Creating Modern Web Applications

Lecture 2: The Browser and HTML
The Browser

• What Browser do you use?
  • Why?
  • Is one browser better than another?
  • What should you use in this course?
• What does it do for you?
  • Magically makes pages appear
  • Allows interaction
  • Supports the user experience
Web Applications

- Front End
- Web Browser
- Mobile Platform
- Back End
- Web Server
- Database
- Server

HTTP
Browsers as Application Front Ends

- Do they make writing quality applications easier or harder?
Using the Browser is Helpful

• Makes it easy to create sophisticated interfaces
  • Including images, videos, dynamics
  • Color, typography, accessibility, ..
  • Adapts to different size windows
  • Works on different platforms

• Powerful declarative syntax for user interfaces
• Easy prototyping of user interfaces
• Much less code to write
Why the Browser is Restrictive

• Using a browser as a front end is **limiting**
  • Limits the **user** interface
  • Limits the **user** experience
  • Limits application control of the interface
  • Interactive applications are more difficult (often more code, slower)
• Other limitations include
  • Limited communications capabilities
  • Limited access to the user machine
  • Separates the UI from the application
  • Limited display capabilities
• **You need to understand the limitations**
Front End Code Restrictions

• Who can the front end code talk to and see
  • Any program/file on the user’s machine or network
    • This is a security/privacy problem
      ─ Requires explicit user approval; Discouraged by today’s browsers
    • Best not to assume this
  • Restricted local storage (cookies, html5 storage, if allowed by user)
  • The web server
    • Actually any socket on the machine serving the pages
    • Firewalls might limit access to specific ports
    • Generally only the web server and its components
• Talking to the server uses a limited set of protocols
  • URLs, HTTP

1/26/18  Lecture 2: The Browser and HTML
Uniform Resource Locators (URL)

- **HTTP**: `//www.cs.brown.edu:80/people/spr`
  - #reference
  - ?name=value&name1=value1...
- **Examples**
  - Amazon: [https://www.amazon.com/dp/B07456BG8N/ref=ods_gw_ha_rr_p_3pack?pf_rd_p=ece83bcf-c3b4-4c99-b291-7adfd1d50988&pf_rd_r=BN6NP9MBBVFRXX1WNHX](https://www.amazon.com/dp/B07456BG8N/ref=ods_gw_ha_rr_p_3pack?pf_rd_p=ece83bcf-c3b4-4c99-b291-7adfd1d50988&pf_rd_r=BN6NP9MBBVFRXX1WNHX)
What the Browser Does

- You are a browser
  - Someone gives you http://www.cs.brown.edu
  - What do you do?
What the Browser Does

• **Given a URL from the user or a program**
  • Finds the proper server (based on host)
  • Opens a socket to port 80 (or other specified port)
• Sends a **request** on that socket (based on protocol)
  • Server then finds the corresponding data
    • Generally the file referred to; Might be dynamically computed
  • Server sends back the result
• **Browser reads the response**
  • Builds an internal data structure from the response (DOM)
  • Displays the corresponding data structure (magic)
  • **Replaces** the current page (or frame) with the new one
Simple HTTP Request

**GET /people/spr HTTP/1.1**

**Host:** www.cs.brown.edu:80

<crlf>

<crlf>

1/26/18 Lecture 2: The Browser and HTML
HTTP Protocol: Requests

- Basic Verbs are GET and POST
  - GET: effectively header only
  - POST: provides content
  - GET <suburl> HTTP/1.1
- Header fields
  - name: value
  - Describe
    - Who is sending the request
    - Type of data passed and expected (e.g. text/html, text/xml)
    - Length of data; Cookies; Caching information; location; source
- Content
  - Blank line (CRLF) and then actual data
Simple HTTP Response

HTTP/1.1 200 OK
Date: Fri, 27 Jan 2012 10:25:23 EDT
Content-Type: text/html
Content-Length: 234

<html><head> ....
HTTP Protocol: Responses

• HTTP/1.1 `<status> <description>
  • 1xx: OK / continue
  • 2xx: Success in various ways
  • 3xx: Redirection
  • 4xx: Client error
  • 5xx: Server error

• Header fields
  • Content-type, Content-length
  • Date and other optional information
Web Pages are not Simple

- Typically include images
- Can include other web pages (ads)
- Can include multimedia (videos, sounds, ...)
- How is this handled?
HTTP is Stateless

Does this mean:

A. It doesn’t matter where the HTTP request came from?
B. HTTP requests can be linked to other requests using session ids?
C. HTTP requests can come in any order?
D. Each HTTP request is independent of each other?
E. HTTP requests can’t contain any information about the user or the browser?
HTTP is Stateless

- Each request is **independent** of other requests
  - From the same user or other users
  - From the same web page or other pages
  - Each request can be treated the same at the server

- **Why?**
  - Fits browser model of multiple windows, back/forward, ...
  - Don’t have to worry about state, errors, crashes, etc.

- **What’s wrong with this?**
  - It makes web applications much more difficult
  - We’ll get back to this later in the course
HTML

- Need a way of describing what to display
  - Work with all browsers (browser-independent)
  - Window size and shape independent
  - User-creatable
- History
  - SGML: type-setting markup language (EBT)
    - Basis for HTML and XML
  - Used for manuals off-line and on-line
- HTML = { HTML4, XHTML, HTML5 }
HTML Structure

- HTML is a tree structure
  - Internal nodes represent structure
  - Leaf nodes represent content
- Specified textually as a tree
  
  ```html
  <node>
    <subnode field='value'>
      Text in a leaf node
    <leafnode />
  </subnode>
  </node>
  ```
- Maintained internally as a tree (DOM)
- Nodes have names, attributes
- Text may appear at leaves
HTML Components

- **Header**: basic information about the page
  - Styles (CSS): information on how to display
    - Can be in separate files
    - Prelab (tutorial) + Lab Wednesday
  - Scripts (JavaScript)
    - Dynamic interactivity
    - Can be in separate files
    - Next Friday + Monday + Lab following week
- **Body**: the data to display
  - Description of what should be presented on the page
  - Semi-structured text
Simple HTML Example

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Page title. Shown in tabs.</title>
  </head>
  <body>
    <h1>Simple Page!</h1>
    <p>This is a totally <em>bare-bones</em> page.</p>
  </body>
</html>
```
Basic HTML Body Components

• Text
• Descriptions of how to display text
  • <EM>text</EM>, <SPAN CLASS='emph'>text</SPAN>
  • Managed by CSS
• Text organization
  • Headers, paragraphs, blocks, tables
• Page layout and organization
  • DIV, LIST, TABLE, FRAME
• Interactive regions
  • Forms: text fields, buttons, ...
  • Canvas, SVG regions
HTML Block Elements

- H1, H2, H3, H4, H5, H6 (header)
- P (paragraph)
- UL, OL, LI (unordered list, ordered list, list item)
- DIV (logical division)
  - HTML5 div specialization (such as ‘header’, ‘footer’, ‘section’, and ‘article’)
- IMG (image)
- TABLE, TR, TH, TD (tables)
HTML Inline Elements

Tag:  Usage:
• A  • <A href="http://google.com">Google</A>
• STRONG  • <STRONG>Usually Bold</STRONG>
• EM  • <EM>Usually italicized</EM>
• SPAN  • Text which is <SPAN>logically divisible</SPAN>
• IMG  • <IMG src="image.png" alt="description" />
• text  • Arbitrary text to display
HTML Flow and Layout

Spans are only as wide as the content they wrap.

Spans can be put inside divs. Note that regardless of how much horizontal space they take, the div always expands to the width of its parent.

Divs always cause a new line.

Scrolling direction
CSS: Style versus Content

- **Basic Syntax**
  
  Selectors {
    Property : value;
    Property : value;
  }

- **Including inline**
  
  • <STYLE> .... </STYLE>

- **Including via links**
  
  • <LINK rel='stylesheet' href='path.css' type='text/css' />

- **Tricky Parts**: selectors, property names, property values

- **Code Bubbles Before and After**
  
  • [Http://www.cs.brown.edu/people/spr/codebubbles](http://www.cs.brown.edu/people/spr/codebubbles)
  
  • [Http://www.cs.brown.edu/people/spr/codebubbles/indexold.html](http://www.cs.brown.edu/people/spr/codebubbles/indexold.html)
CSS Selectors

- `h3 { ... }`
  - Apply to all h3 tags
- `.emph { ... }`
  - Apply to all elements with class emph
- `#idtag {}`
  - Applies to the element with id = idtag
- Combinations, nestings, etc. are possible

---

CSS Selectors

<table>
<thead>
<tr>
<th>Selector</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p {}</code></td>
<td>Tag selector, all p tags</td>
</tr>
<tr>
<td><code>#para[]</code></td>
<td>Id para (unique)</td>
</tr>
<tr>
<td><code>.para1{}</code></td>
<td>Class para1 (multiple)</td>
</tr>
<tr>
<td><code>p .para[]</code></td>
<td>P tag with class para</td>
</tr>
<tr>
<td><code>div p[]</code></td>
<td>P tag having parent div.</td>
</tr>
<tr>
<td><code>*[]</code></td>
<td>All tags (Universal Selector)</td>
</tr>
<tr>
<td><code>h1, h3, h5[]</code></td>
<td>Only h1, h3 and h5 (grouping)</td>
</tr>
<tr>
<td><code>.para a[]</code></td>
<td>A with parent para class</td>
</tr>
<tr>
<td><code>body{}</code></td>
<td>Parent of all tags</td>
</tr>
</tbody>
</table>
Next Time

• **Monday: Universal Accessibility**
• **Student project proposals are due**
• **Homework:**
  • Assignment 0
  • Prelab 1 due Wednesday
• **Class Prep for Monday**
  Take a web site of your choosing. Using the accessibility options on your computer, try to access it via a screen reader, a high-contrast display, or with 4X or larger magnification or other accessibility feature. Come prepared to discuss your experience.
Final Projects

• Why use external sponsors?
• Student projects
  • Requirements and specifications
  • Importance of having a well-defined project
  • You will need a suite of test users (other than yourselves)
HTML Examples

- Course home page
- Displaying text
- Using CSS and styles