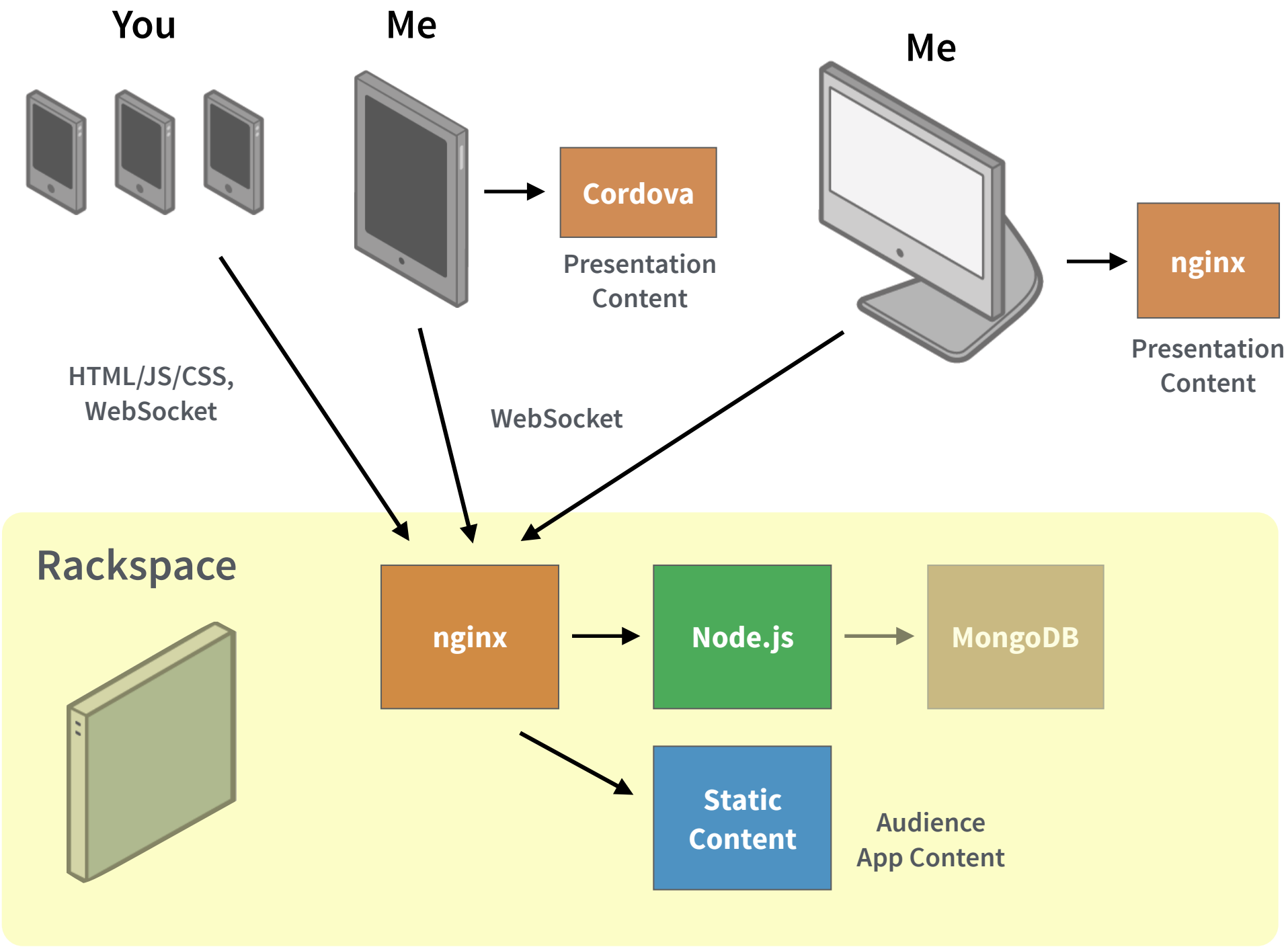
HTML5 Apps With Full-Stack JavaScript

Danny Brian

danny@brians.org

Goals for this session.

- Understand how the components of a full-stack JavaScript deployment plug together.
- Demonstrate the use of these components in a simple multiuser app.
- Show some key JavaScript patterns that help build maintainable web apps.


You
Me
Me
Cordova
Presentation
Content
nginx
Presentation
Content
HTML/JS/CSS,
WebSocket
WebSocket
Rackspace
nginx
Node.js
MongoDB
Static
Content
Audience
App Content

Full-stack JavaScript has three (popular) components:

1. A JavaScript-based client app — HTML5 running in a **browser or container**.

2. A JavaScript-based server — server code running on **Node.js**.

3. A JavaScript-based database — documents stored in **MongoDB**.

Why would you want this?

- You crave a simple, portable stack that's easy to get started.
- You want the benefits of the individual components.

- You like working with JSON, and think it would be nice to use it everywhere.
- You just love JavaScript.

- You want the benefits of a common language runtime.
- You might not.




Another way to think about full-stack JavaScript:

Attempts to bring alternative runtimes to the browser (plugins) largely failed.

The browser won the battle of the web runtime. It grew up.

So instead, we “gave in” and brought the browser technologies to the server.

Not client/server, client/*services*.

- In truth, the best way to build decoupled applications is using web APIs on the backend.
- This means we don’t necessarily need to think in terms of the “stack” anymore. “Three-tier is dead.”
- Many services, many clients. Reuse of components at both ends.

We'll dissect a simple multiplayer quiz.

- Multiple users, with buttons to vote, keeping track of time to answer and score.
- We will write this app top-down, starting with the client and mocking the rest.
- By working with events from the start, we will be able to replace the mocks without rewriting the handlers.

1. The Client

HTML*x*, CSS*x*, JavaScript *x*
on the browser (or web container)



There are hundreds of ways you can build an HTML*x* client.

- Talking here about a “single-page”, modern Web app architecture.
- Use a JavaScript framework like AngularJS, Ext JS, or Polymer.
- Use a cross-compiling framework.
- Write your own framework.
- Use vanilla JavaScript, perhaps with jQuery.
- HTML5 features aren’t technically necessary.



We will use a vanilla JS client.

- Why? Because this isn't a framework presentation, it's about the full stack.
- The client-side architecture shouldn't presume a server-side architecture — that's decoupling.
- The techniques for integrating client and services are similar regardless of the client-side frameworks, with some notable exceptions.



Localhost development.

```
% nginx -p .
% python -m SimpleHTTPServer
% python -m http.server
```



Dissecting the audience client.



Dissecting the presentation client.



Working with custom events.

(Use the browser!)

```
// create a custom event
var fired = document.createEvent("Event");
fired.initEvent("fired", true, true);

// fire the event
document.dispatchEvent(fired);

// handle the event
document.addEventListener("fired",
    function(e) {
        // do stuff
    },
false);
```



The client-side networking options.

- **XMLHttpRequest** — client-initiated, polling
- **Comet** — long-lived HTTP requests
- **WebSocket** — HTML5 full-duplex communication
- **Event Source** — server-sent events



We'll use WebSocket with some fallbacks.

- Best modern browser support among the options.
- Full-duplex, so it'll work for any use case.
- Not necessarily the best option for all use cases — don't abandon XmlHttpRequest or RESTful APIs without reason!
- An optimized use case for Node.js.
- Fallbacks, reconnects, etc. courtesy of socket.io.



The gist of our client socket.io implementation.

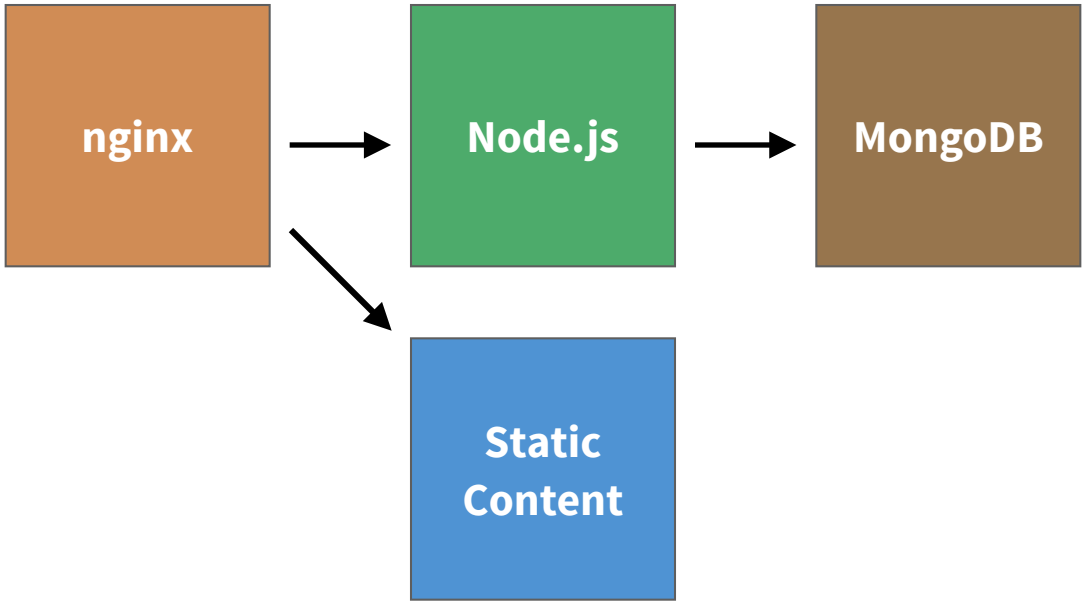
```
<script src="/socket.io/socket.io.js"></script>


var io = io();

io.on("msg", function(e) {
    document.dispatchEvent(fired);
});
```



2. The Server

JavaScript on Node.js

The breakdown of the server.



Working with npm and package.json.

The basics of event-driven programming.

- First-order functions
- Callbacks
- Closure
- Asynchronous patterns

The gist of our server socket.io implementation.

```
// initialize socket.io
var io = require('socket.io')(3000);

// handle a connection
io.on('connection', function(socket){
      console.log('a user connected');

      // handle an event
      socket.on('fired', function(fired){
          console.log('fired: ' + fired);
      });
});
```

```
var http = require('http'),
    path = require('path'),
    fs = require('fs'),
    base = '/home/examples/public_html';

http.createServer(function (req, res)
{
    pathname = base + req.url;
    path.exists(pathname, function (exists)
    {
        if (!exists)
        {
            res.writeHead(404);
            res.write('Bad request 404\n ');
            res.end();
        }
        else
        {
            res.setHeader('Content-Type', 'text/html ');
            // 200 status - found, no errors
            res.statusCode = 200;
            // create and pipe readable stream
            var file = fs.createReadStream(pathname);
            file.on("open", function ()
            {
                file.pipe(res);
            });
            file.on("error", function (err)
            {
                console.log(err);
            });
        }
    });
}).listen(8124);
```

Numbers Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

Rounding out the features.

(Live coding.)

3. The Database

JavaScript/JSON on MongoDB

The gist of our MongoDB integration.

```
var mongodb = require('mongodb').MongoClient;

mongodb.connect('mongodb://localhost:27017/quiz',
  function(err, db) {
    console.log('Connected correctly to server');
    var users = db.collection('users');
    users.update(
        { id : user.id },     // match these docs
        user,                 // set this
        { upsert: true },     // insert if no matches
        function (e, r) { } // callback
    );
  }
);
```

Don’t persist just for the sake of persistence.

Back up to touch on best practices.

- Testing on the server with assertions.
- Modular code on client and server.
- Real data modeling with Mongoose.
- Asynchronous patterns.
- Authentication and authorization.

Some discussion of so-called
“no backend” solutions.

Some discussion of Meteor.



Source code for the demos.

https://github.com/dannybrian/nfjs-fullstack

https://github.com/dannybrian/appresent

http://g4tp.com

http://g4tp.com/2014/html5/