CS530/338 Developing User Interfaces

Lab 3: 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.

Step-by-Step Tutorial

Step 0: Before you start, login to tux and create a new directory called "Lab3" in your public_html/cs338 directory. Copy all the files from Lab2 into the new Lab3 directory. Grab all the files from Lab3 on BBLearn and save them in the Lab3 project directory you just created.

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 (https://designmodo.com/ux-sign-up-form/).

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" >
                <!-- 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 (http://diveintohtml5.info/forms.html) 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 for each form field. Next we will style the form so that the labels appear above the form fields.

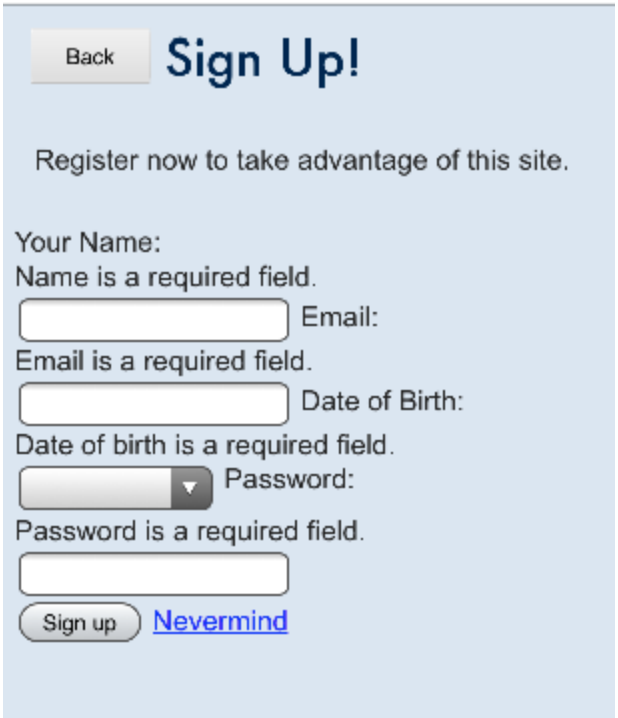
```
<label for="name">Your Name:</label>
<input type="text" name="name" id="name">

<label for="email">Email:</label>
<input type="email" name="email" id="email">

<label for="dob">Date of Birth:</label>
<input type="date" name="dob" id="dob">

<label for="pwd">Password:</label>
<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.



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 let's 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 `form` selector. Use `padding, margins, border` etc. as needed

```
form {


}
```

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

Step 7: 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, margins, 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. **Automatic validation is supported in Safari version 10.1**. To learn more about HTML 5 form validation read this tutorial (http://diveintohtml5.info/forms.html).

Step 8: Next we add a check that a value for a field has been provided.

For each input field in the form, add the `required` tag.

Step 9: Let's add validation that the user is over 13 years old, by specifying a max value in the dob element. You can simply validate that the year is not less than 13 years ago.

Checkpoint: Try it out. If you leave any fields blank, you should see an error message when you press Submit.

Step 9: We will now add styling to the fields, based on the validation and required states. In **forms.css,** add rules for the required, valid and invalid input.

```
input:invalid {
  /* insert your own styles for invalid form input */

}

input:valid {
  /* insert your own styles for valid form input */
}

input:required {
  /* insert your own style to indicate a required form input */
}
```

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 button 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,{enableHighA
ccuracy : 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. (You may have to give permission first).

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="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY"></script>
```

Important note: Google Maps started requiring an API key on June 22, 2016 (after about 10 years

of allowing keyless use). For using Google Maps on your website, you will need to sign up for and implement an API key. Follow the link below to get an API key and replace it with YOUR_API_KEY in the script above:

https://developers.google.com/maps/documentation/javascript/adding-a-google-map - step_3_get_an_api_key

Step 24: Note that when the <body> onload event occurs 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,{enabl
eHighAccuracy : 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 google.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> container 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 centered 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.

Registration form with validation should be similar to the following form (use iOS mobile device to open):
https://www.cs.drexel.edu/~eb452/cs338/Lab3/Register.html

The GPS and Google Maps API for geolocation functionality (use iOS mobile device to open):
https://www.cs.drexel.edu/~eb452/cs338/Lab3/Location.html

When you are done, make sure to submit your URL for Lab 3 in BBLearn.