Licence Pro MI-AW

Module INFO-1

Exemple d'Utilisation de Plugins

- *LocalStorage*
- *Contacts*
- *Geolocation*
- *InAppBrowser*
- *Camera*

Storage HTML 5

- Les « Webview » HTML5 proposent différentes façons de stocker des données locales :
  - Web Storage : hash-map persistante permettant de stocker des couples clé/valeur
  - Web SQL Database : base de données relationnelles SQL
  - Indexed DataBase : similaire à Web Storage : couples clé/valeur. Mais la possibilité d'indexer permet de faire des recherches plus rapidement
- On peut utiliser ces outils de stockage sans avoir besoin d'installer un plugin cordova particulier

Local Storage

- Accessible via l'objet window.localstorage
- Méthodes :
  - Récupérer la clé du ième élément stocké
    var keyName = window.localStorage.key(i);
  - Enregister une paire (clé,valeur) :
    window.localStorage.setItem("key", "value");
  - Accéder à un élément par sa clé :
    var value = window.localStorage.getItem("key");
  - Supprimer un élément par sa clé :
    window.localStorage.removeItem("key");
  - Vider le storage :
    window.localStorage.clear();

Exemple Shifumi(1)

- On complète le modèle pour gérer la persistance des entités « Partie »

```
// Objet dao pour gérer la Persistance des parties dans le local storage.
// On stocke des paires (nomJoeur, partie).
modele.dao = {
    // sauvegarde la partie au format JSON dans le storage
    savePartie: function(partie) {
        window.localStorage.setItem(partie.nomJoueur,
                                    JSON.stringify(partie));
    },
    // charge la partie d'un joueur, si elle existe, depuis le storage
    loadPartie: function(nomJoueur) {
        var partie = window.localStorage.getItem(nomJoueur);
        if (partie === null) {
            // s'il n'y a pas de partie au nom de ce joueur, on en crée une
            return new modele.Partie(nomJoueur,0,0,0);
        }
        else { // sinon on convertit la partie en objet JS
            partie = JSON.parse(partie); // convertit la chaine JSON
             Object.setPrototypeOf(partie,modele.Partie.prototype);
            return partie;
        }
    }
}
```

Exemple Shifumi(2)

- Et on modifie le contrôleur pour utiliser la couche « persistance »

```
////////////////////////////////////////////////////////////////////////////////
controleur.vueAccueil = {
    nouvellePartie: function () {
        // on récupère de l'information de la vue en cours
        var nomJoueur = $("#nomJoueur").val();
        if (nomJoueur === "") {
            alert("Entrez un nom de joueur svp");
        } else {
            // On charge la partie du joueur depuis le localstorage
            controleur.session.partieEnCours = modele.dao.loadPartie(nomJoueur);
            // ... Suite inchangée
        }
    }
};

////////////////////////////////////////////////////////////////////////////////
controleur.vueJeu = {
    jouer: function (coupJoueur) {
        // on interroge le modèle pour voir le résultat du nouveau coup
        var resultat = controleur.session.partieEnCours.nouveauCoup(coupJoueur);
        // le score a changé => on sauvegarde la partie en cours
        modele.dao.savePartie(controleur.session.partieEnCours);
        // ... Suite inchangée
    },
};
```

Plugin contacts (1)

- https://www.npmjs.com/package/cordova-plugin-contacts
- Installation : **cordova plugin add cordova-plugin-contacts**
- Fournit un objet global navigator.contacts qui donne accès à la base de données des contacts

Méthodes	Objets
navigator.contacts.create navigator.contacts.find navigator.contacts.pickContact	Contact ContactName ContactField ContactAddress ContactOrganization ContactFindOptions ContactError ContactFieldType

Plugin contacts (2)

- La méthode **navigator.contacts.pickContact** permet de lancer une interface pour choisir un **unique contact** parmi ceux du téléphone

- C'est une méthode asynchrone qui attend 2 callbacks (CB) en paramètres :

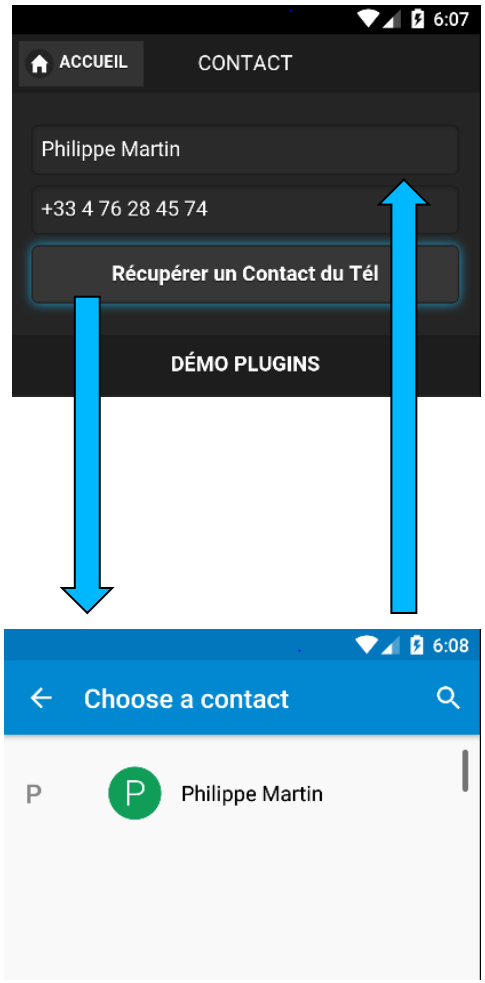
  - `successCB` qui sera exécutée en cas de succès et qui recevra en paramètre un objet **contact**
  - `errorCB` qui sera exécutée en cas d'erreur

Plugin contacts (3)

```
navigator.contacts.pickContact(
 function(contact) {
  console.log('Contact : ' +
   JSON.stringify(contact));
 },
 function(err) {
  console.log('Error: ' + err);
 }
);
```

- L'objet contact contient entre autre les champs :
  - **id**: identifiant *(DOMString)*
  - **displayName**: nom affichable *(DOMString)*
  - **name**: objet contenant tous les attributs du nom du contact. *(ContactName)*
  - **phoneNumbers**: un tableau contenant les numéros de téléphone du contact *(ContactField[])*

Plugin contacts - exemple (4)



Vue

```
<div data-role="main" class="ui-content">
<input type="text" id="contactDisplayName" placeholder="Nom" />
<input type="text" id="contactPhoneNumber" placeholder="Tél" />
<a href="javascript:window.controller.contactController.pickContact()"
  style="text-decoration: none">
  <button>Récupérer un Contact du Tél</button>
</a>
</div>
```

Contrôleur

```
controller.contactController = {
  pickContact: function () {
    // on réinitialise les champs nom et numero
    $("#contactDisplayName").val("");
    $("#contactPhoneNumber").val("");
    // on appelle la méthode du modèle permettant de récupérer un contact
    // en lui passant en paramètre successCB et errorCB
    model.pickContact(
      // successCB : on met à jour dans la vue nom et numero
      // avec le 1er numéro du contact récupéré
      function (unContact) {
        $("#contactDisplayName").val(unContact.displayName);
        $("#contactPhoneNumber").val(unContact.phoneNumbers[0]);
      },
      // erreurCB : on affiche la page erreur avec un message approprié
      function () {
        plugins.toast.showShortCenter("Pas de contact récupéré");
      }
    );
  }
};
```

Plugin contacts - exemple (5)

Modèle

```
////////////////////////////////////////////////////////////////////////////////
// Classe Contact
////////////////////////////////////////////////////////////////////////////////
model.Contact = function (displayName, phoneNumbers) {
  this.displayName = displayName;
  this.phoneNumbers = phoneNumbers;
};

////////////////////////////////////////////////////////////////////////////////
// Méthode pour acquérir un contact du téléphone
////////////////////////////////////////////////////////////////////////////////
model.pickContact = function (successCB, errorCB) {
  navigator.contacts.pickContact(
    function (contactTel) {
      // On récupère tous les numéros de tél du contactTel
      var phoneNumbers = [];
      for (var i = 0; i < contactTel.phoneNumbers.length; i++) {
        phoneNumbers.push(contactTel.phoneNumbers[i].value);
      }
      // On crée une entité Contact
      var unContact = new model.Contact(contactTel.displayName, phoneNumbers);
      // On appelle successCB en lui transmettant l'entité Contact
      successCB.call(this, unContact);
    },
    function (err) {
      console.log("Erreur Capture Contact : " + err.message);
      errorCB.call(this);
    }
  );
};
```

Plugin Geolocation (1)

- https://www.npmjs.com/package/cordova-plugin-geolocation
- Installation : cordova plugin add cordova-plugin-geolocation
- Fournit un objet global navigator.geolocation qui donne accès à la géolocalisation du terminal mobile

Méthodes	Objets
navigator.geolocation.getCurrentPosition navigator.geolocation.watchPosition navigator.geolocation.clearWatch	Position PositionError Coordinates

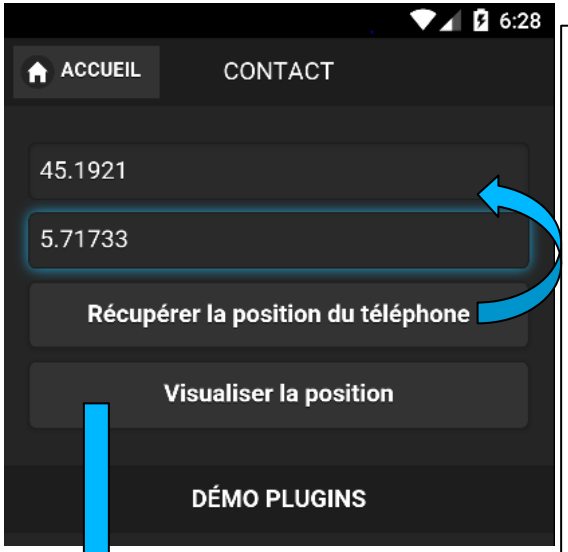
Plugin Geolocation (2)

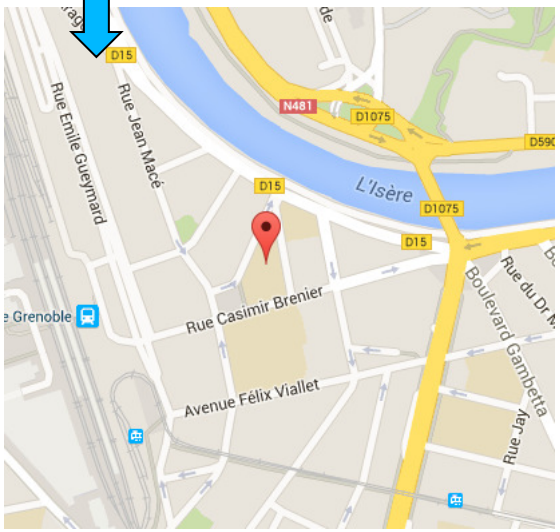
- La méthode **navigator.geolocation.getCurrentPosition** permet de récupérer la position courante

- C'est une méthode asynchrone qui attend 2 callbacks (CB) en paramètre et un tableau d'options :

```
navigator.geolocation.getCurrentPosition(
   geolocationSuccessCB,
   [geolocationErrorCB],
   [geolocationOptions]);
```

Plugin Geolocation - exemple (3)





Contrôleur

```
controller.locationController = {
  pickLocation: function () {
    // on réinitialise les champs latitude et longitude
    $("#locationLatitude").val("");
    $("#locationLongitude").val("");
    // on appelle la méthode du modèle pour récupérer la position
    // en lui passant en paramètre successCB et errorCB
    model.pickLocation(
      // successCB : on met à jour lat. et long. dans la vue
      function (location) {
        $("#locationLatitude").val(location.latitude);
        $("#locationLongitude").val(location.longitude);
      },
      // erreurCB : on affiche un message approprié
      function () {
        plugins.toast.showShortCenter(
           "Impossible de récupérer la position");
      }
    );
    return false;
  },
  showLocation: function () {
    var latitude = $("#locationLatitude").val();
    var longitude = $("#locationLongitude").val();
    var location = new model.Location(latitude, longitude);
    location.showInBrowser(); // Ouvre navigateur sur google maps
    return false;
  }
};
```

Plugin Geolocation - exemple (4)

Modèle

```
////////////////////////////////////////////////////////////////////////////
// Classe Location pour représenter une position
////////////////////////////////////////////////////////////////////////////
model.Location = function (latitude, longitude) {
  this.latitude = latitude;
  this.longitude = longitude;
  // Méthode pour visualiser une position dans un navigateur web
  this.showInBrowser = function () {
    var url = "https://maps.google.com/?q=" + this.latitude + "°," + this.longitude + "°";
    cordova.InAppBrowser.open(url, '_blank', 'location=yes'); // lancement d'un navigateur
  };
};
////////////////////////////////////////////////////////////////////////////
// Méthode pour acquérir la position courante du téléphone
////////////////////////////////////////////////////////////////////////////
model.pickLocation = function (successCB, errorCB) {
  navigator.geolocation.getCurrentPosition(
    function (position) {
      // On instantie une entité Location
      var location = new model.Location(position.coords.latitude,
                                        position.coords.longitude);
      // On appelle successCB en lui transmettant l'entité Location
      successCB.call(this, location);
    },
    function (err) {
      console.log("Erreur Capture GPS : " + err.message);
      errorCB.call(this);
    },
    // Options : position vieille de max 3s, 10s maxi pour répondre, position exacte demandée
      {maximumAge: 3000, timeout: 10000, enableHighAccuracy: true}
  );
};
```

Plugin InAppBrowser

- https://www.npmjs.com/package/cordova-plugin-inappbrowser
- Installation : **cordova plugin add cordova-plugin-camera**
- Permet d'afficher une vue « navigateur web » dans l'application
- Usage :

```
cordova.InAppBrowser.open(url, target, options);
```

  - **url** : l'URL à charger
  - **target** : la cible dans laquelle charger la page web :
    - **"_self"** : dans la webview de votre application
    - **"_blank"** : dans une nouvelle webview « InAppBrowser »
    - **"_system"** : dans le navigateur web par défaut de votre terminal mobile
  - **options** : pour transmettre des options au navigateur
    - **"location=yes"** par défaut

Plugin Camera (1)

- https://www.npmjs.com/package/cordova-plugin-camera
- Installation : **cordova plugin add cordova-plugin-camera**
- Fournit un objet global navigator.camera qui donne accès à l'application capture de photo de votre terminal

Méthodes	Objets
getPicture(successCallback, errorCallback, options) cleanup() onError : function onSuccess : function CameraOptions : Object	Camera .DestinationType : enum .EncodingType : enum .MediaType : enum .PictureSourceType : enum .PopoverArrowDirection : enum .Direction : enum CameraPopoverHandle CameraPopoverOptions

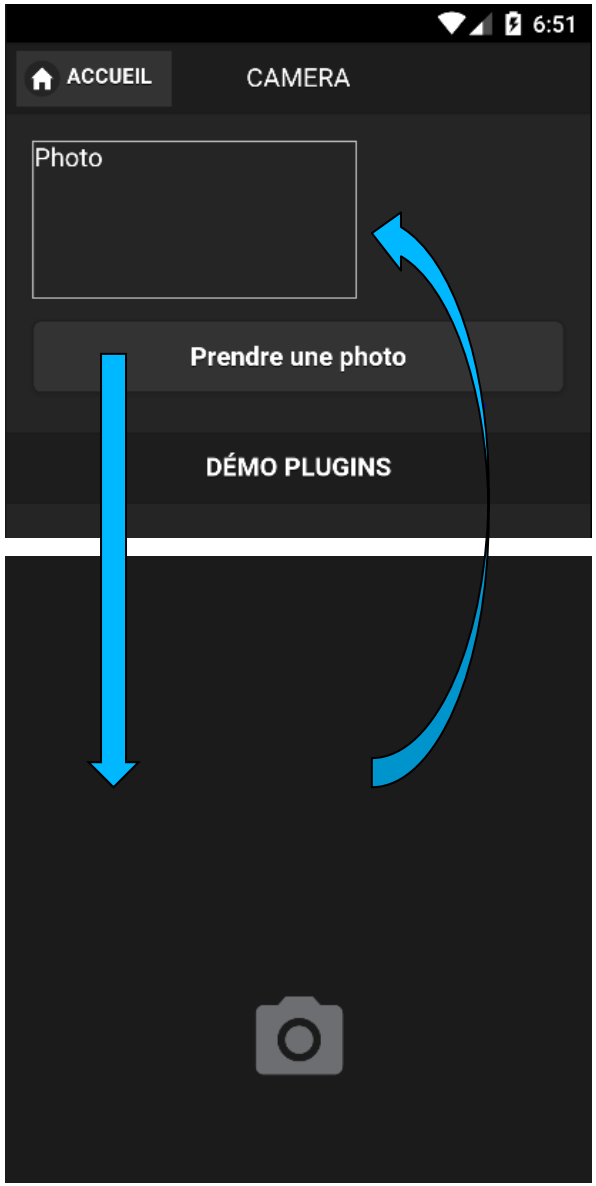
Plugin Camera (2)

- La méthode **camera.getPicture** permet de prendre une photo avec le terminal mobile

- C'est une méthode asynchrone qui attend elle aussi deux callbacks en paramètre plus un tableau d'options pour paramétrer la prise de vue :

```
camera.getPicture(successCallback,
                  errorCallback,
                  options);
```

Plugin Camera - exemple (3)



Contrôleur

```
controller.cameraController = {
  takePicture: function () {
    // Appel méthode du modèle permettant de prendre une photo
    // en lui passant en paramètre successCB et errorCB
    window.model.takePicture(
      // successCB : on met à jour la vue (champ cameraImage)
      function (uneImage) {
        // on récupère un objet Image
        $("#cameraImage").attr("src",uneImage.getBase64());
        uneImage.insert(
          function () { plugins.toast.showShortCenter(
                        "Image Enregistrée");},
          function () { plugins.toast.showShortCenter(
                        "Image non Enregistréee");}
        );
      },
      // erreurCB : on affiche un message approprié
      function () { plugins.toast.showShortCenter(
                    "Impossible de prendre une photo"); }
    );
  }
};

// Pour réinitialiser le champ cameraImage à l'affichage de la page
camera
$(document).on("pagebeforeshow", "#camera",
  function () {
    $("#cameraImage").attr("src","");
  }
);
```

Plugin Camera - exemple (4)

Modèle

```
////////////////////////////////////////////////////////////////////////////////
// Classe Image
////////////////////////////////////////////////////////////////////////////////
model.Image = function (id, imageData) {
  // Attributs
  this.id = id;
  this.imageData = imageData; // l'image Base64
  //
  // Méthode pour obtenir l'image au format Base64 (décompressé) avec en-tête MIME
  this.getBase64 = function() {
    return ""+this.imageData;
  },

  // Méthode pour insérer une nouvelle image en BD
  this.insert = function (successCB, errorCB) {
    var self=this; // pour accéder à l'objet Image dans le succesCB de la requête insert
    model.db.executeSql("INSERT INTO photos (imagedata) VALUES (?)",[this.imageData],
      function (res) {
        self.id=res.insertId; // on met à jour l'id de l'Image après insertion en BD
        successCB.call(this);
      },
      function (err) {
        console.log("Erreur Insertion : " + err.message);
        errorCB.call(this);
      }
    );
  };
};
```

Plugin Camera - exemple (5)

Modèle

```
////////////////////////////////////////////////////////////////////////////////
// Méthode pour capturer une image avec le téléphone encodée en Base64
////////////////////////////////////////////////////////////////////////////////

model.takePicture = function (successCB, errorCB) {
  navigator.camera.getPicture(
    function (imageData) {
      // imageData contient l'image capturée au format Base64, sans en-tête MIME
      // On appelle successCB en lui transmettant une entité Image
      successCB.call(this, new model.Image(0,imageData));
    },
    function (err) {
      console.log("Erreur Capture image : " + err.message);
      errorCB.call(this);
    },
    { quality: 50,
      destinationType: navigator.camera.DestinationType.DATA_URL,
      correctOrientation: true
    }
    // qualité encodage 50%, format base64 (et JPEG par défaut), orientation respectée
  );
};
```

Démo

- Voir le code source complet de l'application de démonstration
  - **DemoPlugins.zip**
- Conseils
  - Le code qui fait appel aux plugins peut avoir besoin d'être adapté en fonction de la plateforme cible pour laquelle on développe
  - Il est donc nécessaire « d'isoler » ce code dans un composant de votre application : dans le modèle (ou dans une partie du modèle)...
  - ... Ainsi, tout le reste de votre application (vue et contrôleur) sera complètement portable d'une plateforme à l'autre

Trucs & Astuces

- Live reload
  - Régresser vers Cordova 8 :
    npm install -g cordova@8.1.2
  - Nettoyer son projet :
    cordova clean
  - Installer le plugin live-reload :
    cordova plugin add cordova-plugin-browsersync
  - Lancer en mode live-reload :
    cordova run browser -- --live-reload
  - *On peut alors modifier son code et observer les résultats juste en rechargeant la page sur le navigateur*
- Inspecter sa BD Sqlite
  - Installer le plugin stetho :
    cordova plugin add cordova-plugin-stetho-android
  - *On peut alors depuis Chrome Inspecter la BD située sur le terminal mobile*