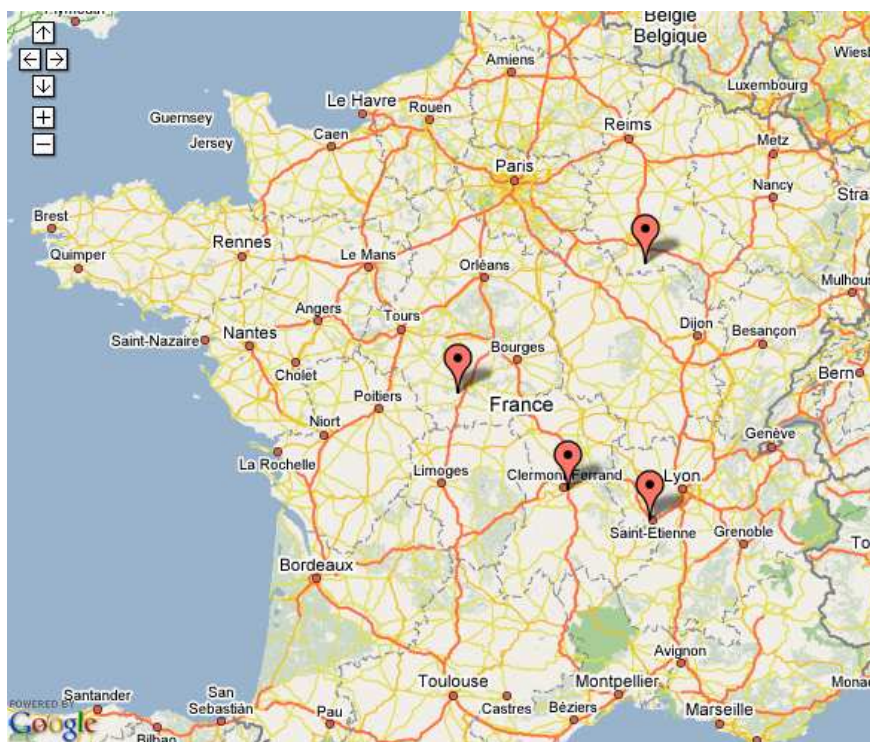


Utilisation de Google Map pour observer une répartition géographique



Remerciements

Tout d'abord nous tenons à remercier notre tuteur, M. Pierre Chatelier, pour toute l'aide qu'il nous a apporté au cours de notre projet, pour ses conseils qui nous ont été précieux lors de la mise en place de notre site, mais également pour sa disponibilité au cours de ces mois de projet.

Nous tenons également à remercier Mme Nadine Giraud, pour son aide quant à la rédaction de notre rapport.

Sommaire

Remerciements	2
Sommaire	3
Introduction	5
I. Le contexte	6
1. <i>L'Institut Universitaire de Technologie de Clermont-Ferrand</i>	6
2. <i>Les cours d'informatique</i>	6
3. <i>Le sujet</i>	7
II. Les Technologies	8
1. <i>xHTML*</i>	8
a. <i>Le langage</i>	8
b. <i>La mise en place d'une page internet</i>	9
c. <i>Les formulaires</i>	9
2. <i>PHP / SQL</i>	12
a. <i>Présentation du langage PHP</i>	12
b. <i>Les variables</i>	13
c. <i>Transmission de données entre pages</i>	14
α-les POST	14
β-les GET	15
γ-les COOKIES	15
d. <i>Compatibilité entre PHP et bases de données</i>	16
e. <i>Le langage SQL*</i>	17
α-La sélection de données	17
β-L'insertion de données	19
3. <i>JavaScript</i>	20
a. <i>Le DOM*</i>	20
b. <i>Le langage JavaScript</i>	21
c. <i>Les fonctions JavaScript</i>	22
d. <i>Les fonctions anonymes*</i>	23
4. <i>AJAX</i>	24

III. La mise en place du site	26
1. L'API Google Map	26
a. La documentation	26
b. Les cartes	27
c. Les marqueurs	30
α. Les gestionnaires de marqueur	30
β. La récupération des marqueurs	32
2. La partie client	34
a. index.php	34
b. Debut.php	35
c. Default.php	36
d. Page_logge.php	36
e. mal_logge.php	38
f. marker.php	38
g. changement.php	39
h. delog.php	39
3. La partie administrateur	40
a. Cryptage des mots de passe	40
b. ajout.php	41
α. admallog.php	42
β. adlog.php	43
γ. nouveau.php	45
δ. retirer.php	46
Bilan	47
1. Résultats	47
2. Difficultés	47
3. Perspectives	48
Conclusion	49
English Summary	50
Bilan Personnel	51
Bibliographie	52
Glossaire	53
Annexes	58

Introduction

Dans le cadre de l'enseignement à l'Institut Universitaire de Technologie (IUT) de Clermont-Ferrand et en particulier des cours d'informatique, nous avons eu pour tâche de réaliser un projet, sous la tutelle de M. Chatelier, Attaché Temporaire d'Enseignement et de Recherche à l'IUT.

Ce projet consiste en la réalisation d'un *site Internet** utilisant *Google Map** afin d'observer une répartition géographique, et plus particulièrement, pouvoir localiser les anciens étudiants, en France et à l'étranger.

Google Map est une application qui permet de cartographier différentes régions du globe, à différentes échelles. Nous avons dû trouver les outils nécessaires à la réalisation de ce site et apprendre à les mettre en place.

Nous présentons tout d'abord les prérequis de ce sujet, c'est-à-dire les différentes technologies dont nous nous sommes servis pour mettre en place notre site. Nous expliquons ensuite comment nous les avons utilisées dans le cadre de notre projet.

I. Le contexte

1. L'Institut Universitaire de Technologie de Clermont-Ferrand

L'Institut Universitaire de Technologie* (IUT) de Clermont-Ferrand compte 2250 étudiants, 174 enseignants et enseignants-chercheurs et 67 collaborateurs chargés de l'administration et des missions techniques.

L'IUT propose des formations (*Diplôme Universitaire de Technologie** (DUT) et licences professionnelles) sur trois sites géographiques : dans le Cantal à Aurillac, dans la Haute-Loire au Puy-en-Velay et dans le Puy de Dôme à Clermont-Ferrand. Les domaines sont nombreux et variés (16 DUT et 17 licences professionnelles). En biologie il y a des formations dans l'agronomie, l'environnement, la bioinformatique, les analyses biologiques et chimiques, la diététique, en Sciences de l'Ingénieur dans les mesures physiques, le génie industriel et maintenance, la chimie, en gestion dans la finance et la comptabilité, les ressources humaines et enfin en Informatique, dans l'informatique pure, la communication et multimédia et les réseaux et télécommunications.

A la fin de chaque parcours, un stage de fin d'étude est obligatoire. Ces stages sont dispersés sur toute la France, certains le font même à l'étranger. De plus, après ce stage, certains commencent à travailler, tandis que d'autres poursuivent leurs études. C'est notamment cette répartition que nous allons observer à la fin de ce projet.

Plus précisément, le département réseaux et télécommunications, dans lequel nous étudions, propose un enseignement scientifique pluridisciplinaire avec des matières comme l'électronique, les télécommunications, les réseaux et l'informatique.

2. Les cours d'informatique

En effet, tout au long de notre formation, les cours d'informatique nous ont permis de nous familiariser avec la manipulation des différents *systèmes d'exploitation**, principalement *Unix**, qui nous était presque inconnu lorsque nous sommes arrivés dans ce département. Nous avons également appris les bases de différents langages de programmation, *C**, *Java** mais aussi des langages de programmation de web tels que *HTML**, *CSS**, *PHP**. Ce dernier cours était organisé par notre tuteur, M. Pierre Chatelier et nous a servi pour la réalisation de notre site.

3. Le sujet

Le but de notre projet était de réaliser un site internet utilisant l'API¹ Google Map afin de visualiser une répartition géographique. Ce projet avait plusieurs objectifs, tout d'abord apprendre le *JavaScript**, un langage de programmation qui n'entrait pas dans le cadre de nos cours. Nous avons donc eu recours à un travail de documentation et d'auto-apprentissage pour nous familiariser avec ce langage. Ensuite, nous avons étudié l'API Google Map, une application qui utilise JavaScript pour fonctionner et qui permet d'insérer une carte sur une page web. Celle-ci permet également de se déplacer sur la surface du globe pour visualiser différents endroits à différentes échelles, mais également de placer des *marqueurs** sur la carte pour voir la localisation d'une personne, d'un lieu... Ces marqueurs peuvent être définis « fixes » ou alors « dynamiques ». Dans le cas dynamique, les coordonnées des marqueurs sont alors stockées dans une *base de données** et la *page web** fait appel à cette base de données à chaque fois qu'elle souhaite afficher une carte. C'est cette méthode que nous avons utilisée. De plus, nous avons eu à réaliser un système d'*authentification** utilisant la base de données afin de permettre à chaque utilisateur enregistré de déplacer son propre marqueur. Enfin, pour finaliser notre projet, nous avons récupéré la liste des étudiants de la promotion 2006 avec leur situation actuelle, et nous avons rentré leurs coordonnées dans la base de données.

¹ API : Application Programming Interface

II. Les Technologies

1. xHTML*

a. Le langage

Le xHTML (eXtensible HyperText Markup Language) est un langage de programmation web relativement simple et très facile à maîtriser. Ce langage est interprété par le *navigateur internet**, ce qui donne la page internet sur laquelle l'utilisateur peut naviguer. Par habitude, il est souvent appelé HTML. Il repose sur l'utilisation de *balises**, des marqueurs entre chevrons. Une balise commence toujours par « < » et se finit par « > ». Par exemple : <balise>

Il existe deux types de balises :

- ✚ Les balises existant par paires, une balise ouvrante, <balise> et une balise fermante </balise>. La balise fermante commence toujours par « / ». Elles sont utilisées pour encadrer des blocs de texte ou de code HTML.

```
<title> Ceci est le titre de la page </title>
```

Entre ces deux balises se trouve le titre de la page.

- ✚ Les balises seules, servent en général à insérer un objet dans la page. Elles se terminent toujours par ' / '.

```

```

Cette balise indique qu'il y a une image à cet endroit. De plus, elle contient un *attribut**. Il s'agit d'un moyen de donner des précisions sur la balise. Sur cet exemple, nous indiquons la source de l'image avec l'attribut « src », c'est l'endroit où il faut aller chercher l'image.

Un attribut important pour toute balise HTML est l'attribut « id ». Celui-ci définit un identifiant unique pour un bloc. Ainsi, si nous souhaitons effectuer des actions sur ce bloc, nous nous reporterons à son identifiant. Ceci est notamment vrai pour le langage JavaScript (II.3.a, page 20).

b. La mise en place d'une page internet

Le contenu d'une page HTML est encapsulé entre les balises `<html>` et `</html>`. Plusieurs lignes sont cependant nécessaires au début d'une page web. Ce sont des balises « Méta », qui donne des indications sur le contenu de la page. Tout d'abord la balise « xml » qui informe sur le document et son encodage.

```
<?xml version="1.0" encoding="UTF-8"?>
```

Ensuite la balise « DOCTYPE » qui informe sur la grammaire utilisé.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Placer une balise « DOCTYPE » permet de dire au navigateur « Je connais les standards du web et cette page est faite avec telle version de XHTML ».

Il existe aussi de nombreuses balises « Méta » optionnelles afin de donner encore plus d'informations tel qu'une description de la page web, les mots-clefs qui servent au référencement sur les *moteurs de recherche**...

```
<meta http-equiv="content-type" content="text/html; charset=UTF-8" />
```

Une page internet est décomposée en deux grandes parties :

- ✚ *L'en-tête* : qui sera contenu entre les balises `<head>` et `</head>`. Elle contient des informations sur la page web, dont la balise `<title>`, les balises `<meta>` et les balises `<script>` qui permettent d'insérer des scripts dans un autre langage de programmation. Nous verrons par la suite que cette balise est notamment utilisée pour insérer du JavaScript (II.3.b, page 41).
- ✚ *Le corps* : délimité par les balises `<body>` et `</body>`. Il contient ce qui est affiché par le navigateur. C'est ici que sont placés les *formulaires**.

c. Les formulaires

Grâce au langage HTML, nous pouvons créer des formulaires, que l'utilisateur remplit. Les données peuvent ensuite être traitées par un langage de programmation comme le PHP. Dans notre site, nous avons eu recours à des formulaires afin d'identifier les utilisateurs, mais aussi pour récupérer les coordonnées d'un marqueur, ce que nous verrons par la suite (III.1.c, page 30).

Les formulaires se définissent grâce à la balise `<form>`. Cette balise contient deux attributs essentiels. Tout d'abord l'attribut « action », qui contient la page web ou l'url de la page web qui traitera les informations, et ensuite l'attribut « method », qui peut prendre deux valeurs : **GET** ou **POST**. Nous montrerons les différences entre ces deux méthodes en partie II.2.c, page 14.

Voici à quoi ressemble le squelette d'un formulaire.

```
<form action="index.php" method="post"> </form>
```

Maintenant, il faut remplir ce formulaire. Pour cela, nous nous servons de la balise `<input />`. Elle permet d'afficher des *zones de saisie** dans un formulaire. Les différentes zones de saisie sont indiquées via l'attribut « type ». De plus, à chaque zone de saisie il faut également affecter un attribut « name ». Cet attribut sera utile lors du traitement des données du formulaire, c'est en fait l'identifiant de la zone de saisie.

Voici les différents types de zone de saisie :

✚ Les zones de texte* :

Elles permettent d'afficher du texte ou permettre à un visiteur de rentrer des informations. La valeur attribut « type » permettant d'insérer une zone de texte est tout simplement « text ».

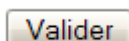
```
<input type="text" name="login" />
```

✚ Les zones de texte de type mot de passe :

Tout comme les zones de texte normales, elles permettent de saisir du texte. Afin d'éviter à des personnes indiscreètes de voir le mot de passe tapé, les caractères sont remplacés par des étoiles. Le type de cette zone de saisie est « password ».


```
<input type="password" name="pass" />
```

Les boutons*



Ils servent à « valider » le formulaire et donc envoyer les données des zones de saisies vers la page de traitement (la page définie dans la balise <form> par l'attribut « action »). Le type pour un bouton est « submit ». Il existe également l'attribut « value » qui permet de définir ce qui sera écrit sur le bouton.

```
<input type="submit" name="bouton" value="valider" />
```

 Il existe d'autres type de zones de saisie comme par exemple les *cases à cocher**, les *zones d'options** ou encore les *listes déroulantes**. Nous n'en parlerons pas dans ce rapport car elles ne nous ont pas été utiles pour la mise en place du site.

Voici un exemple de formulaire que nous utilisons dans notre site.

```
<form action="index.php" method="post">  
  
  Login :      <input type="text" name="login" />  
  Mot de passe : <input type="password" name="pass" />  
                <input type="submit" value="Valider"/>  
  
</form>
```

Et voici ce code vu par un navigateur.

Login :

Mot de passe :

Ainsi, avec ce formulaire, lors du clic sur le bouton « valider », les données « login » et « pass » sont envoyées. Ces données contiennent les valeurs des champs de texte auxquelles elles correspondent.

La page qui traitera ces informations est écrite dans le langage PHP, qui permet de lier plusieurs pages d'un site web entre elles mais aussi de mettre en relation ces pages avec une base de données. Nous allons donc maintenant étudier le principe du langage PHP.

2. PHP / SQL

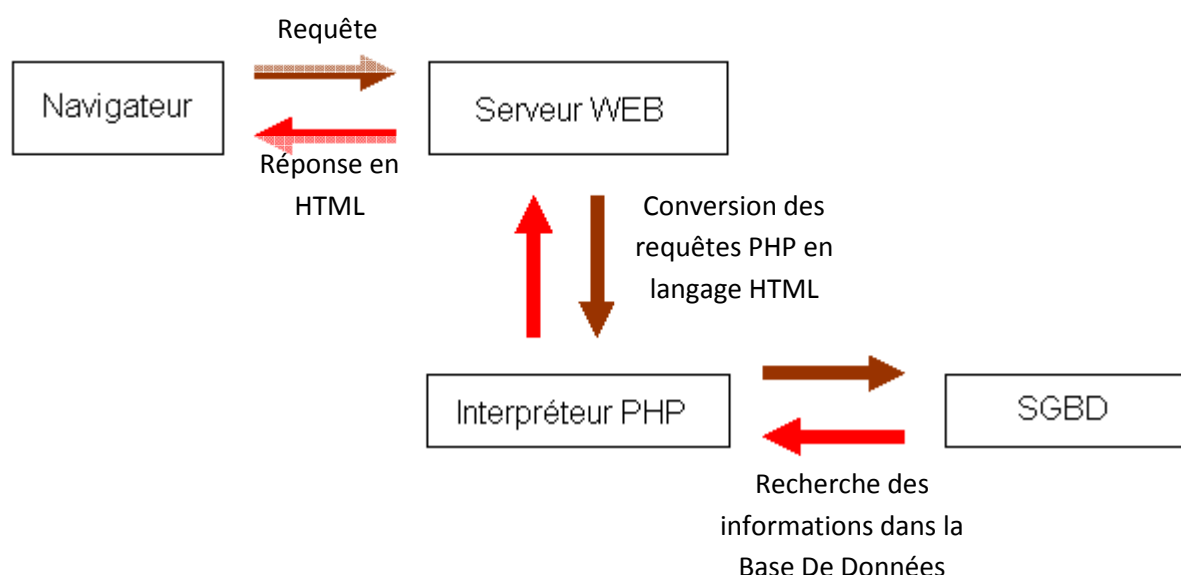
a. Présentation du langage PHP

L'évolution dans le milieu des technologies Internet a exigé un dynamisme des pages afin qu'elles s'adaptent aux exigences des utilisateurs. Pour permettre ce dynamisme, le langage PHP, facilement intégrable avec le HTML puisque l'on peut l'intégrer au sein de ce même code, a été créé. PHP est officiellement un sigle récursif pour HyperText Preprocessor car le serveur doit traiter la requête PHP avant de renvoyer le code HTML correspondant.

Les principaux atouts du PHP sont :

- Il dynamise les pages Web et permet donc de les adapter à l'utilisateur
- Sa liberté d'utilisation : le code source de Zend Engine (nom de l'interpréteur dans le cas du langage PHP) est accessible à tous.
- Sa capacité à gérer les transmissions de données d'une page à l'autre en utilisant les fonctions *POST** ou *GET**, *les sessions** ou encore *les cookies**.
- Sa compatibilité avec les Systèmes de Gestion de Bases de Données (SGBD)
- Le fait que le code PHP soit indisponible au public (qui ne peut avoir accès qu'au résultat après interprétation). Cela permet de garder les mots de passe à la base de données inaccessibles aux visiteurs.

Intégrer du code PHP dans une page nécessite la présence d'un serveur web. C'est celui-ci qui va générer du langage HTML. On peut schématiser une demande de page PHP sur un site de la manière suivante, l'interpréteur et le SGBD se trouvant bien évidemment sur le serveur :



Pour que le serveur comprenne qu'il doit interpréter les instructions qui lui arrivent il faut lui signaler le code PHP entre les balises « `<?php` » et « `?>` ».

La fonction principale du langage PHP est la fonction d'écriture, « `echo` » qui permet de créer la page HTML de façon dynamique. Par exemple :

```
<p>
  Ceci est du HTML pur
  <?php echo 'Ceci est du langage PHP'; ?>
</p>
```

Une fois la page chargée nous pouvons remarquer que la partie PHP a bien été interprétée par le serveur puisque ce code a été remplacé dans le code source, lisible par l'utilisateur qui a chargé la page, par :

```
<p>
  Ceci est du HTML pur
  Ceci est du langage PHP
</p>
```

b. Les variables

Si nous souhaitons afficher une suite de caractères plusieurs fois ou encore le garder pour une quelconque raison en mémoire, nous utilisons les variables. Pour comparer à l'exemple précédent nous avons ceci :

```
<?php
  $login= "Ceci est du langage PHP";
  echo $login;
?>
[...]
```

```
<?php
  echo $login;
?>
```

Une fois interprété, ce code devient :

```
Ceci est du langage PHP
...
Ceci est du langage PHP
```

Dans cet exemple nous montrons d'une part qu'il existe une façon différente de la précédente pour écrire le texte mais également que la variable définie ici, « login », peut être appelée de partout dans le code PHP : en effet elle garde la valeur attribuée même entre plusieurs blocs PHP différents dans une même page.





Ce langage de programmation est tout aussi capable de gérer les chiffres, en effet si on lui met :

```
<?php
    $nombre=25+12 ;
    echo $nombre ;
?>
```

L'interpréteur PHP considère la variable nombre comme un entier et enregistre dans cette même variable le résultat de l'opération demandée (ici le serveur renvoi « 37 »).

c. *Transmission de données entre pages*

L'utilisation des variables dans une seule page, comme dans les exemples qui ont précédé, est très utile pour réaliser des fonctions, de simples scripts, mais ces variables peuvent également être utilisées lors des transmissions de données entre plusieurs pages. Pour transmettre ces données on a plusieurs possibilités différentes :

-  Les POST
-  Les GET
-  Les COOKIES
-  Les SESSIONS

α-les POST

Tout comme pour la méthode GET, il s'agit là de données envoyées à partir d'un formulaire. Dans le cas du POST, les données passent d'une page à l'autre de manière invisible pour l'utilisateur. En effet ce dernier complète les champs du formulaire puis envoie les données mais ne voit pas ce qui a été transmis. Cette technique est assez pratique mais elle a un défaut majeur qui est qu'il faut à tout prix cliquer sur l'envoi d'un formulaire pour conserver les données nous concernant : donc les liens normaux ne permettent pas de garder notre identifiant en mémoire (de même pour GET).

Dans la page qui reçoit les données, nous récupérons un *tableau associatif** de données \$_POST.

β-les GET

Dans ce deuxième cas les données peuvent être très bien générées dans un formulaire, mais, à partir du moment où ces variables sont lisibles et modifiables dans *la barre d'adresse des navigateurs**, elles peuvent tout aussi bien être définies tout simplement dans *l'URL de la page**. Ainsi on peut obtenir des adresses de sites de forme semblable à cet exemple :

```
http://www.site.fr/page.php?login=valeur
```

Dans cet exemple on souhaite ouvrir la page "page.php" contenue sur le serveur site.fr avec la variable login à laquelle est attribuée la valeur « valeur ».

Les données envoyées en GET sont recueillies dans un tableau associatif \$_GET.

γ-les COOKIES

Un cookie est un petit fichier envoyé par un serveur web à un utilisateur au cours d'une connexion afin de conserver des variables propres à la personne qui navigue. Ce fichier s'inscrit sur le disque dur de l'utilisateur dans le but d'être réutilisé plus tard par d'autres pages web. En général, les cookies sont utilisés pour personnaliser un service ou suivre la trace de l'utilisateur à des fins marketings. Au départ, leur rôle étant de transférer des données d'une page à l'autre.

Pour initialiser un cookie on utilise la fonction « setcookie() » qui prend les paramètres suivants :

```
setcookie("nom_du_cookie", "valeur_a_retenir", "date_d'expiration");
```

Ensuite, nous récupérons les données dans une table : la table \$_COOKIE. Les clés que nous pouvons trouver dans cette-même table sont identiques aux noms des cookies que l'on a émis. En effet, pour avoir plusieurs clés et donc plusieurs variables retenues il nous faut émettre plusieurs cookies différents (portant des noms distincts pour ne pas écraser de cookies existants).

Nous récupérons la donnée en mémoire dans un cookie que nous souhaitons mettre dans une variable « \$variable », à une date antérieure à la date d'expiration de la manière suivante :

```
$variable=$_COOKIE['nom_du_cookie'] ;
```

Les cookies ont cependant un grand défaut : une grande partie des utilisateurs bloque les cookies par mesure de sécurité. Il a donc fallu créer une façon tout aussi performante mais qui utilise une technique différente comme les sessions que nous montrons dans la sous-partie suivante.

δ-les SESSIONS

Cette technique diffère des cookies dans la mesure où les données ne sont pas renvoyées par le client à chaque demande de page web mais elles sont stockées sur le serveur.

Nous commençons une session en utilisant la commande « *session_start()* » puis en définissant les variables de la session. Pour illustrer, cela se fait de la manière suivante :

```
session_start() ;  
$_SESSION['variable1']="valeur1";  
$_SESSION['variable2']="valeur2";  
$_SESSION['variable3']="valeur3";
```

Nous avons mis ici en valeur le fait que nous pouvons enregistrer de multiples valeurs dans le tableau associatif des sessions. Mais il ne faut pas oublier le fait que le nom des variables ne doit pas être identique car là aussi, comme dans le cas des cookies, on remplacerait la valeur déjà existante.

Pour ne pas saturer le serveur avec des sessions ouvertes, il faut comme pour les cookies, terminer les sessions. Pour les fermer il faut tout d'abord détruire toutes les variables de session puis détruire la session telle qu'elle. Ainsi une session prend fin de la manière suivante :

Tout d'abord, nous effaçons toutes les variables de la session à détruire en mémoire.

```
$_SESSION = array();
```

Ensuite nous détruisons la session.

```
session_destroy();
```

d. Compatibilité entre PHP et bases de données

Une manière plus générale de transmission de données entre plusieurs pages est de les garder dans une base de données (BDD). Pour ceci nous avons juste besoin d'un identifiant qui nous soit propre pour garder notre authentification (souvent un cookie) mais la majorité des données sont gardées dans la BDD.

Elles permettent de stocker bien plus de renseignements que ne pouvait le faire un cookie et sont sécurisées par un mot de passe, un pseudonyme de connexion, et l'adresse du serveur qui contient la base de données.

Ainsi, en utilisant le logiciel qui a pour objet de gérer la BDD, le plus utilisé dans les milieux Internet (MySQL), nous utilisons les accès à la BDD de la manière suivante :

Connexion au serveur qui contient la base de données :

```
mysql_connect("adresse_du_serveur","nom_d'utilisateur","mot_de_passe");
```

Connexion à la base de données de notre choix :

```
mysql_select_db("nom_de_la_BDD");
```

Nous retenons les données dans une variable, elles ne peuvent cependant pas encore être directement interprétées :

```
$retour=mysql_query("requête_SQL");
```

Pour récupérer les données, nous devons transformer la variable récupérée précédemment en tableau associatif. Grâce à l'utilisation d'une instruction « *while* », chaque ligne récupérée dans la BDD est traitée avec une boucle. A chaque passage, l'instruction traite la ligne suivante, et s'il n'y en a pas, termine la boucle.

```
while ($tableau=mysql_fetch_array($retour)){  
    echo $tableau['ID'] . ' -- ' . $tableau['login'] . '<br />' ;  
}
```

Dans cet exemple, nous affichons les données ID et login parmi tout ce que l'on a récupéré en mettant à l'écart les autres données. Pour cela, on utilise le caractère de concaténation, le point ".". Ainsi, le résultat visuel sera de la forme : « ID – login ».

```
1 -- julien  
2 -- krzysztof
```

Enfin, on se déconnecte de la base de données :

```
mysql_close();
```

e. Le langage SQL*

α-La sélection de données

Le langage SQL sert tout d'abord à communiquer au Système de Gestion de Bases de Données pour dire quels sont les champs à faire parvenir jusqu'au client. Ce langage est utilisé en formulant des requêtes dont voici un exemple :

```
SELECT * FROM matableSQL
```

Ceci peut se traduire par : "Prendre tout ce qu'il y a dans la *table** qui s'appelle "matableSQL".

Analysons chaque terme de cette requête :

✚ « *SELECT* » : en langage SQL, le premier mot indique quel type d'opération doit faire MySQL. La requête *SELECT* demande à MySQL d'afficher ce que contient une table.

✚ « *** » : après le *SELECT*, nous devons indiquer quels champs MySQL doit récupérer dans la table. Si nous ne sommes intéressés que par les champs "ID" et "login", il faudra taper :

```
SELECT ID, login FROM matableSQL;
```

Si nous voulons prendre tous les champs, nous tapons « *** ». Cette petite étoile peut se traduire par « tout » : « Prendre tout ce qu'il y a... »

✚ « *FROM* » : c'est un mot de liaison. Ca se traduit par « dans ». *FROM* fait la liaison entre le nom des champs et le nom de la table.

✚ « *matableSQL* » : c'est le nom de la table dans laquelle il faut aller piocher.

Pour trier et ordonner les recherches, nous pouvons ajouter certains critères de sélection grâce aux mots-clés « *WHERE* », « *ORDER BY* » et « *LIMIT* ». Ces critères sont placés en fin de commande.

WHERE : signifie « lorsque » et permet de ne sélectionner que les lignes (une ou plusieurs, dans une BDD elles sont généralement appelées les entrées) des BDD qui correspondent à certains critères définis selon cet exemple : « *WHERE login="un_login"* »

ORDER BY : signifie « ordonner en fonction de » et sert à ranger les données. Par exemple : « *ORDER BY dateDInscription* »

LIMIT : permet de ne sélectionner qu'une partie des résultats. Il prend deux paramètres :

- ✚ L'entrée à partir de laquelle on commence la sélection
- ✚ Le nombre d'entrées à sélectionner

Ainsi : *LIMIT 4, 10*

Permet de sélectionner les dix résultats à partir du cinquième inclus, puisque le premier élément correspond à 0... et le cinquième à 4.

En combinant toutes les options présentées ci-dessus nous obtenons une requête de la forme :

```
SELECT * FROM matableSQL WHERE diplome="DUT" ORDER BY dateDObtention LIMIT 0, 47;
```

Mais il faut bien compléter cette BDD avant de chercher des renseignements dedans. La partie qui suit est donc dédiée à la présentation des manières d'insertions de données.

β-L'insertion de données

Pour ajouter des entrées à une table existante, nous utilisons l'instruction « INSERT INTO », qui s'utilise de la sorte :

```
INSERT INTO nomTable(liste des champs à remplir) VALUES(valeurs à placer dans les champs);
```

ou

```
INSERT INTO nomTable VALUES(valeurs à placer dans les champs);
```

Nous pouvons bien évidemment également modifier des données dans des entrées déjà existantes :

```
UPDATE table SET champ='valeur' WHERE champ2='valeur2';
```

Cependant si les conditions sont trop vagues cette commande peut changer plusieurs entrées de la table.

Autre possibilité : créer une nouvelle table : pour cela nous utilisons l'instruction CREATE TABLE de la façon suivante :

```
CREATE TABLE nom_table (  
    id INT not null AUTO_INCREMENT,  
    prenom VARCHAR (50) not null,  
    nom VARCHAR (50) not null,  
    dateDeNaissance DATE not null,  
    autreInfo VARCHAR (90) not null,  
    PRIMARY KEY (id)  
)
```

Nous pouvons remarquer les noms de *champs** dans cet exemple, au début de chaque ligne. Pour chaque champ il faut spécifier le type de donnée qui va être stockée (INT : entier , VARCHAR (50) : chaîne d'au plus 50 caractères , DATE : format de date) ainsi que certaines spécificités (NOT NULL : doit être définie, ne doit pas être laissée vide ; AUTO_INCREMENT : on peut laisser ce champ vide car la valeur sera affectée automatiquement grâce à un compteur inclus dans la table qui s'incrémente de 1 à chaque ajout de données dans la table (pas les modifications de données déjà existantes). Enfin l'attribut PRIMARY KEY évite qu'il n'y ait deux valeurs identiques pour le champ sur lequel on l'applique dans la base.

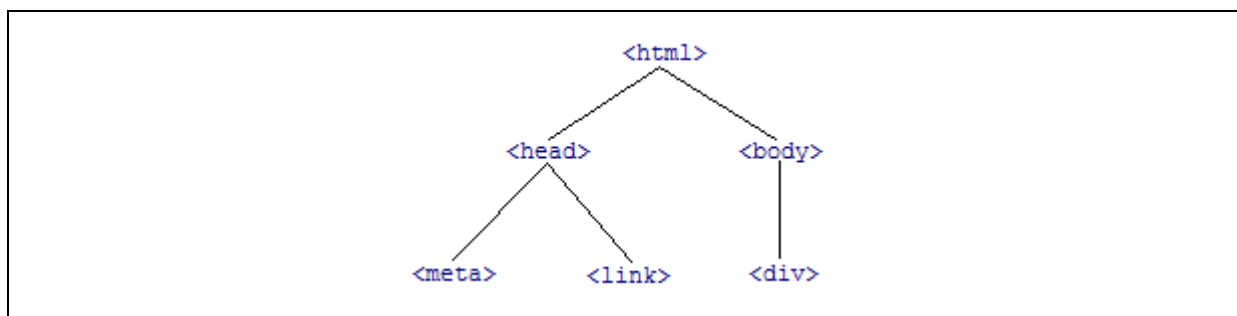
3. JavaScript

a. Le DOM*

Avant de présenter le langage, nous avons choisi d'exposer un outil utilisé par JavaScript. Les fonctions JavaScript agissent en général sur le **DOM** (*Document Object Model*). Le DOM a été standardisé par le *W3C** (World Wide Web Consortium), une organisation qui a été créée pour promouvoir la compatibilité des technologies web. Le DOM décrit la structure des documents HTML et XML* de façon à ce qu'ils puissent être manipulés via un navigateur Web et permet à des *scripts** d'accéder ou de mettre à jour le contenu, la structure ou le style d'un document, d'une page internet. Au fil du temps, le DOM a été standardisé autour de JavaScript et permet de représenter un document sous la forme d'un *arbre logique**. Cet arbre reprend, à partir de la balise `<html>`, l'ordre de construction de la page internet.

Chaque élément généré à partir d'un balisage (un paragraphe, un titre, un champ de texte dans un formulaire) forme un *nœud**. Ceci correspond au DOM de niveau 1. Voici l'exemple simple d'un arbre logique représentant une page internet

```
<html lang="fr" xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta content="text/html; charset=UTF-8" http-equiv="content-type"/>
    <link media="screen" type="text/css" href="DSTP-IC3-v1.css" rel="stylesheet">
  </head>
  <body>
    <div class="page">
  </body>
</html>
```



La balise `<html>` a deux « descendants » directs, `<head>` et `<body>`. La balise `<head>` a également deux « descendants » directs, `<meta>` et `<link>`. La balise `body` n'a qu'un seul descendant direct, `<div>`. La balise `<link>` est donc un « descendant » indirect de la balise `<html>`. Ainsi, en JavaScript, si nous effectuons une action sur la balise `<body>`, nous effectuerons une action non seulement sur la balise `<body>` mais également sur tous ses descendants directs.

Le DOM de niveau 2 permet plusieurs évolutions et notamment d'identifier plus rapidement un nœud ou un groupe de nœuds au sein d'un document avec la méthode « *getElementById()* ». Celle-ci prend en paramètre l'identifiant d'un bloc, ce que nous avons vu dans la partie HTML (II.1.a, page 8). Cette fonction permet d'éviter d'écrire le chemin complet du nœud dans l'arborescence. Elle permet également, lorsque l'arborescence du code change, de ne pas avoir à changer le chemin du nœud dans le code JavaScript.

b. Le langage JavaScript

JavaScript est un langage de programmation complètement lié au HTML, son code est intégré directement dans le code HTML. Contrairement au langage PHP, c'est le navigateur qui interprète le langage JavaScript, et non le serveur web.

JavaScript est considéré comme un « langage *objet** et événementiel ». Cela signifie que nous pouvons créer des objets sur une page, avec des propriétés et des *méthodes**, et leur associer des actions en fonction d'événements déclenchés par le visiteur (passage de souris, clic ...).

Le JavaScript est inséré dans le code d'une page web grâce aux balises `<script>` et `</script>`. La balise `<script>` prend l'attribut « `type` » afin de savoir de quel langage de programmation se trouve entre les balises.

```
<script type="text/javascript">  
// c'est ici que nous mettrons le code JavaScript  
</script>
```

JavaScript utilise également des variables qui sont déclarées avec l'instruction « `var` ». Il existe trois types de variables, les variables « basiques », les variables de type « objet » et les tableaux. Nous ne parlerons que des deux premières car nous n'avons pas utilisé de tableaux lors de notre projet.

✚ Les variables « basiques » :

Il s'agit des variables de type entier, chaîne de caractères, structures ... contrairement à d'autres langages, les variables n'ont pas besoin d'être typées. C'est-à-dire qu'il n'est pas nécessaire de préciser que la variable que nous déclarons est un entier ou un caractère. JavaScript fait la concordance lui-même.

```
var nom_projet="GoogleMap";
```

De plus, comme nous pouvons le voir sur cet exemple, chaque instruction en JavaScript se finit par « `;` ».

✚ Les variables « objets » :

Un objet représente une entité en JavaScript. Il rassemble des « méthodes », qui peuvent être comparées à des fonctions internes à l'objet. La déclaration de ce type de variable se fait toujours avec l'instruction « **var** ». Pour créer un objet il faut utiliser le mot clé « **new** » suivi du type d'objet.

```
var une_date=new Date(annee,mois,jour,heure,min);
```

Cette instruction permet de créer un objet de type « **Date** ». Cet objet prend en paramètre cinq variables, l'année, le mois, le jour et l'heure.

c. Les fonctions JavaScript

Le langage JavaScript permet aussi de déclarer des *fonctions**, qui pourront être ensuite appelées à n'importe quel endroit de la page. Pour déclarer une fonction, on utilise le mot clé « **function** » suivi de son nom, des paramètres éventuels et enfin des instructions entre accolades.

```
function ma_fonction(parametre1, parametre2)
{
  //liste d'instructions
}
```

Avec JavaScript, nous avons donc vu que nous pouvions effectuer des actions sur le DOM. Par exemple :

```
document.getElementById("zonedetexte").value="Ce texte est écrit grâce au Javascript";
```

L'élément du DOM qui est modifié est :

```
document.getElementById("zonedetexte")
```

Il s'agit du bloc contenu dans le « document », c'est-à-dire dans la page internet, ce bloc ayant l'identifiant « **zonedetexte** »

Dans cet exemple, c'est l'attribut « **value** » de la zone de texte qui est modifié et qui prend pour valeur la chaîne de caractères qui suit.

```
.value="Ce texte est écrit grâce au Javascript";
```

Avec JavaScript, nous pouvons également exécuter des fonctions ou actions sur le DOM lors d'événements. Les événements sont des actions de l'utilisateur sur la page, par exemple un clic de souris.




Ce sont les gestionnaires d'événements qui permettent d'associer une action avec un évènement. Pour créer un gestionnaire d'évènement, il faut ajouter un attribut à la balise sur laquelle devra être effectué l'évènement.

Le nom de l'attribut correspond au type de l'évènement tandis que la valeur de l'attribut correspond à l'action JavaScript ou à la fonction à effectuer. Par exemple :

```
<span onMouseOver="fonction1()"> bloc sur lequel s'applique l'évènement </span>
```

Ainsi dans cet exemple, à chaque fois que l'utilisateur passe sa souris sur le bloc ``, la fonction « `fonction1` » est appelée.

En effet, l'évènement « `onMouseOver` » correspond au passage d'une souris. Il existe un certain nombre d'évènements prédéfinis. Ceux que nous avons utilisés pour réaliser notre site sont :

-  `onLoad` : Se produit lorsque le navigateur de l'utilisateur charge la page en cours.
-  `onUnload` : Se produit lorsque le navigateur de l'utilisateur quitte la page en cours.
-  `onSubmit` : Se produit lorsque l'utilisateur clique sur le bouton de soumission d'un formulaire

*d. Les fonctions anonymes**

Les fonctions anonymes sont apparues dans le langage Lisp, l'un des tous premiers langages de programmation. Ce sont des fonctions, comme leur nom l'indique, qui ne sont pas identifiées par un nom. Elles sont créées à la volée, quand le besoin s'en fait sentir. L'existence d'une fonction anonyme dépend de données inconnues au moment de l'écriture mais qui seront créées par le programme lors de l'exécution.

Celles-ci sont en général créées pour des fonctions qui attendent d'autres fonctions en paramètres. Cela permet d'alléger le code et se révèle très pratique pour des fonctions qui ne sont utilisés qu'une seule fois dans le programme. Nous pouvons également les stocker dans des variables pour les réutiliser par la suite

Voici un exemple de fonction anonyme :

```
var maFonction = function(message)
{
    alert(message);
}
```

Nous stockons dans la variable « *maFonction* » la fonction anonyme qui prend en paramètre « *message* » et qui affiche à l'écran ce paramètre grâce à la méthode « *alert* ». Pour utiliser cette fonction il suffit d'exécuter l'instruction :

```
maFonction('Bonjour');
```

Nous aurions très bien pu créer une fonction s'appelant « *maFonction* » contenant la méthode « *alert* » mais nous souhaitons simplement montrer comment utiliser une fonction anonyme en JavaScript.

Le langage JavaScript permet donc la gestion des événements et leur insertion dans le code HTML, la création de variables mais aussi de fonctions et fonctions anonymes.

JavaScript est également utilisé par la technologie AJAX, qui permet de simplifier un site internet en faisant faire certaines actions au navigateur et non plus au serveur. Cette technologie nous a aussi été utile afin de réaliser notre site.

4. AJAX

AJAX¹ est un acronyme désignant une méthode informatique de développement d'applications Web. Ce n'est pas une technologie à proprement dit mais un terme évoquant un ensemble de technologies web, telles que XHTML, CSS, JavaScript, XML ...

Les applications web traditionnelles permettent aux utilisateurs d'effectuer des actions (suivre un lien, valider un formulaire ...). Le serveur reçoit alors des données et renvoie une nouvelle page. Ce fonctionnement utilise inutilement de la *bande passante**, une grande partie du code XHTML étant commune aux différentes pages. Une requête HTTP* doit être réalisée à chaque interaction avec l'application. Le temps de réponse de l'application dépend donc fortement du temps de réponse du serveur, ce qui conduit à des interfaces utilisateurs plus lentes que si elles étaient effectuées en local.

En utilisant AJAX, les applications peuvent envoyer des requêtes au serveur HTTP pour récupérer uniquement les données nécessaires. Il s'agit de la requête HTTP *XMLHttpRequest*. Du côté client, nous utilisons le langage JavaScript pour traiter ces données. Les applications sont alors plus réactives, la quantité d'informations échangées entre le navigateur et le serveur http étant beaucoup moins importantes. L'avantage principal de cette méthode est la vitesse à laquelle une application AJAX répond aux actions de l'utilisateur. Ces actions sont traitées localement par le navigateur et ne nécessitent donc pas l'envoi de requêtes vers le serveur.

¹ Asynchronous JavaScript And XML

L'autre fonctionnalité d'AJAX est la possibilité d'analyser et de travailler avec des documents XML. Le langage XML (eXtensible Markup Language) est un langage de balisage généraliste recommandé par le W3C pour servir de modèle à toutes sortes de langage de balisage. Il est principalement utilisé pour le partage de données entre différents systèmes. Dans notre projet, nous l'avons utilisé pour « échanger » des données entre une base de données et la fonction JavaScript qui les traite (III.1.c, page30). Voici un exemple de structure de fichier XML.

```
<markers>
<marker lat="3.12" lng="45.81">
<marker lat="2.34" lng="50.03">
...
</markers>
```

Nous définissons une balise ouvrante « `<marker>` » et une balise fermante « `</marker>` », tout comme dans le HTML.

Dans ces balises, il y a des « éléments », dans ce cas les balises « `<marker ... >` », et ces éléments contiennent des « attributs », ici « `lat` » et « `lng` », pour la latitude et la longitude géographique.

JavaScript permet de récupérer ces informations. Il existe deux méthodes qui s'appliquent à des documents XML, mais avant tout, il faut assigner le fichier XML à une variable. Il existe différentes méthodes mais voici celle dont nous nous sommes servis durant notre projet, celle-ci étant fournie par Google Map.

```
var xml = GXml.parse(data);
```

La méthode « `getElementsByTagName()` » permet de récupérer un élément de l'objet XML. Par exemple :

```
var markers = xml.documentElement.getElementsByTagName("marker");
```

Dans ce cas, la variable « `markers` » sera un tableau étant donné qu'il y a plusieurs éléments dont le nom est `marker`.

A partir de cet élément, nous pouvons récupérer l'attribut grâce à la méthode « `getAttribute()` ». Par exemple, pour récupérer l'attribut « `lat` » du premier élément `marker`, il faudra exécuter l'instruction :

```
markers[0].getAttribute("lat");
```

Nous avons utilisé ces deux méthodes dans le JavaScript qui a servi à incorporer l'API Google Map dans notre site. Après avoir montré les technologies nécessaires à notre site internet, nous allons maintenant expliquer comment nous les avons mises en place

III. La mise en place du site

1. L'API Google Map

Une API est une interface de code source fournie par un éditeur et qui permet de recourir aux fonctions d'un logiciel depuis un autre logiciel. Dans notre cas, il s'agira de recourir aux fonctionnalités de Google Map depuis notre page web.

Google Map est un service gratuit de carte géographique et de plan en ligne. Il permet à un utilisateur, à partir d'une carte mondiale, de pouvoir zoomer jusqu'à l'échelle d'une rue. Nous pouvons visualiser la carte sous deux formes, un plan classique avec nom des rues, quartiers, villes et un plan en image satellite.

Comme nous l'avons dit précédemment, l'API Google Map est écrite en JavaScript. Afin de pouvoir utiliser cette API, il faut incorporer les fonctions fournis par Google. Pour cela, dans le HTML, on charge le code source grâce aux balises « <script> ».

```
<script src="http://maps.google.com/maps?file=api&v=2&key=abcdefg"
      type="text/javascript">
</script>
```

C'est dans ce code que sont chargées tous les objets et toutes les méthodes de Google Map. Ceux-ci étant présentés dans la documentation en ligne de Google Map.

a. La documentation

Tout d'abord, celle-ci est écrite en anglais, il nous a fallu faire un effort de traduction afin de comprendre cette documentation. Cependant, faisant tous les deux notre stages dans un pays anglophone, ce fut une bonne expérience car dans le futur, nous serons certainement amené à lire voire rédiger des documentations en anglais.

De plus, la documentation est beaucoup basée sur des exemples. Au début, nous avons testé ces exemples pour comprendre comment fonctionnait l'API. Ensuite, nous avons mis en place nos propres fonctions pour réaliser ce que nous voulions. Nous avons également cherché des informations sur différents forums afin de compléter nos connaissances et résoudre certains « *bugs** » que nous avons dans notre site.

Nous allons maintenant présenter les principales fonctionnalités utiles pour afficher une carte et des marqueurs sur une page internet.

b. Les cartes

✚ Insérer une carte sur une page :

Il faut tout d'abord créer un « bloc » qui contiendra la carte à l'aide de la balise « `<div>` » et lui définir une taille en attribut grâce à un attribut de style. Ce code est inséré dans le « `<body>` » du HTML.

```
<div id="map" style="width: 1000px; height: 500px"></div>
```

Ensuite, dans le JavaScript, nous devons créer une fonction, qui sera appelée au chargement de la page, et qui crée la carte.

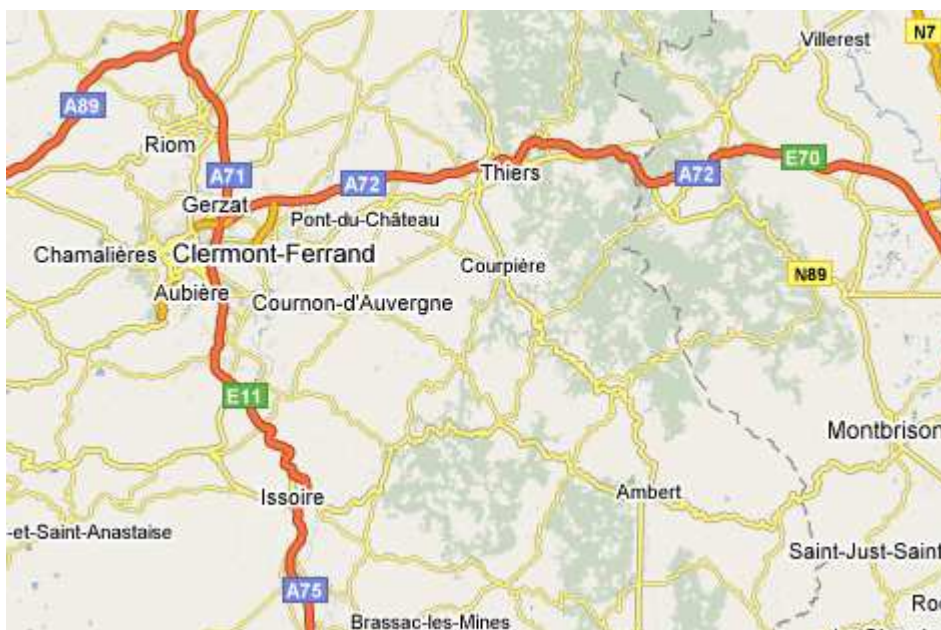
Dans cette fonction, nous créons un objet « GMap2 » que nous appelons « *map* » et mettons en paramètre le bloc que nous venons juste de créer. Comme nous l'avons expliqué, GMap2 est un objet défini dans le code source fourni par Google.

```
var map = new GMap2(document.getElementById("map"));
```

L'objet GMap2 contient plusieurs méthodes. Afin d'initialiser la carte, il faut se servir de la méthode « *setCenter* ».

```
map.setCenter(new GLatLng(45.756505, 3.111534), 6);
```

Cette méthode prend deux paramètres, un objet de type « *GLatLng* », qui correspond à des coordonnées (latitude et longitude), et un nombre qui correspond à un niveau de *zoom**. En appliquant cette méthode à la carte, nous la centrons sur les coordonnées données en premier paramètre et le zoom par défaut est celui donné en deuxième paramètre.



Exemple de carte crée par Google Map¹

Cependant, pour toutes les méthodes utilisant l'API Google Map, il faut tester si le navigateur est compatible avec l'application avant d'insérer la carte, c'est pourquoi toutes nos fonctions utilisant Google Map seront entourées d'une condition :

```
if (GBrowserIsCompatible()) {  
  //liste d'instructions  
}
```

Enfin, il reste à appeler la fonction JavaScript au chargement de la page.

```
<body onload="load()" onunload="GUnload()">
```

La fonction « *GUnload* » termine la communication avec Google Map lorsque l'utilisateur quitte la page.

¹ Source Personnelle

✚ Ajouter des contrôles sur la carte :

Notre carte apparait maintenant sur notre page web, mais ce n'est pas très attractif. En effet, à ce point-ci, nous ne pouvons ni bouger la carte, ni changer l'échelle. Google Map permet ces contrôles, ce sont en fait des méthodes qui sont associés à l'objet GMap2, « *addControl* »

```
map.addControl(new GSmallMapControl());  
map.addControl(new GMapTypeControl());
```

L'objet « *GSmallMapControl* » permet de se déplacer sur la carte mais également de zoomer/dézoomer. L'objet « *GMapTypeControl* » permet de changer le type de la carte, mode classique, mode satellite ou mixte.



✚ Les gestionnaires d'événements

Pour créer un gestionnaire d'événement, il faut appeler la méthode « *GEvent.addListener* », avec comme paramètres l'élément sur lequel « l'écouteur » sera placé, le type de l'événement et l'action à effectuer. Par exemple :

```
GEvent.addListener(mark, "drag", function(){  
    document.getElementById("latitude").value=this.getPoint().lat();  
    document.getElementById("longitude").value=this.getPoint().lng();  
});
```

Nous pouvons voir que l'événement se déclenchera sur l'élément mark, qui est en fait un marqueur, dont nous parlerons après. Le type de l'événement est « drag », cet événement correspond à un déplacement d'un marqueur. L'action à effectuer est une fonction générée « à la volée » car elle n'est utilisée qu'à cet endroit du code. Cet exemple est une bonne illustration de l'utilisation des fonctions anonymes.

Lorsque l'utilisateur déplace un marqueur, les zones de texte ayant les id « latitude » et « longitude » prennent les valeurs de la latitude et de la longitude courante du marqueur.

c. Les marqueurs



Un marqueur permet de représenter des coordonnées sous la forme d'une petite icône. Les icônes sont fournies par défaut mais il est possible de créer ses propres icônes, ce que nous effectuons plus bas dans cette partie.

La méthode de base pour ajouter un marqueur sur une page est « *GMap2.addOverlay* ». Cette méthode est simple à utiliser mais présente des inconvénients. Dans le cas où la carte contient beaucoup de marqueurs, la méthode « *addOverlay* » ralentit le chargement de la page et à certaines échelles, certains marqueurs ne pourront pas s'afficher.

α. Les gestionnaires de marqueur

Un « *Marker Manager* », gestionnaire de marqueurs, liste d'objets et de méthodes gérant les marqueurs, propose des solutions à ces problèmes mais aussi des méthodes pour récupérer les coordonnées des marqueurs, ce qui n'est pas possible avec un simple « *addOverlay* ».

Pour utiliser un gestionnaire de marqueurs, il faut créer un objet « *GMarkerManager* » et lui passer une carte en paramètre.

```
var mgr = new GMarkerManager(map);
```

Nous pouvons également spécifier une liste d'options pour améliorer les performances.

- ✚ *maxZoom* : le niveau maximum du zoom supporté par le gestionnaire de marqueurs. La valeur par défaut est la valeur maximale supportée par Google Map.
- ✚ *borderPadding* : Cette option définit une marge en *pixel**. Tous les marqueurs situés à cette distance, en dehors de la carte, seront quand même affichés sur celle-ci, sur ses bords.
- ✚ *trackMarkers* : spécifie si oui ou non les mouvements d'un marqueur doivent être suivis par le gestionnaire. En mettant cette option à « *true* », cela permet d'avoir accès aux méthodes « *setPoint* », qui permet de définir les coordonnées d'un marqueur, et « *getPoint* », qui permet de les récupérer et qui nous servira plus tard dans le site (III.2.d, page36).

Pour spécifier cette liste d'options, nous pouvons les mettre directement en paramètre du « *GMarkerManager* » ou alors créer une variable qui contient ces options et la mettre, de même, en paramètre du « *GMarkerManager* ».

```
var mgrOptions = { borderPadding: 50, maxZoom: 15, trackMarkers: true };  
var mgr = new GMarkerManager(map, mgrOptions);
```

Il nous faut maintenant ajouter des marqueurs à ce « *Marker Manager* ». Tout d'abord il faut créer un objet qui correspond aux coordonnées du marqueur, « *GLatLng* », le même type d'objet qui sert pour centrer la carte. Cet objet est lui-même passé en paramètre d'un autre objet, « *Gmarker* », qui correspond au marqueur.

Une des options du « *GMarker* » permet de déplacer les marqueurs « *draggable: true* ».

En mettant un nouvel objet en paramètre, nous pouvons également changer la forme du marqueur. Cet objet est de type « *GIcon* » et possède plusieurs attributs.

```
var icon = new GIcon();
```

Il faut tout d'abord attribuer une image « *image* » et une ombre « *shadow* » à cette icône :

```
icon.image = "http://labs.google.com/ridefinder/images/mm_20_red.png";  
icon.shadow = "http://labs.google.com/ridefinder/images/mm_20_shadow.png";
```

Dans ce cas, nous avons choisi une icône fournie par Google Map. 

Ensuite, il faut définir une taille à cette image et à cette ombre grâce à l'objet « *GSize* » :

```
icon.iconSize = new GSize(12, 20);  
icon.shadowSize = new GSize(22, 20);
```

Et Enfin, définir un point d'ancrage pour l'icône, qui sert à relier l'image à ombre. :

```
icon.iconAnchor = new GPoint(6, 20);
```

Notre nouvelle icône est maintenant créée. A partir de ce point, si nous souhaitons afficher un marqueur avec la nouvelle icône, il faudra appeler celle-ci en paramètre du « *GMarker* ».

```
var mark= new GMarker(point, icon);
```

Nous pouvons donc créer des marqueurs, avec différentes icônes et différentes options.

Maintenant, afin d'ajouter un marqueur au « *Marker Manager* », il faut appeler la méthode « *addMarker* » avec en paramètre l'objet « *GMarker* ». Le deuxième paramètre correspond au niveau de zoom à partir duquel le marqueur devra s'afficher. « 0 » signifie que le marqueur s'affiche pour tous les zooms.

```
var point = new GLatLng(45.345, 3.287);  
var mark= new GMarker(point);  
mgr.addMarker(mark, 0);
```

Enfin, une fois que tous les marqueurs ont été ajoutés au « *Marker Manager* », il faut mettre à jour celui-ci avec la méthode « *refresh* ».

```
mgr.refresh();
```

Les marqueurs créés s'afficheront maintenant sur la carte.

β. La récupération des marqueurs

Google Map offre une fonctionnalité intéressante en ce qui concerne l'affichage dynamique de marqueurs, la méthode « GDownloadURL ». Celle-ci permet de télécharger des données de type XML qui contiennent une liste de latitudes et longitudes géographiques. Une fois les données téléchargées, nous créons un marqueur pour chacune des coordonnées récupérées.

Les données peuvent être récupérés via un fichier statique XML, comme dans l'exemple que nous allons développer, ou alors générés par un script PHP, comme ce que nous avons fait dans notre projet (III.2.f, page 38).

Nous allons traiter la récupération à partir d'un exemple :

```
GDownloadUrl("marker.xml", function(data) {  
    var xml = GXml.parse(data);  
    var markers = xml.documentElement.getElementsByTagName("marker");  
    for (var i = 0; i < markers.length; i++) {  
        var point = new GLatLng(  
            parseFloat(markers[i].getAttribute("lat")),  
            parseFloat(markers[i].getAttribute("lng")));  
        var mark= new GMarker(point);  
        mgr.addMarker(mark, 0);  
    }  
    mgr.refresh();  
});
```

Le premier paramètre de cette fonction, « marker.xml », est le fichier statique XML qui contient les informations sur les marqueurs, c'est ce fichier qui sera traité dans la deuxième partie de la méthode. Nous nous servons dans cette méthode des propriétés d'AJAX que nous avons vu en partie II.4, page 24.

```
var xml = GXml.parse(data);  
var markers = xml.documentElement.getElementsByTagName("marker");
```

Dans celle-ci, nous récupérons le fichier XML comme vu précédemment et à l'aide d'une boucle, nous traitons chaque marqueur un par un. Cette boucle continue tant que tous les marqueurs n'ont pas été traités.

```
for (var i = 0; i < markers.length; i++) {
```

Ensuite, à chaque marqueur, nous créons une variable qui contient les coordonnées du marqueur. Ces coordonnées sont récupérées dans le fichier XML.

```
var point = new GLatLng(  
    parseFloat(markers[i].getAttribute("lat")),  
    parseFloat(markers[i].getAttribute("lng")));
```

Après ceci, il faut créer le marqueur en lui donnant les coordonnées précédentes et l'ajouter au gestionnaire de marqueurs.

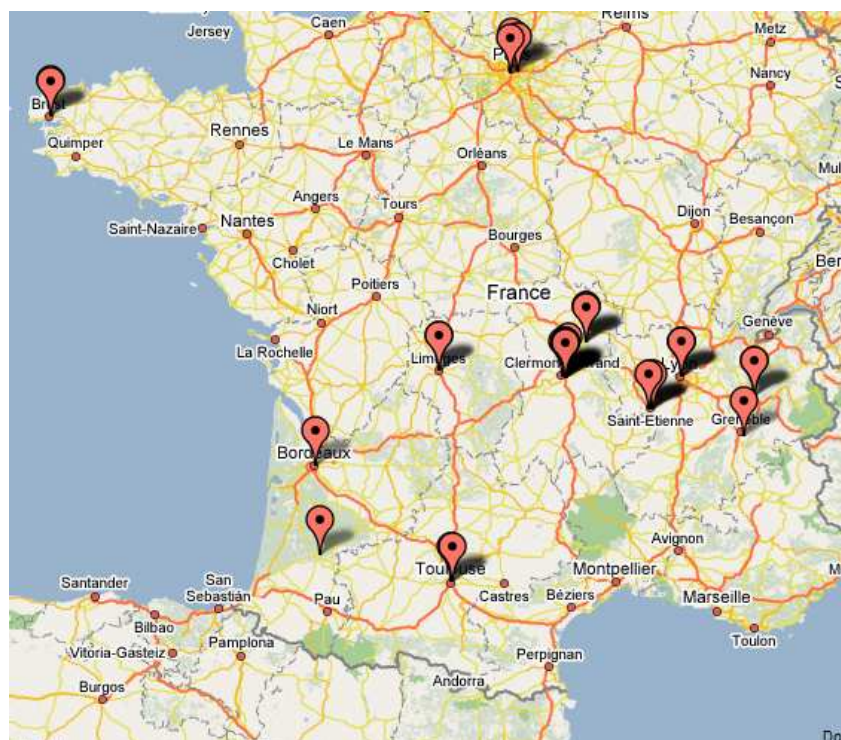
```
var mark= new GMarker(point);  
mgr.addMarker(mark, 0);
```

Enfin, une fois la boucle finie, nous mettons à jour le gestionnaire de marqueurs afin qu'il affiche tous les marqueurs qui lui ont été ajoutés.

```
mgr.refresh();
```

Dans notre site nous n'avons pas utilisé de fichier statique mais nous avons fait en sorte que les données soient générées à chaque appel à la fonction « *GDownloadUrl* » grâce à un fichier en PHP, « *marker.php* », que nous verrons dans la partie suivante (III.2.b.).

Voici à quoi ressemble notre carte une fois tous les marqueurs de la base de données téléchargés.



Exemple de cartes avec marqueurs¹

¹ Source Personnelle

2. La partie client

Dans cette partie nous décrivons et détaillons tous les fichiers composant la partie cliente de notre site web.

Pour commencer, nous montrons le fichier qui s'exécute en premier lorsqu'on entre juste l'adresse du serveur, « index.php ».

a. index.php

Ce fichier commence par vérifier si les variables « \$_POST['login'] » et « \$_POST['pass'] », envoyées potentiellement par le formulaire de la page default.php, sont définies : si ces dernières sont bien définies et correspondent à une personne inscrite dans la base de données, nous créons une session et affichons la page propre à l'utilisateur logué, si elles sont bien définies mais ne correspondent à aucun enregistrement de la base de données nous affichons la page par défaut complétée d'un message de mauvaise authentification, si encore ces variables ne sont pas définies, nous affichons la page par défaut.

Pour récupérer le login et le stocker dans une session, nous utilisons les instructions suivantes :

```
$_SESSION['login']=$_POST['login'];  
$login=$_SESSION['login'];
```

Pour vérifier si la personne existe bien, on utilise la requête SQL :

```
SELECT COUNT(*) AS nb_rep FROM correspondances WHERE login='$login'
```

Si cette valeur vaut 1 alors nous créons une variable « \$test=1 » sinon « \$test=0 ».

```
$requete=mysql_query("SELECT COUNT(*) AS nb_rep FROM correspondances  
                      WHERE login='$login'");  
$resultat=mysql_fetch_array($requete);  
if($resultat['nb_rep']==1)$test=1;  
else $test=0;
```

C'est en fonction de la variable « \$test » que nous voyons si l'utilisateur existe dans la base de données ou pas. S'il y est bien présent nous comparons le mot de passe du formulaire avec le mot de passe de la base de données, pour comparer nous utilisons la fonction « crypt() », que nous présenterons dans la partie III.3.a, page 40.

Si l'utilisateur n'est pas présent, nous annonçons que l'utilisateur est mal logué.

```
        if($test==1){
            if(crypt($pass,$login)==$resultat['password']){
                require("../page_logge.php");
                exit;
            }
            else{
                require("../mal_logge.php");
                exit;
            }
        }

        else
        {
            require("../mal_logge.php");
            exit;
        }
    }
}
```

b. Debut.php

Ce fichier est un entête pour les différents fichiers qui affichent la carte Google Map : default.php, page_logge.php et mal_logge.php.

Debut.php contient le début de l'entête HTML : les méta-variables et le début des scripts en ce qui concerne la carte : l'inclusion de la page de scripts, et le début de la fonction load (la fin de cette fonction que nous lancerons au chargement de la page varie en fonction de l'authentification(ou pas) de la personne).

```
<?
echo '
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
<title>Suivi des etudiants de la promo RT 2006</title>
<script
src="http://maps.google.com/maps?file=api&v=2&key=AB
    type="text/javascript"></script>
<script type="text/javascript">
function load() {
    if (GBrowserIsCompatible()) {
        var map = new GMap2(document.getElementById("map"));
        map.addControl(new GSmallMapControl());
        map.addControl(new GMapTypeControl());
        map.setCenter(new GLatLng(45.756505,3.111534), 6);
        var mgrOptions = { borderPadding: 50, maxZoom: 15, trackMarkers: true };
        var mgr = new GMarkerManager(map, mgrOptions);
    }
};
?>
```

c. Default.php

Dans ce fichier nous poursuivons la définition de la fonction load() qui se lance au chargement de la page.

Cette page présente tous les différents marqueurs avec la même icône. Nous laissons au navigateur du client la possibilité d'aller chercher les marqueurs dans le fichier marker.php pendant l'exécution de la fonction load(), dans la méthode « GDownloadUrl ».

```
for (var i = 0; i < markers.length; i++) {  
    var point = new GLatLng(parseFloat(markers[i].getAttribute("lat")),  
        parseFloat(markers[i].getAttribute("lng")));  
    var mark= new GMarker(point);  
    mgr.addMarker(mark, 0);  
}
```

Dans cette boucle nous affichons chaque marqueur après le précédent et ce le nombre de fois que nous trouvons la balise marqueur dans le fichier généré par marker.php. Chacune des balises contient les champs « lat » et « lng » qui sont la latitude et la longitude correspondant au marqueur dans lequel ces attributs se trouvent. Tous les marqueurs affichés seront identiques.

d. Page_logge.php

Dans cette page l'utilisateur a dû correctement remplir le formulaire (login, mot de passe) et il est désormais capable de déplacer son propre marqueur (et ce seul marqueur).

Dans ce fichier nous devons différencier l'utilisateur logué des autres utilisateurs :

```
if(markers[i].getAttribute("id")==".$login."){  
    var point = new GLatLng(parseFloat(markers[i].getAttribute("lat")),  
        parseFloat(markers[i].getAttribute("lng")));  
    var mark= new GMarker(point, {draggable: true});  
    mark.enableDragging();  
    mgr.addMarker(mark, 0);  
    map.setCenter(new GLatLng(parseFloat(markers[i].getAttribute("lat")),  
        parseFloat(markers[i].getAttribute("lng")), 12);  
}
```

Les marqueurs sont repérés, dans le fichier généré par marker.php, par un identifiant qui est le pseudo de la personne à qui le marqueur correspond. Ainsi nous pouvons rechercher le login de la connexion au login dans les attributs des marqueurs. Nous autorisons alors de bouger cet unique marqueur et plaçons le centre de la carte sur ce marqueur.

En ce qui concerne tous les autres marqueurs, ils sont concernés par :

```
else {  
    var point = new GLatLng(parseFloat(markers[i].getAttribute("lat")),  
        parseFloat(markers[i].getAttribute("lng")));  
    var mark= new GMarker(point, icon);  
    mgr.addMarker(mark, 0);  
}
```

Nous changeons alors d'icône pour un autre plus petit et nous ne permettons pas à la personne de déplacer ce marqueur là. Ensuite nous plaçons le code même de la page : nous délimitons la carte, nous mettons un lien vers « delog.php » pour se déloguer et enfin nous mettons des zones de texte permettant à l'utilisateur de savoir à quelles coordonnées il a placé son marqueur dans le cas d'un changement.

Nous plaçons également un gestionnaire d'événement sur les marqueurs de la carte. Etant donné qu'un seul marqueur peut être déplacé, celui de l'utilisateur en cours, l'événement ne se déclenche que sur un mouvement de ce marqueur.

```
GEvent.addListener(mark, "drag", function(){  
    GDownloadUrl("changement.php?latitude="+this.getPoint().lat()+  
        "&longitude="+this.getPoint().lng()+"&login='.$login.'");  
    document.getElementById("latitude").value=this.getPoint().lat();  
    document.getElementById("longitude").value=this.getPoint().lng();  
});
```

Lorsque l'utilisateur déplacera son marqueur, événement « drag », la méthode GDownloadUrl est appelé avec en paramètre un fichier, « changement.php », que nous détaillons plus loin dans cette partie.

On envoie à ce fichier PHP trois variables, « latitude », qui prend pour valeur « this.getPoint().lat() », « longitude » qui prend pour valeur « this.getPoint().lng() » et login qui a pour valeur le login de l'utilisateur.

Les deux premières valeurs sont également envoyées vers les deux zones de texte possédant les id « latitude » et « longitude ».

La méthode « this.getPoint().lat() » permet de récupérer la latitude de l'objet en cours (« this »). Celui étant le marqueur qui est déplacé.

e. mal_logge.php

Cette page affiche exactement la même page que la page default.php avec pour seule différence la présence d'un message d'erreur d'authentification. On retrouve cette différence dans cette portion de code :

```
<body onload="load()" onunload="GUnload()">
<div id="map" style="width: 1000px; height: 500px"></div>
Login ou mot de passe incorrect, veuillez reessayer. Merci
```

Après le message on affiche le formulaire comme avec le fichier default.php.

f. marker.php

Ce fichier permet de générer à la volée des données lors du chargement de la méthode « GDownloadUrl ». En effet, à chaque appel à cette fonction, la page marker.php est appelée et génère des données de type XML. Ces données ne sont utilisées que dans cette fonction, elles ne sont stockées nulle part et ne sont donc pas visibles en dehors de la fonction. C'est ce qui rend cette méthode très intéressante car elle n'encombre ni le serveur ni le client.

Pour créer les données, la page se connecte à la base de données, sélectionne toute la table « correspondances » de façon ordonnée par les identifiants. Ensuite nous générons les données qui seront traités.

```
<?
mysql_connect('sql.free.fr','googlemap','████████');
mysql_select_db('googlemap');

$requete=mysql_query("SELECT * FROM correspondances ORDER BY id ");

echo "<markers> \n";
while($resultat=mysql_fetch_array($requete)){
    echo "<marker lat=\"" . $resultat['latitude'] . "\" lng=\"" . $resultat['longituede'] .
        "\" id=\"" . $resultat['login'] . "\"/>\n";
}
echo '</markers>';

mysql_close();
?>
```

Nous affichons les balises <marker> au début et </marker> à la fin. Entre les deux, nous récupérons les données de la requête SQL et pour chaque entrée de la table nous créons une ligne de la forme suivante :

```
<marker lat="3.12" lng="45.81" id="toto">
```

g. changement.php

Ce fichier, de façon similaire à marker.php, est appelé à chaque fois que le marqueur d'une personne est déplacé. Il est déclenché par JavaScript en arrière plan. L'utilisateur ne se rend pas compte qu'une action autre que son déplacement de marqueur est train de s'effectuer simultanément, et de façon invisible. Cette méthode permet de synchroniser quasi instantanément la position du marqueur sur la carte avec les coordonnées présentes dans la base de données. Lorsque nous le déposons à une place donnée, nous mettons alors à jour les données longitude et latitude de la base de données pour le login correspondant à la variable stockée dans la session.

Tout d'abord, nous récupérons les variables latitude et longitude

```
$latitude=$_GET['latitude'];  
$longitude=$_GET['longitude'];
```

Ensuite, nous effectuons la requête SQL :

```
$requete=mysql_query( "UPDATE correspondances SET latitude='$latitude',  
longitude='$longitude' WHERE login='$login'");
```

h. delog.php

Dans ce fichier, on rappelle la session, les variables de session sont supprimées, la session est détruite et pour finir l'utilisateur est renvoyé vers la page d'accueil avec la commande « header ». L'utilisateur est amené sur cette page quand il clique sur le lien « terminer » de « page_logge.php ».

```
session_start();  
$_SESSION = array();  
session_destroy();  
header("Location: ./index.php");
```

3. La partie administrateur

Dans cette partie, nous développons les pages de notre site qui servent à ajouter des utilisateurs dans la base de données ou à en supprimer, tout ceci à l'aide de formulaires, l'administrateur n'ayant pas à faire appel à des connaissances en SQL. Afin de permettre l'accès à cette partie administrateur nous avons rajouté un lien HTML vers la page ajout.php grâce à la balise « <a> ».

```
<a href="ajout.php"> Acces Administrateur </a>
```

Nous développons la page ajout.php dans le b. de cette partie mais nous étudions d'abord pourquoi nous avons choisi de *crypter** les mots de passe, non seulement lors de l'ajout de nouveaux utilisateurs par l'administrateur, mais également ceux déjà présents dans la base de données.

a. Cryptage des mots de passe

Nous avons choisi de mettre en place un cryptage des mots de passe contenus dans la base de données afin de sécuriser notre site. En effet, si une personne avait accès frauduleusement à notre base de données, elle ne pourrait pas connaître les mots de passe des utilisateurs car ceux-ci seraient cryptés.

Au début de notre projet, nous avons ajouté des utilisateurs dans la base de données manuellement, nous n'avions donc pas crypté les mots de passe. Nous avons du créer une page en PHP faisant ceci, à la fois pour la table que nous avons appelé « *correspondances* », qui contient les informations sur les utilisateurs enregistrés avec leur login, mot de passe et coordonnées, et pour la table « *admin* » qui contient les login et mot de passe des administrateurs. La structure de ces deux pages est identique.

Nous nous connectons à la base de données et nous faisons une boucle correspondant au nombre d'entrées dans la table.

```
mysql_connect('sql.free.fr','googlemap','██████████');  
mysql_select_db('googlemap');  
for ($i=1;$i<40;$i++) {  
    //...  
}
```

A chaque passage de la boucle nous effectuons le traitement suivant :

```
$requete=mysql_query("SELECT * FROM correspondances WHERE `id` = $i");  
$resultat=mysql_fetch_array($requete);  
$pass=$resultat['password'];  
$log=$resultat['login'];
```

Tout d'abord, nous faisons la requête SQL qui permet de récupérer le mot de passe de l'utilisateur ainsi que le login. Il faut ensuite crypter ce mot de passe et le stocker dans la base de données à la place de l'ancien.

```
$passc=crypt($pass,$log);  
$requete=mysql_query("UPDATE `correspondances` SET `password` = '$passc'  
WHERE `id` = $i LIMIT 1");
```

La fonction qui permet de crypter est donc la fonction « *crypt()* ». Celle-ci permet d'effectuer le « *hachage* » du mot de passe. Dans le cas de cette fonction, le hachage est effectué entre deux chaînes de caractères. Nous avons choisi le nom d'utilisateur comme deuxième paramètre. Nous aurions pu choisir une suite de lettres fixes mais dans ce cas, si deux utilisateurs avaient le même mot de passe, le « hachage » aurait été identique, ce qui n'est pas le cas avec cette solution.

Cette technique de codage ne permet pas de retrouver le mot de passe original. Afin de comparer le mot de passe entrée par l'utilisateur dans le formulaire au mot de passe contenu dans la base de données, il faut « hacher » le mot de passe du formulaire avec le login de l'utilisateur et le comparer au mot de passe haché contenu dans la base de données.

La technique inverse consisterait à décoder le mot de passe de la base de données et le comparer directement à celui du formulaire mais cela voudrait dire que le cryptage est effectué sur la base d'une clé de cryptage et ainsi, une personne connaissant cette clé pourrait très facilement trouver les mots de passe de tous les utilisateurs.

b. ajout.php

Ce fichier est donc le point de départ de la partie administrateur de notre site. C'est sur cette page qu'est renvoyé l'utilisateur lorsqu'il clique sur le lien « Accès Administrateur » en page principale. Elle est organisée de la même façon que la page index.php (III.2.a, page 34).

Il est basé sur le formulaire suivant, le formulaire de connexion des administrateurs :

Page d'ajout d'utilisateurs
Réservé aux administrateurs

Login :

Mot de passe :

```
<form action="ajout.php" method="POST">
Login :      <input type="text" name="log" style="width:10;" />
Mot de passe : <input type="password" name="pass" style="width:10;" />
              <input type="submit" value="Valider"/>
</form>
```

Ce formulaire est affiché par défaut lorsque l'on vient sur la partie administrateur. En effet, au début de cette page, nous testons si les champs du formulaire ont été remplis et effectuons le traitement adéquat.

```
if(isset($_POST['log']) && isset($_POST['pass'])) {
//verification du mot de passe
//connexion à la page nouveau.php si mot de passe vérifié
//sinon renvoi vers ajout.php
}
else
{
//c'est ici que nous insérons le formulaire
}
```

Si les champs ont été remplis, on les récupère dans deux nouvelles variables

```
$log=$_POST['log'];
$pass=$_POST['pass'];
```

On se connecte à la base de données et on vérifie via une requête SQL utilisant « COUNT » qu'il y a bien un utilisateur ayant ce login. Cette requête est la même que celle utilisée dans « index.php ». Le traitement pour tester le mot de passe est également identique, seul le nom de la table dans la requête SQL change.

Si l'authentification a réussi, on se réfère à la page « adlog.php », si l'authentification a échoué, on se réfère à « admallog.php »

α. admallog.php

Cette page reprend simplement le formulaire de connexion des administrateurs en indiquant que l'authentification a échoué

identification incorrecte, veuillez reessayer.

Login :

Mot de passe :

6. *adlog.php*

Cette page est appelée lorsqu'un administrateur est authentifié. Elle contient deux formulaires, un pour ajouter un nouveau marqueur dans la base de données et un pour en supprimer.

Le formulaire d'ajout est le suivant :

Login :	<input type="text"/>
Prenom :	<input type="text"/>
Nom :	<input type="text"/>
Mot de passe :	<input type="password"/>
Lien vers photo :	<input type="text"/>
Latitude :	<input type="text"/>
Longitude :	<input type="text"/>
	<input type="submit" value="Valider"/>

```
<form action="nouveau.php" method="POST" onsubmit="return confirmation(this) ">
Login :      <input type="texte" name="sqllogin">
Prenom :    <input type="texte" name="sqlprenom">
Nom :       <input type="texte" name="sqlnom">
Mot de passe : <input type="password" name="sqlpassword">
Lien vers photo : <input type="texte" name="sqlphoto">
Latitude :   <input type="texte" name="sqllatitude">
Longitude :  <input type="texte" name="sqllongitude">
             <input type="submit" value="Valider">
</form>
```

Ce formulaire permet de rentrer un nouvel utilisateur en entrant manuellement toutes les valeurs des champs, y compris la latitude et la longitude. Nous pouvons voir que nous avons ajouté un événement JavaScript sur ce formulaire, « onsubmit ».

```
onsubmit="return confirmation(this) "
```

Cet événement se déclenche lorsque l'utilisateur envoie un formulaire, lors d'un clic sur le bouton valider ou lors de l'appui sur la touche « entrée ». La syntaxe de la fonction est un peu spéciale dans ce cas « return » permet de signifier que l'on récupère le code de retour de la fonction.

Le code JavaScript de la fonction est le suivant :

```
function confirmation(){  
  if(confirm("Voulez vous vraiment ajouter cet utilisateur ?"))  
  {  
    return true;  
  }  
  else {  
    return false;  
  }  
}
```

L'instruction «`confirm("Voulez vous vraiment ajouter cet utilisateur ?")`» ouvre une *boite de dialogue** qui contient le texte en paramètre avec un bouton « OK » et un bouton « Annuler ».



Si l'utilisateur choisit « Annuler », le code de retour de la fonction sera faux et le formulaire ne sera donc pas envoyé, le navigateur restera sur la même page.

Si l'utilisateur choisit « OK », le code de retour de la fonction sera vrai et le formulaire sera envoyé vers la page « nouveau.php ».

Cette fonction permet de demander la confirmation de l'administrateur pour ajouter un nouvel utilisateur.

Le formulaire de suppression est le suivant :

Login de l'utilisateur à supprimer:

```
<form action="retirer.php" method="POST" onsubmit="return suppression(this)">  
Login de l'utilisateur a supprimer: <input type="text" name="logsuppr">  
                                <input type="submit" value="Supprimer">  
</form>
```

Tout comme pour le formulaire d'ajout, nous avons mis un événement « onsubmit » avec une fonction de confirmation comparable à la précédente.

```
function suppression(){  
  if(confirm("Voulez vous supprimer cet utilisateur ?"))  
  {  
    return true;  
  }  
  else {  
    return false;  
  }  
}
```

Cette fonction affiche la boîte de dialogue suivante



γ. *nouveau.php*

Cette page récupère les données du formulaire d'ajout et ajoute l'utilisateur dans la base de données. Tout d'abord, nous devons tester si les champs du formulaire sont bien définis et ne sont pas vides. (Nous testons tous les champs exceptés celui de la photo que nous avons créés au début de notre projet mais que nous n'avons pas eu le temps de mettre en place, cela ne dérange donc pas si ce champ est vide).

```
if(isset($_POST['sqllogin']) && isset($_POST['sqlprenom'])  
  && isset($_POST['sqlnom']) && isset($_POST['sqlpassword'])  
  && isset($_POST['sqllatitude']) && isset($_POST['sqllongitude'])  
  && ($_POST['sqllogin'] != "") && ($_POST['sqlprenom'] != "")  
  && ($_POST['sqlnom'] != "") && ($_POST['sqlpassword'] != "")  
  && ($_POST['sqllatitude'] != "") && ($_POST['sqllongitude'] != ""))
```

Si un des champs est vide nous réécrivons la page précédente en précisant qu'un des champs n'était pas rempli. Si tous les champs sont renseignés et après s'être connecté à la base de données, on effectue la requête pour ajouter le marqueur.

```
$requete=mysql_query( "INSERT INTO `correspondances` (  
  `id`, `login`, `prenom`, `nom`, `password`, `photo`, `latitude`, `longitude` )  
VALUES (', ',$sqllogin', '$sqlprenom', '$sqlnom', '$sqlpassword', '$sqlphoto',  
  '$sqllatitude', '$sqllongitude');");
```

Ensuite, on réécrit de nouveau le formulaire pour permettre à l'administrateur d'ajouter encore un nouveau marqueur.

δ. *retirer.php*

Cette page reçoit les données du formulaire pour supprimer un utilisateur. Ce n'est en fait que le login de l'utilisateur qui est reçu. On teste donc si le champ était rempli.

```
if(isset($_POST['logsuppr']) && ($_POST['logsuppr']!=""))
```

Si c'est le cas, on récupère le login dans une nouvelle variable après s'être connecté à la base de données. Ensuite on effectue la requête SQL de suppression.

```
$suppr=$_POST['logsuppr'];  
$requete=mysql_query("DELETE FROM `correspondances`  
WHERE `login` = \"\$suppr\" LIMIT 1");
```

Dans ce que nous avons fait, nous ne vérifions pas qu'il existe effectivement une entrée correspondant au login que nous avons rentré. La requête est donc effectuée même si personne dans la base de données n'a de login correspondant, rien ne sera supprimé, ce qui n'est pas gênant pour la navigation

Bilan

1. Résultats

Ce projet nous a permis de mettre en place un site internet à l'adresse <http://googlemap.free.fr>. Ce site permet de visualiser la répartition géographique des étudiants de la promotion 2006. Nous avons mis en place deux parties, tout d'abord la partie cliente, qui permet de visualiser la carte, les marqueurs, de déplacer son propre marqueur. Cette partie est basée sur JavaScript et son utilisation avec Google Map mais utilise également du PHP. La partie administrateur, que nous avons réussi à mettre en place à la fin de notre projet, qui permet via une interface web d'ajouter et de supprimer des marqueurs. Cette partie ne comprend pas l'affichage des cartes et marqueurs.

2. Difficultés

Nous avons confronté plusieurs difficultés durant notre projet. La première d'entre elles était de comprendre comment fonctionnait exactement l'envoi de données à la volée : en effet la première version de notre site web créait sur le serveur un fichier `markers.xml` qui contenait la liste des marqueurs que nous souhaitions afficher. Notre tuteur de projet, Pierre Chatelier, nous a alors parlé de cette possibilité de ne créer le fichier de données sur les marqueurs que sur l'ordinateur de la personne qui demande à afficher une page. Cette possibilité était encore nouvelle pour nous et il nous a fallu un peu de temps pour bien assimiler cette technologie.

Une autre difficulté s'est montrée lorsque nous souhaitions rentrer plus dans les détails du projet : certaines fonctions que nous voulions intégrer au site web n'ont pas pu l'être par manque de temps.

Nous avons aussi essayé de faire en sorte que nous puissions bouger plusieurs marqueurs et que les valeurs soient quand même changées pour le marqueur correspondant dans la base de données. Ceci s'est révélé plus difficile que prévu et nous avons vite décidé de fixer les marqueurs que nous ne pouvions pas bouger et de n'en laisser qu'un seul sur lequel l'utilisateur avait les pleins pouvoirs.

Ceci a amené un souci majeur : comment reconnaître son marqueur sachant que tous les marqueurs sont identiques : nous avons alors regardé la documentation Google Map et, à l'aide de notre tuteur de projet, décidé de modifier les icônes des marqueurs des utilisateurs dont on ne peut pas déplacer le marqueur et de ne laisser en grand que le notre.

3. Perspectives

Comme nous l'avons évoqué ci-dessus nous n'avons pas eu assez de temps pour intégrer toutes les fonctions souhaitées dans la page internet. Ainsi nous avons pensé à permettre à l'utilisateur de pouvoir changer l'échelle de la carte d'un simple défilement de molette de souris. En effet cette fonctionnalité faciliterait nettement la recherche de lieux, personnes... Nous pensions qu'il s'agissait juste d'un ajout de contrôle classique mais en recherchant de la documentation sur Internet, nous avons constaté qu'il s'agissait en fait de coder une fonction JavaScript entière.

De plus nous avons songé à intégrer une info-bulle qui s'affiche lorsque la souris pointe sur un marqueur particulier. Dans cette info-bulle nous aurions souhaité afficher une photo de l'étudiant mais nous avons manqué de temps et n'avons pas encore pu réaliser ces fonctions supplémentaires qui rendent plus pratique l'utilisation de ce site. En effet pour l'instant nous ne pouvons identifier à qui correspond un marqueur particulier alors que ce dernier est bien attribué à une personne particulière.

Pour l'info-bulle il faudrait intégrer un champ url-photo dans la page générée à la volée : marker.php, la table SQL, étant déjà prête à accueillir cette donnée. Cependant il faudrait aussi regarder les fonctions par lesquelles compléter le JavaScript.

Conclusion

Un site internet est un moyen nouveau pour se documenter, s'informer, se divertir... Ils sont de plus en plus nombreux et de plus en plus développés.

Désormais, avec toutes les technologies disponibles, il est beaucoup plus facile de construire un site web qui soit dynamique, attractif. Ces technologies telles que HTML, PHP, JavaScript ou encore AJAX sont offertes au grand public.

Dans le cadre de notre projet, nous les avons exploitées et les avons combinés à l'API Google Map, service de cartographie en ligne. Toutes ces technologies étant nouvelles pour nous, il nous a fallu nous informer et nous documenter afin de comprendre comment elles s'utilisaient.

Nous avons dû mettre en place une carte reflétant la répartition des étudiants de la promotion précédente dans notre département, pour cela nous avons dû créer une base de données, consultée grâce au langage PHP et qui envoie les informations au script Google Map via un procédé basé sur AJAX, une des dernières technologies à la mode dans la construction de site web. Ce Script traite ensuite ces informations pour donner la carte et les marqueurs qui s'affichent dans le navigateur.

Toute la difficulté de ce projet consistait à apprendre à combiner les différents langages, ce que nous avons fait. Afin de le finaliser, nous avons également rempli la base de données avec les coordonnées des vrais étudiants de la promotion 2005, ceci grâce à une liste que nous a remis le secrétaire du département. Nous avons également mis en ligne notre site à l'adresse <http://googlemap.free.fr> pour les personnes désirant se rendre compte du devenir des anciens étudiants.

Nous pensons que ce projet est une réussite dans le sens où nous avons réalisé un site correspondant bien aux objectifs attendus au début. Même si certaines améliorations sont à apporter, notre site reste simple et ludique vu de l'utilisateur même s'il s'avère plus sophistiqué vu du développeur.

English Summary

We, Krzysztof Porczak and Julien Vey, students in second year of Network-Engineering and Telecommunications, had to work on a project during many weeks in order to obtain our two-year University Diploma. We were supervised for this project by Dr. Pierre Chatelier.

The topic we had to lead during all these weeks was Google Map, an Application Programming Interface that enables insertion of a map on a website. We had to create a website on which we can find actual localisation of former students of the University. The aim of such a topic is to teach us new web-programming languages but this can be used like a new way to gather information about former students in the future because of its usefulness. Moreover this topic is all the more interesting that Internet is evolving very quickly since the mid 1990s.

To begin working on this project we had to learn new languages: JavaScript and PHP. Then we tried to insert a first Google map on our website. Once we arrived to obtain a map on our website, we added each time a new function such as show a marker on the map, show many markers on the map, enable people to drag these markers, recover geographical coordinates from a marker. Finally we inserted PHP language in the website in order to make of this website a dynamic one, that means that we enabled people to log on and then they are allowed to move their own marker, but not one another. Moreover we added some administration web pages that enable authorized users to add a new user or delete an existing one. The result is what we expected but we hadn't time enough to complete the project because we hadn't done any window that appears when the mouse points at a marker and this window should have had a picture of the people whose marker it is and his name. Moreover we'd prefer that people could change the zoom by wheel scrolling but we hadn't time enough to make this function work.

During this project we had to overcome several difficulties, a technical one was to understand some of the technologies and methods we had to use, especially AJAX which was totally unknown for us.

This work was a success as far as we managed to make a real website including Google Map although it doesn't include all the functions we wanted because the work was rather long and we'd need at least three weeks more to finish the project. This project gave us a lot, first of all in professional skills: actually the creation of this website made us learn new web-programming languages like PHP and JavaScript and such skills can be appreciated as long as Internet is more and more developed so most of firms need to have their own website. What's more is that the realisation of this project taught us to have a better time management as well as to share tasks in a better way. This share of tasks taught us to explain how works a code in a concrete way. Moreover as long as we had to get some new knowledge to make this project we learnt to be autonomous too. Indeed we often had to search in many documents by ourselves.

Bilan Personnel

Ce projet a été très enrichissant au niveau technique comme au niveau personnel. Le fait de travailler en binôme nous a permis de développer un esprit d'équipe. Nous avons du apprendre à gérer notre temps, à partager les tâches afin d'être plus efficaces. Par exemple afin de résoudre les différents problèmes qui subsistaient sur le site, l'un de nous s'est chargé de la partie PHP (Krzysztof) pendant que l'autre avait en charge la partie JavaScript et l'API Google Map (Julien).

De plus, nous avons dû faire preuve d'autonomie. Nous avons eu recours à un travail de documentation, tout d'abord pour mettre en place l'API Google Map dans la page et voir comment elle marchait, puis ensuite pour savoir compléter la page avec les différents langages de programmation utilisés, à savoir le XHTML, le PHP, le SQL et le JavaScript. Lors de la réalisation de ce projet, nous n'avions pas encore fait le cours sur les langages de programmation web. Nous avons donc dû nous instruire par nous même afin de savoir les utiliser, notamment en cherchant des cours sur internet.

De plus, les langages que nous avons vu lors ce projet nous seront utiles lors de nos prochaines années d'études

Bibliographie

Ouvrages :

Jean Carfantan, JavaScript, édition Micro Application, 2005

Larry Ullman, PHP & MySQL, édition Campus Press, 2003

Sites internet :

<http://www.siteduzero.com>

<http://www.google.com/apis/maps/documentation>

<http://www.econym.demon.co.uk/googlemaps/>

Glossaire

- ✚ API : Application Programming Interface. Interface de programmation d'application, un jeu de fonctions ou de méthodes, utilisés pour accéder à certaines fonctionnalités.
- ✚ Arbre logique : En informatique, un arbre est une structure de données récursive, représentant un arbre au sens mathématique.
- ✚ Attribut : Une valeur associée à un élément. Cette valeur est généralement composée d'un nom et d'une valeur. Par exemple, l'élément <a> possède généralement un attribut « href » dont la valeur est une adresse.
- ✚ Authentification : L'authentification consiste, pour un système informatique, à effectuer la vérification de l'identité d'une entité (personne, machine...).
- ✚ Balise : Élément du langage HTML. Les balises définissent la structure et la mise en forme d'un document hypertexte.
- ✚ Bande passante : C'est la quantité d'information que peut véhiculer un canal de communication.
- ✚ Barre d'adresse : zone de texte permettant la saisie d'une adresse internet.
- ✚ Base de données : Une base de données est un ensemble structuré et organisé permettant le stockage de grandes quantités d'informations afin d'en faciliter l'exploitation (ajout, mise à jour, recherche de données). Une base de données se traduit physiquement par un ensemble de fichiers sur disque.
- ✚ Boite de dialogue : Fenêtre qui s'affiche à l'écran lors de l'utilisation d'un logiciel. Cette fenêtre est interactive et permet de donner ou de demander des informations relatives au programme.
- ✚ Bouton : Élément graphique contenant une icône ou du texte, sur laquelle un lien hypertexte déclenche une action.
- ✚ Bug : Défaut de conception ou de réalisation d'un programme se manifestant par des anomalies de fonctionnement.
- ✚ C : En informatique, C est un langage de programmation impératif. C'est un des langages les plus utilisés.
- ✚ Case à cocher : Case cliquable servant à indiquer un choix.
- ✚ Champ : Espace réservé à la consignation d'un type de données sur une fiche, dans notre cas, sur une base de données.

- ✚ Cookie : En informatique, un cookie (mot anglais signifiant « biscuit ») est un petit ensemble d'informations envoyé par un serveur HTTP à un navigateur Web, qui est ensuite automatiquement renvoyé lors de chaque nouvelle connexion à ce serveur HTTP.
- ✚ Cryptage : Méthode qui assure la confidentialité et la sécurité de l'information véhiculée sur Internet. Les données sont brouillées, et donc illisibles pendant le transport, puis décodées à l'aide de la même méthode sur la machine du destinataire.
- ✚ CSS : Le langage CSS (Cascading Style Sheets : feuilles de style en cascade) est utilisé pour décrire la présentation d'un document structuré écrit en HTML ou en XML, et c'est le World Wide Web Consortium (W3C) qui en a la direction.
- ✚ Diplôme Universitaire de Technologie : Le diplôme universitaire de technologie (DUT) est un diplôme de l'enseignement supérieur français dispensé dans les IUT. Il est orienté vers l'insertion professionnelle des étudiants, mais propose aussi une solide formation théorique qui permet la poursuite d'études dans des écoles d'ingénieurs, ou des DEUG et IUP ou encore des Licences professionnelles.
- ✚ DOM : Une spécification du World Wide Web Consortium (W3C) décrivant la structure des documents HTML (Hypertext Markup Language) et XML (Extensible Markup Language) de façon à ce qu'ils puissent être manipulés via un navigateur Web.
- ✚ Fonction : En informatique, une fonction est un ensemble d'instructions réalisant une certaine tâche.
- ✚ Fonction anonyme : Une fonction anonyme est, comme son nom l'indique, une fonction qui n'est pas identifiée par un nom.
- ✚ Formulaire : page Web interactive abritant des rubriques prédéfinies à renseigner par le visiteur. La récupération des données se fait généralement grâce à un script PHP.
- ✚ GET / POST : méthodes implémentées dans un site internet. elles permettent à un client de demander un document.
- ✚ Google Map : Service de cartographie en ligne proposé par Google et pouvant être incorporé sur une page web.
- ✚ Hachage : C'est une fonction basée sur le principe de transformation de bloc de taille variable en bloc de taille fixe qui appliquée sur le document produit une empreinte de celui-ci.
- ✚ HTML : Le HTML, abréviation de l'anglais Hypertext Markup Language, aussi appelé langage HTML, rarement traduit littéralement en langage de balisage hypertexte, est le langage informatique créé et utilisé pour écrire les pages Web. HTML permet en particulier d'insérer des hyperliens dans du texte, donc de créer de l'hypertexte, d'où le nom du langage.

- ✚ Java : Java est une technologie composée d'un langage de programmation orienté objet et d'un environnement d'exécution. Préalablement nommé Oak, il a été créé par James Gosling et Patrick Naughton chez Sun Microsystems avec le soutien de Bill Joy. Le langage Java fut officiellement présenté le 23 mai 1995 au SunWorld. ...
- ✚ JavaScript : langage permettant de contrôler le navigateur et le HTML avec les richesses fonctionnelles que ne permet pas le langage HTML.
- ✚ IUT : En France, les Instituts universitaires de technologie (IUT en abrégé) proposent des formations post-bac qui amènent au niveau technicien supérieur après 2 années d'étude (bac+2) : le DUT. Les IUT dépendent des universités, contrairement aux BTS (autres diplômes bac+2) qui sont au sein de lycées.
- ✚ Liste déroulante : liste affichant tous les choix possibles. L'internaute sélectionne une seule réponse au moyen de la souris ou du clavier.
- ✚ Marqueur : petite icône qui s'affiche pour indiquer la position géographique d'une personne, d'un lieu.
- ✚ Méthode : En informatique, une méthode est une fonction membre d'un objet.
- ✚ Moteur de recherche : Un moteur de recherche est un logiciel permettant de retrouver des ressources (pages Web, forums Usenet, images, vidéo, etc.) associées à des mots quelconques. Certains sites Web offrent un moteur de recherche comme principale fonctionnalité. On appelle alors moteur de recherche le site lui-même.
- ✚ Navigateur : Logiciel qui sert d'interface entre l'utilisateur et le web et permet d'explorer les ressources d'Internet et en particulier de rechercher, consulter des documents, et exploiter les liens hypertextes qu'ils comportent.
- ✚ Nœud : partie de la tige où s'attache la base d'une feuille, en informatique c'est la base de laquelle descendent des éléments.
- ✚ Objet : entité regroupant des données et des procédures et fonctions opérant sur ces données. Par abus de langage, terme générique pour désigner une instance.
- ✚ Page Web / Page Internet : La page Web est l'unité de consultation du World Wide Web. Ce terme a une signification pratique. Il n'a pas de définition technique formelle. C'est un document informatique qui peut contenir du texte, des images, des formulaires à remplir et divers autres éléments multimédia et interactifs.
- ✚ PHP : C'est un langage de script exécuté côté serveur. La syntaxe provient de langage comme le C, le Java ou le Perl. Langage Open Source, PHP est gratuit sous licence GNU GPL. Les scripts sont écrits de manière simple puis intégrés au sein d'une page HTML. Il séduit aussi par son interfaçage simplifié avec les bases de données.
- ✚ Pixel : Le pixel ou point est l'unité de base d'une image numérique. Son nom provient de l'expression anglaise picture element, c'est-à-dire, « élément d'image » ou « point élémentaire ».

- ✚ Requête HTTP : Requête utilisant HyperText Transfert Protocol. Protocole standard de transmission des pages Web sur Internet.
- ✚ Script : Un script est un programme écrit dans un langage interprété. A la différence des programmes écrits en C ou en C++ (ou autre langage compilé), les scripts sont traduits en langage machine au fur et à mesure de leur exécution. Ils sont en général plus lents à exécuter, mais aussi plus faciles à écrire. Les langages utilisés pour écrire un script sont par exemple bash, perl, php, python...
- ✚ Sessions : programmation. Intervalle de temps compris entre le premier accès par un utilisateur à un site internet et la clôture de toutes les fenêtres de son navigateur. Elle se termine automatiquement au bout d'un certain temps défini.
- ✚ Site internet : site Web est un ensemble de pages Web et d'éventuelles autres ressources du World Wide Web, hyperliées en un ensemble cohérent ayant une adresse Web et conçu pour être consulté avec un navigateur Web. On dit simplement site si le contexte s'y prête. On dit aussi site Internet par métonymie, Internet contenant le World Wide Web, mais le terme « site Internet » peut porter à confusion (par exemple avec un site FTP) et n'a pas littéralement de réelle signification.
- ✚ SQL : Structured query language (SQL), traduisez Langage structuré de requêtes, est un langage informatique standard, destiné à interroger ou piloter (modifier contenu et structure) une base de données.
- ✚ Système d'exploitation : Un système d'exploitation (SE ou OS en anglais pour Operating System) est un ensemble cohérent de logiciels permettant d'utiliser un ordinateur et tous ses éléments (ou périphériques). Il assure le démarrage de celui-ci et fournit aux programmes applicatifs les interfaces pour contrôler les éléments de l'ordinateur. ...
- ✚ Table : Une table est une structure fondamentale d'une base de données. Les informations sont organisées sous forme de champs, appelés colonnes et d'enregistrements, appelés lignes.
- ✚ Tableau associatif : tableau qui associe une valeur à une clé.
- ✚ Unix : UNIX est un système d'exploitation, à usage principalement professionnel, conceptuellement ouvert (nombreux outils qui font une chose et la font bien), multitâche et, multiutilisateur, créé en 1969. Il a donné naissance à toute une famille de systèmes, dont les plus populaires en 2004 sont GNU/Linux, BSD et Mac OS X. Les UNIX sont tous ces systèmes qui suivent la norme POSIX.
- ✚ URL : Uniform Resource Locator. La traduction littérale (localisateur uniforme des ressources) ne veut pas dire grand chose. L'URL constitue l'adresse d'une page web. On l'appelle également adresse réticulaire ou adresse universelle. Il s'agit d'une adresse permettant de lire et mémoriser plus facilement les adresses IP. La conversion des adresses IP en adresse URL est faite par DNS ...

- ✚ W3C : Le World Wide Web Consortium, abrégé W3C, est un consortium fondé en octobre 1994 pour promouvoir la compatibilité des technologies du World Wide Web telles que HTML, XHTML, XML, CSS, PNG, SVG et SOAP. Le W3C n'émet pas des normes, mais des recommandations.
- ✚ XML : XML (Extensible Markup Language ou langage de balisage extensible) est un standard du World Wide Web Consortium qui sert de base pour créer des langages balisés spécialisés; c'est un « méta langage ». Il est suffisamment général pour que les langages basés sur XML, appelés aussi dialectes XML, puissent être utilisés pour décrire toutes sortes de données et de textes. Il s'agit donc partiellement d'un format de données.
- ✚ Zone de texte : Un contrôle dans lequel un utilisateur peut entrer du texte. Les zones de texte enrichi permettent d'entrer des données avec des options de mise en forme, ce que ne permettent pas les zones de texte standard.
- ✚ Zone d'option : Un contrôle dans lequel l'utilisateur peut choisir tel ou tel option en fonction de ce qu'il désire
- ✚ Zone de saisie : comprend toutes les différentes zones que peut contenir un formulaire (zone de texte, case à cocher ...)
- ✚ Zoom : mouvement optique semblant rapprocher (zoom avant) ou éloigner (zoom arrière) vivement le sujet de l'œil du spectateur.

Annexes

Script issu de « page_logge.php », qui affiche différents marqueurs selon qu'il s'agisse de celui de l'utilisateur en cours ou des autres. Il permet d'illustrer beaucoup de possibilités de Google Map.

```
function load() {
if (GBrowserIsCompatible()) {
    var map = new GMap2(document.getElementById("map"));
    map.addControl(new GSmallMapControl());
    map.addControl(new GMapTypeControl());
    map.setCenter(new GLatLng(45.756505,3.111534), 6);

    var mgrOptions = { borderPadding: 50, maxZoom: 15, trackMarkers: true };
    var mgr = new GMarkerManager(map, mgrOptions);

    var icon = new GIcon();
    icon.image = "http://labs.google.com/ridefinder/images/mm_20_red.png";
    icon.shadow = "http://labs.google.com/ridefinder/images/mm_20_shadow.png";
    icon.iconSize = new GSize(12, 20);
    icon.shadowSize = new GSize(22, 20);
    icon.iconAnchor = new GPoint(6, 20);
    icon.infoWindowAnchor = new GPoint(5, 1);

    GDownloadUrl("marker.php", function(data) {
        var xml = GXml.parse(data);
        var markers = xml.documentElement.getElementsByTagName("marker");

        for (var i = 0; i < markers.length; i++) {
            if(markers[i].getAttribute("id")==".$login.") {
                var point = new GLatLng(parseFloat(markers[i].getAttribute("lat")),
                    parseFloat(markers[i].getAttribute("lng")));
                var mark= new GMarker(point, {draggable: true});
                mark.enableDragging();
                mgr.addMarker(mark,0);
                map.setCenter(new GLatLng(parseFloat(markers[i].getAttribute("lat")),
                    parseFloat(markers[i].getAttribute("lng"))), 12); }
            else {
                var point = new GLatLng(parseFloat(markers[i].getAttribute("lat")),
                    parseFloat(markers[i].getAttribute("lng")));
                var mark= new GMarker(point, icon);
                mgr.addMarker(mark,0); }

            GEvent.addListener(mark, "drag", function(){
                GDownloadUrl("changement.php?latitude="+this.getPoint().lat()+"&longitude="
                    +this.getPoint().lng()+"&login='.$login.'");
                document.getElementById("latitude").value=this.getPoint().lat();
                document.getElementById("longitude").value=this.getPoint().lng();
            });
        }
        mgr.refresh();
    });
}}
```


Code HTML de la page d'ajout d'utilisateur dans la base de données. Cette page illustre l'utilisation de HTML et notamment des formulaires.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
  <title>Suivi des etudiants de la promo RT 2006</title>
  <script type="text/javascript" src="verif.js"></script>
</head>
```

```
<body>
```

Vous pouvez maintenant rentrer un nouvel utilisateur

```
<form action="nouveau.php" method="POST" onsubmit="return confirmation(this)">
Login :          <input type="texte" name="sqllogin">
Prenom :        <input type="texte" name="sqlprenom">
Nom :           <input type="texte" name="sqlnom">
Mot de passe :  <input type="password" name="sqlpassword">
Lien vers photo : <input type="texte" name="sqlphoto">
Latitude :      <input type="texte" name="sqllatitude">
Longitude :     <input type="texte" name="sqllongitude">
                <input type="submit" value="Valider">

</form>
```

ou en supprimer

```
<form action="retirer.php" method="POST" onsubmit="return suppression(this)">
Login de l'utilisateur <input type="text" name="logsuppr"> <input type="submit" value="Supprimer">
supprimer:

</form>
```

```
</body>
```

```
</html>
```

Extrait de la sauvegarde de la base de données qui montre quelques commandes SQL.

```
CREATE TABLE `correspondances` (  
  `id` int(11) NOT NULL auto_increment,  
  `login` varchar(40) collate latin1_general_ci NOT NULL,  
  `prenom` varchar(20) collate latin1_general_ci NOT NULL,  
  `nom` varchar(20) collate latin1_general_ci NOT NULL,  
  `password` varchar(30) collate latin1_general_ci NOT NULL,  
  `photo` varchar(100) collate latin1_general_ci default NULL,  
  `latitude` double NOT NULL,  
  `longitude` double NOT NULL,  
  PRIMARY KEY (`id`),  
  KEY `login` (`login`)  
) ENGINE=MyISAM AUTO_INCREMENT=72 DEFAULT CHARSET=latin1  
  COLLATE=latin1_general_ci AUTO_INCREMENT=72 ;  
  
INSERT INTO `correspondances` VALUES (1, 'isaitelm', 'Isam', 'Ait El Maati',  
  'isXhpKaVOdt1E', 'url://', 46.1251286082803, 3.42430114746094);  
INSERT INTO `correspondances` VALUES (2, 'frbailla', 'François', 'Baillaud',  
  'frVymbfwmK5HA', 'url://', 45.7612359397437, 3.10973167419434);  
INSERT INTO `correspondances` VALUES (3, 'mabonnel', 'Mathieu', 'Bonnell',  
  'maHS7cg20t2nA', 'url://', 44.8361519955309, -0.574722290039062);
```