Mobile Visualforce

Force.com Platform Workshop

Mobile Visualforce

Table of Contents

Prerequisites

Force.com

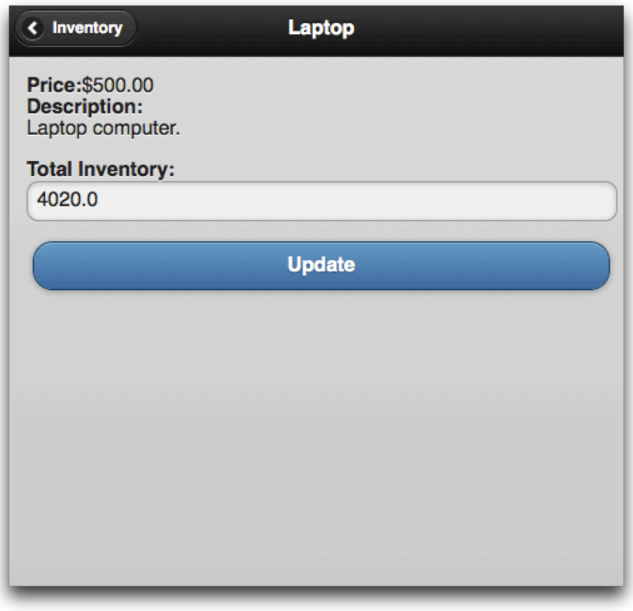
Sign up for a free Force.com Developer Edition (DE) organization (org):
http://www.developerforce.com/events/regular/registration.php

Data Model

This tutorial assumes you have created the Warehouse data model. If you have not, you can install it from this link:
https://login.salesforce.com/packaging/installPackage.apexp?p0=04tE0000000Pzdj

Developing Mobile Apps with Visualforce

The basic UI that is automatically built with the objects you create in Force.com may not suffice for all situations. You might want specific branding, or perhaps a mobile interface. For needs such as this, you can use Force.com's Visualforce technology. This tutorial teaches you how to build a hosted mobile app using Visualforce combined with open source technologies such as jQuery Mobile. The end result is a mobile app, running on the salesforce.com infrastructure.



Price:$500.00
Description:
Laptop computer.

Total Inventory:
4020.0

Step 1: Enable Visualforce Development Mode

If you previously enabled Visualforce Development Mode, you may skip this step.

Development Mode embeds a Visualforce page editor in your browser. It lets you see code and preview the page at the same time. Development Mode also adds an Apex editor for editing controllers and extensions. To enable Development Mode:

1. Click **Your Name | Setup | My Personal Information | Personal Information**.
2. Click **Edit**.
3. Select the **Development Mode** checkbox and click **Save**.

Step 2: Create a Visualforce Page

In this step, you create a Visualforce page that is an inventory count sheet.

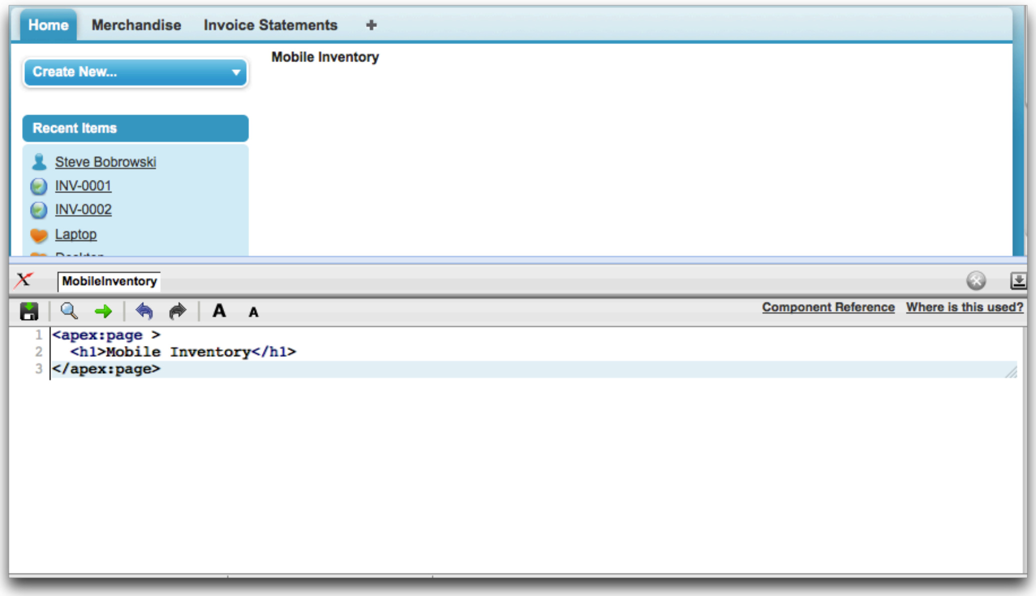
1. In your browser, append `/apex/MobileInventory` to the URL for your Salesforce instance. For example, if your Salesforce instance is `https://na1.salesforce.com`, enter:

   ```
   https://na1.salesforce.com/apex/MobileInventory
   ```

2. You should see an error message: *Page MobileInventory does not exist*. Click the **Create Page MobileInventory** link to create the new Visualforce page.
3. When development mode is enabled, you can edit the page directly by clicking on **MobileInventory** in the lower left corner.
4. Replace the default text with some real content. Change the contents of the `<h1>` tag to `Mobile Inventory`, and remove the comments. The code for the page should now look like this:

   ```
   <apex:page >
     <h1>Mobile Inventory</h1>
   </apex:page>
   ```

5. Click the **Save** icon at the top of the Page Editor. The page reloads to reflect your changes.



```
<apex:page >
  <h1>Mobile Inventory</h1>
</apex:page>
```

Tell Me More

If you are already familiar with Web development, Visualforce should be easy to learn because it simply extends the power of standard page markup. Pages can include Visualforce components like `<apex:page>` to define specific functionality (in this case, the overall structure of the page itself), along with standard HTML tags such as `<h1>`.

Step 3: Add jQuery and jQuery Mobile

Visualforce lets you utilize standard web technologies, which means that you can leverage existing JavaScript and CSS frameworks to make your work easier. Since you're creating a mobile friendly interface, utilizing the popular jQuery and jQuery libraries can help you build your UI smartphone-ready with very little effort. You just need to update Visualforce to reference the files, in this case from a globally hosted CDN system.

1. If you don't have the page editor open, append `/apex/MobileInventory` to the URL for your Salesforce instance as you did earlier.
2. Modify the attributes of the `<apex:page>` tag to remove the standard style sheet, header, and sidebar by entering the following code.

   ```
   <apex:page standardStylesheets="false" showHeader="false" sidebar="false">
   ```

3. Reference the jQuery resources you need from by adding the following lines just under the `<apex:page>` tag.

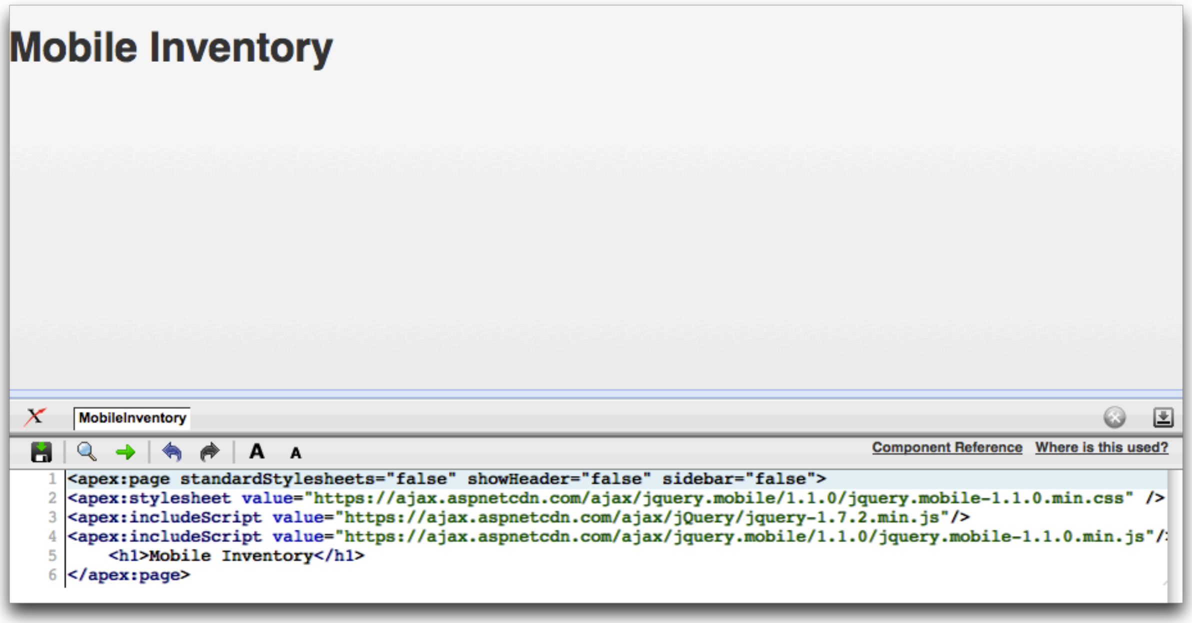
   ```
   <apex:stylesheet value="https://ajax.aspnetcdn.com/ajax/jquery.mobile/1.1.0/
   jquery.mobile-1.1.0.min.css" />
   <apex:includeScript value="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-
   1.7.2.min.js"/>
   <apex:includeScript value="https://ajax.aspnetcdn.com/ajax/jquery.mobile/1.1.0/
   jquery.mobile-1.1.0.min.js"/>
   ```

4. Verify that your code looks like the following:

   ```
   <apex:page standardStylesheets="false" showHeader="false" sidebar="false">
   <apex:stylesheet value="https://ajax.aspnetcdn.com/ajax/jquery.mobile/1.1.0/
   jquery.mobile-1.1.0.min.css" />
   <apex:includeScript value="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-
   1.7.2.min.js"/>
   <apex:includeScript value="https://ajax.aspnetcdn.com/ajax/jquery.mobile/1.1.0/
   jquery.mobile-1.1.0.min.js"/>
         <h1>Mobile Inventory</h1>
   </apex:page>
   ```

5. Click the **Save** icon at the top of the Page Editor.

Note how the page looks very different now that you are using a style sheet. The title is in a different font and location, and the standard header and sidebar are no longer present.



```
<apex:page standardStylesheets="false" showHeader="false" sidebar="false">
<apex:stylesheet value="https://ajax.aspnetcdn.com/ajax/jquery.mobile/1.1.0/jquery.mobile-1.1.0.min.css" />
<apex:includeScript value="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.7.2.min.js"/>
<apex:includeScript value="https://ajax.aspnetcdn.com/ajax/jquery.mobile/1.1.0/jquery.mobile-1.1.0.min.js"/
    <h1>Mobile Inventory</h1>
</apex:page>
```

Tell Me More
The `apex:includeScript` and `apex:stylesheet` in this example demonstrates how simple it is to use externally hosted JavaScript and CSS files. However, the Force.com platform can also host such files as well. You can upload zips as *static resources* and then build pages that reference components within your static resources.

Step 4: Add a Controller to the Page

Visualforce's Model-View-Controller design pattern makes it easy to separate the view from the underlying database and logic. In Force.com's MVC architecture, the controller (typically an Apex class) serves as an intermediary between the view (the Visualforce page) and the model (the object in the Force.com database). For example, the controller might contain the logic that executes when a user clicks a button to retrieve data that the view can then display.

By default, all Force.com objects have standard controllers that you can use to interact with the data associated with the object. So, in many cases, you don't need to write the code for the controller yourself. You can extend the standard controllers to add new functionality, or create custom controllers from scratch. In this tutorial, you use a default controller.

1. If the Page Editor isn't open on your Visualforce page, click **Page Editor** to edit the page.
2. Modify your code to enable the *Merchandise__c* standard controller by editing the first `<apex:page>` tag. The editor ignores whitespace, so you can enter the text on a new line.

   ```
   <apex:page standardStylesheets="false" showHeader="false" sidebar="false"
   standardController="Merchandise__c">
   ```

3. Next, add the standard list controller definition (see *Tell Me More* below).

```
<apex:page standardStylesheets="false" showHeader="false" sidebar="false"
standardController="Merchandise__c" recordSetVar="products">
```

4. Click the **Save** icon at the top of the Page Editor.

You won't notice any change on the page. However, because you've indicated that the page should use a controller, and defined the variable *products*, the variable will be available to you in the body of the page and it will represent a list of *Merchandise* records.

Tell Me More
The *recordSetVar* attribute enables a standard list controller, which provides additional controller support for listing a number of records with pagination. You set it to *products*. This creates a new variable, *products*, that contains the set of records that the Visualforce page can then display.

Also note the reference to *Merchandise__c* -- you previously created a *Merchandise* custom object and the "__c" portion denotes that as a custom object when referencing it from Visualforce, Apex, or API's. This differentiates the object from a standard object such as Contacts or Accounts.

Step 5: Add a List View with jQuery Mobile

Visualforce's MVC model lets you directly pull records into a list without requiring any additional logic. To create our mobile interface with ease, add some HTML that leverages the CSS with jQuery Mobile to create an HTML page that is friendly for mobile devices.

1. Edit the MobileInventory page to replace `<h1>Mobile Inventory</h1>` with this HTML body:

```
<body>
<div data-role="page" data-theme="b" id="mainpage">
  <div data-role="header">
    <h1>Warehouse Inventory</h1>
  </div>
  <div data-role="content">
    <ul data-inset="true" data-role="listview" data-theme="c" data-dividertheme="c">
    </ul>
  </div>
</div>
</body>
```

2. In between `<ul>` and `</ul>`, place an `<apex:repeat>` component which will loop through the current set of Merchandise records to display a list, like so:

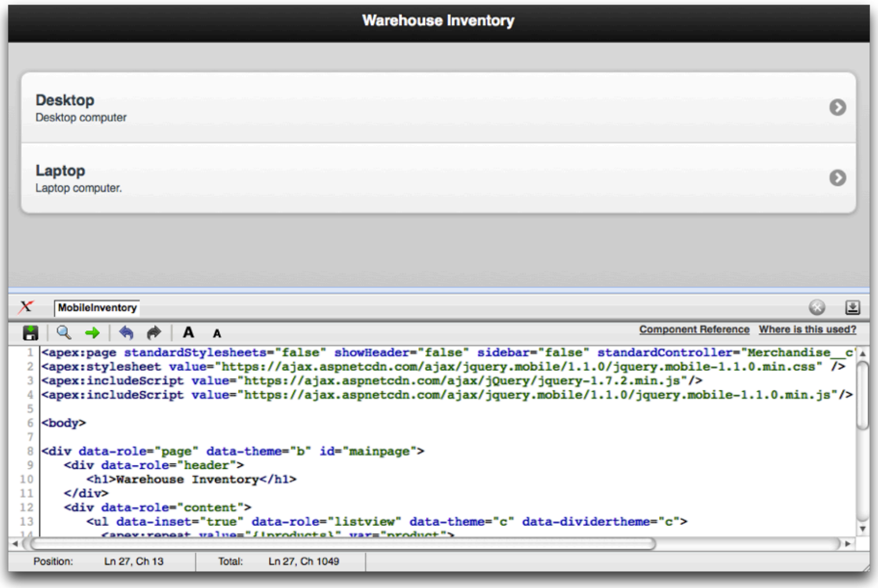
```
<apex:repeat value="{!products}" var="product">
  <li>
    <a href="#detailpage{!product.Id}">
      <h3><apex:outputField value="{!product.Name}"/></h3>
      <p><apex:outputField value="{!product.Description__c}"/></p>
    </a>
  </li>
</apex:repeat>
```

3. Your resulting page should look like this:

```
<apex:page standardStylesheets="false" showHeader="false" sidebar="false"
standardController="Merchandise__c" recordSetVar="products">
<apex:stylesheet value="https://ajax.aspnetcdn.com/ajax/jquery.mobile/1.1.0/
jquery.mobile-1.1.0.min.css" />
<apex:includeScript value="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-
1.7.2.min.js"/>
<apex:includeScript value="https://ajax.aspnetcdn.com/ajax/jquery.mobile/1.1.0/
jquery.mobile-1.1.0.min.js"/>

<body>

<div data-role="page" data-theme="b" id="mainpage">
   <div data-role="header">
      <h1>Warehouse Inventory</h1>
   </div>
   <div data-role="content">
      <ul data-inset="true" data-role="listview" data-theme="c" data-dividertheme="c">
        <apex:repeat value="{!products}" var="product">
         <li>
             <a href="#detailpage{!product.Id}">
               <h3><apex:outputField value="{!product.Name}"/></h3>
               <p><apex:outputField value="{!product.Description__c}"/></p>
             </a>
         </li>
        </apex:repeat>
      </ul>
   </div>
</div>

</body>
</apex:page>
```

4. Click **Save**.

The page now uses the jQuery Mobile CSS to create a list of records. Each row displays the item's name, description, and current inventory count.



```
<apex:page standardStylesheets="false" showHeader="false" sidebar="false" standardController="Merchandise__c
<apex:stylesheet value="https://ajax.aspnetcdn.com/ajax/jquery.mobile/1.1.0/jquery.mobile-1.1.0.min.css" />
<apex:includeScript value="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.7.2.min.js"/>
<apex:includeScript value="https://ajax.aspnetcdn.com/ajax/jquery.mobile/1.1.0/jquery.mobile-1.1.0.min.js"/>

<body>

<div data-role="page" data-theme="b" id="mainpage">
   <div data-role="header">
      <h1>Warehouse Inventory</h1>
   </div>
   <div data-role="content">
      <ul data-inset="true" data-role="listview" data-theme="c" data-dividertheme="c">
```

Position: Ln 27, Ch 13 Total: Ln 27, Ch 1049

Tell Me More
jQuery Mobile lets you use simple `<div>` elements to define multiple application pages using just one Visualforce/HTML page. jQuery Mobile's approach reduces the amount of traffic between the client and the server, which is perfect for mobile application development. Additionally, jQuery Mobile's approach automatically styles lists in a mobile-friendly way.

For this tutorial, you are simply outputting all of the products in *Merchandise__c* on the page. This works here as the *Warehouse* application is unlikely to have hundreds of rows of data. However, if you are working with larger data sets, you can exert finer control by creating a custom controller and adding a specific *SOQL* statement. For example, SOQL supports an *OFFSET* filter that makes it easier to get the next 100 records and use pagination in your app.

Step 6: Add Detail Pages

This step shows you how to add functionality so that a user can manually update the current inventory when it changes. To do that, you'll add the ability to drill down into each product's details. Using the same `<apex:repeat>` component you used to create the list view, add a series of `<div>` elements to the page to describe the items in the warehouse.

1. Below the last `<div>` element, and just before the `</body>` tag, add the following Visualforce component:

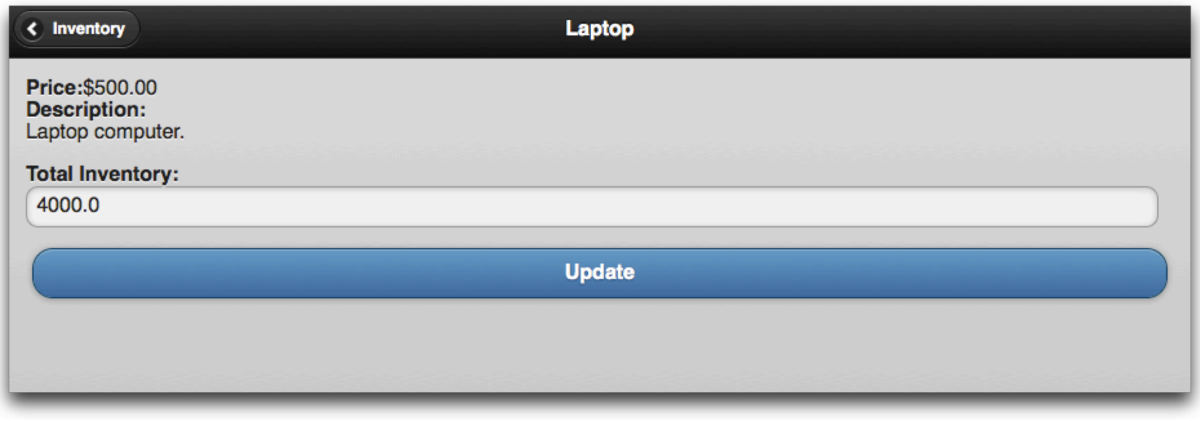
```
<apex:repeat value="{!products}" var="product">
<div data-role="page" data-theme="b" id="detailpage{!product.Id}">
   <div data-role="header">
      <a href='#mainpage' id="backInventory" class='ui-btn-left' data-
icon='arrow-l'>Inventory</a>
      <h1><apex:outputField value="{!product.Name}"/></h1>
   </div>
   <div data-role="content">
       <strong>Price:</strong><apex:outputField value="{!product.Price__c}" /
><br />
       <strong>Description:</strong><BR />
       <apex:outputField value="{!product.Description__c}" />
       <p>
       <strong>Total Inventory:</strong><input type="text" id="inventory{!
product.Id}" value="{!product.Total_Inventory__c}" />
       </p>
<button id="update{!product.Id}" data-id="{!product.Id}" class="updateBtn"
>Update</button>
   </div>
</div>
</apex:repeat>
```

2. The resulting page should look like this:

```
<apex:page standardStylesheets="false" showHeader="false" sidebar="false"
standardController="Merchandise__c" recordSetVar="products">
<apex:stylesheet value="https://ajax.aspnetcdn.com/ajax/jquery.mobile/1.1.0/
jquery.mobile-1.1.0.min.css" />
<apex:includeScript value="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-
1.7.2.min.js"/>
<apex:includeScript value="https://ajax.aspnetcdn.com/ajax/jquery.mobile/1.1.0/
```

```
jquery.mobile-1.1.0.min.js"/>

<body>

<div data-role="page" data-theme="b" id="mainpage">
   <div data-role="header">
      <h1>Warehouse Inventory</h1>
   </div>
   <div data-role="content">
      <ul data-inset="true" data-role="listview" data-theme="c" data-
dividertheme="c">
<apex:repeat value="{!products}" var="product">
         <li>
             <a href="#detailpage{!product.Id}"><h3><apex:outputField value="{!
product.Name}"/></h3>
             <p><apex:outputField value="{!product.Description__c}"/></p></a>
         </li>
</apex:repeat>
      </ul>
   </div>
</div>
<apex:repeat value="{!products}" var="product">
<div data-role="page" data-theme="b" id="detailpage{!product.Id}">
   <div data-role="header">
      <a href='#mainpage' id="backInventory" class='ui-btn-left' data-
icon='arrow-l'>Inventory</a>
      <h1><apex:outputField value="{!product.Name}"/></h1>
   </div>
   <div data-role="content">
       <strong>Price:</strong><apex:outputField value="{!product.Price__c}" /
><br />
       <strong>Description:</strong><BR />
       <apex:outputField value="{!product.Description__c}" />
       <p>
       <strong>Total Inventory:</strong><input type="text" id="inventory{!
product.Id}" value="{!product.Total_Inventory__c}" />
       </p>
<button id="update{!product.Id}" data-id="{!product.Id}" class="updateBtn"
>Update</button>
   </div>
</div>
</apex:repeat>

</body>
</apex:page>
```

3. Click **Save**.
4. When the page reloads, click on any item on the list.
5. Click the **Inventory** button in the upper left to return to the main list.



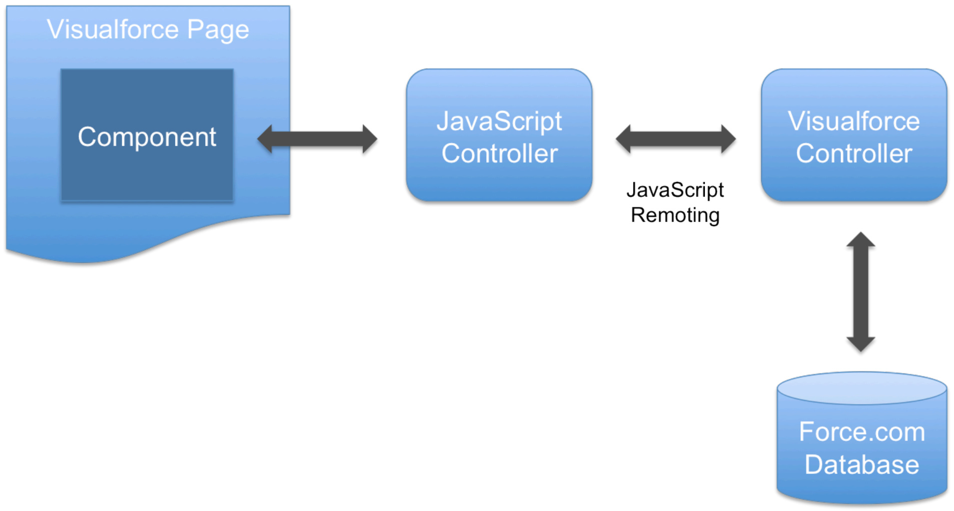
Price:$500.00
Description:
Laptop computer.
Total Inventory:
4000.0

Tell Me More

The page leverages apex:repeat to create a series of HTML div's that represent the detail pages. jQuery Mobile properly styles and controls these pages -- by utilizing the framework, you've created a navigable index of your merchandise that is mobile-friendly.

Step 7: Add an Apex Extension for Updating Records

The interface lets a user browse through the current inventory, but there's no way to update the record itself. There are many ways to implement data manipulation with Apex and Visualforce. The most convenient way is to use a standard controller and the `<apex:form>` component because you can modify the record reference in the ID of the page URL without adding custom Apex logic; however, there's some overhead with this approach, and it doesn't align with our use of JavaScript and jQuery Mobile. Instead, this tutorial teaches you how to use a feature of Apex and Visualforce called *JavaScript Remoting*. This lesson shows how to use JavaScript in a Visualforce page that references Apex logic in a custom extension to the standard controller to manipulate a database record.



1. Update the `<apex:page>` component at the top of the page to include a reference to an Apex extension by modifying the tag to look like this:

```
<apex:page standardStylesheets="false" showHeader="false" sidebar="false"
standardController="Merchandise__c" extensions="MobileInventoryExtension"
recordSetVar="products">
```

2. Click **Save**. When you see an error indicating that the *MobileInventoryExtension* does not exist, click the link that reads: **Create Apex class 'public with sharing class MobileInventoryExtension'**.
3. When you see an error, click **Create Apex Method ...**
4. Look for **MobileInventoryExtension** beside *MobileInventory* at the top of the editor. Click **MobileInventoryExtension**.
5. Update the code so that it looks like this:

```
public class MobileInventoryExtension {

    //Constructors.  Needed to use as an extension.
    public MobileInventoryExtension(ApexPages.StandardController c) {}
    public MobileInventoryExtension(ApexPages.StandardSetController c) {}

    //Remote Action function allows JavaScript to call Apex directly
    @RemoteAction
    public static String updateMerchandiseItem(String productId,
Integer newInventory) {
List<Merchandise__c> m = [SELECT Id, Name, Price__c, Total_Inventory__c,
Description__c from Merchandise__c WHERE Id =: productId LIMIT 1];
            if(m.size() > 0) {
                m[0].Total_Inventory__c = newInventory;
                try {
                    update m[0];
                    return 'Item Updated';
                } catch (Exception e) {
                    return e.getMessage();
                }
            }
            else {
                return 'No item found with that ID';
            }
        }
}
```

6. Click **Save**.

Tell Me More

Visualforce controllers come in three flavors: standard, custom, and extensions.

- *Standard controllers* are automatically available and offer baseline functionality for creating, editing, and deleting records.
- *Custom controllers* are custom logic that you code.
- A *controller extension* is also custom logic, but you can associate it with standard or custom controllers. This object-oriented approach enables great portability and maintenance of code.

In this tutorial, you are using the standard controller for the *Merchandise* object with the recordsetvar attribute to enable automatic access to a list of data. The

MobileInventoryExtension Apex class provides an *updateMerchandiseItem* method that the Visualforce page can call via JavaScript Remoting.

Step 8: Add JavaScript to Trigger the Update

The mobile application now has a functional model (querying the *Merchandise* records), a view (the HTML jQuery Mobile interface), and a controller (the custom Apex extension), but still needs functionality to update records. With JavaScript Remoting, the Apex code added in Step 7 is visible directly to JavaScript and now just requires some code to trigger the logic itself.

1. If you are still editing *MobileInventoryExtension* code, click **MobileInventory** to return to the Visualforce page.
2. Above the `<body>` element, add the following `<head>` element:

```
<head>
  <title>Mobile Inventory</title>
   <meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1.0" />
  <script>
       var j$ = jQuery.noConflict();
      var dataChanged = false;

      j$(document).ready(function() {

          j$(".updateBtn").click(function() {
              var id = j$(this).attr('data-id');
              var inventory = parseInt(j$("#inventory"+id).val());
              j$.mobile.showPageLoadingMsg();
MobileInventoryExtension.updateMerchandiseItem(id,inventory,handleUpdate);
          });
     });


      function handleUpdate(result,event) {
          alert(result);
          if(result == 'Item Updated') { dataChanged = true; }
          j$.mobile.hidePageLoadingMsg();
      }

 </script>
</head>
```

3. Click **Save**.
4. When the page reloads, click on any item on the list.
5. In the *Total Inventory* field, change the number for the record.
6. Click **Update**.
7. After a brief loading animation, you should see an alert display *Item Updated*.

You can now update Merchandise records via the mobile app interface.

Summary

In this tutorial, you've used many of the core technologies of the Force.com platform.

Visualforce enables you to easily create custom interfaces to interact with your data in the cloud. In this case, you used jQuery Mobile to make a mobile-friendly page, only one of many solutions Visualforce can provide.

Apex lets you provide custom business logic that you can expose via Visualforce. And Apex is not limited to working with Visualforce -- it can interact with page layouts and can even be exposed as an API.

Additional Reading

You have created a mobile friendly page in the cloud using Apex and Visualforce - but what about having this page as an application on a mobile device? The Force.com platform offers the Mobile SDK which provides an easy to use framework for building native applications as well as HTML5 based applications for iOS and Android. Check out the Mobile SDK home page for more:

http://developer.force.com/mobilesdk

And check out the Workbooks home page for the Mobile SDK workbook as well:

http://developer.force.com/workbooks