High Performance Single Page Application with Vue.js

Premise

Static HTML and simple web-pages are already a history now. The novel web applications are advanced and do a lots of functionalities. Also, the amount of data we show on a page and the relationship between them is growing large. For an application with a lot of related information scattered on a page, it is crucial to maintain the consistency between them. For a client, We had to develop one such app, which is extensively data-rich and we wanted it to be completely swift and seamless.

Vanilla JavaScript

We initially were against any javascript frameworks. Even when there were lots of them flourishing in the market, there are a few factors(like maintainability and additional-syntaxes) that insisted for a second opinion on going behind a framework.

But the challenge of creating such a data-rich application that consistently maintains state seamlessly required lots of javascript code and the maintenance of it will be a real complex one.

Going with a Framework

A Javascript framework describes a specified structure of how the code should be presented. Mostly like a code-template, along with some helpers, constructors etc. to solve/simplify a specific problem or bring your architecture to an order.
If we go with a framework, it provides the state-maintenance out of the box, though not all of them. Reusability and Code-Maintenance are other two important reasons behind going with a framework. In addition to these, we also get the following benefits while going with a framework.

- Component Based
- Strong community for support
- Third party libraries and useful components to deal with things
- Browser extensions for debugging
- Well crafted for single-page applications

The factors

We were looking for the following criterion to choose the right Framework.

1. **Light:** Should be simple enough and target the problem
2. **Performance:** Should be fast and provide a seamless UX
3. **Robust:** Should be strong enough to scale for larger requirements
4. **Faster Development:** Should be developer friendly
5. **Quick learning curve:** Should be easily learnable

The Initial Candidates

At a high level, we chose the following frameworks based on the popularity.

- EmberJS
- Angular
- ReactJS
- VueJS

Out of these, Angular and EmberJS are having relatively a larger community support and highly used in many web applications. We ruled out Angular due to its breaking changes release after releases. The reason we didn't choose Ember is that it requires a large learning curve. We had to get up to speed with the application. Eventually we chose to compare ReactJS and VueJS due to their less learning curve and their Virtual-DOM support.

React vs Vue

Both of the frameworks share many similarities due its architectural design and developers(React core developers have contributed a lot to VueJS).

Common Features

1. **Reactive functional programming**—data flows and the propagation of change in view components. (=observable + observer + schedulers)
2. **Virtual DOM** manipulation instead of HTML/XML DOM.
3. **Components** are Reactive/Stateful and Simple/Stateless.
4. **Routers** for navigation with component constraints/filters/props.
5. **Native** and custom directive support for HTML or JSX as attributes.
6. **Global state management** is highly scalable with Vuex(Vue)/Redux(React).
7. Functionalities like **Data/Event bindings, Mixins, Filters and Helpers** .
8. **Un-opinionated** app structure with build systems.
9. **Tree shaking** support with Webpacks 2.
10. Karma as **test framework** along with Mocha/Jest
11. **REST API support** and XHR request handler with axios.
12. Chrome **dev-addon tools** for debugging Vue/React and also for Vuex/Redux.
13. **VS Code extensions** for both React and Vue dev coding.

Other Features

1. **Runtime performance** are exceptional and comparatively fast. Refer the benchmarks
2. **Component Rendering** is more optimized in Vue compare to React due its intelligent state listeners and computed/watch methods.
3. **Size** of React framework is higher than Vue due to its rich set of directive and functions. But, Vue does not have rich set of directives to keep the lib size (~30KB gzipped) small and clean. Please check the Vue creator's reply for one of the feature request.
4. **Adoption** of React is quite difficult for a HTML developer/designers. It is kind of a full-stack framework/library, everything is just JavaScript. ex: JSX, ES6, CSS loading and etc. But, Vue is simple and you can use HTML, Scoped CSS, ES6 and JSX within same components.
5. **Community support** is higher for React and Facebook is backing it. VueJs is created by Evan You and it is slowly becoming famous with community due to its performance and pros.
6. **Popularity** of React (stars/forks) and contributors are higher than Vue (stars/forks) contributors.

7. **Hybrid App** is possible with React Native (React) and Weex (Vue)
8. **Documentation** is one of the plus for Vue compare to React docs
9. **Licensing** — Both Vue and React are licensed under MIT. Please check the links for the permissions, limitations and conditions. However React earlier was under BSD + Patent license.

A Sample Code

React

```
class HelloMessage extends
React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="World!" />,
  mountNode
);
```

### Vue JS

```
<div id="app">
  {{ message }}
</div>
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello World!'
  }
})
```

We can clearly see that the Vue code is very simple and more readable than React. Additionally, to work with reactjs, we had to learn a new templating language called JSX, whereas with VueJS, its all plain HTML and Javascript.

Vue Async Components with vue-router

We also went ahead and explored advanced concepts in both React and VueJS and have found that the VueJS continuously grabbed our attention. For example, the concept of Async-Components with vue-router had solved one of out loading-time issue considerably.

For our login-page whose app.js size was initially 9.5 MB and took longer load. The problem we identified here is the synchronous loading of Konva editor that bloated up the initial app.js file.
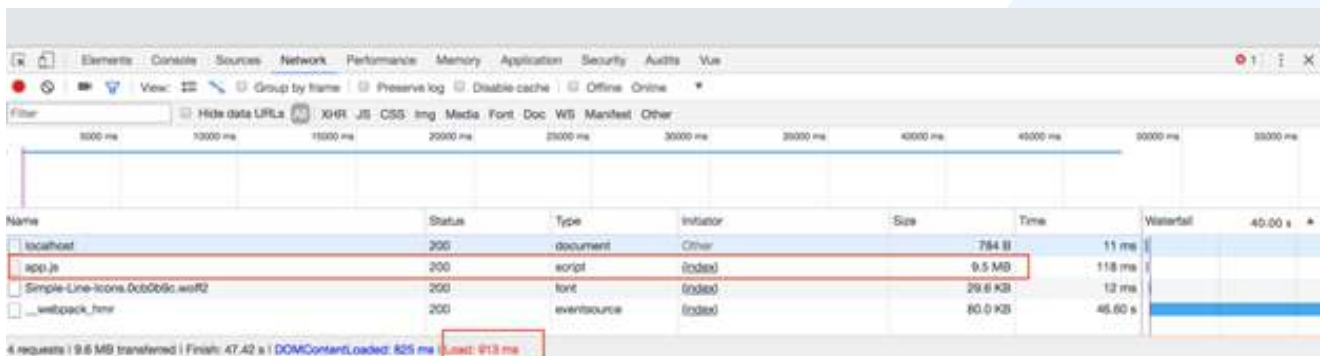
Before using Async-Components and vue-router

With Async-Components and vue-router, we were able to load the konva editor asynchronously which significantly reduce the initial app.js size and made the component load lazily only when needed. And the wow-factor here is the amount of change involved in this optimization is just a line of code as shown below.
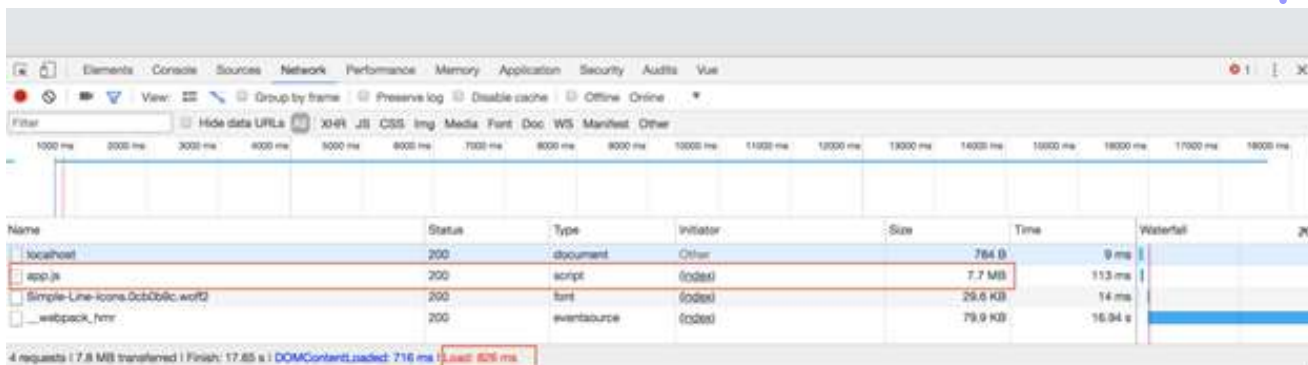In the router file, we had to change the line

From

```
import konvaEditor from './views/editor'
```

TO

```
const konvaEditor = () => import('./views/editor')
```

After using Async-Components and vue-router

Here it is obvious that the app.js file size has reduced from 9.5MB to 7.5MB which is approximately 21%.

Vue.js and HTML5 contenteditable

HTML5 contenteditable is another interesting feature we had tried with VueJS. When a made a contenteditable field as reactive with VueJS, we started observing it to behave strange.

We started noticing inconsistent issues like,

- duplicate content of the input value with both upper and lower case texts
- input values are non reactive, i.e., not changing to upper-case
- weird behavior on input with line breaks

All these issues are happening because of the same reason, and Evan You, the creator of Vue.js explains it this way

""when you use {{ }} bindings, Vue will attempt to diff the text nodes - which will likely be made out of sync by contenteditable because how browsers handle/split the text nodes are completely unpredictable.""

So, to solve this and make contenteditable fit properly with Vuejs reactive components, we had to use v-html and v-text directives to render the content within contenteditable node. This perfectly worked well using v-html.

While using v-html for user input fields, it is the developers responsibility to make sure to sanitize the input from user to avoid XSS attacks.

Conclusion

With VueJS, we have built a complete app which is performant, robust, functional, data-rich and the whole development went so smooth.   Thus choosing Vue js for our application, developer productivity went up and so their happiness and in turn our customers are happy.