CS220 Human-Computer Interaction
Spring 2015

Lab 4: Forms and Geolocation

In this lab we will continue to develop our web app. You will learn to:

- Create and style HTML 5 forms
- Use JavaScript to validate your forms
- Use the Geolocation API to retrieve location information
- Use the Google Maps API
- Connect the Geolocation API and the Google Maps API

Task

Create a registration form with validation similar to the following form (use iOS mobile device to open):
http://cs.wellesley.edu/~hci/mobile220/Register.html

Use the GPS and Google Maps API for geolocation functionality (use iOS mobile device to open):
http://cs.wellesley.edu/~hci/mobile220/Location.html

Step-by-Step Tutorial

Step 1: Create a new HTML 5 file called Register.html.
Create a new CSS file called forms.css

Before we start designing our registration form, it is worthwhile thinking more broadly about the design of sign-up forms. Is there anything to design in a minimalistic sign-up form? Isn't it too simple to focus on? Read this thoughtful discussion of the UX design of sign-up forms.

Step 2: Working in **Register.html**, create an empty registration form:

```
<!DOCTYPE html>
<html>
<head>
	<meta name="apple-mobile-web-app-capable" content="yes">
	<meta name="apple-mobile-webapp-status-bar-style" content="black">
	<meta name="viewport" content="width=device-width, user-scalable=no" >
	<title>Register</title>
	<link rel="stylesheet" href="forms.css">
	<link rel="stylesheet" href="iphone.css">
</head>
<body>
	<header class="secondary"> Sign Up! </header>
	<button class="back" onclick="location.href='index.html';">Back</button>
```

```
      <div class="info">
      <p>Register now to take advantage of this site. </p>
      <form method="post" id="register" onSubmit=”return validateForm();”>
             <!-- FORM FIELDS GO HERE -->
             <br />
             <input type="submit" value="Sign up">
             <a href="index.html">Nevermind</a>
      </form>
</div>
<script src="stay_standalone.js"></script>
</body>
</html>
```

Step 3: Now add form fields (study and then copy the code below) to your document inside the `form` tags (in the HTML code there is a comment that specifies where to add the code).

HTML5 introduces a number of new input types. These new input types give hints to the browser about what type of keyboard layout to display for on-screen keyboards. This tutorial provides a detailed summary of all new HTML 5 input types and their support by various browsers.

Note that in the code below we add a label and an error message for each form field. Next we will style the form so that the error message is hidden and the labels appear above the form fields.

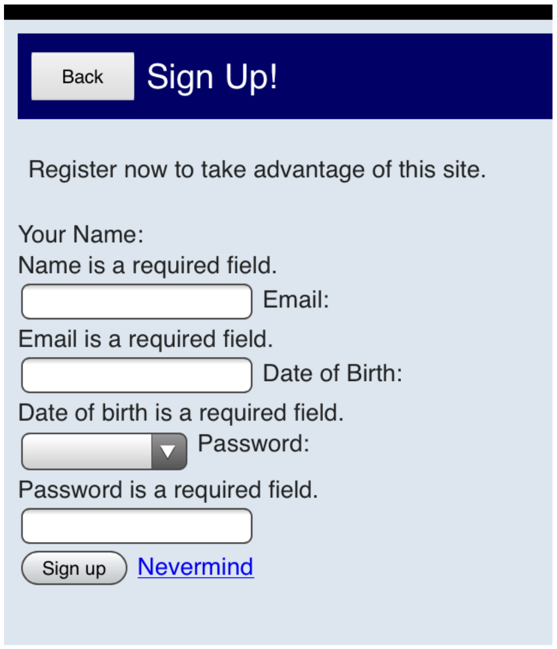
```
<label for="name">Your Name:</label>
 <div class="error" id="name-error">Name is a required field.</div>
 <input type="text" name="name" id="name">

 <label for="email">Email:</label>
 <div class="error" id="email-error">Email is a required field.</div>
<input type="email" name="email" id="email">

 <label for="dob">Date of Birth:</label>
 <div class="error" id="dob-error">Date of birth is a required field.</div>
<input type="date" name="dob" id="dob">

<label for="pwd">Password:</label>
<div class="error" id="pwd-error">Password is a required field.</div>
<input type="password" name="pwd" id="pwd">
```

Checkpoint: For now, here’s what your **Register.html** file should look like in a browser window. Test your form and observe the different on-device keyboards that are displayed for each input type.



Register now to take advantage of this site.

Your Name:
Name is a required field.
Email:
Email is a required field.
Date of Birth:
Date of birth is a required field.
Password:
Password is a required field.

Step 4: Now let's add some style. Switch to **forms.css,** and add a few rules.

Note that since we linked our Register.html file to iphone.css all the global rules defined in iphone.css are applied in Register.html. In addition, in forms.css we add rules specific to forms.

First lets add a rule to make the labels appear above the input fields.

```
label {
      display: block;
}
```

Step 5: And, adjust the size of the input fields. Use `height`, `font-size`, and `margins` as needed. Write your rules within the `input` selector.

```
input {





}
```

Step 6: Finally, style the form itself as well as the surrounding <div> tag. Add your definitions within the .info and form selectors. Use `padding`, `margins`, `border` etc. as needed

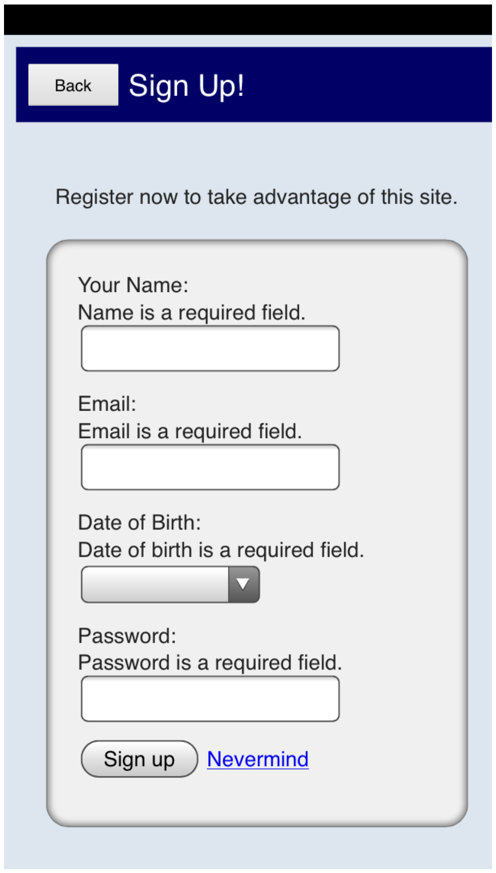
```
.info {

}

form {


}
```

Checkpoint: Here's what things could look like now:


Back
Sign Up!

Register now to take advantage of this site.

Your Name:
Name is a required field.

Email:
Email is a required field.

Date of Birth:
Date of birth is a required field.

Password:
Password is a required field.

Sign up
Nevermind

Step 7: Ok. Time to style the error messages. We want these to float to the right of their corresponding fields, which we can do with the **position: absolute** property.

```
.error {
      background-color: LightYellow;
      color: DarkRed;
      width: 160px;
      padding: 10px;
      border: 1px solid gray;
      position: absolute;
      left: 120px;
      box-shadow: 3px 3px 3px #999999;
      border-radius: 7px;
}
```

Take a look in the browser. If all goes well, these should look something like post-it notes stuck on the right side of the form.

Step 8: When things look right, add a rule that changes the **visibility** property in .error to **hidden.** We'll use JavaScript to make these error messages appear when they're needed.

Step 9: Finally, we need to fix the Submit button. Here we're using a special CSS attribute selector to apply a rule only to input elements with the "submit" type. We will add an image to the button (see below). Also add rules for `width, height, margings, and font-size: 16px;`

```
input[type='submit'] {

      /* Add more rules here */

      -webkit-border-image: url("blueButton.png") 0 14 0 14 stretch;
      }
```

Form Validation

One exciting aspect of HTML5 forms is automatic input validation. HTML5 offers automatic validation of required fields, and of valid email and web addresses. HTML 5 also validates numbers based on min and max attributes. **Sadly, automatic validation is not yet supported by Safari**. Thereby, we will use JavaScript to implement form validation. To learn more about HTML 5 form validation read this tutorial.

Optional step: Create a new HTML 5 file with a simple form, which utilizes the HTML 5 automatic validation features. Test the form using Opera and Chrome.

Studying our **Register.html**, you'll notice that there's a JavaScript function call, **validateForm()**, embedded in the form element. This function is called through the **onsubmit** event, which means that when the user presses the Submit button, this function is invoked. `validateForm` returns true or false. The form will be submitted only if the function returns true.

Step 10: Create a new JavaScript file called `forms.js`. Add this file to Register.html (paste the following line towards the end of the <body> section):

```
<script src="forms.js"></script>
```

Step 11:

Study the following function and then add it to **forms.js**:

What helper functions does the function use?
When does it returns true?

```
function validateForm()
{
      var result=true;

      if(!validateField("pwd"))
      result=false;

      if(!validateDOB())
```

```
                result=false;

        if(!validateField("email"))
                        result=false;

        if(!validateField("name"))
                        result=false;

        return result;
 }
```

Step 12: Next we define a helper function to check that a value for a field has been provided.

```
function validateField (id) {

        // Get the input element with the id passed as argument
        var el=document.getElementById(id);
        // Get the user-entered value for that input field
        var value=el.value;
        // Get the corresponding error message element
         var err=document.getElementById(id+"-error");
        // If the value field is empty show an error message
        if (value == "") {
                err.style.visibility = "visible";
                el.focus();
                return false;
         }
        else {
                err.style.visibility = "hidden";
                return true;  }
}
```

Step 13: We also define a function that validates the date of birth entered: users must be at least 15 for signing up for this website.

```
function validateDOB()
{
        //Get the dob field
        var dob= document.getElementById("dob");
        // If the value field is empty show an error message
        if(dob.value=="")
        {
                var err=document.getElementById("dob-error");
                err.style.visibility = "visible";
                //set the focus on the dob field
                dob.focus();
                return false;
        }

        else return true;
}
```

Step 14 (optional): Add your own code to this function to limit accepted users to age 15 and above.

Checkpoint: Try it out. If you leave any fields blank, you should see an error message when you press Submit. However, the error message is still there after you entered a value. We will fix it now.

Step 15: To hide an error message after a value was entered into that field we will write a function that will be called when the event `onblur` occures. The **onblur** event occurs when an object loses focus – when the user leaves a particular field.

In **Register.html** add the following event handler to each input element except the dob field (Make sure to change "name" to the actual id of each element):

```
<input type="text" name="name" id="name" onBlur="onBlur('name');">
```

The onblur event of dob input field will invoke the `onBlurDob` function:

```
<input type="date" name="dob" id="dob" onBlur="onBlurDOB();">
```

Step 16: Add the following function to forms.js:

```
function onBlur(id) {
	// Get the input element with the id passed as argument
	var el=document.getElementById(id);
	// Get the user-entered value for that input field
	var value=el.value;
	// Get the corresponding error message element
	var err=document.getElementById(id+"-error");
	//If the value field is not empty hide the error message
	if (value != "")
		err.style.visibility="hidden";
}
```

Step 17: Now add the onBlurDOB function:

Note that if you added code to test the age of your users you will need to modify this function to support your changes.

```
function onBlurDOB()
{
	var dob=document.getElementById("dob");
	// Get the user-entered value for that input field
	var value=dob.value;
	// Get the corresponding error message element
	var err=document.getElementById("dob-error");
	//If the value is not empty hide the error message and reset its content
	if (value != "") {
		err.style.visibility="hidden";
		err.innerHTML="Date of birth is a required field";   }
 }
```

Finally, test your form. When all fields are validated no error messages should be displayed and the form should be submitted.

GPS and Google Maps

Knowing where the users are can add a lot to the user experience. With HTML 5 Geolocation (JavaScript based) API we can easily access location information. The geolocation API returns the coordinates (latitude, longitude) of where the user is.

Since this can compromise user privacy, the position is not available unless the user approves it.

The GPS function that is built into iOS is the `navigator.geolocation` object. There are two methods for getting the GPS:

- `getCurrentPosition` – this will get you the GPS position once
- `watchCurrentPosition` – this will get the GPS position on a timed interval

Step 18: Create a new HTML 5 file, name it Location.html. Link to this page from the Go for a Ride buttom in index.html.

Copy the following content into your new Location.html file:

```
<!DOCTYPE html>
<html>
<head>
        <meta name="apple-mobile-web-app-capable" content="yes">
       <meta name="apple-mobile-webapp-status-bar-style" content="black">
       <meta name="viewport" content="width=device-width, height=device-height,
       initial-scale=1.0, maximum-scale=1.0, target-densityDpi=device-dpi" />
       <title>Location</title>
       <link rel="stylesheet" href="iphone.css">
</head>
<body onload="getGPSLatLng()">
       <header class="secondary"> Sign Up! </header>
       <button class="back" onclick="location.href='index.html';">Back</button>
       <button class="location" id="get_coords" onClick="getGPS()">Get GPS
       Coordinates</button>
       <div id="geolocate">
              <input id="address" type="text" placeholder="Enter an address here">
             <button class="location" id="geo"
             onclick="displayMapByAddress()">Geolocate</button>
             </div>
              <div id="basic_map" style="width:320px;height:240px;"></div>
              <div id="gps_coords">
                     <h1>GPS coordinates below </h1>
              </div>
</body>
</html>
```

Step 19: To style this page, add the following rules to iphone.css:

```
       button.location {
       display: inline;
       -webkit-border-image: url("blueButton.png") 0 14 0 14 stretch;
       width:120px;
       height: 30px;

       }

     #geolocate{
       /* add rules here */
     }
```

Step 20: Next we will use the Geolocation JavaScript API to retrieve the user position and display its position on screen.

In Location.html, add the following <script> element at the end of the <body> section. This <script> element contains variable declarations that will be used for storing location information:

```
<script>
    var updateLocation;
    var lat;
    var lon;
</script>
```

Step 21: Next, we will implement the function getGPS() that is invoked when the button "gps_coords" is clicked. Study the following code and add it to the same <script> section from Step 20.

```
/*This function uses the geolocation.getCurrentPosition method.
The geolocation.getCurrentPosition takes 3 parameters:
a success function,
an error function,
and options in the form of JSON syntax.
This function is invoked when the user click on the get_coords button.
*/

function getGPS()
{
    //The enableAccuracy option provides you with the highest accuracy
but consumes a lot of battery life.
    navigator.geolocation.getCurrentPosition(successGPS,errorGPS,{enab
leHighAccuracy : true});
}

/* This function is invoked by the geolocation.getCurrentPosition
method used in getGPS()
*/
function successGPS(position)
```

```
{
    //store the location latitude and longtitude info
    lat=position.coords.latitude;
    lon=position.coords.longitude;

    //create a new <p> element that displays the current latitude and
longitude, add the new <p> to the div
    var div=document.getElementById("gps_coords");
    var par=document.createElement("p");
    par.innerHTML="lat="+lat+" ,lon="+lon+"<br>";
    div.appendChild(par);

}

/* This function is invoked when there is an error in reading GPS
info*/
function errorGPS()
{
    alert("GPS Error");
}
```

Checkpoint: Try it out. You should see the coordinates of your current location appear when you click the Get GPS Coordinates button.

Step 22: Study and add the following function to your <script>:

```
/*This function provide continuous feedback. It returns GPS
coordinates to the callback function approximately once per second and
stores them in a global variable.
Note that this function consumes a lot of battery life.*/

function watchGPS()
{
    updateLocation=navigator.geolocation.watchPosition(successGPS,
errorGPS, {enableHighAccuracy : true});

}
```

Now, change the onClick function call of `<button class="location" id="get_coords" >` to invoke the function watchGPS().

Checkpoint: Try it out. What happens when you click the Get GPS Coordinates button?

To save battery change the onClick function call of `<button class="location" id="get_coords" >` back to getGPS().

Step 23: Now that we can retrieve location information, we will display the results in a map. To do so we need access to a map service that can use latitude and longitude, like Google Maps.

Add the following <script> to your <body>:

```
<script type="text/javascript"
src="http://maps.googleapis.com/maps/api/js?sensor=false"></script>
```

Step 24: Note that when the <body> onload event occures a function called getGPSLatLng() is invoked. This function is similar to getGPS() but it calls a different success method - `successGPSLatLng(position)`, which initializes a Google Map object centered on the current GPS location.

Study the following code and add it to your <script> section:

```
/*This function is used to retrieve GPS coordinates and pass them to a
function, which in turn initializes a Google Map centered on the
current location of the device.*/
function getGPSLatLng()
{
    navigator.geolocation.getCurrentPosition(successGPSLatLng,errorGPS
,{enableHighAccuracy : true});
}




/*This function initializes a Google Map object centered on the
current location of the device */

function successGPSLatLng(position)
{
    //store current latitude and longitude information
    lat=position.coords.latitude;
    lon=position.coords.longitude;

    //create a new googl.maps.LatLng object
    var latlng=new google.maps.LatLng(lat,lon);

    //set Google Map options
    var myOptions={
     zoom: 15,
     center: latlng,
     mapTypeId: google.maps.MapTypeId.ROADMAP
    }

    //create a new map, pass its <div> contsainer as parameter
    var map= new google.maps.Map(document.getElementById("basic_map"),
myOptions);

    //place a marker with current latitude and longitude on the map
    var marker= new google.maps.Marker({position: latlng, map: map});

}
```

Checkpoint: Test your application. A map should be displayed centered on your current location.

Step 25: Finally, we will use the geocode method of the `Geocoder` object, to find the location information of a particular address.

After the user enters text to `<input id="address" type="text" placeholder="Enter an address here">` and clicks on the Geolocate button, the map will be updated to display and center on the GPS location of the requested address.

Clicking on the Gelocate button invokes the `displayMapByAddress()`function. Study the code of this function and add it within the same <script> in which you implemented the previous functions:

```
/*This function retrieves the address entered to the address field and
uses Google geocoding to find its location (longtitude and latitude,
it then calls the initializeByMap function */

function displayMapByAddress()
{
    var address=document.getElementById("address");

    //create a Geocoder object and use its geocode method. Pass the
address as a parameter using JSON syntax
    var geocoder=new google.maps.Geocoder();

    //geocode methods takes two parameters: an address, and a function
which processes the results. In our case this function calls the
function initializeByAddress.
    geocoder.geocode({'address': address.value},
    function(results,status)
    {
      initializeByAddress(results[0].geometry.location);
    });
}
```

Step 26: Study and add the following function within the same <script>:

```
/*This function initialize a map centerd on the location passed as
parameter*/
function initializeByAddress(location){

    var myOptions={
      zoom: 15,
      center: location,
      mapTypeId: google.maps.MapTypeId.ROADMAP
    }

    var map= new google.maps.Map(document.getElementById("basic_map"),
myOptions);

    var marker= new google.maps.Marker({position: location, map:
map});

}
```

Checkpoint: Test your application. Enter an address and click on Geolocation. The application should update the map to display the requested address.