Institut National des Sciences
Appliquées de Rennes

Scénarii V&M

Documentation technique

# Développement mobile
# Utilisation de Cordova

*Réalisé par :*
Hassan EL OMARI ALAOUI

# Table des matières

# 1 Préparer le terrain

Avant d'installer Cordova, il faut tout d'abord commencer par installer Node.js et les kit de développement logiciels (SDK) pour chaque plateforme que vous envisagez d'utiliser.

## 1.1 Node.js

L'installation de Node.js est assez facile. Pour Windows et Mac OS, vous devez lancer l'exécutable qui se trouve sur le lien suivant

- `http://nodejs.org/download/`

Après l'installation de Node.js, vous devriez être capable de faire appel à *node* et *npm* depuis la ligne de commande. Dans le case contraire, Node.js n'a pas été bien installé.
Concernant Linux, vous pouvez soit télécharger le fichier compressé sur le même site ou bien installer Node.js depuis un terminal

```
1 ~$ sudo apt-get install python-software-properties
2 ~$ sudo apt-get update
3 ~$ sudo apt-get install nodejs
```

## 1.2 SDK

### 1.2.1 iOS

**XCode**
Le développement d'applications mobiles iOS grâce à Cordova requiert seulement l'installation de XCode qui est un environnement de développement pour Mac OS X et pour iOS. L'installation de ce logiciel se fait directement sur AppStore.

### 1.2.2 Android

**Java JDK**

**Linux Ubuntu**

- Installation de Java8
  Ouvrez un terminal et lancer les commandes suivantes

```
1 ~$ sudo add-apt-repository ppa:webupd8team/java
2 ~$ sudo apt-get update
3 ~$ sudo apt-get install oracle-java8-installer
```

- Ajout au path
  Créez un fichier *bash_profile.sh* et insérez-y ces lignes

```
#! /bin/bash
clear

JAVA_HOME=/usr/lib/jvm/java-8-oracle
```

```
PATH=$PATH:$HOME/bin:$JAVA_HOME/bin
JRE_HOME=/usr/lib/jvm/java-8-oracle/jre
PATH=$PATH:$HOME/bin:$JRE_HOME/bin
export JAVA_HOME
export JRE_HOME
export PATH
```

puis exécuter ce fichier grâce à la commande

```
1 ~$ ./bash\_profile.sh
```

**Windows**

- Installation de Java8
  Vous pouvez télécharger le JDK en utilisant ce lien
  - `http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html`
- Ajout au path
  Après l'installation de Java JDK, il faut l'ajouter au path du système. Pour ce faire, rendez vous dans *Propriétés système*, *Paramètres système avancés* puis, *Variables d'environnement* et modifier le chemin *PATH* en ajoutant l'emplacement du dossier *bin* qui se trouve dans le dossier de Java JDK que vous avez installé.

## SDK Android

**Linux Ubuntu**

- Téléchargement du SDK
  Vous pouvez télécharger le sdk android depuis ce lien
  - `http://developer.android.com/sdk/index.html`
- Installation
  1. Commencer par décompresser le fichier zip téléchargé, puis installer les composants en lançant l'exécutable *android* qui se trouve dans le dossier *android-sdk/sdk/tools*

     ```
     1 ~$ ./android
     ```

     et installer les Tools Android API 17
  2. Ajout au path
     Deux choix se portent à vous, soit
     - Vous modifiez le fichier *bash_ profile.sh* que vous avez ajouter

       ```
       #! /bin/bash
       clear

       JAVA_HOME=/usr/lib/jvm/java-8-oracle
       PATH=$PATH:$HOME/bin:$JAVA_HOME/bin
       JRE_HOME=/usr/lib/jvm/java-8-oracle/jre
       ```

```
PATH=$PATH:$HOME/bin:$JRE_HOME/bin
ANDROID_TOOLS=/usr/lib/android-sdk/sdk/tools
PATH=$PATH:$HOME/bin/$ANDROID_TOOLS
ANDROID_PLATFORMTOOLS=/usr/lib/android-sdk/sdk/platform-tools
PATH=$PATH:$HOME/bin/$ANDROID_PLATFORMTOOLS
export JAVA_HOME
export JRE_HOME
export ANDROID_TOOLS
export ANDROID_PLATFORMTOOLS
export PATH
```

- ou vous lancez directement cette commande depuis un terminal en étant dans le dossier *android-sdk/sdk/tools*

```
1 ~$ echo "export PATH=\${PATH}:\$(pwd)" >> ~/.
    bashrc
```

Faire de même pour le dossier *android-sdk/sdk/platform-tools*
Si vous choisissez le premier choix, n'oubliez pas d'exécuter *bash_ profile.sh*.

**Windows**

Commencer par télécharger le fichier zip qui se trouve dans ce lien
- `http://developer.android.com/sdk/index.html`

Puis, installer le SDK. Ensuite, comme pour Java JDK, vous devez aussi ajouter l'emplacement du SDK dans le *PATH*.

**Apache Ant**

**Linux Ubuntu & Windows**

- Téléchargement
  - `http://ant.apache.org/bindownload.cgi`
- Installation
  - Décompression du fichier zip
  - Ajout au path
    L'ajout de l'emplacement du logiciel au PATH système se fait de la même manière que pour les deux autres logiciels.

# 2 L'interface de ligne de commande

## 2.1 Installation de Cordova

Concernant l'installation de Cordova, il faut utiliser le module *npm* de Node.js, ouvrez un terminal et lancer la commande

```
1 ~$ npm install -g cordova
```

Si vous êtes sous OS X ou Linux, commencer la commande par *sudo* si vous n'êtes pas administrateur de votre système, ce qui est généralement le cas.

## 2.2 Création d'un répértoire

Comme pour tout projet, il faut commencer par créer un répertoire qui va accueillir tous les fichiers de développement. Pour cela, ouvrez un terminal et lancer la commande suivante

```
1 ~$ cordova create hello com.example.hello HelloWorld
```

Cordova se charge de créer tous les fichiers indispensables pour commencer votre application.

## 2.3 Ajout d'une plateforme

Avant de construire un projet, il faut spécifier une ou plusieurs plateformes de développement. Pour cela, il faut en ajouter :

```
1 ~$ cordova platform add nom_platform
```

Tel que *nom_platform* représente une des plateformes mobiles où vous voulez faire fonctionner votre application

- android
- ios
- amazon-fireos
- blackberry10
- firefoxos
- wp8
- windows8
- ...

La possibilité de lancer ces commandes dépend de votre machine. Par exemple, pour ajouter une plateforme iOS, vous devez développer sur Mac.

## 2.4 Compilation et lancement de l'application

Afin de tester l'application, il faut

- (re)compiler les sources en lançant la commande suivante

  ```
  1 ~$ cordova build
  ```

  ou

  ```
  1 ~$ cordova build nom_platform
  ```

  pour une plateforme déjà ajoutée
- émuler l'application
  . Pour android, lancer l'émulateur android puis exécuter la commande

  ```
  1 ~$ cordova emulate android
  ```

  Si vous voulez tester l'application directement sur votre téléphone

  ```
  1 ~$ cordova run android
  ```

  . Pour iOS, vous pouvez directement lancer la commande

  ```
  1 ~$ cordova run ios
  ```

qui lancera l'application sur votre téléphone si celui ci est raccordé à votre Mac, ou dans le cas contraire lancera un émulateur iOS sur votre Mac.
Vous pouvez aussi émuler l'application directement sur votre navigateur web en utilisant l'extension *Ripple* que vous pouvez télécharger sur le site
- `http://emulate.phonegap.com/`
Ceci évite de (re)compiler les sources pour tester les fonctions natives de Cordova après chaque modification.

# 3 Modules

Afin que l'application mobile communique avec les différentes caractéristiques de votre téléphone, il faut ajouter des modules (plugins).

## 3.1 Caméra

Le module caméra est utilisé afin de prendre une photo ou de récupérer des photos depuis la galerie de photos du téléphone. Pour utiliser ce plugin, vous devez premièrement l'installer en lançant la commande suivante dans un terminal

```
~$ cordova plugin add org.apache.cordova.camera
```

L'utilisation du module se fait grâce à la méthode

```
navigator.camera.getPicture( cameraSuccess, cameraError,
    cameraOptions );
```

avec par exemple

```
var cameraSuccess = function(imageData) {
  // photo prise
};
var cameraError = function(message) {
  // erreur lors de la prise de photo
};
var cameraOptions = {
  quality : 50,
  destinationType : Camera.DestinationType.FILE_URI,
  sourceType : Camera.PictureSourceType.CAMERA,
  allowEdit : true,
  encodingType: Camera.EncodingType.JPEG,
  targetWidth: 100,
  targetHeight: 100,
  popoverOptions: CameraPopoverOptions,
  saveToPhotoAlbum: false
};
```

**Options**

- **quality** : qualité de l'image (0 - 100)
- **destinationType** : format de la variable de retour

  - Camera.DestinationType.DATA_URL : string encodé en base64
  - Camera.DestinationType.FILE_URI : image file URI
  - Camera.DestinationType.NATIVE_URI : image native URI (assets-library :// sur iOS ou content :// sur Android)
- **sourceType** : définit la source de l'image
  - Camera.PictureSourceType.PHOTOLIBRARY : sélectionner une image depuis la galerie de photos
  - Camera.PictureSourceType.CAMERA : capturer une image
  - Camera.PictureSourceType.SAVEDPHOTOALBUM : sélectionner une image depuis la galerie de photos
- **allowEdit** : autoriser l'édition de l'image avant sa sélection
- **encodingType** : type d'encodage de l'image retournée
  - Camera.EncodingType.JPEG
  - Camera.EncodingType.PNG
- **targetWidth** : largeur de l'image (entier)
- **targetHeight** : longueur de l'image (entier)
- **popoverOptions**
- **cameraDirection** : choisir la caméra à utiliser
  - Camera.Direction.BACK : caméra arrière
  - Camera.Direction.FRONT : caméra frontale
- **mediaType** : le type de média à sélectionner depuis l'album photo
  - Camera.MediaType.PICTURE : sélectionner les images
  - Camera.MediaType.VIDEO : sélection les vidéos
  - Camera.MediaType.ALLMEDIA : sélectionner tous les médias
- **saveToPhotoAlbum** : sauvegarder l'image dans l'album photo (booléen)

### Exemples

1. Récupérer l'image encodée en base64

```
1   navigator.camera.getPicture(onSuccess, onFail, {
2     quality: 50,
3     destinationType: Camera.DestinationType.DATA_URL
4   });
5
6   function onSuccess(imageData) {
7     var image = document.getElementById('myImage');
8     image.src = "" + imageData;
9   }
10
11  function onFail(message) {
12    alert('Erreur : ' + message);
13  }
```

2. Récupérer l'emplacement de l'image

```
1   navigator.camera.getPicture(onSuccess, onFail, {
2     quality: 50,
3     destinationType: Camera.DestinationType.FILE_URI
4   });
5
```

```
6     function onSuccess(imageURI) {
7       var image = document.getElementById('myImage');
8       image.src = imageURI;
9     }
10
11    function onFail(message) {
12      alert('Erreur : ' + message);
13    }
```

## 3.2 Géolocalisation

Ce module fournit des informations à propos de la position de l'appareil comme la latitude, la longitude ou la vitesse. L'installation de ce module se fait grâce à la commande

```
1  ~$ cordova plugin add org.apache.cordova.geolocation
```

Les méthodes disponibles pour ce module sont

```
1    navigator.geolocation.getCurrentPosition
2    navigator.geolocation.watchPosition
3    navigator.geolocation.clearWatch
```

**navigator.geolocation.getCurrentPosition**
Cette méthode retourne la position à la fonction callback de succès en injectant comme paramètre un objet *Position*. Si une erreur survient, un objet *PositionError* est passé à la fonction callback d'erreur. L'appel à cette méthode se fait comme suit

```
1    navigator.geolocation.getCurrentPosition(
        geolocationSuccess, [geolocationError], [
        geolocationOptions]);
```

**Paramètres**
- **geolocationSuccess** : fonction callback de succès (obligatoire)
- **geolocationError** : fonction callback d'erreur (optionnel)
- **geolocationOptions** : options de géolocalisation (optionnel)

**Exemple**

```
1  var onSuccess = function(position) {
2      alert('Latitude: ' + position.coords.latitude + '\n' +
          'Longitude: ' + position.coords.longitude + '\n' + '
          Altitude: ' + position.coords.altitude + '\n' + '
          Accuracy: ' + position.coords.accuracy  + '\n' + '
          Altitude Accuracy: ' +
          position.coords.altitudeAccuracy  + '\n' + 'Heading:
           ' + position.coords.heading + '\n' + 'Speed: ' +
          position.coords.speed + '\n' + 'Timestamp: ' +
          position.timestamp + '\n');
3  };
```

```
4 function onError(error) {
5     alert('code: '    + error.code    + '\n' +
6           'message: ' + error.message + '\n');
7 }
8 navigator.geolocation.getCurrentPosition(onSuccess, onError
    );
```

**navigator.geolocation.watchPosition**
Cette méthode retourne la position de l'appareil à chaque fois qu'un changement de position est détecté. L'appel à cette méthode se fait comme suit

```
1 var watchID = navigator.geolocation.watchPosition(
    geolocationSuccess, [geolocationError], [
    geolocationOptions]);
```

**Paramètres**
- **geolocationSuccess** : fonction callback de succès (obligatoire)
- **geolocationError** : fonction callback d'erreur (optionnel)
- **geolocationOptions** : options de géolocalisation (optionnel)

**Retour**
La méthode retourne un *watchID* qui peut être utilisé afin d'arrêter de surveiller le changement de position grâce à la méthode *navigator.geolocation.clearWatch*.

**Exemple**

```
1 function onSuccess(position) {
2     var element = document.getElementById('geolocation');
3     element.innerHTML = 'Latitude: ' +
          position.coords.latitude + '<br />' + 'Longitude: '
          + position.coords.longitude + '<br />' + '<hr />' +
          element.innerHTML;
4 }
5 function onError(error) {
6     alert('code: ' + error.code + '\n' + 'message: ' +
          error.message + '\n');
7 }
8 var watchID = navigator.geolocation.watchPosition(onSuccess
    , onError, { timeout: 30000 });
```

**navigator.geolocation.clearWatch**
Cette méthode arrête de surveiller le changement de position de l'appareil.

```
1 navigator.geolocation.clearWatch(watchID);
```

**Paramètres**
- **watchID** : L'id qui a été retourné par la méthode *navigator.geolocation.watchPosition* (String)

**Exemple**

```
1 var watchID = navigator.geolocation.watchPosition(onSuccess
      , onError);
2
3 // ... du code
4
5 navigator.geolocation.clearWatch(watchID);
```

**geolocationOptions**
Paramètre optionnel afin de personnaliser la recherche de la position

**Options**
- **enableHighAccuracy** : Si cette propriété est à true, alors le framework utilisera des méthode de géolocalisation plus précise comme le système de positionnement par satellite. Par défaut, l'appareil utilise des méthodes basées sur un système de réseaux.
- **timeout** : Le temps maximum d'attente avant que la fonction callback de succès soit appelée. Si la fonction n'est pas appelée dans ce délai, la fonction de callback d'erreur est alors appelée.
- **maximumAge**

## 3.3 Stocakge des données

Stocker des données pour une application mobile est indispensable afin de sauvegarder des préférences, récupérer des données, ou d'interagir avec un site web. Pour cela, Cordova dispose de plugins.

### 3.3.1 LocalStorage

Comme son nom l'indique, *localStorage* stocke des données localement sous forme d'une pair de clé-valeur. Cet objet (*localStorage*) dispose de quatre méthodes qui sont
- **localStorage.getItem(key)** : Retourne la valeur associée à la clé *key*
- **localStorage.setItem(key, value)** : Sauvegarde la pair key-value
- **localStorage.removeItem(key)** : Supprime un objet grâce à sa clé *key*
- **localStorage.clear()** : Supprime tous les objets savegardés dans localStorage

L'utilisation de ces méthodes est synchrone.

### 3.3.2 WebSQL

Contrairement à *localStorage* qui ne peut stocker que des pairs clé-valeur, le module *WebSQL* peut interagir avec une base de données côté client de manière asynchrone.

### Méthodes

1. **openDatabase**

```
1 openDatabase(name, version, displayName, estimatedSize
    , creationCallback);
```

Cette méthode ouvre une base de données en lui passant comme paramètres
- **name** : le nom de la base de données (obligatoire)
- **version** : la version de la base de données (obligatoire)
- **displayName** (obligatoire)
- **estimatedSize** : la taille estimée des données qui vont être stockées dans la base de données (obligatoire)
- **creationCallback** : cette fonction est appelée si la base de données n'a pas encore été créée (optionnel)

L'objet de retour est *Database*.

2. **transaction & readTransaction**

```
1 transaction(callback, errorCallback, successCallback);
2 readTransaction(callback, errorCallback,
    successCallback);
```

Ces deux méthodes effectuent une transaction de manière asynchrone, elles ont comme paramètres
- **callback** : cette fonction contient les instructions à effectuer de manière asynchrone (obligatoire)
- **errorCallback** : en cas d'erreur, cette fonction est appelée (optionnel)
- **successCallback** : en cas de succès : cette fonction est appelée (optionnel)

La différence entre ces deux méthodes est que *transaction* est en mode lecture/écriture et *readTransaction* est en mode lecture seulement.

3. **executeSql**

```
1 void executeSql(sqlStatement, arguments, callback,
    errorCallback);
```

La méthode *executeSql* exécute une requête sql, elle doit être lancée dans une fonction de transaction. Les paramètres sont
- **sqlStatement** : la requête sql
- **arguments**
- **callback** : callback en cas de succès
- **errorCallback** : callback en cas d'erreur

### Exemple

```
1 function prepareDatabase(ready, error) {
2   return openDatabase('documents', '1.0', 'Offline document
      storage', 5*1024*1024, function (db) {
3     db.changeVersion('', '1.0', function (t) {
```

```
4         t.executeSql('CREATE TABLE docids (id, name)');
5       }, error);
6     });
7   }
8
9   function showDocCount(db, span) {
10    db.readTransaction(function (t) {
11      t.executeSql('SELECT COUNT(*) AS c FROM docids', [],
            function (t, r) {
12        span.textContent = r.rows[0].c;
13      }, function (t, e) {
14        // couldn't read database
15        span.textContent = '(unknown: ' + e.message + ')';
16      });
17    });
18  }
19
20  prepareDatabase(function(db) {
21    // got database
22    var span = document.getElementById('doc-count');
23    showDocCount(db, span);
24  }, function (e) {
25    // error getting database
26    alert(e.message);
27  });
```

### 3.3.3 Synchronisation des données avec un site web

Afin de synchroniser les données avec un site web, il faut tout d'abord autoriser l'accès au fichier php (du site web) qui interagit avec la base de données et qui retourne les enregistrements récupérés. Ce mécanisme s'appelle le *Cross-origin resource sharing* CORS. Ainsi, pour activer le (CORS), deux solutions sont envisageables

1. La première méthode consiste à modifier le fichier de configuration du serveur. Il faut tout d'abord activer les header dans Apache en lançant la commande suivante dans un terminal

   ```
   ~$ sudo a2enmod headers
   ```

   puis faire des modifications dans la balise <Directory>, <Location>, <Files> ou <VirtualHost>

   - Autoriser tous les domaines

     ```
     Header set Access-Control-Allow-Origin "*"
     ```

   - N'autoriser qu'un seul domaine

     ```
     Header set Access-Control-Allow-Origin "http://www.
         domain.com"
     ```

   - Autoriser plusieurs domaines (ici domain1 et domain2)

     ```
     SetEnvIf Origin "http(s)?://(www.)?(domain1.com|
         domain2.com)\$" AccessControlAllowOrigin=\$0\$1
     ```

```
Header set Access-Control-Allow-Origin \%{
   AccessControlAllowOrigin}e env=
   AccessControlAllowOrigin
```

Ensuite, vous devez redémarrer votre serveur avec la commande

```
~$ sudo service apache2 reload
```

Si vous n'avez pas la possibilité de modifier le fichier de configuration du serveur, vous pouvez inclure ces lignes directement dans le fichier *.htaccess*.

2. La deuxième méthode consiste à modifier directement le fichier php qui interagit avec la base de données en ajoutant

```
<?php
header("Access-Control-Allow-Origin: *");
```

Après avoir activé le CORS, il ne reste plus qu'à faire un appel au fichier php depuis votre application mobile en utilisant Ajax.

### Exemple

#### Côté serveur

```
<?php
header("Access-Control-Allow-Origin: *");
if (isset($_GET['selectclient']) {
    $clients = array(); // tableau Ã  retourner
    $pdo = new PDO('mysql:host=localhost;dbname=bdd', $user
        , $pass);// connexion Ã  la bdd
    $req = $pdo->prepare('SELECT * FROM client');
    if (!$req->execute()) {
        return json_encode(new Error(errorcode,
            errormessage));// en cas d'erreur retourner un
            code et un message d'erreur
    }
    while ($donnees = $req->fetch()) {
        $clients[] = new Client($donnees['id'], $donnees['
            nom'], $donnees['prenom']);
    }
    return json_encode($clients);
}
?>
```

#### Côté client

```
function selectClient() {
  $.ajax({
    url:"http://www.domaineserveur.com/bdd.php?selectclient
        ",
    method: "GET",
    dataType: "json",
```

```
6       success: function(data) {
7         if (data.errorcode !== undefined) {// erreur lors de
              la rÃ©cupÃ©ration de donnÃ©es
8           console.log('Erreur : ' + data.errorcode + ' => ' +
                data.errormessage);
9         } else {
10          $.each(data, function(i, client){
11            console.log('Client ' + i + ' : ' + client.nom +
                  ' ' + client.prenom);
12          });
13        }
14      },
15      error: function(jqXHR, textStatus, errorThrown) {
16        console.log(jqXHR.responseText);
17      }
18    });
19  }
```

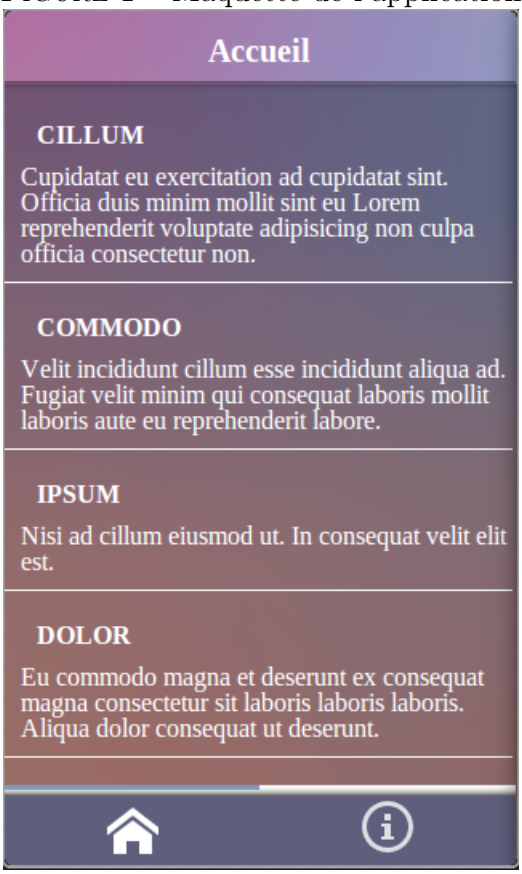
# 4 Hello World

Cette application consiste à lister des titres de vidéos et leur description, et à chaque fois que l'utilisateur clique sur un titre, ce dernier est redirigé vers la vidéo.

## 4.1 Design

En ce qui concerne le design, j'ai décidé de choisir un template au format photoshop gratuit et le reproduire ainsi que de télécharger des icônes gratuites sur le site `https://icomoon.io/`.

FIGURE 1 – Maquette de l'application




Comme vous voyez dans l'image ci-dessus, l'application se compose de trois parties essentielles.

### 4.1.1 L'en-tête (Header)

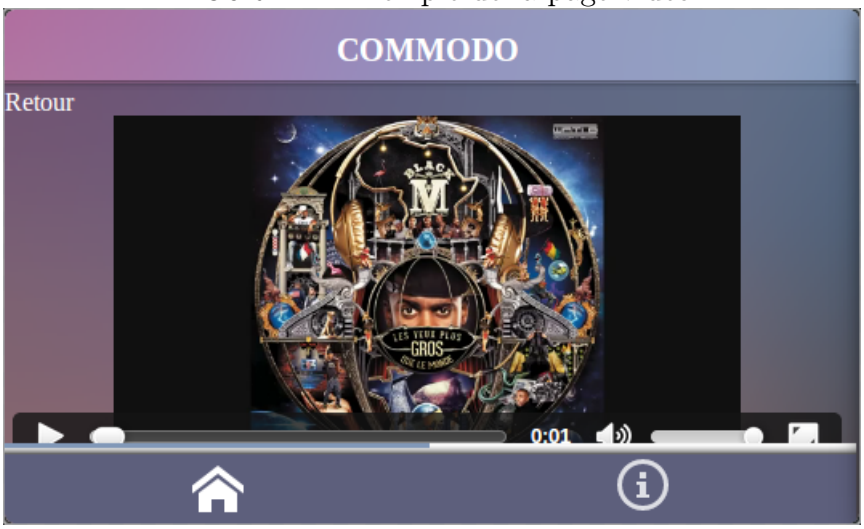
Cette partie consiste à afficher le titre de la page au fur et à mesure de la navigation. Le code css associé est le suivant

```
1  .header {
2    position: absolute;
3    top: 0;
4    left: 0;
5    right: 0;
6    background: url("../img/shadow.png") no-repeat;
7    background-position-y: 100%;
8    height: 50px;
9    line-height: 50px;
10   color: #fff;
11   text-align: center;
12   font-size: 20px;
13   font-weight: 700;
14 }
```

### 4.1.2 Le corps (Body)

Concernant le corps de l'application, il sert à lister les différentes vidéos si l'utilisateur se trouve à la page d'accueil ou les mentions légales s'il est présent dans la page "about". Si l'utilisateur clique sur un titre de vidéo, ce dernier sera redirigé vers une autre page où le corps de l'application sera la vidéo.

FIGURE 2 – Exemple de la page vidéo




Une partie du code css associé au corps de l'application

```
.view {
  position: absolute;
  top: 50px;
  left: 0;
  right: 0;
  bottom: 50px;
}
.panel {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  overflow-x: hidden;
  overflow-y: scroll;
  -webkit-overflow-scrolling: touch;
}
/*page video*/
.video {
  color: #fff;
  background: url("../img/shape.png") no-repeat;
  background-position-y: 100%;
  background-position-y: 100%;
  padding: 10px;
}
```

### 4.1.3 Footer

Cette partie consiste à naviguer entre la page d'accueil et la page de mentions légales. Une partie du code css associé

```
1  .footer {
2    position: absolute;
3    bottom: 0;
4    left: 0;
5    right: 0;
6    height: 50px;
7    line-height: 50px;
8    background: url("../img/footer.png");
9    text-align: center;
10 }
11 .footer a {
12   width: 50%;
13   height: 100%;
14   display: block;
15   float: left;
16   box-sizing: border-box;
17   overflow: hidden;
18   zoom: 1;
19 }
```

## 4.2 Côté client

Pour cette application, j'ai décidé d'utiliser un fichier json pour stocker les vidéos sous le format

```
1 {
2   "id": id de la vidÃ©o,
3   "title": titre ,
4   "content": description,
5   "link": url local de la vidÃ©o
6
7 }
```

## 4.3 Côté client

Contrairement aux sites web qui sont en général constitués d'un fichier HTML pour chaque page, les applications mobiles doivent être fluides et donc n'utiliser qu'un seul fichier HTML comme pour les applications web mono-page (SPA). La navigation se fera en Javascript. Or, maintenir une application complexe en utilisant Javascript peut s'avérer difficile. Ainsi, l'utilisation des framework facilite le développement. Pour cet exemple, j'ai décidé d'utiliser le framework *AngularJs*, qui est un framework développé par *Google* dédié pour la réalisation des SPA. Ce framework utilise le design pattern MVC (Model-View-Controller), ainsi j'ai décidé de répartir mes fichiers javascript dans trois dossiers

- **controller** : pour les fichiers controller
- **service** : pour les fichiers modèle

- **libs** : pour les librairies externes (angularjs, jquery)

ainsi qu'un fichier qui se place dans la racine qui consiste à configurer l'application.

### 4.3.1 Configuration

La partie configuration consiste à créer un nouveau module et définir les pages que l'utilisateur peut voir.

```
1  var app = angular.module('MyVideos', ['ngRoute']); // crÃ©ation d'un nouveau module et utilisation du service ngRoute
2  app.config(function($routeProvider) {
3      $routeProvider.when('/', { // si on est Ã  la racine, on fait appel Ã  home.html
4          templateUrl: 'include/home.html',
5          controller: 'Home'
6      }).when('/about', { // /about, on fait appel Ã  about.html
7          templateUrl: 'include/about.html',
8          controller: 'About'
9      }).when('/video/:id', { // /video/:id, on fait appel Ã  video.html (id reprÃ©sente une valeur passÃ©e dans l'url qu'on peut rÃ©cupÃ©rer par $routeParams)
10         templateUrl: 'include/video.html',
11         controller: 'Play'
12     }).otherwise({ // sinon, on redirige vers la page d'accueil
13         redirectTo: '/'
14     });
15 });
```

### 4.3.2 Modèle

Le modèle est la partie chargée d'interagir avec une base de données. Il comporte des méthodes pour récupérer, ajouter modifier ou supprimer des données. Dans cet exemple, j'ai eu besoin d'un seul fichier modèle qui est *service.js*. Il dispose de deux méthodes, la première pour récupérer toutes les vidéos et l'autre pour récupérer une vidéo grâce à son identifiant.

```
1  app.service('VideosService', function($http, $q) {
2      var self = this;
3      self.videos = false;
4      this.getAll = function() {
5          var deferred = $q.defer(); // on utilise le service $q pour attendre les donnÃ©es
6          if (self.videos !== false) { // si les vidÃ©os sont dÃ©jÃ  rÃ©cupÃ©rÃ©es
7              deferred.resolve(self.videos);
8          } else { // sinon
9              $http.get('json/videos.json').success(function(data, status) {
10                 self.videos = data; // on sauvegarde les donnÃ©es
```

```
11                  deferred.resolve(data); // la promesse
                        retournera les donnÃ©es rÃ©cupÃ©rÃ©es
12              }).error(function(data, status) {
13                  deferred.reject('Impossible de rÃ©cupÃ©rer
                        les vidÃ©os'); // la promesse retournera
                         une erreur
14              });
15          }
16          return deferred.promise; // on retourne la promesse
17      }
18      this.get = function(id) {
19          var deferred = $q.defer();
20          // on fait appel Ã  la fonction getAll pour
                rÃ©cupÃ©rer toutes les vidÃ©os et on utilise
                then pour attendre la promesse puis on fait
                appel Ã  deux fonctions de callback
21          var videos = self.getAll().then(function(videos) {
22              // fonction de succÃ¨s
23              angular.forEach(videos, function(video) {
24                  if (video.id == id) {
25                      deferred.resolve(video); // on retourne
                            la vidÃ©o quand les id sont Ã©gaux
26                  }
27              });
28          }, function(msg) {
29              // fonction d'erreur
30              deferred.reject(msg);
31          })
32          return deferred.promise; // on retourne la promesse
33      }
34  });
```

### 4.3.3 Vue

L'objectif de la vue est d'afficher les données renvoyés par le modèle passant par le contrôleur. Elle a aussi pour tâche d'interagir avec l'utilisateur lorsque ce dernier effectue une action (clic, sélection d'un bouton, ...). Ces événements sont ainsi envoyés au contrôleur.
Dans cet exemple, j'ai décidé d'utiliser trois vues.

#### Page d'accueil

Afficher la liste des vidéos présentes dans le serveur (fichier json).

```
1  <section id="wrapper" class="panel">
2      <div id="scroller">
3        <div class="video" ng-repeat="video in videos"
             ng-click="goTo('video/' + video.id)">
4          <header>{{video.title}}</header>
5          <body>
6              {{video.content}}
7          </body>
8          <footer>{{video.footer}}</footer>
9        </div>
10     </div>
```

```
11  </section>
```

**Explication**

- Les chaînes qui sont entre accolades seront remplacées par les valeurs fournies par le contrôleur.
- La directive *ng-repeat* consiste à répéter l'élément dont elle est l'attribut autant de fois que de vidéos
- La directive *ng-click* consiste à rediriger vers la vidéo lorsque l'utilisateur clique sur l'élément

**Page de mentions légale**
Afficher les mentions légales

**Page vidéo**
Afficher la vidéo, ainsi qu'un lien *Retour*, afin de retourner vers la page d'accueil.

```
1  <div class="panel">
2    <a href="#home">Retour</a>
3    <video width="100%" height="100%" ng-src="{{video.link}}"
         controls></video>
4  </div>
```

### 4.3.4 Contrôleur

Le contrôleur réalise la synchronisation entre la vue et le modèle. Il s'occupe de la mise à jour de la vue lorsque les données sont modifiées et de la demande de modification des données au modèle lorsque la vue lui envoi des événements. J'ai décidé ainsi de spécifier un contrôleur pour chaque vue.

**Page d'accueil**
Son but est de demander au modèle la récupération des vidéos et ensuite les fournir à la vue.

```
1  app.controller('Home', function($scope, VideosService,
       $rootScope, $location) {
2      $scope.videos = []; // tableau de vidÃ©os vide Ã  l'
           initialisation
3      $rootScope.header = 'Accueil'; // header est tjrs
           Ã©gale Ã  Accueil (c'est le scope (vue) qui s'en
           charge)
4      // demander au modÃ¨le (service) la rÃ©cupÃ©ration de
           toutes les vidÃ©os et l'utilisation de then pour
           attendre la promesse
5      VideosService.getAll().then(function(videos) {
6          // fonction succÃ¨s
7          $scope.videos = videos; // fournir Ã  la vue les
               vidÃ©os
8      }, function(msg) {
9          // fonction d'erreur
```

```
10          console.log(msg);
11      });
12      // rediriger vers une vidÃ©o
13      $scope.goTo = function(url) {
14          $location.url(url);
15      }
16  });
```

**Page de mentions légale**
Modifier le titre de l'en-tête.

```
1  app.controller('About', function($scope, $rootScope) {
2      $rootScope.header = 'About'; // modifier le header (
           scope parent)
3  });
```

**Page vidéo**
Faire appel à une vidéo.

```
1  app.controller('Play', function($scope, VideosService,
       $routeParams, $rootScope, $sce) {
2      $scope.video = {}; // objet vide Ã  l'init
3      $rootScope.header = ""; // header vide Ã  l'init
4      // demande au modÃ¨le la rÃ©cupÃ©ration d'une vidÃ©o
           grÃ¢ce Ã  l'id qui est passÃ© par url, puis l'
           utilisation de then pour attendre la promesse
5      VideosService.get($routeParams['id']).then(function(
           video) {
6          // fonction succÃ¨s
7          $scope.video = video; // fournir la vidÃ©o Ã  la
               vue
8          $rootScope.header = video.title.toUpperCase(); //
               modification du header
9          $scope.video.link = $sce.trustAsResourceUrl(String(
               video.link)); // dire Ã  angluar js que le lien
               est fiable
10     }, function(msg) {
11         // fct erreur
12         console.log(msg);
13     });
14 });
```

### 4.3.5 Conclusion

Comme nous avons pu le remarquer, l'utilisation du framework *AngularJs* facilite la création d'application monopage. En effet, il y a moins de code à écrire, puisque c'est *AngularJs* qui s'occupe de la gestion de l'historique et la synchronisation entre la vue et le contrôleur.

## 4.4 Télécharments

Pour télécharger l'application, exécuter la commande

```
1 ~$ git clone https://github.com/sizo0/
     cordova_helloworld.git
```

# Références

[1] Apache : *Documentation Apache Cordova*. http://cordova.apache.org/docs/en/3.5.0.

[2] Jonathan Boyer | Grafikart : *Tutoriel Apache Cordova*. http://www.grafikart.fr/tutoriels/cordova/, 2013.