CS50
Basics of
Mobile Web Application Development

Tim Torgenrud

Tim Torgenrud focuses on data systems and integration for Stanford's computing application support team.

He has been working in Stanford IT Services since his student years in the mid-1980s.

He was a member of the team that established the initial stanford.edu web service for the campus.

Scotty Logan

Scotty Logan first used a web browser in 1993, and started running a web server a few months later.

Since 1999, he has worked in a variety of roles within Stanford IT Services.

He worked on the team that made Stanford's mobile home page, the IT Services home page, and the mobile version of Stanford Who.

He is a member of the IT Services Strategy and Architecture group, and is working on cloud and API platforms.

NOTICE

It has come to our attention that students with Gmail accounts have been receiving recent communications from Continuing Studies in their Spam Folder. An easy way to remedy this is:

Go into your Spam Folder

Click the small check box next to the email.

Click the button that appears at the top of the page that says "Not Spam." It is under the white search box.

At this time, it appears that only Gmail users have been affected. Keep in mind, if you too have a Gmail account, there is a good chance there are a few emails in your Spam Folder as well.

WiFi


Wi-Fi: On
Turn Wi-Fi Off
itlab
ClaybornsCasa
eduroam
LPCH_GUEST
LPCH_VOIP
shaymatt
SHC_Public
Stanford
Stanford Visitor
Devices
hpsetup
Join Other Network...
Create Network...
Open Network Preferences...

in the beginning...

there was the web

it wasn’t the desktop web

it wasn’t the mobile web

it was just the web

1993/1994

The Tardis WWW Server

This is a quick description of how the WWW server on `www.tardis.ed.ac.uk` *is set up, and where all the relevant files live.*

The pages containing descriptions of the hardware that the server runs on (a DEC MicroVAX II) have yet to be written.

`Www.tardis` currently runs the CERN httpd server. The files used by the server can be split into those belonging to users and those belonging to the server.

User Files

Each user on the Tardis system can set up their own set of WWW pages and scripts by creating a subdirectory of their home directory called `public_html`. In this directory there should be a file called `index.html`, which is the user's *homepage*. This page is accessed when only the user's directory is specified in the URL, e.g.

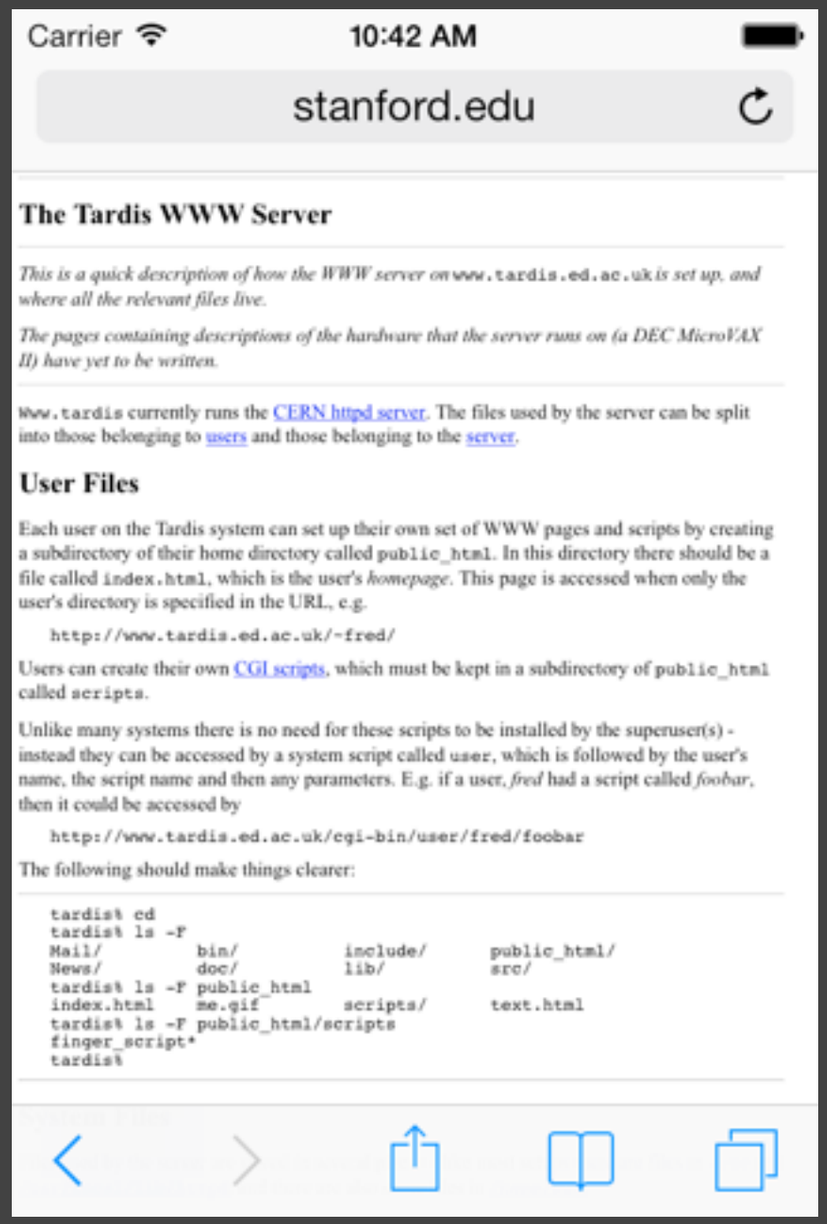
```
http://www.tardis.ed.ac.uk/~fred/
```

/home/www/src/

All the source for the `httpd` daemon is kept in this directory, along with the source for various utilities, such as `runhttpd`

Scotty Logan, scotty@tardis.ed.ac.uk, September 2 1994

that old stuff



was mobile friendly

Mobile Browsers

Originally

some support for HTML

maybe CSS & JavaScript

Smartphone Browsers

supported HTML, CSS and JavaScript

pretended to be 960px wide

Native App Pros

Available from app store

Easily discovered

Downloaded and installed on device

Available offline

Full device access

camera, location, storage, etc.

Native App Cons

Unique development tools per platform

Development platform limitations

Separate codebase for each platform

Dev Platforms

	iOS	Android	Windows Phone	Web
Mac OS X	✅	✅	🚫	✅
Windows	🚫	✅	✅	✅
Linux	🚫	✅	🚫	✅

Dev Platforms

	iOS	Android	Windows Phone	Web
Language	Objective C	Java, C++	C#	HTML, CSS, JavaScript
Dev Tool	Xcode	Eclipse, Android Studio	Visual Studio	Text Editor, Browser

Android

Wide range of device capabilities

- Screen size
- Screen resolution
- Hard vs soft keyboard

Scalable UIs need very good designers

Perceived lack of paying customers

iOS

Can get lost in app store

Only one app store

Review process can take a while

Mobile Web

Really just the web

Optimized for mobile devices

- Smaller screen
- Touch interface

"Distracted" user

- One eyeball, one thumb (Luke W)

Mobile Web

Initially, just pinch & zoom

m.thing.com

- Frameworks (Sencha, jQuery Mobile)

Mobile First

Progressive Enhancement

Responsive Design

Hybrid Apps

Single language

e.g. Xamarin (C#)

Or, native app wrapper + web app

Embedded web browser and content

Tools like Cordova / PhoneGap

Thousands of hybrid apps in app stores

(mobile) web app anatomy 101

Structure: HTML5

```
<!DOCTYPE html>
<html>
  <head>
    …
  </head>
  <body>
    …
  </body>
</html>
```

Style: CSS3

```
<!DOCTYPE html>
<html>
   <head>
      <link rel='stylesheet' href='…'>
   </head>
   <body>
      …
   </body>
</html>
```

Control: JavaScript

```
<!DOCTYPE html>
<html>
  <head>
    <link rel='stylesheet' href='…'>
    <script src='foo.js'></script>
  </head>
  <body>
    …
  </body>
</html>
```

HTML5 Semantic Tags

HTML5 Semantic Tags

More than just and

New elements have semantic meaning

nav - navigation

article, section - pieces of content

header, footer - above & below content

aside - additional content

CSS Selectors

Used to select elements for styling

Also used by jQuery Mobile framework

element selector

```
…
  <style>
  article { font-weight: bold }
  </style>
```

```
…
  <article id=‘one’ class=‘first’>
  …
  </article>
  <article id=‘two’ class=‘second’>
  …
  </article>
```

id selector

```
…
  <style>
  #one { font-weight: bold }
  </style>
…
  <article id=‘one’ class=‘first’>
  …
  </article>
  <article id=‘two’ class=‘second’>
  …
  </article>
```

class selector

```
…
  <style>
  .second { font-weight: bold }
  </style>
```

```
…
  <article id=‘one’ class=‘first’>
  …
  </article>
  <article id=‘two’ class=‘second’>
  …
  </article>
```

CSS Selector Game

CSS Diner

http://flukeout.github.io

JavaScript

aka ECMA Script

object oriented, functional language

functional model from scheme

prototypal inheritance from sather

syntax from C / Java

name from Java + politics

Unearthing the Excellence in JavaScript

JavaScript: The Good Parts

O'REILLY® | YAHOO! PRESS

Douglas Crockford

JavaScript Types

```
% node
> typeof "Hello, World!";
'string'
> typeof 'Hello, World!';
'string'
> typeof 3.14;
'number'
> typeof true;
'boolean'
> typeof false;
'boolean'
```

Object Literals

```
> var obj = {
... msg: "hello",
... pi: 3.14
... }
undefined
> typeof obj
'object'
```

Object Access

```
> obj.msg
'hello'
> obj[‘msg']
‘hello'
> Object.keys(obj)
[ 'msg', 'pi' ]
```

JSON - quick version

"JavaScript Object Notation"

JS object literals with stricter syntax

keys must be quoted

all strings must use " rather than '

no functions

JSON

```
{
  "msg": "hello",
  "pi": 3.14,
  "things": {
    "a": 1,
    "b": 2
  }
}
```

We'll come back to JSON

More Types

JavaScript also has

- Arrays
- Regular Expressions

but...

More Types

```
% node
> typeof /^abc/
'object'
> typeof [ 1, 2, 3 ]
'object'
> /^abc/ instanceof RegExp
true
> [ 1, 2, 3 ] instanceof Array
true
> typeof Math.floor;
'function'
```

Array Indexing

```
> var a = [ 1, 2, 3];
> a.length;
3
> a[1];
2
> a[0];
1
> for (i = 0; i < a.length; i++)
... { console.log(i, a[i]); }
0 1
1 2
2 3
```

Comments

```
/* C style comments
   are allowed */

// as are C++ style comments

/** some tools also support
 *  JavaDoc style comments
 * @constructor
 */
```

Functions - Procedural

```
// take an array and return a new array
// with 1 added to each array element
function plus1a (a) {
   if (a instanceof Array) {
      var r = [];
      for (var i = 0; i < a.length; i++) {
         if (typeof a[i] === 'number') {
            r[i] = a[i] + 1;
         } else {
            r[i] = 0; // could also use null, or just skip
      }
      return r;
   }

   throw new Error("Arrays only!”);
}
```

Functional Programming

Functions can be used like other values:

assigned to variables

passed to other functions

functions are “first class objects”

Very useful for

asynchronous callbacks

event handlers

Functional Programming

CS geeks love functional programming

My degree is in AI & CS

From the University of Edinburgh

Home of the

"Laboratory for Foundations
of Computer Science"

I did a lot of Standard ML

and Lisp, and Scheme

Functions

```
> var plus1 = function (n) {return n + 1; }
undefined
> function plus2 (n) { return n + 2; }
undefined
> plus1(3)
4
> plus2(3)
5
```

Functional

```
// take an array and return a new array
// with 1 added to each array element
function plus1a (a) {
  if (a instanceof Array) {
    return a.map(function (n) {
      return (typeof n === 'number')
        ? n + 1
        : 0; // or null, or undefined
    });
  }

  throw new Error("numbers or arrays only!");
}
```

Anonymous Functions

(or “lambda expressions” in CS)

A function that’s not assigned a name

A function that’s defined inline

```
return a.map(function (n) {
…
});
```

For CS Geeks

"these are not the closures you are looking for"

(OK, CS & Star Wars Geeks, who don't mind ending sentences with prepositions)

Anonymous?

```
> function () {}.name
''
> var f = function() {}
> f.name
''
```

```
> function f () {}
> f.name
'f'
> var f = function foo () {}
> f.name
'foo'
```

Debugging Tip

Name functions

Especially useful when using minimized JS

You should be using minimized JS in production

Unless production never has bugs

Without Names

```
> ( function () {
... ( function () {
..... throw new Error('failed!');
..... } )()
... } )()
Error: failed!
    at repl:3:7
    at repl:4:4
    at repl:5:4
    at REPLServer.self.eval (repl.js:110:21)
    at Interface.<anonymous> (repl.js:239:12)
    at Interface.EventEmitter.emit (events.js:95:17)
    at Interface._onLine (readline.js:202:10)
    at Interface._line (readline.js:531:8)
    at Interface._ttyWrite (readline.js:760:14)
    at ReadStream.onkeypress (readline.js:99:10)
```

With Names

```
> ( function testfail () {
... ( function fail () {
..... throw new Error('failed!');
..... } )()
... } )()
Error: failed!
    at fail (repl:3:7)
    at testfail (repl:4:4)
    at repl:5:4
    at REPLServer.self.eval (repl.js:110:21)
    at Interface.<anonymous> (repl.js:239:12)
    at Interface.EventEmitter.emit (events.js:95:17)
    at Interface._onLine (readline.js:202:10)
    at Interface._line (readline.js:531:8)
    at Interface._ttyWrite (readline.js:760:14)
    at ReadStream.onkeypress (readline.js:99:10)
```

REPL?

More LISPy Schemey goodness:

Read-Eval-Print Loop

Back to the code...

```
// take an array and return a new array
// with 1 added to each array element
function plus1a (a) {
  if (a instanceof Array) {
    return a.map(function (n) {
      return (typeof n === 'number')
        ? n + 1
        : 0; // or null, or undefined
    });
  }

  throw new Error("numbers or arrays only!");
}
```

Simplistic Array.map

```
// assumes an array called a
map = function (fn) {
  var result = [],
      i = 0;
  for ( ; i < a.length; i++) {
    result.push(fn(a[i], i, a));
  }
  return result;
}
```

Ternary Conditional

condition ? <cond. true> : <cond. false>

Short form of

```
if (condition)
  then { var = 1; }
  else { var = 2; }
```

```
( n % 2 === 0) ? 'even' : 'odd'
```

Just like C, C++, Java, PHP, etc.

OK, better than C, etc

```
var someFn =
  (   browser.isIE
   && browser.version < 10)
? brokenBrowserFn
: workingBrowserFn;
```

Functions in Literals

```
var obj = {
  msg: "hello",
  things: [ 'a', 'b', 'c' ]
  nth_thing: function (n) {
    return this.things[n];
  }
};

> obj.nth_thing(1);
'b'
```

Object Inheritance

JavaScript supports multiple techniques:

- Pseudo-classical
- Prototypal
- Functional

Another coming in ECMAScript 6

Pseudo-Classical

```
var Mammal = function (name) {
  this.name = name;
};
Mammal.prototype.getName = function () {
  return this.name;
};
Mammal.prototype.says = function () {
  return this.saying || '';
}

> var c = new Mammal('cat');
> c.getName();
'cat'
```

Pseudo-Classical

```
var Cat = function (name) {
  this.name = name;
  this.saying = 'meow';
};
Cat.prototype = new Mammal();

> var norton = new Cat('norton');
> norton.says();
'meow'
```

Kinda messy

Can be improved with helper functions added to the Function class

see

"JavaScript: The Good Parts"

by Douglas Crockford

Classical Helpers

```
Function.prototype.method =
  function (name, func) {
    this.prototype[name] = func;
    return this;
  };

Function.method('inherits',
  function (Parent) {
    this.prototype = new Parent();
    return this;
  });
```

Classical with Helpers

```
var Mammal = function (name) {
   this.name = name;
}.
  method('getName', function () {
    return this.name;
  }).
  method('says', function () {
    return this.saying || '';
  });

var Cat = function (name) {
    this.name = name;
    this.saying = 'meow';
  }.
  inherits(Mammal).
  method('get_name', function () {
    return this.says() + ' ' + this.name + ' ' + this.says();
  });
```

Classical Output

```
> var norton = new Cat('Norton');
> norton.says();
'meow'
> norton.get_name();
'meow Norton meow'
>
```

Prototypal

```
var Mammal = {
  name : 'Herb the Mammal',
  get_name : function () {
    return this.name;
  },
  says : function () {
    return this.saying || '';
  }
};

var myCat = Object.create(Mammal);
```

Prototypal Output

```
> myCat.name = 'Henrietta';
> myCat.saying = 'meow';
> myCat.says();
'meow'
>
```

No Privacy!

```
> var norton = new Cat('Norton');
> norton.says();
'meow'
> norton.saying = 'MEEEOW!!';
> norton.says();
'MEEEOW'
>
```

Function Scope

JavaScript uses lexical function scoping

- not lexical block scoping like C

Only a single “level” of scoping within a function

C Scoping

```
#include <stdio.h>

int main() {
    int y = 2;
    printf("Before %d\n", y);
    {
      int y = 3;
      printf("During %d\n", y);
    }
    printf("After %d\n", y);
}
```

Output

```
% cc -o scoping scoping.c
% ./scoping
Before 2
During 3
After 2
%
```

JavaScript Scoping

```
function scoping () {
  var y = 2;
  console.log('Before', y);
  {
    var y = 3;
    console.log('During',y);
  }
  console.log('After', y);
};
```

output

```
> scoping();
Before 2
During 3
After 3
>
```

Inner functions

```
function get_fn () {
  var x = 2,
      fn = function () {
        return x;
      };

  return fn;
};
```

output

```
> var fn = get_fn();
> fn();
2
>
```

Functional Inheritance

Uses function scope to add privacy

Allows for private members

functions and variables

No “protected” or “friend” support

Functional “Class”

```
var mammal = function (spec) {
  var that = {};
   that.get_name = function () {
      return spec.name;
   };
   that.says = function () {
      return spec.saying || '';
   };
  return that;
};
```

Functional Inheritance

```
var cat = function (spec) {
  spec.saying = spec.saying ||
'meow';
  var that = mammal(spec);
  that.purr = function () {
    return ‘rrrrrrrrrrrrrrr’;
  };
  return that;
};
```

Functional Output

```
> var norton =
...cat({ name: 'Norton' });
> norton.says();
'meow'
> norton.get_name();
'Norton'
>
```

Private members

```
function taskList () {
   var  tasks = [],
        that = {
           add: function (taskName) {
              tasks.push(taskName);
              save();
              return tasks.length - 1;
           },
           get: function (taskId) {
              return tasks[taskId];
           }
        };

   function save() { console.log('Saving...'); }

   return that;
}
```

Private members

```
> var t = taskList()
> var id = t.add('Testing')
Saving...
> t.get(id)
'Testing'
> t.tasks
undefined
> t.save()
TypeError: Object #<Object> has no method 'save'
    at repl:1:4
    ...
```