trouvera donc un dossier « common » pour contenir les services et composants communs. On y trouvera aussi le contrôleur général de l'application qui gère notamment l'état connecté/déconnecté.

Les services notables sont ceux qui permettent de communiquer avec le serveur via l'API REST. Dans le souci du principe de responsabilité unique, un premier service - authHttpService - fait office de couche inférieure : son seul rôle est d'envoyer des requêtes avec un jeton de sécurité issu du service d'authentification (loginService). Au-dessus, on trouve restClientService qui va traiter spécifiquement les appels REST, notamment les codes d'erreurs. L'ensemble peut être encapsulé dans taskService qui, en conjonction avec waitingService et errorService, fournira à l'utilisateur des informations sur le déroulement des opérations.

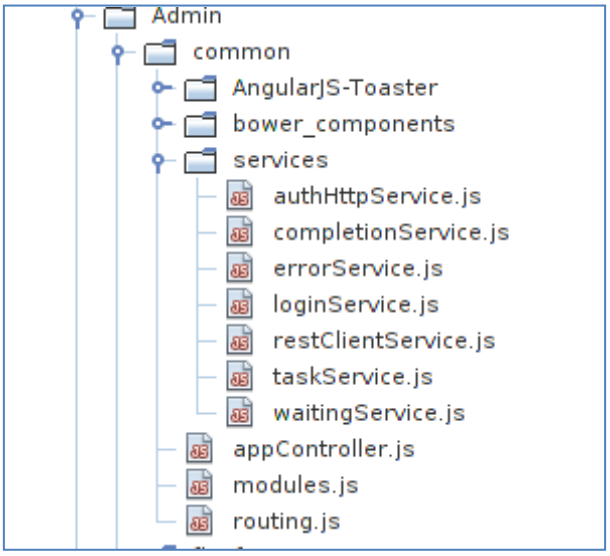


Figure 5-10 : services communs et contrôleur global

Ensuite, chaque espace fonctionnel de l'administration dispose de son propre contrôleur, propre routeur, ses propres services et sa propre vue.

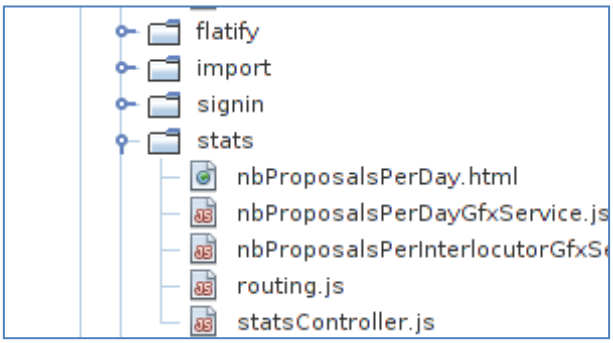


Figure 5-11 : répartition fonctionnelle

L'architecture locale propre à une fonctionnalité reste donc la même : le contrôleur gère les entrées via le service $scope en liaison avec la vue. Il utilise ces entrées pour faire appel à des services spécifiques à la fonctionnalité, et met à jour la vue.

Typiquement, dans le cadre d'un CRUD simple, tel que la gestion des tablettes, pour l'ajout d'une tablette la séquence est :

- L'utilisateur clique sur le bouton « Envoyer » après avoir rempli les champs ;
- L'événement est attaché à une fonction du contrôleur qui va :
    - Faire appel au service $scope pour récupérer les données de la vue ;
    - Faire appel au service tabletService pour effectuer l'ajout;
        - tabletService va faire appel au service restClientService pour enclencher l'appel REST au serveur ;
            - restClientService va utiliser le service authHttpService pour effectuer la requête avec un jeton de sécurité ;
                - authHttpService retournera une Promise sur le résultat de la requête ;
    - La Promise sera exploitée par le contrôleur qui en attendra l'exécution pour mettre à jour la vue.

Le code du contrôleur est disponible à l'annexe 4.

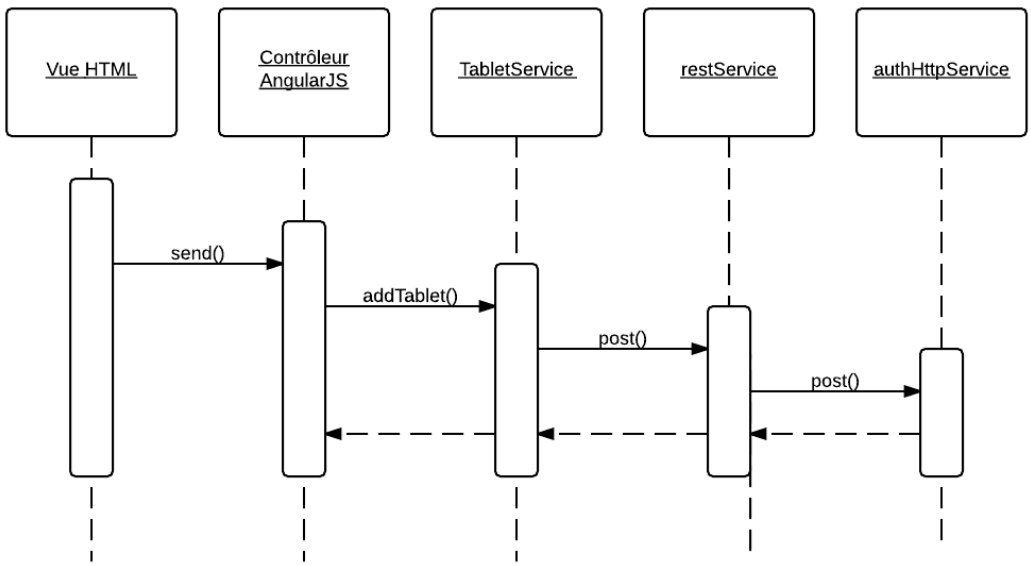


Figure 5-12 : Diagramme séquence AngularJS

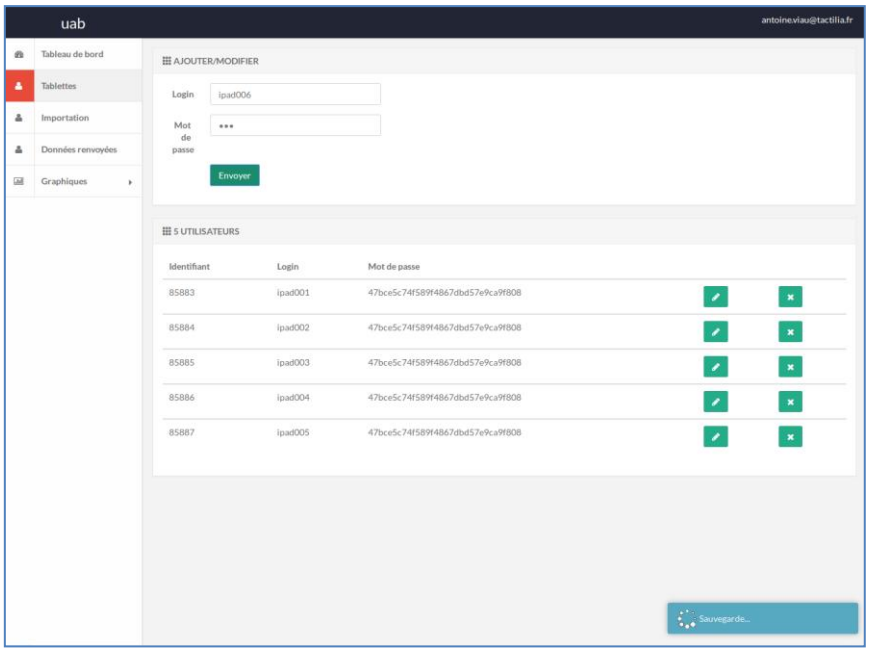


Figure 5-13 : gestion des tablettes dans l'administration

5.2.4. Conception des graphiques avec D3

Les graphiques ont été mis en place en partant de la bibliothèque d'exemples disponible sur le site de D3 (mbostock/d3 s.d.). La base visuelle est ainsi contrôlée par avance et la conception consiste en une adaptation.

Dans la logique expliquée précédemment, et dans l'optique AngularJS, chaque graphique est disposé dans un service. Celui-ci contient les interfaces nécessaires au paramétrage – typiquement des dates de début et fin de période à afficher – et la méthode de rafraîchissement. Le contrôleur est alors en charge de récupérer les données entrées dans la vue, puis de s'en servir pour paramétrer le service graphique correspondant.

Comme expliqué en 4.1.3, la construction se fait par la création et la mise à jour d'éléments SVG. Il est à noter que D3 est une librairie qui exploite au mieux nombre de subtilités du JavaScript et son « mode d'emploi » demande une certaine adaptation afin de bien en cerner le fonctionnement. De plus, étant donné le peu de modularité du JavaScript (donc du couplage lâche) et la densité de fonctions de la librairie, on perd énormément en lisibilité. A ce titre, le code est particulièrement commenté (voir annexe 3) afin que chacun, y compris l'auteur, puisse bien comprendre ce que produit le code et comment il le produit.

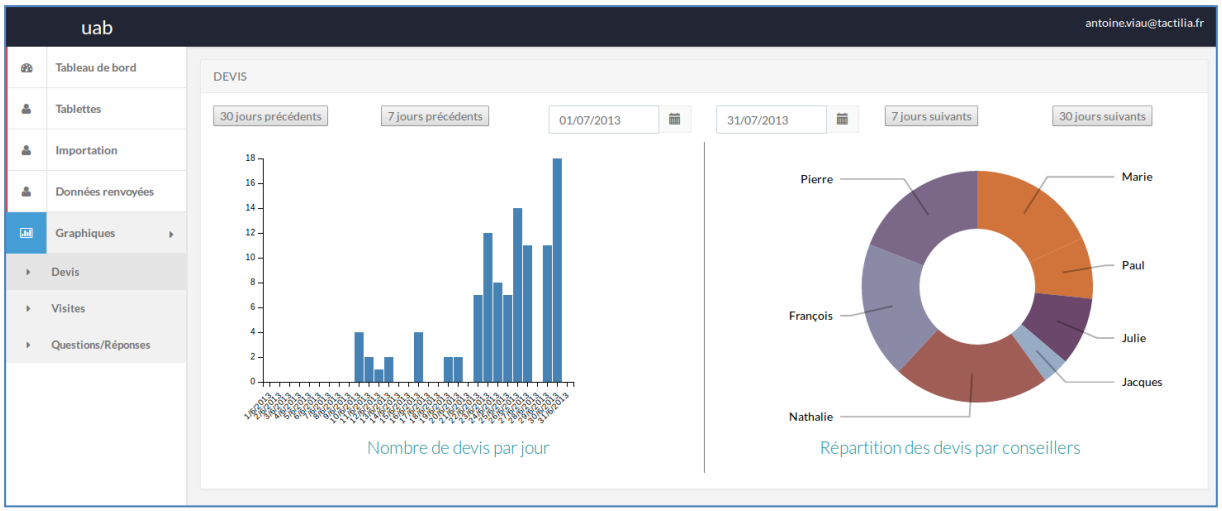


Figure 5-14 : graphiques dans l'administration avec AngularJS et D3

5.3. Mise à jour des données ExpoTouch de la version 1 à la version 2

Le transfert des données des clients historiques se fait par un outil spécifique qui exploite l'architecture en place. Ces données sont stockées dans une base Postgresql gérée par Drupal et son format opaque. Heureusement, l'accès aux données est simplifié par deux choses :

- On n'est intéressé que par les données remontées par les tablettes (devis, visites, etc.) : on n'a donc pas l'intégralité de la base à récupérer ;
- Postgresql permet la création de vues sur la base, permettant ainsi un formatage des données avant exploitation.

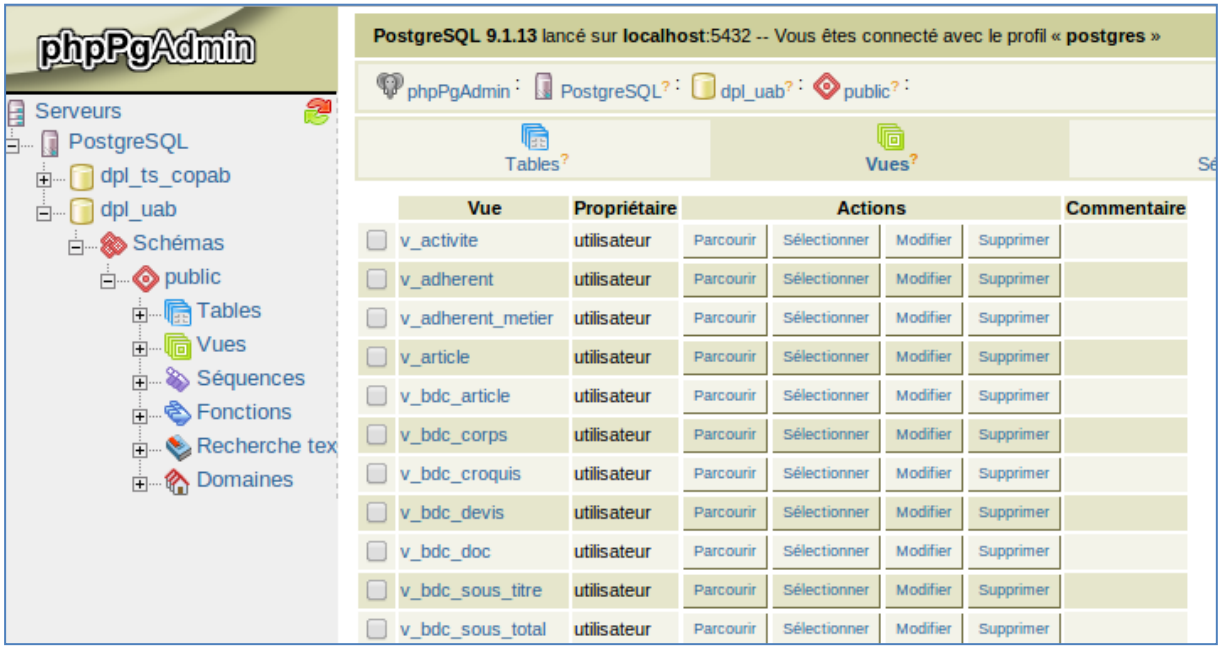


Figure 5-15 : exploitation des vues Postgresql via phpPgAdmin

L'outil de mise à jour va commencer par utiliser l'importateur pour insérer le catalogue en se basant sur les fichiers JSON. Puis, à partir des vue fournies par Postgres, il va récupérer chaque donnée et l'insérer en utilisant l'API REST. De cette façon, la mise à niveau est utilisable pour toutes les coopératives historiques sans avoir à écrire de code spécifique.

La configuration se fait par un fichier au format YAML (acronyme récursif de *YAML Ain't Markup Language*). Ce format est utilisé par Symfony pour tous ses propres fichiers de configurations. Il propose donc très logiquement une librairie pour manipuler des fichiers YAML. Celui de l'outil de mise à niveau décrit les informations d'accès aux bases de données source (Postgresql) et destination (Mysql), ainsi que les spécificités de la coopérative :

```
business:
    name: uab
    importer: UabImporter
postgres:
    host: localhost
    user: postgres
    password: antoine
    database: dpl_uab
database:
    driver: pdo_mysql
    host: localhost
    user: root
    password: xxxxxxxxxx
    dbname: expotouch2
directories:
    importFrom: /home/expotouch1/data/uab/json
    exportTo: /home/expotouch2/data/uab/
```

On y distingue en premier lieu ce qui fait la particularité de la version 2 : la transformation en licence via la mise en place d'une entité « Business ». Ainsi, une même base sera à même de gérer plusieurs coopératives. Ensuite, on trouve les informations concernant la base de données source (Posgresql). Les vues sont pré-générées manuellement et respectent une nomenclature prédéfinie.

Enfin, pour le processus d'importation, on dispose des informations de connexion à la base destination (Mysql) et les dossiers source et destination des fichiers médias. On notera que le processus d'importation est optionnel dans celui de mise à niveau. En effet, à la première mise à jour des tablettes il y a aura forcément importation. Mais le découplage des services en permet une intégration facile, c'est un confort de développement que l'on peut exploiter.

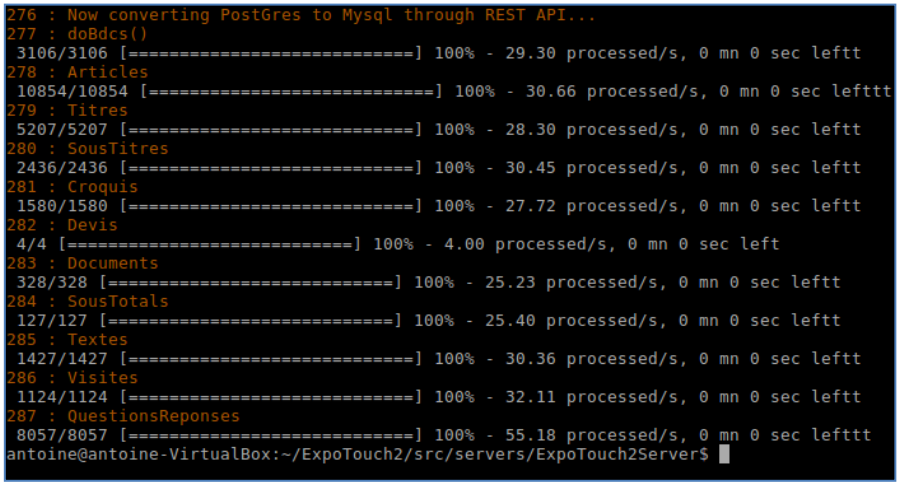


Figure 5-16 : phase d'emploi de l'API REST par l'outil de mise à niveau de la solution

5.4. Mise en place des tests

La politique de tests est surtout axée autour de l'API REST. Il s'agit de s'assurer que les appels des tablettes vers le serveur fonctionnent bien, et de découvrir les problèmes éventuels bien avant la mise en production. Cela implique de reproduire plusieurs éléments « lourds » de l'architecture dans un environnement cloisonné sans avoir à subir de trop fortes contraintes de performances ou de configurations. En effet, pour tester l'API REST on doit disposer d'un client simulant une tablette, du serveur traitant les requêtes et une base de données de tests. Pour cela, Symfony et Doctrine fournissent des composants prêts à l'emploi pour les tests fonctionnels :

- Un client de tests apte à toutes les opérations HTTP et gérant les cookies ;
- Un SGBD résidant uniquement en mémoire.

Aucun fichier de configuration n'est nécessaire, tout se faisant par introspection (annotations) avec une durée de vie limitée à celle des tests. Ainsi, à chaque test une nouvelle base de données est générée en mémoire et remplie avec le minimum (une coopérative, un administrateur, une tablette identifiée). Le processus bien qu'assez lourd se déroule dans un laps de temps suffisamment court pour être exploitable. L'ensemble des tests (plus d'une centaine) prend moins de 5 minutes sur une machine « normalement » performante (Intel i7-4800QM).

5.5. Intégration continue

L'intégration continue est essentiellement la mise en place d'une solution d'automatisation d'outils de validation et de mesures. Le choix s'est porté sur Jenkins qui est le standard open-source dans le monde de l'intégration continue Java. Cette solution est suffisamment générique pour s'interfacer facilement avec les divers outils PHP.

Le rôle est Jenkins est celui d'orchestrer les outils de validations et de mesures, de fournir des rapports et une interface de contrôle. Les outils sont configurés indépendamment via le très classique système Ant (voir annexe 5 pour le fichier build.xml).

La première étape utilise l'interpréteur PHP lui-même : une option en ligne de commande permet de juste valider la syntaxe des sources. Ensuite, Jenkins appelle PHPUnit afin de valider l'ensemble des tests unitaires et d'intégration. A ce stade, on peut considérer que l'ensemble de projet est dans un état « sain », voire apte au déploiement.

Toutefois, les outils de mesures sont là pour garantir la qualité du code. L'outil PDepend (PDepend s.d.) va analyser l'ensemble des sources et donner certaines métriques :

- Complexité cyclomatique (CYCLO) : il s'agit de l'ensemble des chemins d'exécution possibles d'une portion de code. Chaque condition, boucle, méthode, exception incrémente ce nombre. Plus il est élevé, plus le code est complexe et devient difficile à maintenir ;
- Nombre de lignes de code (LOC), de classes (NOC), de méthodes (NOM) ;
- Nombre moyen de classes dérivées (ANDC) ;
- Profondeur moyenne de hiérarchie des classes (AHH) ;
- Etc.

L'analyse de ces métriques peut être assez subjective. Par exemple, on cherchera assez logiquement à réduire la complexité cyclomatique, mais chercher à réduire la profondeur moyenne de hiérarchie pour réduire le couplage peut parfois être un non-sens. Le résultat donné par PDepend est un ensemble de chiffres bruts et un graphique récapitulatif :

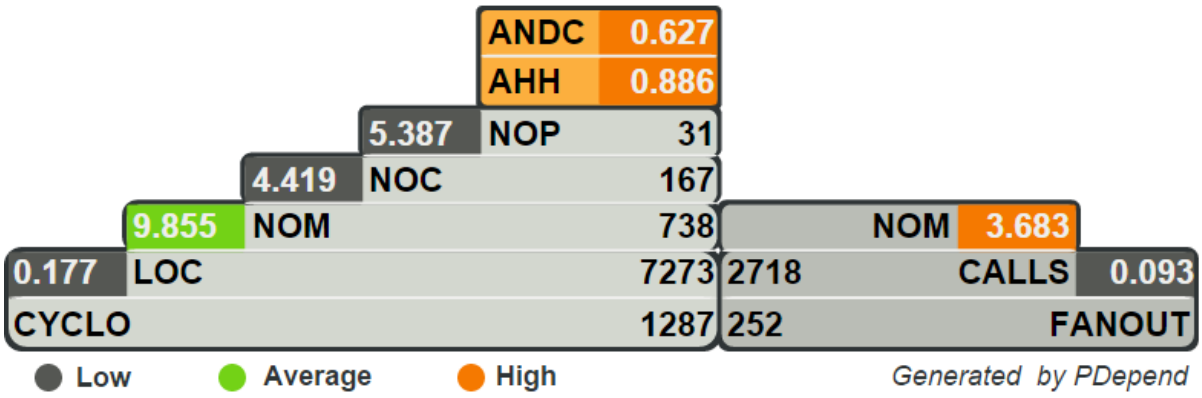


Figure 5-17 : pyramide de synthèse générée par PDepend

Autre outil très important : le détecteur de redondance. La base même d'un code « sain » est qu'il soit WET. Cet acronyme est un jeu de mot basé sur l'opposition à l'acronyme DRY qui signifie *Don't Repeat Yourself* (ne vous répétez pas). Un code WET est donc un code avec le moins de duplications possibles, sources de nombreuses erreurs, mauvaise conception et maintenance impossible.

5.6. Déploiement

PHP étant interprété, le gestionnaire de sources Git peut faire office de solution de déploiement. Cela est d'autant plus possible que Git offre la possibilité d'exécuter des commandes à l'interception de certains événements. Ainsi, si un serveur Git reçoit une mise à jour de sources sur une branche donnée, cela peut provoquer un ensemble d'actions nécessaires à une mise en production.

Les intérêts d'une telle mise en place sont multiples :

- Une fois installé, le processus de déploiement se résume à une commande : pousser les nouveaux sources vers le serveur distant ;
- Le déploiement peut être confié à plusieurs personnes en configurant simplement les droits d'accès en écriture dans le dépôt Git ;
- Si un problème arrive et qu'une régression est nécessaire, on peut tout aussi simplement revenir en arrière en exploitant l'historique des instantanés de Git ;
- On dispose de fait d'une journalisation implicite des déploiements, très pratique pour détecter les origines des erreurs.

5.7. Journalisation

Autant à des fins de développement que de production, il est indispensable d'avoir un suivi précis des flots d'exécution du système. Cela est d'autant plus vrai que le langage PHP est interprété et faiblement typé, laissant autant de facilités d'introspection que de risques d'erreurs non détectables statiquement.

La journalisation se fait via la librairie Monolog, extension possible de Symfony, qui respecte le standard PSR-3. L'instanciation d'un objet de journalisation se fait à la racine de chaque sous-système. L'objet est alors transmis aux objets métiers qui prennent la responsabilité de l'utiliser de façon suffisamment exhaustive. Cela se traduit par un emploi plus ou moins intensif, et du choix du niveau de journalisation : debug, info, avertissement, erreur, erreur critique, etc. Par ailleurs, la racine d'un sous-système peut intercepter les événements de journalisation afin qu'ils soient exploités de la manière appropriée. C'est notamment le cas de l'importation via l'administration : toutes les étapes sont consignées via l'objet transmis à l'importateur, et l'intercepteur stocke chaque ligne du journal en base de données pour permettre sa consultation ultérieure.

De même, l'importateur en ligne de commande intercepte les événements afin de les afficher en temps réel avec une couleur adaptée au niveau de journalisation de l'information (typiquement, du rouge pour une erreur).

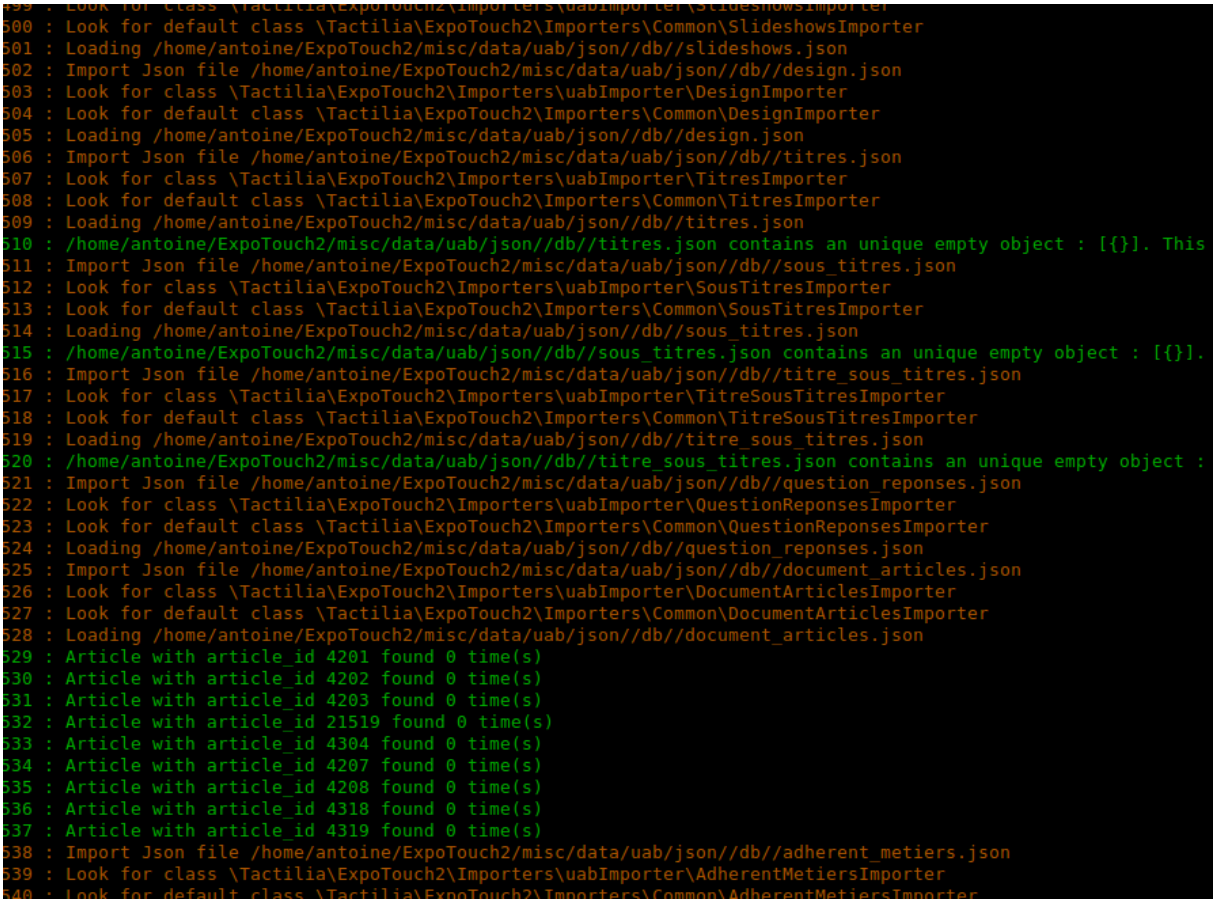


Figure 5-18 : journalisation et distinction des niveaux par couleurs

5.8.Gestion des importations

La transformation en licence de la solution implique de pouvoir gérer l'importation de plusieurs clients. Pour des raisons de performances on ne peut pas prendre le risque d'avoir un nombre inconnu d'importations simultanées. C'est donc une file d'attente qui est mise en place pour traiter les importations successivement. A ce titre, le système les gère en CRUD appuyé par REST : un processus d'importation est une ressource que l'on crée (démarrage) et que l'on peut détruire (annulation). La ressource se compose d'une heure de départ, d'une heure de fin et d'un journal de bord. Enfin, la ressource est consommée dès qu'il n'y en a aucune autre en attente.

Comme vu précédemment (voir 5.5) chaque étape d'un processus est journalisée de façon plus ou moins exhaustive

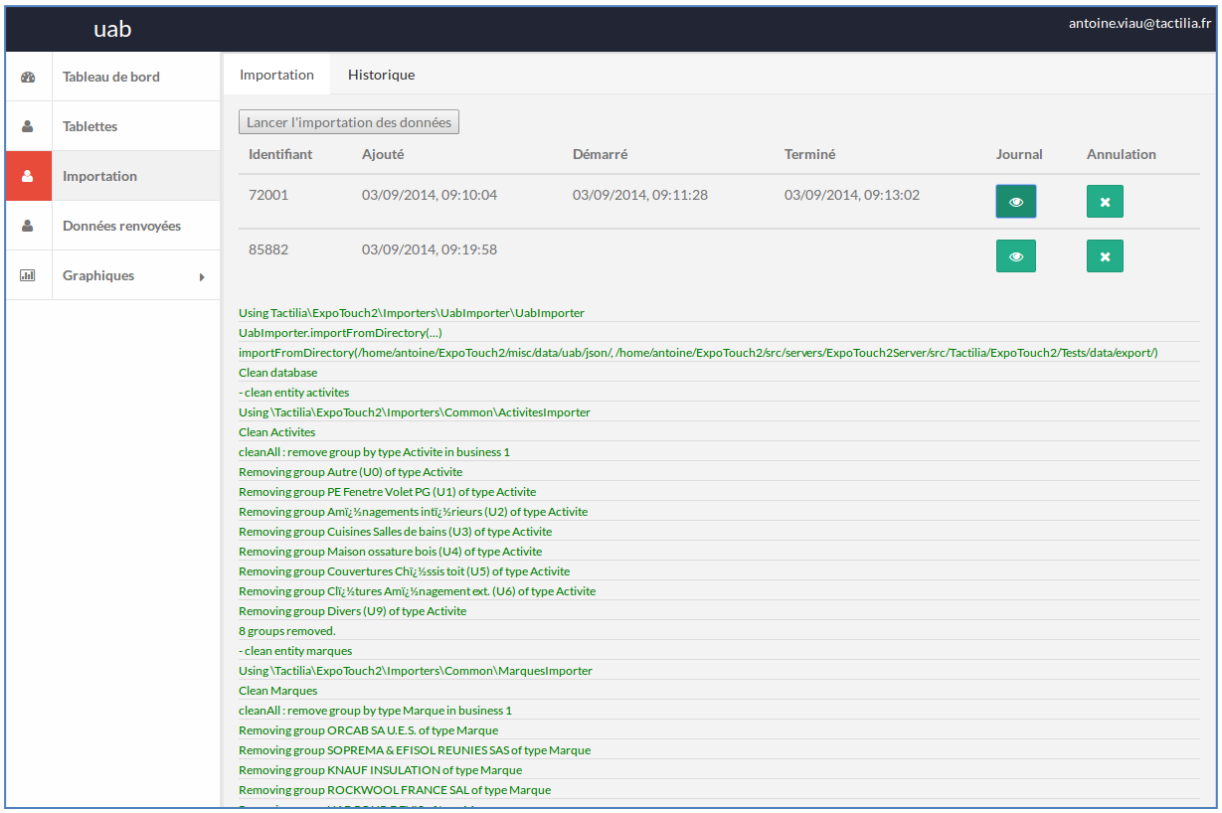


Figure 5-19 : importation et journalisation dans l'interface d'administration

6. Bilan

ExpoTouch version 2 va prochainement entrer en phase de recette auprès des clients historiques. Des accords commerciaux sont en cours pour permettre la diffusion de la solution auprès de nouvelles coopératives, et les premiers retours des prospects sont encourageants. Des réflexions sont en cours pour déterminer si la partie tablette doit d'abord suivre le même chemin que la partie serveur, à savoir une refonte complète.

6.1. Bilan des objectifs

6.1.1. Maintenance

L'évaluation de la maintenance logicielle se fait avant tout dans le temps. Toutefois, on peut déjà établir que la base de cet objectif est atteinte : les architectures globales et locales sont basées sur des standards et de la formalisation. Cela se traduit par le choix de canevas (Symfony, AngularJS), de surcouches (Doctrine, D3, Flatify) et de méthodes (REST). Le « code maison » est implémenté au sein de ces technologies et s'oblige à les respecter. En conséquence, la maintenance est avant tout une affaire d'apprentissage des technologies choisies, offrant à toute personne travaillant sur le projet un capital supplémentaire pour d'autres projets. Par ailleurs, l'intégration continue fournit des outils et métriques pour garantir une qualité logicielle aussi exemplaire que possible.

6.1.2. Performances

Malheureusement, il n'a pas été encore possible de faire des tests sur les machines de production des clients historiques. Toutefois, par l'emploi de la virtualisation, il a été possible de simuler les capacités de ces machines afin d'évaluer les performances d'ExpoTouch 2. Le résultat est très satisfaisant : le temps moyen d'importation de la coopérative la plus chargée en données a été divisé par 6 (d'une heure à 10 minutes). Ce type de métrique étant très remarquable, l'expérience utilisateur en est considérablement améliorée et l'effet commercial très positif.

6.1.3. Intégrité des données

La construction d'un modèle de données adapté aux spécifications fonctionnelles a corrigé l'opacité de la solution précédente basée sur Drupal. La masse de données brutes est considérablement diminuée, passant de plus de 500 Mo à moins de 30 ! De fait, les sauvegardes et restaurations sont beaucoup plus véloces. Cela a permis de maintenir le système précédent pour permettre un retour en arrière en cas de problème. Toutefois, cette

solution n'est pas satisfaisante car elle trouvera vite sa limite avec l'augmentation du nombre de clients. D'autre part, son monolithisme n'est pas du tout adapté à des contraintes de sécurité. Il reste donc à déterminer un système de sauvegarde et restauration sélectif.

6.1.4. Industrialisation

La recherche et l'analyse des choix technologiques ont permis de distinguer clairement ce vers quoi la société Tactilia va tendre en termes de développement. La principale orientation étant la volonté de concevoir des architectures fortement découplées en se basant sur des échanges REST et des applications Web à page unique (SPA). Cette approche va nous permettre de factoriser les logiques métiers autant pour les applications Web que pour celles tournant sur tablettes connectées. Par ailleurs, la mise en place de la chaîne de développement a été une bonne façon d'évaluer de nouvelles pratiques de travail. Il en ressort tout l'intérêt et l'importance de l'intégration continue et ses composants : contrôle des sources, tests, déploiement contrôlé, etc.

Il reste encore de nombreuses pistes à explorer, notamment en ce qui concerne l'industrialisation de la conception logicielle côté client HTML5/Javascript. AngularJS, D3 et Flatify permettent la mise en place d'architectures à des niveaux d'abstraction plus ou moins élevés, mais ils n'interviennent pas dans la chaîne de développement. Pour cela, il commence à apparaître des outils qui se regroupent dans un écosystème dédié, permettant d'automatiser certaines tâches : tests, compression, packaging, etc. Ils sont encore jeunes, mais présentent un avenir prometteur et seront étudiés pour une éventuelle intégration au sein du pôle développement de Tactilia.

6.2.Bilan personnel

D'un point de vue personnel, ce projet a été le premier travail où j'ai pu mettre en œuvre les méthodes et techniques apprises au CNAM. Cela s'est notamment déroulé via la formalisation des spécifications fonctionnelle, mais aussi – et surtout – par une approche industrielle de la conception logicielle. Citons en particulier l'emploi d'outils logiciels tels que les canevas, les patrons de conception et la formalisation UML.

Enfin, c'est aussi l'aboutissement de trois années passées au CNAM où après 15 ans d'expérience professionnelle en tant que développeur, j'ai dû me remettre en cause et accepter de revoir nombre de choses que je pensais connaître et maîtriser. J'en ressors avec une nouvelle approche de mon métier, de nouveaux bagages et l'envie de les exploiter…

Annexes

Annexe 1 : API REST

Quelques éléments de l'API REST.

Les codes de retour HTTP sont utilisés pour indiquer le résultat d'une opération. Il ne faut donc pas se baser uniquement sur le contenu retourné. Typiquement, on aura :

- 200 pour un succès ;
- 404 pour une ressource non trouvée (par exemple effacement d'un devis inexistant) ;
- 400 pour un format de données invalide (par exemple, contenu non JSON) ;
- 406 pour une valeur incorrecte (par exemple, date du 31 février) ;
- Etc. (voir http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html)

Action	Authentification tablette
URL	POST /api/tct/user/login
Contenu	{ « username » : « xxx », « password » : « xxx »}
Retour	Jeton de sécurité (identifiant de ressource)
Remarques	

Action	Déconnexion tablette
URL	DELETE /api/tct/user/login/{resourceId}
Contenu	Identifiant de ressource d'authentification
Retour	
Remarques	Validation par code HTTP 200

Action	Création d'un devis
URL	POST /api/tct/choice_form
Contenu	{ "civilite": "M.", "nom": "Dujardin", "adresse1": "Hollywood", "adresse2" : "", "code_postal": "75000", "ville": "Paris", "tel": "555555555", "portable": "666666666", "email": "dujardin@oscar.com", "num_adherent": "141", "nom_adherent": "Hazanavicius", "commentaires": "Tip top!", "type_chantier": "Neuf", "constructeur_code": "XXWWYY", "conseiller_code": "ERPPOE", "creation_cf": "1", "date": "27022012", "heure": "130234", "lock": "0", "date_lock": "" }
Retour	Identifiant du devis : { « id » : « xxx » }
Remarques	Pour les booléens, mettre "0" ou "1"

Action	Suppression d’un devis
URL	DELETE /api/tct/choice_form/{id}
Contenu	
Retour	Identifiant du devis effectivement effacé.
Remarques	

Action	Modifier un devis
URL	PUT /api/tct/choice_form/{id}
Contenu	Voir Création d’un devis
Retour	
Remarques	

Action	Récupérer l’index des devis
URL	GET /api/tct/choice_form
Contenu	
Retour	Un tableau JSON des URL des devis.
Remarques	Le résultat est paginé, par défaut à 20 URL par pages. Le nombre d'URL par page ainsi que le numéro de la page à retourner peuvent être donnés par des paramètres ajoutés à l'URL REST. Exemple : /api/tct/choice_form?page=1&pagesize=2 « page=1 » correspond à la livraison de la page n°1, tandis que « pagesize=2 » indique que chaque page devra afficher au plus deux URL. Note : « pagesize » est limité à 200.

Action	Récupérer un devis
URL	GET /api/tct/choice_form/{id}
Contenu	
Retour	Même format que « Création d'un devis »
Remarques	

Action	Création d'une visite
URL	POST /api/tct/date_visit

Contenu	{
	"date_start": "16042012165050",
	"date_end": "16042012174029",
	"id": "ipad1",
	"code_adherent" : "AD001" //chaîne vide si visiteur venu par lui-même
	}

Retour	date_start, date_end : le format des dates d'arrivée et de départ des visiteurs sont des chaînes de 14 caractères de la forme « JJMMAAAAHHMMSS ». id : identifiant de la tablette du conseiller. code_adherent : le code de l'adhérent ayant envoyé le visiteur en salle d'exposition, le cas échéant.
Remarques	

Action	Index des visites
URL	GET /api/tct/date_visit
Contenu	
Retour	Retourne un tableau d'URL vers les dates de visites créées. Le résultat est paginé, par défaut à 20 URL par pages. Le nombre d'URL par page ainsi que le numéro de la page à retourner peuvent être données par des paramètres ajoutés à l'URL REST.
Remarques	

Annexe 2 : Guide de style de codage Symfony

Ci-dessous, l'exemple de référence du style de codage de Symfony. On trouvera les détails à cette adresse : http://symfony.com/fr/doc/current/contributing/code/standards.html

```php
<?php

/*
 * This file is part of the Symfony package.
 *
 * (c) Fabien Potencier <fabien@symfony.com>
 *
 * For the full copyright and license information, please view the LICENSE
 * file that was distributed with this source code.
 */

namespace Acme;

class FooBar
{
    const SOME_CONST = 42;

    private $fooBar;

    /**
     * @param string $dummy Some argument description
     */
    public function __construct($dummy)
    {
        $this->fooBar = $this->transformText($dummy);
    }

    /**
     * @param string $dummy Some argument description
     * @param array  $options
     *
     * @return string|null Transformed input
     */
    private function transformText($dummy, array $options = array())
    {
        $mergedOptions = array_merge($options, array(
            'some_default' => 'values',
        ));

        if (true === $dummy) {
            return;
        }

        if ('string' === $dummy) {
            if ('values' === $mergedOptions['some_default']) {
                $dummy = substr($dummy, 0, 5);
            } else {
                $dummy = ucwords($dummy);
            }
        }

        return $dummy;
    }
}
```

Annexe 3 : Code D3 commenté

```
/*
 * Ce module affiche le nombre de bons de choix par conseiller sous forme d'un pie-chart (camembert) troué.
 * Il est animé au rafraîchissement des données.
 */
angular.module("et2").factory("nbProposalsPerInterlocutorGfxService", ["restClientService",
function(restClientService) {


        var _startDate = null;
        var _endDate = null;
        var _svgContainer;
        var pie;
        var arc;
        var outerArc;
        var dataKey;
        var arcColor;
        var width = 660;
        var height = 350;
        var radius = Math.min(width, height) / 2;
        var colorBase = ["#98abc5", "#8a89a6", "#7b6888", "#6b486b", "#a05d56", "#d0743c", "#ff8c00"];
        var colors = [];


        /*
         * Création du graph : création et insertion des élement DOM/SVG,
         * puis des diverses fonctions et fonctions-objets qui seront utilisés
         * lors du rafraîchissement des données.
         */
        var _createGraph = function() {
            /*
             * Création du SVG et des groupes pour les arcs, les textes et les lignes.
             */
            _svgContainer = d3.select("#perInterlocutor")
                .append("svg")
                .attr("width", width)
                .attr("height", height)
                .append("g");
            _svgContainer.append("g").attr("id", "slices");
            _svgContainer.append("g").attr("class", "labels");
            _svgContainer.append("g").attr("class", "lines");


            /*
             * d3.layout.pie()
             * https://github.com/mbostock/d3/wiki/Pie-Layout
             * Constructs a new pie function with the default value accessor (number),
             * sort comparator (descending value), start angle (0) and end angle (2π).
             * The returned layout object is both an object and a function. That is:
             * you can call the layout like any other function, and the layout has additional
             * methods that change its behavior. Like other classes in D3, layouts follow the
             * method chaining pattern where setter methods return the layout itself, allowing
             * multiple setters to be invoked in a concise statement.
             *
             * En clair et en français : nous obtenons une fonction-objet. Les méthodes de l'objet
```

```
             * permettent de "calibrer" le comportement de la fonction lorsqu'elle sera appelée.
             * La fonction est appelée avec un tableau de données :
             * pie([10,20,30])
             * Elle retournera un tableau d'objets du type {startAngle:xxx, endAngle: yyy, etc.}
             * qui sera associé au DOM via data(...).
             * Par la suite, lors du parcours des sections (enter() pour la création) chaque objet est
             * transmis (ici dans attrTween()).
             */
            pie = d3.layout.pie()
                .sort(null)
                .value(function(d) {
                    return d.nbProposals;
                });

            /*
             *  d3.svg.arc()
             *  https://github.com/mbostock/d3/wiki/SVG-Shapes#arc
             *  Constructs a new arc generator with the default innerRadius-, outerRadius-, startAngle-
             *  and endAngle-accessor functions (that assume the input data is an object with named
             *  attributes matching the accessors)
             *
             *  Encore une fonction-objet.
             *  Son appel (arc(x)) retournera une string directement exploitable par SVG
             *  (attribut d="" de <path>)
             *  Ses méthodes permettent de définir la forme de l'arc (épaisseur, rayon, etc.)
             */
            arc = d3.svg.arc()
                .outerRadius(radius * 0.8)
                .innerRadius(radius * 0.4);

            /*
             * Même topo.
             * outerArc est utilisé pour les textes et les lignes.
             */
            outerArc = d3.svg.arc()
                .innerRadius(radius * 0.9)
                .outerRadius(radius * 0.9);

            /*
             * Centrage bête et méchant du pie.
             */
            _svgContainer.attr("transform", "translate(" + width / 2 + "," + height / 2 + ")");

            /*
             * Nous avons donc des données qui sont automatiquement converties en pourcentage (fonction-
objet pie)
             * eux-même représentés par des arcs (fonction-objet arc).
             * Ces données vont changer dans le temps : des nouvelles valeurs vont êtres ajoutées, d'autres
vont être
             * retirées et encore d'autres vont évoluer.
             * Exemple :
             *  - A t=0 nous avons Pierre = 10, Paul = 20, Jacques = 30.
             *  - A t=1 nous avons Marie = 40, Pierre = 15, Paul = 25
```

```
             * Comment faire pour que D3 fasse la liaison entre Pierre et 15 sachant que l'indexation
ordinale simple
             * ne fonctionne pas (à t=0 Pierre avec l'index 0 ; à t=1 son index est 2) ?
             * Tout simplement en fournissant une fonction d'indexation : elle prendra en entrée une donnée
(un objet)
             * et retournera une clef pour cet objet.
             * La fonction ci-dessous va donc choisir le nom de l'interlocutor comme clef. Elle sera donnée
en second
             * paramètre de selection.data().
             * On aurait pu prendre autre chose, comme un hash md5 ou un autre attribut fixe dans les
données.
             * Soit dit-en passant, l'utilisation du nom risque de poser problème si nous avons deux
intercutors
             * avec le même nom.
             * TODO : comprendre pourquoi d.data.interlocutorName et non pas d.interlocutorName
             */
            dataKey = function(d) {
                var key = d.data.interlocutorName;
                return key;
            };

            /*
             * On veut à présent choisir une couleur pour chaque arc du pie-chart.
             * Nous allons donc donner une fonction qui retournera une couleur en fonction
             * de la donnée passée en argument.
             * Petite magouille pour les couleurs :
             * on prend une palette de c couleurs, on la duplique n fois
             * dans un tableau, puis on mélange le tout.
             */
            for (var n = 0; n < 10; n++) {
                for (var i = 0; i < colorBase.length; i++) {
                    colors.push(colorBase[i]);
                }
            }
            for (var j, x, i = colors.length; i; j = Math.floor(Math.random() * i), x = colors[--i],
colors[i] = colors[j], colors[j] = x)
                ;
            arcColor = function(d) {
                var index = Math.floor(Math.random() * colors.length);
                return colors[index];
            };
        };

        var _refreshGraph = function(data) {
            /*
             * Les arcs.
             */
            /*
             * On sélectionne donc les éléments de classe .slice depuis le groupe (g) #slices
             */
            var slices = _svgContainer.select("#slices").selectAll(".slice");
            /*
             * Puis on y associe les donnée.
             * Rappel : pie() prend les données "brutes" en entrée (data), les convertit pour être
manipulables
             * par la fonction arc() et les retourne ainsi converties dans un tableau :
```

```
             * [ {startAngle: xx, endAngle: yy}, etc. ]
             * Chaque objet de ce tableau sera associé à un élément .slice du DOM.
             * Pour savoir à quel élément l'associer, D3 utilise la fonction dataKey.
             * A t=0 il n'y aucun élément, D3 met l'objet dans le DOM avec sa clef.
             * A t=1 les éléments ont changé (modif, ajout, suppression), et en cas de modif la clef permet
de savoir
             * quel élément du DOM doit évoluer.
             */
            var join = slices.data(pie(data), dataKey);
            /*
             * La section enter() retourne les données pour lesquelles il n'y a aucun élément du DOM
associé.
             * Elles se présentent sous la forme d'une sélection au sens D3 : un tableau de tableaux avec
des méthodes
             * (append, insert, call, etc.). C'est un "tableau-objet".
             */
            var e = join.enter();
            /*
             * append() retourne une sélection D3 (tableau-objet) qui contient des méthodes permettant de
jouer
             * avec le DOM (attr(), style(), etc.).
             */
            var a = e.append("path");
            /*
             * attr() est donc une méthode d'un tableau-objet : elle va parcourir son tableau et pour
chaque élément
             * elle associe une valeur à un attribut.
             * Même chose pour style().
             * On voit ci-dessous que le parcours permet l'appel d'une fonction pour chaque élément. La
fonction appelé
             * reçoit en argument la donnée, et elle retourne la valeur à mettre dans le style.
             */
            a.attr("class", "slice")
                .style("fill", arcColor);

            join.transition()
                .duration(1000)
                .attrTween("d", function(d) {
                    this._current = this._current || d;
                    var interpolate = d3.interpolate(this._current, d);
                    this._current = interpolate(0);
                    return function(t) {
                        return arc(interpolate(t));
                    };
                });
            join.exit().remove();

            /*
             * Les textes.
             */
            var text = _svgContainer.select(".labels").selectAll("text")
                .data(pie(data), dataKey);

            text.enter()
                .append("text")
```

```
        .attr("dy", ".35em")
        .text(dataKey);

    function midAngle(d) {
        return d.startAngle + (d.endAngle - d.startAngle) / 2;
    }

    text.transition().duration(1000)
        .attrTween("transform", function(d) {
            this._current = this._current || d;
            var interpolate = d3.interpolate(this._current, d);
            this._current = interpolate(0);
            return function(t) {
                var d2 = interpolate(t);
                var pos = outerArc.centroid(d2);
                pos[0] = radius * (midAngle(d2) < Math.PI ? 1 : -1);
                return "translate(" + pos + ")";
            };
        })
        .styleTween("text-anchor", function(d) {
            this._current = this._current || d;
            var interpolate = d3.interpolate(this._current, d);
            this._current = interpolate(0);
            return function(t) {
                var d2 = interpolate(t);
                return midAngle(d2) < Math.PI ? "start" : "end";
            };
        });

    text.exit().remove();

    /*
     * Les lignes.
     */
    var polyline = _svgContainer.select(".lines").selectAll("polyline")
        .data(pie(data), dataKey);

    polyline.enter()
        .append("polyline");

    polyline.transition().duration(1000)
        .attrTween("points", function(d) {
            this._current = this._current || d;
            var interpolate = d3.interpolate(this._current, d);
            this._current = interpolate(0);
            return function(t) {
                var d2 = interpolate(t);
                var pos = outerArc.centroid(d2);
                pos[0] = radius * 0.95 * (midAngle(d2) < Math.PI ? 1 : -1);
                return [arc.centroid(d2), outerArc.centroid(d2), pos];
            };
        });
```

```
            polyline.exit()
                .remove();



        };

        var _getDataAndRefreshGraph = function(startDate, endDate) {
            return restClientService.get('/api/tct/stats/proposals/nb/interlocutor/' + startDate + '/' +
endDate)
                .then(function(data) {
                    var graphData = [];
                    // Pour version démo !
                    var fakeNames = ["Marie", "Paul", "Julie", "Jacques", "Nathalie", "François", "Pierre",
"Agathe", "Nicolas", "Jeanne", "Jacques", "Charlotte", "Vincent", "Aurore", "Antoine", "Claudine"];
                    var toFake = [];
                    var totalProposals = 0;
                    for (var i = 0; i < data.length; i++) {
                        graphData[i] = {};
                        var interlocutorName = data[i].interlocutorName;
                        if ( toFake[interlocutorName] === undefined ) {
                            toFake[interlocutorName] = fakeNames.shift();
                        }
                        graphData[i].interlocutorName = toFake[interlocutorName];
                        graphData[i].nbProposals = +data[i].nbProposals;
                        totalProposals += +data[i].nbProposals;
                    }
                    for (var i = 0; i < graphData.length; i++) {
                        graphData[i].percent = Math.round(graphData[i].nbProposals / totalProposals *
10000.0) / 100;
                    }
                    _refreshGraph(graphData);
                },
                    function(data) {
                        alert("fail :" + data);
                    });
        };

        var _jsDateToExpoTouch2Date = function(jsDate) {
            var day = jsDate.getDate() > 9 ? '' + jsDate.getDate() : '0' + jsDate.getDate();
            var month = jsDate.getMonth() + 1;
            month = month > 9 ? '' + month : '0' + month;
            var expoTouch2Date = day + month + jsDate.getFullYear();
            return expoTouch2Date;
        };

        /**
         * Ci-dessous l'API publique du service.
         */
        return {
            getStartDate: function() {
                return _startDate;
            }
            ,
            getEndDate: function() {
```

```
                return _endDate;
            }
            ,
            createGraph: function() {
                return _createGraph();
            }
            ,
            refreshGraph: function(startDate, endDate) {
                _startDate = startDate;
                _endDate = endDate;
                return _getDataAndRefreshGraph(_jsDateToExpoTouch2Date(startDate),
_jsDateToExpoTouch2Date(endDate));
            }
        };
    }]);
```

Annexe 4 : Contrôleur CRUD standard sous AngularJS (tabletController.js)

```
angular.module('et2').controller('usersController', ['$scope', 'userService', 'taskService',
    function usersController($scope, userService, taskService) {

        var _refreshList = function() {
            return taskService.executePromise("Chargement...", userService.getUsers())
                .then(function(users) {
                    $scope.users = users;
                    for (var i = 0; i < $scope.users.length; i++) {
                        $scope.users[i].isEditing = false;
                    }
                });

        };

        $scope.createUser = function() {
            taskService.executePromise('Sauvegarde...', userService.createUser($scope.userLogin,
$scope.userPassword))
                .then(_refreshList);
        };

        $scope.editUser = function(user) {
            for (var i = 0; i < $scope.users.length; i++) {
                $scope.users[i].isEditing = false;
            }
            user.isEditing = true;
        };

        $scope.cancelEdit = function(user) {
            user.isEditing = false;
        };

        $scope.updateUser = function(user) {
            taskService.executePromise('Mise à jour...', userService.updateUser(user.id, user.login,
user.password), 'Sauvegarde effectuée')
                .then(_refreshList);
        };

        $scope.deleteUser = function(user) {
            if (confirm("Etes-vous certain de vouloir effacer l'utilisateur " + user.login + " ? ")) {
                taskService.executePromise('Effacement...', userService.deleteUser(user.id))
                    .then(_refreshList);
            }
        };

        _refreshList();

    }]);
```

Annexe 5 : Fichier de configuration de l'intégration continue (build.xml)

```
<?xml version="1.0" encoding="UTF-8"?>

<project name="expotouch" default="build">
 <target name="build"
   depends="prepare,lint,phploc,pdepend,phpcpd,phpunit"/>

 <target name="build-parallel"
   depends="prepare,lint,tools-parallel,phpunit"/>

 <target name="tools-parallel" description="Run tools in parallel">
  <parallel threadCount="2">
   <sequential>
    <antcall target="pdepend"/>
    <antcall target="phpmd-ci"/>
   </sequential>
   <antcall target="phpcpd"/>
   <antcall target="phpcs-ci"/>
   <antcall target="phploc"/>
   <antcall target="phpdox"/>
  </parallel>
 </target>

 <target name="clean" description="Cleanup build artifacts">
  <delete dir="${basedir}/build/api"/>
  <delete dir="${basedir}/build/coverage"/>
  <delete dir="${basedir}/build/logs"/>
  <delete dir="${basedir}/build/pdepend"/>
 </target>

 <target name="prepare" depends="clean" description="Prepare for build">
  <mkdir dir="${basedir}/build/api"/>
  <mkdir dir="${basedir}/build/coverage"/>
  <mkdir dir="${basedir}/build/logs"/>
  <mkdir dir="${basedir}/build/pdepend"/>
  <mkdir dir="${basedir}/build/phpdox"/>
 </target>

 <target name="lint" description="Perform syntax check of sourcecode files">
  <apply executable="php" failonerror="true">
   <arg value="-l" />

   <fileset dir="${basedir}/src">
    <include name="**/*.php" />
    <modified />
   </fileset>

  </apply>
 </target>

 <target name="phploc" description="Measure project size using PHPLOC">
  <exec executable="phploc">
```

```
    <arg value="--log-csv" />
    <arg value="${basedir}/build/logs/phploc.csv" />
    <arg path="${basedir}/src" />
   </exec>
  </target>

  <target name="pdepend" description="Calculate software metrics using PHP_Depend">
   <exec executable="pdepend">
    <arg value="--jdepend-xml=${basedir}/build/logs/jdepend.xml" />
    <arg value="--jdepend-chart=${basedir}/build/pdepend/dependencies.svg" />
    <arg value="--overview-pyramid=${basedir}/build/pdepend/overview-pyramid.svg" />
    <arg path="${basedir}/src" />
   </exec>
  </target>

  <target name="phpmd"
          description="Perform project mess detection using PHPMD and print human readable output. Intended
for usage on the command line before committing.">
   <exec executable="phpmd">
    <arg path="${basedir}/src" />
    <arg value="text" />
    <arg value="${basedir}/build/phpmd.xml" />
   </exec>
  </target>

  <target name="phpmd-ci" description="Perform project mess detection using PHPMD creating a log file for
the continuous integration server">
   <exec executable="phpmd">
    <arg path="${basedir}/src" />
    <arg value="xml" />
    <arg value="${basedir}/build/phpmd.xml" />
    <arg value="--reportfile" />
    <arg value="${basedir}/build/logs/pmd.xml" />
   </exec>
  </target>

  <target name="phpcs"
          description="Find coding standard violations using PHP_CodeSniffer and print human readable
output. Intended for usage on the command line before committing.">
   <exec executable="phpcs">
    <arg value="--standard=${basedir}/build/phpcs.xml" />
    <arg path="${basedir}/src" />
   </exec>
  </target>

  <target name="phpcs-ci" description="Find coding standard violations using PHP_CodeSniffer creating a log
file for the continuous integration server">
   <exec executable="phpcs" output="/dev/null">
    <arg value="--report=checkstyle" />
    <arg value="--report-file=${basedir}/build/logs/checkstyle.xml" />
    <arg value="--standard=${basedir}/build/phpcs.xml" />
    <arg path="${basedir}/src" />
   </exec>
  </target>
```

```
 <target name="phpcpd" description="Find duplicate code using PHPCPD">
  <exec executable="phpcpd">
   <arg value="--log-pmd" />
   <arg value="${basedir}/build/logs/pmd-cpd.xml" />
   <arg path="${basedir}/src" />
  </exec>
 </target>

 <target name="phpdox" description="Generate API documentation using phpDox">
  <exec executable="phpdox"/>
 </target>

 <target name="phpunit" description="Run unit tests with PHPUnit">
  <exec executable="phpunit" failonerror="true">
  <arg value="-c" />
  <arg value="${basedir}/app/phpunit.xml.dist" />
</exec>
 </target>
</project>
```

Bibliographie

Ouvrages

Bacco, Alexandre. *Développez votre site web avec le framework Symfony2.* Simple IT, 2013.

CNAM. «Cours de l'unité d'enseignement GLG204.»

Gamma, Erich, Richard Helm, Ralph Johnson, et John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley.

Martin, Robert C. *Coder Proprement.* Traduit par Hervé Soulard. Pearson, 2009.

Meyer, Bertrand. *Object-Oriented Software Construction.* Prentice Hall, 1988.

Richardson, Leonard. *Restful Web Services.* O'Reilly, 2007.

Sites Web

«Code Coverage Analysis.» *PHPUnit.* https://phpunit.de/manual/current/en/code-coverage-analysis.html (accès le 08 01, 2014).

Doctrine Project. http://www.doctrine-project.org/projects/orm.html (accès le 08 01, 2014).

«Draft Specification for ES.next (Ecma-262 Edition 6).» *EcmaScript.* http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts (accès le 08 01, 2014).

Drupal. https://www.drupal.org/ (accès le 08 01, 2014).

«Drupal Database Schema.» *Drupal.org.* https://www.drupal.org/node/1785994 (accès le 08 01, 2014).

Fielding, Roy Thomas. «Architectural Styles and the Design of Network-based Software Architectures.» Dissertation for the degree of Doctor of philosophy in Information and Computer Science, University of California, Irvine, 2000, 76-106.

«Flatify.» *GitHub.* https://github.com/iarouse/dist-flatify (accès le 08 01, 2014).

Fowler, Martin. «Continuous Integration.» *Martin Fowler.* http://www.martinfowler.com/articles/continuousIntegration.html (accès le 08 01, 2014).

Git. http://git-scm.com/ (accès le 08 01, 2014).

«kriskowal/q.» *GitHub.* https://github.com/kriskowal/q (accès le 08 01, 2014).

«mbostock/d3.» *GitHub.* https://github.com/mbostock/d3/wiki/Gallery (accès le 08 01, 2014).

Minar, Igor. *AngularJS Google Plus page.* https://plus.google.com/+AngularJS/posts/aZNVhj355G2 (accès le 08 01, 2014).

«PDepend.» *PDepend.* http://www.pdepend.org (accès le 08 01, 2014).

PHP Framework Interop Group. http://www.php-fig.org/ (accès le 8 1, 2014).

PHPUnit. https://phpunit.de/ (accès le 08 01, 2014).

«SCALABLE VECTOR GRAPHICS (SVG).» *W3C.* http://www.w3.org/Graphics/SVG/ (accès le 08 01, 2014).

Symfony. http://symfony.com/ (accès le 08 01, 2014).

«Tags.» *StackOverflow.* http://stackoverflow.com/tags (accès le 08 01, 2014).

«XMLHttpRequest Level 1.» http://www.w3.org/TR/XMLHttpRequest/ (accès le 08 01, 2014).

Liste des figures

Liste des tableaux

Résumé :

Tactilia est une société de services informatiques spécialisée dans les équipements mobiles et tactiles. Elle commercialise la solution ExpoTouch depuis fin 2012 à destination des coopératives d'artisans disposant de salles d'exposition. ExpoTouch permet à des conseillers équipées de tablettes tactiles de proposer les produits et services des adhérents des coopératives, de collecter des informations clients/prospects et de réaliser des devis. Construite sur mesure pour trois coopératives et victime son succès, la solution a atteint un palier en termes de performances et d'usages. Il a donc été décidé de lancer la version 2 d'ExpoTouch sous une forme unique vendue en licence d'utilisation. C'est aussi l'occasion pour Tactilia de déterminer un nouveau périmètre de technologies qui seront employées sur de futurs projets. Ce mémoire présente l'analyse, les recherches et évaluations de technologies, la conception et la réalisation de la version 2 d'ExpoTouch, côté serveur. Les objectifs sont : une plateforme unique, de meilleures performances, des fonctionnalités supplémentaires, une meilleure gestion des données.

Mots-clés : tablette, tactile, PHP, Symfony, ORM, Doctrine, REST, JavaScript, AngularJS, Data-Driven Documents (D3)

Abstract :

Tactilia is a software company dedicated to touch devices. One of its products is a solution called ExpoTouch, launched late 2012, used by three artisan cooperatives with showrooms. ExpoTouch allows counselors equipped with tablets to show the products and services of cooperatives members, collect customer / prospect information and produce quotes. Victim of its own success, the solution has reached a plateau in terms of performance and uses. It was therefore decided to launch version 2 as a licensed product. This is also an opportunity for Tactilia determine which technologies will be used on future projects. This paper presents the analysis, research and technology assessment, design and implementation of ExpoTouch version 2 (server side). The objectives are: a single platform, better performance, more features, better data management.

Keywords : tablet, tactile, PHP, Symfony, ORM, Doctrine, REST, JavaScript, AngularJS, Data-Driven Documents (D3)