# PhoneGap Workshop

## Step 1: Choosing a Local Storage Option

1. Examine the different persistence options in /work/js/storage.
   * MemoryStore defined in memory-store.js
   * LocalStorageStore defined in ls-store.js
   * WebSqlStore defined in websql-store.js

* Test the application with the different stores. To change the local mechanism option for the application:
  + In work/index.html: add a script tag for the corresponding .js file: memory-store.js, ls-store.js, or websql-store.js.
  + In work/main.js: Instantiate the specific store in the initialize() function of the app object: MemoryStore, LocalStorageStore, or WebSqlStore.

## Step 2: Setting Up a “Single Page Application”

1. In index.html: remove the HTML markup inside the body tag.
2. In main.js, define a showHomeView() function inside the app object. Implement the function to programmatically add the Home View markup to the body element.

* showHomeView: function() {  
   var html =  
   "<div class='header'><h1>Home</h1></div>" +  
   "<div class='search-view'>" +  
   "<input class='search-key'/>" +  
   "<ul class='employee-list'></ul>" +  
   "</div>"  
   $('body').html(html);  
   $('.search-key').on('keyup', $.proxy(this.findByName, this));  
  }

1. In the app object’s initialize() method, pass an anonymous callback function as an argument to the constructor of the persistence store. The store will call this function after it has successfully initialized. In the anonymous function, call the showHomeView() function to programmatically display the Home View.

* initialize: function() {  
   var self = this;  
   this.store = new MemoryStore(function() {  
   self.showHomeView();  
   });  
  }

## Step 3: Using Handlebar Templates

Modify index.html as follows:

1. Create an HTML template for the Search View:

* <script id="home-tpl" type="text/x-handlebars-template">  
   <div class='header'><h1>Home</h1></div>  
   <div class='search-view'>  
   <input class='search-key'/>  
   <ul class='employee-list'></ul>  
   </div>  
  </script>

1. Create an HTML template to render the employee list items:

* <script id="employee-li-tpl" type="text/x-handlebars-template">  
   {{#.}}  
   <li><a href="#employees/{{this.id}}">{{this.firstName}} {{this.lastName}}</a></li>  
   {{/.}}  
  </script>

Modify main.js as follows:

1. In the app initialize() function, add the code to compile the two templates defined above:

* this.homeTpl = Handlebars.compile($("#home-tpl").html());  
  this.employeeLiTpl = Handlebars.compile($("#employee-li-tpl").html());

1. Modify showHomeView() to use the homeTpl template instead of the inlined HTML:

* showHomeView: function() {  
   $('body').html(this.homeTpl());  
   $('.search-key').on('keyup', $.proxy(this.findByName, this));  
  }

1. Modify findByName() to use the employeeLiTpl template instead of the inlined HTML:

* findByName: function() {  
   var self = this;  
   this.store.findByName($('.search-key').val(), function(employees) {  
   $('.employee-list').html(self.employeeLiTpl(employees));  
   });  
  }

## Step 4: Creating a View Class

1. Create a file called HomeView.js in work/js, and define a HomeView class implemented as follows:

* var HomeView = function(store) {  
    
   this.render = function() {  
   this.el.html(HomeView.template());  
   return this;  
   };  
    
   this.findByName = function() {  
   store.findByName($('.search-key').val(), function(employees) {  
   $('.employee-list').html(HomeView.liTemplate(employees));  
   });  
   };  
    
   // Wrap view in a div used to attach events  
   this.el = $('<div/>');  
   this.el.on('keyup', '.search-key', this.findByName);  
  }  
    
  HomeView.template = Handlebars.compile($("#home-tpl").html());  
  HomeView.liTemplate = Handlebars.compile($("#employee-li-tpl").html());

1. Modify main.js as follows:
   * Remove showHomeView()
   * Remove findByName()
   * Modify the initialize function() as follows:
   * initialize: function() {  
      var self = this;  
      this.store = new MemoryStore(function() {  
      $('body').html(new HomeView(self.store).render().el);  
      });  
     }

## Step 5: Scrolling

## Step 6: Routing / Multiple Views

1. In index.html, define a template defined as follows:

* <script id="employee-tpl" type="text/x-handlebars-template">  
   <div class='header'><a href='#' class="button header-button header-button-left">Back</a><h1>Details</h1></div>  
   <div class='details'>  
   <img id='image' src='../img/{{firstName}}\_{{lastName}}.jpg' style="float:left;margin:10px;"/>  
   <h1>{{firstName}} {{lastName}}</h1>  
   <h2>{{title}}</h2>  
   <ul>  
   <li><a href="tel:{{officePhone}}">Call Office<br/>{{officePhone}}</a></li>  
   <li><a href="tel:{{cellPhone}}">Call Cell<br/>{{cellPhone}}</a></li>  
   <li><a href="sms:{{cellPhone}}">SMS<br/>{{cellPhone}}</a></li>  
   </ul>  
   </div>  
  </script>

1. Create a file called EmployeeView.js in work/js, and define an EmployeeView class implemented as follows:

* var EmployeeView = function(employee) {  
    
   this.render = function() {  
   this.el.html(EmployeeView.template(this.employee));  
   return this;  
   };  
    
   this.employee = employee;  
    
   this.el = $('<div/>')  
    
   }  
  EmployeeView.template = Handlebars.compile($("#employee-tpl").html());

1. In the app’s initialize(), define a regular expression that matches employee details urls.

* this.detailsURL = /^#employees\/(\d{1,})/;

1. In the app’s registerEvents() function, add an event listener to listen to URL hash map changes:

* $(window).on('hashchange', $.proxy(this.route, this));

1. In the app object, define a route() function implemented as follows:

* route: function() {  
   var hash = window.location.hash;  
   if (!hash) {  
   $('body').html(new HomeView(this.store).render().el);  
   return;  
   }  
   var match = hash.match(app.detailsURL);  
   if (match) {  
   this.store.findById(Number(match[1]), function(employee) {  
   $('body').html(new EmployeeView(employee).render().el);  
   });  
   }  
  }

1. Test the application.

## Step 7: Using the Location API

1. In index.html, add the following list items to the employee template:

* <li><a href="#" class="add-location-btn">Add Location</a></li>

1. In EmployeeView, register an event listener for the click event of the Add Location list item:

* this.el.on('click', '.add-location-btn', this.addLocation);

1. In EmployeeView, define the addLocation event handler as follows:

* this.addLocation = function(event) {  
   event.preventDefault();  
   console.log('addLocation');  
   navigator.geolocation.getCurrentPosition(  
   function(position) {  
   $('.location', this.el).html(position.coords.latitude + ',' +position.coords.longitude);  
   },  
   function() {  
   alert('Error getting location');  
   });  
   return false;  
  }

1. Test the Application

## Step 8: Using the Contacts API

1. In index.html, add the following list items to the employee template:

* <li><a href="#" class="add-contact-btn">Add to Contacts</a></li>

1. In EmployeeView, register an event listener for the click event of the Add to Contacts list item:

* this.el.on('click', '.add-contact-btn', this.addToContacts);

1. In EmployeeView, define the addToContacts event handler as follows:

* this.addToContacts = function(event) {  
   event.preventDefault();  
   console.log('addToContacts');  
   if (!navigator.contacts) {  
   app.showAlert("Contacts API not supported", "Error");  
   return;  
   }  
   var contact = navigator.contacts.create();  
   contact.name = {givenName: app.currentEmployee.firstName, familyName: app.currentEmployee.lastName};  
   var phoneNumbers = [];  
   phoneNumbers[0] = new ContactField('work', app.currentEmployee.officePhone, false);  
   phoneNumbers[1] = new ContactField('mobile', app.currentEmployee.cellPhone, true); // preferred number  
   contact.phoneNumbers = phoneNumbers;  
   contact.save();  
   return false;  
  }

1. Test the Application

## Step 9: Using the Camera API

1. In index.html, add the following list items to the employee template:

* <li><a href="#" class="change-pic-btn">Change Picture</a></li>

1. In EmployeeView, register an event listener for the click event of the Change Picture list item:

* this.el.on('click', '.change-pic-btn', this.changePicture);

1. In EmployeeView, define the changePicture event handler as follows:

* this.changePicture = function(event) {  
   event.preventDefault();  
   console.log('changePicture');  
   if (!navigator.camera) {  
   app.showAlert("Camera API not supported", "Error");  
   return;  
   }  
   var options = { quality: 50,  
   destinationType: Camera.DestinationType.DATA\_URL,  
   sourceType: 1, // 0:Photo Library, 1=Camera, 2=Saved Photo Album  
   encodingType: 0 // 0=JPG 1=PNG  
   };  
    
   navigator.camera.getPicture(  
   function(imageData) {  
   $('#image').attr('src', "data:image/jpeg;base64," + imageData);  
   },  
   function() {  
   alert('Error taking picture');  
   },  
   options);  
    
   return false;  
  };

1. Test the Application

## Step 10: CSS Transitions

1. Inside the app object, define a slidePage() function implemented as follows:

* slidePage: function(page) {  
    
   var currentPageDest,  
   self = this;  
    
   // If there is no current page (app just started) -> No transition: Position new page in the view port  
   if (!this.currentPage) {  
   $(page.el).attr('class', 'page stage-center');  
   $('body').append(page.el);  
   this.currentPage = page;  
   return;  
   }  
    
   // Cleaning up: remove old pages that were moved out of the viewport  
   $('.stage-right, .stage-left').not('.homePage').remove();  
    
   if (page === app.homePage) {  
   // Always apply a Back transition (slide from left) when we go back to the search page  
   $(page.el).attr('class', 'page stage-left');  
   currentPageDest = "stage-right";  
   } else {  
   // Forward transition (slide from right)  
   $(page.el).attr('class', 'page stage-right');  
   currentPageDest = "stage-left";  
   }  
    
   $('body').append(page.el);  
    
   // Wait until the new page has been added to the DOM...  
   setTimeout(function() {  
   // Slide out the current page: If new page slides from the right -> slide current page to the left, and vice versa  
   $(self.currentPage.el).attr('class', 'page transition ' + currentPageDest);  
   // Slide in the new page  
   $(page.el).attr('class', 'page stage-center transition');  
   self.currentPage = page;  
   });  
    
  }

1. Modify the route() function as follows:

* route: function() {  
   var self = this;  
   var hash = window.location.hash;  
   if (!hash) {  
   if (this.homePage) {  
   this.slidePage(this.homePage);  
   } else {  
   this.homePage = new HomeView(this.store).render();  
   this.slidePage(this.homePage);  
   }  
   return;  
   }  
   var match = hash.match(this.detailsURL);  
   if (match) {  
   this.store.findById(Number(match[1]), function(employee) {  
   self.slidePage(new EmployeeView(employee).render());  
   });  
   }  
  }

More topics:

* Conditional css
* Touch vs click
* Viewport
* Testing
* External template loader
* Plugins