DEUTSCH Rémi VINCENT Pierre LOIGNON Lucas

université st charles, académie aix-marseille

Rapport de projet

(PIA)

Table des matières

[I. Introduction 2](#_Toc479088652)

[II. Le sujet 2](#_Toc479088653)

[III. Organisation des taches 2](#_Toc479088654)

# Introduction

Ce rapport vise à présenter le travail réalisé pour l’UE Programmation Informatique Appliqué, sous la direction de Mme REN, par DEUTSCH Rémi, LOIGNON Lucas et VINCENT Pierre.

Nous évoquerons d’abord le sujet de notre projet et les raisons de notre choix, puis nous parlerons de l’organisation et la planification des tâches, ensuite nous expliquerons en détail l’architecture de notre projet.

# Le sujet

Nous avons imaginé un jeu de carte basé sur un système de tour par tour et de coût par carte. Nous avons décidé, pour cette UE, de mettre en place un système d’administration de base de données lié à notre concept de jeu.

En effet la mise en place de ce type de jeu implique tout d’abord une gestion et une création de base de données, il faut que les joueurs puissent créer un compte, composer leurs deck, changer les informations liées à leurs comptes bref tout autant de fonctionnalités à implémentées avant de pouvoir commencer le développement du jeu à proprement parlé.

Pour les besoins de cette UE nous avons décidés de nous limiter à la mise en place d’une base de données cohérente avec notre concept de jeu ainsi qu’au développement d’une application web pouvant administrer notre base de données.

Notre base de données doit donc pouvoir contenir différentes informations essentielles à la gestion du jeu, par exemple les decks de chaque joueurs, les comptes des joueurs ou encore les cartes que peuvent posséder les joueurs.

Pour la gestion de notre base de données nous avons choisi MySQL car c’est une solution facile d’accès, gratuite et open source.

Pour la gestion de notre serveur, nous avions besoin d’une solution permettant la communication asynchrone indépendamment des requêtes classiques du protocole http dans un souci d’évolution. En effet nous avons le projet de continuer le développement de l’API de gestion et du jeu en lui-même, pour l’implémentation du jeu nous avions besoin de ce type de communication.

Nous avons donc choisi d’utiliser JAVA EE avec le conteneur de Servlets Tomcat car les sockets de java remplissaient nos spécifications, de plus JAVA EE est un langage souvent demandé dans le milieu professionnel ce qui n’a fait que renforcer notre choix de vouloir apprendre de nouvelles technologies. Bien sûr comme toute application web nous avons également utilisé les langage HTML, CSS et Javascript pour le rendu du coté client.

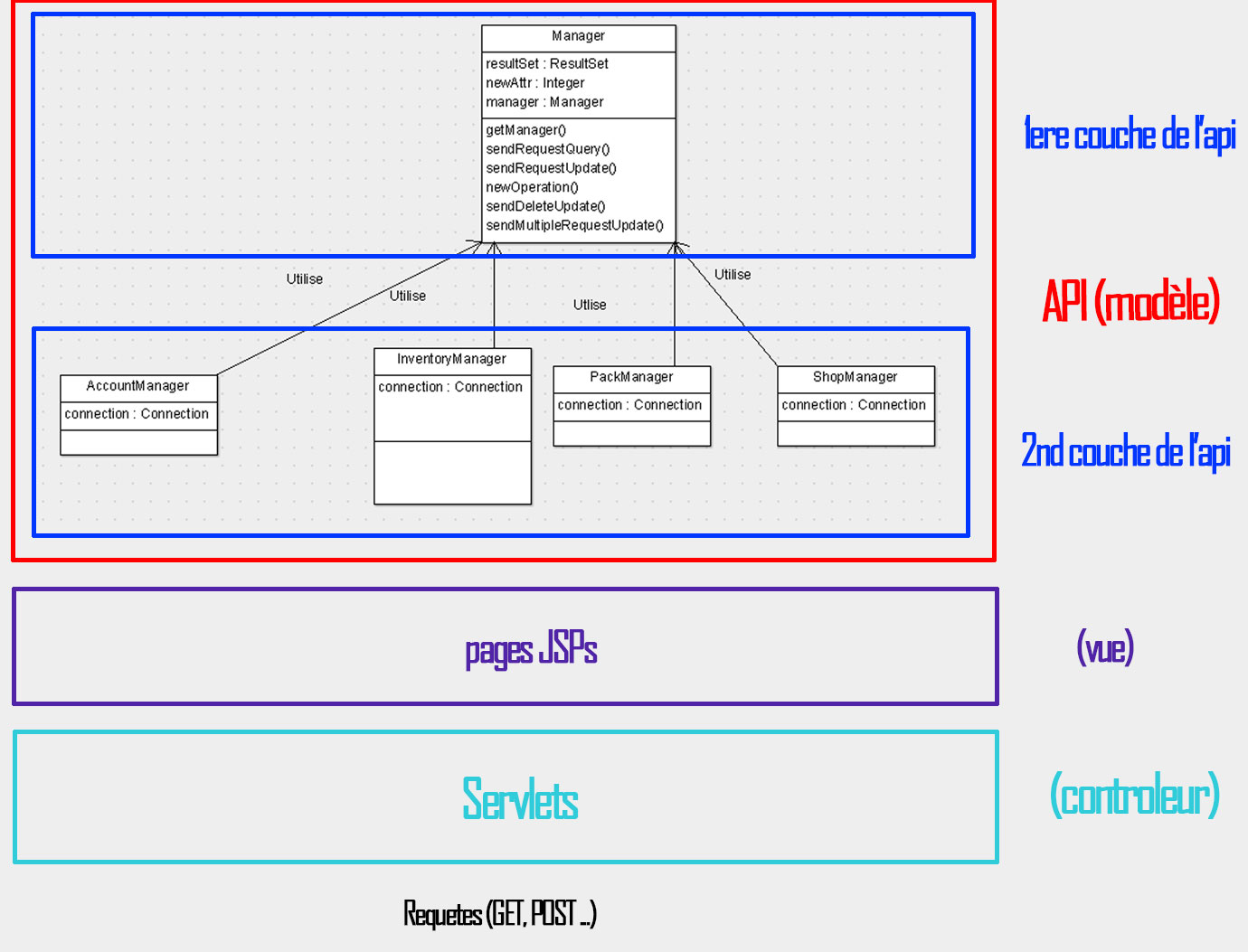
# Organisation des taches

Pour permettre une organisation des taches optimale nous avons opté pour une méthode agile : SCRUM.

Nous avons découpé les taches en plusieurs sprints de 2 semaines chacun, nous avons progressé en implémentant des « couches » en commençant par mettre en place une base de données dans les premiers temps, puis commencé l’implémentation de la première couche de l’API de gestion pour communiquer avec notre base de données puis nous avons implémenté la seconde couche de l’API qui se charge de faire les traitements des données et enfin nous avons terminé avec la mise en place de toutes les pages JSP et du CSS du coté client.

Le fichier de gestion avec la méthode SCRUM sera fournie en annexe de ce document.

# Architecture du projet

L’architecture du projet suit une approche objet avec un pattern de type MVC (Modèle Vue Contrôleur). Voici un schéma de l’architecture du serveur :

## Modèle

Le modèle est essentiellement l’API mise en place pour communiquer et administrer notre base de données, elle se charge de faire tous les traitements métiers de l’application. Cette API se décompose en deux couches.

### 1ere couche

La première couche de l’API va utiliser la classe JDBC (Java DataBase Connectivity) qui permet de dialoguer avec la base de données distante. Cette couche va implémenter des fonctions très basiques qui vont permettre d’envoyer des requêtes et de récupérer leurs retours sans traiter ces informations. La classe Manager qui implémente la première couche est un singleton, en effet nous ne voulions pas permettre la création de plusieurs connexions à la base de données.

### 2nd couche

La deuxième couche de l’API va permettre de traiter les informations récupérées avec la première couche. En effet les informations données par la première couche doivent être interprété afin de pouvoir faire un rendu graphique par exemple. Les classes qui se charge d’implémenter ce type de traitement sont dans le package nommé « Manager ».

La classe « AccountManager » va se charger de faire tous les traitements liés aux comptes joueurs dans la base de données. Par exemple créer un compte ou récupérer un email. La classe « InventoryManager » va se charger d’effectuer les traitements liés aux possessions d’un joueur (son inventaire). Par exemple récupérer son deck et créer les objets de vue qui lui sont associé. La classe « PackManager » va se charger d’effectuer les traitements liés à la création et l’administration de pack dans notre boutique. La classe « ShopManager » va se charger d’effectuer les traitements liés à la boutique.

Toutes ces classes de la 2nd couche de l’API vont utiliser un objet de type Manager à chaque fois qu’elles aurons besoin de communiquer avec la base de données.

## Vues

Afin de représenter au mieux ce qui se trouve dans notre base de données, nous avons mis en place toute une série d’objet de vue qui représente un objet de la base de données. Tous les objets de vues se trouve dans le package « View ». Par exemple la classe Carte se charge de représenter une carte, ses attributs sont remplis avec des entrés de la base de données tel que : son ID, son image, son cout, son nom, sa description etc.

La vue est assez simple, elle se contente d’afficher les objets de vue construit à partir de l’API. Cet affichage se fait via une série de page JSP (Java Server Page) qui permet un rendu dynamique d’un page HTML.

## Contrôleurs

Les contrôleurs sont les servlets, les servlets se chargent de recevoir les requêtes http puis de rediriger vers les pages JSP demandées. On peut donc, à la réception d’une requête, examiner les attributs de celle-ci afin d’appliquer un contrôle sur ce que les utilisateurs nous envoient et de rediriger la requête vers une page d’erreur si nécessaire. Les servlets sont routées grâce au fichier web.xml se trouvant dans le dossier WEB-INF qui définit toutes les routes et les URLs en associant une URL à une servlet.

## Coté client

Coté client notre application se présente sous la forme d’un site web classique.

<Insérer capture d’écran>