Project 1, Part I - Progressive Web Apps with Vue.js

By Constantin Miranda

0.1. Learning Objectives

- Understand the possibilities and limitations of the Vue.js framework.
- Understand the structure of progressive web apps written in Vue.js.

0.1. Deadlines & Credit

Part	Deadline	Credit
Part I	Tue 9/10, 2:55pm	20 points

0.1. Git Repository & Submission

Clone `git@github.coecis.cornell.edu:info4340-fa2019/YOUR_GITHUB_USERNAME-hw1.git`. Replace **YOUR_GITHUB_USERNAME** in the URL with **your actual GitHub username**. This is usually your NetID.

Submit **all** materials to your GitHub repository for this assignment.

Tip: Commit and push your changes every time you work on your project. Every time you commit and push you store your changes on the GitHub server. This acts as a back-up for your work. It also means that if you forget to submit before the deadline, there's something already on the server that the TAs can grade.

1. Part I: Progressive Web Apps with Vue.js

You'll be building a Progressive Web App (PWA) for this homework.

> Progressive Web Apps are web applications that are regular web pages or websites, but can appear to the user like traditional applications or native mobile applications.
> Source: https://en.wikipedia.org/wiki/Progressive_web_applications

Because PWAs are built with standard HTML5 technologies you don't need any special tools to build an app. You can build a PWA with just the technologies and techniques you learned in INFO 1300. However, using a framework, like Vue.js, can save you a lot time and effort.

We'll be using the Vue.js framework to build our apps this semester.

As UX designer, you'll need to work with the software engineers to design interfaces that they can actually code. Before designing your app, you should first understand the tools that will be used to build the app and any technical possibilities and limitations of these tools. If you don't know the limitations of these tools, you may produce designs that are technically unfeasible or extremely difficult to code.

In this part, you will study how to work with the Vue.js framework.

1.1. Getting Started with a Vue.js Project

You should have already cloned your repository for this assignment and opened the project in Visual Studio Code.

Vue.js projects are simply node.js packages (libraries) that import the Vue.js packages (libraries). Node.js is simply a JavaScript runtime that you can use outside of a web browser. All node.js packages have a **package.json** which tells node.js about your package. **Take a look at *package.json*.** Notice that the *vue* package is specified in the *dependencies* section. This tells Node that your app (package) will use the *vue* package.

In order for you to run your app, you'll need to install the dependent packages. First run `npm install` inside of your repository. **npm** is the **N**ode **P**ackage **M**anger. This will tell Node.js to install all of the dependent packages for your project into the *node_modules* directory.

To run a Vue.js app, you need issue the `npm run serve` command. Check out the *scripts* section in **package.json**. Observe that *serve* will simply run the `vue-cli-service serve` command for you. This command launches a local web server written in JavaScript via Node.js. To stop the local web server, press **control + c**.

1.2. How are Vue.js PWAs Structured?

Let's take a look at the files that make this application work. We won't explain everything, so you'll need to refer to Vue's documentation or ask for help if you don't understand a piece of code. Remember, we are all here to learn, and when we help each other, we all learn more.

- **main.js**: This is the main JavaScript file that drives your Vue application. The Vue.js build tool looks here to figure out how to build the rest of the application. This file imports Vue.js and creates the Vue instance. It also tells the application to render **App.vue** on-screen.
- **App.vue**: This is the main single-file component for your application. As you can see, the only element we're using here is `<router-view>`. This allows us to place a "view" on-screen and switch between "views" as a user navigates our application.
- **router.js**: Defines what view to display. Notice that the path is set to `/`. This is the root path (http://localhost:8080/) and currently is pointing to the **Announcements.vue** view. When a user navigates to our application, the first thing they will see is what we coded into **Announcements.vue**.
- **Announcements.vue**: This is the main view of the application. It imports and places the two *components* for our application on-screen: **InputAnnouncement.vue** and **ViewAnnouncement.vue**. The **Annoucements.vue** view also handles passing data from the **InputAnnouncement.vue** component to the **ViewAnnouncement.vue** component.
- **InputAnnouncement.vue**, **ViewAnnouncement.vue**: These are the two components for our application.

  One handles all of the elements and logic for collecting information from the user. It uses `$emit` and `submit()` to send the data back to **Announcements.vue**.

  The other handles all of the elements and logic for displaying the input announcement in a table. It receives the input data via **Announcements.vue** (remember that **InputAnnouncement.vue** sent the data to **Announcements.vue** to then get passed to this component).

1.3. Building a User Interface in Vue.js

To help you see how these views and components come together to build app, we're going to add some input components to the interface and display them back to the user.

1. We'll need a form to enable users to input an announcement, just like we learned in INFO 2300. Let's start by adding the input components to the user interface.

   In the `<template>` section of **InputAnnouncement.vue** add the following input components after the `<h4>`:

   ```
   <input
     placeholder='Enter Text Here'
     v-model='tempAnnouncement'
     @keyup.enter='submit'
   />
   <button v-on:click='submit'>Set Announcement</button>
   ```

2. When a user submits the form, we'll want to store the data they entered so we can show it back to them later.

   The `v-model='tempAnnouncement'` in the input component tells Vue to store the user's data for this input component in the `tempAnnouncement` property. We need to create this property.

   In the `<script>` section, add the following code block to create the `tempAnnouncement` property.

```
data () {
  return {
    tempAnnouncement: ''
  }
},
```

3. Next, we want to send the information that the user input in `InputAnnouncement.vue` to the `ViewAnnouncement.vue` component.

   We will do so using the `$emit` feature in Vue. Copy-paste this code block below the `data()` block:

```
methods: {
  submit: function () {
    this.$emit('inputData', this.tempAnnouncement)
    this.tempAnnouncement = ''
  }
}
```

4. Before we start connecting everything, we need to import the components.

   At the top of our `<script>` tag in the **Announcements.vue** view add the following code:

```
import InputAnnouncement from '@/components/InputAnnouncement.vue'
import ViewAnnouncement from '@/components/ViewAnnouncement.vue'
```

   And we need to register them. Paste this code block directly below `name: 'Announcements'`:

```
components: {
  InputAnnouncement,
  ViewAnnouncement
},
```

5. We need to capture the information sent from `InputAnnouncement.vue`.

   You can access the data you `$emit`-ed by referencing the key (`inputData`) and calling a method to handle the passed data.

   Add the following component tag to **Announcements.vue**. This pulls the data from the **InputAnnouncement** component into our main view.

   ```
   <InputAnnouncement @inputData='addAnnouncement' />
   ```

   Now, we will want to store the data somewhere, so add the following code block below the `components:` property:

   ```
   data: function () {
     return {
       announcementData: ''
     }
   },
   ```

   Now, we want to write the method, `addAnnouncement()` that we are calling in our component tag. This method will take the passed data and assign it to our `announcementData` data property.

   ```
   methods: {
     addAnnouncement (variable) {
       this.announcementData = variable
     }
   }
   ```

6. We've now captured the data from **InputAnnouncement.vue** and stored it in a data property in our main view. How do we send data from our main view to another component? We'll use a `prop:` in conjunction with a `watch` .

   In **Announcements.vue**, add the following component tag. We are passing `aanouncementData` as a `prop:` titled 'announcementPassed'.

   ```
   <ViewAnnouncement :announcementPassed='announcementData' />
   ```

   We need to register our `prop:` in **ViewAnnouncement.vue**. Copy-paste the following code block below the `name:` property:

   ```
   props: {
     announcementPassed: {
       type: String
     }
   },
   ```

   We now want to watch our `prop:` for any changes. We'll achieve this using a `watch:` property. Copy-paste this code block below your `props:` section:

   ```
   watch: {
     announcementPassed: function () {
       var inputData = { 'date': 'Latest Announcement', 'announcement': this.announcementPassed }
       this.announcementList = inputData
     }
   }
   ```

7. Finally, we need to control when the components display.

   When an announcement has not been entered, we want **InputAnnouncement.vue** to display. When an announcement has been entered, this component should disappear and **ViewAnnouncement.vue** should display.

   In **Announcements.vue**, add `inputData: false` to your `data:` property. It should now look like this:

```
data: function () {
  return {
    announcementData: '',
    showInput: true
  }
},
```

   When users input data, we want the `showInput` boolean to change to `false`. Modify your `addAnnouncement()` method to match the following:

```
methods: {
  addAnnouncement (variable) {
    this.announcementData = variable
    this.showInput = false
  }
}
```

   Finally, we want to control the `<template>` elements using a `v-show` directive. Modify your component tags to match the following:

```
<InputAnnouncement v-show='this.showInput' @inputData='addAnnouncement' />
<ViewAnnouncement v-show='!this.showInput' :announcementPassed='announcementData' />
```

8. Refresh the app in your browser. Clicking the 'Set Announcement' button should now display the user's announcement in a table.

1.4. Using Node.js Libraries

If you notice, the date stamp is hardcoded to say "Latest Announcement". Our users will want something more specific. We are going to use Moment.js to generate a time stamp in a text string.

First, stop your development server using **control + c**. Install Moment.js using `npm install --save moment`. Notice your **package.json** and **package-lock.json** files have been modified to include Moment.js.

Import the library at the top of your `<script>` tag in `ViewAnnouncement.vue`. Modify the `inputData` associative array to use Moment.js to generate a time stamp. Assign the time stamp to the `date:` field in the array.

1.5. Vue.js Resources

Web frameworks constantly change. What's hot now, won't be hot in 5 years. However, conceptually they all work similarly. The best way to learn about a framework is to utilize its reference documentation.

You need to use this documentation to complete this homework.

- Vue Documentation: https://vuejs.org/v2/guide/syntax.html
- Vue CLI 3 guide: https://cli.vuejs.org/config/
- Moment.js guide: https://momentjs.com/
- `v-if` Documentation: https://vuejs.org/v2/guide/conditional.html#v-if
- `v-on` Documentation: https://vuejs.org/v2/guide/events.html