RAPPORT DE PROJET WEB

GODARD Thibault IG3

SOMMAIRE :

I] CADRE FONCTIONNEL :

1) Exigences fonctionelles :

- Cahier des charges :

Tout d'abord, il s'agit de réaliser une application Web dynamique utilisant une base de donnée relationnelle sur un sujet au choix.
Le cahier des charges comporte néanmoins certaines contraintes techniques :

- doit être compatible avec tout type de navigateur internet : IE11, Safari, Chrome, Firefox (aussi dans leurs dernières versions bien entendu)

- doit comporter une architecture MVC.

- le choix des langage est libre. (mais à justifier)

- le choix d'une base de donnée libre. (mais relationnel)

- Contraintes : 5 tables, 2 triggers.

- Présentation et choix des technologies utilisées :

-> Pourquoi un site dynamique : (INTRODUCTION A PHP)

PHP fait partie d'un des langages permettant de créer un site dynamique : personnalisé à l'utilisateur (afficher le nom de la personne, sa photo, messagerie privée … En gros, personnalise l'expérience du visiteur. C'est différent de statique !
On peut résumer son fonctionnement comme une relation :

Client / serveur entre notre machine / la machine possédant (stockant le site web et l'envoyant à notre machine (le serveur)).

Statique -> Client demande au serveur la page -> le serveur affiche la page statique en envoyant le code HTML ET CSS
Dynamique (cahier des charges) -> le serveur génère d'abord la page (personnalisée) et l'envoie ensuite au client.

Etant donné qu'il faudra personnaliser chaque page web de notre site pour chacun des clients, cela justifie l'utilisation d'un langage permettant un site web dynamique.
Il existe de nombreuses autres technologies (langages) permettant de créer un site dynamique, je vais donc tenter de justifier le choix des technologies que j'ai utilisées.

-> **HTML et CSS** (langages de description) sont des « standards » dont on ne peut se passer dans le cadre de la création d'un site web, qui sera d'abord statique avant d'être dynamique.

-> **PHP** (Rôle de serveur/langage de « programmation ») :
C'est un langage que seuls les serveurs comprennent et qui permet de rendre notre site dynamique (comme indiqué précédemment). C'est PHP qui va « générer » la page web.
On peut aussi utiliser Java, Python, Ruby et pleins d'autres (moins connus et moins performants.)

	Php	Ruby	Python
Principale caractéristique		programmation agréable et flexible	code très lisible
Facilité d'utilisation	+	++	+
Facilité d'apprentissage	++ Très bien documenté	+	+++
Nombre de lignes de codes	+	++	+
Temps d'exécution moyen	Plutôt lent	Intermédiaire	Plutôt rapide

PHP ne semble pas être le choix le plus judicieux cependant il est très bien documenté (par une importante communauté) et c'est pour cela que j'ai préféré choisir ce dernier.

-> **MySQL** : Il existe un t'as d'autres outils :

- **Oracle** : c'est le SGBD le plus célèbre, le plus complet et le plus puissant. Il est malheureusement payant (et cher même s'il existe des versions gratuites limitées).
- **Microsoft SQL Server** : (édité par Microsoft), on l'utilise souvent en combinaison avec ASP .NET, bien qu'on puisse l'utiliser avec n'importe quel autre langage. Il est payant (même s'il existe des versions gratuites limitées).

- **PostgreSQL** : Ce SGBD est quant à lui libre et gratuit comme MySQL, propose des fonctionnalités plus avancées mais dispose d'une communauté un peu moins importante, que MySQL.

A l'image de mon choix pour PHP, j'ai préféré choisir MySQL car ils interagissent facilement ensemble, de par le package WAMP (pour ma part sous Windows) créé à cet effet.

2) Exigences non-fonctionnelles :

- Idée personnelle de projet :

Pour ma part, j'ai décidé de faire un site web qui pourrait être utilisable dans le futur, dans le cadre de Polytech, plus précisément du BDA. J'ai eu l'idée de créer une plateforme inter-école locale (sur Montpellier et ses alentours) permettant d'organiser des évènements.

Tout d'abord, ce site devait s'adresser aux étudiants musiciens, (de Musitech et autres organisations d'étudiants) dans le but d'organiser des soirées à Montpellier.
Ces soirées auraient lieu dans des espaces et lieux diverses.
Le principe serait que des groupes effectueraient des candidatures pour participer à ces soirées, et des administrateurs seraient chargés de valider, ou non, ces candidatures
Ainsi ma base de données comporterait :

- Des groupes
- Des candidatures
- Des soirées

Ensuite j'ai élargi cette idée de soirées en tant qu' « évènements » qui pourraient être des festivals, des concerts ou autres évènements. Des utilisateurs uniques (étudiant quelconque) pourraient donc proposer des évènements, et indiquer leur participation à ces évènements via ce site web. Ainsi les utilisateurs du site seraient des groupes, des administrateurs, ou des simples utilisateurs. Ma base de données comporterait :

-Des comptes utilisateurs (de type : Groupe, Admin ou simple utilisateur)
-Des soirées (de type : soirée ou évènement simple)
-Des candidatures (seulement pour les soirées constituées de groupes)

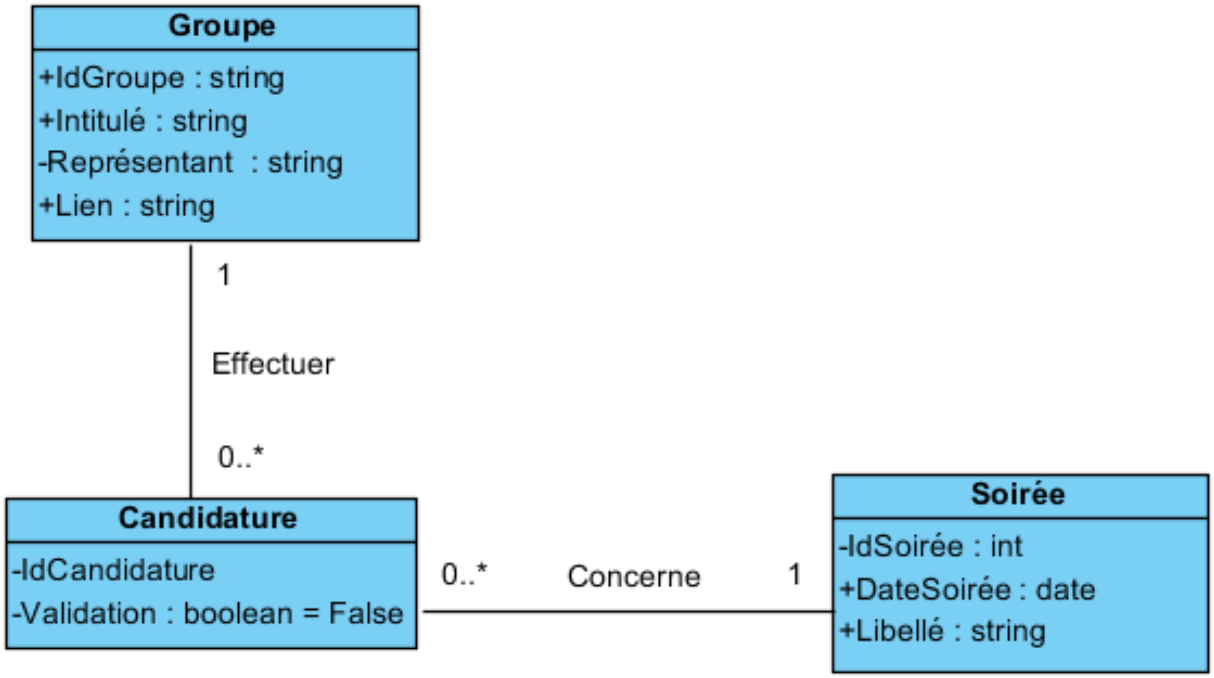
A noter que seuls les administrateurs pourront créer des évènements de type 'Soirée', auxquels pourront postuler les groupes. Naturellement, puisque ce sont eux qui géreront les candidatures.
Je ne vais pas m'attarder sur les détails avant de vous avoir présenté tout ça sous forme diagramme. Venons-en :

II] ARCHITECTURE LOGICIEL :

1) Spécification :

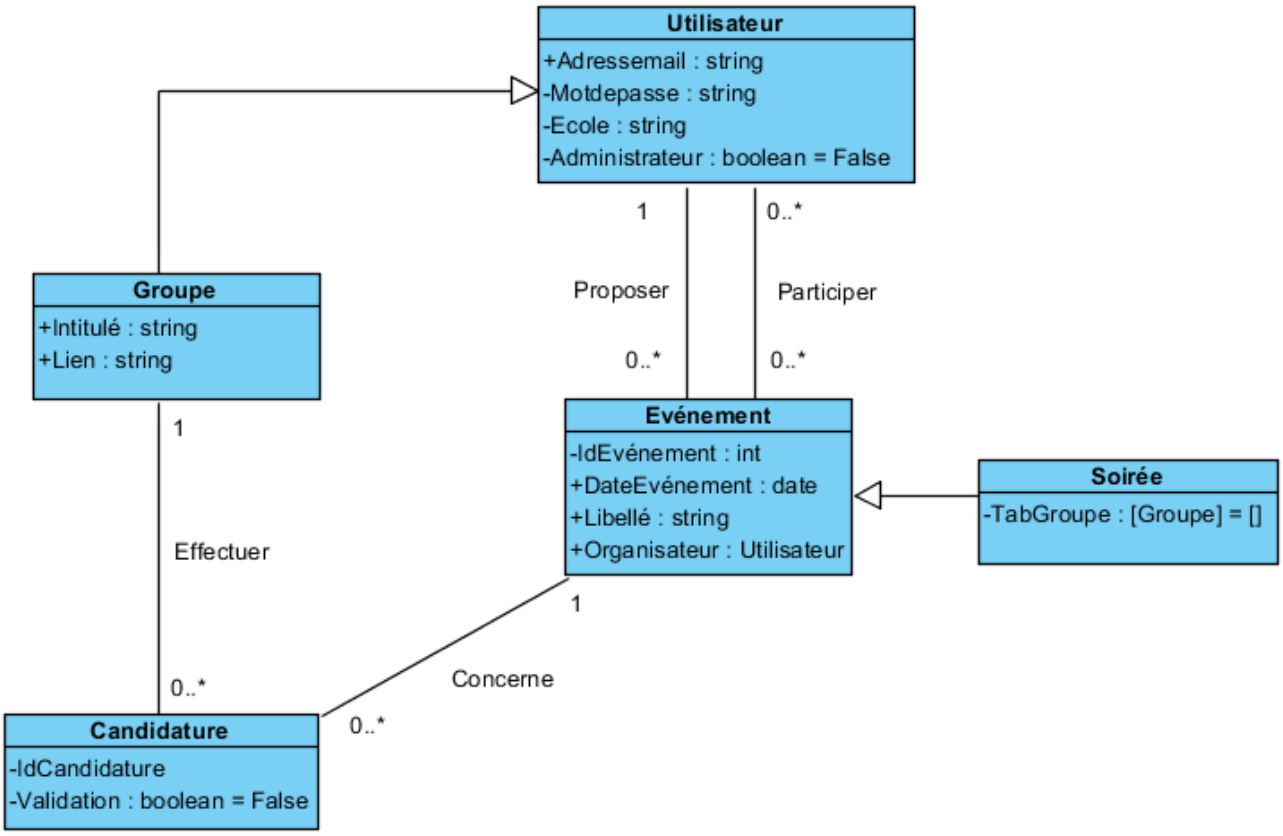
- Diagramme de classes entre les différents acteurs :
- Justification des multiplicités.

De la même manière, j'ai effectué 2 diagrammes. Un premier diagramme très simple présentant la vue candidature des groupes aux soirées.



- Un groupe peut effectuer 0 à n candidatures.
- Une candidature donnée désigne un et un seul groupe donnée.
- Une soirée peut être concernée par 0 ou n candidatures.
- Une candidature ne concerne qu'une et 1 seule candidature.

Ici, on n'a pas encore la notion d' « utilisateur », on représente simplement la vue candidature. Un super-utilisateur pourra éventuellement valider les candidatures, mais il sera le seul à pouvoir juger de la participation d'un groupe à une soirée, ce qui n'est pas possible dans la pratique.
J'ai donc créé les utilisateurs (pouvant être administrateurs).



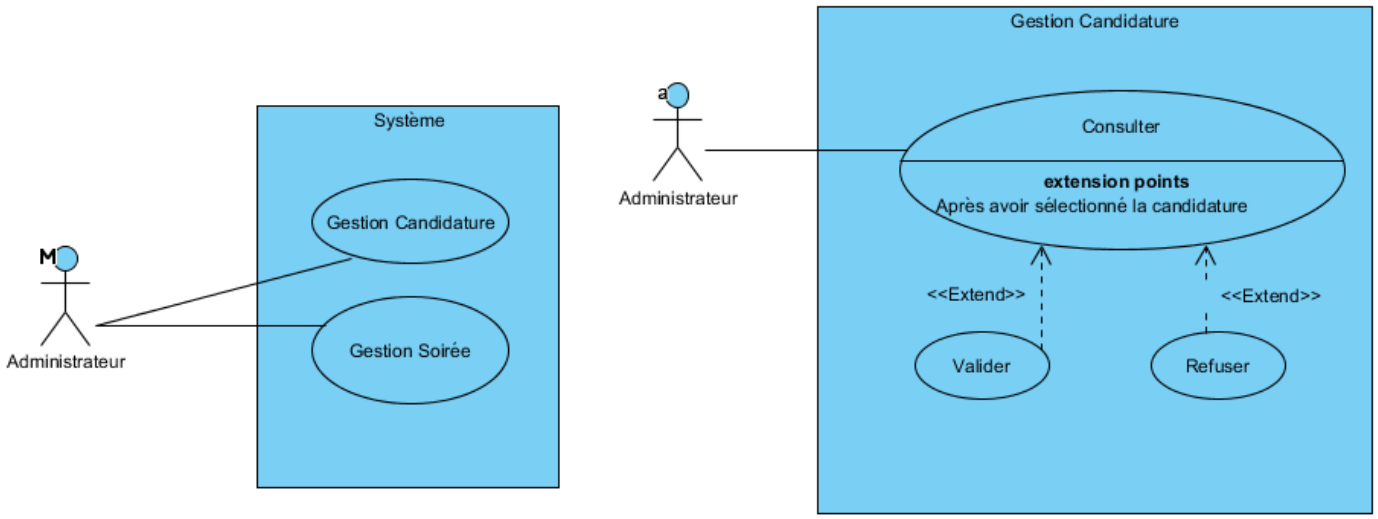
S'ajoute à la précédente liste les propriétés suivantes :
- Un utilisateur peut proposer 0 à n évènements.
- Un évènement donné a été créé (proposé) par un et 1 seul utilisateur.
- Un utilisateur peut participer à 0 ou n évènements différents.
- Un évènement est participé par 0 à n utilisateurs.

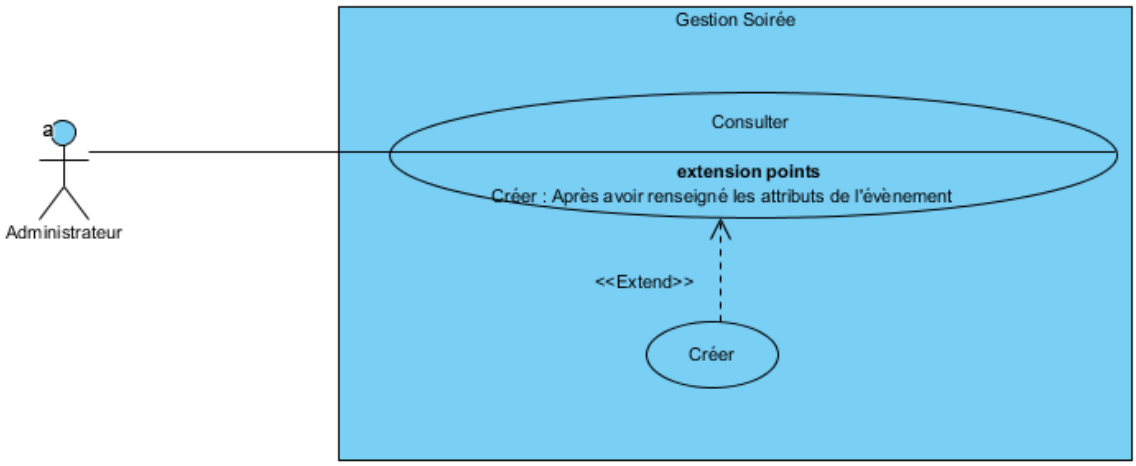
Ainsi, j'ai décidé de regrouper administrateurs et utilisateurs simples dans une même classe utilisateur, en utilisant un attribut « Administrateur » de type booléen.
D'ailleurs, rappelons que seuls les administrateurs peuvent créer des événements de type 'Soirée', et valider les candidatures :
Le booléen devra dont être à 'True' pour que cet utilisateur puisse effectuer ces manipulations, d'où l'utilisation de triggers.
En effet, le diagramme des cas d'utilisations montre bien les possibilités offertes aux utilisateurs.

2) Spécification des cas d'utilisation :

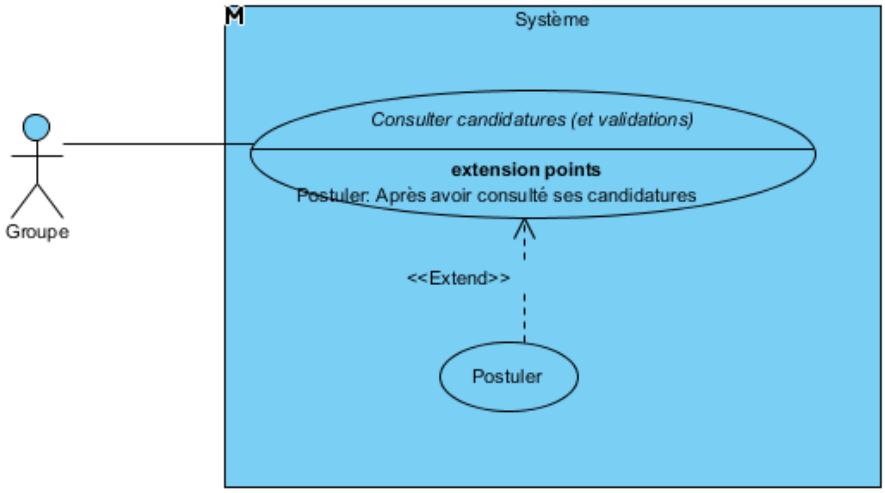
- Diagramme des cas d'utilisation :

L'administrateur :

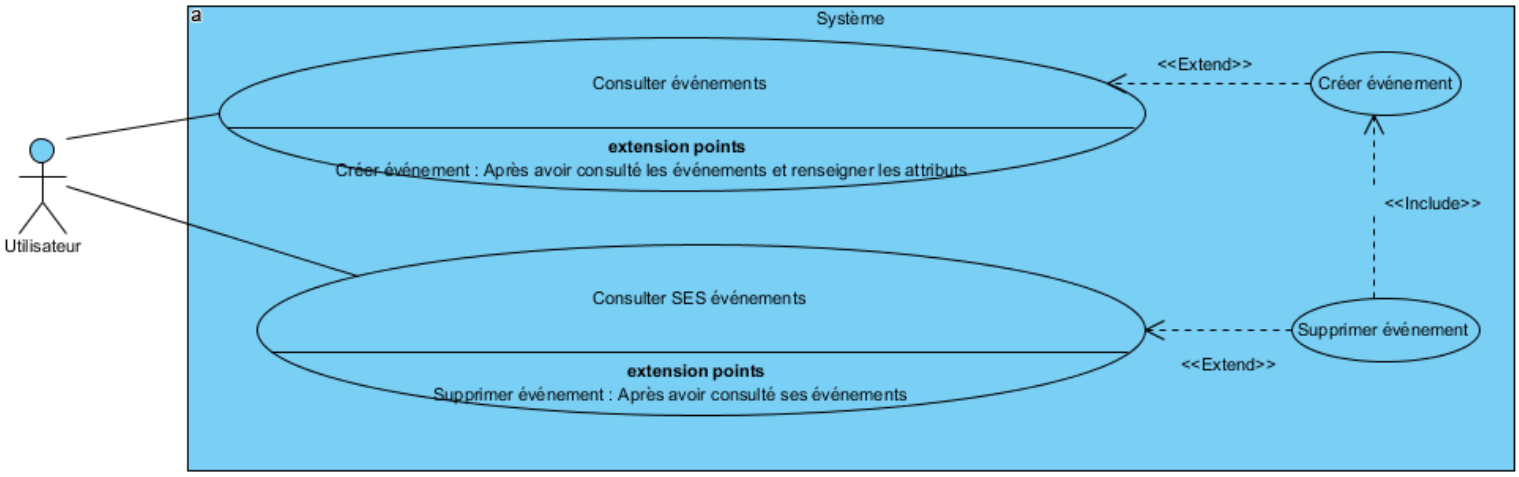




Le groupe :



L'utilisateur (de manière générale) :



III] ARCHITECTURE DE DEPLOIEMENT :

IV] ANALYSE DES RESULTATS :

1) Bilan

2) Autocritique