HTML And Node.js Development System (HANDS) An approach for web development

Brian Carter, *Chippewa Software Technology*

***Abstract*— In this paper, a very compelling proposition is presented, an approach for web development that is a hybrid of Single Page Applications (SPA) and server side applications. The proposed architecture provides a simplistic approach for development. The focus is on HTML, JavaScript, Cascading Style Sheets, and Node.js. The architecture provides all the components required for students to start developing web applications and web services. The HTML And Node.js Development System (HANDS) is a hybrid approach leveraging the simplicity of plain old HTML pages, the AJAX injection of HTML from SPA frameworks, and the server-side processing found in the Node.js framework. An open source starter kit and examples are provided.**

***Index Terms*—education, framework, web development, html, javascript, node.js, sqlite**

I. Introduction

Web development has become one of the most in-demand skills. To meet this need, the addition and retention of students in this field is critical. Web development has a steep learning curve and many obstacles. One obstacle is getting students developing HyperText Markup Language (HTML), JavaScript (JS), and Cascading Style Sheets (CSS) applications that include server side processing.

The typical approach to web development training includes a brief introduction to HTML. Very quickly courses introduce an online HTML online website generator or a framework that hides the complexities of front-end and backend (server side) development. Often this is to give students confidence that they can create complex web applications. The downfall of this approach, students learn the details of the framework and not an in-depth understanding of the foundational elements of web applications. As shown in figure 1, the primitives of HANDS include HTML, JS, CSS, and Node.js.

This is compounded when the student enters into a corporate software development position. The majority of companies have a well-established framework (open source or propriety) for development. This has formed a void in the in-depth understanding of the fundamentals. Often, the simple request and response concept is not understood. This is especially apparent in software development shops that have rich libraries that wrap web functionality and where the development is strictly divided between client-side and server-side resources.

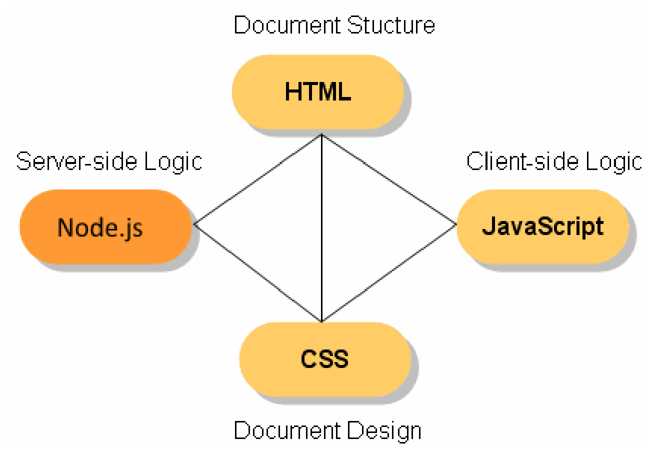


Fig. 1. HANDS primitives.

After developing HANDS, the question was asked: If the primitive, foundational elements can deliver any type of web application, why introduce libraries that hide functionality and increase complexity? The position of this paper is to build an architecture where the focus is the HTML page while only introducing libraries after understanding the concepts delivered in such libraries. With this approach, when issues occur later in the development phase, the developers have the foundation to find and understand the root problems and provide solutions.

In this paper, the focus is on HTML, CSS, and JS. A basic website is created and studied. An understanding of the basic elements are required before moving on. The goal is to propose HANDS as an approach to learning web development and to provide a foundation for enterprise development.

II. HyperText Markup Language

HTML, short for HyperText Markup Language, is the authoring language used to create documents on the World Wide Web (WWW). Initially, HTML was designed to publish documents, much like one would publish a journal of articles. HTML defines the structure and layout of a Web document by using a variety of tags and attributes. A web browser can read HTML files and compose them into visible web pages. The

This work was supported in part by Sullivan University and Chippewa Software Technology.

Brian Carter is with Sullivan University, College of Information and Computer Technology, Louisville, KY 40205 USA (e-mail: bscarter@sullivan.edu) and is with Chippewa Software Technology, La Grange, KY 40031 USA (e-mail: briancarter@chipsofttech.com).

browser transforms the tags into content by interpreting the tags and attributes to display the page. HTML describes the structure of a website semantically along with cues for presentation, making it a markup language rather than a programming language. The first publicly available description of HTML was a document called "HTML Tags", first mentioned on the Internet by Berners-Lee in late 1991 [1, 2].

HTML elements form the building blocks of all websites. HTML allows images and objects to be embedded and can be used to create interactive forms to post data back to the server. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. Figure 2 shows the primitive elements of an HTML page along with a few common tags. The example provides the most basic template for HAND using 9 lines of code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>This is HANDS</title>
  </head>
  <body>
      <p><b>HANDS</b> basic template</p>
  </body>
</html>
```

Fig. 2. HANDS basic HTML template.

The text between <html> and </html> describes the web page, and the text between <body> and </body> is the visible page content. The markup text *'<title>This is HANDS</title>'* defines the browser page title that shows up in the tab. The tag <p> is for a paragraph and is displayed on the page. An online tool to share code examples and provide an easy starting point for students to practice is jsfiddler.net. The code in figure 2 has been placed in a fiddle for readers to run: http://jsfiddle.net/briancarter/fbqn0kou/.

HTML documents imply a structure of nested HTML elements. These are indicated in the document by HTML tags, enclosed in angle brackets: <p>. In the example, the extent of an element is indicated by a pair of tags: a start tag <p> and an ending tag </p>. The text content of the element, if any, is placed between these tags. Tags may also enclose additional tags, including a mixture of tags and text. An example is the bold tag <b> which is included inside the paragraph tag <p>. This indicates further, nested, elements, as children of the parent element. A full tutorial on HTML is beyond the scope of this paper. A good reference is w3schools.com[3] for additional tutorials and examples.

III. JAVASCRIPT

JavaScript (JS or js) is a dynamic computer programming language [4]. It is added to an HTML page and used by the web browser to implement client-side (in the user's browser) interaction, control the browser, communicate asynchronously, and alter the HTML document content that is displayed. It is also used in server-side network programming with frameworks such as Node.js. In figure 3, the example shows how to embed scripts written in the JS language which affect the behavior of HTML web pages.

```
<!DOCTYPE html>
<html>
    <head>
        <title>This is HANDS</title>
    </head>
    <body>
        <p><b>HANDS</b> basic template</p>
        <p id="p1"></p>
<script>
  document.getElementById("p1").innerHTML =  "Hello from HANDS!";
</script>
    </body>
</html>
```

Fig. 3. HANDS basic template with JavaScript.

As shown, JS is added to the HANDS template by including the <script> tag. The document references the HTML page. GetElemementById references the tag with the id of "p1" which is the <p> tag. This basic JS example shows the fundamentals on how an HTML page can be manipulated. This is a very powerful concept and it is delivered without the need for a framework. See working example: http://jsfiddle.net/briancarter/bsxnx3qL/. There are many tutorials and examples on the web [3] for in-depth study of JS.

IV. CASCADING STYLE SHEETS

Web browsers can also use Cascading Style Sheets (CSS) to define the look and layout of the HTML page. Think of CSS as the artist of web development; changing the color of elements, the font sizes, and the layout. The W3C, maintainer of both the HTML and the CSS standards, encourages the use of CSS [5]. CSS is designed primarily to enable the separation of document content from document presentation, including elements such as the layout, colors, and fonts [6]. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple HTML pages to share formatting by specifying the relevant CSS in a separate .css file, improve team dynamics by allowing separation of responsibilities, reduce complexity and repetition by reuse of common presentation formats, and reduce maintenance change impacts.

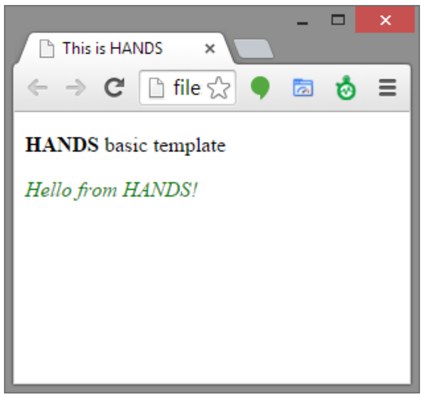
Extending the example from figure 3, the use of CSS is shown in figure 4. The <style> tag is included in line to change the color of the hello paragraph. There are many features and functions for CSS and additional training material can be found on the web [3].

```
<!DOCTYPE html>
<html>
  <head>
    <title>This is HANDS</title>
    <style>
      p.hello {color:green;font-style: italic}
    </style>
  </head>
  <body>
    <p><b>HANDS</b> basic template</p>
    <p id="p1" class="hello"></p>
<script>
 document.getElementById("p1").innerHTML =  "Hello from HANDS!";
</script>
  </body>
</html>
```

Fig. 4. HANDS basic template with JavaScript and CSS.

V. Front-end development

In previous sections, the fundaments of web development were briefly covered (HTML, JS, CSS). These items are often referred to as front-end development; front-end referring to items that run in the users browsers and provides the user interface experience. Figure 5 shows the example page.



HANDS basic template

Hello from HANDS!

Fig. 5. HANDS basic template as shown in web browser.

For optimization and for separation of concerns, the JS and CSS code is typically consolidated into separate external files. This allows the browser to cache the content more effectively. This also separates concerns have the three fundamental aspects of web development in their own maintainable files. For HANDS, the approach is to keep the code together and as close to the area of responsibility as possible. This provides the students a single file to understand the elements and how they interact. Once the students understand this concept, the best practice of separation of individual files can be applied.

VI. Back-end development

Server-side (commonly referred to as back-end) refers to operations that are performed by the server in a client–server relationship. Where the client is the HTML page and the server is a web server that runs on a remote server, reachable from a user's browser. Operations are performed server-side because the program requires access to information or functionality that is not available directly to the client, only available on other secure network servers.

Server-side operations also include processing and storage of data from the browser to a server. If data is entered on the browser, the JS on the page can call a web service on a server to store the information. This allows for information to be shared among multiple users, provides for scalability (servers have more resources), lightens the work load on the local browser, provides backup and restore abilities of the data, and is more secure. There are several platforms for creating web applications and web services on the server. For HANDS, we leverage the training system HENS [8] to provide the infrastructure and development tools for writing Node.js and SQLite web applications. Follow the instructions to setup HENS or setup your own Node.js environment. HANDS will run on any Node.js environment.

Node.js is an asynchronous event driven framework for server side processing. It is designed to build scalable network applications. With Node.js, web sites and web services can be built that expose data from multiple sources. This provides our frontend applications with information to display.

HANDS is a Node.js application. The high level design is shown in figure 6. As shown, the main application file is app.js. This file is responsible for starting the server and responding to all request. It has a router that will route service calls to an API handler or server the assets. The API services are restful, accepting and replying using JavaScript Object Notation (JSON). All other calls are considered assets or page calls, where HANDS will return the appropriate content. An important concept, is HANDS will process all requests. If a page requests an HTML, JS, CSS, or image file, HANDS will process the request and stream the file to the user's browser.

For this paper, a simplified version of HANDS is presented. This simplified view is the foundation and must be understood by the reader. A simple front-end page is created to call the backend application which returns JSON data. The full enterprise framework is open source and can be studied in detail [9].

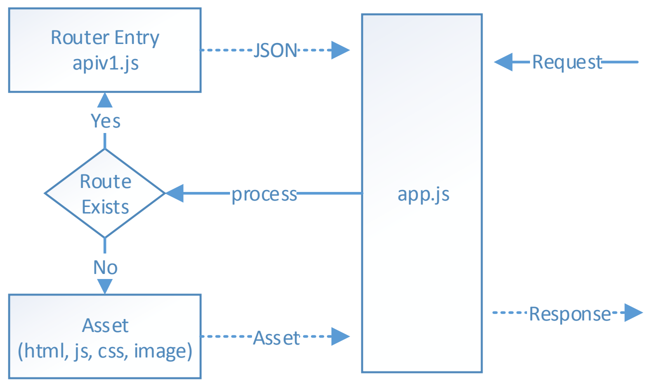


Fig. 6. HANDS basic template as shown in web browser.

The back-end application is shown in figure 7. In the Node.js code, a server is created and listening on localhost port 8080. When the server receives a request, it looks at the url to determine if it is the API call (/api/v1/teams) or a request for the root site (/); this is an example of a basic router. The router will process the API call and return back a JSON response that has an array with two items. The request for the root page will stream the "index.html" page to the user.

```
var http = require('http');
var url = require('url');
var path = require('path');
var fs = require('fs');

http.createServer(function (req, res) {
  var uri = url.parse(req.url).pathname;
  switch(uri) {
    case '/api/v1/teams':
      res.writeHead(200, {'Content-Type': 'application/json'});
      res.end('[{"team":"Team A"}, {"team":"Team B"}]');
      break;
    case '/':
      var pathname = path.join(process.cwd(), 'index.html');
      fileStream = fs.createReadStream(pathname);
      fileStream.pipe(res);
      break;
  default:
      break;
  }
}).listen(8080, "127.0.0.1");

console.log('Server running at http://127.0.0.1:8080/');
```

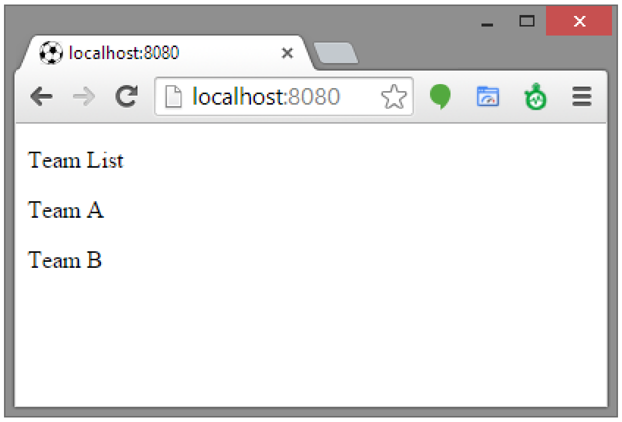
Fig. 7. HANDS basic server template: app.js.

The next step is to create a front-end page to call the back-end server. Create a page called "index.html" as shown in figure 8. The page includes a reference to jQuery to simplify the support of AJAX on different browsers. The page has a div placeholder for the data retrieved from the service ("teamlist"). An inline JavaScript section calls the service, builds the html, and injects the html into the "teamlist" div.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title></title>
 <script src="//ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js">
 </script>
</head>
<body>
    <p>Team List</p>
    <div class="teamlist"></div>
<script>
  var html = '';
  $.ajax({
    url: '/api/v1/teams',
    type: 'GET',
    success: function(data) {
      $.each(data, function (i, value) {
        html += '<p>' + value.team + '</p>';
    });
    $('.teamlist').html(html);
  }
});
</script>
</body>
</html>
```

Fig. 8. HANDS basic html template: index.html.

The JavaScript in the page can remain inline or get separated out to an external JS file. The approach given, provides the student with a view of all code while the separation may provide some performance optimization. The result of viewing the page in a browser is shown in figure 9. As shown, the Node.js application is running on localhost (127.0.0.1) port 8080. The application's router processes the request. Since it is the root page, the "index.html" page is streamed to the browser. The browser processes the script tag and calls the API. The Node.js application processes the request and returns back the JSON data with Team A and Team B. The page successfully receives the data and injects the html. The final page lists both teams.



Team List

Team A

Team B

Fig. 9. HANDS basic template as shown in web browser.

The design of individual pages with content injected using web service APIs is the design approach for HANDS. This approach leverages the simplicity of HTML and AJAX to create a web application. It avoids the complexity and issues found in single page applications where all pages are dynamically generated. HANDS keeps the simplicity of HTML only pages and provides the ability for server-side processing using AJAX request. It simplifies the pure server delivery approach, by only requesting the data needed for the requested page. This hybrid approach, provides a simple and clear design for students to study while delivering a framework that is enterprise ready and extensible.

The process of adding additional HTML pages, adding APIs, and updating the router continues until a Minimum Viable Product (MVP) is ready. The application is deployed to the web servers and configured. The deployment of Node.js is beyond the scope of this paper and more details can be found on the web [10].

VII. Conclusion

The position presented in this paper is an approach for web development called HANDS. This is a hybrid approach, leveraging the simplicity of plain old HTML pages, the AJAX injection of HTML, and the server-side processing found in the Node.js framework. The approach presented, allows the student to see the interaction between HTML, JS, and Node.js. This interaction is at the primitive level, not abstracted in any framework wrapping code. The key point is to understand the primitives which are used in all frameworks to position the student for future issues and additions of new features.

HANDS provides a design pattern that is easily picked up by students while providing the foundation for enterprise applications. HANDS allows for the addition of client or server side packages to extend the foundation. While extensibility is enabled, the paper takes a position that all packages used should be well understood. HANDS source and examples are available as open source [9].

VIII. Appendix

Complete final source code for files.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title></title>
  <script src="//ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js">
</script>
</head>
<body>
    <p>Team List</p>
    <div class="teamlist"></div>
<script>
    var html = '';

    $.ajax({
        url: '/api/v1/teams',
        type: 'GET',
        success: function(data) {
            $.each(data, function (i, value) {
                html += '<p>' + value.team + '</p>';
            });

            console.log(html);
            $('.teamlist').html(html);
        }
    });
</script>

</body>
</html>
```

app.js

```
var http = require('http');
var url = require('url');
var path = require('path');
var fs = require('fs');

http.createServer(function (req, res) {
  var uri = url.parse(req.url).pathname;
  switch(uri) {
    case '/api/v1/teams':
        res.writeHead(200, {'Content-Type': 'application/json'});
        res.end('[{"team":"Team A"}, {"team":"Team B"}]');
      break;
    case '/':
        var pathname = path.join(process.cwd(), 'index.html');
        fileStream = fs.createReadStream(pathname);
        fileStream.pipe(res);
      break;
    default:
      break;
  }
}).listen(8080, "127.0.0.1");

console.log('Server running at http://127.0.0.1:8080/');
```

IX. References

[1] "Tags used in HTML". World Wide Web Consortium. November 3, 1992. Retrieved November 16, 2008.

[2] "First mention of HTML Tags on the www-talk mailing list". World Wide Web Consortium. October 29, 1991. Retrieved April 8, 2007.

[3] W3Schools Tutorial. Retrieved from http://www.w3schools.com/, November 1, 2014.

[4] JavaScript specification. Retrieved from http://www.w3.org/standards/webdesign/script, November 1, 2014.

[5] W3 specifications. Retrieved from http://www.w3.org/standards/webdesign, November 1, 2014.

[6] "What is CSS?", World Wide Web Consortium. Retrieved December 2010, Retrieved from http://www.w3.org/standards/webdesign/ htmlcss#whatcss, November 1, 2014.

[7] Carter, Brian. "HTML Educational Node.js System (HENS)", To be Published. Annual Global Online Conference on Information and Computer Technology – CICT, December 2014.

[8] HANDS open source code. Retrieved from https://github.com/ ChipSoftTech/HANDS.

[9] Node.js Documentation. Retrieved from http://nodejs.org.