# ShiftPlanning
# Vision
# Version 2.0

# Historique des révisions

| Date | Version | Description |
| --- | --- | --- |
| < 24/05/2016> | <1.0> | Le choix de l'architecture |
| < 13/06/2016> | <2.0> | Le choix de l'architecture |
| | | |
| | | |

# Table des matières

# Architecture du Logiciel

## 1. Introduction

Le but du présent document est de décrire la structure du logiciel **ShitPlanning** en termes d'éléments et les relations qui existent entre eux et de garantir que la structure conçue répond aux besoins du système et que l'architecture en conséquence est maintenable et extensible. Ce document est destiné aux membres de l'équipe, aux architectes, aux concepteurs, et aux superviseurs du projet.

Les Documents référencés sont :

- Document de vision 1.0
- Glossaire 1.0
- Spécifications supplémentaires 1.0
- Modèle des cas d'utilisation 1.0

## 2. Objectif du logiciel

L'objectif est de proposer un logiciel de gestion de projets adaptés au contexte multinational des organismes tels que les groupes industriels ou les ONG internationaux. Ce logiciel permet de planifier et suivre l'avancement des activités d'un projet, de leurs affecté des ressources humaines et matériels, en tenant compte des caractéristiques local et préférences culturelles des acteurs d'un projet.
Le Reporting est particulièrement important. Une assistance doit être apportée au planning des séances de travail impliquant certain membres de projet.

### 2.1 Contexte

Ce projet vient à l'issu de la mauvaise organisation entre les groupes de travail et son impact sur la progression des projets , mais aussi pour remédier au manque remarqué sur les applications permettant la gestion des planning d'un groupe de travail .

## 2.2 Besoins non fonctionnels

L'application **ShiftPlanning**à pour qualités

- **simplicité d'utilisation :** le système doit être simple d'utilisation et nécessiter un temps d'adaptation court pour un débutant.
- **Fiabilité et performance :** le temps de réponse doit être minimal.

L'application **ShiftPlanning**à pour contraintes :

- **Portabilité :** l'application à réaliser étant une application répartie, il est primordiale, pour garantir la portabilité, de prendre en considération lors de la conception l'hétérogénéité des machines, tant au niveau des systèmes d'exploitation que du codage.
- **Modularité :** la modularité facilite grandement la détection des erreurs et leur correction lors de la phase de développement ainsi que la maintenance du logiciel.

# 3. Structure

Considérons un ensemble de machines dont l'informaticien est le propriétaire. Notre application adopte le principe d'architecture distribuée dans la mesure ou, en pratique, on consacre une zone partagée de mémoire de chaque unité de stockage et on la rend accessible de n'importe quel noeud appartenant au réseau. Ainsi nous aurons un bassin commun de ressources : délocalisation des données.
L'application doit gérer :

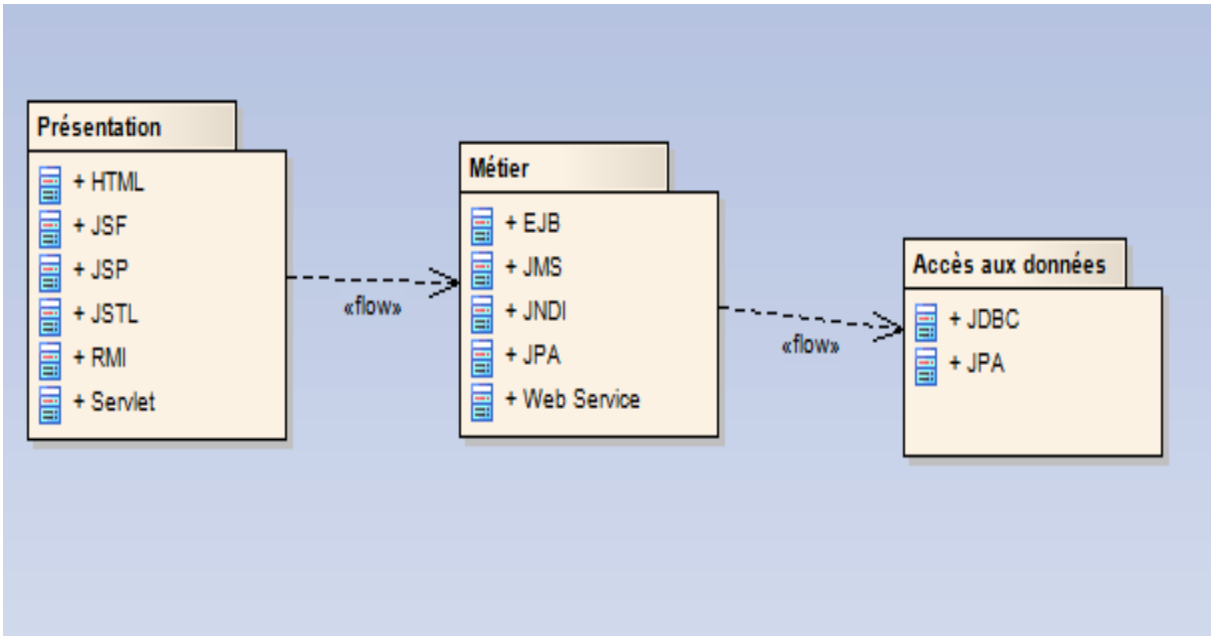
- **Virtualisation**
- **Ouverture :** les services de l'application sont mis à la disposition sur un réseau spécialisé permettant de mutualiser des ressources de stockage et utilisent des techniques standardisées qui permettent de s'en servir avec plusieurs supports de stockages
- **Hétérogénéité :** une ressource peut avoir plusieurs versions, prenons l'exemple d'un programme disponible pour plusieurs OS ou un document avec plusieurs versions telles que les versions du même document.

### 3.1 Vue des couches :

L'architecture en trois tiers est une architecture de programmation qui propose de séparer une application en trois parties :

- La présentation qui s'occupe de l'affichage sur le poste de travail des données du système et interaction avec l'utilisateur
- Les traitements métier ensemble de règles métiers de l'application
- La couche accès de données qui manipule et conserve les données.

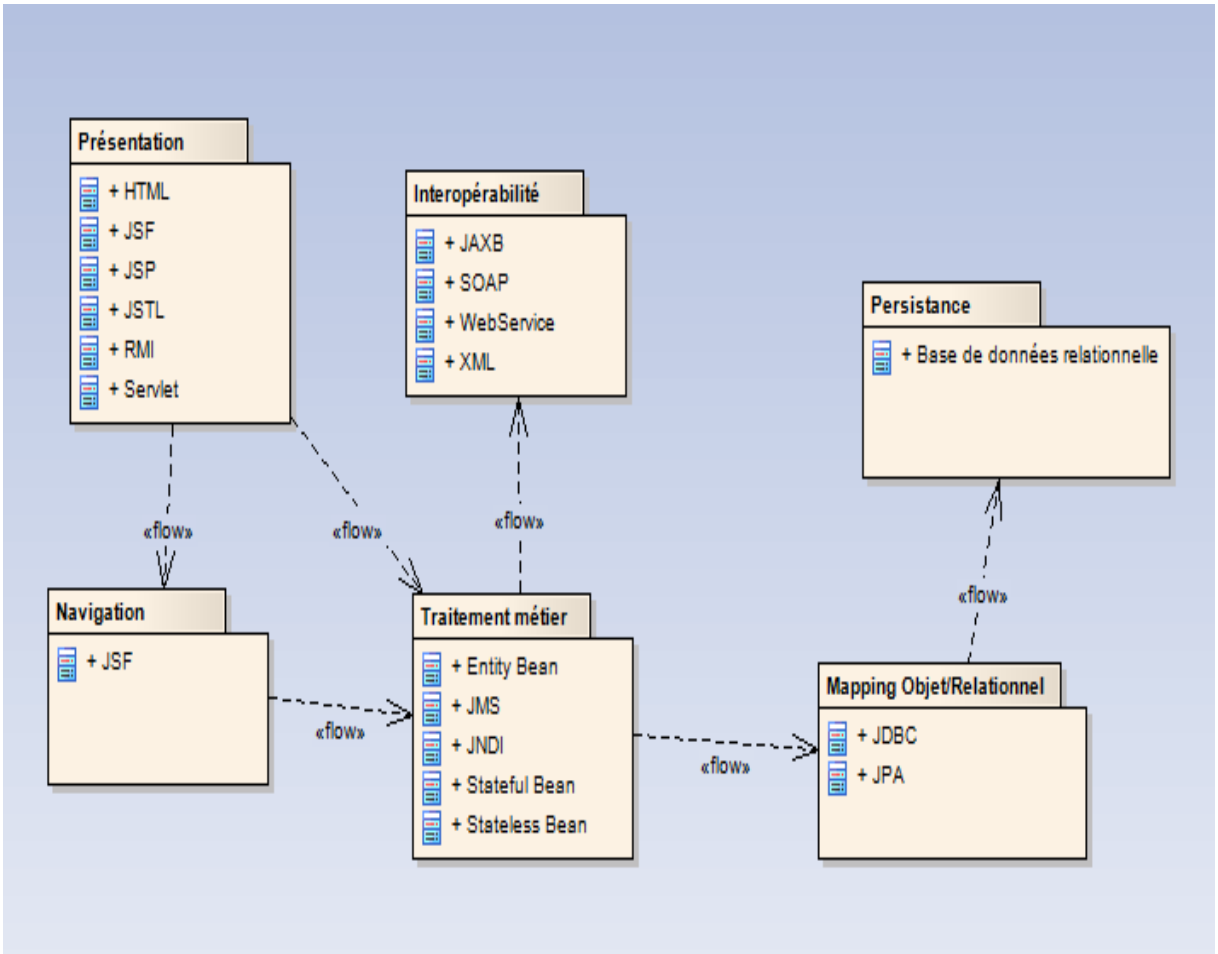
Ci-après un diagramme de paquetage UML représentant ces trois couches. Chaque sous-système contient les technologies JAVA EE qui seront utilisées dans l'application




### 3.2 Architecture applicative :

Le précèdent modèle en trois couches peut être affiné pour être plus fidèle a l'architecture finale de l'application.

Ci-après un diagramme de paquetage décrivant les couches de l'application




Couche traitement métier

En pratique on trouve au niveau de la couche métier

Des entités dont la persistance est assurée par la couche de mapping

Des stateless Beans qui proposent des méthodes pour manipuler les entités (crud)

Des message-driven beans qui assurent les traitements asynchrones

Les API JNDI pour accéder au service de nommage et javaMail pour envoyer des emails pour le consultant externe

Couche de mapping objet/relationnel

Elle transforme la représentation physique des données en une représentation objet et inversement

Ce mécanisme sera assuré par JPA qui utilise le protocole JDBC pour exécuter les appels SQL

Couche d'interopérabilité

Pour dialoguer avec les partenaires