Projet CICEROW

***C****artographie* ***I****nformatique et* ***C****hronologie d’****E****vénements* ***R****eliés* ***O****btenus via* ***W****ikipédia*



Simon Fabry

Manon Girard

Adélaïde Lorieux

Valentin Sasyan

# Présentation du sujet

Notre projet nous a été commandé par Emmanuel Fritsch.

Il consiste à créer une interface permettant de visualiser spatialement et temporellement des événements historiques appartenant à un thème donné à partir de données Wikipédia : L’utilisateur doit pouvoir choisir un thème historique parmi une liste et obtenir tous les événements géoréférencés de Wikipédia sur une carte et une frise chronologique liés à ce thème.

Les thèmes correspondent soit à des Portails de Wikipédia (par exemple [le portail de la Rome Antique](http://fr.wikipedia.org/wiki/Portail:Rome_antique)), soit à une page Wikipédia (par exemple [la page de la Guerre de Trente ans](http://fr.wikipedia.org/wiki/Guerre_de_Trente_Ans)).

## Cahier des charges

Le commanditaire nous a demandé par ordre d’importance de

* **-Créer un modèle de données à stocker**

Ce modèle doit notamment comprendre :

Le nombre de langue dans lequel une page Wikipédia est disponible

La longueur de la page Wikipédia

Le nombre de visites de la page

Et toute information que nous trouverons pertinente

* **-Permettre la visualisation des données avec un viewer et une frise chronologique**

L’utilisation de l’API google TimeMap (<https://code.google.com/p/timemap> ) nous a été recommandée par le commanditaire.

Pour la frise chronologique, l’étendue d’un événement doit pouvoir être modifiée selon son importance.

* **-Créer un « aspirateur » de données**

Langage de programmation:

Nous avons le choix entre du PHP pur, du PHP avec du JavaScript côté client ou du Node.

L'aspirateur doit être récursif : Pour chaque lien, on vérifie que la page appartient au thème (nom du portail) et si oui, on traite la page et chaque lien dans cette page.

Les informations qui ne sont pas géolocalisées doivent tout de même être aspirées pour pouvoir être affichées sur la frise chronologique.

* **-Créer une interface de personnalisation de la carte**

L'utilisateur doit pouvoir choisir les événements qu'il veut voir apparaître sur la carte et sur la frise chronologique, et modifier leur apparence.

* **-Optionnel : descendre en granularité dans l'aspirateur**

Récupération d’informations sur des événements qui n'ont pas de page Wikipédia mais seulement un paragraphe dans une page en rapport avec le thème (Par exemple, dans la page sur la bataille de Verdun, paragraphe sur les combats autour de Douaumont).

Pour chaque étape, des tests unitaires doivent être effectués.

# Solution proposée

## Choix techniques et fonctionnels

**Langages de programmation  utilisés:**

PHP avec du JavaScript côté client

Développement de la Base de données avec mysql et du PHP orienté objet

**Aspirateur de données**

Pour chaque portail ou page qui correspond à un thème, le parseur analyse toutes les pages liées à ce portai ou cette page.

Le parseur permet de récupèrer uniquement les informations relatives à des pages ayant des infobox (cadre à droite de la page). En effet, il est trop difficile d’identifier les données utiles sans infobox.

Les informations stockées d’une page sont : son identifiant, sa longitude, sa latitude, le type d’infobox (conflit militaire, traité…), le nombre de langues, le nombre de visites, la longueur de la page, l’année, le mois et le jour du début de l’événement, l’année, le mois et le jour de la fin de l’événement, la date de la dernière mise à jour de la page, la distance au portail

Nous avons laissé de côté le point optionnel car associer une date avec un événement au milieu d’une page sans logiciel capable d’analyser un texte de façon intelligente nous paraît très difficile à mettre en place.

**Personnalisation d’une carte**

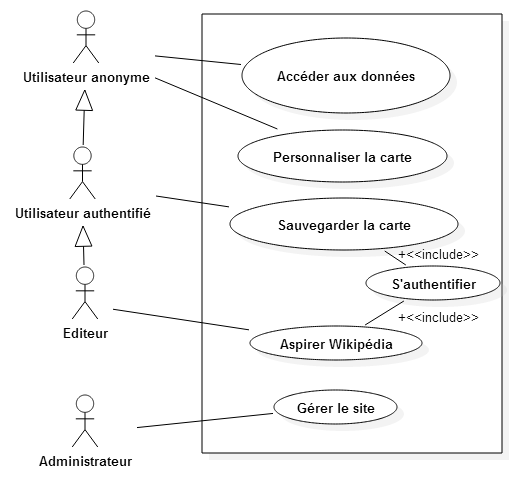
Possibilité de changer le symbole, la taille, et la couleur d’un événement.

Génération de thèmes (choix d’un symbole, d’une taille et d’une couleur) automatiques selon le type d’infobox.

L’utilisateur pourra décide d’afficher les X éléments les plus importants d’un thème. Nous avons décidé que l’importance d’une page serait calculée selon cette formule :

## Architecture

### Diagramme de cas d’utilisation

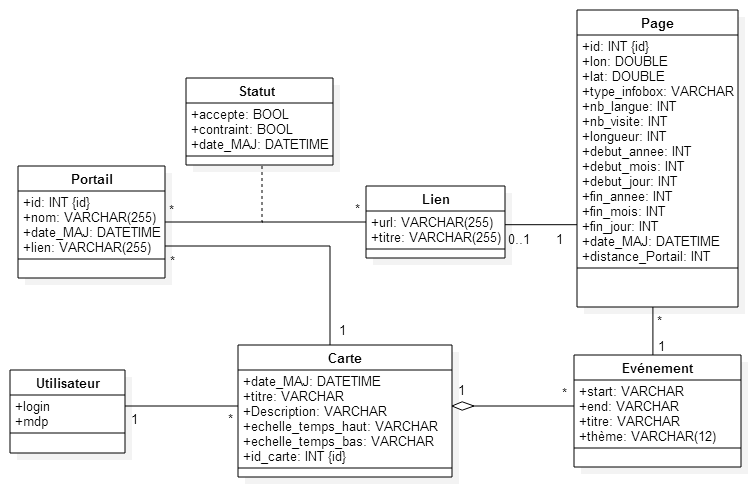


Toutes les données du site (c’est-à-dire les cartes générées accompagnées d’une frise chronologique) sont accessibles par n’importe quel **utilisateur anonyme**. Ces **utilisateurs anonymes** peuvent aussi décider de Personnaliser une carte. Cependant, seuls les **utilisateurs authentifiés** peuvent Sauvegarder une carte personnalisée.

Les « **éditeurs** » sont des utilisateurs enregistrés privilégiés qui peuvent aussi mettre à jour les pages associées à un portail et créer de nouveaux portails (« Aspirer Wikipédia »).

Un **administrateur** gère le côté technique du site. Son rôle se différencie bien des autres acteurs dont il n’hérite donc pas.

### Base de données :



La BDD comporte deux parties à considérer séparément dans un premier temps : une partie « aspiration des données » (objets **Portail**, **Statut**, **Lien** et **Page**) et une partie « personnalisation des cartes » (objets **Utilisateur**, **Carte** et **Evénement**).

#### Partie Aspiration des données :

Un utilisateur « éditeur » va dans un premier temps créer un **Portail** puis ajouter au portail des **Lien**s. Chaque **Lien** est stocké puis considéré ou non comme pertinent. Si le **Lien** est considéré comme pertinent (accepte = true) un objet **Page** est créé comportant les informations supplémentaires à stocker.

L’éditeur peut forcer l’acceptation ou le rejet d’une **Page**.

#### Partie Personnalisation des cartes :

Lorsqu’un utilisateur veut personnaliser une **Carte**, l’application crée dans un premier temps l’objet **Carte**, associé à un **Portail**. Puis toutes les **Page**s associées au **Portail** de la **Carte** sont recherchées et un **Evénement** associé à chaque **Page** est créé.

L’utilisateur peut ensuite modifier ou supprimer les **Evénement**s pour personnaliser la carte.

### Echange des données :

Beaucoup de données sont échangées entre JavaScript et PHP au moyen de la notation JSON.

#### Echanges JSON__EditEchanges pour la partie Edition des portails :

Le programme peut échanger au choix :

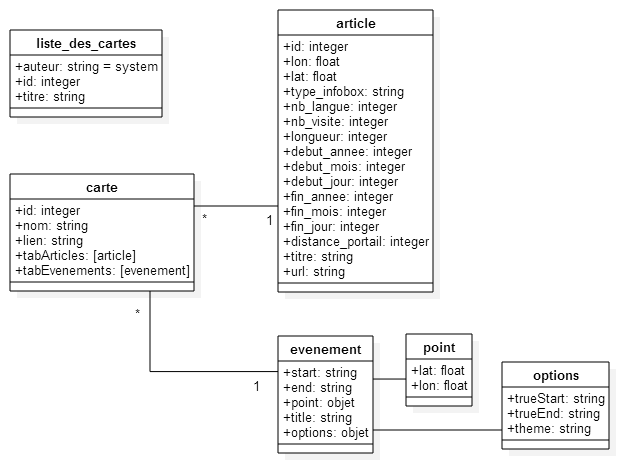
* Une liste de portails, c’est-à-dire un tableau « liste\_des\_portails », contenant comme informations le **nom** du portail, la date de la dernière mise à jour (**date\_MAJ**), et le lien vers le portail.
* Une liste des articles à ajouter, c’est-à-dire un tableau « liste\_des\_articles\_a\_ajouter », contenant toutes les informations nécessaires pour compléter l’objet « Page » dans la BDD.

#### Echange pour la partie Viewer :

Le programme peut échanger au choix :

* La liste des cartes disponibles, c’est-à-dire le tableau « **liste\_des\_cartes** », contenant l’identifiant, le nom et l’auteur de chaque carte. Pour chaque portail, est générée automatiquement et dynamiquement une « carte auto », référencée comme ayant pour auteur « systeme ».
* Une **carte**, c’est-à-dire un objet reprenant les informations de l’objet **Carte** de la BDD et un **tableau** :
  + d’événements (**tabEvenements**) lors qu’il s’agit d’une carte personnalisées.
  + d’articles (**tabArticles**) lorsqu’il s’agit d’une « carte auto ».

Le programme transforme côté client les **Articles** en **Evénements** affichables.



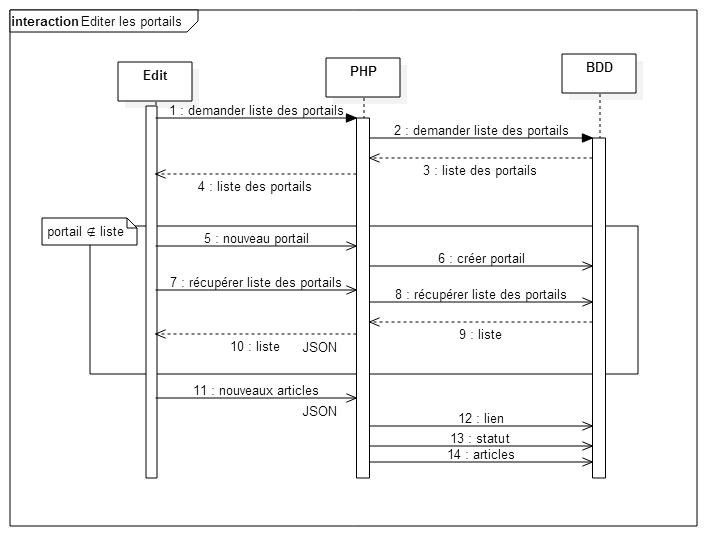
### Diagrammes de séquence

#### Edition des portails :

La création d’une carte s’effectue par un éditeur sur le site web de l’application. Cela est effectué coté client via JavaScript. Dans un premier temps le client demande au serveur la liste des portails. Puis le client tape le nom d’un portail.

Si le portail est inexistant (c'est-à-dire pas dans la liste), on demande au client s’il veut créer le portail. Dans ce cas-là, le client vérifie que le portail existe sur le site de Wikipédia puis ajoute le portail.

Si le portail existe, le client affiche les informations disponibles sur le portail et les interactions disponibles (notamment la mise à jour).



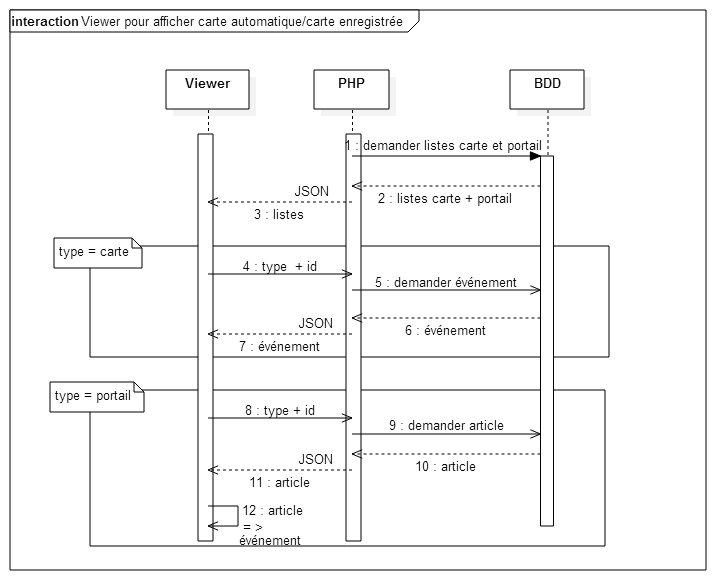
#### Viewer :

Le Viewer est effectué sur le site de l’application. Il est accessible aux utilisateurs et aux visiteurs. Le travail est effectué en JavaScript coté client.

Le client demande dans un premier temps la liste des cartes et des portails (carte « auto »).

L’utilisateur va en choisir un.

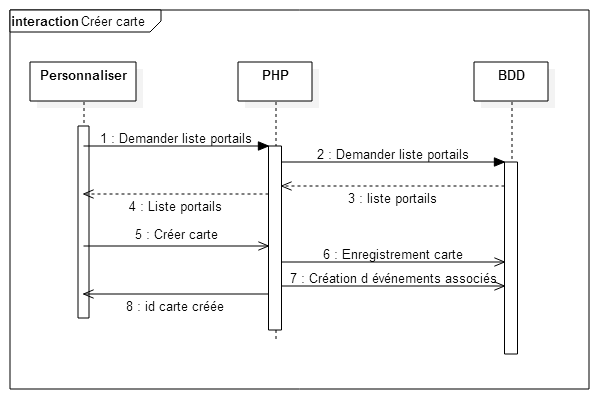
Si c’est un portail qui est choisi (c’est-à-dire une « carte auto ») le client demande au serveur tous les **Article**s associés au portail et les convertis en **Evénement**s avant de les afficher.



#### Création des cartes :

La fonction est traitée côté client en JavaScript.

Le client se connecte au serveur et récupère la liste des portails. L’utilisateur choisi un **Portail** et crée ainsi une carte. Le programme PHP crée alors un **Evénement** associé à la **Carte** pour chaque **Page** associée au **Portail**. Il retourne ensuite l’identifiant (id) de la carte.



#### Chargement d’une carte :

La fonction est traitée côté client en JavaScript.

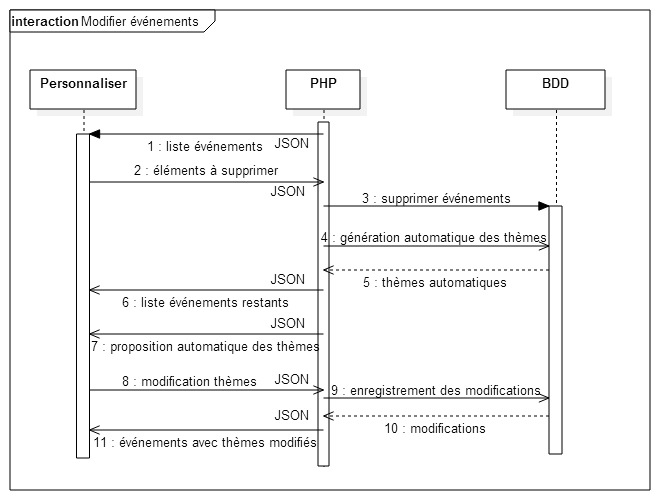
Le client se connecte au serveur et récupère la liste des cartes. L’utilisateur choisi une **Carte**. Le client demande au serveur les informations relatives à la carte et les affiche.



#### Modification des évènements liés à une carte :

La fonction est traitée côté client en JavaScript.

Le client se connecte au serveur et récupère la liste des **Evénement**s associés à la **Carte**. L’utilisateur choisit les **Evénement**s à supprimer. Le client renvoie la liste au serveur. Ce dernier effectue une génération automatique des thèmes sur tous les **Evénement**s, puis renvoie la liste des **Evénement**s avec thème au client. Le client les affiche. L’utilisateur peut modifier le thème de chaque **Evénement**. Le client renvoie les informations au serveur.



# Avancement



Diagramme de Gantt au 18/03

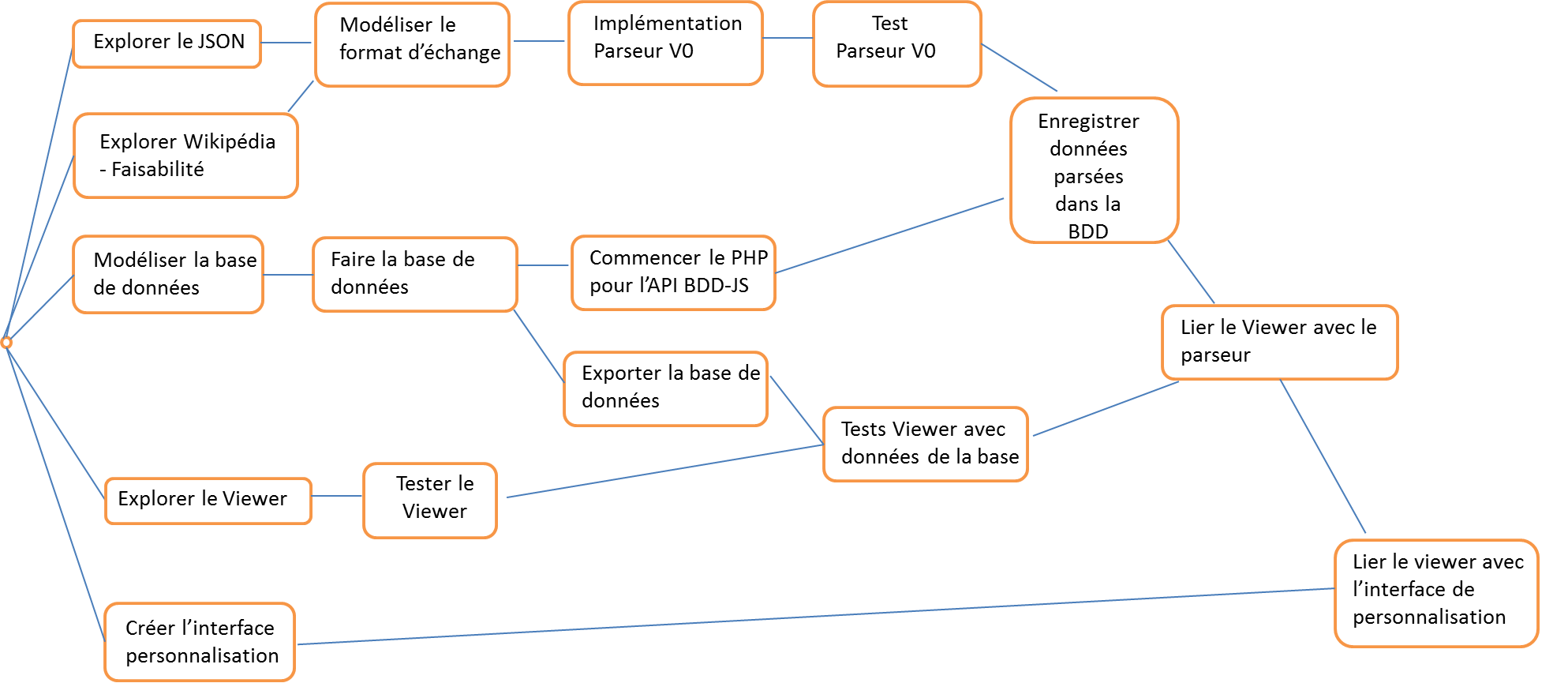


Diagramme Pert : planification des tâches à effectuer