INFO3180 - Lab 6 (20 marks)

Due: ~~March 24, 2020~~ April 26, 2020 at 11:55 PM

The goal of this lab is to become familiar with VueJS and some of the basic concepts. You will be building a small single page application that will display a home page and a News page that will fetch News articles from a 3rd party News API.

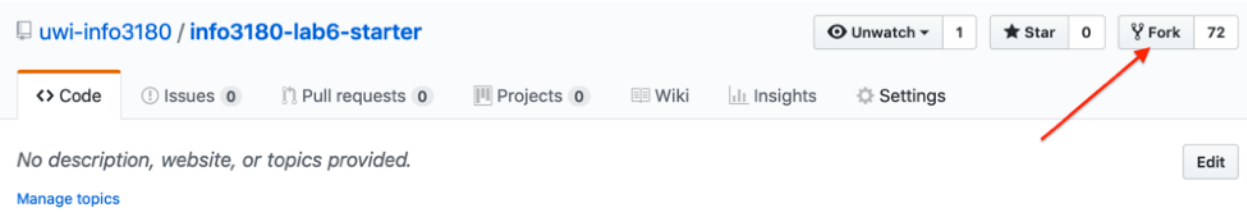
Learn VueJS
Here are some helpful links to learn about VueJS. Why VueJS? https://player.vimeo.com/video/247494684 VueJS Guide: https://vuejs.org/v2/guide/ Learn VueJS 2 Step by Step: https://laracasts.com/series/learn-vue-2-step-by-step

Debugging VueJS
It is recommended that you also install the VueJS Devtools browser extension to help with debugging your VueJS application. You may follow the instructions and download the extension at the following link: https://github.com/vuejs/vue-devtools#vue-devtools

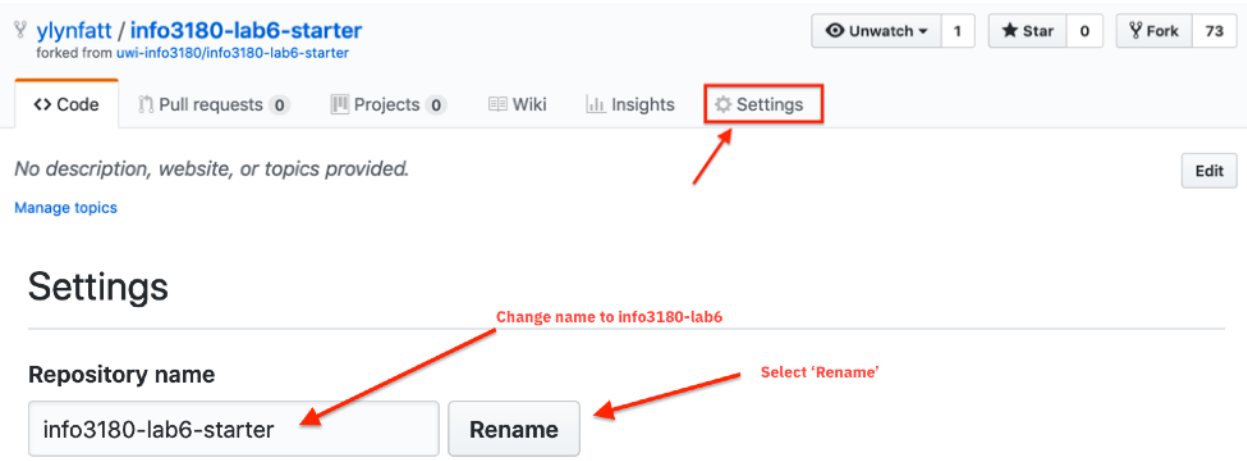
Fork and Clone the Repository

Start with the example at: https://github.com/uwi-info3180/info3180-lab6-starter

Fork the repository to your own account



In github.com rename it by going to settings and changing the name to **`info3180-lab6.`**



To start working on your code, clone it from your newly forked repository for example:

If you are working in Cloud9 or if you are working locally on your own computer (ie. NOT in Cloud9) :

```
git clone https://github.com/{yourusername}/info3180-lab6
```

Now, as always ensure you remember to activate your **virtual environment BEFORE** doing any package installation with pip or running python!

Now install the requirements and start the development server.

```
pip install -r requirements.txt
python run.py
```

Loading VueJS and our initial application page

Take a look in your **views.py** file. Notice how instead of using **render_template()**, we are instead using **app.send_static_file** to load our **index.html** file from our static folder. Since both VueJS and the Flask Jinja2 templating engine utilize the same **{{ }}** template syntax, they will both conflict. We could change the template syntax for either VueJS or Jinja2. Likewise we could use the Jinja2 **{% raw %}** and **{% endraw %}** template tags to signify that anything between the two tags should not be processed by Jinja2. In our case though we will opt to instead send the **index.html** file as simply a static file that should not be rendered or parsed by Jinja2 at all.

Now look at the end of your **index.html** file you will see we have loaded our VueJS library (and Vue Router which we will use later in this lab) and our app.js file.

```
<script src="/static/js/vue.js"></script>
```

```
<script src="/static/js/vue-router.js"></script>
<script src="/static/js/app.js"></script>
```

You may also notice that we have a ** ** with an "**id**" attribute with value "**app**". If you look in your **app.js** file you will see that we have instantiated a new **Vue** instance and told it that the element that we want to find and control for our Vue app is "**app**". Every Vue application starts by creating a new Vue instance with the Vue function.

```
let app = new Vue({
  // options
})
```

Let us now break up our layout into some components.

"Components are one of the most powerful features of Vue. They help you extend basic HTML elements to encapsulate reusable code. At a high level, components are custom elements that Vue's compiler attaches behavior to."
~ VueJS Guide

The basic structure of a component is as follows:

```
Vue.component('my-component', {
    // some options
});
```

You will notice that we have two components so far "**app-header**" and "**app-footer**". And if you look at the code for these, they each have a **template** property defined that contains the HTML code we want Vue to

use whenever those components are referenced. There is also a **data** property where we can store some data to use within our component.

```
Vue.component('app-footer', {
    template: `
        <footer>
            <div class="container">
                <p>Copyright &copy; {{ year }} Flask Inc.</p>
            </div>
        </footer>
    `,
    data: function() {
        return {
            year: (new Date).getFullYear()
        }
    }
});
```

Note: These components must be placed before you instantiate your Vue instance. You are also encouraged to use back ticks (``) instead of quotes for your template syntax if it spans multiple lines.

If you look back in your **index.html** file you will notice the two new components being used. **<app-header></app-header>** and **<app-footer></app-footer>**.

Open the VueJS DevTools in your Web browser and click on the **<Root>** component, then on the right side of the DevTools you should see the '**welcome**' data property. Change it's value in the DevTools and you should see it automatically change on the page.

Now, it's your turn to create a component. Our component is going to be a list of News items taken from a News API. Using the example components previously mentioned, create a new component called '**news-list**', which contains a template that looks like the following:

```
<div class="news">
    <h2>News</h2>
    <ul class="news__list">
         <li class="news__item">News item 1</li>
         <li class="news__item">News item 2</li>
         <li class="news__item">News item 3</li>
    </ul>
</div>
```

And then use that new component in your **index.html** file just below the paragraph with the welcome message by adding **<news-list></news-list>**.

If all goes well you should see something like Figure 2 below:

FIGURE 2

Great! Now you've created your first component. But it's not very useful, let us now connect it to an API so we can fetch some actual news.

News API

Visit http://newsapi.org/ and sign up for an account. You will be given an API key which we will use shortly.

Note: Keep this key private and remember to remove it from your application code before committing to Github.

Take a moment to read the documentation for this API and how it is used, as you will be creating your own API for your final project.

https://newsapi.org/docs/

Done reading the documentation? I hope so.

Now let us modify our '**news-list**' component. Add a **created()** function just below the template property (see example below).

```
Vue.component('news-list', {
    template: `…`,
    created: function() {
        fetch('https://newsapi.org/v2/top-headlines?
country=us&apiKey=<your-api-key>')
          .then(function(response) {
            return response.json();
          })
          .then(function(data) {
            console.log(data);
          });
    }
```

```
});
```

Each Vue instance goes through a series of initialization steps when it is created. Along the way, it also runs functions called **lifecycle hooks**, giving users the opportunity to add their own code at specific stages. The **`created()`** function is one of these lifecycle hooks and will be called as soon as our **`news-list`** component is created and we can use this lifecycle method to fetch the data from our API. In order to get our news from the News API, we will use the native **`fetch`** api to make our AJAX request. Update the **`apiKey`** property in the query string of the url with the API key you obtained when you signed up for the News API.

Now save your changes and reload your webpage. If you look in the browser console you will see an object printed to the logs that looks similar to *Figure 3* below. Pay attention to the '**`articles`**' property and you will see that it contains an array of objects with the '**`author`**', '**`title`**', '**`description`**', etc for each news article. See *Figure 4* below.

FIGURE 3

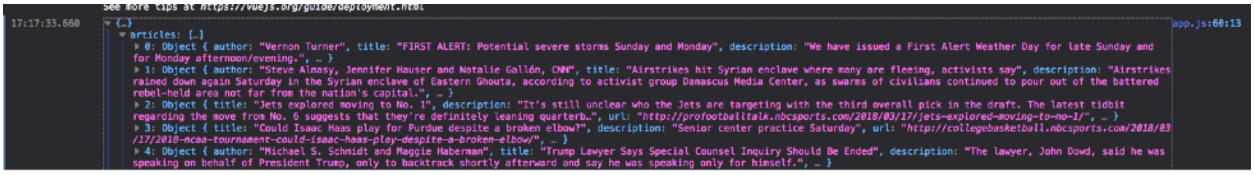


FIGURE 4

Now what we will do is save those articles to the VueJS **`data`** property and then display a list of the titles of those news articles. To do this, add the data property to your '**`news-list`**' component just after the **`created()`** function.

```
Vue.component('news-list', {
    template: `…`,
    created: function() {
        let self = this;

        fetch('https://newsapi.org/v2/top-headlines?
country=us&apiKey=<your-api-key>')
          .then(function(response) {
            return response.json();
          })
          .then(function(data) {
            console.log(data);
            self.articles = data.articles;
          });
    },
    data: function() {
        return {
            articles: []
        }
    }
});
```

Notice in our **data** property we are returning an object of other properties that our component will use. In this case we have created an empty articles array. In our **created()** function we have also updated it to store the articles we got from the News API to our articles data property.

This time open the VueJS DevTools you installed at the start of this lab. You should see something like what is shown in *Figure 5*. If you click on the **<news-list>** component (on the left of the DevTools) you should see '**articles**' array in the data properties (on the right of the DevTools).

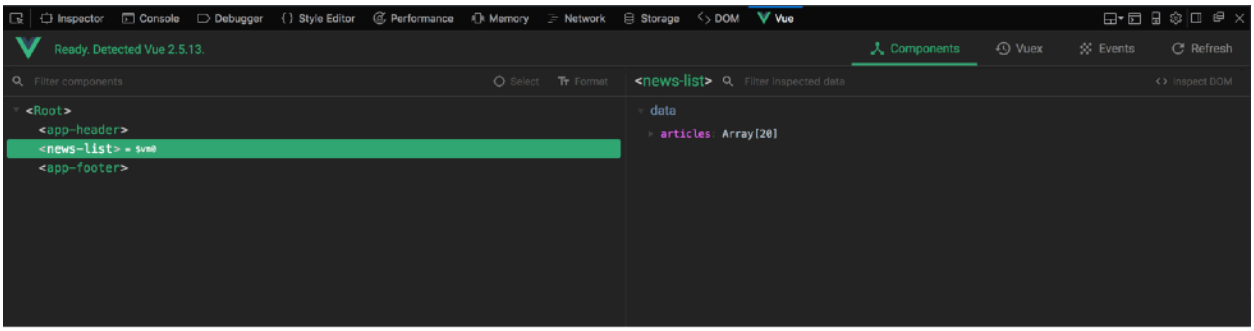


FIGURE 5: VUEJS DEVTOOLS

Fantastic, now let us print out the titles of these articles in our template. Since we have an array of articles, we will need to use the **`v-for`** VueJS directive to iterate over our list of articles. In the template property of the '**`news-list`**' component make the following change:

```
<div class="news">
    <h2>News</h2>
    <ul class="news__list">
        <li v-for="article in articles"
class="news__item">{{ article.title }}</li>
    </ul>
</div>
```

It should now look like *Figure 6*.

FIGURE 6: LIST OF ARTICLE TITLES

Great! You've successfully utilized the News API to display each **`title`** of the Top Headline articles. Now let us display some other details. Try to add the **`description`** and a **`photo`**. What other properties can you use from the News API to display the description and photo? When you have found these properties, update your template to display them along with the title.

Note: Displaying the image might be a little tricky. Instead of just using the '**`src`**' attribute by itself on an **`<img />`** tag, try using '**`:src`**'. This is shorthand for '**`v-bind:src`**' which is another VueJS directive that allows us to bind a string to the attribute.

Our News App is now taking shape, but it doesn't look so visually appealing. See if you can style it to look like the following:

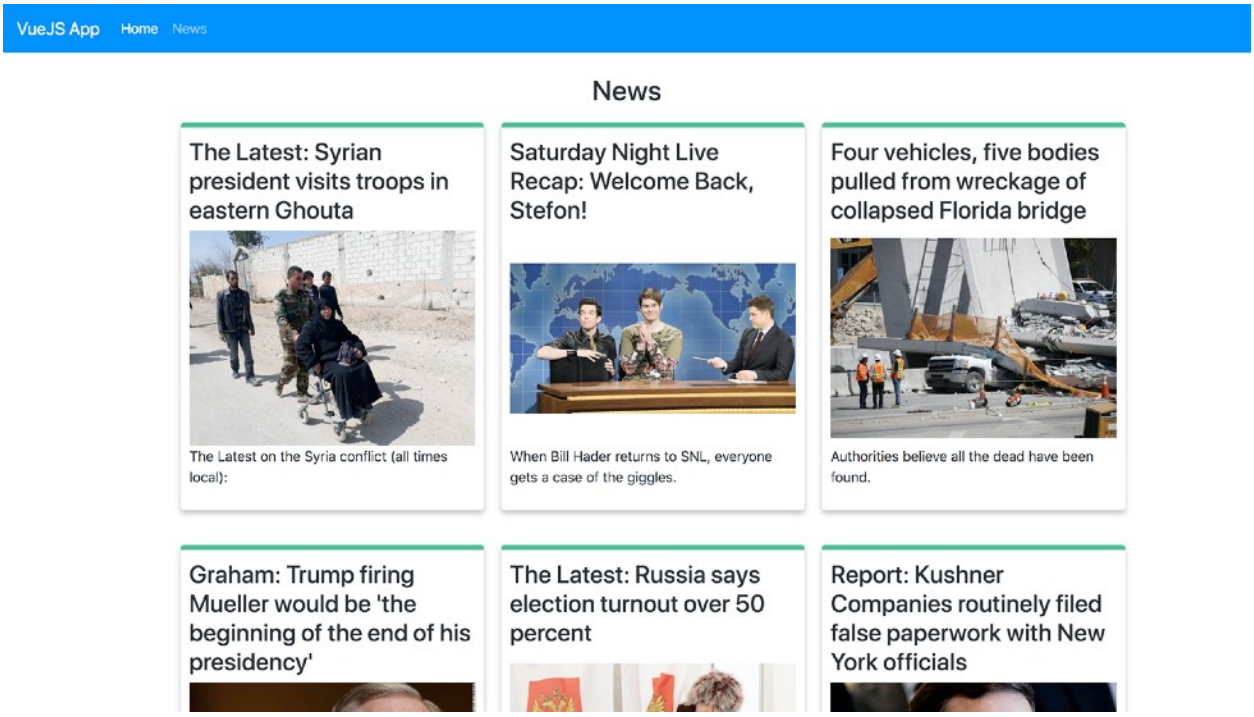


FIGURE 7: NEWS SECTION STYLED TO LOOK MORE VISUALLY APPEALING

Two-way Data Binding and Event Handling

Now we will introduce the concept of **two-way data binding** and **event handling** with VueJS.

Go to your '**`news-list`**' component and update the template to include the following code:

```
<div class="form-inline d-flex justify-content-center">
    <div class="form-group mx-sm-3 mb-2">
        <label class="sr-only" for="search">Search</label>
        <input type="search" name="search" v-model="searchTerm"
id="search" class="form-control mb-2 mr-sm-2" placeholder="Enter
search term here" />
        <p>You are searching for {{ searchTerm }}</p>
    </div>
</div>
```

You will notice the form input field has a new attribute. The **`v-model`** directive is what creates *two-way data bindings* on form input and texture elements. You will also need to add searchTerm to your data property object:

```
data: function() {
    return {
        articles: [],
        searchTerm: ''
    }
},
```

Save the changes and reload your page. Now try typing in the input field and see what happens. As you type you should see the text from the input field print out in the paragraph with *'You are searching for ...'*.

Now let us connect this to our News API and instead when we click on a button it will update the list of articles with what we searched for. Update your '**`new-list`**' component with a new **`methods`** property below the data property:

```
Vue.component('news-list', {
    …
    data: function() {…},
    methods: {
        searchNews: function() {
        let self = this;

        fetch('https://newsapi.org/v2/everything?q='+
self.searchTerm + '&language=en&apiKey=<your-api-key>')
          .then(function(response) {
            return response.json();
          })
          .then(function(data) {
            console.log(data);
            self.articles = data.articles;
          });
        }
    }
});
```

And then update the template property by replacing the paragraph with a button like below:

```
<button class="btn btn-primary mb-2"
@click="searchNews">Search</button>
```

Notice on our button we are using the **@click** directive, which is shorthand for **v-on:click**. This is similar to **click** event handler you are familiar with from Vanilla JavaScript. When the button is clicked our **searchNews()** function will be called and this will fetch news articles that match our search term and then update the articles list in our data property.

Routing

Even though we are building a single page application, there are times we will need to have other 'pages' dynamically loaded into our single page. We will use the **Vue Router** library for VueJS to allow for this. This way when we click on certain links they will dynamically load another component.

Vue Router Documentation: https://router.vuejs.org/en/essentials/getting-started.html

First, let us extract some code from our **index.html** file and create a component that will represent our home page. The code we will extract is:

```
<img src="/static/images/logo.png" alt="VueJS Logo">
<h1>{{ welcome }}</h1>
```

And the component will be:

```
const Home = Vue.component('home', {
   template: `
      <div class="home">
        <img src="/static/images/logo.png" alt="VueJS Logo">
        <h1>{{ welcome }}</h1>
      </div>
   `,
   data: function() {
       return {
           welcome: 'Hello World! Welcome to VueJS'
       }
   }
});
```

Note: We have assigned our component to a variable called '**Home**'. We will use this later.

Next, we will assign our News list component created previously to a variable called '**`NewsList`**'.

```
const NewsList = Vue.component('news-list', { … });
```

Now, that both components have been assigned to variables we will now define our routes and initialize VueRouter. Each route should map to a component:

```
const router = new VueRouter({
  mode: 'history',
  routes: [
      { path: '/', component: Home },
      { path: '/news', component: NewsList }
  ]
});
```

Next, change our Vue instance from:

```
let app = new Vue({
    el: '#app',
    data: {
        welcome: 'Hello World! Welcome to VueJS'
    }
});
```

to instead be:

```
const app = new Vue({
  el: '#app',
  router
})
```

Then, in **index.html** change:

```
<div class="text-center">
    <img src="/static/images/logo.png" alt="VueJS Logo">
    <h1>{{ welcome }}</h1>
    <news-list></news-list>
</div>
```

to:

```
<!-- component matched by the route will render here -->
<router-view></router-view>
```

Finally, update your '**app-header**' component and change the navigation links from:

```
<a class="nav-link" href="#">Home <span class="sr-
only">(current)</span></a>
```

to instead use VueRouter's **<router-link>** tag:

```
<router-link to="/" class="nav-link">Home</router-link>
```

Your nav should now look something like this:

```
<ul class="navbar-nav mr-auto">
  <li class="nav-item active">
    <router-link to="/" class="nav-link">Home</router-link>
  </li>
  <li class="nav-item active">
    <router-link to="/news" class="nav-link">News</router-link>
  </li>
</ul>
```

Now view your website again and you should see your Home component load initially and then if you click your News link, then your News component should load. You have now successfully implemented some basic routing using VueJS.

Note: Before you commit your code to your Github repository, please ensure you remove your API Key from the query string of the URL's in your application. This key is not to be shared with anyone else.

Submission

Submit your code via the "Lab 6 Submission" link on OurVLE. You should submit the following link:

- Your Github repository URL for your Flask app e.g. https://github.com/{yourusername}/info3180-lab6

Grading

1. The 'news-list' component should be created with `Vue.component('news-list', {})` (1 mark)
2. The 'news-list' component should have a '`template`' property and the HTML template for the news-list component should have the following:
    a. A search field with the attribute `v-model="searchTerm"` (1 mark)
    b. A button with the attribute `@click ="searchNews"` or `v-on:click = "searchNews"` (1 mark)
    c. An unordered list with a list item with the attribute `v-for="article in articles"`` (1 mark)
    d. The following properties should be printed `{{ article.title }}`, `{{ article.description }}` (1 mark)
    e. An `<img>` tag with `:src="article.urlToImage"` or `v-bind:src="article.urlToImage"` (1 mark)
3. The 'news-list' component should have a '`created`' property with `fetch()` function that makes an AJAX request to the News Api. (2 marks)
4. The AJAX call to the News API should get the 'articles' returned and update the 'articles' property of the 'data' property on the component instance. (2 marks)
5. The 'news-list' component should have a '`methods`' property which will have a `searchNews` function that uses the `fetch()` function to make an AJAX request. (2 marks)
6. The VueRouter instance should have two routes, one for the Home component and the other for the `NewsList` component. (2 marks)
7. In the `index.html` file you should see `` `<router-view></router-view>` `` (1 mark)

8. In the `app.js` file you should see another component called 'app-header' that has a template property with HTML code. In that HTML you should see `` `<router-link to="/news" class="nav-link">News</router-link>` `` (1 mark)
9. When you view the web page and click the 'News' link it should display the news items and you should be able to search. (2 marks)
10. The News page should look similar to the screenshot. (2 marks)