CATEGORIE SOCIALE
Avenue Montesquieu, 6 - 4101 Seraing (Jemeppe sur Meuse)

Conception et réalisation d'une application web mobile touristique

Jonathan MARGRÈVE

Travail de fin d'études présenté en vue de l'obtention du grade de Bachelier en écriture multimédia

Année académique : 2012 - 2013

Remerciements

Ce projet n'aurait pu voir le jour sans l'implication active de la Fédération du Tourisme de la Province de Namur. En outre, je souhaite remercier Maxime Hanson qui a été décisif dans la réalisation graphique de l'application ainsi que Patrick Spapen et les autres membres de la Fédération qui ont toujours été disponibles pour les réunions de concertation.

Je tiens aussi à remercier Michel Frenay, analyste-programmeur chez NSI IT Software & Services, et Albert Lemort, mon superviseur de TFE. Ces personnes m'ont été d'une aide précieuse pour la structuration et la rédaction de ce travail de fin d'études.

Enfin, je voudrais remercier toutes les personnes qui ont participé de près ou de loin à l'élaboration de ce travail tant au niveau graphique, informatique que rédactionnel.

Table des matières

Liste des abréviations 5
Introduction 6
Contexte institutionnel 8
Cahier des charges 9
Contenu et structure 11
Web app versus native app 11
Le contenu 12
Le module d'administration 12
Les flux 13
Arborescence du projet 14
Description pratique 15
Module d'administration & API de gestion des données 15
Utilisation du flux XML 15
Accès et sécurité du module d'administration 16
JQuery Mobile 17
Responsive Web Design 17
Les attributs data et le modèle multipages 18
Theme Roller JQuery Mobile 19
Google Maps API v3 20
Affichage des marqueurs et infobulles 22
Wunderground API 24
PhoneGap Build 25
Code de l'application 27
Page principale (index.html) 27
Les CSS et polices 28
Les images 28
Les thèmes 29
Script principal (index.js) 29
Script agenda (agenda.js) 31
Script carte (carte.js) 33
Script recherche (recherche.js) 34

Script météo (meteo.js) 35
Scripts JQuery et JQuery Mobile 36
Pour aller plus loin 37
Conclusions 38
Abstract 39
Bibliographie 40
Webographie 41
Annexes 43
Annexe 1 : Extrait du sondage clientèle effectué par le Commissariat général du Tourisme 43
Annexe 2 : Extrait de la structure du flux XML récupéré par l'API 44
Annexe 3 : Exemple de code et d'affichage avec les data-role 46
Annexe 4 : Schéma montrant la différence entre le modèle multipages et les pages séparées 47
Annexe 5 : Theme Roller JQuery Mobile 48
Annexe 6 : Méthode $.getJSON du script *villes.js* 49
Annexe 7 : Condition de filtrage des évènements "Meuse en Fête" utilisée dans *agenda.js* 50
Annexe 8 : Utilisation de variables globales et de variables locales dans *carte.js* 51
Annexe 9 : Code HTML du panneau coulissant dans la division-page id="recherche" 52
Glossaire 53

Liste des abréviations

AJAX: Asynchronous Javascript and XML.

App: Application

API: Application Programming Interface

BD: Base de données

CSS: Cascading Style Sheets

FTPN: Fédération du Tourisme de la Province de Namur

GM: Google Maps

GPS: Global Positioning System

HTML: Hypertext Markup Language

ID: Identifiant

JQ: JQuery

JQM: JQuery Mobile

JS: Javascript

JSON: Javascript Object Notation

PHP: Hypertext Preprocessor

QR code: Quick Response code

RWD: Responsive Web Design

URL: Uniform Resource Locator

Web App: Application web

XML: Extensible Markup Language

Introduction

Dans ce travail de fin d'études, j'ai voulu associer deux grandes activités qui sont, à la base, très éloignées l'une de l'autre mais qui, avec l'amélioration croissante des nouvelles technologies, sont vouées à être fusionnées. Il s'agit du tourisme et du multimédia.

Amateur de folklore et de culture wallonne, mais aussi passionné par les nouvelles technologies, il me paraissait intéressant de confronter deux domaines opposés d'un point de vue générationnel. En effet, les musées et autres parcours touristiques sont davantage prisés par les 35-65 ans que par la population "junior", d'après un sondage réalisé par l'Observatoire du Tourisme Wallon (extrait du sondage en annexe 1).
Plusieurs organisations et sociétés ont déjà pris l'initiative de se lancer dans ce tout nouveau secteur qu'est l'eTourisme. Par exemple, la ville de Dinant a sorti deux applications mobiles gratuites permettant de découvrir Dinant et ses alentours. L'une se veut ludique (Le fantôme de Dinant), l'autre plus informative (Visit Dinant)[1]. De même, la société iBeaken a véritablement fait de l'eTourisme son leitmotiv. Cette entreprise est présente dans plus de dix pays et propose à ses clients de créer des QR codes reliés à des textes historiques ou culturels référencés sur le site internet d'iBeaken[2].

Au début de mes recherches et réflexions sur le sujet, il avait été question de réaliser un parcours touristique en version web mobile. Après l'envoi de mails aux différentes administrations et sociétés, le projet que la Fédération du Tourisme de la Province de Namur (FTPN) me proposa s'alignait parfaitement avec mon idée de départ. Il consistait en la création d'une application mobile devant reprendre les différentes manifestations printanières et estivales de l'évènement annuel namurois Meuse en Fête (voir le fascicule joint au rapport).

De là, je proposai la réalisation d'une application web mobile à l'aide de techniques de développement multiplateformes telles que JQuery Mobile et PhoneGap Build.

Ce travail écrit va donc servir dans un premier temps à décrire le contexte institutionnel dans lequel a évolué le projet.

Ensuite, je décrirai les contenus et la structure générale de l'application.

Enfin, j'entamerai une description plus pratique des différents menus de l'application et je détaillerai les outils de développement utilisés pour la réalisation de Meuse en Fête App.

Pour une meilleure compréhension des termes mentionnés dans ce travail, n'hésitez pas à vous référer au *Glossaire* situé à la fin du rapport.

[1] www.dinant-tourisme.com
[2] www.ibeaken.com

Consultation du travail pratique

L'application est disponible à l'adresse suivante :
www.meuse-en-fete.mobi
Comme expliqué dans le *Cahier des charges*, il s'agit d'une application web destinée aux dispositifs mobiles. Cependant, jusqu'à la date de présentation du travail, l'application est aussi accessible depuis le navigateur web d'un ordinateur afin de faciliter sa consultation. Après cette date, l'application sera uniquement effective pour les navigateurs des terminaux mobiles.
Pour accéder facilement à l'application en ligne sur votre tablette ou votre smartphone, vous pouvez scanner le QR code ci-contre avec le lecteur de code-barres 2D de votre dispositif mobile.

Contexte institutionnel

La Fédération du Tourisme de la Province de Namur est une ASBL créée en 1926 dont les membres du Conseil d'Administration proviennent du Conseil Provincial, du secteur public et du secteur privé représentant l'ensemble des professionnels du tourisme. Le but de la FTPN est de promouvoir le territoire provincial namurois repris sous l'enseigne Pays des Vallées.

Ses missions:

- Valoriser l'image touristique de la Province de Namur « Pays des Vallées ».
- Promouvoir le potentiel touristique du « Pays des Vallées » en Belgique comme dans les pays limitrophes et développer sa commercialisation.
- Développer l'offre touristique en l'adaptant aux exigences du marché.
- Faciliter la promotion et la mise en marché de projets touristiques par la mise en place d'outils performants.
- Fédérer les professionnels du tourisme afin de dynamiser et de coordonner les initiatives.
- Informer les partenaires de sa politique touristique, assurer la mise en œuvre et le suivi de ses projets.

Depuis plusieurs années, la FTPN a pour objectif d'être présente sur différents supports web afin de fournir des outils modernes adaptés à la demande. Pour ce faire, la Fédération a ouvert récemment 22 points d'accès Wifi publics dans les endroits les plus touristiques de la Province de Namur. Le but étant de faciliter l'accès à Internet pour obtenir un maximum d'informations touristiques sur la région, mais aussi pour permettre d'échanger et de partager des témoignages, des photos, des vidéos, etc.

C'est grâce à cette vision moderne du tourisme que mon projet d'application web mobile touristique a pu voir le jour.

Cahier des charges

Commanditaire du projet

La Fédération du Tourisme de la Province de Namur (FTPN).

Contexte

La FTPN est de plus en plus active dans l'eTourisme par le biais des nouvelles technologies.
La Fédération souhaite réaliser une application mobile dédiée à l'évènement annuel namurois :
Meuse en Fête.

Objectifs

Il s'agit de proposer aux touristes une alternative à l'agenda papier (voir fascicule joint au rapport). L'agenda reprend l'ensemble des évènements se déroulant le long de la Meuse en Province de Namur (en France et en Province de Liège par extension).

Méthode

La solution proposée est de concevoir un site internet à l'aide du framework JQuery Mobile. Ce framework permet de modeler le site web aisément afin que son graphisme et sa structure soient adaptables à n'importe quelle largeur d'écran. Donc, puisqu'il est possible de consulter un site internet sur n'importe quel smartphone (plateforme), on parle de développement multiplateformes. Ce qui signifie qu'on développe une seule fois pour différents types d'appareils mobiles (formats différents et systèmes d'exploitation différents). Au final, on parle de web application ou *web app*.

Dans la suite du projet, l'utilisation d'un autre framework nommé PhoneGap est envisagée. Il permet de "transformer" l'application web (qui est en l'occurrence un site internet) en une application téléchargeable sur les stores Google Play et Apple Store.

Accès au site internet : www.meuse-en-fete.mobi

Description du site

Le site se compose de 6 menus :

- Menu *Villes* contenant les informations sur les villes et leurs manifestations
- Menu *Agenda* détenant la liste de tous les évènements
- Menu *Carte* affichant tous les évènement sur la carte de la Province de Namur
- Menu *À proximité* permettant la recherche d'évènements selon certains critères
- Menu *À propos* servant à se renseigner sur les objectifs de l'application
- Menu *Pays des Vallées* qui est un lien vers le site internet de la FTPN

Module d'administration

Un module d'administration est créé sur le site internet. Il gère l'ajout, l'édition et la suppression des villes qui servent à la localisation et à l'affichage des évènements dans les différents menus.

Accès au module d'administration : www.meuse-en-fete.mobi/admin

Ressources

Les ressources utilisées sont de types texte et image. Elles proviennent soit d'une base de données où sont contenues les informations sur les villes encodées dans le module d'administration, soit du site internet de la FTPN sur lequel est présent un fichier XML contenant la liste des évènements se déroulant en Province de Namur.

Charte graphique

La charte graphique respecte les couleurs de Meuse en Fête dont la couleur dominante est proche du bleu Klein .

Partenaires du projet

Un soutien technique pour la réalisation graphique est apportée par un infographiste de la FTPN, de plus l'encodage des informations dans le module d'administration est aussi à la charge de la FTPN.

Calendrier d'action

- Octobre : premier rendez-vous à Namur au siège de la FTPN
- Novembre : second rendez-vous à la FTPN et établissement des objectifs à atteindre
- Décembre : définition de la structure de base
- Janvier : définition de la structure finale, de l'administration et des types de contenu
- Février : rectifications éventuelles à apporter à la structure et au contenu
- Mars : version alpha prête
- Avril : version bêta prête
- Mai : version finale prête. Présentation à la presse
- Juin : version finale téléchargeable sur Google Play

Contenu et structure

Ce chapitre introduit les contextes, les choix technologiques et les sources de données du projet réalisé.

Web app versus native app

Comme mentionné en introduction, il a été convenu que la consultation des évènements de Meuse en Fête se ferait sous forme d'une *web app*.

Une *web app* ou application web est un site internet développé et mis en forme spécialement pour les appareils mobiles. Il se consulte dans un navigateur web.
Donc, les données que l'on consulte sont stockées sur un serveur web.

Cette notion est opposée à *native app* ou application native qui est développée pour le système d'exploitation de l'appareil mobile (Android, iOS, ...). Celle-ci est installée sur le dispositif, accessible depuis le menu en activant l'icône correspondante ; son exécution est locale (côté client).

Une web app ne permet pas d'accéder à toutes les fonctionnalités offertes par le smartphone. On ne peut pas accéder à l'appareil photo, aux notifications ou au GPS du téléphone. Alors qu'avec une application native, on a un accès total aux éléments techniques et graphiques de l'appareil.
Cependant, une web application est beaucoup plus simple, moins coûteuse et plus rapide à développer car celle-ci utilise le HTML5, le CSS3 et le Javascript qui sont les principaux langages de base pour les développeurs web. De plus, ces langages sont multiplateformes.

En natif, il faut connaître le langage orienté objet comme le Java, pour le système Android, et l'Objective-C, pour iOS. Il faut aussi se familiariser avec les environnements de développement (Xcode pour iOS et Eclipse pour Android).

Grâce à un tout nouvel outil, il est possible d'établir un compromis. Il s'agit de PhoneGap. Ce framework -créé en 2005 par Nitobi Software et racheté par Adobe en 2011- permet la création d'applications mobiles utilisant HTML5, CSS3 et Javascript mais en les encapsulant dans une *native app*.
La bibliothèque de PhoneGap, lorsqu'elle est liée au dossier de l'application, permet l'accès à une grande partie des fonctionnalités du dispositif mobile. Ce qui en fait dès lors une application hybride (mi-native, mi-web).

Chacun possède sa propre opinion sur ces deux méthodes de création d'application mobile. La plupart des développeurs Java légitimeront le développement natif et ils considéreront souvent l'application web ou hybride comme étant de qualité inférieure. Mais en ce qui

concerne Meuse en Fête, l'application web convient parfaitement à la réalisation du projet. Elle permet une méthode de développement plus rapide, moins couteuse et plus accessible.

Tableau comparatif entre *web app* et *native app* tiré du livre *Créez votre WebApp* de Élisabeth Sancey.

Web App	Sujet	Native App
★★★★★	Plateformes	★★★
★★★★★	Outils de développement	★★★
★★★★★	Coût de développement	★
★★★★★	Correction des bogues	★★
★★★★★	Cycle de développement	★★★
★★★	Vitesse	★★★★
★★★★	Accès aux fonctionnalités	★★★★★
★★★★★	Durée de vie	★★
★★★★★	Processus de validation	★★
★★★★★	Diffusion sur les stores	★★★★★

Le contenu

Les informations telles que la position d'une ville ou la date d'un évènement sont fournies à l'application de deux manières. Soit par l'administration, pour la gestion des villes, soit via des flux externes XML provenant du site internet www.paysdesvallees.be.

Le module d'administration

Le menu de l'administration se compose de trois sous-menus permettant d'ajouter une ville, de la modifier ou de la supprimer.
Pour chaque ville, on peut spécifier les informations suivantes :

- Un nom unique
- Une position sur la Meuse permettant de trier les villes dans une liste d'aval en amont
- Un pays (Belgique, France ou Pays-Bas)
- Une latitude et une longitude. Cela va permettre de situer la ville sur la carte générée par Google Maps dans l'application
- Des informations d'ordre général, comme un texte descriptif, l'adresse de l'office du tourisme, un numéro de téléphone, etc

Ces informations sont stockées dans une base de données. Il est possible de les modifier ou de les supprimer via les deux autres sous-menus.

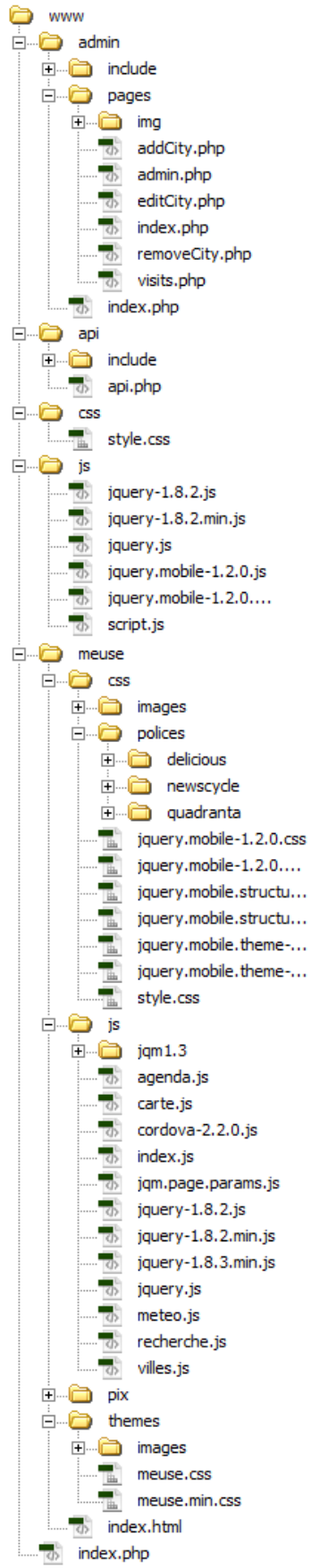
Les renseignements sur les villes doivent donc être encodés manuellement.

Les flux

Les évènements, eux, sont réceptionnés de manière dynamique via un flux XML.
Ce flux est déjà utilisé pour l'affichage des manifestations sur le site internet de la FTPN (www.paysdesvallees.be). Il contient une foule d'informations sur les visites, balades et autres fêtes de village se déroulant en Province de Namur.

En annexe 2 se trouve un aperçu de la structure du flux XML utilisée. Celle-ci sera décrite de manière plus technique dans la section *Description pratique*.

Arborescence du projet



Description pratique

Ce chapitre présente le fonctionnement de l'application et les "outils" qui ont été utilisés pour sa réalisation. Le tout est illustré par des captures d'écran et des parties de code.

Module d'administration & API de gestion des données

Comme expliqué ci-avant dans la section *Contenu et structure*, le module d'administration regroupe la gestion des villes présentes dans l'application.

Dans un premier temps, ce module permettait aussi l'encodage des évènements présents dans la brochure. Vu le nombre important de données (plus de 80 évènements), il m'a été conseillé d'utiliser les flux XML présents sur le site du "Pays des Vallées" (www.paysdesvallees.be/flux/evenements.xml).

Pour des raisons qui seront expliquées dans les rubriques suivantes, j'ai préféré ne pas utiliser de PHP dans les pages de l'application. On ne peut donc pas faire appel à la base de données dans la page HTML directement. Pour pallier ce manque, les renseignements sur les villes et le flux XML relatif aux évènements sont récupérés dans une page PHP nommée *api.php* . Cette page, comme son nom l'indique, sert d'API personnelle et se trouve à la racine du site dans un dossier *api*. Elle fonctionne comme un proxy qui fera office d'intermédiaire entre la base de données et l'application, et entre les données du flux et l'application.

Cette API implémente deux méthodes *getCities* et *getEvents* qui retournent chacune un objet JSON contenant les informations sur les villes et sur les manifestations.

L'application, ou plutôt un script de l'application (par exemple, *agenda.js* qui récupère tous les évènements et les affiche sous forme de liste HTML), fait alors appel à la page *api.php*, avec comme argument dans l'url, la méthode correspondante. Le script récupère l'objet JSON contenant les données renvoyées par la méthode et décode celui-ci pour en extraire les informations sur les villes ou sur les évènements.

Utilisation du flux XML

Pour comprendre plus aisément les explications qui suivent, vous pouvez vous référer à l'annexe 2 du document.

Quand un flux XML est demandé à l'*api.php* avec la méthode *getEvents,* cette dernière parcourt tous les évènements présents dans le flux. En d'autres termes, elle parcourt tous les éléments `<evenement id="00000">` présents dans l'élément racine `<evenements>` . Ensuite, elle les trie en fonction de la date de début via le nœud `<date_debut>01/07/2013</date_debut>` .

Quand le tri est terminé, la méthode *getEvents* encode cette liste dans un objet JSON qui est renvoyé vers le script JS ayant invoqué cette méthode de l'API.

Du côté de l'application (côté client), prenons par exemple le script *agenda.js* affichant le contenu du menu Agenda.
La méthode JQuery *$.getJSON* permet de récupérer l'objet JSON renvoyé par l'API.
Cette méthode (ou fonction) fournit alors un objet *data* qui contient les données triées sur les évènements. Ensuite, à l'aide d'une boucle *$.each* recevant en paramètre *data,* on parcourt chaque sous-section de l'objet *data* (dans le flux XML, il s'agit des données contenues dans chaque nœud `<evenement id="00000">...</evenement>`). Dans la boucle, chaque nœud sera représenté par le nom de variable *_event*. Dès lors, on peut accéder aux renseignements stockés dans l'objet *data*. Par exemple, pour récupérer le nom francophone de l'évènement, il suffira d'écrire *_event.name_fr*.

Le flux XML reprend également des évènements en rapport avec d'autres manifestations.
Il faut donc filtrer les évènements et ne retenir que ceux concernant Meuse en Fête.
Pour cela, il suffit de placer la condition suivante dans le code : si le type d'évènement est égal à "Meuse en Fête", alors on récupère les données de l'élément traité par la boucle *$.each*. Sinon, on passe à l'élément suivant.

Tous les autres scripts de l'application tels que *carte.js*, *villes.js*, etc. fonctionnent de la même manière en listant tous les évènements ou toutes les villes selon l'objectif final du script. Ensuite, chacun utilisera ces données de façon différente. Exemple : afficher les évènements sur une carte, récupérer les évènements se déroulant à Dinant, etc.

Accès et sécurité du module d'administration

Au niveau de la sécurité et de l'accès à la partie administration, un formulaire de connexion est accessible à l'adresse suivante : www.meuse-en-fete.mobi/admin.
Les sessions PHP sont utilisées afin de stocker les informations de connexion sur le serveur (identifiant de session en cours, nom de la session en cours, ...). Cela sécurise fortement l'accès aux données car, contrairement aux cookies, les données ne sont pas stockées sur l'ordinateur de l'utilisateur.
En fait, les informations sont stockées dans un fichier sur le serveur. Pour chaque session, un nouveau fichier est créé, chacune possède un identifiant généré aléatoirement. On peut indiquer à la session qu'elle soit détruite après un certain laps de temps, sinon elle est détruite automatiquement à la fermeture du navigateur.
En pratique, lorsqu'on se connecte au module d'administration de Meuse en Fête App une session PHP démarre. Un identifiant de session lui est attribué, par exemple : *12345*, et est stocké dans une variable. Cet ID est valable pour toutes les pages de l'administration tant que le navigateur est ouvert. Au début de chaque page PHP de l'administration, on vérifie si l'ID de la session en cours est bel et bien égal à celui stocké en variable. S'il est différent une

redirection vers la page d'accueil de l'administration est effectuée où l'utilisateur devra encoder son login.

Exemple :

```php
<?php
if($_SESSION['id']!=session_id())
    header("Location: ../index.php");
}
?>
```

Donc, quand on quitte le navigateur, la session est détruite. Si on ouvre à nouveau le navigateur et qu'on tente d'accéder à la suppression des villes sans passer par le formulaire d'identification, on est redirigé vers ce formulaire car l'ID de session en cours n'est pas identique à celui stocké en variable (en l'occurrence aucun n'a été créé).

JQuery Mobile

Commençons par détailler JQuery : c'est une bibliothèque Javascript/AJAX qui interagit avec les éléments HTML5 de manière plus simple que le Javascript seul. Il utilise notamment l'itération implicite qui permet d'économiser la rédaction de nombreuses lignes de code. Exemple d'itération implicite pour masquer tous les paragraphes du document :

```javascript
// Sans JQuery, Javascript seul
var paragraphe = document.getElementsByName("p");
for(var i in array){
    paragraphe[i].style.display = "none";
}
//En JQuery on écrira simplement
$('p').hide();
```

Passons à JQuery Mobile : il s'agit d'un framework fournissant des bibliothèques d'éléments HTML5, CSS3 et Javascript en utilisant les mêmes librairies que JQuery. Il permet de visionner un site sur des appareils mobiles grâce au Responsive Design de CSS3 (voir ci-dessous). Son système de navigation est basé sur AJAX, gérant notamment des transitions animées entre les différentes pages et utilisant des widgets tels que des barres d'outils, des listes ou des accordéons.

L'usage de JQM pour la conception de l'application Meuse en Fête est vite apparu comme indispensable car il permet un énorme gain de temps au niveau codage et fournit d'emblée un design sobre et esthétique. De plus, il est multiplateforme.

Responsive Web Design

Le Responsive Design est un concept regroupant plusieurs techniques de développement web. Cela permet qu'un même site web s'affiche harmonieusement sur tout type d'appareil. Le style de la page s'adapte à la largeur de l'écran, change la taille de la police, supprime des

contenus devenus inutiles, etc. Le tout de manière dynamique.
Une des techniques du RWD est l'utilisation des *media queries*. Ce sont des règles CSS qui s'activent en fonction de la gamme d'appareil utilisée ou en fonction de la largeur de la fenêtre du navigateur.
Par exemple, si on veut rétrécir de moitié la taille de tous les paragraphes de la page lorsque celle-ci est affichée dans une fenêtre dont la largeur est inférieure à 800 pixels, il faut écrire la règle suivante:

```
@media screen and (max-device-width : 800px){
    p{
        width: 50%;
    }
}
```

@media sert à spécifier le type d'appareil ciblé. *screen* identifiera les écrans.
Pour déterminer si un appareil mobile est utilisé, on remplacera ou on ajoutera à la suite de *screen*, le media *handheld*. Cependant, *handheld* est ignoré par beaucoup de navigateurs mobiles comme Safari de iOS qui le considère comme un media *screen*. Il faut donc agir sur la largeur et la hauteur de l'écran.

Les feuilles CSS présentes dans le dossier regroupant les scripts de JQM intègrent déjà des *media queries* par défaut. Ce qui permet le redimensionnement automatique de la fenêtre et des boutons de l'application.
Cependant le menu principal de l'application n'est pas géré par JQuery Mobile, il est ajouté manuellement, du coup il ne respecte pas le redimensionnement de la fenêtre. Donc, sur une tablette numérique, les icônes ont la même taille que sur un smartphone. Pour pallier ce manque, des *media queries* personnalisées sont ajoutées à la CSS. Elles permettent l'agrandissement des icônes lorsque la largeur et la taille du dispositif correspondent aux dimensions d'une tablette. De plus, on affiche le menu en deux rangées de trois icônes lorsque l'appareil est en position paysage.

Les attributs data et le modèle multipages

Une page JQM n'est pas totalement similaire à une page HTML classique. La page JQM se décompose en divisions typées JQM identifiées grâce aux attributs *data-role*. Ce sont des attributs spécifiques à JQM qui permettent l'interaction entre HTML5 et JQuery. D'autres attributs existent tels que *data-transition* pour la gestion des transitions entre pages, *data-icon* pour l'affichage des icônes dans les boutons, *data-theme* pour le choix du thème.
Grâce à l'attribut *data-role*, on sait dès lors indiquer qu'une telle division est l'en-tête, une autre le contenu ou encore une page. Grâce à une méthode *data()* présente dans la bibliothèque JQuery, JQM peut alors décoder le *data-role* et appliquer le style ainsi que le design responsive correspondant au type de la division.

En annexe 3 se trouve un exemple de code et d'affichage avec les *data-role*.

Grâce à l'attribut *data-role="page",* on indique à JQM que la division est une page. Il la traite donc en tant que telle et l'affiche comme une page à part entière. Il est alors possible d'indiquer dans plusieurs divisions le *data-role="page"* et ainsi avoir toutes les pages dans un seul fichier HTML. C'est le modèle multipages. Cela évite la création de nombreuses pages séparées et réduit le temps de chargement entre chaque page.
L'inconvénient du modèle multipages est que le temps de latence sera plus important au début car il charge toutes les pages-divisions en une fois, mais on obtient un affichage direct des pages lorsqu'on clique sur les boutons du menu.
Cependant, une architecture avec des pages séparées est conseillée lorsque le nombre de pages est important.

Voir en annexe 4 le schéma montrant la différence de structuration des pages entre le modèle multipages et le modèle avec pages séparées.

Theme Roller JQuery Mobile

Theme Roller est un éditeur permettant de customiser aisément chaque élément JQuery utilisé dans les pages HTML. Un Theme Roller JQM est également disponible. Il permet de coloriser les différentes parties comme le header, le footer ou les boutons via une petite interface logicielle en ligne. Cette interface se compose de plusieurs fenêtres représentant une page HTML standard de couleur grise au format mobile. À l'aide d'une palette de couleurs, on peut appliquer plusieurs coloris aux différentes sections de l'affichage.
On peut aussi créer plusieurs thèmes et utiliser des thèmes déjà existants. Quand on a déterminé le choix des couleurs, on choisit la version de JQuery Mobile utilisée (JQM 1.3.0 au 20 février 2013) et on exporte un dossier contenant tous les fichiers CSS créés en fonction des choix effectués dans l'interface du Theme Roller. On place ensuite ce dossier dans celui de l'application et on lie les feuilles de style à la page HTML (voir dans la rubrique *Code de l'application*).
Dans le code HTML, on indiquera dans les balises le thème correspondant (a, b, c, d, ...) grâce à un attribut data.

Exemple:

```
<div data-role="page" data-theme="a" id="accueil">
    ...
</div>
```

En annexe 5, se trouve une capture d'écran du Theme Roller JQuery Mobile.

Google Maps API v3

L'API la plus utilisée dans l'application est celle de Google Maps. Il s'agit d'une cartographie dynamique regroupant l'ensemble des technologies permettant d'afficher une carte sur le web (définition donnée dans le livre *Développer avec les API Google Maps* publié chez *Dunod*).

Le principe est le suivant : côté client, l'utilisateur interroge les serveurs Google via son navigateur web, afin de récupérer la carte désirée. Google Maps lui renvoie les informations sur la carte demandée sous forme d'images appelées *tuiles*. Ces *tuiles* sont mises côte à côte dans le navigateur de l'utilisateur pour afficher la carte. À chaque fois que le client (l'application web) réinterroge les serveurs Google, un zoom ou un déplacement sur la carte par exemple, ceux-ci renvoient les nouvelles informations adéquates ainsi que les nouvelles *tuiles*.

Le meilleur exemple d'utilisation AJAX

Google Maps est aussi un excellent exemple pour comprendre le principe AJAX. En effet, lorsque la carte est affichée et qu'on effectue un déplacement, de nouvelles *tuiles* sont chargées et d'autres sont supprimées sans que la page HTML soit rechargée entièrement. Les interactions entre l'application web et les serveurs Google Maps se font de manière transparente. C'est exactement la base de fonctionnement du système AJAX.

Le schéma ci-contre montre bien le processus au cours du temps des transmissions de données avec et sans AJAX entre une application (client) et un serveur. On peut voir sur la partie basse du schéma que, lors de l'échange des données, l'application continue à être exécutée. Alors que dans la partie haute, modèle classique sans AJAX, l'application est interrompue pendant l'échange des données.

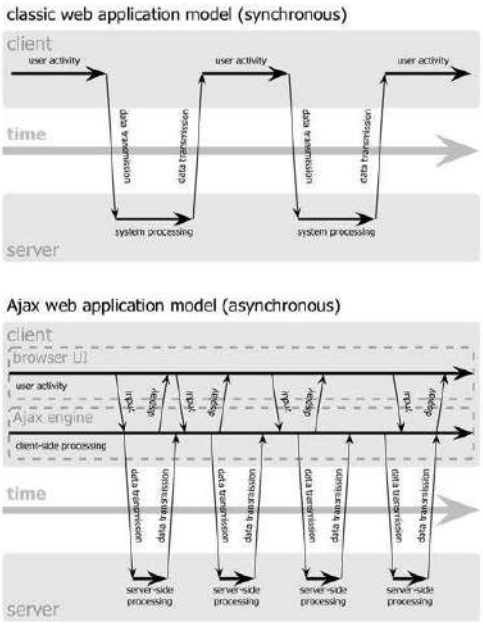


Schéma provenant du site internet : www.scriptol.fr/ajax/ajax-garrett.php

Initialisation de la carte

Dans la page *index.html,* se trouve la page-division carte dans laquelle on place une division vide `<div data-role="content" id="map_canvas">` . C'est dans cette section que le script *carte.js* génère la map.

Veuillez vous référer à la rubrique *Code de l'application* pour plus de précisions sur le codage.

Pour l'initialisation, le script possède une fonction *initialize()* qui décrit la carte. À la base, il définit 3 variables contenant des objets:

- *latlng*: permet de spécifier un point de coordonnées GPS utilisé pour centrer la carte.
- *myOptions*: contient les options de la carte comme la valeur du zoom, le point de centrage de la carte (*latlng*) et le type de carte à utiliser (ROADMAP pour les cartes routière, SATELLITE pour les cartes vue du ciel, etc.). Depuis l'API version 3 de Google Maps, les options de la map sont spécifiées à la création de l'objet.
- *map*: sert à détecter la division vide (*map_canvas*) dans laquelle la carte Google va être affichée grâce à la méthode Javascript *document.getElementById()*.

Voici la fonction:

```
var lat_carte=50.34896578114507;
var lon_carte =4.8779296875;
var zoom_carte = 9;

function initialize(){
    var latlng = new google.maps.LatLng(lat_carte,lon_carte);
    var myOptions = {
        zoom: zoom_carte,
        center: latlng,
        mapTypeId: google.maps.MapTypeId.ROADMAP,
        backgroundColor: '#fff',
        disableDefaultUI: false,
        DoubleClickZoom: false,
        draggable: true
    };
    map =
    new google.maps.Map(
        document.getElementById("map_canvas"),
        myOptions
    );
    detectBrowser();
}
```

*new google.maps.** permet la création des objets qui sont présents dans la bibliothèque Google. Pour utiliser ces nommages d'objets il faut placer la référence au script suivant dans *index.html* avant la référence au script *carte.js*:

```
<script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false">
</script>
```

Il est également possible d'afficher différemment la carte en fonction de l'appareil utilisé. Par exemple, des boutons de zoom mieux adaptés aux téléphones mobiles.
Pour ce faire, une fonction spéciale est utilisée à la fin de *initialize()*. Elle permet d'identifier le type de navigateur grâce à l'objet Javascript *navigator.userAgent* . Quand le navigateur est de type "iPhone" ou "Android", on met à jour les options de la carte. Plus précisément, le style d'affichage du contrôle de navigation.
Dans le code, la fonction s'appelle *detectBrowser()*:

```
function detectBrowser(){
    var useragent = navigator.userAgent;

    var mapdiv = document.getElementById("map_canvas");

    if(useragent.indexOf('iPhone')!=-1||useragent.indexOf('Android')!=-1){
        mapdiv.style.width = '95%';
        mapdiv.style.height = '92%';
        myOptions = {
            navigationControlOptions : {
                style : google.maps.NavigationControlStyle.SMALL
            },
            mapTypeControlOptions : {
                style : google.maps.MapTypeControlStyle.DROPDOWN_MENU
            }
        };
        map.setOptions(myOptions);
    }
}
```

PS: Les options décrites ci-dessus sont les plus courantes. Cependant, il existe une multitude d'options pouvant être utilisées. Elles sont toutes présentes dans la documentation de Google Maps à l'adresse suivante:
https://developers.google.com/maps/documentation/javascript/reference .

Affichage des marqueurs et infobulles

Les marqueurs sur une carte Google Maps sont le meilleur moyen d' indiquer la position géographique d'un lieu. De plus, quand ceux-ci sont associés à l'affichage d'une infobulle, ils deviennent souvent la raison d'être de la carte Google Maps.

Les marqueurs se paramètrent aussi dans la fonction *initialize()* mais après la définition de la map. La variable représentant le marqueur contient un objet *google.maps.Marker* contenant la position du lieu (ou de l'évènement dans le cas de l'application Meuse en Fête) et le nom de la map sur laquelle le marqueur doit être affiché.

D'autres options peuvent lui être ajoutées. Par exemple :

- *clickable*: indique si le marqueur est cliquable ou non.
- *cursor*: spécifie le type de pointeur utilisé. Comme un curseur, une flèche, une main, etc. (aucun intérêt dans le cas d'une application mobile car aucun curseur n'est présent sur un écran tactile).
- *draggable*: permet de rendre le marqueur déplaçable sur la carte.
- *flat*: active l'ombre du marqueur sur la carte.
- *title*: affiche un titre lors du survol sur la carte (aucun intérêt dans le cas d'une application mobile car le survol n'existe pas).
- *icon*: permet de changer l'image du marqueur et de la remplacer par une image personnalisée. Il s'agit d'un objet dans lequel on indique l'URL, la taille de l'image, etc. Cette option est utilisée dans l'application et remplace les marqueurs roses, par défaut, par des marqueurs bleus plus en accord avec le thème de Meuse en Fête.
- *shadow*: permet de personnaliser l'ombre de l'image du marqueur.

Concernant les infobulles, celles-ci s'affichent lorsque l'utilisateur clique (touche) le marqueur. Pour que cette interaction soit réalisée, il faut créer un évènement de type *click*. Celui-ci s'écrit comme suit:

```
google.maps.event.addListener(markerFtpn,'click',function(){
    infosMarkerFtpn.open(map,markerFtpn);
    infosMarkerEvent.close();
});
```

Description : lorsque le clic sur le marqueur *markerFtpn* est détecté, on ouvre l'infobulle de ce marqueur présent sur la carte *map*. On ferme toutes les autres infobulles ouvertes sur la carte pour n'afficher que celui cliqué.

Il faut également prévoir la fermeture de l'infobulle lors du clic sur la carte. Voici le code, qui est similaire à celui de l'ouverture de la bulle d'informations:

```
google.maps.event.addListener(map,'click',function(){
    infosMarkerFtpn.close();
});
```

Les méthodes *open()* et *close()* présentes dans les codes ci-dessus se réfèrent à l'infobulle en question. Celle-ci est écrite de la manière suivante:

```
infosMarkerFtpn = new google.maps.InfoWindow({
    content: '<p>Fédération du Tourisme de la Province de Namur ... ',
    disableAutoPan: false,
    maxWidth: 40
});
```

La variable *content* est évidemment le contenu de la bulle, *maxWidth* indique la largeur en pixels, et *disableAutoPan*, s'il est initialisé à *false*, permet de déplacer la carte si une partie de l'infobulle est masquée par le bord de la map.

Wunderground API

Cette API s'avère très utile pour l'application. Elle permet de récolter les données météorologiques d'un endroit donné. Par ailleurs, elle est très simple d'utilisation.

Pour débuter avec Wunderground, il faut se rendre sur le site internet www.wunderground.com/weather/api/ .
Contrairement à Google Maps une inscription est nécessaire pour pouvoir utiliser la bibliothèque et obtenir une clé API (chaîne de caractères formée de chiffres et de lettres). Cette clé permet l'utilisation de l'API en la spécifiant lors des appels aux serveurs de Wunderground. Dans la partie *Documentation* du site, sont listées sur la gauche toutes les requêtes possibles de l'API. Détaillons certaines d'entre elles:

- *forecast*: retourne un résumé des prévisions météorologiques pour les trois prochains jours. On peut connaître les minimas et les maximas, le taux d'humidité (min-max), les dates, le type de conditions. Il y a aussi moyen de récupérer l'URL d'une image représentant la météo actuelle.
- *forecast10day*: pareil que *forecast* mais pour les dix prochains jours.
- *satellite*: retourne l'URL d'une image satellite infrarouge représentant l'évolution météorologique.
- *conditions*: renvoie toutes les informations météorologiques courantes.
- *history*: permet de récolter les informations météorologiques d'une date précise.
- *geolookup*: récupération des coordonnées GPS et autres d'un lieu.

Ensuite, il faut spécifier l'endroit pour lequel on veut récolter les informations météorologiques.
Ce paramétrage est réalisé dans l'URL passée à la requête AJAX du script. Celle-ci doit se composer de la façon suivante:

http://api.wunderground.com/api/
[clé API]/
[une ou plusieurs requêtes séparées par '/']
/lang:FR/q/
[nom du pays]/
[nom de la ville].json

Une requête réelle donne:

http://api.wunderground.com/api/**e01ff0ddf3fd0000**/**geolookup/conditions/forecast**/lang:FR/q/**Belgique**/**Namur**.json

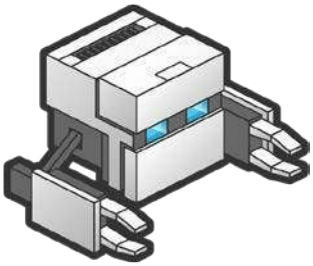
Puis, dans le script *meteo.js* de l'application, est exécutée la requête AJAX composée de trois paramètres : *url* qui sert à envoyer la requête ci-dessus, *dataType* qui indique le type de données que Wunderground doit renvoyer (dans notre cas du JSON) et *success* qui permet de récupérer les données de l'objet JSON et à les placer dans des variables.

Veuillez vous référer à la rubrique S*cript météo* pour plus de détails.

PS: Wunderground API est gratuite d'utilisation mais sous certaines conditions. D'abord, il faut afficher le logo de Wunderground à côté des données affichées dans la page.
Ensuite, le nombre de requêtes est limité à 10 par minute et 500 par jour en version gratuite. Donc, si dix utilisateurs visionnent l'application en moins d'une minute, la clé sera désactivée pour le reste de la journée.
Si on veut éviter cela, il faut choisir un abonnement payant. Par exemple, 20$ par mois pour 5000 appels aux serveurs par jour et 100 requêtes par minute.

PhoneGap Build

Le sujet du TFE traite d'une application web mobile. Donc, une application dont le design est adapté au mobile et qui doit être ouverte dans un navigateur web.
Cependant, il est aussi important de s'intéresser à la réalisation d'une application hybride (voir chapitre *Contenu et structure*). Cela permet la publication de l'application dans le Play Store (boutique en ligne des applications mobiles Android) et/ou dans l'Apple Store (boutique en ligne d'Apple). Il est alors possible d'installer l'app sur son appareil et de la démarrer depuis le menu.



Pour créer cette version hybride, il faut l'aide de PhoneGap Build. PhoneGap Build est une version en ligne du framework PhoneGap qui utilise les mêmes bibliothèques que ce dernier pour construire une application native à partir de fichiers HTML, CSS et Javascript.

Le principe est simple. Lorsque le site est terminé, il suffit de compresser le dossier contenant les pages et les scripts au format ZIP. Attention, à cause des appels AJAX, il ne doit y avoir aucun code PHP dans les pages. Celles-ci doivent être au format HTML.
De plus, il faut parfois modifier certaines lignes de code. En effet, dans Meuse en Fête app, lors de l'activation de certains boutons, une redirection de page est effectuée pour passer des paramètres dans l'url. Par exemple : le nom de la ville, la latitude, la longitude, etc.
La redirection se fait via la fonction *window.location.replace()* et ressemble à ceci:

```
window.location.replace("www.meuse-en-fete.mobi/meuse/index.html?lat= ... #agenda");
```

Pourtant l'adresse de redirection ne sera pas la même lorsque l'application sera compilée par PhoneGap Build. Elle devra ressembler à ceci:

```
window.location.replace(file:///android_asset/index.html?lat=...#agenda);
```

En effet, le dossier racine de l'application n'est plus le dossier 'www' du site internet mais bien le dossier 'android_asset' de la compilation.

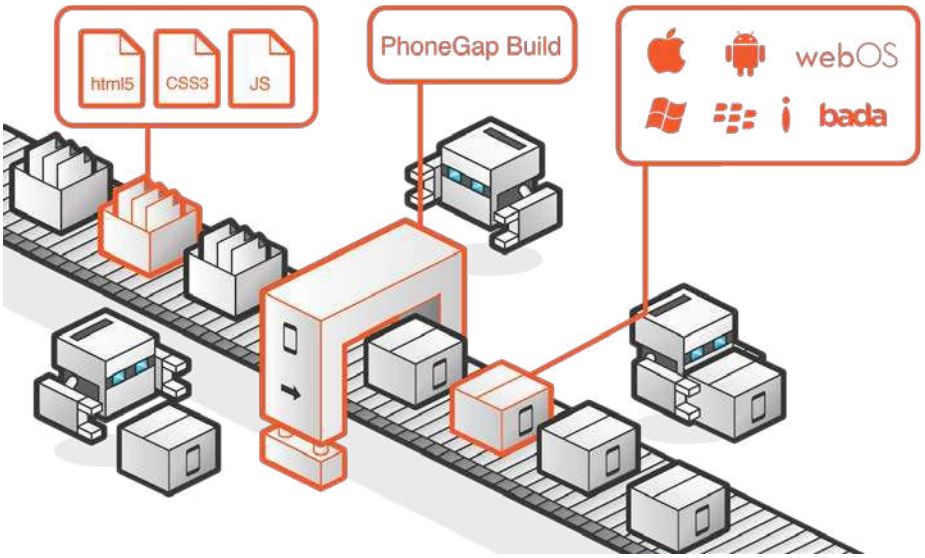
Enfin, il y a encore une dernière chose à effectuer avant la compression. Si on veut que l'application possède une icône et un titre personnalisés dans le menu, il faut le spécifier dans un fichier *config.txt* à placer à la racine du site (fichier différent pour iOS).

Dans ce fichier, il est également possible d'indiquer les préférences suivantes :

- La description de l'application
- Le nom de l'auteur
- Bloquer le basculement en mode paysage de l'écran
- Spécifier une image à afficher pendant le chargement de l'application lors de son démarrage
- Activer ou désactiver les accès aux fonctionnalités du téléphone

Concernant le dernier point, ces accès ne sont possibles que si la librairie *cordova-2.2.0.js* a été liée à la page *index.html*. Si cela a été fait, on peut, par exemple, détecter si la connexion réseau est activée ou encore accéder à l'appareil photo du smartphone depuis l'application.

Image représentant le principe de fonctionnement de PhoneGap Build :



Lorsque toutes les indications précédentes ont été effectuées, on peut compresser le dossier au format ZIP.

On se rend ensuite sur le site web de PhoneGap Build (https://build.phonegap.com). Comme cette technologie a été rachetée par Adobe, il est nécessaire d'avoir un compte Adobe pour se connecter et utiliser le logiciel en ligne.

Dans l'interface, en cliquant sur *Applications,* on a un formulaire qui permet de télécharger le fichier ZIP. Quand cela est fait, l'icône de l'application, son nom et sa description s'affichent. À côté, deux boutons apparaissent, ils permettent soit de supprimer l'app du

stockage en ligne de PhoneGap Build, soit de générer l'application.
On clique sur *Prêt pour la génération* (*Ready to build* en anglais).

PhoneGap Build compile alors le fichier pour créer une application native à partir d'une application web pour les différentes plateformes mobiles (iOS, Android, Windows Phone, Blackberry, HP et Symbian). Cela peut prendre un certain temps.

Quand le processus est terminé, il suffit de scanner le QR code situé à droite de l'écran avec un smartphone, et le fichier compilé correspondant au système d'exploitation de l'appareil se télécharge automatiquement (format *apk* pour la plateforme Android).

Pour finir, il ne reste plus qu'à installer le fichier téléchargé sur le smartphone et l'icône de l'application apparaîtra dans le menu.

Une restriction supplémentaire est présente pour la génération de l'application pour iOS. Il est nécessaire d'avoir une clé de signature qui permet d'activer la compilation de l'application. Mais pour obtenir cette fameuse clé, il est nécessaire de posséder un compte développeur chez Apple (99$/an). On ne peut donc tester directement l'application sur son iPhone ou iPad sans inscription chez Apple.

NB : Depuis la dernière suite CS6 d'Adobe, Dreamweaver intègre un service qui permet de générer et de tester le fichier compilé par PhoneGap Build directement dans le logiciel.

Code de l'application

Dans cette partie, un schéma décrivant le processus d'action de chaque script se trouve au début de certaines rubriques. Vous pouvez également vous référer à l'arborescence du projet pour comprendre les explications données ci-dessous.

Page principale (index.html)

Cette page HTML est le point d'entée du site et elle est la seule du dossier. C'est la page dans laquelle vont s'afficher toutes les données de l'application. Comme expliqué dans la description pratique de JQuery Mobile, cette page est composée de plusieurs divisions utilisant *le data-role="page"* qui permet à JQM d'interpréter ces divisions en tant que page. Rappelons qu'il s'agit du modèle multipages.

Dans la partie `<head>` du fichier, se trouvent tous les scripts Javascript permettant la gestion du contenu HTML, ainsi que les feuilles de styles CSS mettant en forme les éléments HTML. Pour les scripts JS, on place toujours en premier la libraire Phonegap (*cordova-2.2.0.js*), puis les scripts contenant les bibliothèques JQuery et JQuery Mobile. Comme les méthodes JQuery sont utilisées dans les scripts de l'application, il est primordial de les déclarer avant les scripts personnalisés (*agenda.js*, *carte.js*, ...). De même, le script Google Maps doit être appelé avant *carte.js* afin que les méthodes Google Maps soient utilisables dans ce dernier.

La partie `<body>` contient les éléments qui s'affichent dans le navigateur. Ce sont donc les différentes divisions interprétées en tant que page par JQuery. L'application affiche à l'écran automatiquement la première division qui est la page d'accueil avec les différents boutons menant vers les sous-menus.

Comme expliqué dans les pages précédentes, la division est identifiée par son `data-role="page"` et contient deux sous-divisions : une représentant l'en-tête (`data-role="header"`) et une représentant le contenu (`data-role="content"`).

Dans l'en-tête, on indique, grâce aux `data-role`, qu'il a une position fixe (il restera au sommet de la page lors du défilement de celle-ci). L'attribut `data-tap-toggle="false"` signifie que le header ne doit pas être masqué lorsqu'on tape une fois sur l'écran (par défaut : initialisé à *true*).

Le contenu comprend un tableau HTML dans lequel les images-boutons du menu sont présentes. Ces boutons sont entourés par la balise `<a>` qui sert à établir le lien vers la page correspondante (pour rappel, la page qui n'en est pas vraiment une). Ce lien est défini par l'attribut `href="#agenda"`.
#agenda indique l'identifiant de la division-page à afficher lors du "tapotement" sur celui-ci.
La balise <a> contient les *data-role* suivants :

- `data-transition`, qui permet de spécifier une transition entre les changements de page
- `data-prefetch`, qui sert à précharger la page suivante pour plus de rapidité

Les divisions sont identifiées grâce à l'attribut *id*. Par exemple, la division-page de la carte sera distinguée par *id="carte"*.

Dans les divisions qui suivent celle de l'accueil, chacune a soit un contenu dynamique rempli par un script (*index.js*, *agenda.js*, *carte.js*, *villes.js* et *recherche.js*), soit un contenu fixe comme la division *À propos* et la division *Recherche*, qui est aussi éditée par un script (voir ci-après).

Les CSS et polices

Le dossier CSS (voir l'arborescence) contient les feuilles de styles de JQuery Mobile (utilisant le responsive design), un dossier contenant des polices téléchargées sur Google Fonts, un dossier contenant les images utilisées par JQM et une CSS générale *style.css* qui formate le design de l'application.

Les images

Les images utilisées dans l'application comme les images des boutons du menu sont stockées dans le dossier *pix*.

Les thèmes

Le dossier *themes* contient les scripts et images du ou des thèmes générés par le Theme Roller.

Script principal (index.js)

Ce script est le premier à être exécuté dans la page HTML. Toutes les actions de ce script sont contenues dans une fonction JQuery `$(document).ready()`. Cette dernière indique qu'il faut attendre le chargement complet de la page avant d'exécuter le code Javascript situé dans la fonction.

Le script commence par une méthode JQuery *$.ajaxSetup* qui permet d'afficher (et de masquer) l'image de chargement lorsque les données sont en train d'être téléchargées dans l'application.

Ensuite, débute une fonction *$('.agendaBtn').bind('click',function(){...})* qui permet de placer un évènement *click* sur le bouton agenda du menu. Le but de cette fonction est d'effacer les paramètres passés dans l'URL lors du *click*.

```
$('.agendaBtn').bind('click', function (n) {
    var _url = $(location).attr('href');
    var _arr_url = _url.split('?');
    if (_arr_url.length >= 2) {
    window.location.replace(_arr_url[0]+"#agenda");
        return false;
    }
});
```

Une fonction *click* déjà formatée existe également. Dans le code, elle est utilisée pour effectuer le rafraîchissement de la page lors du *clic* sur le bouton *refresh*.

```
$('.refreshButton').click(function() {
    location.reload();
});
```

Une autre méthode *bind()* est utilisée lorsqu'on clique sur le bouton "retour" situé dans l'en-tête. En fait, la fonction *bind()* permet de changer le comportement par défaut d'un évènement. En effet, le bouton *back* (ou "retour") renvoie par défaut vers le menu principal. Mais dans certains cas, lorsqu'on se trouve dans le menu *Villes* par exemple, et qu'on accède aux informations de la ville ou à la carte de la ville, l'utilisateur n'a pas forcément envie d'être redirigé vers le menu principal quand il sélectionne le bouton "retour". C'est pourquoi grâce à cette fonction *bind(),* on peut spécifier pour chaque bouton *back* de chaque sous-menu, une action différente à effectuer.
On passe dans l'url les paramètres de localisation (latitude, longitude, ...), le nom du menu en cours et d'autres variables permettant d'identifier la section dans laquelle on se trouve, et vers quelle section il faut être renvoyé lors du retour.

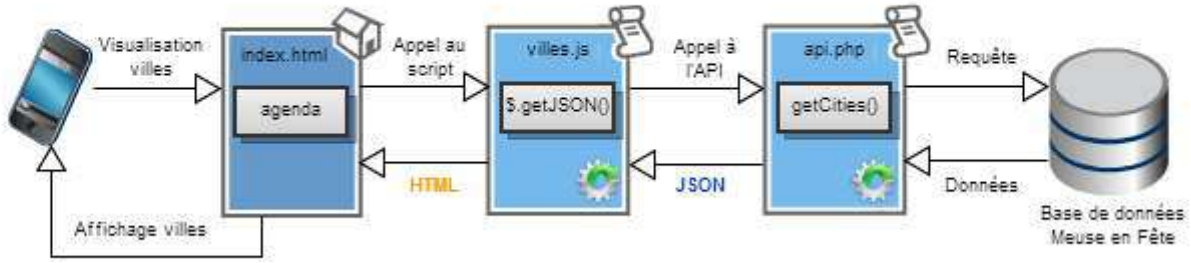
Pour finir, le script possède une fonction *$.getJSON* qui a déjà été développée dans la description pratique au sein de la rubrique *Utilisation du flux XML*. Cette fonction interroge l'API avec la méthode *getCities* pour récupérer les données sur les villes créées dans la partie *administration*. Ces villes sont ajoutées une à une à la division ayant pour identifiant *id="listCities"* qui elle-même est comprise dans la division-page *id="villes"*. Ainsi lorsqu'on clique sur le bouton *Villes* du menu, on obtient un affichage direct (sans chargement) de la liste des villes.

Remarques :

1. Dans les lignes ci-dessus et dans les paragraphes qui vont suivre, j'utilise le terme "cliquer" mais il va de soi que pour les dispositifs mobiles tactiles, il faut parler de "tapotement".
2. En fin de script est présente une fonction *defaultDiacriticsRemovalMap()* qui retire les accents et caractères spéciaux d'une chaîne de caractères, souvent mal gérés par les langages informatiques.
   Cette fonction est souvent utilisée dans les scripts de ce projet.

Script villes (villes.js)

Schéma représentant les étapes du flux d'actions depuis l'envoi d'une requête jusqu'à l'affichage des données renvoyées en passant par les différents scripts JS et PHP.



Quand on a effectué un choix dans la liste des villes, une nouvelle page affiche les renseignements de la ville sélectionnée. Ces informations se répartissent dans quatre sous-menus :

- *Infos* : description de la ville et coordonnées de l'office du tourisme
- *Plan* : carte Google de la ville avec le marqueur de chaque évènement
- *Agenda* : liste de tous les évènements se déroulant dans la localité
- *Météo* : prévisions météorologiques pour la ville en question

Les données d'une ville sont ajoutées par le script *villes.js* à la division *id="ville"*.
À ne pas confondre avec *id="ville**s**"* qui est la liste des villes.

En fait, quand la ville a été sélectionnée dans la liste, des paramètres sont passés dans l'url de la page. C'est grâce à ces paramètres que *villes.js* sait quelle ville il doit traiter.

On peut voir dans le script en annexe 6 l'appel à l'API dans la fonction *$.getJSON()* via une méthode *getCities()*. Une boucle est effectuée sur le résultat renvoyé et une condition gère l'affichage des données filtrées.

Cette condition est très importante, elle permet de comparer le nom de la ville qui est en train d'être parcourue par la boucle *for* (*name*), et le nom de la ville qui est présent dans l'URL (*nameGet*). *nameUrl* est le *nom* de la ville formaté pour le comparer au nom récupéré dans l'URL *nameGet* (suppression des accents et conversion en minuscule).
On récupère le *nameGet* grâce à la fonction *getURLParameter()*. Celle-ci sera abondamment utilisée dans les différents scripts.

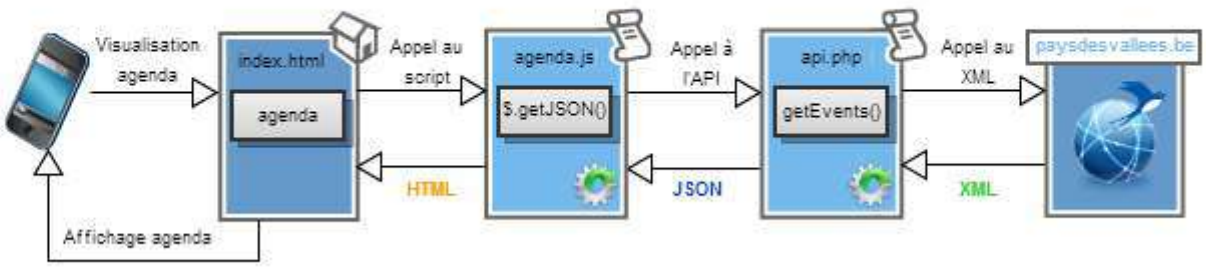
Lorsque la condition devient vraie, elle exécute le code situé en son sein. Le nom de la ville est ajouté au *content* par la méthode *html()*, les liens des sous-menus sont également ajoutés mais via la méthode *append()*.

Ensuite, les informations du sous-menu *Infos* sont déjà ajoutées à la division-page id="villeInfos" afin que la page s'affiche directement sans chargement des données.

Les données des autres sous-menus sont chargées par les autres scripts. Par exemple, la carte de la section *Plan* sera chargée dans la division-page *id="carte"* par *carte.js*, la même division qui est utilisée pour l'affichage de la carte principale depuis le menu général.
En effet, il est plus pratique de réutiliser une division déjà existante que d'en recréer une nouvelle et de surcharger le code.

Script agenda (agenda.js)

Schéma représentant les étapes du flux d'actions depuis l'envoi d'une requête jusqu'à l'affichage des données renvoyées en passant par les différents scripts JS et PHP.



Ce script est en grande partie la raison d'être de l'application. Il permet l'affichage de tous les évènements, d'une sélection d'évènements ou d'un évènement précis relatif à une ville.

D'abord, il faut noter que le code de *agenda.js* est englobé dans une fonction générale *initializeAgenda()* qui est exécutée uniquement quand la page du menu *Agenda* est affichée. Pour ce faire, on utilise dans la division *id="agenda"*, présente dans *index.html*, le code Javascript suivant:

```
<script type="text/javascript">
    $("div:jqmData(role='page'):last").bind('pageshow',function()
        {
            initializeAgenda();
        }
    );
</script>
```

Cette fonction *initializeAgenda()* commence par vider le contenu de la liste des évènements grâce à la méthode:

```
$('#agenda .content #listEvents').html("");
```

Si on ne place pas cette instruction au début, les évènements s'ajouteront à la suite des autres à chaque fois qu'on accédera au menu *Agenda*.

Puis, on définit la variable *all*. Elle permet de savoir le type d'agenda à afficher. *agenda.js* n'est pas seulement utilisé pour le menu *Agenda*, il est aussi appelé dans l'agenda des villes ou encore pour l'affichage d'un évènement précis sélectionné dans la liste des évènements.
Donc, on va analyser l'URL et utiliser plusieurs fois la fonction *getURLParameter()* pour détecter si tel ou tel paramètre se trouve dans l'URL.
Par défaut *all* est initialisé à 1, ce qui veut dire que l'agenda doit afficher tous les évènements.
Si *getURLParameter()* trouve comme paramètre *ville* dans l'URL, alors on affectera la valeur 0 à *all*, on récupère la valeur du paramètre dans *villeCur* et on affecte une division spéciale au *content* de la division *id="agenda"* pour lister les évènement de la ville donnée.
Si le paramètre *idEvent* est identifié dans l'URL, alors *all* est redéfini à -1. Le contenu de *idEvent* est placé dans une variable *idCur*.

Enfin, si dans l'URL, les paramètres *start* ou *end* ne sont pas trouvés, on initialise ces paramètres respectivement à 0 et à 30. S'ils s'y trouvent, on récupère leur contenu et on les affecte aux variables de même nom *start* et *end*. Pourquoi? A la base quand *all* était égal à 1, l'ensemble des évènements était affiché, soit environ 80 éléments. Dès lors, un problème de lenteur survenait et l'écran se figeait pendant une quarantaine de secondes lors du chargement, étant donné l'énorme quantité d'informations présente dans le flux XML. En utilisant les variables *start* et *end*, le chargement est limité par tranche de 30

évènements. Il suffit juste d'ajouter deux boutons *Précédent* et *Suivant* afin de naviguer vers les 30 évènements antérieurs ou postérieurs à la liste actuelle.

Ensuite, la méthode *$.getJSON*, décrite dans les chapitres précédents, est utilisée avec la méthode *getEvents()*. Une boucle est effectuée, on ne retient que les évènements concernant Meuse en Fête et on extrait les données de chacun en les formatant ensuite pour l'affichage.
En annexe 7, se trouve un extrait de code montrant la condition de filtrage de la liste des évènements. Elle ne "laisse passer" que les évènements de type "La Meuse en fête".

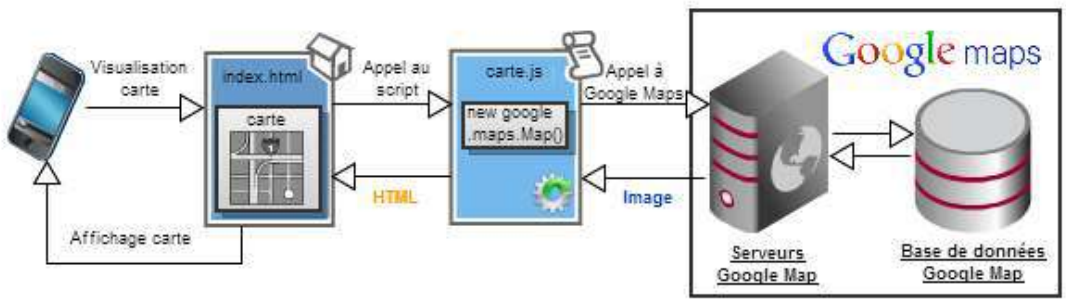
Enfin, on affiche le ou les évènement(s) en fonction de ce qui a été décidé auparavant grâce à la fonction *getURLParameter()* .
Si *all=1*, on affiche chaque évènement parcouru par la boucle mais uniquement ceux compris entre les variables *start* et *end*. Donc, lors du premier affichage du menu *Agenda*, entre 0 et 30.
Si *all=0*, on affiche uniquement les évènements dont le nom de la ville, récupéré dans l'URL, est égal à celui de la ville de l'évènement parcouru par la boucle.
Si *all=-1*, seul l'évènement dont l'identifiant est égal à *idCur* est affiché dans la fenêtre. Cela se produit lorsqu'on sélectionne un évènement dans la liste pour afficher ses détails.

Remarques :

1. Quand *all* est égal à -1 et que l'évènement concerné a été trouvé, l'instruction *return false;* doit être placé à la fin de la condition pour indiquer à la boucle *$.each* qu'elle n'a plus besoin de parcourir le reste de la liste. Attention, l'instruction *break* qui est couramment utilisée dans d'autres langages de programmation pour ce genre d'interruption ne fonctionne pas dans ce cas.
2. Notez l'utilisation de la méthode *trigger('create')* à la fin de chaque méthode *append()* ou *html()*. Elle permet le redéclenchement de la création de l'objet JQM afin que les modifications sur celui-ci soient appliquées.

Script carte (carte.js)

Schéma représentant les étapes du flux d'actions depuis l'envoi d'une requête jusqu'à l'affichage des données renvoyées par Google Maps.



Comme dans *agenda.js*, une fonction *initialize()* permet l'exécution du code uniquement lorsque la division-page correspondante est affichée. Un ensemble de variables est déclaré avant cette fonction. Les déclarer au début de cette façon, les rend accessibles à n'importe quel endroit du code. On parle de variables globales. Si elles avaient été déclarées au début de la fonction *initialize()*, en tant que variables locales, elles n'auraient été utilisables qu'uniquement dans cette fonction.
Voir annexe 8.

Dans *initialize(),* les coordonnées du point de centrage de la carte sont récupérées.
Plusieurs cas sont envisageables :

- Les paramètres *lat* et *lon* (latitude et longitude) n'existent pas dans l'URL. Ils sont alors définis par défaut avec un zoom moyen. Ce sont les paramètres de la carte générale accessible depuis le menu principal.
- Les paramètres *lat* et *lon* sont définis, un paramètre *ville* est trouvé mais ni *idEvent*, ni *agenda_ville* n'existent dans l'URL. La latitude et la longitude sont extraites de l'URL et la valeur 14 est attribuée au zoom. Ce sont les paramètres de base permettant l'affichage du plan de la ville.
- Les paramètres *lat* et *lon* sont définis, aucune ville n'est trouvée dans l'url ou un *idEvent* est détecté dans l'URL. Les paramètres qui suivent sont ceux qui sont utilisés pour l'affichage sur la carte d'un évènement précis depuis l'agenda général.
- Les paramètres *lat* et *lon* sont définis, un paramètre ville est trouvé dans l'URL et le paramètre *agenda_ville* est présent dans l'URL. Les paramètres qui suivent sont ceux qui sont utilisés pour l'affichage sur la carte d'un évènement précis depuis l'agenda d'une ville.

Une fois que tous les paramètres sont déterminés, on attribue ces données à l'objet Google Maps pour créer la carte souhaitée.

À la fin de *initialize()*, se trouve la fonction *addEvent()* qui permet l'ajout d'un marqueur et d'une infobulle pour chaque évènement sur la carte. Cette fonction utilise à nouveau la méthode JQuery *$.getJSON()* qui a recours à la méthode *getEvents()* et qui va boucler sur les évènements du flux XML pour en extraire ceux de Meuse en Fête.

Script recherche (recherche.js)

Ce script est un mix d'*agenda.js* et de *villes.js*. Il récupère la liste des villes et des évènements via l'API, respectivement par les méthodes *getCities()* et *getEvents*.
La section correspondant à ce script est la division-page *id="recherche"*. Celle-ci contient déjà du code HTML. En effet, une division *id="mypanel"* est présente au début de la section servant à définir le panneau de recherche d'évènements, accessible depuis le bouton

"Rechercher" présent dans le corps de la page. Ce panneau utilise un attribut *data-role="panel"* qui est interprété par JQM en tant que tableau coulissant.
Se référer à l'annexe 9 pour visionner le code HTML du panneau coulissant.

Dans ce menu, on peut rechercher des évènements à proximité d'une ville ou de la position actuelle de l'utilisateur. On spécifie aussi un rayon de kilomètres dans lequel l'évènement doit se trouver et à quelle date il doit avoir lieu.

Les villes présentes dans la liste déroulante sont récupérées depuis l'API. Ce sont les villes autour desquelles on peut rechercher des manifestations. Pour utiliser l'option "Ma position", le système de géolocalisation doit être activé. Sinon, un message indiquant "Aucun résultat pour les critères sélectionnés" s'affiche.

Une fois les critères correctement définis et la recherche exécutée, le panneau se ferme et l'application identifie les évènements satisfaisant aux informations spécifiées. Cette action s'effectue lorsqu'on clique sur le bouton "Chercher" présent dans le panneau qui réagit au clic grâce à la fonction suivante:

```
$("#searchPanel_btn").click(function(){ ... }
```

Ensuite, lorsque les paramètres sont récupérés, ils sont utilisés dans la boucle parcourant les évènements du flux XML afin d'afficher les manifestations respectant la date choisie et le rayon de kilométrage autour de la ville sélectionnée. Les évènements répondant aux conditions sont affichés dans le contenu HTML à la place du bouton "Rechercher".

Si tous les critères ont été sélectionnés mais qu'aucun évènement ne correspond, le message "Aucun résultat pour les critères sélectionnés" s'affiche à nouveau.
Dans le script, la variable *empty* vérifie qu'au moins un évènement a été trouvé.

Remarque :

1. Pour calculer la distance entre le point d'origine du rayon de kilométrage et son extrémité, la fonction *getDistance()* est utilisée. Notez la présence de l'objet Math (bibliothèque Javascript) qui permet l'utilisation de fonctions mathématiques (PI, cosinus, arrondi, racine carrée, ...).

Script météo (meteo.js)

Ce script permet la récupération des données météorologiques depuis l'API Wunderground (voir rubrique *Wunderground API*).

Comme dans *agenda.js,* le code est contenu dans une fonction d'initialisation afin que celui-ci ne soit exécuté qu'au moment de l'affichage de la division-page *id="villeMeteo"*.

Dans cette fonction, la première opération consiste à récupérer le nom de la ville se trouvant dans l'url.
Ensuite, la méthode JQuery *$.ajax* est invoquée. Elle charge une page externe dans la page HTML. Elle contient 3 paramètres: *url*, *dataType* et *success*.
L'*url* est l'adresse de la page depuis laquelle les informations doivent être chargées. Dans cette URL, se trouvent les méthodes de l'API Wunderground décrites dans le chapitre précédent.
Le *dataType* est le type de données renvoyées par l'API, ici du JSON.
Pour terminer, *success* est la fonction qui est exécutée et qui récupère les données JSON lorsque l'API renvoie bien les données demandées.

success contient donc une fonction avec en paramètre *parsed_json* qui représente les données sous forme JSON.
Pour récupérer le taux d'humidité par exemple, on doit accéder au tableau *relative_humidity* compris dans *current_observation* lui-même contenu dans le parsed_json.
L'accès à cette donnée est donc réalisée par l'instruction :

```
parsed_json['current_observation']['relative_humidity'];
```

De cette manière, nous obtenons toutes les informations nécessaires comme les prévisions à 3 jours, la température et même les icônes représentant la météo actuelle ou prévue pour la ville désignée.

Lorsque toutes les données sont récupérées, les informations, formatées en HTML via la méthode *append()* de JQuery, sont ajoutées à la division *id="widgetMeteo"*.

Scripts JQuery et JQuery Mobile

Le dossier *js* contient également les scripts des bibliothèques JQuery et JQM qui permettent l'utilisation des méthodes comme *append()* et *html()* ou le formatage des éléments HTML grâce aux attributs *data*.

Pour aller plus loin

Lorsque l'application web mobile est finalisée, il est possible de transformer cette web app en application hybride (mi-native, mi-web) grâce au logiciel en ligne PhoneGap Build comme mentionné dans la *Description pratique*. La bibliothèque PhoneGap donne accès aux fonctionnalités du dispositif mobile, nous permettant de vérifier l'état de la connexion, par exemple, et d'afficher un bandeau transparent par-dessus la page demandant d'activer le Wifi ou la 3G dans le cas où aucune connexion n'est détectée.
Une version Android a donc été créée et publiée sur Google Play.
Il est également prévu de publier l'application sur l'Apple Store.

Des boutons de réseaux sociaux ont été ajoutés à l'application afin que l'utilisateur puisse partager l'évènement auquel il participe sur Facebook ou sur Twitter.
Il avait aussi été question d'ajouter un bouton *Check-in* relié à l'API du nouveau réseau social Foursquare. Cet ajout a été testé mais des erreurs étaient systématiquement renvoyées.
Ce bouton a donc été retiré mais il serait tout de même intéressant d'approfondir l'utilisation de l'API Foursquare car elle s'applique bien au concept du projet.

Étant donné la notoriété croissante de Meuse en Fête, un projet d'extension de l'évènement est lancé en 2013. Il s'agit de Sambre en Fête qui regroupe les évènements estivaux le long de la Sambre. Une application Sambre en Fête est donc prévue pour les années à venir. Celle-ci reprendra l'ensemble des fonctionnalités développées pour Meuse en Fête mais offrira un design et un contenu différent.

Concernant la publicité et le marketing, l'application a été brièvement présentée à la presse et à l'ensemble des acteurs du tourisme namurois lors de la conférence de presse de Meuse en Fête le 30 avril 2013.
Les médias sociaux et le bouche à oreille sont les principaux moteurs de propagation de l'application. La diffusion publicitaire se fait également via le site internet et les *newsletters* de la FTPN, ainsi que par des brochures touristiques. Cette publicité sera amplifiée et améliorée en 2014 en cas de succès lors de la saison 2013 de Meuse en Fête.

Une grande partie des touristes qui utiliseront l'application ne seront pas francophones.
Il faut donc aussi prévoir une version néerlandaise, anglaise et allemande afin de viser un public le plus large possible.
Pour le contenu, il suffira de récupérer les données de la langue correspondante dans le flux XML et d'adapter le module d'administration pour l'ajout de textes dans les différentes langues.

Conclusions

Meuse en Fête App est un projet qui m'a amené à gérer et à associer différents types de langages comme le HTML, le CSS, le Javascript et le PHP. J'ai pu découvrir la puissance et la simplicité du framework JQuery Mobile, qui pour moi, est une formidable alternative au développement long et coûteux d'une application mobile native.

J'ai également pu être impliqué dans la promotion d'un évènement touristique majeur en Wallonie. J'ai appris à décrire mes objectifs de manière claire et précise à mes commanditaires, à écouter leurs demandes et à tenir compte des contraintes imposées.
Je sais à présent comment se déroule la gestion d'un projet.

Enfin, je me rends compte de l'importance que le multimédia a dans le secteur touristique. Je pense que le tourisme est destiné à s'allier aux nouvelles technologies, afin d'attirer plus de touristes jeunes et étrangers. L'eTourisme est une source importante de débouchés et mérite de recevoir encore plus d'attention de la part des administrations régionales.
À présent, je sais vers quel secteur me diriger pour mon avenir professionnel.

Abstract

Objectifs

Prouver qu'il est possible de développer une application mobile grâce à des langages informatiques de base.
Montrer qu'une application web peut rivaliser avec une application native.

Approche

Utilisation du framework JQuery Mobile, qui permet la création d'un site web adaptable au mobile.
Utilisation du logiciel PhoneGap Build qui permet de transformer une application web en application hybride (mi-native, mi-web).

Résultats

Une application web possède moins de fonctionnalités qu'une application native mais le développement d'une web app est une méthode plus rapide, moins couteuse et plus accessible.
PhoneGap Build permet d'augmenter les fonctionnalités de la web app, ce qui offre une réelle alternative au développement d'applications au format natif.

Conclusion

JQuery Mobile et PhoneGap Build sont deux outils puissants permettant d'élargir l'accès au développement mobile. Cela donne une belle opportunité au secteur touristique de se lancer facilement et de manière peu couteuse dans l'utilisation des nouvelles technologies.

Bibliographie

Agenda... *Agenda 2012 : Fédération du Tourisme de la Province de Namur - La Meuse en Fête en Province de Namur*. Namur : Fédération du Tourisme de la Province de Namur - édition octogone, 2012, 51 p.

CHAFFER, Jonathan et SWEDBERG, Karl. *jQuery, Simplifiez et enrichissez vos développements JavaScript*, traduit par Hervé Soulard avec la contribution technique de Didier Mouronval. Paris : Pearson Education France, 2009. (Programmation, Développement web), 408 p.

ERNOTTE, Roland. *Documents hypertextes et multimédias. Partie 1 - Introduction à XML : 2012-2013*. Jemeppe (Seraing) : Haute École de la Province de Liège, catégorie sociale, 2012, 107 p.

ERNOTTE, Roland. *Documents hypertextes et multimédias. Partie 2 - Introduction à jQuery, Introduction à Ajax : 2012-2013*. Jemeppe (Seraing) : Haute École de la Province de Liège, catégorie sociale, 2012, 90 p.

Fleuve... *Un fleuve, une vallée : La Meuse, en Belgique & en France*. Namur : Fédération du Tourisme de la Province de Namur, 2012, 42 p.

GIFFORD, Matt. "Automatiser le développement d'applis mobiles". Dans : *Web Design*, Paris, n°48, avril 2013, p. 84-87.

GOBLET, Fabien, et al. *Développer avec les API Google Maps*. Paris : Dunod, 2010. (Études, développement & intégration), 221 p.

LEMORT, Albert. *Interfaces de programmation : 2012-2013*. Jemeppe (Seraing) : Haute École de la Province de Liège, catégorie sociale, 2012. Notes de cours.

SANCEY, Élisabeth. *Créez votre Web App*. 1ère Édition. Boulogne-Billancourt : 2012. (100% Visuel), 255 p.

Webographie

Actualités scientifiques et technologiques. www.techno-science.net (consulté en avril 2013).

Adobe Systems Inc. Adobe PhoneGap Build. https://build.phonegap.com (consulté en avril 2013).

CHARLOT, François. WebApp or not WebApp. Dans : Clever Garden, Clever Age, Clever Presence - 100% digital. http://fr.clever-age.com (consulté en avril 2013).

Fédération des gîtes de Wallonie. Allo chambre d'hôtes. www.allochambredhotes.be (consulté en janvier 2013).

Fédération du Tourisme de la Province de Namur. Portail touristique du Pays des vallées. www.paysdesvallees.be (consulté en avril 2013).

GANS, Nicolas. jQuery mobile / PhoneGap. Dans : Slideshare - Present yourself. http://fr.slideshare.net (consulté en mars 2013).

GARRETT, Jesse James. "Ajax: une nouvelle approche pour les applications Web". Dans : Scriptol. www.scriptol.fr/ajax/ajax-garrett.php (consulté en mars 2013).

Get latitude and longitude coordinates. http://getlatlon.yohman.com (consulté en janvier 2013).

Gliffy Inc. Gliffy. www.gliffy.com (consulté en avril 2013).

Google Developpers. Google Maps JavaScript API v3. https://developers.google.com/maps/documentation/javascript/reference (consulté en février 2013).

Google Maps. Dans : https://maps.google.com (consulté en avril 2013).

Google Maps API Icon Shadowmaker. Dans : www.cycloloco.com (consulté en novembre 2013).

GUGLIELMETTI, Rafael. PHP - Utilisation des sessions. Dans : Sources - Ressources

Haute Meuse dinantaise. www.dinant-tourisme.com. (consulté en novembre 2012).

Icon Finder. Dans : www.iconfinder.com (consulté en avril 2013).

Loc8 SPRL. iBeaken. www.ibeaken.com. (consulté en octobre 2012).

Observatoire du Tourisme wallon. "Sondage clientèle tourisme wallon: Bilan 2012". Dans : *CGT - Direction de la stratégie touristique*. http://strategie.tourismewallonie.be (consulté en avril 2013).

PHP - Les boucles. Dans: Vulgarisation informatique. www.vulgarisation-informatique.com (consulté en avril 2013).

PHP/MySQL. www.phpsources.org (consulté en mars 2013).

PNG Factory. Dans: www.pngfactory.net (consulté en avril 2013).

Rapport d'activités FTPN 2012. Dans : Portail touristique du Pays des vallées. www.paysdesvallees.be (consulté en avril 2013).

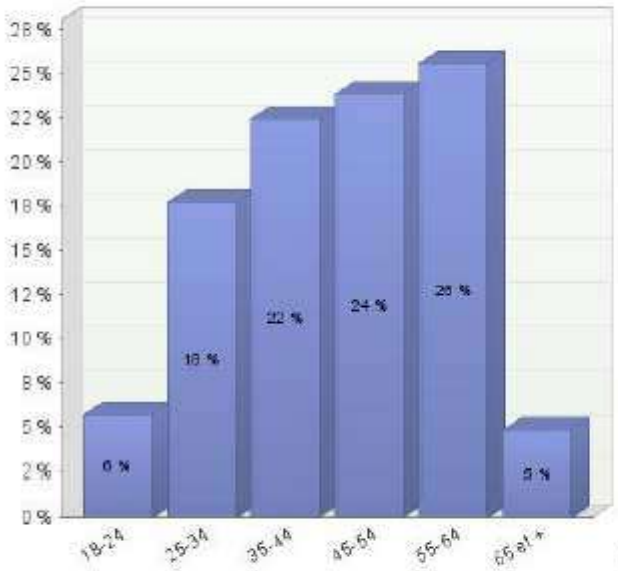
Wikipedia. Dans : https://fr.wikipedia.org (consulté en avril 2013).

Weather Underground Inc. Wunderground. www.wunderground.com (consulté en février 2013).

Annexes

Annexe 1 : Extrait du sondage clientèle effectué par le Commissariat général du Tourisme

- **Répartition par âge :**



n = 1807

Notons ici un biais lié à l'étude sous forme électronique. De fait, la part des 65 ans et plus est sous-représentée dans l'étude par rapport aux personnes de cette tranche d'âge qui viennent effectivement passer des vacances en Wallonie.

Annexe 2 : Extrait de la structure du flux XML récupéré par l'API

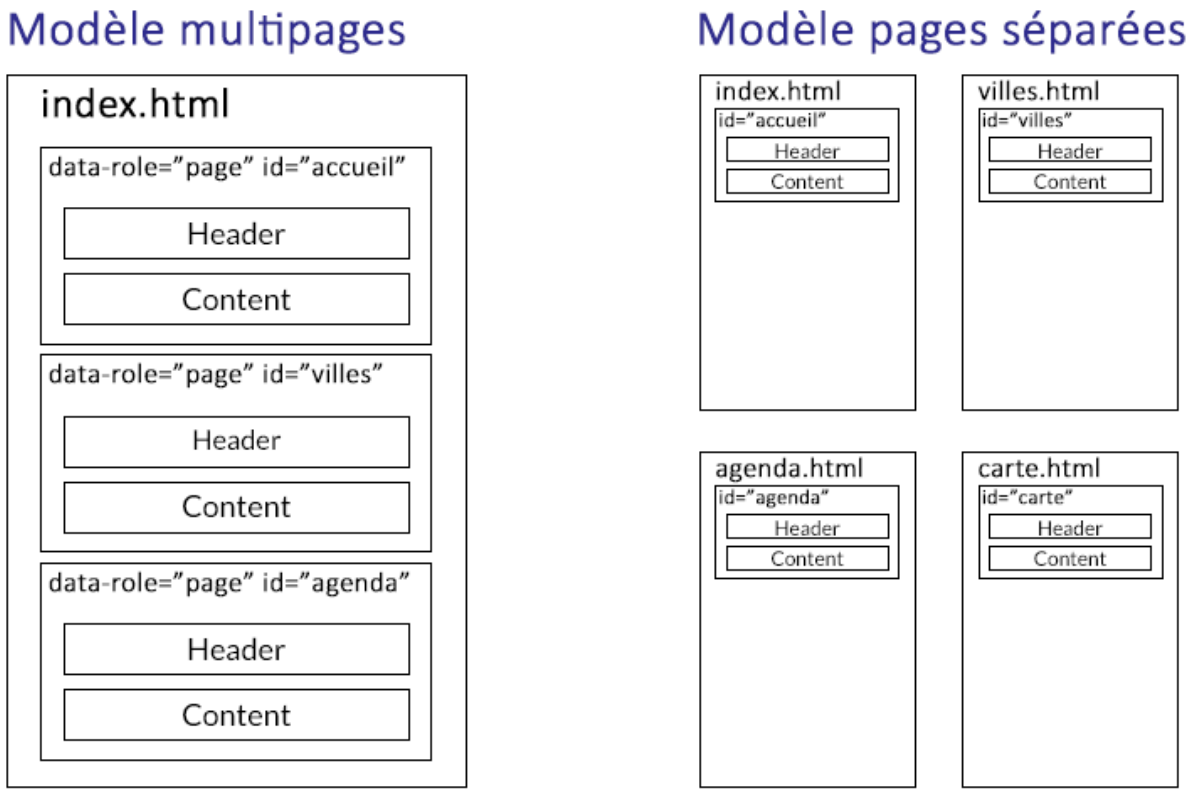
```
<evenements>
    <evenement id="721434102">
        <name_fr>Fête autour de l'eau</name_fr>
        <name_nl>Feest rond het water</name_nl>
        <name_en>Celebration around the Water</name_en>
        <name_de/>
        <type>Manifestation</type>
        <date_modif>27/03/2013</date_modif>
        <classement_type/>
        <classement_valeur/>
        <rue/>
        <numero/>
        <boite/>
        <localite>Profondeville</localite>
        <cp>5170</cp>
        <commune>Profondeville</commune>
        <longitude>4.8697</longitude>
        <latitude>50.37841</latitude>
        <contact>
            <prenom>Sophie</prenom>
            <nom>...</nom>
            <rue>...</rue>
            <numero>27</numero>
            <localite>Profondeville</localite>
            <cp>5170</cp>
            <commune>Profondeville</commune>
            <telephone>
                ...
            </telephone>
            <fax/>
            <gsm>...</gsm>
            <email>...</email>
        </contact>
        <communication_medias>
            <communication_media>
                <type>E-mail</type>
                <valeur>sophie.dardenne@base.be</valeur>
            </communication_media>
            <type>Site</type>
            <valeur>
            http://profondevilletiersmonde.wordpress.com/presentation
            </valeur>
            </communication_media>
        </communication_medias>
        <resume_fr>
            C'est faire un pas de plus vers une autre idée de la fête..
        </resume_fr>
```

```
        <resume_en>
            ...
        </resume_en>
        <resume_de/>
        <descriptif_de/>
        <descriptif_en/>
        <descriptif_fr/>
        <descriptif_nl/>
        <options/>
        <photos>
            <photo> http://www.paysdesvallees.be/servlet/  ...
            </photo>
        </photos>
        <videos/>
        <moyens_paiement/>
        <dates>
            <dates>
                <date_debut>02/08/2013</date_debut>
            </dates>
            <dates>
                <date_fin>04/08/2013</date_fin>
            </dates>
        <types>
            <type>Fêtes</type>
            <type>La Meuse en Fête</type>
        </types>
    </evenement>
    <evenement id="721434102">
        ...
    </evenement>
    ...
</evenements>
```

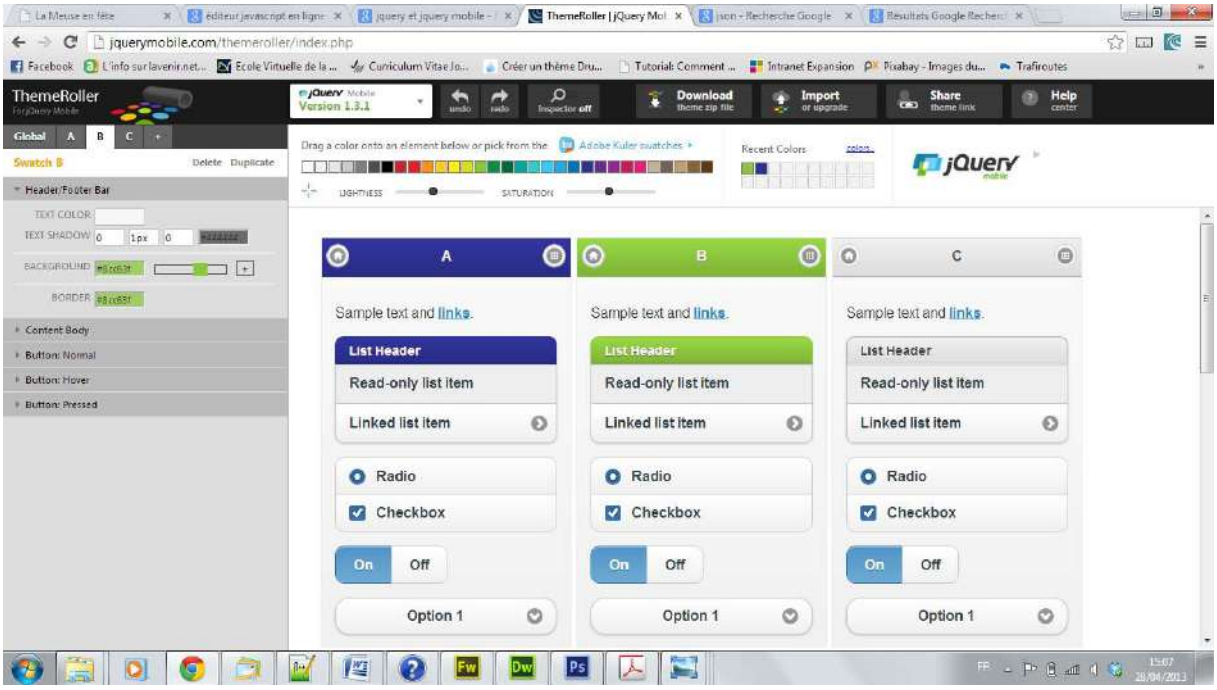
Annexe 3 : Exemple de code et d'affichage avec les data-role

```
<div data-role="page" data-theme="a" id="ville">
    <div id="header" data-role="header" data-position="fixed" data-tap-toggle="false">
        <a data-transition="none" href="#villes" data-role="button" data-icon="back">Accueil</a>
        <img src="pix/header_logo.png"/>
    </div>
    <div data-role="content" class="content">
        <h1 class="titreVille"></h1>
    </div>
</div>
```

Annexe 4 : Schéma montrant la différence entre le modèle multipages et les pages séparées



Annexe 5 : Theme Roller JQuery Mobile



Annexe 6 : Méthode $.getJSON du script *villes.js*

```
$(function(){
    $.getJSON("http://meuse-en-fete.mobi/api/api.php?method=getCities&jsoncallback=?",
    function(data){
            for(aCity in data){
                var city = data[aCity];
                var name = city.nom;
                var text = (city.texte).replace(/\n|\r\n|\r/g, "<br>\n");
                var address = (city.adresse).replace(/\n|\r\n|\r/g, "<br>\n");
                var tel = city.tel;
                var mail = city.mail;
                var web = city.web;

                var nameUrl = removeDiacritics(name.toLowerCase());
                var nameGet = getURLParameter("ville");

                if(nameGet==nameUrl){

                    $('.content .titreVille').html(name);

                    $('#ville .content').append(
                        '<a data-transition="none" id="infosBtnVille" ... \n\
                        <a data-transition="none" href="#carte" ... \n\
                        <a data-transition="none" href="#agenda" ... \n\
                        <a data-transition="none" href="#villeMeteo" ... \n\
                    ').trigger('create');

                    $('#villeInfos .content').append(' ... ');
                    break;
                }
            }

    });
});
```

Annexe 7 : Condition de filtrage des évènements "Meuse en Fête" utilisée dans *agenda.js*

```
$(function(){
    $.getJSON("http://meuse-en-fete.mobi/api/api.php?method=getEvents&jsoncallback=?",
    function(data){

        var nbEvts=0;
        var active_start=0;

        $.each(data, function(i, _event){

            // tableau des types d'événement
            var types = _event.types.type;

            for(var n in types){

                var type = types[n];

                // condition identifiant les events de la Meuse en fête
                if(type == "La Meuse en Fête"){
                    var _event = data[i];
                    var name = _event.name_fr;

                    ...

                }

            }

        });

    });

});
```

Annexe 8 : Utilisation de variables globales et de variables locales dans *carte.js*

```
// variables globales utilisables par toutes les fonctions
var map;
var markerFtpn;
var markerEvent; // markerEvent est utilisé dans addEvents()
...

function initialize(){
        // récupération des coordonnées de la ville à afficher sur la carte
        ...

        // Ajout de la position de l'utilisateur
        ...

        // Ajout des marqueurs et infobulles des events
        addEvents()

        detectBrowser();
}

function addEvents(){
    // variables locales propres à la fonction
    var _event;
    var lat;
    var lon;
    var name;
    ...
    $(function(){
    $.getJSON("http://meuse-en-fete.mobi/api/api.php?method=getEvents&jsoncallback=?",
        function(data){
            ...
            markerEvent = new google.maps.Marker({ ... });
            ...
        });
    });
    ...
}
```

Annexe 9 : Code HTML du panneau coulissant dans la division-page id="recherche"

```
<div data-role="page" data-theme="a" id="recherche">
    <!-- panel -->
    <div data-role="panel"  id="mypanel" data-position="left" ... >
        <h2>Recherche</h2>
        <h4><span style="color: #fc0800;">*</span>Dans un rayon de...</h4>
        <input type="range" name="rayon" id="rangePanel" value="10"/></p>
        <hr/>
        <h4><span style="color: #fc0800;">*</span>Près de...</h4>
        <select name="citiesPanel_list" id="citiesPanel_list">
            <option value="geolocation">Ma position</option>
        </select>
        <hr/>
        <h4>À la date du...</h4>
        <input type="date" id="datePanel" name="date" value=""/>
        <hr/>
        <p style="color: #fc0800;">* champs obligatoires</p>
        <p><button data-rel="close" id="searchPanel_btn" ... >Chercher</button><br/>
        <button data-theme="a" data-rel="close" ... >Fermer</button></p>
    </div>
    <!-- /panel -->

    <div id="header" data-role="header" data-position="fixed" ... >
        <a href="#accueil" data-role="button" data-icon="back" ... >Accueil</a>
        <img src="pix/header_logo.png"/>
        <a href="#mypanel" data-role="button" data-icon="search" ... >Recherche</a>
    </div>

    <div data-role="content" id="content_recherche">
        <h1>À proximité</h1>
        <div id="search_results" data-role="collapsible-set" data-iconpos="right">
            <p><a href="#mypanel" data-role="button" ... >Recherche</a></p>
            <p style="color: #fc0800; font-size: 13px;">Important: ... </p>
        </div>
    </div>

</div>
```

Glossaire

Accordéon : plugin JQuery permettant d'afficher des rubriques rétractables à la façon d'un accordéon.

AJAX : type de conception de pages web. Permet l'actualisation de certaines données d'une page sans le rechargement total de la page.

Android : système d'exploitation open-source pour smartphone et tablettes tactiles.

API : bibliothèque logicielle regroupant des méthodes et des procédures spécifiques à une utilisation. Cela permet une programmation plus simple.

Argument URL : variable écrite dans l'URL d'une page pouvant être récupérée par programmation.

Balise HTML : ou élément HTML, définit la structure d'un document HTML selon certaines normes. Elle contient des attributs et un contenu. Grâce aux balises, on peut spécifier qu'une telle partie de la page est un en-tête, un paragraphe ou encore un lien.

Boucle : en informatique, c'est une structure de contrôle permettant de répéter un certain nombre de fois les mêmes instructions. Il en existe plusieurs types.

Customiser : hérité de l'anglais *customize* et dérivé du mot français *customer*. Signifie créer un produit sur mesure à partir d'un produit de base.

CSS : feuilles de styles en français. Définissent la présentation graphique d'une page web via des règles appliquées au contenu HTML d'un document.

eTourisme : ou tourisme électronique, désigne les activités touristiques sur internet et autour des nouvelles technologies.

Footer : pied de page.

Framework : espace de travail fractionné en modules. C'est un groupe de bibliothèques et de composants qui permettent le développement rapide d'applications.

Header : en-tête de page.

HTML : langage servant à décrire et à formater la structure d'une page web.

Login : chaîne de caractère permettant d'identifier un utilisateur.

iOS : système d'exploitation mobile développé par Apple pour iPhone, iPad et iPod Touch.

Javascript : langage de script orienté objet, souvent utilisé dans les documents HTML. Il permet une interaction avec l'utilisateur par rapport aux opérations qu'il effectue (Exemple: survol d'une image avec la souris).

JQuery : bibliothèque Javascript libre permettant de simplifier l'utilisation commune de Javascript, HTML et CSS.

Latence : intervalle de temps entre la fin de l'émission d'un évènement et le début de la réaction à celui-ci.

Map : terme anglais signifiant *carte*. Souvent utilisé dans le domaine du jeu vidéo pour définir l'espace de jeu virtuel.

Méthode : fonction type appartenant à l'interface d'un objet informatique. Elle définit le comportement d'un objet.

Multiplateforme : logiciel conçu pour fonctionner sur différents types de plateformes (systèmes d'exploitation).

Nœud : en XML, il compose la structure du document XML. Un nœud peut être composé d'autres nœuds. Il est formé d'une balise ouvrante et fermante. Par exemple, dans un squelette humain un nœud serait un os et il serait défini par des attributs indiquant sa taille et sa position.

Objet : entité informatique représentant une partie du monde réel et regroupant un certain nombre de variables et de fonctions.

Paramètre : donnée gérée et identifiée par une partie du code.

PhoneGap : framework open-source visant le développement sur appareil mobile.

Plugin : aussi appelé *module d'extension* en français. C'est un bout de code pouvant être ajouté à un code existant afin de l'améliorer.

Police : liste de caractères typographiques faisant partie de la même famille. Elle possède plusieurs tailles et plusieurs styles.

QR code : code-barres en deux dimensions composé de petits carrés, lui-même ayant une forme carrée. Chaque point correspond à une série d'informations. Il est destiné à être lu par les smartphones comportant un appareil photo ainsi qu'un logiciel permettant le décodage des QR codes. L'information reçue par le smartphone peut être une URL, un numéro de téléphone, etc.

Racine : c'est le dossier principal d'un projet. Il contient tous les autres dossiers et fichiers du projet.

Smartphone : *téléphone intelligent* en français. Téléphone évolué possédant des fonctions similaires aux ordinateurs (lire des vidéos, de la musique, recevoir des mails, ...).

Redirection : action, automatique et programmable, effectuée pour un changement de page web.

Variable : associe un nom à une valeur (ou un objet).

Widget : assemblage de Javascript, de HTML et de CSS. Ce sont des applications utilisées occasionnellement.

XML : c'est un métalangage (qui englobe d'autres langages). Il s'agit d'un cadre définissant les langages d'une même famille pouvant interagir entre eux.

ZIP : format de fichier permettant la compression de données sans perte d'informations. Le nom *zip* provient de l'anglais qui signifie *fermeture éclair*. Le choix de ce nom est dû à la vitesse très rapide de la compression dans ce format.


Meuse en fête - Sambre en fête
PLUS DE 100 MANIFESTATIONS DU 09/05 AU 01/09/2013
www.paysdesvallees.be
La Meuse en fête
La Sambre en fête