Page de garde

Résumé

Remerciements

Liste des figures

Liste des abréviations

Liste des tableaux

Sommaire

# Introduction

# Partie 1

# Présentation du stage

## Sommaire

## Société Coheris

### Présentation

### Produits

### Organisation

### En chiffres

## Mission

### Sujet de stage

### Besoin

### Équipe

### Environnement de travail

### Planning prévisionnel

# Partie 2

# Travaux effectués

## Sommaire

1. **Etude de mise en œuvre de l’approche Domain Driven disign**

Les travaux réalisés dans cette partie sont les plus intéressants par rapport aux autres tâches qui m’ont été confié pendant le stage. En effet, les conceptions réalisées la fin de cette partie vont être appliqué sur les différents modules développés par la suite.

* 1. **Objectifs**

Afin d’avoir une démarche unifiée de construction des modules et prendre en compte la complexité croissante dans la future, il faudra adopter une approche de conception de logiciel plus agile, modulaire et simple à évoluer. L’équipe R&D CRM ont mis d’accord d’adopter l’approche Domain Driven Disign à travers le design pattern « CQRS-Event Sourcing » pour répondre aux besoins mentionnées. Le premier travail qui m’a ´été attribué est de faire la conception à partir d’un module exemplaire en appliquant l’approche DDD en prenant compte différentes type de déploiement. Le déploiement de module peut être en:

* Mode 1 : le Frontend et le Backend dans le meme serveur
* Mode 2 : le Frontend et le Backend sur deux différents serveurs
* Mode 3 : Backend en mode API REST

Pour que le travail soit bien cadrer, une spécification fonctionnelle a été réalisée par le Product Manager décrit les fonctionnalités attendus du module.

L’objectif de cette première étape d’une durée de 2 semaines est multiple : la documentation de la conception de déférents types de déploiement et bien sûr comprendre le fonctionnement et comment mettre en œuvre le design pattern « CQRS-Event Sourcing » en respectant les modes de déploiement.

1. **Etude et Réalisation**

L’architecture est multicouche (N-Layered) et assure les concepts d’inversion de contrôle et d’injection de dépendances. Selon les besoins mentionnées par rapport aux modes de déploiements, le module va être divisé en 7 couches. Chaque couche est encapsuler dans un « .jar » à part.

* + 1. ***Modes de déploiements du module Coheris***

Pour répondre aux mieux aux besoins des clients, le déploiement du module change pourra changer

Du module peut changer le module peut être déployé

#### Mode 1

Dans ce mode de déploiement, le module est encapsulé dans un seul serveur. Un d’utilisation de ce mode

Shéma1

#### Mode 2

Shéma 2

#### Mode 3

Shéma 3

#### Les différentes Couches du module

Application : C’est la couche qui expose les services exploitable du module par la couche api ou les contrôleurs (managed bean) des composants JSF de la couche Gui. Ces services sont des implémentations des interfaces définies dans la partie shared-kernel. L’interaction se fait soit par des commandes pour un traitement d’écriture, soit par des queries pour les traitements de lecture via un composant appelé finder. Ce dernier, n’est pas besoin de traversé les autres couches pour appliquer son traitement de lecture. Il cherche directement ses résultats par des requêtes criteria sur le modèle de données. En fin, et pour que les autres couche peuvent se communiquer entre eux via des événements, des listeners sont mis en place pour Handler les évènements qui circule dans le module.

Domain : c’est le centre du module. Cette couche comporte toutes les classes correspondant aux éléments du modèle du domaine. Il encapsule les entités et le logique métier du module. Aussi, ici on trouve les définitions des repositories qui sont

Infrastructure

Shared-kernel

Api : Une API Restful pour la partie API de l’architecture doit être mise en place mis en place pour pouvoir exploiter le module comme étant une API service.

Api-client :

Gui :

Découplage entre le couche domain qui contient et infrastructure par la mise en œuvre du pattern Event Sourcing. En effet, la couche application est la chef d’orchestre entre les autres couches. Elle recoit les commande ou les queries.

Un diagramme qui explique les dépendances entre le module core et n modules après

1. **Bilan**

L’objectif de cette première étape a ´été tenu et a constitué une excellente introduction au sujet principal de mon stage. La conception qui a été réalisé va me servir à organiser le développement selon les différentes couches pour chaque module. En effet, afin de réaliser dans de bonnes conditions les développements des modules avec la nouvelle architecture, il était indispensable que je sois formé sur l’outil de build gradle pour écrire taches de build avec un langage de scripts Groovy.

1. **Module Scheduler**
   1. **Objectifs**

Afin de planifiés des traitements souvent longs dans les différentes modules de la nouvelle offre, on a besoin d’un module Scheduler. C’est un gestionnaire de taches, paramétrable et intégrable selon le besoin. En effet, le module social a besoin de planifié des taches de fond qui collecte les posts et les commentaires Facebook dans les différentes Fanpage pour les qualifié après. L’objectif de ce travail planifié sur 2 mois est de maitriser la nouvelle architecture, le développement d’un prototype complet qui peut être déployé sur les différents modes.

Une étude complète et détaillée doit être réalisée en prenant soin d’effectuer des comparatifs avec la solution existante des Framework qui gère les un système de planification, de mettre en valeur les points jugés intéressants et les éventuelles régressions. De plus, un certain nombre de points techniques doivent être vérifiés.

La mission a été d´découpée en plusieurs tâches. L’étude a été réalisée sur 1 semaines, l’installation et la manipulation la nouvelle architecture a également dure 1 semaines, enfin le développement d’un prototype a été accompli en 2 étapes de 3 semaines.

* 1. **Etude détaillée**

Etude du choix du Framework

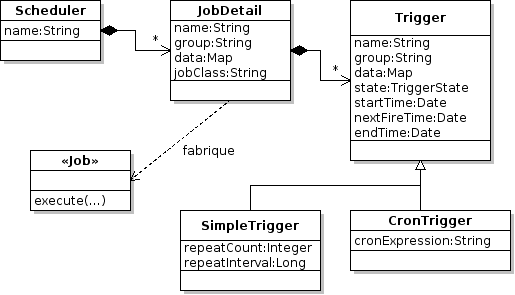
### Depuis la Java Entreprise Edition 7, l'API Concurrency Utilities1.0 (JSR 236) de Java propose un système de planification de tâches riches et simple à utiliser. On peut spécifier des annotations pour définir une tache avec l’expression cron []. En revanche, Il existe plusieurs schedulers disponibles la plateforme J2EE qui sont plus évolué et qui sont opensource. Quartz est un projet terracota qui qui propose des composants orientés entreprise JAVA. Mon rôle a donc a été de réaliser une veille technologique afin d’acquérir des informations techniques et d’établir la liste des points forts et faibles pour chaque API [].

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Framework | documentation | Communauté autour | Clusturing | shema de donnée | configuration | Standard JEE |
| **Concurrency Utilities 1.0** | Peu de documumentation:  documentation standard | Encore nouveau pas beaucoup | Non | non | Pas de configuration | oui |
| Quartz Scheduler | Documumentation riche : exemples, tutoriaux… | Beaucoup plus | OUI | Inclus et gérer automatiquement | Un fichier .properties | Non |

Le choix a été sur Quartz Scheduler vu qu’il gére lui-même sa base de données des taches et il supporte le mode Clusturing qui est une contrainte impérative pour les modules nouvelles offre de Coheris. A noter, Quartz ne gére pas l’historique des taches qui a été exécuter mais il fournit un listner qui va etre servie pour sauvgarder l’historique d’exécution pour chaque tache.

Architecture

En se basant sur le DDD, il faudra trouver une approche pour branché l’API Quartz dans cette architecture. En effet, l’api se base sur la une structure de donnés suivante :



* **Trigger:** le *quand exécuter*, ça décrit une ou série de moments, il y a plusieurs variantes: Cron, périodique, one shot...
* **Job:** le *quoi exécuter*, c'est un bout de code
* **Job Detail:** l'instance de Job, elle a un ou plusieurs triggers et éventuellement des paramètres
* **Job Group:** un ensemble de Job Details ou de Triggers que l'on manipule ensemble: arrêt, démarrage, annulation...
* **Scheduler:** le moteur chargé de la planification et de l'exécution des tâches planifiées.

Documentation de l’API Restful du module

* 1. **Réalisation**

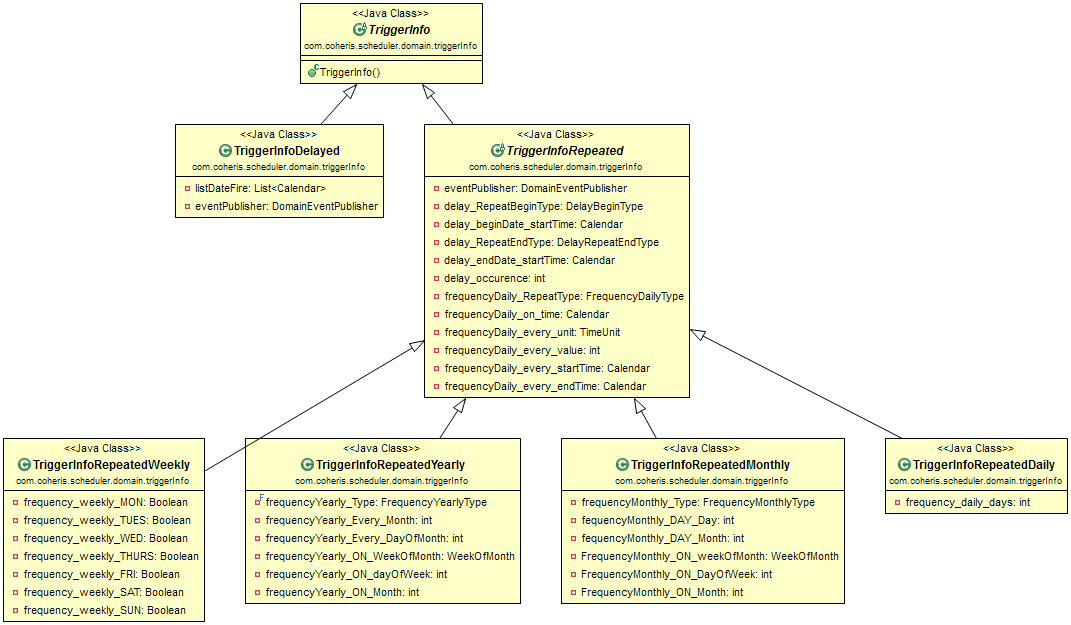
Le travail d’´etude a ´et´e coupl´e `a une phase de validation technique. Cette ´etape a pour rˆole de s’assurer que des points pr´ecis peuvent ˆetre r´ealis´es facilement, soit car int´egr´es en standard dans le produit, soit car on peut ais´ement les mettre en place grˆace `a l’API fournie

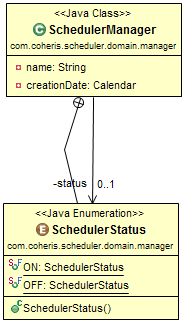
Selon les spécifications des fonctionnalités du module, il doit permettre de planifier des taches : quotidienne, hebdomadaire, mensuelle et annuelle. Aussi, on peut définir un traitement qui pourra etre executer par plusieurs tache planifier. Finalement, on peut consulter l’historique de chaque tache exécuté et son état d’exécution.

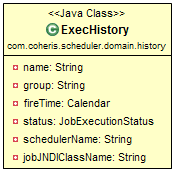
Cela ramène à définir 5 modèle suivant l’architecture DDD, qui doit être exposés dans la partie Domain du module :

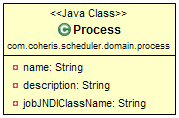
* **JobInfo :** l'entité qui contient les informations nécessaire pour créer un job
* **TriggerInfo :** l'entité qui collecte les informations nécessaire pour créer un Trigger quartz
* **SchedulerManager:** l'entitité qui représente l'entité scheduler du quartz
* **ExecHistory :** l'entité qui contienne les informations catchées du listener du quartz sur les exécutions
* **Process :** l'entité qui représente les infos nécessaire pour le traitement à exécuter











Le branchement du quartz doit être coté Infrastructure pour qu’on puisse l’exploiter comme étant une Base de données externe. Pour La partie configuration de l’API quartz a été externalisée d’une façon programmatique à partir des paramètres définie dans le fichier standalone.xml du serveur.

Les composants IHM

Enfin, pour tester le prototype du module, il faut à réaliser une Interface Homme Machine pour qu’on puisse visualiser les taches. J’ai été mené à désigner des maquettes, qui ont été validé par le Product Manager, qui exploite les différentes fonctionnalités développer dans le module. Il m’a été demandé de se baser d’une part sur le Framework Bootstrap pour que les composant soit résponsives, et d’autre part sur Primefaces qui propose une bibliothèque riche de composnant qui se base sur JSF facile à intégrer.

Tableau de bord : où on peut trouver d’une part, les taches en cours d’exécution et son état de chaque d’autre part une vue global sur l’état du scheduler par des statistiques faite par un diagramme.

Gestionnaire de taches : C’est dans cet vue on peut ajouter des taches selon un formulaire dynamique. Et Aussi on trouve plus des détails sur chaque tache.

Gestionnaire des traitements : C’est dans cet vue on peut ajouter des traitements selon un formulaire dynamique.

Gestionnaire d’Historiques : où se trouve l’historique de tous les taches qui ont été exécuter

Conclusion

* 1. **Bilan**

1. **Desktop Application**
   1. **Objectifs**
   2. **Etude de l’existant**
   3. **Réalisation**
   4. **Bilan**
2. **Module Social et l’Intégration SPAD RealTime**
   1. **Objectifs**
   2. **Etude de l’existant**
   3. **Réalisation**
   4. **Bilan**

# Partie 3

# Bilan

## Sommaire

## Résultats obtenus

### Mise en œuvre de l’approche Domain-driven design

### Planificateur de tache de hautes disponibilités

### Refonte de l’IHM du module social existant

### Intégration de la Textmining dans le module social

## Difficultés rencontrées

### Compilation des modules découplés

### Tester l’efficacité de l’algorithme de Textmining

## Perspectives

* 1. **Intégration de SPAD Real Time dans le module mail**
  2. **Portabilité des composants IHM**

Conclusion

Ressources

Glossaire

Annexes

# Planning

# Architecture d’un module avec Domain Driven Disign, Event Sourcing et CQRS