A PLACE FOR NODE.JS IN THE COMPUTER SCIENCE CURRICULUM [*]

Scott Frees
Department of Computer Science
Ramapo College of New Jersey
Mahwah, NJ 07430
201 684-7726
sfrees@ramapo.edu

ABSTRACT

While there are many server-side frameworks available, node.js is unique in its ability to help solve well-recognized challenges in teaching web development. Using node.js allows for the consolidation of language throughout the application stack. The platform supports a smooth learning curve by allowing students to build knowledge gradually through the use of modular, open source components. This paper details the advantages of using node.js in an undergraduate web development course and examines its value to the CS curriculum as a whole.

1. INTRODUCTION

Web development in the undergraduate CS curriculum presents significant challenges. Web development requires students to gain knowledge in up to five different languages before creating web applications of significant complexity: HTML, CSS, JavaScript, a server side language such as PHP, and SQL. The number of new languages and API's can prove challenging for students, and can limit an instructor's ability to cover important topics [1, 2].

While largely categorized as "elective", the ACM 2008/2013 curriculum standards incorporate many web-related topics. These recommendations do not specify which languages (particularly server-side) should necessarily be taught however. As pointed out by Connelly [3], web development is not prominently featured in CS education

[*] Copyright © 2015 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

research, and many departments have found it difficult to adequately cover the topic. It is interesting to note that non-traditional CS education, such as offerings from code.org and others, include web development prominently. Perhaps these services are gaining popularity because they are filling a gap in the typical undergraduate CS curriculum.

This paper proposes node.js - a platform for developing server-side web logic in JavaScript - as a way to overcome many challenges in teaching web development. The consistent use of JavaScript on both the server and client allows students to acquire more knowledge depth than time would otherwise allow. Node.js's architecture and philosophy lends itself to teaching, as the internals of web development (sockets, HTTP, templates, etc.) are not as hidden from the student as they are in more heavyweight frameworks. The use of node.js also introduces important core concepts such as event-driven programming and functional programming, which reinforce other parts of the curriculum.

2. TEACHING WEB DEVELOPMENT - BACKGROUND

While students who have taken previous CS coursework pick up HTML and CSS quickly (at least the basic parts), developing skills in JavaScript and a server side language - all in the same semester - can be a tall order. Learning these new languages is on top of acquiring an understanding of networking and the HTTP protocol, as well. Courses attempting to cover the entire breadth of web development are often limited to “surface-level” treatment of each topic due to practical time constraints, leading to an unsatisfying experience.

One solution to this problem is to split web development into two separate courses - allowing instructors to dedicate more time to specific topics. This strategy, proposed by several [3, 4], has advantages; but it is challenging for many departments to adopt because of curricular constraints.

Another way to improve is through the careful choice of languages and topics covered. Stepp et. al [1] has published informal survey results which indicate HTML/CSS and JavaScript are covered in virtually every course covering client-side development – their dominance on the client-side leaves little choice in adoption. There is little consensus on the server however, with PHP, Java/JSP, Ruby on Rails, ASP.NET, and Python being common choices among educators.

3. CHOOSING SERVER-SIDE LANGUAGES AND FRAMEWORKS

There are many excellent choices of server-side platforms available to both professional web developers and students. When deciding which to use for teaching undergraduates, the author proposes three qualities as important considerations.

- **Language Competency:** The student should already know, or be able to quickly understand, the underlying language used (i.e. Ruby when working with Ruby on Rails).
- **Smooth Learning Curve:** The learning curve to become productive with the full platform should be smooth, allowing students to work gradually up from lowlevel

considerations such as networking and HTTP to higher-level concepts such as MVC.

- **Transferrable Skills:** Skills acquired while learning the server-side platform should be transferrable, and reinforce other aspects of the curriculum.

The Java world is dominated by the likes of JSP, Spring, and several other enterprise frameworks. Java's ecosystem is rich, however the learning curve can be very challenging. On the Microsoft platform, the use of C# in ASP MVC is currently a favored approach by many [5]. In comparison to Java, the ASP MVC framework can offer a more forgiving learning curve - however the size, scope, and rate of change of the .NET platform can cause difficulty in the undergraduate setting.

While PHP bears strong resemblance to C syntactically, it does require specific instruction. Unlike other scripting languages, PHP has lower residual value to the student, as its use is largely isolated to web development. While framework use is critical to productivity, teaching PHP frameworks such as those surveyed in[6, 7] can hide lower level concepts of web programming from the student.

Ruby is a general-purpose language - however as any C/C++/Java developer who has made the transition to Ruby can attest - the language is quite different than Cstyle languages typical of introductory courses. Students must spend significant time learning the language itself. The Ruby on Rails framework dominates web development in Ruby, and the framework is quite large and complex. In addition, Rails can be an impediment to learning web fundamentals, as there is a huge jump in level of abstraction from socket and HTTP level programming and the MVC and scaffolding paradigm at the core of Rails.

4. NODE.JS – A NEW OPPORTUNITY

Node.js, released in 2011 by Ryan Dahl, is an open source platform that packages the Google V8 JavaScript engine and a core set of JavaScript modules designed to support interaction with the host OS. Google's V8 engine is an open source C++ implementation of the JavaScript runtime, developed as part of the Chrome web browser. Importantly, V8 uses JIT compiling to achieve near-native performance, vastly outperforming traditional interpreter-based approaches.

What makes node.js unique (aside from its use of JavaScript) is it's handling of I/O. Multi-threaded web servers typically dedicate a thread to each browser connection to avoid blocking calls made by one from slowing other connections. Capacity is often challenging with this approach however, due to the resources required for maintaining many threads. Node.js serves all requests in a single thread, avoiding blocking entirely by exposing only asynchronous APIs for I/O functionality. All HTTP requests are served out of the same thread, as part of an event loop that handles the execution of all callbacks. The design accommodates massive concurrency on I/O bound applications.

From a pedagogical perspective, node.js is distinct in its ability to provide language consistency (JavaScript) across all facets of a course. In addition, it offers a unique blend of high-level abstraction and exposure to low-level detail. As shown in Figure 1, the simple creation of a HTTP server is trivial using the node.js core library. While the library abstracts away the TCP sockets and pesky HTTP request parsing, the details of response codes, headers, as well as the mechanics of dealing with the output stream are not hidden from the programmer.

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
 res.end("<!doctype html><html><body><p>Hello World</p></body></html>");
}).listen(3000, '127.0.0.1');
```

Figure 1 - A simple http server using node's http module to respond with a default HTML page for all requests. The lambda function provided to the createServermethod is called with associated HTTP request and response object whenever the server receives an incoming HTTP request.

5. ADVANTAGES OF TEACHING WITH NODE.JS

Node.js, and its associated frameworks (notably, connect [8] and Express [9]) compares favorably to other platforms in the criteria introduced in Section 3.

5.1. Language Competency

JavaScript is a C-style language that while vastly different "under the hood" from C and C++, is syntactically familiar for most students - in contrast to languages like Ruby and Python. More importantly, JavaScript is a required language for a web course because of its client-side importance. Re-using JavaScript on the server is helpful, as typically the cognitive load associated with switching between languages for the browser and the server slows learning and reduces the amount of depth a course can cover a language in.

5.2. Smooth Learning Curve

The typically large gap between low-level network/HTTP protocol programming and the use of server side frameworks can be an impediment to learning. Node.js offers a refreshing alternative to this - it is easy to gradually add concepts such as query/form processing, cookie and session management, templates, and routes gradually over time - without adopting a "heavy" framework all at once. In addition, nearly all facets of server-side development can utilize JavaScript - including templates using Embedded JavaScript (EJS) and interaction with the JSON based databases (i.e. MongoDB). With each step toward greater complexity, students continue to rely on the JavaScript skills they have built throughout the semester.

For each additional concept, the instructor can replace previous "custom" solutions with components from connect or Express. Using this strategy, students can implement solutions to problems such as reading posted data themselves, yet eventually work with examples leveraging professional-grade frameworks.

```
qs = require('querystring');
//.. creation of http server ...

function process_post(req, res) {
  var body = "";
  req.on('data', function (chunk) {
    body += chunk;
  });
  req.on('end', function() {
    }
  });
}
```

```
var connect = require('connect');
  var app = connect().use(
    connect.bodyParser());
//.. further connect configuration and
//      creation of http server

function process_post(req, res) {
  for ( q in req.body ) {
    console.log(q + " -> " + post[q]);
  }
}     var post = qs.parse(body);
    for ( q in post ) {
      console.log(q + " -> " + post[q]);
```

Figure 2 - The code on the left utilizes the HTTP request's data stream directly to capture the posted data as a string. The right side uses connect to parse the request body more robustly.

5.3. Transferrable Skills

When used in small browser scripts, students are not necessarily exposed to advanced concepts such as JavaScript's prototypical object/class implementation, closures, and the event-driven nature of many of its libraries. Using node.js on the server allows a much greater amount of time to be spent learning these concepts, which ultimately builds better client-side skills as well.

5.3.1. Functional Programming and Closures

While JavaScript has a C-like syntax, it has much in common with functional languages such as Lisp. While not a "pure" functional language, JavaScript supports functions as first class objects, it supports lambdas, and implements closures. An introduction to these concepts can serve as either a powerful reinforcement of, or a gentle introduction to, the types of concepts commonly covered in an organization of programming languages course.

5.3.2. Asynchronous I/O and Event-Driven API's

When using AJAX client-side, students must quickly get accustomed to programming in the event-driven / asynchronous style by registering functions as handlers to be called when events (such as the arrival of data from the HTTP server arrives at the browser) occur. For students accustomed to procedural, commandline programming - this can be rather jolting. Once again, node.js reinforces this concept; forcing the students to think in terms of event-driven, asynchronous, and streaming API's on the server side as well - even when performing common operations such as reading files from the local disk.

```
var fs = require('fs');
var stream = fs.createReadStream('foo.txt')
```

```
stream.on('data', function (data) {
  console.log("Read " + data.length + " bytes");
});
stream.on('end', function () {
  console.log("Read entire file...");
});
console.log("Stream opened");
```

Figure 3 - Reading a file asynchronously. Note - the words "Stream opened" will appear on the console immediately after creating the stream - followed by the invocation of the appropriate handlers as data becomes available on the stream.

5.3.3. Applications Beyond the Web

With the advent of WebGL (an implementation of the OpenGL ES 2.0 standard in JavaScript) JavaScript can be used to create stunning 3D games and simulations within the browser. The popular MongoDB uses a JavaScript / JSON syntax for all of its administration and querying facilities. In both cases, achieving competency in JavaScript allows a student to quickly become productive with these growing technologies. Node.js also heavily embraces the Unix style of programming – encouraging small, interchangeable modules. A close look at most node.js programs will reveal concepts familiar to students studying Unix, such as piping input and output streams (files, sockets, stdin, etc).

6. A COURSE OUTLINE

In a 300-level web development course at Ramapo College, a "server-side first" approach is used. Initial focus is on learning underlying networking programming and the organization of the HTTP protocol. The JavaScript language is taught in "waves", initially students are exposed to just enough to begin testing HTTP. There were two required textbooks [10][11], covering HTML, CSS, and JavaScript language. Two optional textbooks on node.js [12][13] were encouraged.

Weeks 1-2 - Networking and HTTP basics

A rudimentary HTTP server is built with TCP to handle GET and POST requests and respond with headers and static html files on disk using node.js's fsmodule. The use of the fsmodule offers a very brief introduction to asynchronous programming.

Weeks 3-4 - Server-side routing and control flow

Node's HTTP server replaces the functionality built in class and students begin to build more robust server-side logic. The course begins to focus more heavily on the core JavaScript language - providing the students with the skills to build their own routing and control flow on the server.

Weeks 5-7 - HTML and Templates

Students create more robust content by learning the details of HTML and forms. Embedded JavaScript (EJS) is introduced through the ejsmodule, the installation of which offers the opportunity to explore node.js's built-in package management system - npm. Cookies and Sessions are introduced, using the connectmodule.

Weeks 8 - Presentation with CSS

All of CSS is covered over the space of 3-4 lectures, including rule-cascading, layout, and CSS3 concepts.

Weeks 9-10 - Client-Side JavaScript with jQuery

Client-side programming is covered primarily with jQuery. These lectures can largely focus on DOM manipulation and HTML event handling, as students have a firm grasp of the core JavaScript language. JSON is formally introduced (students have already grown accustom to object literals in JavaScript) through AJAX. XML is only briefly discussed, keeping in line with the industry trend towards JSON [14].

Weeks 11-12 – Routes, REST, and Authentication

Server-side functionality developed over the course of the semester is reimplemented in the *Express framework.* Routes are introduced to show students a modern example of program organization and to cover REST. The HTTP authorization lessons include TLS/SSL and password hashing.

Week 13 - Using a Database

Rather than introducing SQL as yet another language, the freely available MongoDB is presented instead. The MongoDB query language is JavaScript, and result sets are returned through asynchronous JavaScript callbacks as JavaScript object literals.

Week 14 - Overview of other Technologies

Overviews of Ruby on Rails, ASP.NET MVC, and PHP are presented as server-side alternatives. CSS pre-processors (SASS and LESS) are introduced along with the popular front-end MVMM framework, AngularJS.

7. REFLECTIONS AND CONCLUSIONS

The author has taught web development at Ramapo College six times, using ASP.NET or Java previous to 2014. The migration of the course to node.js has been eye opening, allowing instruction to proceed from topic to topic in a much more cohesive and smooth manner because of the common JavaScript footing. Its debatable if node.js is the always best choice at the professional level, however it was clear JavaScript's ubiquity provided a level of consistency simply not possible when adopting other server-side languages.

Not only were more topics able to be covered, student projects also reflected a better mastery of topics. Also striking was the absence of student difficulty when it came to setting up their development environments and servers - as the node.js platform is trivial to install and get up and running on any operating system.

REFERENCES

[1] Stepp, M., Miller, J., Kirst, V. (2009) A “CS 1.5" introduction to web programming, *Proc. of the 40th ACM technical symposium on Computer science education*, Chattanooga, TN, USA

[2] Adams, D. R. (2007) Integration Early: a New Approach to Teaching Web Application Development. *J. Comput. Small Coll.* 23(1), 97-104.

[3] Connolly, R. (2012) Criticizing and modernizing computing curriculum: the case of the web and the social issues courses. *Proc. of the Seventeenth Western Canadian Conference on Computing Education*. ACM, New York, NY, USA, 52-56.

[4] Wang, X., McKim, J.C, (2013). The opportunities and challenges to teach web programming in computer science curriculum CS2013. *J. Comput. Sci. Coll.* 29(2), 67-78.

[5] Gupta, P., Mata-Toledo, R. (2011) Utilizing ASP.NET MVC in Web Development. *J. Comput. Sci. Coll*, 27 (3), 10-14.

[6] Chao, J., Parker, K. & Davey, B. (2013). Navigating the Framework Jungle for Teaching Web Application Development. In E. Cohen & E. Boyd (Eds.), *Proceedings of Proceedings of the Informing Science and Information Technology Education Conference* 2013

[7] Lancor, L., Katha, S. (2013) Analyzing PHP frameworks for use in a project-based software engineering course. *Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13)*. ACM, New York, NY, USA, 519-524.

[8] Connect - High quality middleware for node.js. http://www.senchalabs.org/connect/

[9] Express - node.js web application framework. http://expressjs.com/

[10] Larson, R. (2013) Beginning HTML and CSS. Wrox.

[11] Flanagan, D. (2011) JavaScript - the definitive guide - 6th Edition. O'Reilly Media.

[12] Tiexeira, P. (2012) Professional Node.js: Building Javascript Based Scalable Software. Wrox.

[13] Cantelon, M., Harter, M., Holowaychuk, T.J., Rajlich, N. (2013) Node.js in Action. Manning Publications.

[14] D. Crockford (2006) JSON: the fat-free alternative to XML. *Proceedings of XML 2006*, http://www.json.org/fatfree.html