# Lab 6 Vue

## Background

You now have some experience server side (PHP) and client side (html/javascript) programming. To review, PHP is a server side scripting language which executes as quickly as it can when someone requests the web resource. All your echo or print statements are written to a temporary file and returned to the requester.

You may have thought this pattern a bit odd: using one language server side to generate 2 client side languages. Indeed it is odd, but nevertheless, PHP is one of the most common server side languages.

There are 2 modern day remedies that I wouldn’t exactly call cures, but do help the situation.

1. *At least* make the server side and client side languages the same so the connection between them isn’t so clunky and programmers can reuse their knowledge of JavaScript. Enter nodejs.

Nodejs is simply the part of chrome that can execute javascript([V8](https://v8.dev/)) along with some additional useful libraries. These libraries include things like web servers, file writing, mysql, etc. that are not part of traditional client-based javascript. With these additions, you could now have all your client side and server side code written in JavaScript--one language to rule them all. As much as I dislike JavaScript, it is a whole heck of a lot easier to use than PHP.

Nodejs’s main purpose is to be a web server--to replace the need for PHP. In fact when you visit the [about page for nodejs.org](https://nodejs.org/en/about/), the demo program shows how to write a basic web server program. Though it’s still a general purpose language, and nothing stops you from using it to run arbitrary JavaScript files that aren’t webservers. Just know that some things will be missing like the DOM.

Nodejs faced a difficult issue. Recall that JavaScript’s initial purpose was to be an itsy bitsy bit of code included along with some HTML for enhanced interactivity. Having such humble beginnings, JavaScript lacked some basics--perhaps most importantly, a module system (see [chapter 10 of Eloquent Javascript](https://eloquentjavascript.net/10_modules.html) for more details). So the community invented a module system out of existing JavaScript constructs and called this method CommonJS. Since then, ECMA/JavaScript has evolved. In one of these evolutions (ES6), modules were released as built it constructs. Naturally, the built in modules are different than the ad hoc CommonJS. So right now there are TWO ways to create and use modules: CommonJS modules and the built in module system we call "ES6 modules". What a confusing mess! At the very least since CommonJS is simply vanilla JavaScript, you can use both module systems within the same file when you need to.

If JavaScript is to be used beyond client side web scripting, it also needed a way for developers to share code with each other when they've solved a common problem others are likely to face. The package manager for Node, which you might call the Node Package Manager, is called npm. I wonder why?

npm is the main package manager for JavaScript libraries. It is a central location to where people publish their libraries and also a program called “npm” which is included when you download node that helps you download and manage libraries for your projects. Of course, being about as old as node itself, it uses the first module system CommonJS.

Well naturally, web developers wanted to be able to use some of the vast functionality of javascript libraries found in npm in their web pages. To help make this jump from an npm/node program into what we call a *web application*, a number of programs appeared. Fittingly, most of these are written in JavaScript and executed via nodejs in the development cycle:

* [**Browserify**](http://browserify.org/)
  + allows you to use the nodeJS module system syntax (CommonJS) to “compile” for use in a browser.
* [**Webpack**](https://webpack.js.org/)
  + does the same thing as Browserify, but has some additional functionality and is quite a bit more popular than browserify.
* [**Babel-**](https://babeljs.io/)
  + a transpiler which allows programmers to write javascript using the newest additions without worrying that a particular browser doesn’t support the new javascript yet. Babel transpiles to an older version of JS. Frankly, I don’t really care about supporting older browsers. People using IE11 aren’t the kind of users I want anyway. Nevertheless, I use babel on almost every web project so I can write in TypeScript (a superset of JS that allows me to use types when writing JS) and transpile down to JavaScript. It is interesting how JavaScript was meant as a programming language, yet fewer and fewer people write JS directly and instead opt to write code that transpiles to it.

This leads us to the second “treatment”.

2. Shift most (if not all) of the work to the client. Instead of generating a custom experience for each visitor by using a PHP or nodejs script, serve everyone a static webpage which includes code for the client to run to create their own customized experience. This will inevitably end up with some additional requests to a server, but you either create those services separately or use an existing service like firebase.

For example, using Firebase Authentication and Firebase Firestore, you can write a web app that authenticates AND fetches a user’s personal information 100% client side--no server side code writing necessary. Cool! This has the added benefit that you don’t need PHP or mysql or nodejs installed and running somewhere taking up resources on your server. Your server simply serves static webpages that include a bunch of JavaScript. This makes caching and serving many clients at once very easy.

However, now we have a new problem which is the focus of this lab. The problem is simply this:

Writing a large web application naturally consists of frequently *displaying* information to the user and *getting* information from the user.

In other words, you have some data in JavaScript that you wish to turn into HTML, and you have HTML elements that get data from the user you need to turn into JavaScript. This back and forth binding is perfectly possible in vanilla javascript but:

1. Has significant boilerplate
2. Does not scale well (like at all).

Consider the following example:

|  |
| --- |
| <**div**>Please enter the cost of the item:</**div**>  $<**input** oninput="recalculate()" type="number" id="txt" />  <**div** id="output"></**div**>  <**script**>  const TAX = 0.065;  function recalculate() {  const cost = parseFloat(document.getElementById("txt").value);  console.log(cost);  const total = cost + cost \* TAX;  document.getElementById("output").innerText = "$" + total;  }  </**script**> |

You can interact with this example [here](https://people.eecs.ku.edu/~p098k866/example.html).

What if I wanted to color the background according to the total cost? What if I had many outputs based on many inputs? What if I also had outputs that depended on several inputs? What if I needed to display a list dynamically? What if the rest of the lab is phrased in questions?

You might be able to see how this can get pretty complicated and messy pretty quickly. This is where **JavaScript Frameworks** come into play. There are three popular JavaScript Frameworks:

1. [Angular](https://angular.io/)
2. [React](https://reactjs.org/)
3. [Vue](https://vuejs.org/)

Of these 3, Vue is my favorite because it is about an order of magnitude easier to set up and get started using than Angular or even React. It is heavily influenced by Angular but is much easier to use. In the words of Vue’s creator,

*“I figured, what if I could just extract the part that I really liked about Angular and build something really lightweight.”*

Watch the video on [this page](https://vuejs.org/) by clicking the Why vue.js? Button for a small introduction to Vue.

In this lab, we will be using Vue.js to create a small web application. Additionally, we want it to be pretty, so in addition to the *JavaScript* framework Vue.js, we will be using the most popular *front-end* framework known as [Bootstrap](https://getbootstrap.com/).

Bootstrap is nothing more than a collection of CSS classes along with some JavaScript files to enhance the css classes. For example, bootstrap includes classes for accordions and tooltips, but they won’t work properly without the javascript which finds elements of the specified classes and adds the associated functionality. As CSS becomes increasingly sophisticated, I wouldn’t be surprised if a future version of Bootstrap doesn’t require any JS and you need only include a bootstrap CSS stylesheet in your html page.

Note I prefer to use Visual Studio Code with the live server extension installed, but you can use any editor you like.

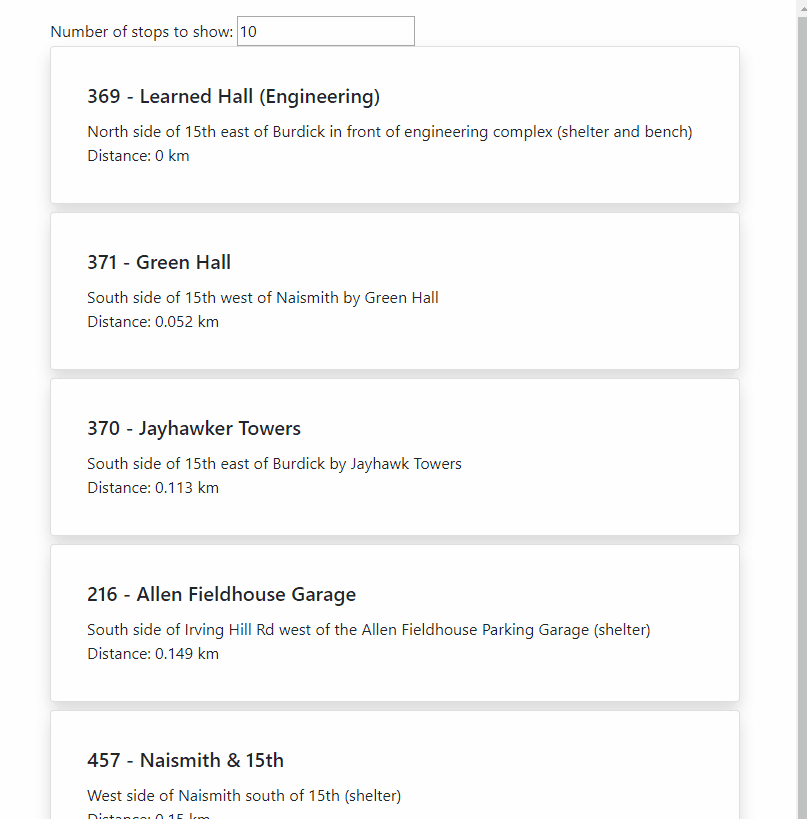
There are TWO ways to use Vue:

1. Use npm and node to create a project, add vue to it (an npm package), add bootstrap (an npm package), install webpack (an npm package), and eventually build the project into html, css, and js components.
2. Link to hosted versions of bootstrap and hosted versions of vue in an html page.

Each method has its benefits. A couple reasons to use the second method are:

1. It’s much easier.
2. If your user has visited any other page on the internet that also linked to these publicly hosted resources, the browser already has these files in its cache and the load will be very fast.

In this lab, you will be creating a web app which queries the user’s location and dynamically shows you the closest X number of bus stops in Lawrence. Cool!



## Instructions

## Part 1

#### Step 1: Setup.

Here is some boilerplate HTML for a blank page. The ‘meta’ tags ensure that the webpage will display nicely on a mobile device.

|  |
| --- |
| <!DOCTYPE html> <**html** lang="en">  <**head**>  <**meta** charset="UTF-8" />  <**meta** name="viewport" content="width=device-width, initial-scale=1.0" />  <**meta** http-equiv="X-UA-Compatible" content="ie=edge" />  <**title**>Vue Lab</**title**>  </**head**>  <**body**></**body**> </**html**> |

#### Step 2: Bootstrap.

The next step is to add bootstrap to your page. [According to the bootstrap website](https://getbootstrap.com/docs/4.3/getting-started/introduction/), you just need to add links to the stylesheet and javascript like so:

|  |
| --- |
| <!DOCTYPE html> <**html** lang="en">  <**head**>  <**meta** charset="UTF-8" />  <**meta** name="viewport" content="width=device-width, initial-scale=1.0" />  <**meta** http-equiv="X-UA-Compatible" content="ie=edge" />  <**title**>Vue Lab</**title**>  <**link**  rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"  crossorigin="anonymous"  />  </**head**>  <**body**>  <**script** src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></**script**>  <**script**  src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"  integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"  crossorigin="anonymous"  ></**script**>  <**script**  src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"  integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"  crossorigin="anonymous"  ></**script**>  </**body**> </**html**> |

**Note:** The integrity tags help prevent your web app from running malicious code. Including JavaScript and CSS from sources other than your own server is inherently dangerous. After all, if I’m an attacker and I know 1 million websites use bootstrap JS code, well that’s a pretty good place to attack! So we know what a *normal* version of the files should look like, and we tell the browser to compare the hash of the retrieved file to the integrity value *before* running it. If it doesn’t match, don’t run it!

##### Step2b

* Now ensure bootstrap is working properly. Copy and paste in the [example jumbotron element](https://getbootstrap.com/docs/4.3/components/jumbotron/) **above** the script tags **within** the body tag. Feel free to explore the other bootstrap components, layouts, and utilities accessible by adding classes to your html elements. The cards used in the gif above use some shadow utilities for extra coolness.
* Try a couple out you find interesting on some elements in your page. Feel free to keep some to make your page unique.

### Step 3: add Vue

Now add Vue to your webpage.

* Follow [these directions](https://vuejs.org/v2/guide/) to add the Vue script tag just before your </body> tag.
* Place the following in your body element **before** all your script tags:

|  |
| --- |
| <div class="container">  <div id="app">  Message: {{ message }}  </div> </div> |

* + The container class is a bootstrap class that helps make our page responsive & centered.
  + We are about to include some JS that initializes a Vue instance that works by looking for something with an id of “app”
  + {{ message }} is Vue syntax called *interpolation*-- all JS frameworks have this capability though it may look & function slightly differently. ‘message’ is going to be a JS variable and Vue is *automagically* going to write out the value for me. The best part is when you change the value in message, Vue automagically updates the DOM!
* Create a new script tag just **after** all the other tags but before </body> which includes:

|  |
| --- |
| const app = new Vue({  el: '#app',  data: {  message: 'Hello Vue!'  } }) |

* + We have created a new Vue object and instructed which HTML element to bind to and 1 data property called message.
* Now view your webpage in a browser. Open up the console and type:
  + app.message = “whatever”

To see that Vue reacts to the changed property!

* Read through the rest of [the vue guide page](https://vuejs.org/v2/guide/) down to Vue Components (which is outside the scope of this lab).
* Add a data property called ‘todos’ and initialize it with a few todo items as strings.
* Use the [v-for](https://vuejs.org/v2/guide/list.html) directive to display each item in the todo list. Note:
  + [v-for](https://vuejs.org/v2/guide/list.html) has to be used within the vue component. I.e. the div with an id of “app”
  + For best results, set the [key](https://vuejs.org/v2/guide/list.html#key) vue attribute to the stop’s id field.
* Once again, open the web page in a browser and open up the console to manipulate app.todos in various ways: push, pop, sort, etc.
* **Take a screenshot** of your page and console interactions to turn in for part 1 of this lab.

## Part 2

Part 2a: *You may remove properties added from part 1 if you wish.*

* Make a data property called **stops** and initialize it to an empty list.
* Using the [fetch web api](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch), fetch the stops by visiting the page at: <https://utils.pauliankline.com/stops.json> and store them in **stops.**
  + You may want to trigger this in the [‘mounted’ life cycle hook](https://vuejs.org/v2/guide/instance.html#Instance-Lifecycle-Hooks) of your vue app object.
* Add a data property called **numStops** and [bind it to an input element](https://vuejs.org/v2/guide/forms.html#number) at the top of your page. This will control how many stops are displayed to the user. Initialize **numStops** to 10.
* Create a [computed property](https://vuejs.org/v2/guide/computed.html) called **filteredStops** which returns the first **numStops** number of **stops**
* Display the first **numStops** of **filteredStops** to the user as [bootstrap cards](https://getbootstrap.com/docs/4.3/components/card/) using the **v-for** directive.
  + Use each stop’s **name** as the card title and the stop’s **description** property as the card’s text.
* Open up the console and change the value of app.numStops. Notice that vue automatically re-computes and displays the filteredStops property whenever numStops changes. Vue automatically detects these dependencies and re-renders the DOM when necessary. Very cool!

Part 2b:

* Create a new data property(/ies) to store the user’s **location**.
  + You will need to obtain the user’s location using the [geolocation api](https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API).
    - Note this has nothing to do with vue.
    - Note 2: geolocation only works from localhost **or** over https connections for privacy concerns.
  + You will also need to ‘watch’ location using the geolocation api.
  + Every time the location is obtained/changed, recalculate the distance property of each stop. The distance property is initialized to 999,999 which is an impossible earthly distance in km or miles since it is >half the circumference of the earth.
  + Note: be careful of the order here with the previous fetch for the stops. Fetch is asynchronous. Make sure the distances are calculated properly even if the position of the user is obtained before you receive the stops.
  + You will need to create a **distance function** which returns the distance between two sets of geographic coordinates. If you struggle with this, cite your sources. You can use whatever units you prefer (km,mi, etc) so long as you indicate the unit to the user in the next steps.
  + Note: If you think the distances all seem a bit off and you are confident your distance function is correct, know that the geolocation api uses whatever location information is available. For the lab computers, which do not have gps’s, a best guess location is used based on IP address. So the coordinates returned by the geolocation api may be based on your router, building connection, KU’s connection, Kansas, etc. As more detailed location information becomes available, the watchPosition callback will be called.
* Modify the HTML within your v-for directive to also display the user’s distance to each stop.
* Modify the code in your filteredStops computed property to return the *closest* **numStops** stops.
* Congratz! If you’ve written everything correctly, you can make this an html page in your people.eecs.ku.edu public\_html directory. Open it on your mobile phone and see the distances change and even the order of the stops change as you walk around.
  + To reiterate, this works because Vue detected that the filteredStops property depends on 2 values now:
    - numStops & distance.

When those change, the DOM is automatically (and efficiently) re-rendered in accordance with the new filteredStops value!