Course Overview

SWE 432, Fall 2018
Design and Implementation of Software for the Web
Web Application Development
Course Topics

• How do we organize, structure and share information?

• How to make applications that are delivered through browsers
  • JavaScript, front-end and back-end development, programming models, testing, performance, privacy, security, scalability, deployment, etc.

• How to design user interactions, focusing on browsers
  • User-centered design, user studies, information visualization, visual design, etc.
Logistics

• No textbook, but suggested readings will be listed on course schedule

• Lab-style work included in many lectures (bring your laptop)
Grading

• 50% Homework
  • 5 assignments, ~2 weeks to do each, all done individually
  • Your code will be autograded; you can resubmit an unlimited number of times until the deadline and view your score
  • Also graded by hand for some non-functional issues

• 10% Quizes
  • Pass/fail (Pass if you are in class and submit a quiz, fail if you don’t)
  • Use laptop or phone to complete the quiz in class

• 15% Midterm Exam, 20% Final Exam
Policies

• My promises to you:

• Quiz results will be available instantaneously in class; we will discuss quiz in real time

• Homework will be graded within 1 week of submission

• Exams will be graded within 1 week
Policies

• Lateness on homework:
  • 10% penalty if submitted UP TO 24 hours after deadline
  • No assignments will be accepted more than 24 hours late
  • Out of fairness: no exceptions

• Attendance & Quizzes:
  • You can miss up to 3 with no penalty
  • Again, out of fairness: no exceptions beyond this
Course Staff

- Prof Jonathan Bell (me)
- Office hour: ENGR 4422 Mon & Weds 11:00am-12:00 pm or by appointment
- Areas of research: Software Engineering, Program Analysis, Software Systems

Two hobbies: cycling, ice cream
Course Staff

• TAs: Mrudla Ichanahalli Anantharamaiah (Mia)

• Office Hours: TBA

• Please, no emails to instructor or TAs about the class: use Piazza
Honor Code

• Refresh yourself of the department honor code

• Homeworks are 100% individual
  
  • Discussing assignments at high level: ok, sharing code: not ok

  • If in doubt, ask the instructor

  • If you copy code, we WILL notice (see some of my recent research results in “code relatives”)

• Quizzes must be completed by you, and while in class
Project Overview

• Build a portfolio-worthy web application piece-by-piece

• Split into four deliverables

• Each component builds on the last but you do not need to reuse any code

• Starts with backend programming, then frontend

• Separate project: interaction design (without programming) in last few weeks of class
Project Overview

GROUP PROJECTS EVERYWHERE
Project Topic

Maybe I should make a Meme Generator
Project Topic

YOU GET A MEME!
YOU GET A MEME!
EVERYONE GETS A MEME!
Project Topic

• First assignment will be released on Weds

• High level:
  
  • HW1: Implement the graphic generator
  
  • HW2: Make a web service
  
  • HW3: Make a frontend for web service
  
  • HW4: Add social features
Web Sites vs Web Apps?

Interactive?

User-generated content?

Informational vs fun?
What is the web?

• A set of standards
  • TCP/IP, HTTP, URLs, HTML, CSS, …

• A means for distributing structured and semi-structured information to the world

• Infrastructure
Perspectives in web development
Systems Perspective

- How can we design **robust**, **efficient**, & **secure** interactions between computers?

- Individual web app may run on
  - Thousands of servers
  - Owned and managed by different orgs
  - **Millions** of clients
  - >TBs of constantly changing data

- What happens when a server crashes?
- How do we prevent a malicious user from accessing user data on a server?
Software Engineering Perspective

- How can we design for **change** & **reuse**?
- Individual web app may
  - **Hundreds of** developers
  - **Millions** of lines of code
  - New updates deployed many times a day
  - Much functionality reused from code built by other organizations
  - Offer API that allows other web apps to be built on top of it
- How can a developer successfully make a change without understanding the whole system?
- What happens when a new developer joins?
Human-Computer Interaction (HCI)

Perspective

- How can we design web apps that are **usable** for their intended purpose?
- Individual web app may
  - **Millions of users**
  - Tens of different needs
- What happens when a new user interacts with the web app?
- How can we make a web app less frustrating to use?
Pre-Web

• “As We May Think”, by Vannevar Bush, in The Atlantic Monthly, July 1945

• Recommended that scientists work on inventing machines for storing, organizing, retrieving and sharing the increasing vast amounts of human knowledge

• He targeted physicists and electrical engineers - there were no computer scientists in 1945
Pre-Web - Memex

- MEMEX = MEMory EXtension
- Create and follow “associative trails” (links) and annotations between microfilm documents
- Technically based on “rapid selectors” Bush built in 1930’s to search microfilm
- Conceptually based on human associative memory rather than indexing
Pre-Web - Memex

Never built
Hypertext and the WWW

- 1965: Ted Nelson coins “hypertext” (the HT in HTML) - “beyond” the linear constraints of text

- Many hypertext/hypermedia systems followed, many not sufficiently scalable to take off

- 1968: Doug Engelbart gives “the mother of all demos”, demonstrating windows, hypertext, graphics, video conferencing, the mouse, collaborative real-time editor

- 1969: ARPANET comes online

- 1980: Tim Berners-Lee writes ENQUIRE, a notebook program which allows links to be made between arbitrary nodes with titles
Origin of the Web


• Became what we know as the WWW

• A “global” hypertext system full of links (which could be single directional, and could be broken!)
Early Browsers

CERN

The European Laboratory for Particle Physics, located near Geneva[1] in Switzerland[2] and France[3]. Also the birthplace of the World-Wide Web[4].

This is the CERN laboratory main server. The support team provides a set of Services[5] to the physics experiments and the lab. For questions and suggestions, see WWW Support Contacts[6] at CERN.


About the Laboratory

Help[13] and General information[14], divisions, groups and activities[15] (structure), Scientific committees[16]

Directories[17] (phone & email, services & people), Scientific Information Service[18] (library, archives or Alice), Preprint[19] Server

1-45, Back, Up, <RETURN> for more, Quit, or Help:
Original WWW Architecture

Links!!
URI: Universal Resource Identifier

URI:  `<scheme>://<authority><path>?<query>`

```
http://cs.gmu.edu/syllabus/syllabi-fall16/SWE432BellJ.html
```

“Use HTTP scheme”

Other popular schemes:
ftp, mailto, file

“Connect to cs.gmu.edu”

May be host name or an IP address
Optional port name (e.g., :80 for port 80)

“Request syllabus/syllabi-fall16/SWE432BellJ.html”

DNS: Domain Name System

- Domain name system (DNS) (~1982)
- Mapping from names to IP addresses
- E.g. cs.gmu.edu -> 129.174.125.139
HTTP: HyperText Transfer Protocol

High-level protocol built on TCP/IP that defines how data is transferred on the web.

HTTP Request:
- GET /syllabus/syllabi-fall16/SWE432BellJ.html HTTP/1.1
- Host: cs.gmu.edu
- Accept: text/html

HTTP Response:
- HTTP/1.1 200 OK
- Content-Type: text/html; charset=UTF-8
- <html><head>...

Reads file from disk

SWE 432 Section 002 Fall 2016 Syllabus and Schedule
"Design and Implementation of Software for the Web"

Class Hours: Tuesdays and Thursdays, 12:00pm-1:16pm  Robinson Hall B226
Grades, Readings available as pdf's: Blackboard
Resources (Announcements, Schedule, Assignments, Discussion):
Piazza - https://piazza.com/gmu/fall2016/swe432001/home

Instructor: Prof. Jonathan Bell
bell@gmu.edu
http://jbell.net
Twitter: @jbell
Office: 4422 Engineering Building; (703) 993-6089
Office Hours: Anytime electronically. Tues 10:30am-12:00pm, or by appointment
HTTP Requests

- Request may contain additional *header lines* specifying, e.g. client info, parameters for forms, cookies, etc.
- Ends with a carriage return, line feed (blank line)
- May also contain a message body, delineated by a blank line

```
HTTP Request
GET /syllabus/syllabi-fall16/SWE432BellJ.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html

```

“GET request”  “Resource”

Other popular types: POST, PUT, DELETE, HEAD
HTTP Responses

"OK response"
Response status codes:
1xx Informational
2xx Success
3xx Redirection
4xx Client error
5xx Server error

"HTML returned content"
Common MIME types:
application/json
application/pdf
image/png
Properties of HTTP

- Request-response
  - Interactions always initiated by client request to server
  - Server responds with results
- Stateless
  - Each request-response pair independent from every other
  - Any state information (login credentials, shopping carts, etc.) needs to be encoded somehow
HTML: HyperText Markup Language

HTML is a **markup language** - it is a language for describing parts of a document.
HTML: HyperText Markup Language

• NOT a programming language

• Tags are added to markup the text, encompassed with <>’s

• Simple markup tags: <b>, <i>, <u> (bold, italic, underline)

  <b>This text is bold!</b>

  This text is bold!
Web vs. Internet

**Web**

- Application layer
- Transport layer
- Internet layer
- Link layer

**Internet**

- PPP, MAC (Ethernet, DSL, ISDN, …)
- IP, ICMP, IPSec, …
- TCP, UDP, …

**HTML**
- DNS, FTP, HTTP, IMAP, POP, SSH, Telnet, TLS/SSL, …

**CSS**

**Browser**
The Modern Web

- Evolving competing architectures for organizing content and computation between browser (client) and web server

- 1990s: static web pages

- 1990s: server-side scripting (CGI, PHP, ASP, ColdFusion, JSP, …)

- 2000s: single page apps (JQuery)

- 2010s: front-end frameworks (Angular, Aurelia, React, …), microservices
Static Web Pages

- URL corresponds to directory location on server
  
  - e.g. http://domainName.com/img/image5.jpg maps to img/image5.jpg file on server

- Server responds to HTTP request by returning requested files

- Advantages
  
  - Simple, easily cacheable, easily searchable

- Disadvantages
  
  - No interactivity
Web 1.0 Problems

- At this point, most sites were “read only”
- Lack of standards for advanced content - “browser war”
- No rich client content… the best you could hope for was a Java applet

https://en.wikipedia.org/wiki/Browser_wars
https://en.wikipedia.org/wiki/Java_applet
Dynamic Web Pages

HTTP Request
GET /syllabus/syllabi-fall16/SWE432BellJ.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html

HTTP Response
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

SWE 432 Section 002 Fall 2016 Syllabus and Schedule
"Design and Implementation of Software for the Web"

Class Hours: Tuesdays and Thursdays, 12:00pm-1:16pm Robinson Hall B228
Grades, Readings available as pdfs: Blackboard
Resources (Announcements, Schedule, Assignments, Discussion): Piazza - https://piazza.com/gmu/fall2016/swe432001/home

Instructor: Prof. Jonathan Bell
bell@cs.gmu.edu
http://jbell.net
Twitter: @jach_bell
Office: 4422 Engineering Building; (703) 993-6088
Office Hours: Anytime electronically: Tues 10:30am-12:00pm, or by appointment
Dynamic Web Pages

HTTP Request
GET /syllabus/syllabi-fall16/SWE432BellJ.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html

Web Server Application

Syllabus Generator Application

Runs a program

Give me /syllabus/syllabi-fall16/SWE432BellJ.html

Here's some text to send back

HTTP Response
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

<html><head>...

There's a standard mechanism to talk to these auxiliary applications, called CGI (Common Gateway Interface)
Server Side Scripting

• Generate HTML on the server through scripts

```html
<!DOCTYPE html>
<html>
  <head>
    <title>PHP Test</title>
  </head>
  <body>
    <p>Hello World</p>
  </body>
</html>
```

• Early approaches emphasized embedding server code inside html pages

• Examples: CGI
Server Side Scripting Site

Browser

HTTP Request

Web Server

HTML templates, server logic, load / store state to database

Database

HTTP Response (HTML)
Limitations

• Poor **modularity**
  
  • Code representing logic, database interactions, generating HTML presentation all tangled
  
  • Example of a Big Ball of Mud [1]
  
  • Hard to understand, difficult to maintain
  
  • Still a step up over static pages!

Server Side Frameworks

- Framework that structures server into tiers, organizes logic into classes
- Create separate tiers for presentation, logic, persistence layer
- Can understand and reason about domain logic without looking at presentation (and vice versa)
- Examples: ASP.NET, JSP
Server Side Framework Site

Browser

Web Server

Database

HTTP Request

HTML

HTTP Response

Presentation tier

Domain logic tier

Persistence tier

Browser sends an HTTP request to the Web Server, which processes the request and returns an HTTP response containing HTML.

Web Server

Presentation tier processes the HTML and is responsible for rendering the content.

Domain logic tier handles the business logic of the application.

Persistence tier manages interaction with the database.

Database

Handles storage and retrieval of data for the application.

GMU SWE 432 Fall 2018
Limitations

• Need to load a whole new web page to get new data

  • Users must *wait* while new web page loads, decreasing responsiveness & interactivity

  • If server is slow or temporarily non-responsive, *whole user interface hangs!*

• Page has a discernible *refresh*, where old content is replaced and new content appears rather than seamless transition
Single Page Application (SPA)

• Client-side logic sends messages to server, receives response

• Logic is associated with a single HTML pages, written in Javascript

• HTML elements dynamically added and removed through DOM manipulation

```html
<b>Projects:</b>
<ol id="new-projects"></ol>

<script>
$( "#new-projects" ).load( "/resources/load.html #projects li" );
</script>

</body>
</html>
```

• Processing that does not require server may occur entirely client side, dramatically increasing responsiveness & reducing needed server resources

• Classic example: Gmail
SPA Enabling Technologies

- **AJAX**: Asynchronous Javascript and XML
  - Set of technologies for sending asynchronous request from web page to server, receiving response

- **DOM Manipulation**
  - Methods for updating the HTML elements in a page after the page may already have loaded

- **JSON**: JavaScript Object Notation
  - Standard syntax for describing and transmitting Javascript data objects

- **JQuery**
  - Wrapper library built on HTML standards designed for AJAX and DOM manipulation

---

JSON

```json
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

https://en.wikipedia.org/wiki/JSON
Single Page Application Site

Browser

events

HTML

HTML elements

Javascript

Web Server

Presentation tier

Domain logic tier

Persistence tier

Database

HTTP Request

HTTP Response (JSON)
Limitations

- Poor modularity *client-side*

  - As logic in client grows increasingly large and complex, becomes Big Ball of Mud
  
  - Hard to understand & maintain
  
  - DOM manipulation is *brittle & tightly coupled*, where small changes in HTML may cause unintended changes (e.g., two HTML elements with the same id)

- Poor reuse: logic tightly coupled to individual HTML elements, leading to code duplication of similar functionality in many places
Front End Frameworks

• Client is organized into separate *components*, capturing model of web application data

• Components are reusable, have encapsulation boundary (e.g., class)

• Components separate *logic* from *presentation*

• Components dynamically generate corresponding code based on component state
  • In contrast to HTML element manipulation, *framework* generates HTML, not user code, decreasing coupling

• Examples: Meteor, Ember, Angular, Aurelia, React
Front End Framework Site

Browser
- Component presentation
- Component logic
- Front end framework

Web Server
- Presentation tier
- Domain logic tier
- Persistence tier

Database

HTTP Request → HTTP Response (JSON)
Limitations

• Duplication of logic in client & server
  • As clients grow increasingly complex, must have logic in both client & server
  • May even need to be written twice in different languages! (e.g., Javascript, Java)
  • Server logic closely coupled to corresponding client logic. Changes to server logic require corresponding client logic change.
  • Difficult to reuse server logic
Microservices

- Small, focused web server that communicates through *data* requests & responses
  - Focused *only* on logic, not presentation
- Organized around capabilities that can be reused in multiple context across multiple applications
- Rather than horizontally scale identical web servers, vertically scale server infrastructure into many, small focused servers
Microservice Site

Browser

- Component presentation
- Component logic
- Front end framework

HTTP Request

Web Servers

- Microservice

HTTP Response (JSON)

Database

HTTP Request

- Microservice

HTTP Response (JSON)
Architectural Styles

- Architectural style specifies
  - how to partition a system
  - how components identify and communicate with each other
  - how information is communicated
  - how elements of a system can evolve independently
Constant change in web architectural styles

- Key drivers
  - Maintainability (new ways to achieve better modularity)
  - Reuse (organizing code into modules)
  - Scalability (partitioning monolithic servers into services)
  - Responsiveness (movement of logic to client)
  - Versioning (support continuous roll-out of new features)
- Web standards have enabled *many* possible solutions
- Explored through *many, many* frameworks, libraries, and programming languages
The web today

• Many technologies for each architectural style
  • Most support more than one

• Applications often evolve from one architectural style to another
  • Leads to applications combining *multiple* architectural styles
  • E.g., Single page app that uses server side scripting for a separate set of pages

• Newer architectural styles not always better
  • More complex, may be overkill for simple sites
Philosophy of the Internet

• Decentralisation: No permission is needed from a central authority to post anything on the Web, there is no central controlling node, and so no single point of failure … and no “kill switch”! This also implies freedom from indiscriminate censorship and surveillance.

• Non-discrimination: If I pay to connect to the internet with a certain quality of service, and you pay to connect with that or a greater quality of service, then we can both communicate at the same level. This principle of equity is also known as Net Neutrality.

• Bottom-up design: Instead of code being written and controlled by a small group of experts, it was developed in full view of everyone, encouraging maximum participation and experimentation.

• Universality: For anyone to be able to publish anything on the Web, all the computers involved have to speak the same languages to each other, no matter what different hardware people are using; where they live; or what cultural and political beliefs they have. In this way, the Web breaks down silos while still allowing diversity to flourish.

• Consensus: For universal standards to work, everyone had to agree to use them. Tim and others achieved this consensus by giving everyone a say in creating the standards, through a transparent, participatory process at W3C.

From http://webfoundation.org/about/vision/history-of-the-web/
Internet Governance

- IETF = Internet Engineering Task Force
- Open, all-volunteer organization
- Organized into working groups on specific topics
- Request for Comments
  - One of a series, begun in 1969, of numbered informational documents and standards followed by commercial software and freeware in the Internet and Unix communities
  - All Internet standards are recorded in RFCs