Chapter 4: Case Study in Pure JavaScript Development

AtWood's Law: Any application that can be written in JavaScript, will eventually be written in JavaScript. – Jeff AtWood, Coding Horror

JavaScript has come along way since May, 1995. In 2013, the language boasts high quality libraries, tools and extensions that make it expressive, powerful and fast. With JQuery and other front-end libraries, developers can quickly write complex Ajax applications. With NodeJS, developers have a robust JavaScript implementation that runs on the server. The developments of the last decade put the idea that JavaScript is a toy language into question. The language's progress raise a number of questions. Is JavaScript good enough to write *real* applications in 2013? All Web applications contain a JavaScript component, but is JavaScript good enough to run the whole stack?

In order to test this question, and push the limits of modern JavaScript, I joined a team of software developers who had an idea for an application written completely in JavaScript. The Exavault team, based on Oakland, builds applications to help large businesses share large files internally using the FTP protocol. For three months, Exavault's five person development team joined me one day a week to build NextFive, a collaboration tool written entirely in JavaScript. By the end of my three months with the team, we were able to launch a beta version of the application good enough to be used internally; NextFive development at Exavault is ongoing.

Project requirements.

NextFive is a tool designed to solve a real problem that Exavault's software development team faces on a daily basis. Exavault's staff work remotely. Although the company is based in Oakland, only two employees regularly work from that location. Most employees are scattered throughout the West coast, including some in Portland and Los Angeles. Exavault's way of doing business in increasingly common in the modern world, especially among software development teams. Advances in version control systems for code management and online meeting tools like Skype have allowed teams to work together from separate parts of the world. But an online-only office environment sometimes causes inefficiencies; how can a manger and colleagues keep track of what people are working on?

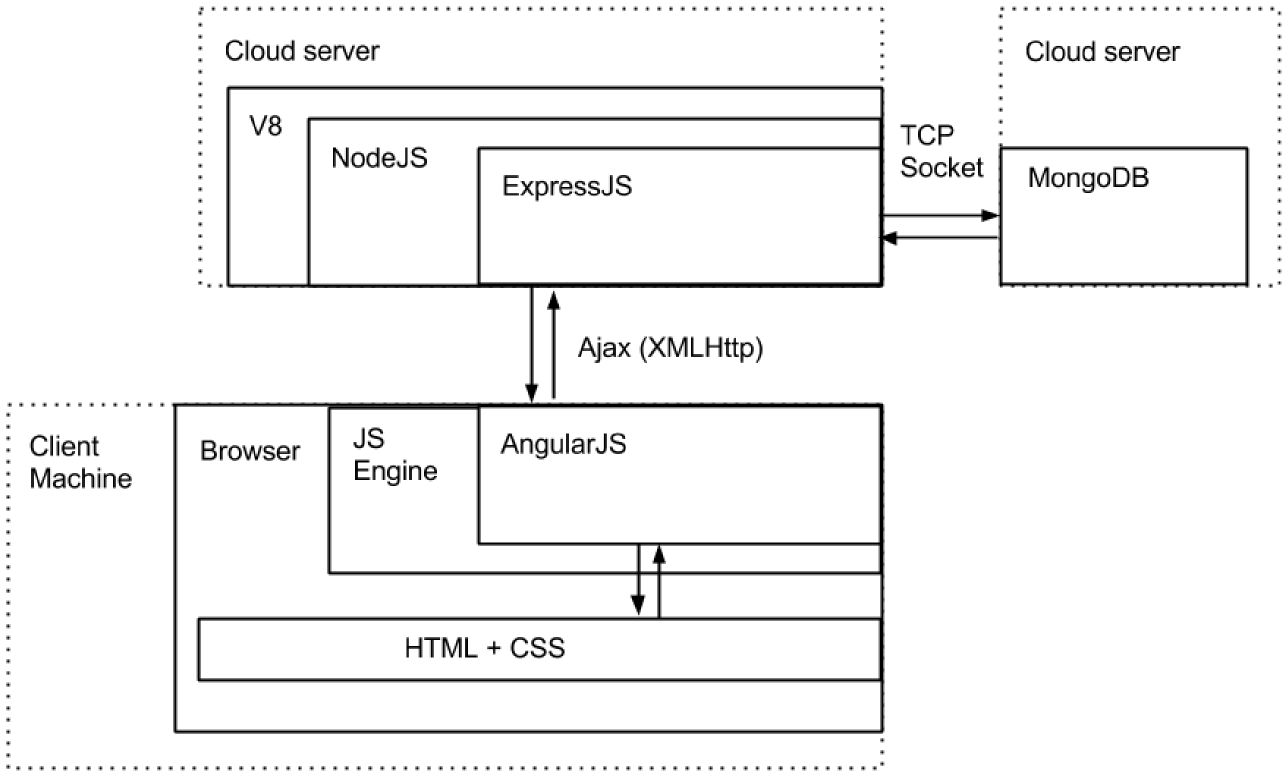
Exavault solves this problem with email reports. At the beginning of each week, every employee writes a short report detailing what they plan to accomplish each day that week and shares it with the rest of the team. The employee's manager and colleagues then have a chance to review her report and get a sense for what she will be working on. This increases the chances that inconsistencies will be caught early and the team's workload for the week will be known to all. NextFive is a tool designed to make this reporting process smoother, for Exavault today and perhaps for other teams in the future. Instead of writing email reports, users of NextFive can write reports in a Web interface, compare reports and keep track of reports over a longer period of time. In order to accomplish its goals, NextFive had to support a few key features:

- Secure login for employees.
- A report writing interface.
- A report browsing and reading interface.
- A report comparison interface.
- Email integration so that reports could be sent to the rest of a team.
- Support for differing roles within a company (for instance, employee and manager).
- Support for different teams within a company (for instance, Engineering and Marketing).
- An interface for changing the settings for a user, team or company.

In September we set out to build a beta version of NextFive supporting all of these features in three months. And of course, we did it all in JavaScript.

Technology stack.

After a number of discussions for the first two weeks, we eventually settled on using a series of JavaScript technologies that together have been called the “MEAN Stack.”[1] MEAN stands for MongoDB, ExpressJS, AngularJS and NodeJS. MongoDB is a popular “NoSQL” database technology that allows for inserting and retrieving data in JavaScript Object Notation (JSON), ExpressJS is a simple Web framework for NodeJS that handles the boilerplate code for writing a Web server and AngularJS is a browser JavaScript framework developed by Google designed for programming interfaces using the Model-View-Controller (MVC) pattern. And of course, NodeJS is an implementation of JavaScript designed to run on a server. This following is a rough sketch of NextFive's architecture using the MEAN stack:



The Angular application manages all user input and all HTML views, including rendering new views in response to user input. When more data is required, be it a snippet of HTML or a report object, or when data should be updated in the database in response to user input, Angular is responsible for making requests to the backend Express application. In turn, Express is responsible for responding to Angular's requests by creating, updating or fetching data in the Mongo database. Express is responsible for handling user authentication and serving the Angular application to the browser, but once the application has been initialized, the Angular application drives the show: Express acts as little more than an interface for Angular to interact with backend data through HTTP requests.

1 http://blog.mongodb.org/post/49262866911/the-mean-stack-mongodb-expressjs-angularjs-and

Understanding the event loop.

Writing backend JavaScript code is very different from writing JavaScript code meant to run in the browser, but strangely, it is also very different from writing other server-side languages. NodeJS is designed so that all input and output is done asynchronously with an event loop. This means that the program does not do any waiting for program input, but it also forces the programmer to adopt an entirely new approach to backend code. The best way to understand this is through a comparison with another language. The following Python code connects to a database and prints the results of a simple query:

```
import sqlite3

db_cursor = sqlite3.connect('test.db').cursor()
db_cursor.execute('SELECT SQLITE_VERSION()')
data = db_cursor.fetchone()
## By the time we reach this line, the database query is complete
print "SQLite version: %s" % data
```

This code works, but it has some hidden costs. A program can retrieve data from a register and perform a computation in 1 CPU cycle, but if the data is not already in a register, some amount of time must be spent retrieving it. It costs time equal to 3 CPU cycles for a program to go to the L1 cache to fetch data. To go to the L2 cache it costs about 14 cycles; its 250 to go to RAM, 41 million to go to disk and 250 million (or more) to fetch data over the network.[2] On line 5, when the Python program is executing the database query, the program is completely halted. The program does not execute the print statement on the final line until the database query is done and the 'data' variable has been assigned. In many ways, this is convenient for the programmer: we can treat assigning variables to the results of database queries as any we would any other kind of variable assignment. But if the query takes a billion CPU cycles to complete, that's a billion calculations that the program is not doing. From a human perspective, a billion cycles is a fraction of a second, but from the programs perspective it is spending a lot of time waiting.

In NodeJS, input and output is done asynchronously.

```
var user = database.findOne({email: "brendan@mozilla.org"});
console.log(user);
```

jfjfj

```
// All input and output must be done in a callback.
User.findOne({email: "brendan@mozilla.org"}, function(error, user) {
  console.log(user);
}
// By the time we reach this line, the database query i
console.log("This will get printed before the user object.");
```

One language, one data format. (1 page)

2 http://blog.mixu.net/2011/02/01/understanding-the-node-js-event-loop/

Minor components and tools. (1 page)

- NPM
- Grunt
- JSHint
- PassPort
- Mongoose
- JQuery
- Node/ExpressMailer

Conclusions. (½ page)

Caveat: testing a giant scalable system was not possible

- Conclusions
  - Advantage of writing *one language* is huge, especially when working on both frontend and backend
    - Actually blurs the line between frontend and backend somewhat
      - Cuts down on interface code between them
      - Data is represented exactly the same way in Mongo, Express, Angular
      - No context switching for full-stack programmer
  - But does that language have to be JavaScript?
    - Other dynamic typed languages are probably just as good for 95%
    - Callback drawbacks
      - Can be confusing to read / write – harder to write really clean code
      - The concept of the event loop can be difficult to grasp
    - We had to use JS anyway: interfacing with HTML
      - Angular has a steep learning curve, but serious advantages
      - However, this is sort of a cop out point. If python could interface with HTML as well, it might be an excellent choice
  - Modern tools and frameworks are great for Web development
    - The hardwork of Web development is dealt with
    - NPM is an excellent package manager
    - Grunt is an excellent build tool
    - Express hides all the nasty parts about writing a web server
    - Heroku → using git to push to the scalable cloud
    - Because the stack is so new, the people building the tools are excellent
      - less battle-tested however