Conception et développement d'une version du site adaptée à l'utilisation sur smartphone ou tablette

Version 0.1

Daniel Filipe Nunes Silva

14 January 2015

Table des matières

CHAPITRE 1

Introduction

Notre monde actuel devient chaque année plus proche de la technologie et se veut devenir petit à petit un monde connecté. Ceci est sans doute une des raisons pour lesquelles ce séminaire de développement d'une platforme de formation en ligne a été lancé. En effet, grâce à l'informatique, nous pouvons élaborer des techniques d'enseignement qui n'étaient pas disponibles il y a quelques années, ce qui sous-entend que la pédagogie de l'époque n'était pas la même qu'aujourd'hui et que celle-ci évolue au fil du temps. De ce fait, il nous faut constamment se mettre à jour si l'on veut explorer d'autres moyens d'étude et d'enseignement. Je vais, dans ce travail, programmer une application web en version mobile qui permettra aux élèves de communiquer avec leurs professeurs. Non pas par l'échange de mots mais par le téléversement d'images. Le but premier de mon travail est de fournir au professeur un outil grâce auquel il pourra par exemple demander à ses élèves de faire un exercice et le photographier avec leurs smartphones avant de l'envoyer sur un serveur auquel celui-là aura accès et pourra donc voir les différentes résolutions du travail de ses élèves. A travers cette technique, le gain de temps en classe pour lancer une correction est donc flagrant car le professeur connait à l'avance le résultat de ses élèves et peut adapter son cours en mesure de ce qu'il a observé auparavant. Je vais, pour arriver à ce dessein, d'abord vous présenter tout ce qui touche au côté mobile, ensuite vous présenter un petit guide de développement d'une application similaire à la mienne et finirai par présenter les caractéristiques de mon application.

CHAPITRE 2

Présentation de jQuery Mobile

2.1 Pourquoi avoir choisi une bibliothèque de ce genre ?

JQuery Mobile est une bibliothèque qui facilite grandement le développement web. Elle a été mise sur pieds par la même maison que la si fameuse bibliothèque Javascript également nommée jQuery. Dans ma partie du projet, il est avant tout question de développement web mobile. Lorsque l'on parle d'un projet mobile, trois grandes approches sont envisageables. On peut aussi bien partir sur le principe de développer une application native, adaptée à un système d'exploitation spécifique. Cette possibilité a l'avantage d'être très optimisée et performante mais nous contraint à développer une application pour chaque support que nous viendrons à utiliser. On peut aussi développer une application dite hybride, à mi-chemin entre le natif et le site web version mobile. La plupart du temps, celle-ci est codée en Javascript, html et css qui sera ensuite compilée pour offrir un rendu à l'allure native mais plutôt basée sur un code qui se rapproche du développement web. Cette alternative a l'avantage d'être multi-platforme mais un peu moins optimisées que les applications natives. Et finalement, je présente la voie que je vais suivre, celle du développement web en version mobile grâce à l'utilisation de cette bibliothèque jQuery Mobile. J'ai adopté ce choix en fonction de mes connaissances en programmation et des possiblités qu'il offre. Du fait que les pages internet codées à l'aide de jQuery Mobile s'affiche sur un navigateur, il devient évident que tous les supports dôtés d'un accès internet puisse profiter de cette interface, et cela quelque soit le système d'exploitation utilisé.

2.2 Existe-t-il des concurents à jQuery Mobile ?

Comme dans la plus grande majorité des domaines, la concurence est de la partie et cela même quand il s'agit de programmation. On peut par exemple citer : Kendo UI, ChocolateChip-UI ou encore bootstrap d'une certaine façon, nous en reparlerons plus tard dans ce travail. Parmi ces concurents, on trouve certains qui disposent de fonctionnalités inédites comme la géolocalisation, des thèmes s'inspirant des surcouches utilisateurs connues et encore des petites différences qui peuvent influencer notre choix de bibliothèque mais qui en fin de compte toutes offre un résultat similaire.

2.3 Le choix de jQuery Mobile plutôt que d'un autre concurent

Après avoir observé différentes bibliothèques et comparé leurs possibilités, ceci ne me paraissait pas un choix très important et j'ai préféré resté sur l'idée de départ qui est jQuery Mobile. J'ai facilement associé celle-ci à la bibliothèque Javascript déjà existante du même nom dont la renommée n'est pas à remettre en question. JQuery Mobile et aussi indirectement jQuery seront au centre de mon travail de recherche et pratique.

CHAPITRE 3

jQuery Mobile et Bootstrap

3.1 Fonctionnement général de jQuery Mobile

La majeure partie de jQuery Mobile se joue dans le balisage de son code html. Dans la mesure où l'on définit si le contenu d'un balisage :

Du contenu :

```
<div>contenu</div>
```

Un lien :

```
<a href="#">lien</a>
```

et pourquoi pas un bouton :

```
<button>bouton</button>
```

deviendra une page, une boîte de dialogue ou encore une liste entre autres. Pour un petit test, faisons l'exemple avec ces trois morceaux de code. Ainsi en ajoutant des classes telles que ui-content pour le contenu :

```
<div class="ui-content">contenu</div>
```

Les scripts jQuery Mobile viendront s'appliquer là-dessus et considéreront ceci comme le contenu d'une page et appliqueront le code css nécessaire. Pareil pour le lien et le bouton qui suivent. Nous pouvons y ajouter la classe 'ui-btn' qui dira aux scripts jQuery Mobile d'appliquer de code css nécessaire pour avoir l'allure d'un bouton.

```
<a href="#" class="ui-btn">lien</a>
<button class="ui-btn">bouton</button>
```

Pour l'utilisation de jQuery Mobile, il est donc nécessaire de travailler avec ce balisage qui permettra à la bibliotheque d'interpréter le code et d'y appliquer les attributs et la mise en page nécessaire avec une allure très agréable à l'utilisation tactile et très bien adaptée aux écrans de taille plutôt réduite que l'on peut retrouver sur un smartphone standard voire sur une tablette de petite taille.

3.2 Fonctionnement général de Bootstrap

Bootstrap ne sera pas au coeur de ce travail mais il est intéressant de comparer ces bibliothèques qui peuvent paraître proches mais qui finalement offrent un rendu relativement opposé. Le principe de bootstrap est basé non pas sur le balisage comme jQuery Mobile mais sur une sorte de grille. Cette grille sera composée d'un certain nombre de colonnes et de lignes où l'utilisateurs pourra positionner les éléments qu'il désire afficher sur sa page. Et c'est lors du

changement de support que l'on peut observer toute la magie de bootstrap car cette grille s'adapte elle-même à l'écran. Ainsi le contenu se dit "responsive", soit "qui s'adapte". Par exemple, un barre de navigation initialement placée sur le côté gauche verticalement si l'on consulte le site sur un écran large pourrait se retrouver horizontalement sur un écrant plus étroit au dessus du contenu principal.

3.3 Similitudes et différences

Ces deux bibliothèques se ressemblent dans la mesure où elles me seraient toutes les deux utiles afin de créer une interface mobile pour ma future application. Elles proposent un affichage qui est facilement utilisable dans des conditions que l'on ne retrouve pas sur un ordinateur de bureau et auxquelles on ne pense pas forcément au cours du développement. Ceci nous permet donc une interface utilisateur adaptée aux besoins d'une personne utilisant cette application mobile. Les deux présentent également un très bon système pour modifier les thèmes et ainsi apporter un côté ludique ou encore plus agréable. Par contre Bootstrap se concentre vraiment sur une allure du site qui s'adapte aux différents formats d'écrans tandis que jQuery Mobile est plutôt dans l'optique de proposer un rendu qui lui est entièrement consacré au mobile en se souciant peu de l'affichage sur une plus grand écran. Malgré cela, on peut dire que l'affichage mobile sur un grand écran n'est pas désagréable notammant si l'on dispose d'un écran tactile mais ce n'est pas le meilleur que l'on puisse avoir.

CHAPITRE 4

Particularités du développement mobile

Du fait d'élaborer une interface mobile, il est indispensable de se demander si ce type de travail présente des spécificités auxquelles on ne fait face en développant une version bureau.

Effectivement, on se rend vite compte par l'utilisation d'un smartphone que la plupart des interfaces utilisateurs subissent une refonte. On observe souvent des boutons plus gros, mieux adaptés au toucher sur les écrans tactiles. De plus, quand on utilise un smartphone, on se retrouve souvent sur un réseau mobile à débit ou volune de données limitées. De ce fait, il serait favorable à l'utilisateur de l'application et au développeur de fournir des pages légères. Dans mon application, je désire notamment intégrer un moyen de compresser les images. L'utilisateur pourra ainsi, lorsqu'il charge un index d'images, avoir un aperçu rapide à toutes celles qui ont été téléversées et par après accéder à un détail si tel en est le souhait pour bénéficier d'une meilleure qualité. Finalement, quand on navigue sur un appareil mobile, on a l'occasion d'éxécuter des gestes ou 'évènements' qui ne se font pas sur un écran avec lequel on interagit avec un clavier ou une souris. Par exemple, le fait de glisser de gauche à droite pour ouvrir une extension de la page ou encore l'appui prolongé sur un élément qui peut être comparé au double-clic sur une interface standard.

En fin de compte, il faut savoir correctement adapter l'interface que l'on élabore à une utilisation sur mobile. Ici jQuery Mobile fait encore des merveilles. Dans la mesure où il est capable de gérer différents évènements et la plupart de ces éléments afin de rendre une version adaptée.

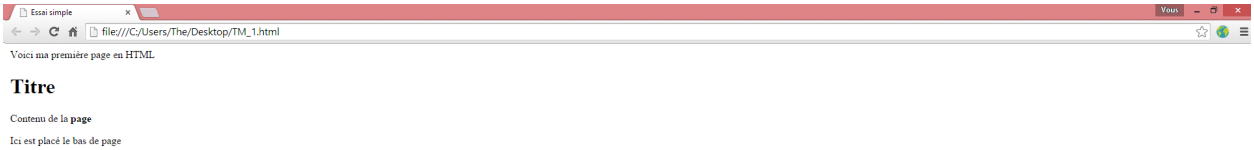
CHAPITRE 5

Technologies utilisées pour le développement

5.1 HTML 5

Abréviation de 'HyperText Markup Language 5', ce langage permet d'introduire le contenu qui sera affiché dans une page. Celui-ci sera sectionné en différentes catégories séparées par des balises. Exemple d'une page simple en html :

```
<!DOCTYPE html>
<html>
    <!-- head est destiné à placer les scripts a charger CSS, Javascript,...,
    spécifier l'encodage ou encore le titre qui sera sur l'onglet de la page -->
    <head>
            <meta charset="utf-8" />
            <link rel="stylesheet" href="style.css"/>
            <script type="text/javascript" src="javascript.js"></script>
            <title>Essai simple</title>
    </head>
    <!-- body contient tout le content chargé pour la page -->
    <body>
        <!-- header correspond à l'entête -->
            <header>Voici ma première page en HTML</header>
            <!-- h1 correspond à un niveau de titre du plus au moins important:
            h1, h2, h3, ..., h6 -->
            <h1>Titre</h1>
            <!-- p correspond a un paragraphe standard et stron à mettre le contenu
            des balises en gras-->
            <p>Contenu de la <strong>page</strong></p>
            <!-- footer correspond au pied de page -->
            <footer>Ici est placé le bas de page</footer>
    </body>
</html>
```



5.2 CSS 3

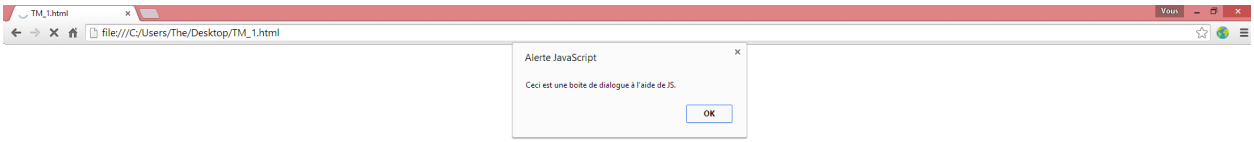
Abréviation de 'Cascading Style Sheets 3', ce langage permet de structurer, personnaliser, modifier le code html. Et cela aussi bien en y ajoutant des couleur, des grilles, des bordures,... Exemple sur la page HTML (dans le fichier style.css, chargé dans la page html d'avant) :

```
/* on applique la police Verdana a tous le corps de texte*/
body
{
    font-family: Verdana;
}

/* On aligne le titre h1 au centre */
h1
{
    text-align: center;
}

/* pour l'entête et le pied de page, on met leur arrière plan en orange */
header, footer
{
    background-color: orange
}

/* on met le texte en rouge dans les paragraphes */
p
{
    color: red
}
```

5.3 Javascript

Le Javascript est un langage qui lui permet d'ajouter de l'interactivité dans ses pages internet car comme vous avez pu le remarquer on se retrouve avec des pages dite 'statiques' qui ne font qu'afficher du contenu. Nous allons ici ajouter une boite de dialogue avec javascript (dans le fichier javascript.js, chargé dans la page html d'avant) :

```
/* la fonction alert permet une boite de dialogue */
alert("Ceci est une boite de dialogue à l'aide de JS.")
```



5.4 jQuery

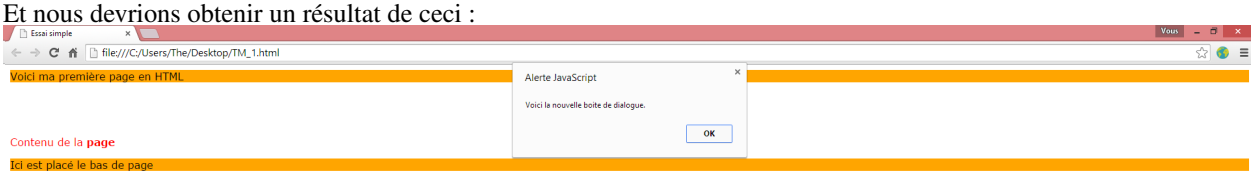
JQuery est une bibliothèque javascript ce qui veut dire qu'elle est basée sur du code javascript avec des fonctions déjà préparée et pour pouvoir les utiliser, nous devons charger un script supplémentaire comme l'on a fait avec le CSS et le JS :

```
<script src="http://code.jquery.com/jquery-1.11.1.min.js"></script>
```

En plus de cela jQuery utilise également des évènements qui, par exemple, lance des fonctions lorsqu'on clique sur une bouton. Sur notre page vous avez pu remarquer que la boite de dialogue s'ouvre avant le chargement de la page. Nous allons donc remplacer le code javascript d'avant par un autre qui affichera cette boite de dialogue seulement après le chargement :

```
/* on se réfère au document ouvert, donc la page qui quand elle est prête
méthode 'ready' éxécute la fonction alert que l'on a vu précédemment*/
$(document).ready(function(){
    alert("Voici la nouvelle boite de dialogue.");
});
```

Et nous devrions obtenir un résultat de ceci :



5.5 Django

Django est également une bibliothèque mais basée sur le langage python cette fois. C'est une bibliothèque web qui permet de créer des sites internet de façon plutôt intuitive une fois que l'on a appréhendé le fonctionnement de celle-ci. Tout comme python, le langage django est relativement facile à comprendre et permet de proposer un site web efficace avec un effort qui est moindre.

5.6 python

Python est donc ce langage qui alimente django et qui lui permet la plupart des actions. Celui-ci est plutôt de haut niveau ce qui signifie qu'il se rapproche plutôt de la façon de penser de l'homme plutôt que de la machine. Un petit exemple qui illustre la syntaxe et l'allure du langage :

```
def somme(nbrUn, nbrDeux):
    """somme(float nbrUn, float nbrDeux) ---> float fonction qui retourne
    la somme de nbrUn et nbrDeux"""
    somme = nbrUn + nbrDeux
    return somme
```

5.7 jQuery Mobile

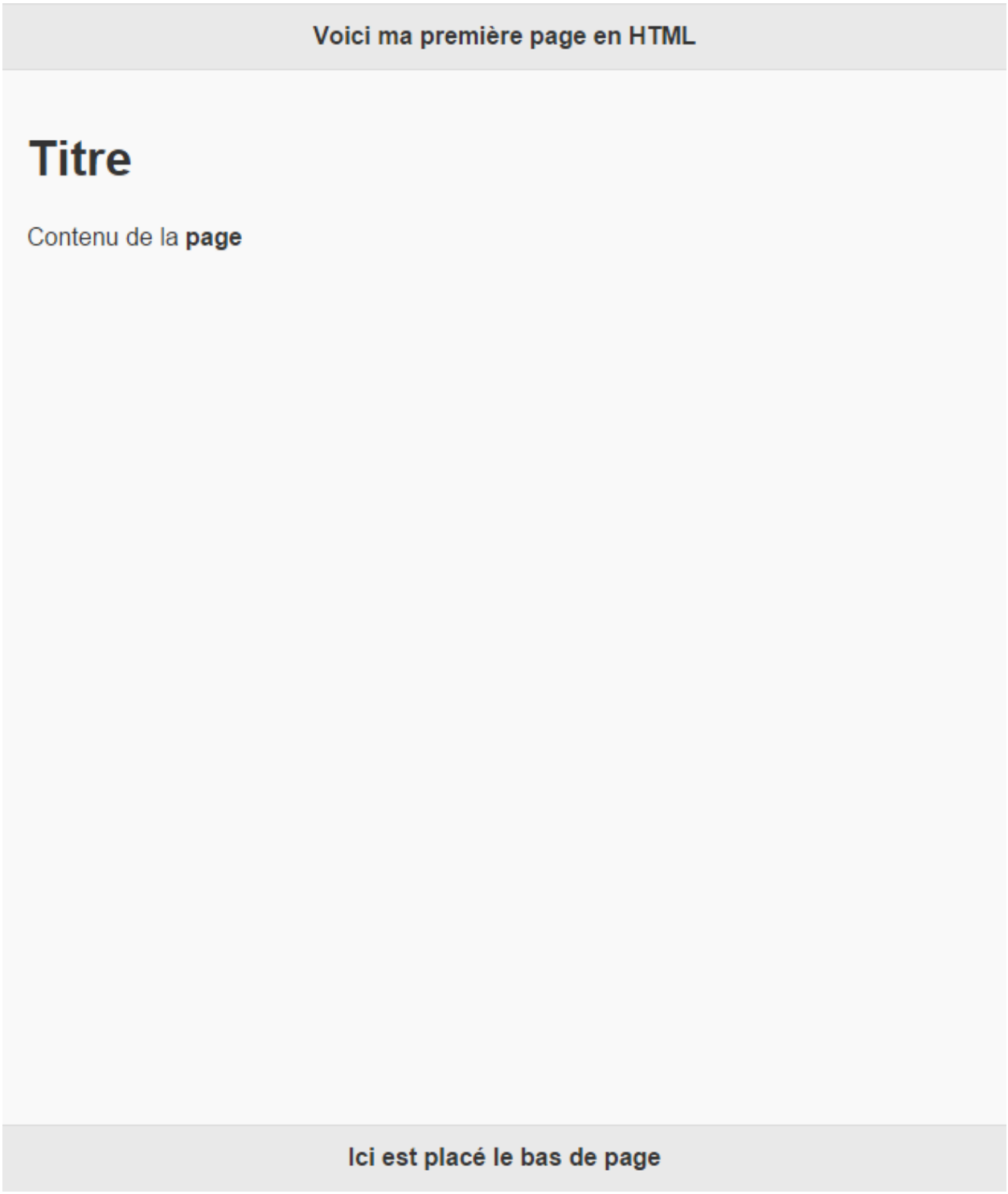
jquery mobile est cette fameuse bibliothèque que j'aurai l'occasion de présenter d'une façon plus approfondie dans ce travail. Tout d'abord, il faut la charger tous les scripts de la même façon que jQuery :

```
<!-- le script CSS qui va charger les éléments de style jQuery Mobile-->

<link rel="stylesheet" href="http://code.jquery.com/mobile/1.4.5/jquery.
mobile-1.4.5.min.css" />

<!-- le script de jQuery qui est aussi nécessaire -->
<script src="http://code.jquery.com/jquery-1.11.1.min.js"></script>

<!-- et enfin celui de jQuery Mobile -->
<script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5
.min.js"></script>
```

A partir de là, il ne nous reste plus qu'à retravailler le code html pour l'adapter au balisage jQuery Mobile. Aussi bien au niveau du nom de la balise, attribuer un id spécifique ou le plus souvent une classe. Voici une page jQuery Mobile de base avec un rendu type 'écran mobile' :

```
<!DOCTYPE html>
    <html>
        <head>
            <meta charset="utf-8" />
            <link rel="stylesheet" href="http://code.jquery.com/mobile/
            1.4.5/jquery.mobile-1.4.5.min.css" />
            <script src="http://code.jquery.com/jquery-1.11.1.min.js">
            </script>
            <script src="http://code.jquery.com/mobile/1.4.5/jquery.
            mobile-1.4.5.min.js"></script>
                    <title>Essai simple</title>
            </head>

            <body>
                <!-- début de page -->
            <div data-role="page" id="page" >

            <!-- début entête -->
            <div data-role="header">
                <h1>Voici ma première page en HTML</h1>
            </div>
            <!-- fin entête -->
```

```
        <!-- début contenu -->
        <div role="main" class="ui-content">
            <h1>Titre</h1>
                <p>Contenu de la <strong>page</strong></p>
        </div>
        <!-- fin contenu -->

        <!-- début bas de page -->
        <div data-role="footer" data-position="fixed">
            <h4>Ici est placé le bas de page</h4>
        </div>
        <!-- fin bas de page -->

        </div>
        <!-- fin de page -->
        </body>
</html>
```



On voit déjà ici la puissance de jQuery Mobile pour la mise en page qui malgré son thème de base reste très agréable et s'adapte bien au petits écrans.

CHAPITRE 6

Guide du programmeur

6.1 Introduction

Dans ce guide du programmeur, je vais vous montrer comment coder une application web qui permet de charger des images sur un serveur. Ce tutoriel prend en compte que vous connaissez un minimum django, le HTML et le CSS.

6.2 Les modèles

Après avoir créé l'application django et réglé le nécessaire dans django, nous allons commencer par les modèles dans le fichier 'models.py' :

```
from django.db import models

class Picture(models.Model):
    #ImageField pour que l'image soit enregistrée comme une image
    image = models.ImageField(upload_to="uploadedImages")

    #tag: court énoncé sur l'image
    tag = models.CharField(max_length=20)

    #description: texte pour décrire l'image
    description = models.CharField(max_length=500)

    #date de l'upload
    date = models.DateField(auto_now_add=True)
```

Ensuite, on procède au fameux ' makemigrations <nomApp> ' et ' migrate '

6.3 Les formulaires

Les formulaires sont indispensables lorsque l'on veut charger une image ou du texte sur un serveur et que l'on désire contrôler si ce que l'utilisateur a saisi est bien le type de données qui est attendu nous créons alors ces formulaires dans le fichier 'forms.py' :

```
from django import forms

#classe pour le téléversement d'images
class ChargementForm(forms.Form):
```

```
    image = forms.ImageField()
    tag = forms.CharField(max_length=20)
    description = forms.CharField(max_length=500)

#classe pour le formulaire de modifications
class ModificationForm(forms.Form):
    tag = forms.CharField(max_length=20)
    description = forms.CharField(max_length=500)
```

On observe une forte similairité avec les classes de modèles. Ceci est dû au fait que les formulaire sont créés pour instancier des objets des classes des modèles.

6.4 Les vues

Ici nous allons créer quelques vues qui serviront à charger l'image sur le serveur, supprimer l'image, afficher toutes les images ou encore modifier les détails d'une image.

```
from django.shortcuts import render, redirect
from <nomApp>.forms import ChargementForm, ModificationForm
from <nomApp>.models import Picture

#vue pour charger une image sur le serveur, si la requête post a lieu et que
#tous les champs du formulaire sont correctement remplis, on sauvegarde cette
#instance. Le boléen sauvegarde sera utile pour affiche un message lorsque
#l'image aura été chargée.
def chargement(request):
    sauvegarde = False

    if request.method == "POST":
        form = ChargementForm(request.POST, request.FILES)
        if form.is_valid():
            image = Picture()
            image.image = form.cleaned_data["image"]
            image.tag = form.cleaned_data["tag"]
            image.description = form.cleaned_data["description"]
            image.save()
            sauvegarde = True
    else:
        form = ChargementForm()

    return render(request, 'chargement.html', locals())

#vue pour afficher toutes les images. On instancie tous les objets de la
#classe Picture et on retourne le template et les images
def index(request):
    images = Picture.objects.all()
    return render(request, 'index_images.html', {'images': images})

#vue pour afficher le détail d'une image. On instancie l'objet de la classe
#picture qui correspond a l'id désiré (récupéré dans l'url) et l'on retourne
#le template, l'instanciation d'image et de formulaire grâce à 'locals()'
def detail_image(request, imageId):
    image = Picture.objects.get(id=imageId)
    form = ModificationForm()
    return render(request, 'image_detail.html', locals())
```

```
#vue pour supprimer une image, active lors d'une requête post et qui supprime
#l'image elle-même ainsi que l'instance de la classe Picture avant de nous
#rediriger vers l'index
def supprimer_image(request, imageId):
    if request.method == "POST":
        image = Picture.objects.get(id=imageId)
        image.image.delete()
        image.delete()
        return redirect('<nomApp>:index')

#vue pour modifier le tag ou la description d'une image, il s'agit du même
#principe que les vues d'avant.
def modifier_image(request, imageId):
    image = Picture.objects.get(id=imageId)
    if request.method == "POST":
        form = ModificationForm(request.POST, request.FILES)
        if form.is_valid():
            image.tag = form.cleaned_data["tag"]
            image.description = form.cleaned_data["description"]
            image.save()
            return redirect('<nomApp>:index')
        else:
            return redirect('<nomApp>:index')
```

6.5 Fichier settings.py

Pour que les images soit enregistrées correctement il est nécessaire de définir ces variables là :

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
MEDIA_URL = os.path.join(BASE_DIR, 'media/')
```

6.6 Fichier urls.py

Nous devons d'abord ajouter le chemin vers les urls de l'application que nous avons créée. Ceci doit dans le fichier urls.py dans les fichiers de l'environnement de travail, ce ne sont pas les urls.py spécifique à l'application que nous sommes en train de créer) :

```
from django.conf.urls import patterns, include, url
from django.contrib import admin
from django.conf.urls.static import static
from django.conf import settings

urlpatterns = patterns('',
    url(r'^admin/', include(admin.site.urls)),
    #cette ligne-ci
    url(r'^<nomApp>/', include('<nomApp>.urls', namespace="<nomApp>")),
)

#Et l'ajout de cette ligne également pour l'accès aux images chargées
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Nous devons aussi créer nos propres url pour l'application que nous sommes en train de creer dans le fichiers urls.py de l'application :

```
from django.conf.urls import patterns, url
from django.conf import settings
from django.conf.urls.static import static

from <nomApp> import views

urlpatterns = patterns('',

    #pour la vue d'index
    url(r'^index/', views.index, name='index'),

    #pour la vue détaillée
    url(r'detail/(?P<imageId>\w+)', views.detail_image, name='detail_image'),

    #pour la vue de chargement
    url(r'^chargement/', views.chargement, name='chargement'),

    #suppression d'image
    url(r'suppression/(?P<imageId>\w+)', views.supprimer_image, name='supprimer_image'),

    #modification d'image
    url(r'modification/(?P<imageId>\w+)', views.modifier_image, name='modifier_image'),
)
```

6.7 Les templates

Nous avons déjà préparé toute la partie cachée qui va s'éxécuter lorsque l'on utilisera notr application mais nous n'avons encore fait aucune page qui nous permettra d'utiliser toutes ces fonctions, nous allons donc nous attaquer aux templates. Les templates présentés sont composés d'éléments de jQuery Mobile, qui seront précisément présentés lors de la démostration des fonctionnalités de mon application utilisée dans sa version finale.

Nous utiliserons la balise blocks pour notre premier template :base.html' :

Ne pas oublier dans un premier temps d'ajouter la directions vers le dossier où se trouvent les templates :

```
TEMPLATE_DIRS = [os.path.join(BASE_DIR, 'templates')]
```

Ensuite placer le fichier 'base.hmtl' dans ce même dossier :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Mobile</title>
        <!-- ligne pour que le contenu s'adapte à l'appreil mobile -->
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <!-- chargement des script et du css nécessaire -->
        {% load staticfiles %}
        <!-- cdn jqm et jq -->
        <link rel="stylesheet" href="http://code.jquery.com/mobile/
        1.4.5/jquery.mobile-1.4.5.min.css" />
        <script src="http://code.jquery.com/jquery-1.11.1.min.js"></script>
        <script src="http://code.jquery.com/mobile/
        1.4.5/jquery.mobile-1.4.5.min.js"></script>
    </head>
    <body>
```

```
        <!-- début de page -->
        <div data-role="page">
        <!-- début entête -->
        <div data-role="header">
        <h1>MonApp</h1>
        </div>
        <!-- fin entête -->

        <!-- début contenu -->
        <div role="main" class="ui-content">
        {% block content %}<p>Content</p>{% endblock %}
        </div>
        <!-- fin contenu -->

        <!-- début bas de page -->
        <div data-role="footer">
        <h4>Mobile</h4>
        </div>
        <!-- fin bas de page -->

        </div>
        <!-- fin de page -->
    </body>
</html>
```

Nous utiliserons ce fichier 'base.html' comme squelette pour toutes les autres pages. Nous aurons uniquement à spécifier que les autres templates sont des extensions de celui-ci et rajouter le contenu entre les balises '{% block contenu %}...{% endblock %}'.

L'index :

```
{% extends "base.html" %}
{% block content %}
<!-- on affiche les images dans une liste -->
<ul data-role="listview" data-split-icon="gear" data-split-theme="a" data-inset="true">
    {% for image in images %}
        <li><a href="#popup{{ image.id }}" data-rel="popup" data-position-to="window" data-transitio
            <img src="{{ image.image.url }}">
            <p>{{ image.tag }}</p>
            <p>id: {{ image.id }}</p></a>
            <a href="{% url 'uploads:detail_uploaded' image.id  %}"></a>

            <!-- popup avec l'image personalisé -->
            <div data-role="popup" id="popup{{ image.id }}">
                <a href="#" data-rel="back" class="ui-btn ui-corner-all
                ui-shadow ui-btn-a ui-icon-delete ui-btn-icon-notext
                ui-btn-right">Fermer</a><img src="{{ image.image.url }}"
                class="ui-corner-all">
            </div>
        </li>
    {% endfor %}
</ul>
{% endblock %}
```

La page de chargement :

```
{% extends "base.html" %}
{% block contenu %}
<!-- quand l'image a été uploadée cette ligne s'afficher -->
```

```
{% if sauvegarde %}
    <p>Cette image a bient été uploadée.</p>
{% endif %}
<div>
    <!-- formulaire pour upload de l'image, data-ajax="false" permet -->
    <!-- d'éviter des chargements propres a jQM qui empêchent le bon -->
    <!-- fonctionnement des formulaires  -->
    <form method="post" enctype="multipart/form-data" action="." data-ajax="false">
        {% csrf_token %}
        <!-- on intègre le formulaire de chargement -->
        {{ form.as_p }}
        <button type="submit" class="ui-btn btn-bottom buttonLoad"
        data-textonly="false" data-textvisible="true" data-msgtext=""
        data-inline="true">Uploader !</button>
    </form>
</div>
{% endblock %}
```

Le détail d'une image :

```
{% extends "base.html" %}
{% block content %}
<!-- image -->
<img src="{{ image.image.url }}"/>
<!-- début liste déroulable -->
<div data-role="collapsibleset" data-theme="a" data-content-theme="a">
    <!-- début premier item -->
    <div data-role="collapsible">
        <h3>Informations</h3>
        <p>Tag: {{ image.tag }}</p>
        <p>Description: {{ image.description }}</p>
        <p>date: {{ image.date }}</p>
    </div>
    <!-- fin premier item -->
    <!-- début deuxième item -->
    <div data-role="collapsible">
        <h3>Modifications</h3>
        <form method="post" action="{% url 'uploads:modify' image.id %}"
        enctype="multipart/form-data" data-ajax="false">
            {% csrf_token %}
            <!-- ici je fais le formulaire manuellement pour résussir a-->
            <!-- mettre une valeur initiale -->
            <label for="tag">Tag: </label>
            <input id="tag" data-clear-btn="true" type="text" name="tag"
            maxlength="20" value="{{ image.tag }}">

            <label for="description">Description: </label>
            <input id="description" data-clear-btn="true" type="text"
            name="description" maxlength="500" value="{{ image.description}}">

            <button type="submit"/>Modifier</button>
        </form>
    </div>
    <!-- fin deuxième item -->
    <!-- début troisième item -->
    <div data-role="collapsible">
        <h3>Zone de danger</h3>
        <p><a href="#popupdelete" data-rel="popup" data-position-to="window"
```

```
        class="ui-btn ui-corner-all ui-shadow ui-btn-inline ui-icon-delete
        ui-btn-icon-left ui-btn-a" data-transition="slide">Supprimer</a></p>
        <!-- popup pour supprimer -->
        <div data-role="popup" id="popupdelete" data-theme="a"
        class="ui-corner-all ui-content">
            <h3>Attention !!!</h3>
            <p>Vous êtes sur le point de supprimer cette image.</p>
            <form method="post" action="{% url 'uploads:delete' image.id %}"
            data-ajax="false">
                {% csrf_token %}
                <button type="submit"/>Supprimer</button>
            </form>
        </div>

    </div>
    <!-- fin troisième item -->
</div>
{% endblock %}
```

Comme les fonctionnalités modifier et supprimer s'éxécute du côté serveur, nous n'avons pas besoin de produire un template pour chacune de celle-ci.

Grâce à ce petit tutoriel, vous avez pu créer un rudimentaire réseau social de photographies en exploitant le jQuery Mobile, ainsi que Django et évidemment tout ce qui est HTML et CSS.

CHAPITRE 7

Ressources utilisées pour mon TM

- https://docs.djangoproject.com/fr/1.7/
- http://openclassrooms.com/courses/apprenez-a-creer-votre-site-web-avec-html5-et-css3
- http://openclassrooms.com/courses/dynamisez-vos-sites-web-avec-javascript
- http://api.jquery.com/
- http://api.jquerymobile.com/