Internet Programming

Week 5

Instructor:
Daniel Slack, P. Eng.
Applied Computer Science
University of Winnipeg

Announcements

- Midterm next week
  - ID is required, make sure to bring it
  - Exam will be 2 hours
    - 6:15 – 8:15
  - One hand-written, **single-sided** information sheet allowed
  - No Phone allowed – academic dishonesty!
  - Bags/jackets off to the side

Announcements

- Feedback is welcome!
  - Assignment communication was great!
  - I can adjust the course
  - Helps mold the course to make it better

Drawing on Canvas

Chapter 16

Lecture 16 Example

- Demonstrate TweetShirt example

Introduction

- Browsers give us several ways to display graphics
  - Simplest is to use styles to position and colour DOM elements
    - Can get you pretty far
    - Problem: Using the DOM for something that it wasn't original designed
- Alternatives:
  1. Scalable Vector Graphics
     1. Not covered in this course
  2. Canvas DOM element

The Canvas Element

- Canvas graphics can be drawn onto a <canvas> element
  - Specified for some height and width
  - Intended to support different styles of drawing
  - Can have multiple canvas elements per page
  - The canvas is transparent by default
    - Can position it on top of other elements to draw on top of them
- Provides pixel operations
  - Create, manipulate, and destroy

```
<canvas id="lookwhatIdrew" width="300" height="300"></canvas>
```

The Canvas Element

- Need to create a context in order to get a drawing interface
  - This is an object whose methods facilitate drawing on the canvas
  - Two currently support drawing styles:
    - *2d* for two-dimensional graphics
    - *webgl* for three-dimensional graphics through the OpenGL interface

Context

```
<p>Before canvas.</p>
<canvas width="120" height="60"></canvas>
<p>After canvas.</p>
<script>
  var canvas = document.querySelector("canvas");
  var context = canvas.getContext("2d");
  context.fillStyle = "red";
  context.fillRect(10, 10, 100, 50);
</script>
```

Failing Gracefully

- Text between canvas elements will be displayed on browsers that do not support the canvas element
- The following code also determines if the browser supports the canvas element

```
if (canvas.getContext) {
        //you have canvas
} else {
        //sorry, no canvas object
}
```

Filling Other Shapes

- What about creating polygons?
  - Is there a fillPolygon method?
    - No
- To create a polygon:
  1. Specify the start of a path: `context.beginPath()`
  2. Specify the start of the path: `context.moveTo(x,y)`
     - Moves the path to a point, without creating a line
  3. Specify the vertices of the polygon: `context.lineTo(x,y)`
     - Similar to `moveTo`, but creates a line
  4. Indicate the path is closed: `context.closePath()`
     - Creates a path from the current point to the starting point

Filling Other Shapes

- To create a polygon continued:
  5. Fill the path in
     - A path is not displayed unless it is drawn with the stroke() method
     a) Set the line width: `context.linewidth = #`
     b) Draw the line: `context.stroke()`
  6. Fill the polygon
     a) Set the fill style if necessary: `context.fillStyle = #`
     b) Fill the polygon: `context.fill()`

Filling Circles

- Our fictitious example requires either circles or squares
- To create a circle:
  1. Specify the start of a path: `context.beginPath()`
  2. Use

```
context.arc(x, y, sAngle, eAngle, counterclockwise)
```

Writing Text To the Canvas

- To add text to the canvas:
  1. Specify the font: `context.font = font`
     - See https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/font
  2. Draw the text on the canvas:

     ```
     context.fillText(text, x, y)
     ```

Adding An Image to the Canvas

- To add an image to the convas
  1. Create an image element:
     `document.createElement("img");`
  2. Set the source property of the element for the image
     - `myImg.src = "someImage.jpg";`
  3. Use `context.drawImage(img, x, y, width, height);`

Lecture 9 Examples

- TweetShirt
  - Let's draw a black-filled rectangle

Aside – Local Web Server

Running Local Web Server

- Many different options
  - IIS, Mongoose, apache, nginx, python
- Simple server for NodeJS
  - NodeJS is Server-side Javascript
- http://www.nodejs.org
  - Select platform
  - Install
  - Should be added to your path

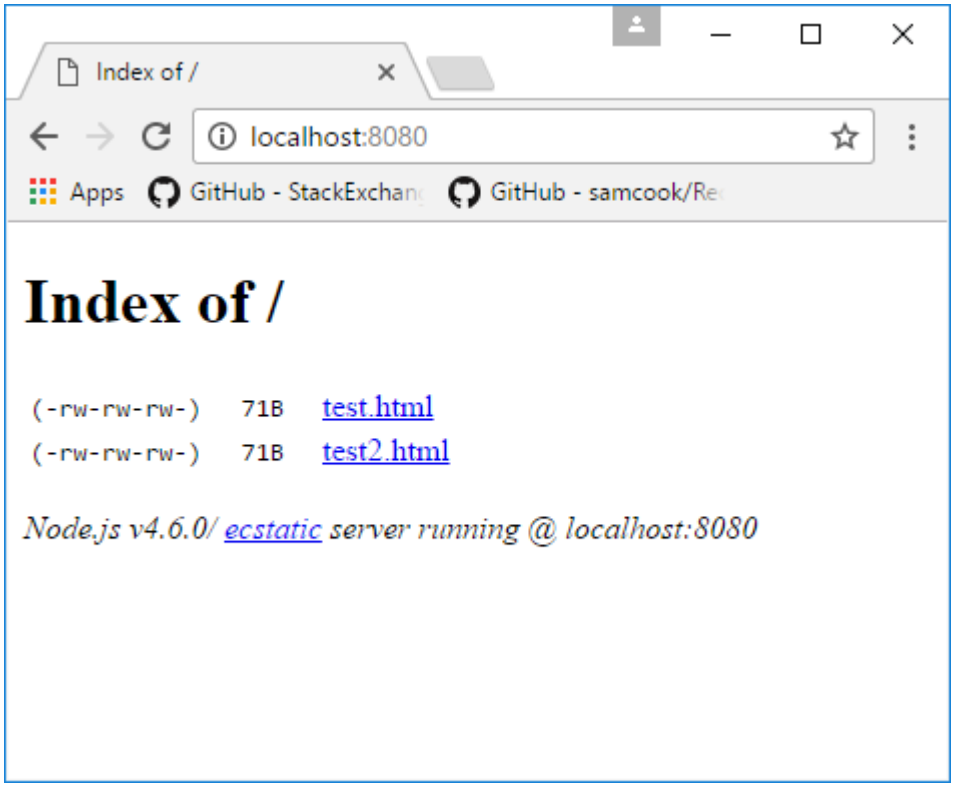
Running Local Web Server

- NPM
  - Node Package Manager
  - Huge repository of JS packages
  - Including “http-server”

```
C:\>npm install -g http-server
```

```
C:\>http-server
Starting up http-server, serving ./
Available on:
  http://192.168.0.17:8080
  http://127.0.0.1:8080
Hit CTRL-C to stop the server
```

Running Local Web Server



Geolocation

Slides Only

Location

- Your users are on the move with mobile devices
  - Great apps will enhance user's experiences based on their location
    - Give directions
    - Make suggestions as to where they may go
    - Make suggestions based on the weather
- HTML5 + Geolocation JavaScript-based API
  - Easily access location information in your pages

Geolocation API

- Geolocation is not considered part of the HTML5 standard
  - It is a standard of the W3C
  - Widely supported
- Geolocation API is not the same as the Google Maps API
  - Geolocation is solely focused on getting your location on earth
  - Google Maps API is a JavaScript library offered by Google
    - Gives you access to their Google Maps functionality
    - *i.e.* if you want to display your user's location in a map

Geolocation API

- Geolocation API specifies that any browser must have the permission of the user to make use of their location

Geolocation API

- Geolocation API gives us longitude and latitude
  - Always as real numbers
    - Rather than in degrees/minutes/seconds notation
    - Aside: How can one convert between the two?

$$\begin{aligned} \mathrm{D} &= \mathrm{trunc}(\mathrm{D}_{\mathrm{dec}}) \\ \mathrm{M} &= \mathrm{trunc}((\mathrm{D}_{\mathrm{dec}} \times 60) \mod 60) \\ \mathrm{S} &= (|\mathrm{D}_{\mathrm{dec}}| \times 3600) \mod 60 \end{aligned}$$

$$\mathrm{D}_{\mathrm{dec}} = \mathrm{D} + \frac{\mathrm{M}}{60} + \frac{\mathrm{S}}{3600}$$

How is Location Determined?

- More than just smartphones are location aware
  - Even browsers running on PCs can do it
- Four different approaches
  - Global Positioning System
    - Supported by many newer mobile devices
    - Extremely accurate location information
      - Based on Satellites
    - May include altitude, speed, and heading
    - Device must be able to see the sky
    - Can take a long time
    - Hard on batteries

How is Location Determined?

- IP Address
  - Uses IP address to map the address to a physical location
  - Can be used anywhere
  - Although the addresses are often resolved to your ISP's local office
- Cell Phone Triangulation
  - Find your location based on your distance from one or more cell phone towers
  - The more towers, the more accurate
  - Works indoors
  - Quicker than GPS
  - Accuracy suffers if only using one tower

How is Location Determined?

- WiFi
  - Uses one or more WiFi access points
  - This method can be very accurate
  - Works indoors and is fast
  - Requires you are stationary
- How would this method work?
- Browser determines which method to use

L10 Example 01

- Simply obtaining longitude/latitude via geolocation api

Points of Interest

- Browsers have a geolocation property in their navigation object **only if** geolocation is supported
- `navigator.geolocation` is an object that contains the entire geolocation API

Mapping Your Position

- Geolocation provides a way to find your location
  - Doesn't provide any tools to visualize your location
- Must rely on third-party tools
  - Google Maps is the most popular
    - Not part of the HTML5 specifications

Geolocation

- API is really simple
  - Has 3 methods
    - `getCurrentPosition`
    - `watchPosition`
    - `clearWatch`
  - Two Properties
    - `coords`
    - `timestamp`

Geolocation

```
getCurrentPosition(
   successHandler, errorHandler, positionOptions
);
```

- `successHandler` is called when a location is determined
- `errorHandler` is called when the browser cannot determine location
- `positionOptions` allow fine tuning

Geolocation

- coords has 3 guaranteed and some non-guaranteed properties
  - Guaranteed
    - `latitude`
    - `longitude`
    - `accuracy`
  - Non-guaranteed
    - `altitude`
    - `altitudeAccuracy`
    - `heading`
    - `speed`

Accuracy

- Every location comes with an accuracy measure
  - In meters
- Location is accurate within a 95% confidence level
- Example:
  - 500 meters accuracy
    - We can count on the location as long as we factor in a radius of 500m
    - Good for city or neighbourhood detection

L10 Example 02

- Find the distance from Portage and Main to browser's location

Google Maps API

- The following can be accessed through JavaScript
- Controls:
  - Zoom, pan, switch between Map and Satellite view, Street view control
- Services:
  - Directions, distance, and street view
- Overlays:
  - For example, heat map overlays, traffic congestion, custom overlays

Google Maps API

- Needs an API Key
  - Your own “key” to access their services
  - Free to register
  - https://developers.google.com/maps/documentation/javascript/get-api-key

L10 Example 03

- Using Geolocation API and Google Maps API to display a map centred on browser's location

L10 Example 04

- Marker Example
  - Put a marker on a map at a specific location using Google Maps API

watchPosition

- Goal: Create an app that tracks your movements in real time
- Using the watchPosition method
  - Watchers your movements and reports your location back as it changes
  - Looks just like getCurrentPosition method
    - Instead, it repeatedly calls the success handler each time position changes

watchPosition

1. App calls watchPosition
   - Passing in a success handler function
2. watchPosition runs in background
   - Constantly monitoring your position
   - Calls success handler when your position changes
3. watchPosition continues until you call clearWatch

L10 Example 05

- watchPosition Example
  - Create a web application that will update your current location on a map

getCurrent and watchPosition Options

- getCurrentPostion (and watchPosition) can have options
  - Control how geolocation computes its values

```
var positionOptions = {
        enableHighAccuracy: false,
        timeout: Infinity,
        maximumAge: 0
}
```

getCurrent and watchPosition Options

- enableHighAccurarcy
  - Can tell browser to use only the most accurate result
  - Only a hint
    - Browser implementations do different things with hint
    - Does not guarantee the most accurate location
  - You are telling the browser to use the most accurate location
    - Even if it is costly
- timeout
  - Controls how long the browser gets to determine its location
    - By default, set to infinity
  - Error handler is called if timeout occurs before location is retrieved

getCurrent and watchPosition Options

- maximumAge
  - Sets the oldest age a location can be before the browser needs to recalculate
    - e.g. If a browser has a location 60 sec. old and maximumAge is 90000 (90 sec.), a call to getCurrentPosition would return the same location
  - Zero means browser always has to recalculate
  - Can be used to make app faster and more power efficient

watchPosition Example

- With map tracking