PROJ
M1-FA-MIAGE

CRM 365
Analyse fonctionnelle

- *Prototype*
  - *Lot 1*
  - *Lot 2*

Groupe :
Antoine Craske
Tarik Djebien
Eric Sitraka Rakotobe
Rudy Stienne

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 trois 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.

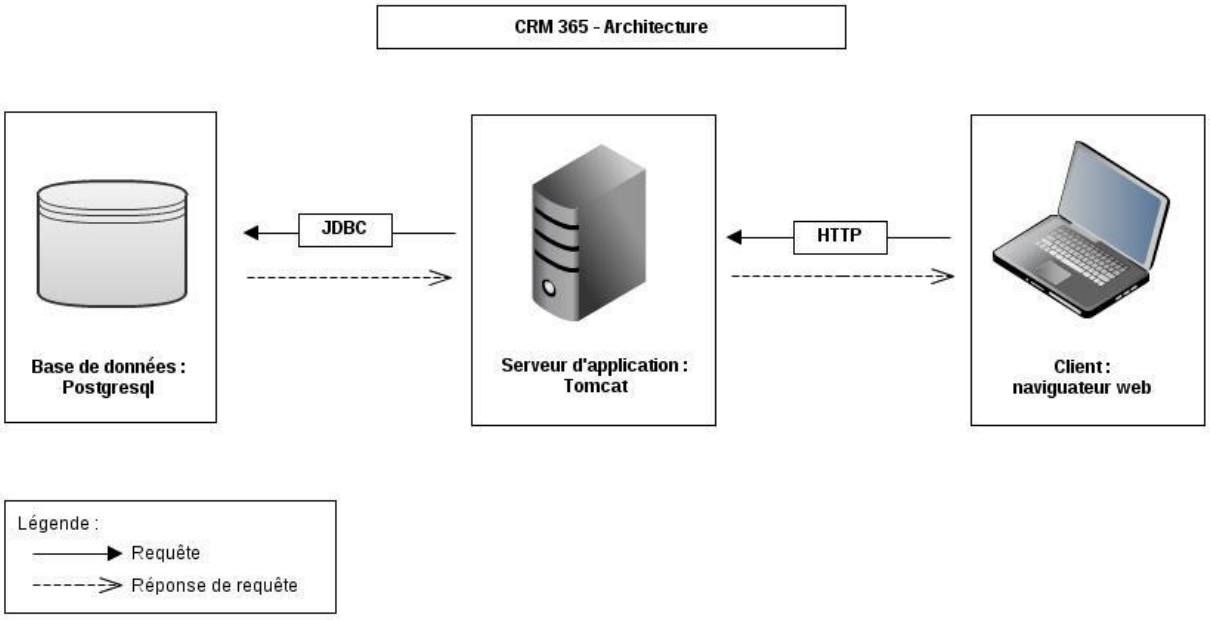
Les schémas ont étés réalisés suivant le formalisme UML afin de simplifier la compréhension à travers la standardisation des documents.

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 MySQL 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'une archive de format WAR. 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, Hibernate, Spring (JDBC, IoC, Security, MVC), JUnit, Mockito, Selenium et Sonar.

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.

Notre application doit être portable sur la plus grande variété de modèle de bases de données utilisés en entreprise. La norme SQL3 n'étant pas respectée par tous les vendeurs, Hibernate nous permettra d'assurer la portabilité sur quasiment tous les modèle de bases de données relationnelle. Ce framework nous assurera le mapping ORM (ie Object Relation Mapping) nous permettant d'être plus productif en évitant d'avoir à réecrire les mécanismes de transformation entre monde objet et relationnel.

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. L'architecture de la partie présentation 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.

Sonar est un outil qui aggrège la plupart des outils d'audit de code du marché Java pour fournir des tableaux de bord de qualité de codage par rapport aux standards, spécifications et bonnes pratiques. Cela nous permettra d'assurer une meilleure maintenabilité de notre application.

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.

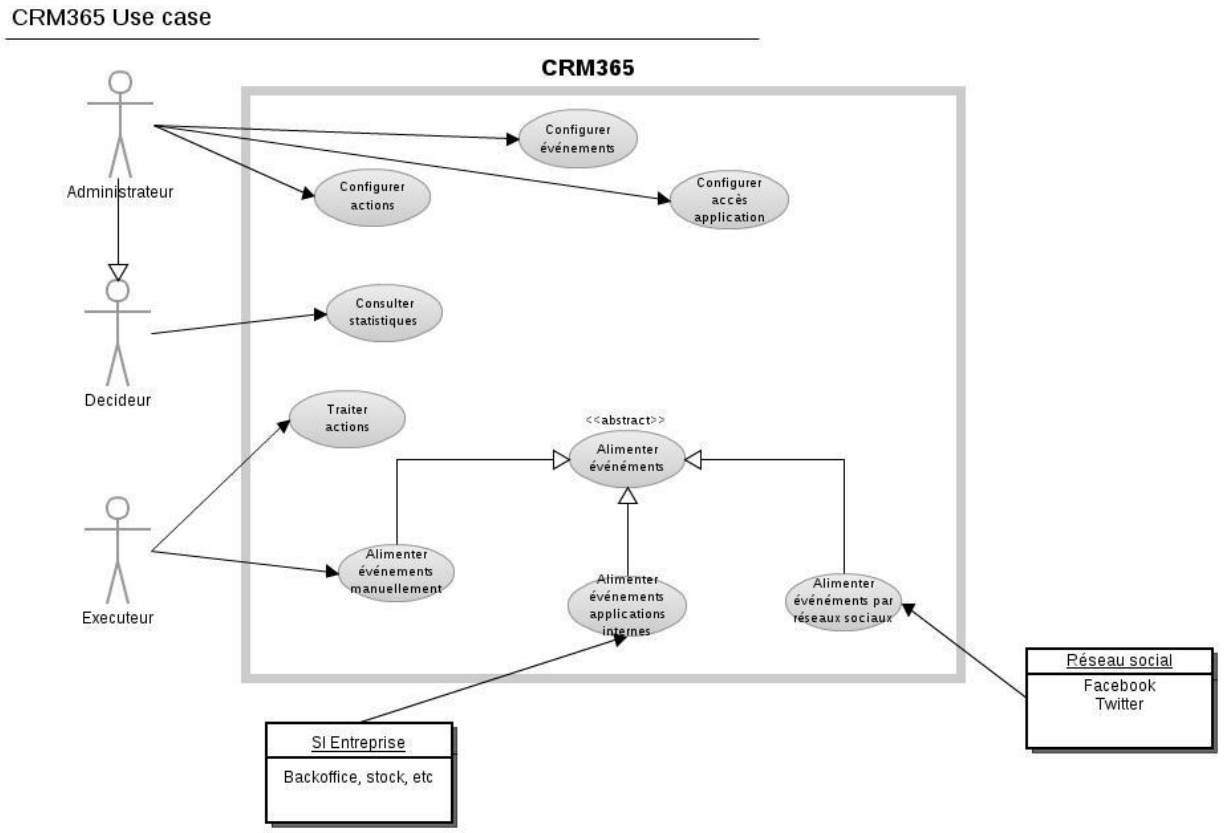
Version	Livrable	Deadline
PROTO	Prototype de l'application avec accès web à 2 pages : - page d'authentification - page accueil Composants et framework de base intégrés pour valider leur intégration	17/02
SPRINT 1	L'entreprise ciblée pour le développement est une agence de vente de téléphone mobile. Le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

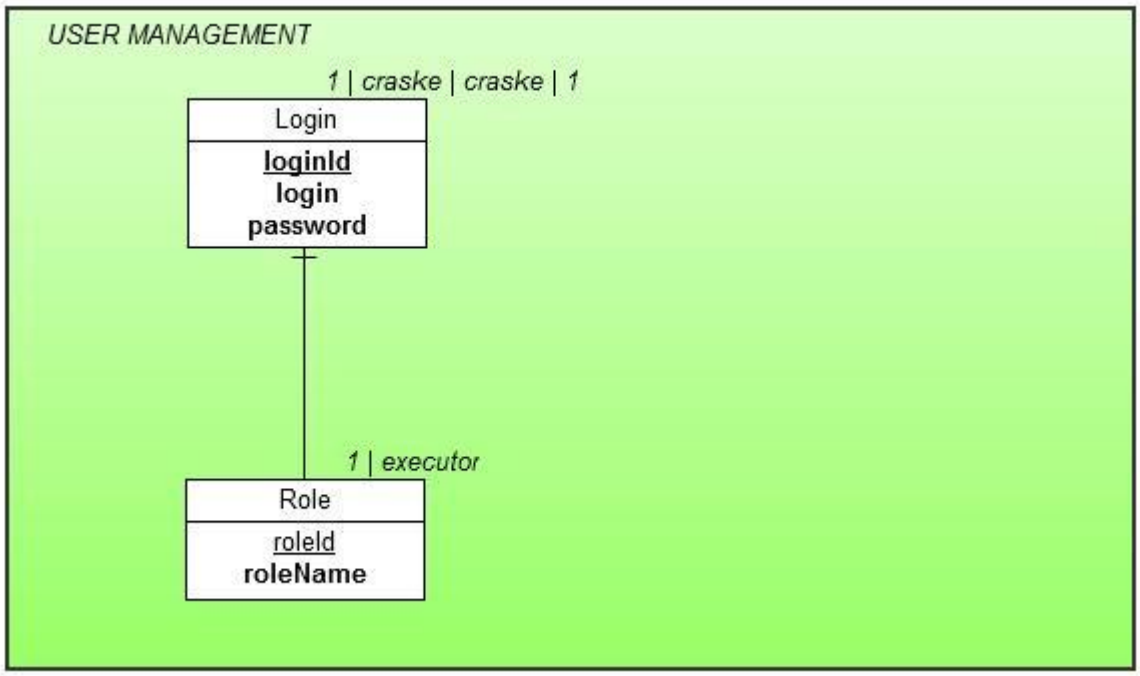
Les fonctionnalités analysées dans ce document sont celles répertoriées pour le prototype et le Sprint 1 :

- 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.

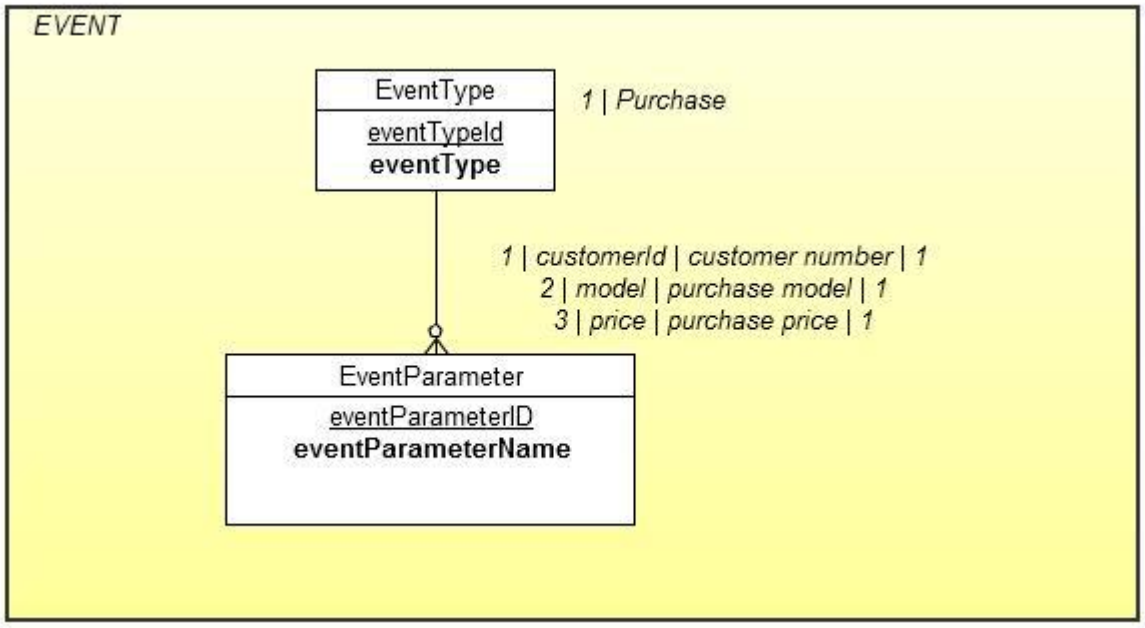
Présentation
Deux pages sont à développer. Une première de connexion avec deux champs login et password. Une seconde pour l'instant vide qui contiendra uniquement l'URL de la home page de l'application. Le header de page contiendra un lien permettra de se déconnecter et revenir sur la page de login.

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 :

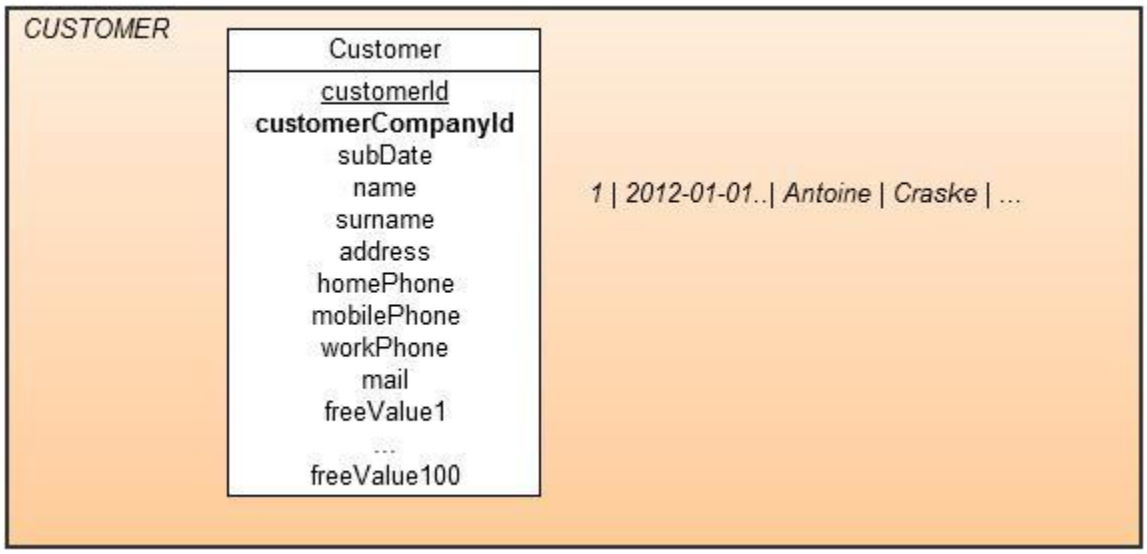
- Créer, modifier, supprimer un événement
- Ajouter, modifier, supprimer des paramètres sur un événement choisi

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.

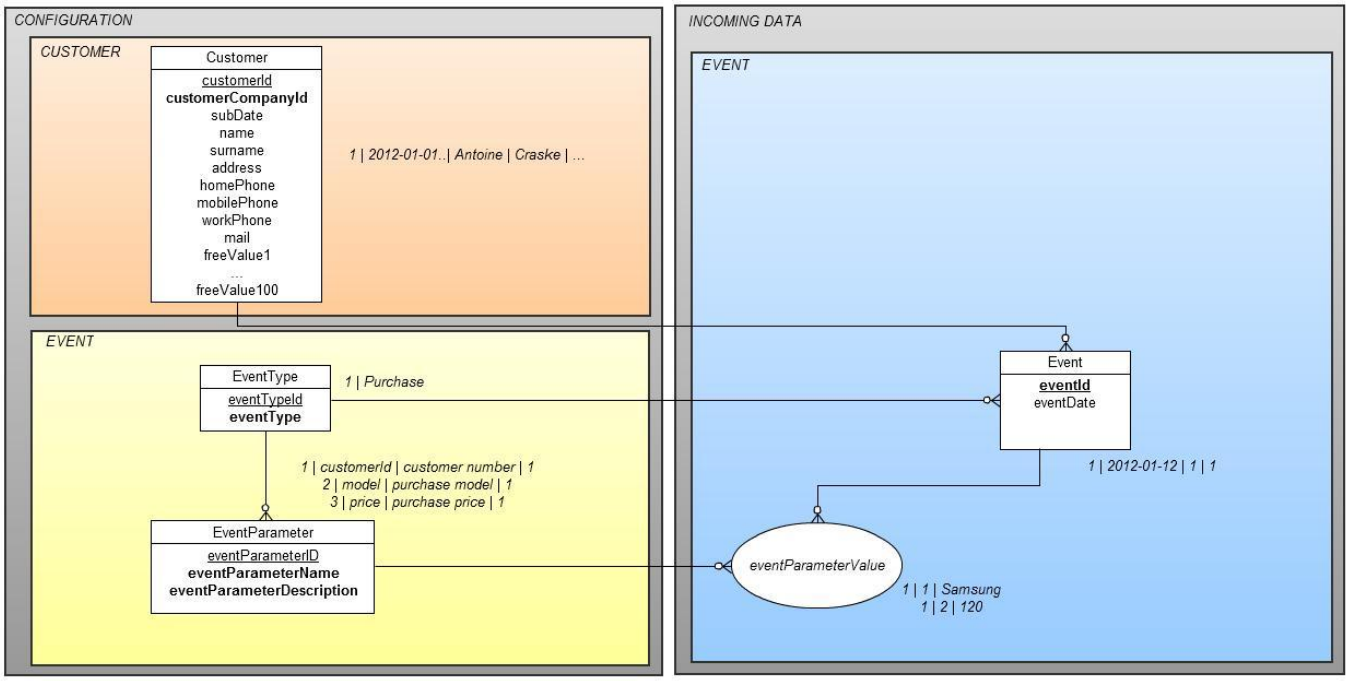
Alimentation d'événements configurés

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".

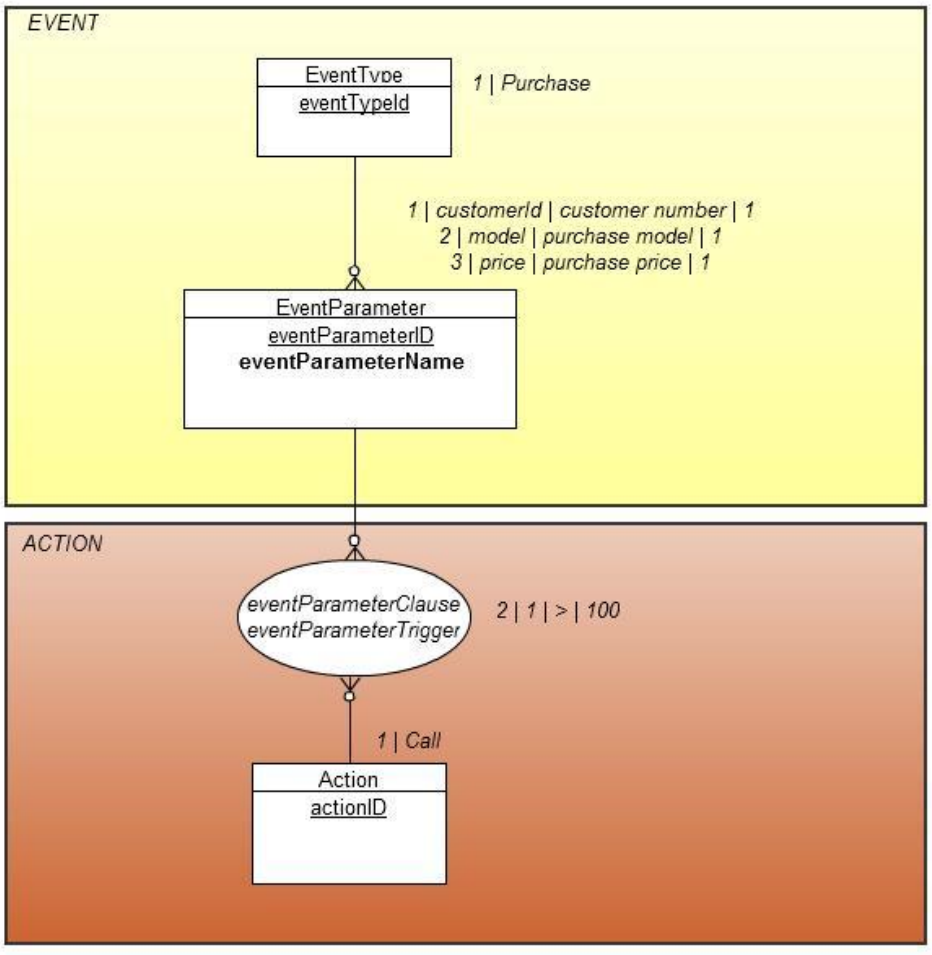
Présentation
Cette fonctionnalité ne nécessite pas de développement sur l'interface web.

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

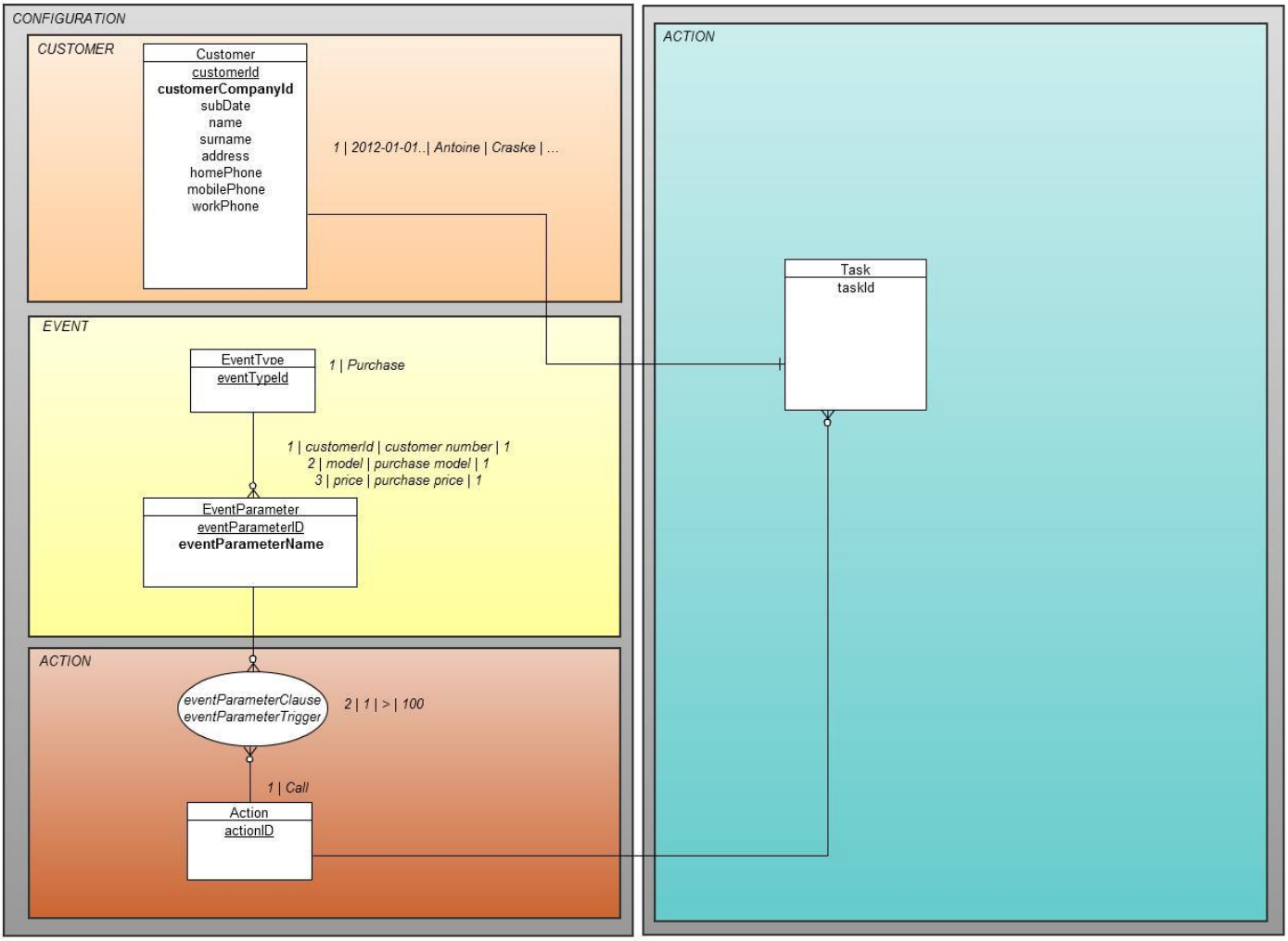
La fonctionnalité requiert la création d'une nouvelle page "Action trigger configuration" dans le groupe "Action". Cette page sera uniquement accessible au rôle administrateur. Elle devra permettre la création, mise à jour et suppression d'une action ainsi que l'ajout, mise à jour et suppression de critères 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.

Workflow des tâches

Cette fonctionnalité doit permettre à un utilisateur de rôle exécutant de traiter les tâches, c'est à dire :

- S'assigner la tâche
- La valider une fois terminée
- Générer un événement suite à la tâche terminée

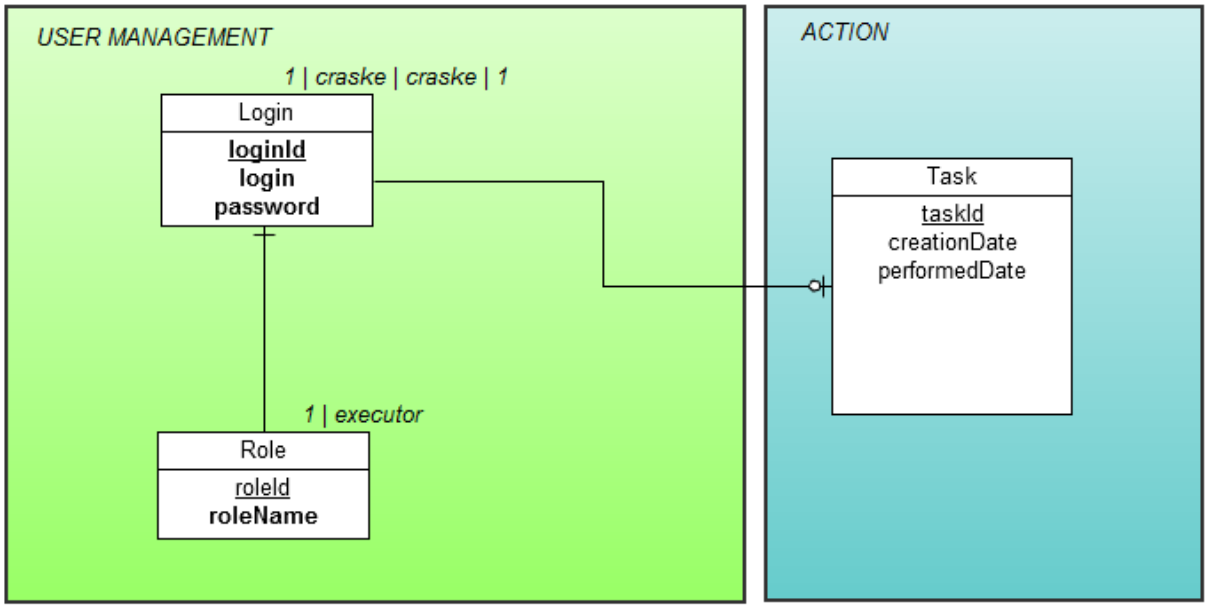
On peut découper cette fonctionnalités en deux livrables :
1/ Réservation, flag tâche réalisée.
2/ Affectation d'une tâche à un groupe lors de sa génération et création d'un event une fois réalisée.

VERSION 1

Réservation, flag tâche réalisée.

Persistance

L'assignement de la tâche est géré par une clef étrangère pointant sur login. Afin de la valider, on remplira un champ "performedDate", on ajoute également un champ "creationDate" qui ne sera pas rempli pour l'instant.



Service

Il faut développer un service d'interface "IWorkflowService" avec les fonctionnalités suivantes :

- boolean assignTask(User)
  renvoie vrai si la tâche a pu être assignée à l'utilisateur, faux sinon
- boolean setPerformedDate(Date)
  renvoie vrai si la date de traitement a pu être fixée, faux sinon
- List<Task> getTasksTodo()
  renvoie la liste des tâches dont la date n'est pas renseignée, ce service utilisera un DAO sur l'entité Tâche.

Présentation
La page “TODO” doit uniquement être accessible par les utilisateurs de rôle exécutant.

La page TODO contient actuellement un tableau avec toutes les tâches, les impacts sont les suivants :
1/ Ajouter un bouton “Réserver” pour chaque tâche.
En cliquant dessus, on doit appeler la fonctionnalité assignTask du service IWorkflowService

2/ Ajouter un bouton “Réalisée” pour chaque tâche
En cliquant dessus, on doit appeler la fonctionnalité setPerformedDate du service IWorkflow service

3/ Il faut ensuite modifier le périmètre des tâches affichées sur la page “TODO” : il faut prendre les tâches dont le champ “performedDate” est vide/null.

VERSION 2

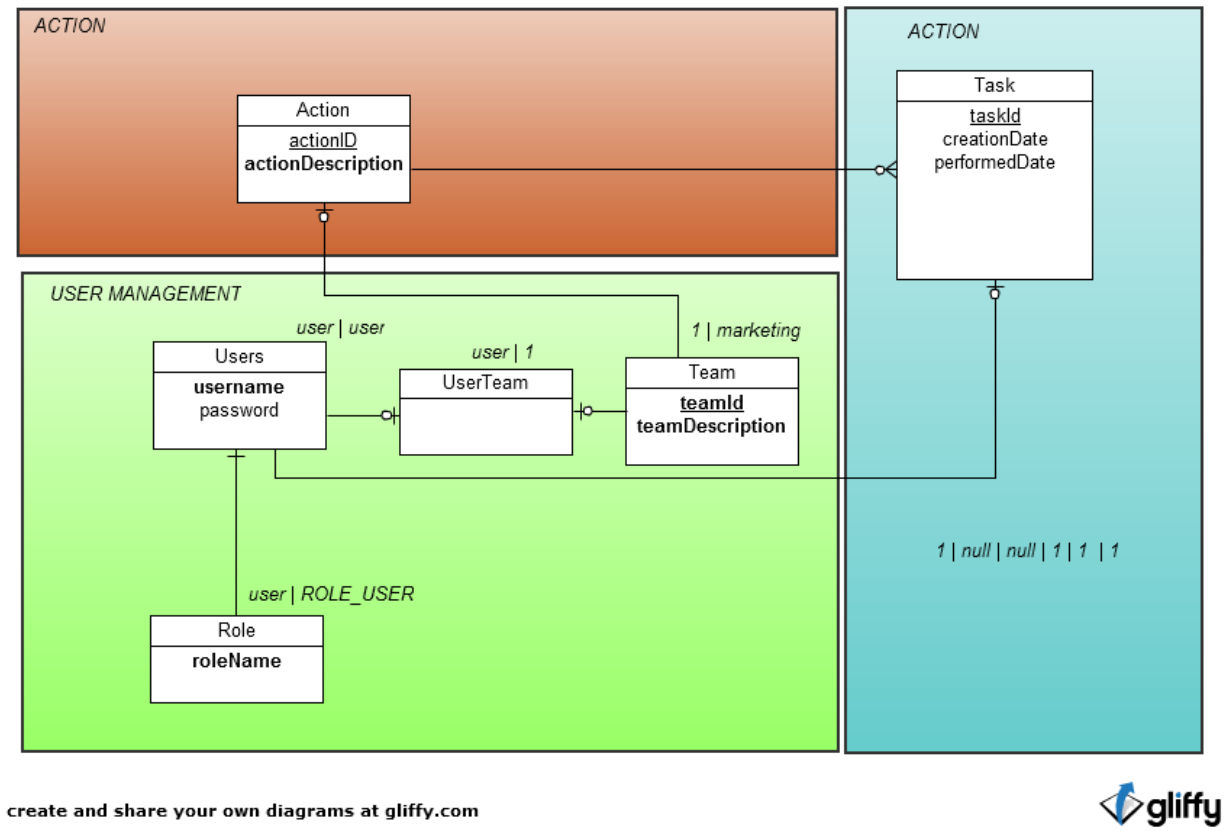
Affectation d'une tâche à un groupe lors de sa génération et création d'un event une fois réalisée.

Persistance

La table Role décrit le rôle fonctionnel de l'utilisateur sur l'application, il faut ajouter une table "Team" qui fera le lien entre des utilisateurs et leur équipe réelle.
Champs : TeamId, TeamDescription

On ajoutera une table d'association entre Team et Users. Users étant une table utilisée par le framework Spring security, la table ne doit pas être modifiée.

On doit ajouter une clef étrangère "targetTeamId" dans la table "ACTION" afin d'affecter la tâche une fois créée à une équipe.



Service

Impacts :

1/ Service de configuration d'action :

- Ajouter un champ "TeamId" pour l'opération de création d'une Action.
- Ajouter automatiquement une configuration d'événement quand une action est configurée.

Exemple dans le service de configuration d'action :

1. On crée une action "call customer"
2. On ajoute un action_trigger "purchasePrice > 100"

La fonctionnalité va ajouter le step 1b suivant dans le service :

1. On crée une action "call customer"

1b. On crée un event_type "call customer" sans event_parameter en appelant le service de création d'event_type

2. On ajoute un action_trigger "purchasePrice > 100"

2/ Service IWorkflowService sur appel de "setPerformedDate" : il faut appeler le service de création d'événement et créer un événement correspondant à l'action traitée.

Exemple (modifications existant en bleu) :

1. La tâche "call customer" est traitée, on appelle setPerformedDate
2. *On crée un event "call customer" sans event_parameter_value, l'event_id est celui de l'event_type "call customer"*

Présentation

Dans la page de configuration des actions, Il faut ajouter un champ "Affectation :" au niveau action avec une liste déroulante contenant toutes les équipes de la table "Team". Une tâche ne peut être affectée qu'à une seule équipe.

La création de l'event en automatique n'a pas de répercussion sur l'interface utilisateur, les changements sont au niveau du service qui est toujours appelé de la même façon.

Administration de l'application

Cette fonctionnalité permettra à l'administrateur de l'application de gérer les utilisateurs :

- Création, suppression, modification, recherche d'un compte utilisateur
- Création, suppression, modification, recherche des équipes
- Modification du rôle d'un utilisateur
- Modification de l'équipe d'un utilisateur

L'accès à cette partie de l'application sera uniquement disponible pour un compte ayant le rôle Administrateur.
Afin de faciliter l'administration, une page sera dédiée à la gestion des utilisateurs et une autre à la gestion des équipes.

Les impacts généraux sont décrits ci-dessous suivis des spécifications détaillées pour la gestion des uilisateurs puis des équipes.

Persistance
Aucune modification n'est nécessaire sur la couche de persistance, les tables Users, Authorities et Team ont étés créées lors de précédents développements.

Service
La création d'un service "AdministrationService" est nécessaire. Il comportera les services spécifiés aux deux paragraphes suivants.

Présentation
Une nouvelle brique nommée "Administration" doit être crée dans le menu contenant deux liens "Users management" et "Team management".
En français : "Administration", "Gestion des utilisateurs", "Gestion des équipes".

Utilisateurs

Cette page permettra de réaliser les actions suivantes :

- Création, suppression, modification, recherche d'un compte utilisateur
- Modification du rôle d'un utilisateur

Persistance

Utilisation des tables "Users" et "Authorities". Pas d'impact.

Service

Le service "AdministrationService" doit être enrichi des signatures et implémentations suivantes :

- User save(User) : création ou modification d'un utilisateur, renvoie null si création ou mise à jour impossible.
- User getUserByUsername(String username) : recherche d'un utilisateur par son login, renvoie l'utilisateur ou null si n'existe pas.
- boolean deleteUser(User) : suppression d'un utilisateur, renvoie vrai si suppression effectuée faux sinon.
- List<User> getAllUsers() : renvoie une liste des utilisateurs.
- List<Role> getUserRole(List<User>) : renvoie les rôles des utilisateurs passés en paramètres, le rôle est à null si l'utilisateur n'a pas de rôle.
- List<Team> getUserTeam(List<Team>) : renvoie les équipes des utilisateurs passés en paramètres, l'équipe est à null si l'utilisateur n'a pas d'équipe.
- void setUserRole(User, RoleEnum) : met à jour dans la table Authorities(username, role) les valeurs passées en paramètres - insertion ou mise à jour sur la clef username.

Il faut créer une énumération "RoleEnum" avec les 3 rôles disponibles dans l'application : ROLE_USER, ROLE_ADMIN, ROLE_DECIDOR.

Présentation

En cliquant sur la page "User management" on doit arriver sur une page contenant un tableau avec les utilisateurs existants. Le format du tableau est défini ci-dessous.

Username	**Password**	**Enabled**	**Role**	**Team**
user	user	1	ROLE_USER	marketing

Les champs username, password doivent être des formulaires dans lesquels l'utilisateur peut modifier les valeurs.
Les champs enabled et role doivent être des listes de sélections :

- Enabled : 0,1
- Role: contenu de l'enum RoleEnum

Sur chaque ligne, on doit avoir un bouton "Save" permettant de sauvegarder un utililsateur, "Delete" permettant de supprimer un utilisateur.

On doit également avoir un bouton “Add user” qui permet d’ajouter une ligne au tableau et de remplir les informations nécessaires. Par défaut, Le champ enabled doit être à 1, team doit être vide, le rôle sur ROLE_USER.

Equipes

Cette page permettra de :

- Création, suppression, modification, recherche des équipes
- Modification de l'équipe d'un utilisateur

Persistance

Utilisation de la table "Users" et "Team". Pas d'impact.

Service

Le service AdministrationService doit être enrichir des méthodes suivantes :

- Team save(Team) : création ou modification d'une équipe, renvoie null si création ou mise à jour impossible.
- Team getTeamByTeamId(Long teamId) : recherche d'une équipe par son id, renvoie l'équipe ou null si n'existe pas.
- boolean deleteTeam(Team) : suppression d'une équipe, renvoie vrai si suppression effectuée faux sinon.
- void setUserTeam(User, Team) : met à jour dans la table User avec les valeurs passées en paramètres - insertion ou mise à jour de la fk teamId dans User sur la clef username.
- Integer getNumberOfMembers(Team) : renvoie le nombre d'utilisateurs appartenant à une équipe.

Présentation

En cliquant sur "Team management", la page affichée doit contenir un tableaux avec les équipes existantes, le format est le suivant :

Team Id	**Description**	**Number of members**
1	marketing	2
2	IT	0

Description doit être un formulaire modifiable par l'utilisateur.

Pour chaque ligne, un bouton "Save Team", "Delete Team" et "See members" doit être disponible. Un bouton "Add Team" doit permettre d'ajouter une ligne au tableau avec le champ teamId à vide non modifiable, description un formulaire modifiable non rempli, et number of members à 0.

"Save team" doit sauvegarder (appel save service) la ligne en cours.

"Delete team" mettre à null les clefs référentielles de Team dans User puis supprimer les équipes de la base.

“See members” doit

- Charger la liste des utililsateurs appartenant à l’équipe. Un nouveau tableau doit apparaître :

Username
user
user2

A côté de chaque ligne, on doit avoir un bouton “Remove” permettant de supprimer un utilisateur d’une équipe.

- Afficher dans un formulaire de sélection tous les utilisateurs avec un bouton “Add to Team” disponible à côté de ce formulaire. Le clic sur ce bouton doit ajouter l’utilisateur à l’équipe en cours, éventuellement le supprimer de l’ancienne.

Accès à la fiche client

Cette fonctionnalité doit permettre à un utilisateur d'accéder à une fiche client contenant toutes les informations clientes depuis n'importe quelle page. Elle sera accessible par un lien rapide sur le header de l'application, le contenu sera chargé sur la partie droite de l'application. Cela permettra de rechercher un client par numéro, nom dans un premier temps. Cette fonctionnalité est semblable à un annuaire simplifié.

Persistance
Aucune modification n'est nécessaire, on utilisera la table existante "Customer".

Service
Un service "CustomerService" doit être crée. On utilisera une méthode ensembliste et générique permettant une évolution rapide par la suite de la fonction de recherche. On se basera sur le framework Hibernate respectant le standard JPA pour faire une recherche dynamique par champ. Ainsi, avec une seule méthode, on assurera une recherche possible sur tous les champs de la table Customer.

- List<Customer> getCustomerBy(String column, Object value) : renvoie le(s) client(s) dont le champ "column" à la valeur "value". On se limite pour l'instant à une recherche sur un seul critère (on ne passe pas une Map<String, Object> de column/value).

En refactoring, on cherchera à simplifier le service en ayant une seule méthode List<Customer> getCustomerByPattern(String pattern). Cette méthode lancera une recherche en like sur les champs de la table Customer. .

Présentation
Un bouton "Search" doit être disponible dans le header, en cliquant dessus, on doit lancer la recherche et afficher dans une pop-up un tableau les résultats comme présenté ci-dessous.

Customer ID	Name	Surname	Home phone	Mobile phone	Address

Aucun champ ne doit être modifiable. Le champ customerId doit être un lien cliquable qui lors du clic, nous dirige vers une page contenant tous les champs de la table Customer listés simplement en colonne / valeur, une colonne par ligne affichée.
On préférera l'utilisation d'un tableau de bordure 0 ou de formattage simple afin d'aligner les champs et faciliter la lecture.

Exemple :
CustomerID: 123
Name:Test etc

Reporting

Cette fonctionnalité doit permettre à un utilisateur d'accéder aux informations entrantes et générées par notre application. Pour répondre à ce besoin, la brique de reporting est nécessaire.

Persistance
Aucune modification n'est nécessaire, on utililsera les tables existantes. Cette fonctionnalité consiste à rendre l'information visibile des utilisateurs.

Les tables concernées sont :
- Reporting événéments : Event_Type, Event
- Reporting action : Acton, Task
- Reporting workflow : Task, Team, User

Service
Dans un premier temps, le service ne sera pas générique pour les événements et les actions.

On aura donc les services suivants dans l'interface IReportingService :
- Evénement
    - Map<Date, Integer> getNumberOfEventsByDays() : renvoie par date le nombre d'événements générés. Le périmètre de date est la date du jour - 7 jours à la date du jour sur la table Event.
    - Map<String, Integer> getEventsRepartition() : renvoie pour chaque description d'événements son nombre d'occurences dans la tables Event. Le périmètre de date est la date du jour - 7 jours à la date du jour. La description de l'événement est récupérée dans la table EventType.

- Action
    - Map<Date, Integer> getNumberOfActionsByDays() : renvoie par date le nombre d'actions générées. Le périmètre de date est la date du jour - 7 jours à la date du jour sur la table Action.
    - Map<String, Integer> getActionsRepartition() : renvoie pour chaque description d'action son nombre d'occurences dans la tables Task. Le périmètre de date est la date du jour - 7 jours à la date du jour. La description de l'action est récupérée dans la table Action

    .
- Workflow
    - Map<Date, Integer> getNumberOfTreatedActionsByDays() : renvoie par date le nombre d'actions traitées. Le périmètre de date est la date du jour - 7 jours à la date du jour sur la table Task. Une tâche est considérée traité quand son champ performedDate est renseigné par une date.
    - Map<String, Integer> getActionsTreatedByUser() : renvoie pour chaque utililsateur le nombre de tâches qu'il a traité. Le périmètre de date est la date du jour - 7 jours à la date du jour. On imaginera dans une seconde version fournir par utilisateur le nombre de tâches affectées. On poura également étendre la remontée d'informations au niveau de l'équipe.

Présentation
L'accès aux données sera possible par le menu "Reporting".

Le lien "Events" doit nous afficher dynamiquement (ie AJAX) une page contenant deux onglets :

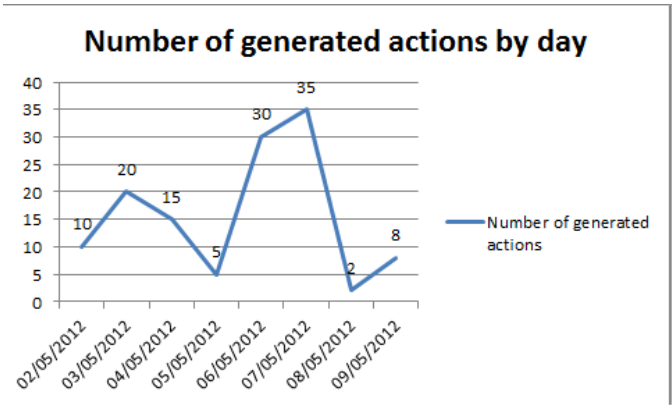
- Le nombre d'événements générées pour chaque jour. Les données doivent être présentées dans un graphique de type line chart.
- Les types d'événements générée par types qui sont les descriptions des types d'événements. La présentation des données se fera dans un graphique de type pie charts.

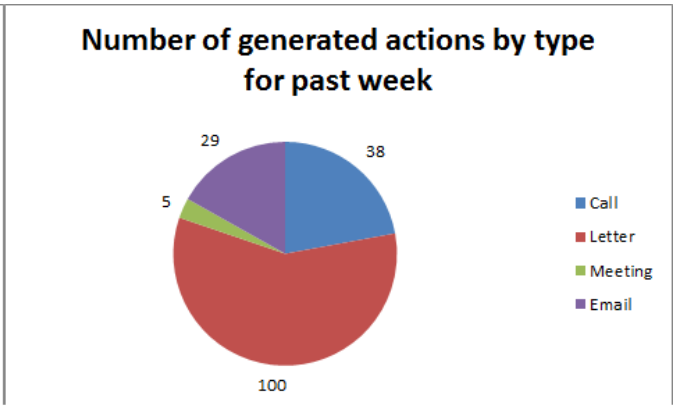
De la même façon,le lien "Action" doit nous afficher dynamiquement (ie AJAX) une page contenant deux onglets :

- Le nombre d'actions générées pour chaque jour. Le périmètre de date est la date du jour - 7 jours à la date du jour. Les données doivent être présentées dans un graphique de type line chart.
- Les actions générées par types qui sont les descriptions des actions. La présentation des données se fera dans un graphique de type pie chart.

Le framework web utilisé pour l'affichage des données est jqplot dont l'intégration et le prototypage ont étés validés.

Les exemples des deux types de reporting attendus sont présentés ci-dessous. La charte graphique est à titre indicatif et les possibilités de représentations seront conditionnées par le framework utilisé. La logique reste la même pour la présentation des événements ou des actions.





Les étapes de développement conseillées pour cette fonctionnalité sont les suivantes. Le développement du service et de la partir présentation peuvent se faire indépendamment en sécurisant les développement service par des tests sur les DAOs, service et utilitaires. La partie présentation est quant à elle sécurisée par des tests fonctionnels réalisés à l'aide de Selenium. La partie service a néanmoins tout intérêt à être développée dans un premier afin de limiter les impacts en couche de présentation si des modifications s'avèrent nécessaires. Des mises à jours fréquentes sur le gestionnaire de code source sont incitées à chaque validation de développement (ie: maven verify permettant de valider la non-régression des parties de code testées).

Pour la partie service (réalisable en TDD) :

1. Développer chaque méthode de service avec les classes de DAOs et utiltaires nécessaires. On utilisera uniquement des interfaces qui seront mockées dans les tests.
   La première étape est donc d'avoir un service fonctionnel en isolation, la validation du fonctionnement est réalisée par les tests
2. Développer chaque DAOs ou classe utilitaire utilisée dans le service. Tester chaque composant en isolation.

Pour la partie présentation :

1. Créer le controlleur ReportingController pour prendre en charge les clics des menus. Afficher une page statique contenu un titre pour valider l'affichage de la page.
2. Afficher via le framework jqplot un graphique de type line chart sans données.
3. A partir d'un bouton refresh sur la page de reporting, envoyer depuis le controlleur un tableau de données en dur correspondant à des données qui seront récupérées depuis le service.
4. Valider l'affichage des données en page de reporting en AJAX.
5. Modifier le controlleur pour envoyer les données du service et non des données en dur.

Un passage de l'outil d'audit de code Sonar est fortement conseillée à la fin des développements.

Personnalisation client

Cette fonctionnalité doit permettre aux utililsateurs de personnaliser des données clients. Deux périmètres sont à différencier :
1. Les données standards clients (nom, prénom, adresse, etc) sur lesquelles CRM365 n'est pas maître, c'est à dire que ces données ne peuvent uniquement être mises à jour depuis le chargement d'un fichier client afin d'éviter les désynchronisations entre différents SI.
2. Les données customisables qui vont permettre à l'utilisateur de rendre l'application plus proche de son métier en associant à un client des informations propres à son métier.

Ce développement est sur le périmètre 2 uniquement, le périmètre 1 étant déjà traité par le chargement de fichier client..

Persistance
La personnalisation devant rester générique pour chaque type de société, 100 champs seront créées dans la table Customer nommés "freeVal1" à "freeVal50".
Pour que l'utilisateur puisse rendre les données plus parlantes, on stockera dans une table un alias sur chacune de ces colonnes "freeValX".

Il faut donc :
1. Ajouter 50 champs freeVal1 à freeVal50 dans la table customer. Type VARCHAR(80)
2. Créer une table "Customer_Alias" sans lien avec d'autres contenant les champs suivants :
   - Id : id technique, bigint auto-increment
   - FreeValName
   - FreeValAlias

Cette table pourra contenir un maximum de 50 lignes, exemple :

id	freeValName	freeValAlias
1	freeVal1	Profil de risque client
2	freeVal2	Année d'ancienneté

Présentation
Deux impacts sur la partie présentation :

1. Permettre la personnalisation des champs, cela revient à ajouter une ligne dans la table "Customer_alias".

Cela sera géré dans une page "Profile customization" dans le menu "Customer". Cette page doit uniquement être accessible à l'administrateur.
L'URL sera "configuration.customer" et sera géré par le controller CustomerController. Les URLs commençant par configuration sont déjà configurées pour être accessibles uniquement par l'administrateur.

2. Permettre la saisie des valeurs : cela se fera dans la page existante de recherche client.
Aujourd'hui sur la page de détail ("customerDetail.jsp"), on a tous les champs de la table Customer.
Il faut :

- Garder les champs standards (nom, prénom) tels quels.
- Pour les colonnes freeValX, n'afficher uniquement celles qui sont configurées dans la table "Customer_Alias".
- Chaque ligne du détail ayant un champ "Customer_Alias" doit être un formulaire de saisie permettant à l'utilisateur de saisir des champs et de sauvegarder les valeurs pour le client choisi.

La page de détail client aujourd'hui :

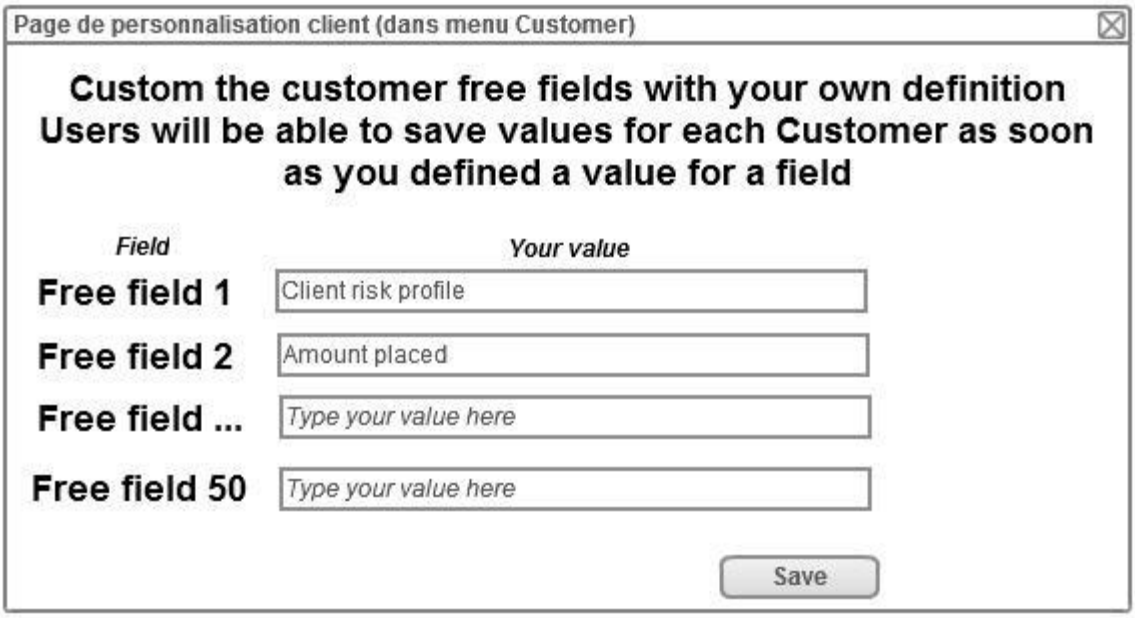
Name	Jean
Surname	Dupont
Address	X Rue du ….
Phone	03 …..

Celle à développer permettant la saisie des valeurs paramétrés. Par exemple avec 5 champs configurés par l'administrateur.

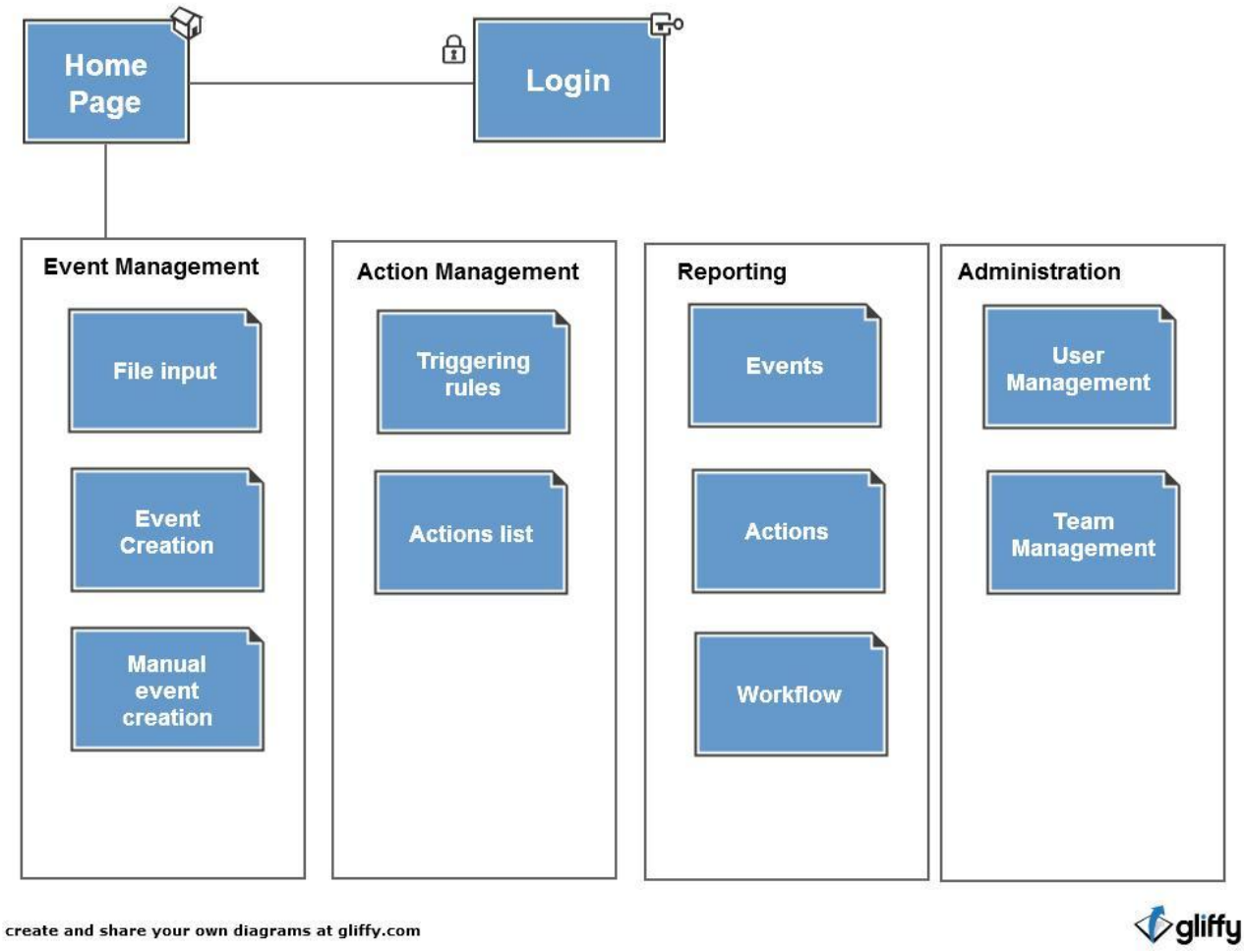
Name	Jean
Surname	Dupont
Address	X Rue du ….
Phone	03 …..
freeVal1	*Risk 1*
freeVal2	*Enter a value*
freeVal3	*Enter a value*
freeVal4	*Enter a value*
freeVal5	*Enter a value*

Le clic sur le bouton de sauvegarde doit mettre à jour les valeurs freeVal configurées dans la table Customer du client visualisé sur la fiche de détail client.

La nouvelle page et les modifications de la page de fiche client sont présentées dans les maquettes écrans ci-dessous.



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

Le déploiement sur des serveurs d'applications différents (Tomcat, JBoss, etc …) n'est pas automatique. Chaque serveur d'application implémente ou non les normes et standards et a ses spécificités.
Le déploiement sur les serveurs d'applications les plus importants du marché seront donc nécessaires.

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

Annexe

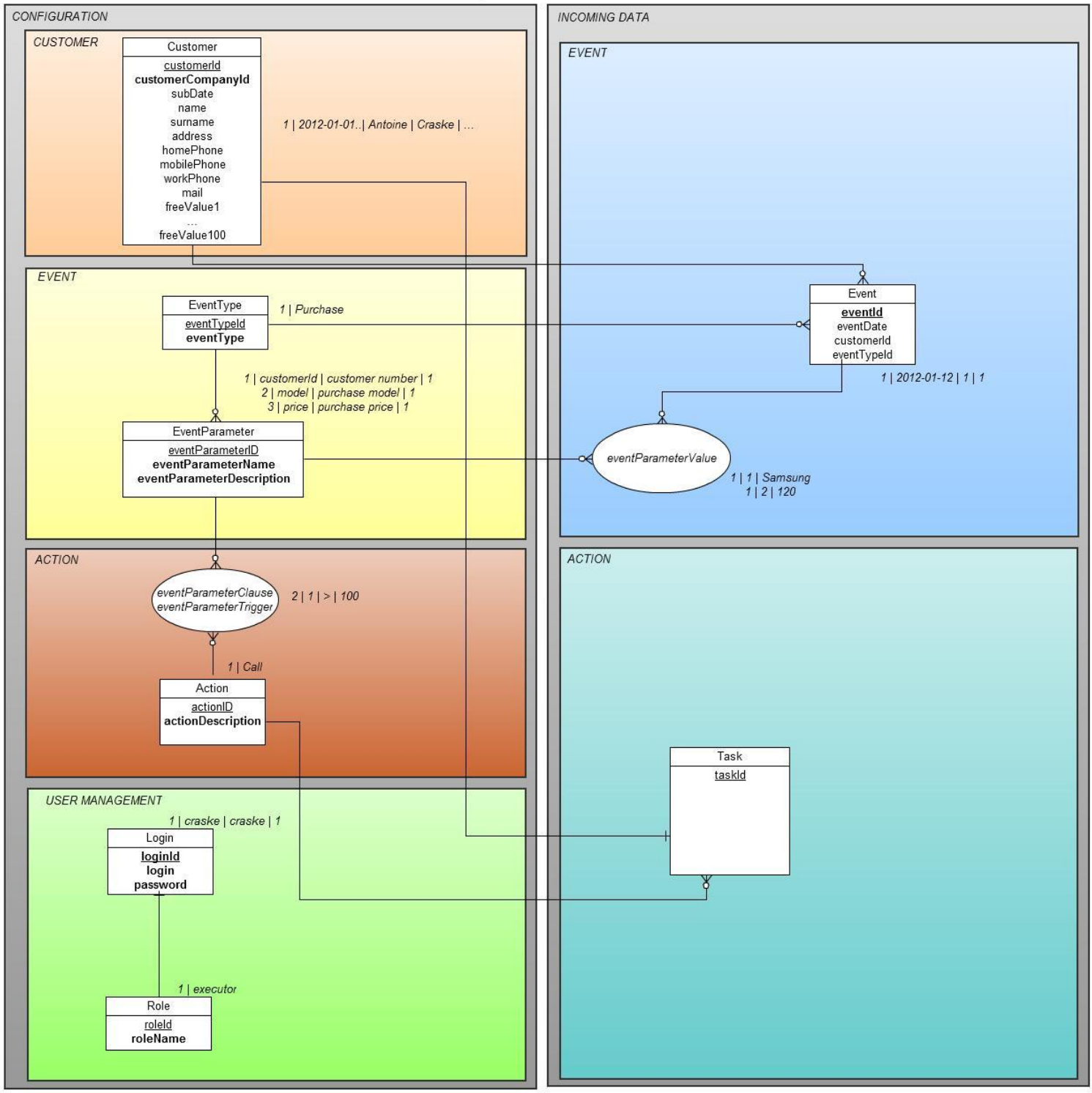
Dictionnaire de données

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 freeValue50	VARCHAR	100	champ libre réservé société

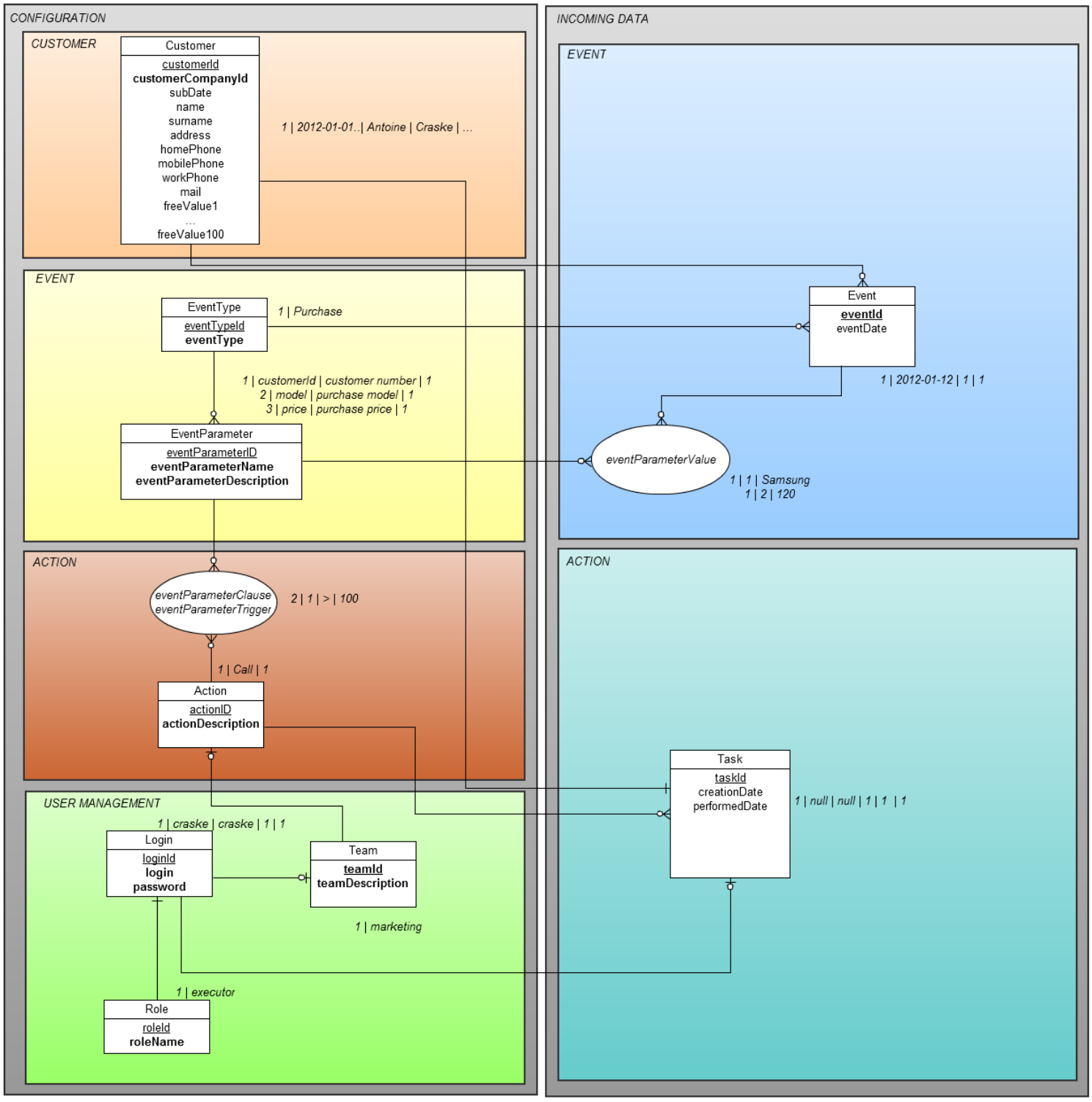
eventParameterClause	VARCHAR	50	critère de génération d'action (>, >=, etc)
eventParameterTrigger	VARCHAR	50	valeur qui déclenchera l'action si le critère appliqué à cette valeur renvoie vrai
actionID	BIGINT	19	id technique
actionDescription	VARCHAR	100	libellé de l'action
taskId	BIGINT	19	id technique
teamId	BIGINT	19	id technique
TeamDescription	VARCHAR	100	Description de l'équipe
performedDate	DATE	23	date de traitement de la tâche

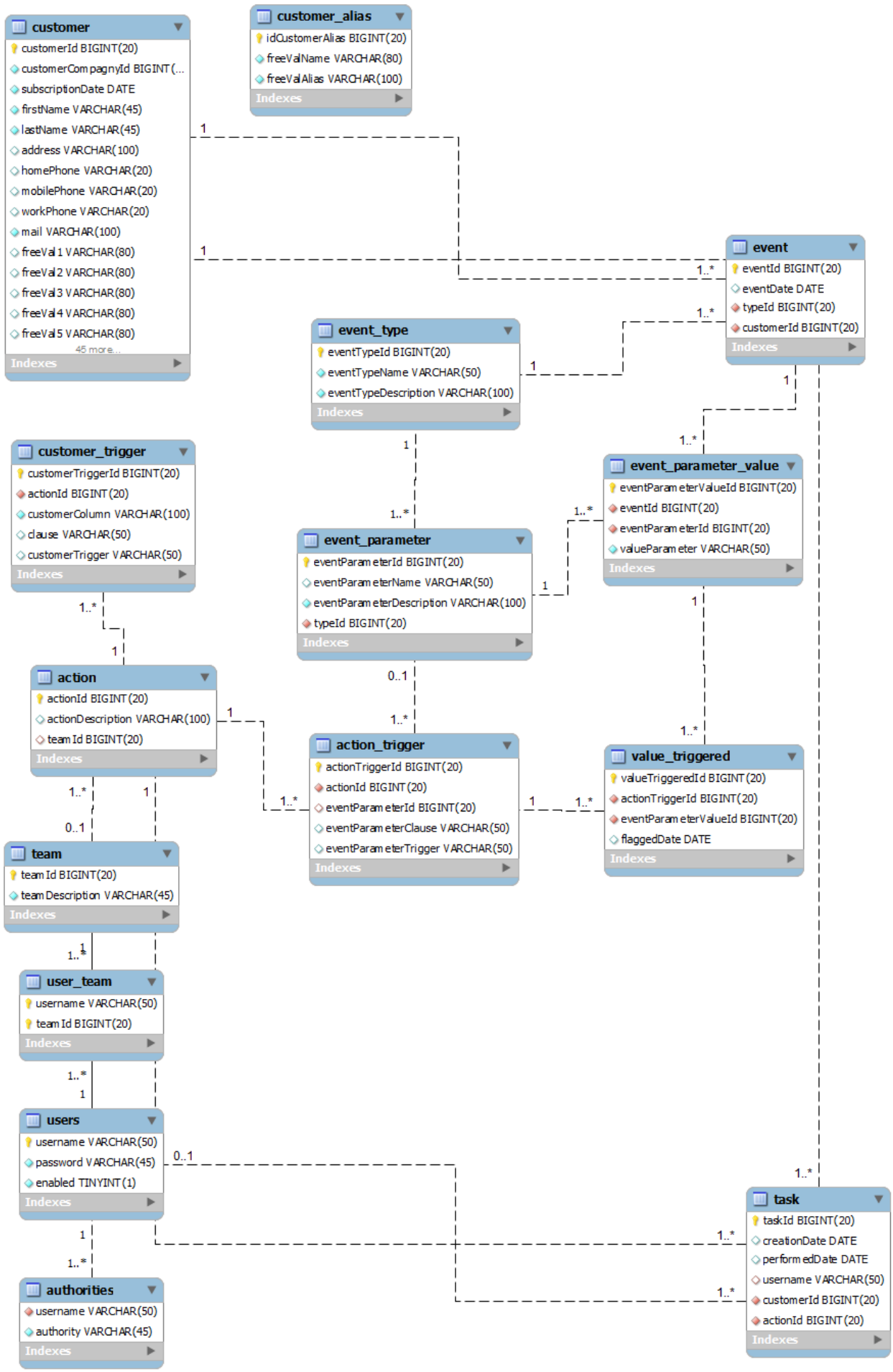
Modèle de données

Sprint 1 : Database model SPRINT 1



Database model PART 2 - SPRINT 1




customer
customerId BIGINT(20)
customerCompagnyId BIGINT(...
subscriptionDate DATE
firstName VARCHAR(45)
lastName VARCHAR(45)
address VARCHAR(100)
homePhone VARCHAR(20)
mobilePhone VARCHAR(20)
workPhone VARCHAR(20)
mail VARCHAR(100)
freeVal1 VARCHAR(80)
freeVal2 VARCHAR(80)
freeVal3 VARCHAR(80)
freeVal4 VARCHAR(80)
freeVal5 VARCHAR(80)
45 more...
Indexes
customer_alias
idCustomerAlias BIGINT(20)
freeValName VARCHAR(80)
freeValAlias VARCHAR(100)
Indexes
event
eventId BIGINT(20)
eventDate DATE
typeId BIGINT(20)
customerId BIGINT(20)
Indexes
event_type
eventTypeId BIGINT(20)
eventTypeName VARCHAR(50)
eventTypeDescription VARCHAR(100)
Indexes
customer_trigger
customerTriggerId BIGINT(20)
actionId BIGINT(20)
customerColumn VARCHAR(100)
clause VARCHAR(50)
customerTrigger VARCHAR(50)
Indexes
event_parameter_value
eventParameterValueId BIGINT(20)
eventId BIGINT(20)
eventParameterId BIGINT(20)
valueParameter VARCHAR(50)
Indexes
event_parameter
eventParameterId BIGINT(20)
eventParameterName VARCHAR(50)
eventParameterDescription VARCHAR(100)
typeId BIGINT(20)
Indexes
action
actionId BIGINT(20)
actionDescription VARCHAR(100)
teamId BIGINT(20)
Indexes
action_trigger
actionTriggerId BIGINT(20)
actionId BIGINT(20)
eventParameterId BIGINT(20)
eventParameterClause VARCHAR(50)
eventParameterTrigger VARCHAR(50)
Indexes
value_triggered
valueTriggeredId BIGINT(20)
actionTriggerId BIGINT(20)
eventParameterValueId BIGINT(20)
flaggedDate DATE
Indexes
team
teamId BIGINT(20)
teamDescription VARCHAR(45)
Indexes
user_team
username VARCHAR(50)
teamId BIGINT(20)
Indexes
users
username VARCHAR(50)
password VARCHAR(45)
enabled TINYINT(1)
Indexes
authorities
username VARCHAR(50)
authority VARCHAR(45)
Indexes
task
taskId BIGINT(20)
creationDate DATE
performedDate DATE
username VARCHAR(50)
customerId BIGINT(20)
actionId BIGINT(20)
Indexes