# PROJ
# M1-FA-MIAGE

## *CRM 365*
## *Analyse fonctionnelle*

- *Prototype*
- *Sprint 1*

*Groupe* :
**Antoine Craske**
**Tarik Djebien**
**Eric Sitraka Rakotobe**
**Rudy Stienne**

*Février* 2012


## Abstract

Ce document a pour objectif de décrire fonctionnellement l'application CRM 365 afin d'en permettre le développement.

Le mode de gestion de projet choisi est orienté agile afin de construire l'application itérativement. Cela nous permettra d'identifier au plus tôt les contraintes et de réduire le risque d'effet tunnel. Les livrables au sein de l'équipe projet seront définis toutes les deux semaines. Ce document présente donc l'analyse des fonctionnalités pour le prototype et le premier sprint.

Chaque sprint fera l'objet d'une analyse fonctionnelle.

L'architecture et les fonctionnalités sont présentées dans un premier temps afin de fournir une vision d'ensemble de l'application. Le planning présente ensuite les contraintes en termes de livrables et de délais. Les trois couches fonctionnelles de l'application sont ensuite détaillées : persistance, service et présentation. Les risques identifiés sont citées dans un dernier temps.

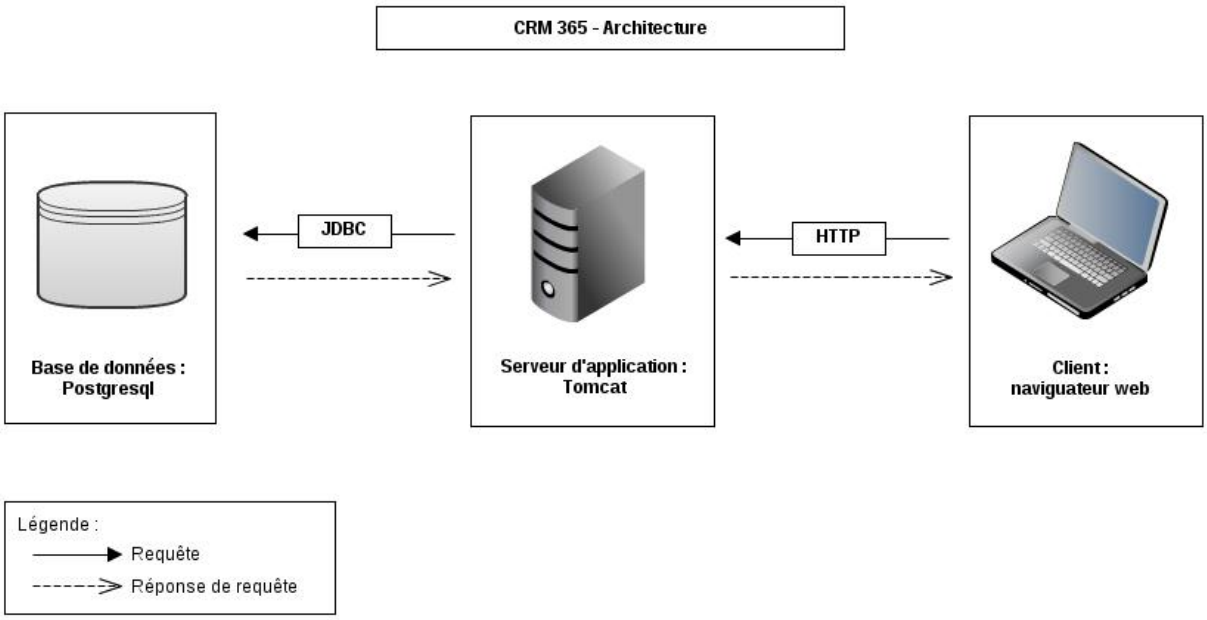
Les schémas ont étés réalisés suivant le formalisme UML afin de simplifier la compréhension à travers la standardisation des documents.

# Tables des matières

# Architecture

## Architecture logicielle

CRM 365 est une application architecturée en 3-tiers. Le schéma ci-dessous présente son architecture.




L'application est déployable et accessible sur un serveur d'application web. Le client accède à l'application via un navigateur web installé sur son poste par le protocole HTTP. Les requêtes sont traitées par le serveur d'application qui route les requêtes vers notre application et nos pages en fonction des URLs demandées. Si besoin, l'application accède à la base de données par le protocole JDBC.
Les technologies indiquées sont celles utilisées durant le développement de l'application. Les contraintes d'intégration sont décrites dans le chapitre suivant.

## Architecture technique

Notre objectif est de rendre CRM 365 complètement portable. Ceci afin d'en faciliter l'intégration dans les systèmes d'informations de PME qui sont tous différents. Pour cela nous devons le rendre portable sous trois aspects : stockage, déploiement, accès.

Concernant le stockage des informations dans la base de données, nous avons choisi d'utiliser le fournisseur Postgresql pour le développement. Nous respecterons les standards de la norme SQL3 pour le périmètre qui lui est défini. Nous n'utiliserons pas de mécanismes propres à un vendeur (i.e. procédures stockées) pour la bonne raison qu'ils nous rendraient dépendant à ce dernier. Ils rendraient indirectement dépendants nos clients qui n'ont pas forcément choisi ce fournisseur pour leur standard de base de données. Les développements devront donc respecter la norme SQL3 afin d'assurer la compatibilité avec les différents modèles.

CRM 365 est une application à déployer sur un serveur d'application web. L'installation se réalisera par installation d'un paquet de format EAR contenant les différents modules WAR et JAR de l'application. Notre contrainte sera de vérifier que le fichier descripteur de déploiement sera portable sur la majorité des serveurs d'application disponibles sur le marché (eg: Glassfish, Tomcat, Jboss, etc).
L'application sera codée en langage Java. La couche présentation utilisera les mécanismes de servlet propres à Java combinés avec le HTML, CSS dans des Jsp. Le language Java est exécuté sur une JVM (machine virtuelle) qui peut tourner sur n'importe quel système d'exploitation.

L'accès à l'application sera réalisé par un navigateur web. L'application doit être portable sur les navigateurs suivants : Mozilla Firefox (version 3.5 et ultérieure), Google Chrome, Internet Explorer (Version 8 et ultérieure). Ces derniers sont les trois navigateurs standards aujourd'hui déployés en entreprise sur lesquels notre application doit fonctionner correctement.

Notre objectif est donc bien de rendre indépendant du matériel l'utilisation de notre application en supportant une variété de systèmes d'exploitation, de moteur de base de données, de serveurs d'applications et de navigateurs.

## Frameworks de développement

L'utilisation de frameworks a été décidée en fonction des connaissances de l'équipe et de nos besoins en productivité. Chaque framework a été choisi pour une utilisation concrète et non par habitude. Les frameworks définis sur le prototype et le premier sprint sont les suivants : Apache Maven, Spring (JDBC, IoC, Security, MVC), JUnit, Mockito, Selenium, Hudson.

Apache Maven orchestrera de manière transversale le cycle de vie et les dépendances de l'application. Ceci nous permettra d'automatiser l'intégration (compilation, tests, packaging, déploiement, audit de code) et de rendre plus flexible l'installation de l'application. La gestion des dépendances pourra être réalisée via un repository local à l'entreprise afin d'éviter de connecter ses serveurs directement à internet pour récupérer les librairies requises. Maven nous permettra par l'automatisation de réduire les risques d'erreurs humaines, de gagner du temps et de standardiser le cycle de vie de l'application.

Différents modules de Spring seront utilisés afin d'augmenter la qualité du code et la productivité (réutilisation modules Spring). La gestion des beans Java par Spring IoC (Inversion of Control) sera réalisée par annotations dans les classes Java. La gestion des beans par XML n'a pas été retenue car parfois redondante et lourde à maintenir. Spring JDBC sera utilisé pour accéder à la base de données par les classes DAO (Data Access Object). L'architecture de l'application respectera le pattern MVC (Model View Controller) afin d'augmenter la qualité de l'architecture et le respect du SRP (Single Responsibility Principle). La gestion des droits d'accès sera géré par les controlleurs via Spring Security.

Le développement est orienté en mode agile, nous avons donc choisi de suivre les principes de l'intégration continue afin de garantir l'évolutivité de notre application. Pour cela, nous avons besoin d'outils pour automatiser le test et l'audit de code. Les tests unitaires qui accompagneront le développement en TDD (Test Driven Development) seront réalisés avec le framework JUnit 4. Afin d'augmenter la qualité et la pertinence de nos tests, nous utiliseront Mockito afin de réaliser les tests en isolation. Cela nous permettra de cibler efficacement l'origine d'un bug. Selenium nous permettra d'effectuer les tests d'intégrations (i.e. est-ce les briques de mon application fonctionnent correctement ensemble) par des tests réalisés directement sur les trois navigateurs web définis.

Le code de l'application est intégré dans un gestionnaire de version SVN. Afin de centraliser l'intégration continue, nous avons décidé d'utiliser Hudson. Cet outil nous permettra de réaliser le cycle de vie complet de l'application à chaque commit sur le SVN. Nous aurons ainsi une validation de compilation, test et déploiement sur une machine distante autre que des machines locales. L'installation d'outil d'audit de code (outils encore non définis aujourd'hui) pourra se faire par configuration uniquement sur Hudson. Nous automatiseront ainsi le processus et limiteront les besoins de configuration locaux qui peuvent faire perdre du temps à l'équipe.
Ces frameworks définis pour le prototype et le premier sprint nous permettront d'être rapidement efficaces et d'assurer l'évolutivité de notre application en conservant la qualité par

les tests et le design, la vélocité par l’automatisation.

# Planning

Le projet sera géré en gestion de projet agile orienté Scrum. Le Test Driven Developement (TDD) sera utilisé afin de garantir la non-régression et la confiance dans le code durant le développement de l'application.
Chaque sprint sera composé des éléments suivants (certains seront supprimés si inutile en fonction objectifs du sprint) :

- Analyse fonctionnelle
    - Analyse UML
    - Analyse technique
    - Risques
- Tests unitaires / Développement (TDD)
    - Modèle, service, vue, ergonomie web
- Refractoring

## Livrables projet

Afin d'avoir un objectif clair sur nos deux livrables, nous devrons prioriser le contenu des sprint en fonction des livrable de chaque lot. Les fonctionnalités les plus risquées seront traitées en priorité.

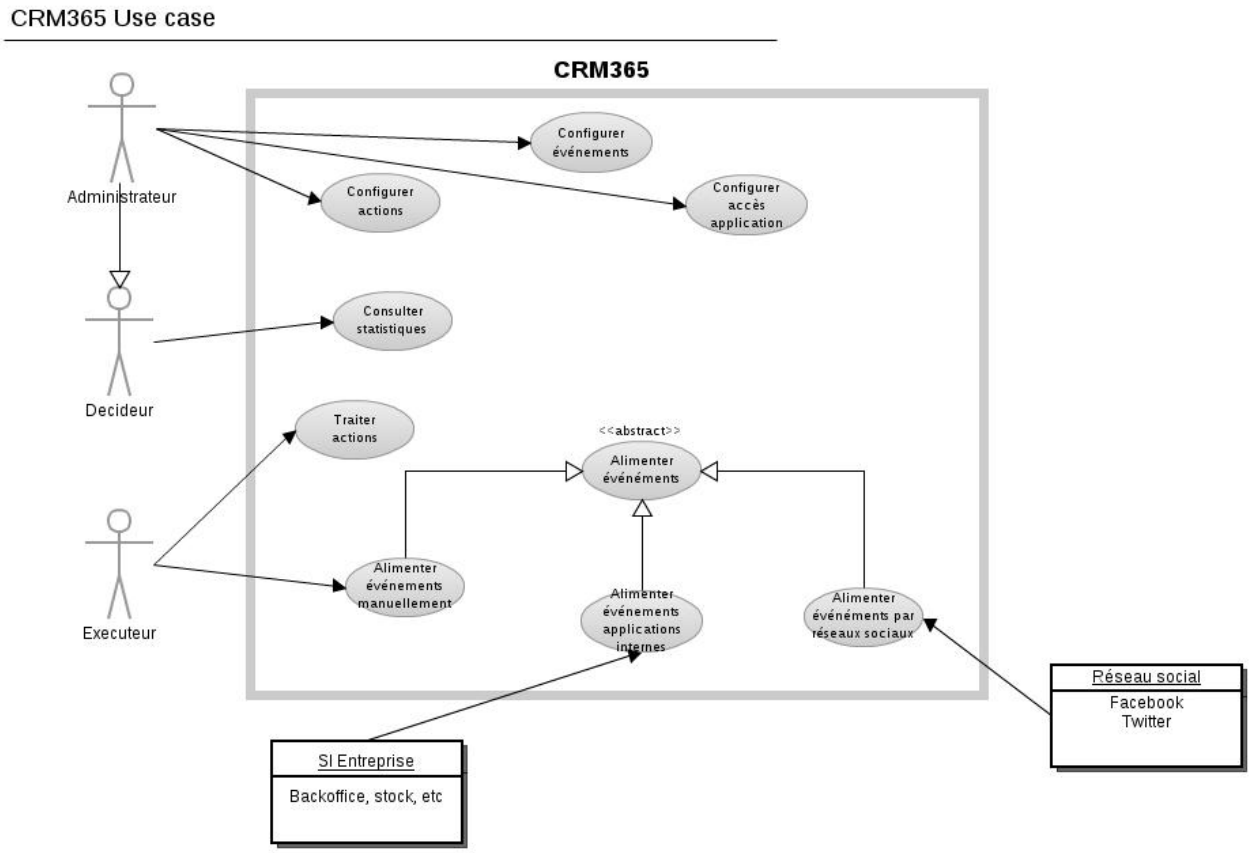
| Version | Livrable | Deadline |
|---|---|---|
| PROTO | Prototype de l'application avec accès web à 2 pages :<br>- page d'authentification<br>- page accueil<br><br>Composants et framework de base intégrés pour valider leur intégration | 17/02 |
| SPRINT 1 | L'entreprise ciblée pour le développement est une agence de vente de téléphone mobile. Le sprint 1 sera réalisé suite à la livraison du prototype.<br>Les événements intégrés automatiquement via des fichiers CSV seront :<br>● Inscription d'un client : date, numéro client, nom, prénom, adresse, téléphone fixe, téléphone mobile, email et informations complémentaires à définir.<br>● Achat d'un mobile : client, date, modèle, prix<br>Les actions générées seront les suivantes :<br>● Condition : Prix mobile > 100€ générera l'action "Appeler le client pour lui offrir une housse de portable".<br><br>Fonctionnalités :<br>- configuration d'événements à récupérer (définition format)<br>- alimentation d'informations clients via fichier CSV<br>- alimentation d'événements via fichier CSV<br>- configuration d'action à générer (définition critères de génération)<br>- service de génération d'actions<br>- visualisation des actions générées | 02/03 |

Le mode de développement agile nous permettra de rapidement valider notre modèle et de mieux nous rendre compte des difficultés et des délais nécessaires pour l'implémentation des livrables. Le développement sera réalisé sur un exemple d'entreprise bien précis pour que les développements soient concrets. Nous gardons à l'esprit qu'ils seront effectués de manière générique afin d'assurer l'adaptation du système aux PME de secteurs différents.

Le diagramme de Gantt est fourni en document additionnel au format PDF.

# Analyse des fonctionnalités

Le diagramme des cas d'utilisation suivant présente les acteurs et les fonctionnalités auxquelles ils auront accès. Cette liste représente la liste exhaustive des fonctionnalités définis à cette date qui seront priorisées lors de la définition des sprints.




Remarques :
L'accès à l'application n'est possible qu'après s'être identifié. Les comptes fournis respecteront donc les trois groupes décrits dans le diagramme : administrateur (1 ou 2 comptes), décideur et exécuteur. Les fonctionnalités seront gérées dans des pages différentes, les groupes auront donc uniquement accès aux pages qui leur sont dédiées.

Le prototype a pour but de construire l'application en fournissant une page de login fonctionnelle intégrant tous les composants de l'architecture (base de données, serveur d'application et client léger web).

Les fonctionnalités analysées dans ce document sont celles répertoriées pour le prototype et le Sprint 1 :

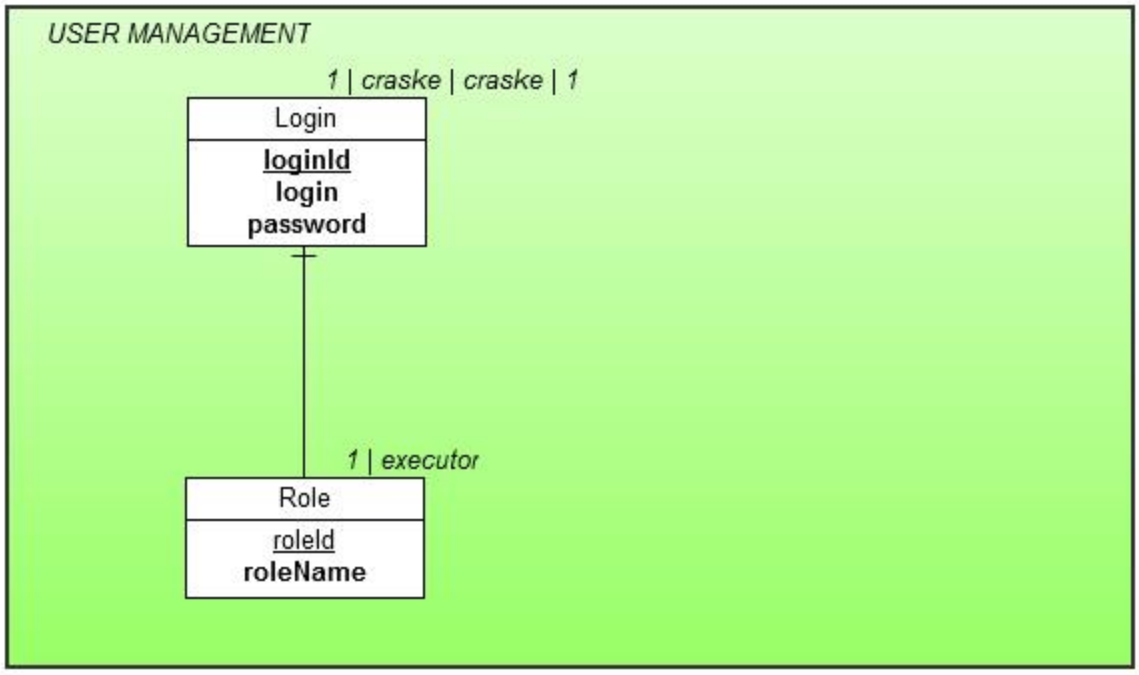
- Authentification

- Configuration d’événements à récupérer
- Alimentation d’informations clients via fichier CSV
- Alimentation d’événements via fichier CSV
- Configuration d’actions à générer
- Service de génération d’actions
- Visualisation des actions générées

## Authentification

La partie authentification consiste à réserver l'application à des utilisateurs disposant d'un login et d'un mot de passe. Le but est de sécuriser l'accès aux fonctionnalités pour les différents rôles (administrateur, décideur, exécutant).

**Persistance**
L'authentification requiert la création de la table "Login" et de la table "Role". Un utilisateur pourra avoir un seul rôle.




**Service**
Un premier service à développer est de valider l'authentification en prenant en paramètre un login et un mot de passe. Ce service doit permettre d'être appelé depuis la couche présentation pour valider les identifiants entrés par l'utilisateur.

Le second service est une classe contrôleur utilisant Spring Security qui interceptera les pages et vérifira le rôle du login en envoyant une requête à la base de données pour savoir si le login peut accéder à la page demandée.

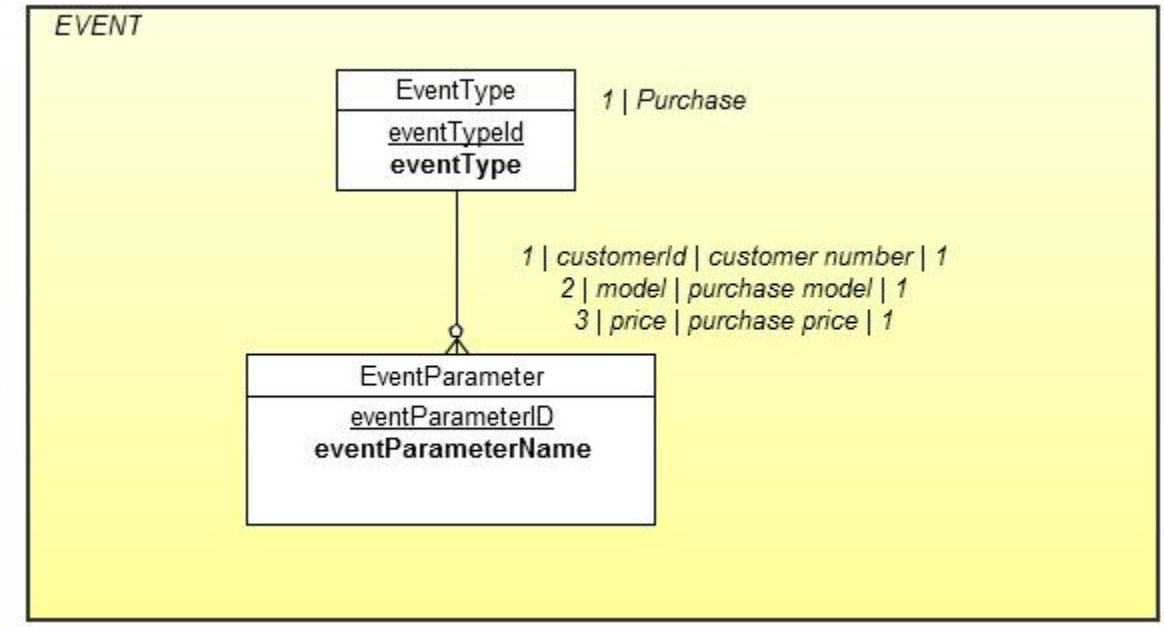
**Présentation**
Deux pages sont à développer. Une première de connexion avec deux champs login et password. Une seconde pour l'instant vide qui contiendra uniquement l'URL de la home page de l'application. Le header de page contiendra un lien permettra de se déconnecter et revenir sur la page de login.

La première page utilisera le service d'authentification pour savoir si les valeurs rentrées sont correctes. Si le login est correct, l'utilisateur doit arriver sur la home page de l'application, sinon il doit redirigé vers la page de connexion avec un message d'erreur indiqué.

## Configuration d'import d'événements automatiques

Cette fonctionnalité consiste à permettre à l'utilisateur de configurer les événements qu'il va récupérer par des fichiers en format CSV. La configuration est constituée de la création d'un événement à récupérer et de la définition du format du fichier. D'un point de vue développement, cette fonctionnalité consiste uniquement à mettre à jour les données saisies par l'utilisateur dans la base de données. Il n'y a pas d'opération d'entréé / sortie avec des fichiers.

**Persistance**
On stockera les événements dans une première table. On attachera ensuite à un événement tout les paramètres requis lors de son import.




Pour les exemples d'enregistrements ci-dessus, le fichier attendu sera nommé "Purchase.csv" avec la première ligne du fichier en header contenant : "PURCHASE;CUSTOMERID;MODEL;PRICE"

Un exemple de fichier valide est :
*"Purchase.csv"*
PURCHASE;CUSTOMERID;MODEL;PRICE
PURCHASE;1;SAMSUNG;120

L'unicité de la clef fonctionnelle "eventType" permettra de sécuriser que l'on puisse pas avoir de conflit. Le nom du fichier sera bien unique pour un événemen (i.e. "Purchase" est forcément alimenté par un fichier "Purchase.csv").

**Service**
Cette fonctionnalité nécessite le développement de deux classes DAO pour chacune des tables

listées ci-dessus avec les opérations CRUD.

**Présentation**
La page de configuration doit être créée. Elle sera présente dans l'arborescence "Event Management" sous le nom "Automatic import configuration". Toutes les pages dans l'arborescence "Event Management" seront uniquement accessible pour le rôle administrateur.

Cette page doit permettre de :

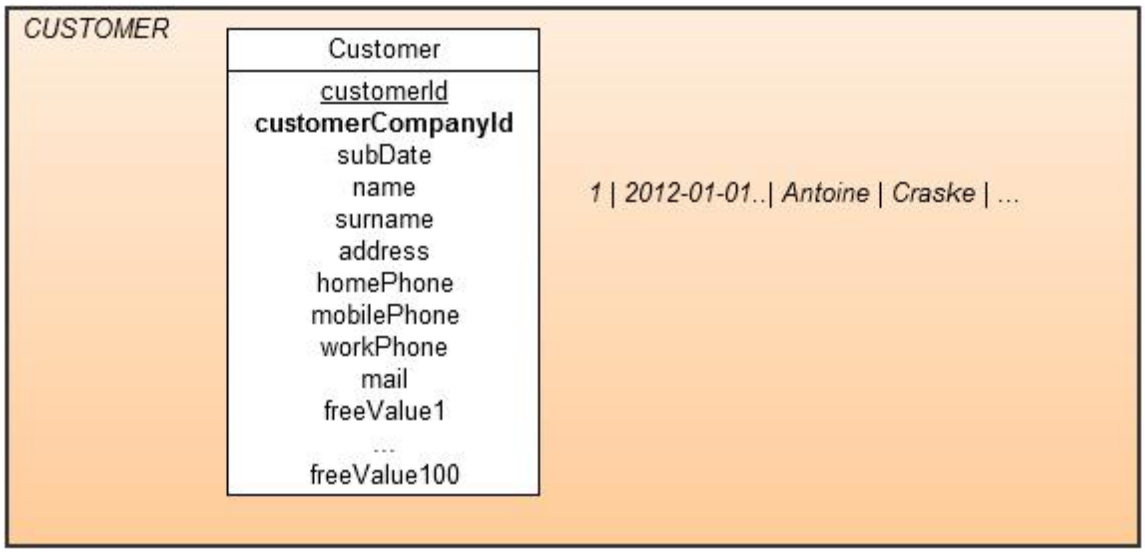
- Créer, modifier, supprimer un événement
- Ajouter, modifier, supprimer des paramètres sur un événement choisi

## Alimentation d'informations clients

L'intégration des données clients est une fonctionnalité centrale de l'application. En effet, cette fonctionnalité permettra d'alimenter les informations nécessaires pour contacter le client. L'alimentation des informations clients ne nécessitent pas de configuration d'un événement dans l'interface web, cette fonctionnalité sera livré de base avec le logiciel. Cette fonctionnalité permettra l'intégration uniquement par fichier en format CSV.

**Persistance**
Les données clients seront centralisées dans une seule table "Customer". Comme le paramétrage de l'alimentation des informations clients n'est pas possible dans l'application, 100 champs libres seront réservés à l'entreprise utilisatrice afin de garder en flexibilité pour l'entreprise utilisatrice. L'administrateur de l'application sera responsable de la cohérence et des données insérées. On peut imaginer un enrichissement de la fonctionnalité où l'administrateur pourrait explicitement décrire quelle est la donnée stockée dans la colonne afin de faciliter la lecture de la fiche client mais ce n'est pas l'objectif dans ce sprint.

La table est définie comme suit :



**Service**
La couche de service aura une classe de DAO sur la table “Customer” permettant d’effectuer les opérations CRUD avec en plus des recherches par id technique, numéro de client, nom et prénom.

L’intégration des données devra être réalisée par un fichier CSV avec un header contenant les mêmes colonnes que la table client de CRM 365. Il devra être nommé “Customers.csv”. Ce nom sera donc réservé pour l’application et il ne doit pas être possible de définir d’import automatique sur ce nom pour éviter les conflits.

CRM 365 ne sera pas maître sur les données. L’application a bien pour but de centraliser l’information, il ne sera donc pas possible de directement mettre à jour des informations de la fiche client dans CRM 365. Il faudra forcément passer par un fichier CSV qui proviendra vraisemblablement du back-office de l’entreprise pour l’insertion et la mise à jour d’information.

**Présentation**
Cette fonctionnalité ne nécessite pas de développement sur la partie interface utilisateur.

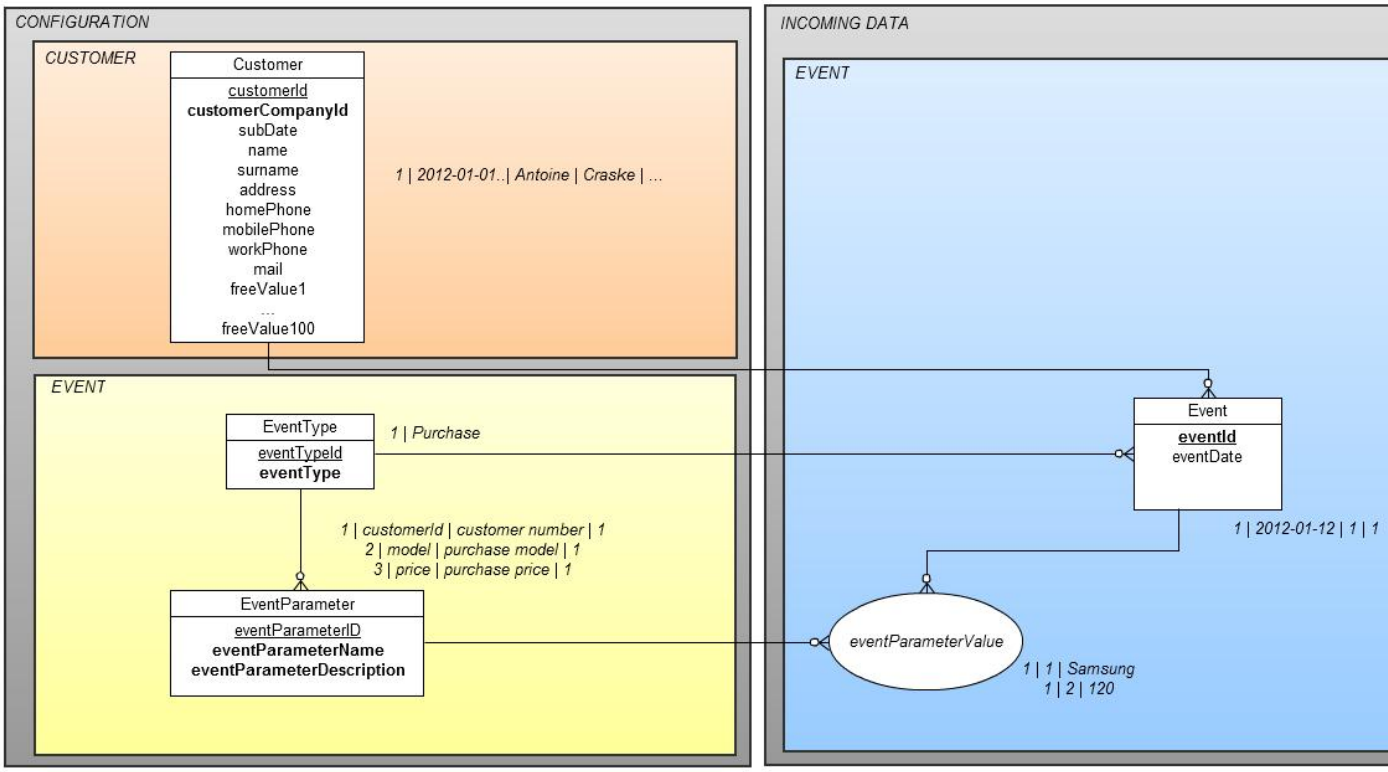
# Alimentation d'événements configurés

Cette fonctionnalité consiste à développer le moteur nécessaire à l'intégration de fichier CSV respectant la configuration d'événement définis dans l'interface web.

**Persistance**
La partie configuration est déjà gérée dans la fonctionnalité permettant de configurer des imports d'événement automatiques. Cette fonctionnalité consiste à stocker dans notre base de données les événements reçus en fichiers CSV.

On va stocker à un premier niveau les événéments reçus avec des informations de dates de réception. On stocke ensuite les valeurs des paramètres reçus dans une table d'association.

Les tables "event" et "eventParameterValue" concernées sont définies ci-dessous :



**Service**
Le service à développer devra dans un premier temps vérifier la validité du fichier. Le traitement sera lancé si un fichier nommé comme l'événement en suffixe ".csv" est présent dans le répertoier de réception défini pour l'application.

Il faut développer un parseur de fichier adapté à notre modèle de données qui devra extraire la première ligne du fichier. Cette ligne doit ensuite être découpée suivant le séparateur csv pour récupérer tout les champs. Il faut ensuite valider (dans l'ordre des paramètres) que les noms des champs sont biens les mêmes que ceux des paramètres. Si le fichier ne respecte pas le paramétrage défini, on lévera une exception.

Une fois que le service a validé le format du fichier, il faut effectuer un traitement batch consistant à effectuer une suite d'opérations unitaires. Une opération unitaire est ici d'insérer une ligne dans la table événement et pour chaque colonne correspondant à un paramètre, insérer les lignes dans la table d'association. La partie optimisation du cache et de la réutilisation des connexions est gérée par le framework Spring JDBC.
Deux classes de DAO avec les opérations CRUD et celles nécessaires à l'optimisation du traitement sont donc également à développer pour les tables "Event" et "EventParameterValues".

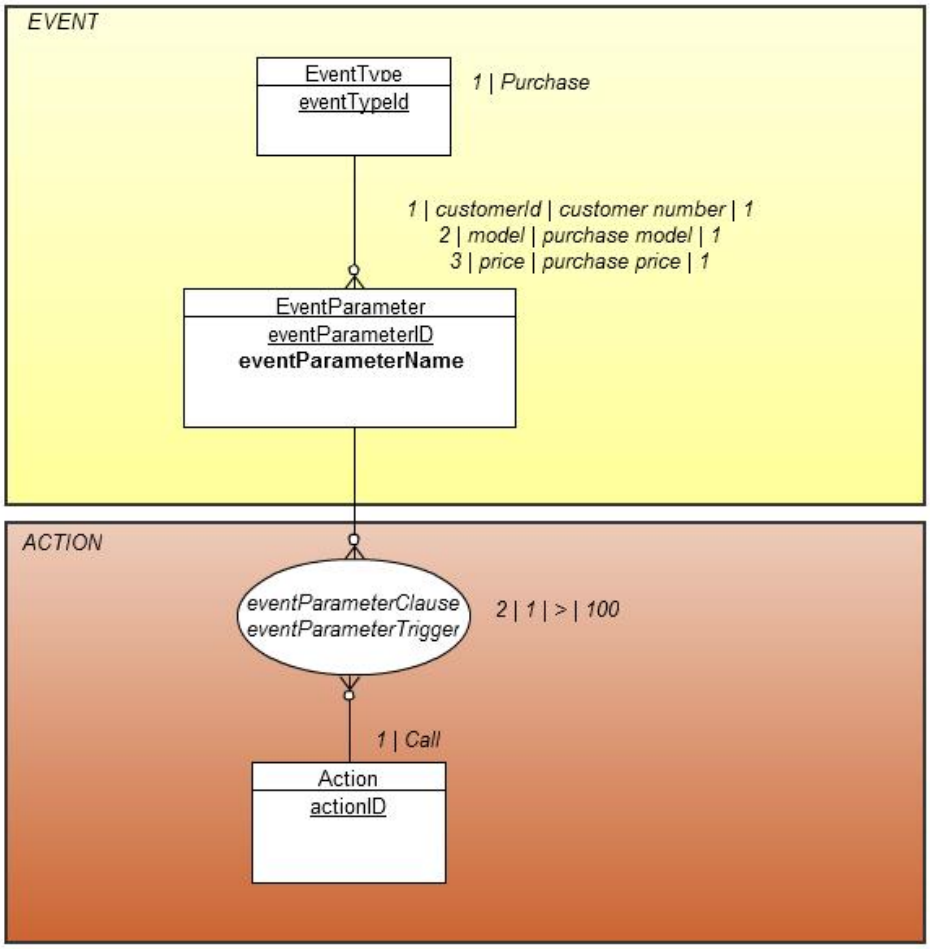
**Présentation**
Cette fonctionnalité ne nécessite pas de développement sur l'interface web.

## Configuration d’actions à générer

Cette fonctionnalité consiste à permettre au rôle administrateur de configurer les critères de génération d’une action.

**Persistance**
La configuration se fera à deux niveaux. Un premier où l’on définira l’action à générer. Un second où l’on associera à l’action les critères pour générer l’action. Les critères sont associés au valeur des paramètres des événements configurés.

La configuration sera donc réalisée dans les tables “Action” et “TriggerCriteria” présentées ci-dessous :




**Service**
Il faut développer les classes DAO associées aux deux tables définies ci-dessus. Elles implémenteront les opérations CRUD classiques.

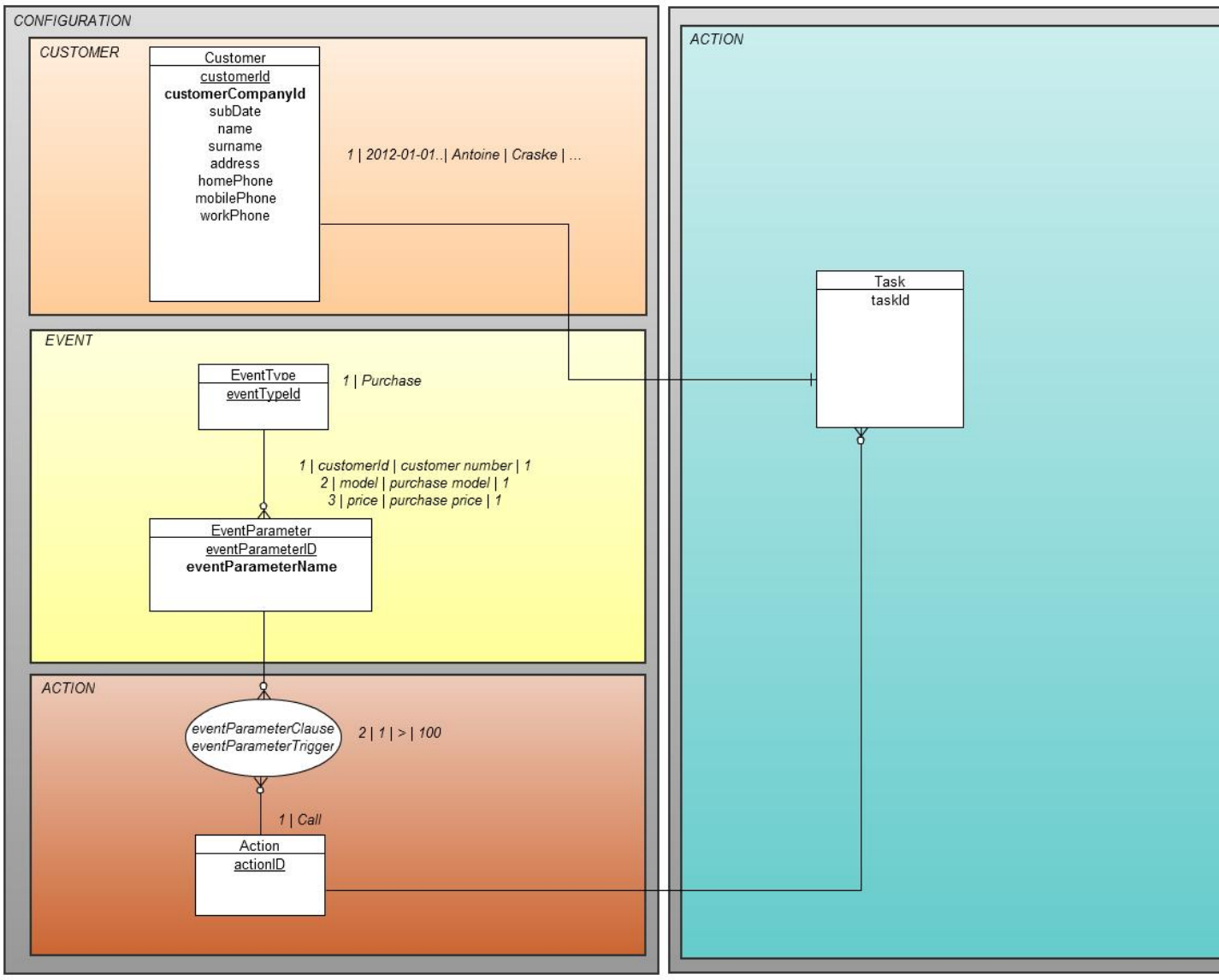
**Présentation**

La fonctionnalité requiert la création d'une nouvelle page "Action trigger configuration" dans le groupe "Action". Cette page sera uniquement accessible au rôle administrateur. Elle devra permettre la création, mise à jour et suppression d'une action ainsi que l'ajout, mise à jour et suppression de critères pour la génération d'une action. Les critères de générations seront disponibles sous forme de liste déroulante afin de sécuriser la saisie et la future génération des actions. Les critères disponibles seront ceux de la norme SQL3.

## Service de génération d'actions

Ce service consiste à générer des actions suite aux critères de génération définis par l'utilisateur. On ne traite pas la mise en place de l'exécution automatique de la génération des actions mais uniquement la création du service.

**Persistance**
Les actions seront stockées dans la table "Task" définie ci-dessous. On conserve le lien avec le client afin de savoir pour quel client l'action doit être exécutée.




**Service**
Le service devra pour toutes les actions définies, vérifier si les critères de générations sont

satisfaits dans les événements intégrés depuis la dernière date de traitement. Le critère appliqué sera celui défini dans la table des critères, il pourra être incorporé dans une chaîne SQL car la saisie des critères a été sécurisée dans l'application.

**Présentation**
Cette fonctionnalité ne nécessite pas de développement sur l'interface web.

## Visualisation des actions générées

Cette fonctionnalité permetta aux utilisateurs d'avoir une vue sur les événements intégrés sur différents périmètres de dates. Ce reporting sera général à l'application, il permettra de rapidement se rendre compte de soucis technique.

**Persistance**
Aucune modification n'est requise sur cette couche car la fonctionnalité exploite des données déjà présentes dans le modèle de données.

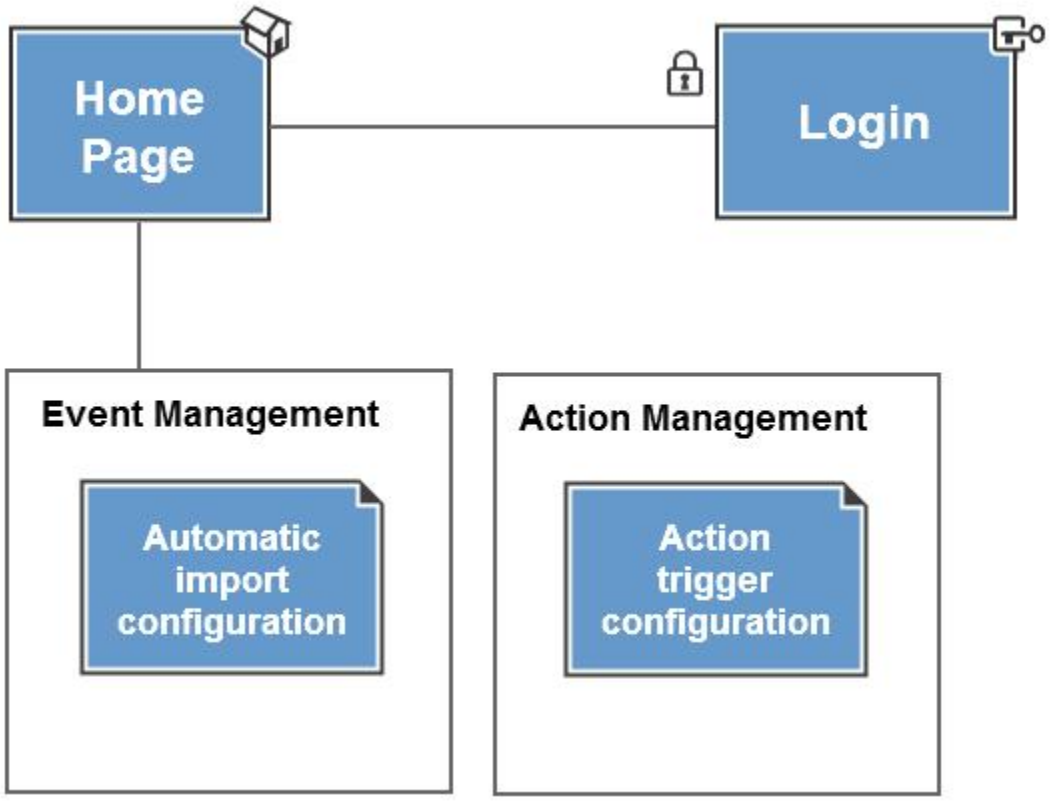
**Service**
Le service à développer doit fournir le nombre d'événements intégrés, le nombre d'actions générées, le nombre de clients insérés et modifiés  en prenant en paramètre une date de début et une date de fin représentant les bornes pour le calcul des données.

**Présentation**
Le service sera utilisé sur la "Homepage". Pour cette première version, on affichera les données ligne par ligne en prenant les périmètres suivants :

- Date du jour
- Jour précédent
- Semaine

Le but n'étant pas de fournir un reporting complet en homepage car il sera développer dans une fonctionnalité dédiée. Cette page sert vraiment de premier niveau de reporting quand on se connecte à l'application pour avoir une vue générale de l'activité de l'application. Les données de chaque périmètre seront affichés les uns à la suite des autres.

# Arborescence des pages web



# Risques identifiés

CRM 365 à des impératifs de :

- Généricité
- Portabilité

Pour palier à cela, chacun des développements effectués devra respecter les standards du ou des langages utilisés ainsi que sur le plan des techniques utilisées.

### Standardisation

Nos développements au niveau base de données utiliseront le langage SQL respectant la norme SQL. Ceci dans le but d'avoir un code qui sera portable et non dépendant d'un SGBD particulier. De plus notre application étant destinée à une utilisation à travers le web, celle-ci devra être compatible avec la plupart des navigateurs qui sont sur le marché et donc susceptible d'être utilisé par nos clients. Ce qui implique aussi pour la partie de l'application coté client une rigueur dans la vérification de compatibilité avec les navigateurs au risque d'avoir une application ne fonctionnant pas correctement pour certains de nos clients.

Ensuite un risque identifié serait d'avoir un code non lisible ou en tout cas pas uniforme entre les différents développeurs. Pour répondre à cela il a été décidé de respecter des conventions de codage (nommage des classes, interfaces et variables) ainsi que sur les commentaires. Des phases de refractoring sont également prévues afin d'améliorer l'architecture logicielle en évitant la duplication de code.

### Etapes de développement

Un autre risque est de ne pas pouvoir s'assurer que l'ensemble du développement respecte ces critères car le projet est assez volumineux en termes de fonctionnalités à implémenter mais aussi également par la durée du projet. L'aspect travail en collaboration est également à prendre en compte.

Afin de gérer et contrôler ce risque il sera primordial d'être rigoureux dans les différents cycles de développement. Pour nous aider dans cela nous mettrons en place différentes étapes dans les développements. Celles-ci s'articulent autour de 3 points principaux :

- Développement de tests unitaires pour chacun des services à implémenter
- Validation des services implémentés par le biais des différents tests unitaires
- Reporting tous les 15 jours afin d'effectuer une revue de code et remédier si besoin à d'éventuels problèmes rencontrés

### Déploiement sur serveurs d'applications

Un risque est aussi présent sur le fait que si l'application doit être déployée sur des serveurs d'applications différents ( Tomcat, JBoss, etc …), que ce déploiement soit bien sur possible.
Pour s'en assurer il faudra tester le déploiement sur différents types de serveurs et donc prévoir

plusieurs descripteurs de déploiement suivant le serveur, car ces descripteurs sont liés aux différents serveurs. Et ces fichiers ne sont pas portables d'un serveur d'application à l'autre.

# Glossaire

**Événement**
Action effectuée par un client (e.g. achat, retour, envoi mail, ouverture de mail, appel téléphonique).

**Action**
Tâche à réaliser.

**CRM**
Customer Relationship Management : Gestion de la relation client - ensemble d' outils et de techniques destinés à capter, traiter, analyser les informations relatives aux clients et aux prospects, dans le but de les fidéliser en leur offrant le meilleur service.

**SI**
Système d'Information : ensemble organisé de ressources (matériels, logiciels, personnel, données et procédures) qui permet de regrouper, de classifier, de traiter et de diffuser de l'information sur un environnement

**O/S**
Système d'exploitation, abrégé SE (en anglais operating system, abrégé OS), est l'ensemble de programmes central d'un appareil informatique qui sert d'interface entre le matériel et les logiciels applicatifs.

**SGBD**
Système de gestion de base de données, c'est un logiciel système destiné à stocker et à partager des informations dans une base de données, en garantissant la qualité, la pérennité et la confidentialité des informations, tout en cachant la complexité des opérations.

**SQL**
Structured Query Language est un langage informatique normalisé qui sert à effectuer des opérations sur des bases de données

**Descripteur de déploiement**
Le descripteur de déploiement est un document XML qui décrit le contexte d'une application WEB (par convention, on le note web.xml)

# Annexe

## Dictionnaire de données

| Champ | Type | Longueur | Description |
|---|---|---|---|
| loginId | BIGINT | 19 | id technique |
| login | VARCHAR | 50 | login de l'application |
| password | VARCHAR | 50 | mot de passe relié au login |
| roleId | BIGINT | 19 | id technique |
| role | VARCHAR | 50 | nom rôle |
| eventTypeId | BIGINT | 19 | id technique |
| eventType | VARCHAR | 50 | libellé d'un événement |
| eventParameterId | BIGINT | 19 | id technique |
| eventParameterName | VARCHAR | 50 | libellé d'un paramètre d'un événement |
| eventParameterDescription | VARCHAR | 50 | description d'un libellé de paramètre d'événement |
| eventId | BIGINT | 19 | id technique |
| eventDate | DATE | 23 | date intégration de l'événement |
| eventParameterValue | VARCHAR | 100 | valeur du paramètre d'un événement |
| customerId | BIGINT | 19 | id technique |
| customerCompanyId | BIGINT | 19 | id client de la société |
| subDate | BIGINT | 19 | date d'insertion du client dans CRM 365 |
| name | VARCHAR | 50 | nom du client |
| surname | VARCHAR | 50 | prénom du client |
| address | VARCHAR | 300 | adresse du client |
| homePhone | VARCHAR | 15 | téléphone domicile du client |
| mobilePhone | VARCHAR | 15 | téléphone mobile du client |
| workPhone | VARCHAR | 15 | téléphone travail du client |
| mail | VARCHAR | 100 | email du client |
| freeValue1 to freeValue100 | VARCHAR | 100 | champ libre réservé société |

| | | | |
|---|---|---|---|
| eventParameterClause | VARCHAR | 50 | critère de génération d'action (>, >=, etc) |
| eventParameterTrigger | VARCHAR | 50 | valeur qui déclenchera l'action si le critère appliqué à cette valeur renvoie vrai |
| actionID | BIGINT | 19 | id technique |
| actionDescription | VARCHAR | 100 | libellé de l'action |
| taskId | BIGINT | 19 | id technique |

# Modèle de données

Sprint 1 : Database model SPRINT 1