Web Development Tools

SWE 432, Fall 2017
Design and Implementation of Software for the Web
Today

- HW1 due next Tuesday
- Material from last time: properties review, cascades
- Web Development Tools
  - Version control with Git
  - Running JavaScript
JSON vs. objects

```javascript
var obj1 = {  prop: "value"  };
var obj2 = {  prop: 'value'  };
var obj3 = {  'prop': "value"  };
var obj4 = {  "prop": "value"  };
```

- All will create an object.
- obj1 or obj2 preferred syntax. Should try to be consistent.
- **Only** obj4 is valid JSON

```javascript
var obj5 = {  "prop": "'value'"  };
var obj6 = {  "prop": "\\value\""  };
```

- Can use different quote types for nesting. Or use \ to escape.
Cascade Pattern

• aka “chaining”
• Offer set of operations that mutate object and returns the “this” object
  • Build an API that has single purpose operations that can be combined easily
  • Lets us read code like a sentence
• Example (String):
  ```javascript
  str.replace("k","R").toUpperCase().substr(0,4);
  ```
• Example (jQuery):
  ```javascript
  $("#wrapper")
  .fadeOut()
  .html("Welcome")
  .fadeIn();
  ```
Version Control
We’ve always had some kind of version control

Carbon copies?
Git

• One of the latest generation of Version Control Systems (VCS)

• Came out of the Linux development community

• Fast

• Supports non-linear development

• Fully distributed (no need for “central” server)

• Able to handle large projects effectively
Linear vs Nonlinear Development

Linear history

Non-linear history
Distribution Model

Centralized Model

Examples:
CVS, Subversion, Perforce
Distribution Model

Centralized Model

Examples:
CVS, Subversion, Perforce

Central VCS Server

Version Database

version 3
version 2
version 1

Computer A
Checkout
file

Computer B
Checkout
file

Distributed Model

Examples:
Git, Mercurial

No central point of failure
No latency when doing most operations

Server Computer

Version Database

version 3
version 2
version 1

Computer A

Version Database

version 3
version 2
version 1

Computer B

Version Database

version 3
version 2
version 1
GitHub

• You don’t need a central server to use git

• But it sure makes it easier

• GitHub: Free open source hosting site
  • For students, free private repositories too!

• Set up a user account

• Create a repository

• Add collaborators
Git, High level

Workspace → Index → Local repository → Remote repository

- add
- commit
- commit -a
- push
- pull
- fetch
- checkout
- diff

E.g. The files you see → Git's local record of changes → Git's local repository → E.g. GitHub
**Typical Use Case**

**Person 1**
- Create Project & Repo
- Push to Github
- Edit/commit
- Edit/commit
- Edit/commit
- Pull/push

**Github**

**Person 2**
- Clone from Github
- Edit/commit/push
- Edit/commit
- Edit/commit
- Pull/push

**Github**
Git Resources

https://try.github.io/levels/1/challenges/1

Download and install git:
https://git-scm.com/downloads

Also: Git Cheat Sheet
IDEs
Integrated Development Environments (IDEs)

- **Integrates** everything you need to develop into a single **environment**

- E.g. text editor (with complex highlighting, auto complete, etc), version control access, debugger, etc.

- You don’t have to use one. It sure makes things handy though!

- We recommend you use WebStorm for this class. You can get a free student license: [https://www.jetbrains.com/student/](https://www.jetbrains.com/student/)
Executing JavaScript
Options for executing JavaScript

• Client Side
  • Pastebin—useful for debugging & experimentation
  • script tag inline—traditional mechanism
  • script include

• Server Side
  • node.js—webserver for JavaScript backends
Hello world

```javascript
var course = { name: 'SWE 432' };
console.log(`Hello ${course.name}!`);
```
Demo: Pastebin

• Careful: be sure to use correct quote character!
Demo: Script Tag Inline

hello.html

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <script> ...
    </script>
  </head>
</html>
```
Demo: Script Tag Include

hello.html

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="... .js"></script>
  </head>
</html>
```
Node.js

- Node.js is a *runtime* that lets you run JS outside of a browser
- We’re going to write backends with Node.js
- Why use Node?
  - Easy to get into after learning JS (it’s JS)
  - Event based: really efficient for sending lots of quick updates to lots of clients
- Why not use Node?
  - Bad for CPU heavy stuff
- Download and install it: https://nodejs.org/en/
  - We recommend LTS (LTS -> Long Term Support, designed to be super stable)
Demo: Node.js
Node Package Manager
Working with libraries

What’s wrong with this?

No standard format to say:

• What’s the name of the module?
• What’s the version of the module?
• Where do I find it?

Ideally: Just say “Give me React 15 and everything I need to make it work!”
A better way for modules

- Describe what your modules are
- Create a central repository of those modules
- Make a utility that can automatically find and include those modules

Your app

Dependences
Configuration

Package Manager

Modules that magically appear

Assumes dependencies magically exist

Declares what modules you need

Provides the modules to your app
NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies
- Declarative approach:
  - “My app is called helloworld”
  - “It is version 1”
  - You can run it by saying “node index.js”
  - “I need express, the most recent version is fine”
- Config is stored in json - specifically package.json

Generated by npm commands:

```json
{
   "name": "helloworld",
   "version": "1.0.0",
   "description": "",
   "main": "index.js",
   "scripts": {
      "test": "echo \"Error: no test specified\" && exit 1"
   },
   "author": "",
   "license": "ISC",
   "dependencies": {
      "express": "^4.14.0"
   }
}
```
Installing tools with NPM

- Installing module globally used for command line tools that are installed from npm repository.
- Some common tools we'll see
  - babel-cli -- command line interface (CLI) for Babel transpiler from ES6 (and JSX) to ES5
  - webpack -- bundler that processes require and import statements to generate single js file
  - nodemon
Nodemon

• What happens when your file changes?
  • Could use Control C to kill the process
  • And then start it again with node app.js
  • But that’s a lot of typing…..

• nodemon
  • Runs whatever is specified by “main” in package.json
Using NPM

• Your “project” is a directory which contains a special file, package.json

• Everything that is going to be in your project goes in this directory

• Step 1: Create NPM project
  \texttt{npm init}

• Step 2: Declare dependencies
  \texttt{npm install <packagename>}

• Step 3: Use modules in your app
  \texttt{var myPkg = require(“packagename”)}

• Do NOT include node_modules in your git repo! Instead, just do
  \texttt{npm install}

  • This will download and install the modules on your machine given the existing config!

\url{https://docs.npmjs.com/index}
NPM Scripts

• Scripts that run at specific times.

• npm start --- starts web server

https://docs.npmjs.com/misc/scripts
Demo: Hello World Server

1: Make a directory, myapp
2: Enter that directory, type `npm init` (accept all defaults)
3: Type `npm install express`
4: Create text file app.js:
   ```javascript
   const express = require('express');
   const app = express();
   const port = process.env.port || 3000;
   app.get('/', (req, res) => {
     res.send('Hello World!');
   });

   app.listen(port, function () {
     console.log('Example app listening on port ' + port);
   });
   ```
5: Type `node app.js`
6: Point your browser to `http://localhost:3000`
const express = require('express');
// Import the module express

const app = express();
// Create a new instance of express

const port = process.env.port || 3000;
// Decide what port we want express to listen on

app.get('/', (req, res) => {
    var course = { name: 'SWE 432' };
    res.send(`Hello ${course.name}!`);
});
// Create a callback for express to call when we have a “get” request to “/“. That
callback has access to the request (req) and response (res).

app.listen(port, function () {});
// Tell our new instance of express to listen on port.
In Class Activity

• Build a web server that returns a string with the current date and time.

Readings for next time

• Intro to Node and Express: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

• Hosting Node apps on Heroku: https://devcenter.heroku.com/articles/getting-started-with-nodejs#introduction