BLOG

Build an App with Vue.js: From Authentication to Calling an API

Learn how to build a Vue.js app and how to easily get up and running with JWT authentication.

Ryan Chenkie

November 16, 2015

This post is out of date. We have an updated article that covers the same topic. You can find it here: Vuejs 2 Authentication Tutorial.

TL;DR: There are a ton of great JavaScript frameworks out there, and it can be a little overwhelming to keep up with them all. The learning curve for these frameworks can also be a bit steep. Vue.js is a breath of fresh air in this regard. In this tutorial, we'll see how easy it is to get up

and running with a Vue.js app and how we can easily add authentication to it. Check out the repo to get the code.

We are lucky to have plenty of JavaScript frameworks to choose from these days but, at the same time, it can be quite fatiguing to keep up with all of them. Some have a steep learning curve and require a lot of time for developers and their teams to become comfortable with. Others might be easy to learn, but perhaps lack some features that are crucial to a project. In either case, the level of complexity associated with learning a new framework can often hinder adoption and leave developers and teams frustrated.

If you're still choosing a framework for your Single Page App (SPA), or if you just want to learn a new technology, I believe Vue.js is one of the best frameworks you can pick. I love Vue.js for its simplicity and elegance, and how I can be super productive with it without needing to spend tons of time learning. In my experience, Vue.js **just works** and gets out of my way when developing applications.

Those are some anecdotal selling points, but let's cut to the hard facts: what exactly is Vue.js and how does it differ from other frameworks? If you're familiar with AngularJS 1.x, then Vue.js will probably look pretty familiar. In fact, Vue is heavily inspired by Angular. So what's the difference then? Essentially, Vue has a much simpler and cleaner API, is more flexible, and claims better performance.

Vue.js is firstly a view layer for applications that allows for reactive data binding and composable view components, and many developers use it only for their view layers. However, when combined with other tools in the Vue ecosystem, such as **vue-router**, **vue-resource**, and **vue-loader**, we get all the benefits of a great SPA framework while simplicity and developer experience are maintained.

What We'll Build: A Vue.js Authentication App

To demonstrate how easy it is to get up and running with a full SPA using Vue, we'll build a simple app that retrieves Chuck Norris quotes from a NodeJS backend. Vue can easily be mixed with other technologies, and you can use Vue for as much or as little of your app as you wish. To demonstrate Vue's full potential though, we'll build the whole front-end SPA with Vue components and follow Vue's pattern for large-scale applications. The front-end app will be totally decoupled from the back end, and we'll make HTTP requets to RESTful endpoints on our server.

We'll also demonstrate how we can easily add authentication to our Vue.js app. We'll put **Login** and **Signup** components in place to show how we can retrieve and save a user's JWT, and then send it back to the server for accessing protected endpoints.

Rather than listing out how Vue implements certain features and comparing them to other frameworks, we'll let the code speak for itself. Again, if you're familiar with Angular, it will be easy to reason about Vue's features and syntax.

Installation and Setup

Everything we need to start our component-based Vue.js app is on NPM. To get started, let's pull down what we need by creating our `package.json` file and specifying the packages we need. We can take full advantage of ES6 for our Vue components, and to make that happen, we'll use Babel. We'll also bundle everything up with Webpack and use **hot reloading** to see changes to our modules happen immediately. If you wish, you can also use other build tools (like Browserify) instead of Webpack.

```
// package.json

...

  "devDependencies": {
    "babel-core": "^6.1.2",
```

```
    "babel-loader": "^6.1.0",
    "babel-plugin-transform-runtime": "^6.1.2",
    "babel-preset-es2015": "^6.1.2",
    "babel-runtime": "^6.0.14",
    "css-loader": "^0.21.0",
    "style-loader": "^0.13.0",
    "vue-hot-reload-api": "^1.2.1",
    "vue-html-loader": "^1.0.0",
    "vue-loader": "^7.0.1",
    "webpack": "^1.12.3",
    "webpack-dev-server": "^1.12.1"
  },
  "dependencies": {
    "bootstrap": "^3.3.5",
    "vue-resource": "^0.1.17",
    "vue-router": "^0.7.5",
    "vue": "^1.0.7"
  }

...
```

Once the rest of our `package.json` file is in place, we can install everything.

```
npm install
```

To make **Webpack** work, we need a configuration file for it. Let's put in a file called `webpack.config.js` and populate it.

```
// webpack.config.js
```

```
module.exports = {
  // the main entry of our app
  entry: ['./src/index.js', './src/auth/index.js'],
  // output configuration
  output: {
    path: __dirname + '/build/',
    publicPath: 'build/',
    filename: 'build.js'
  },

  module: {
    loaders: [
      // process *.vue files using vue-loader
      { test: /\.vue$/, loader: 'vue' },
      // process *.js files using babel-loader
      // the exclude pattern is important so that we don't
      // apply babel transform to all the dependencies!
      { test: /\.js$/, loader: 'babel', exclude: /node_modules/ }
    ]
  },

  babel: {
    presets: ['es2015'],
    plugins: ['transform-runtime']
  }
}
```

In this config file, we're first specifying where our app's main entry point is and what the output path should be. The bundled JavaScript will be served as one file called `build.js`.

In the `module.loaders` array, we're setting up processing for our Vue components. These components have an extension of `.vue` and are processed by `vue-loader`.

That's all the configuration we need for now. Once we have our files in place, we just need to run `webpack-dev-server --inline --hot` to bundle and serve everything.

Setting Up the Back End

We're using our trusty nodejs-jwt-authentication-sample to retrieve Chuck Norris quotes. Clone the repo wherever you like (here we're putting it in a `server` directory) and follow the readme for installation steps.

Setting Up the Vue Components

Let's get started with the actual components for our app. But first, what exactly is a Vue component and how does it work? Vue components allow us to specify a **template**, a **script**, and **style** rules all in one file. If you're familiar with React, this will likely be familiar. This move toward composition and splitting the app into small components is helpful for maintainence and reasoning about the app.

To see how this works, let's start with the `Home` component.

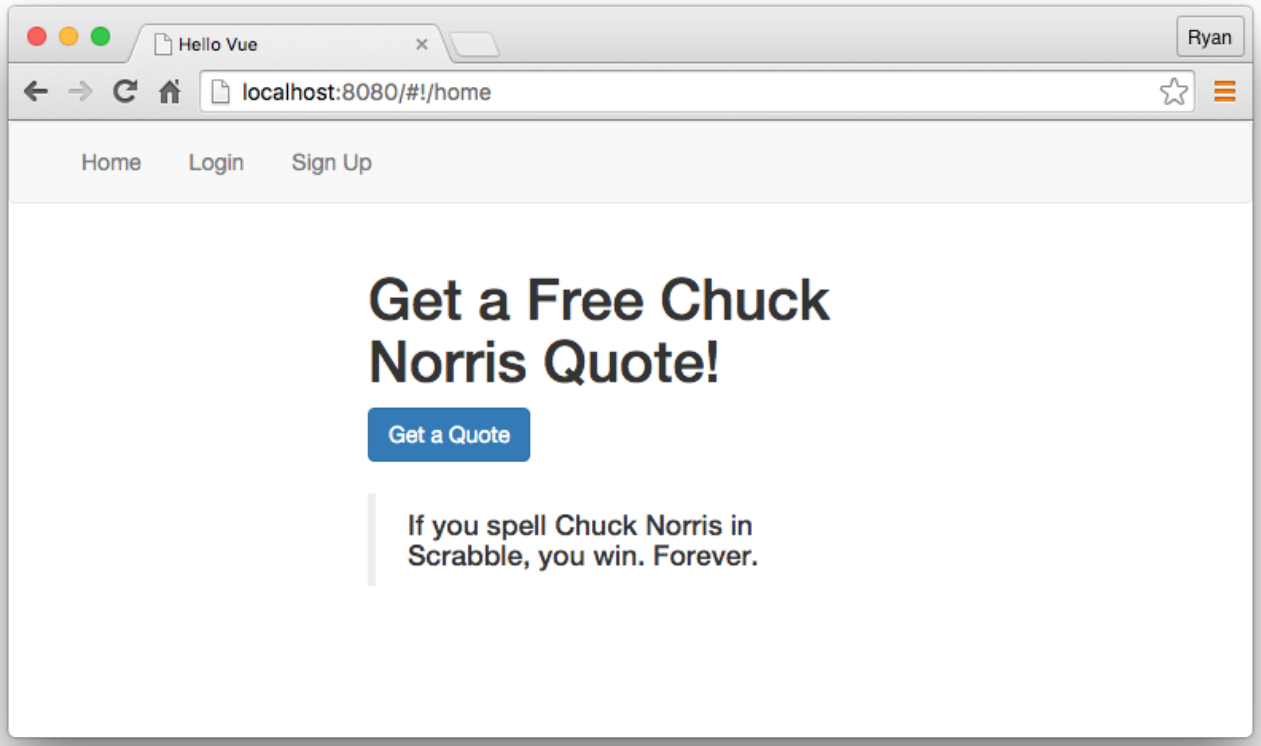
```
<!-- src/components/Home.vue -->

<template>
  <div class="col-sm-6 col-sm-offset-3">
    <h1>Get a Free Chuck Norris Quote!</h1>
    <button class="btn btn-primary" v-on:click="getQuote()">Get a Quote</button>
    <div class="quote-area" v-if="quote">
      <h2><blockquote>{{ quote }}</blockquote></h2>
    </div>
  </div>
</template>

<script>
```

```
export default {
  data() {
    return {
      quote: ''
    }
  },
  methods: {
    getQuote() {
      this.$http
        .get('http://localhost:3001/api/random-quote', (data) => {
          this.quote = data;
        })
        .error((err) => console.log(err))
    }
  }
}
</script>
```

The **template** is just some simple markup with a button that calls the method `getQuote`. We can notice some similarities to Angular in this code already. The template uses directives like `v-on:click` for click events, and `v-if` to conditionally show and hide the `quote-area` div. Vue also uses the double curly brace syntax for templating, which is how we take care of rendering the `quote` property.

The **script** area exports an object that is converted into a **component constructor function** by Vue. It has on it a `data` method and a `methods` object where we can register custom methods. When we want to register a data property to be used in the template, we need to do so in the `data` method. If we were to leave out the `quote` property from the returned object, that property wouldn't be rendered in the template. The `getQuote` method makes an HTTP request to our back end and sets the returned data on the `quote` property.

This gives us a good idea of what Vue components look like, but this won't work quite yet because we need to set up our app's main entry point, as well as a main **App** component. Here's how this component will render once everything is set up:



Setting Up index.js and App.vue

The `index.js` file is the place where we set up our main imports and do other configuration like routing. Let's set up everything we'll need for the whole app right now.

```
// src/index.js

import Vue from 'vue'
import App from './components/App.vue'
```

```
import Home from './components/Home.vue'
import SecretQuote from './components/SecretQuote.vue'
import Signup from './components/Signup.vue'
import Login from './components/Login.vue'
import VueRouter from 'vue-router'
import VueResource from 'vue-resource'
Vue.use(VueResource)
Vue.use(VueRouter)

export var router = new VueRouter()

// Set up routing and match routes to components
router.map({
  '/home': {
    component: Home
  },
  'secretquote': {
    component: SecretQuote
  },
  '/login': {
    component: Login
  },
  '/signup': {
    component: Signup
  }
})

// Redirect to the home route if any routes are unmatched
router.redirect({
  '*': '/home'
})

// Start the app on the #app div
router.start(App, '#app')
```

We're importing some components we've yet to create, as well as `vue-router` and `vue-resource`. For the app to recognize `vue-router` and `vue-resource`, we just need to call `Vue.use` on them. We can set up our route definitions with the simple `map` method on our instance of `vue-router`. The reason we're exporting this instance is so we can get a reference to it in our other components.

```
<!-- src/components/App.vue -->

<template>
  <nav class="navbar navbar-default">
    <div class="container">
      <ul class="nav navbar-nav">
        <li><a v-link="'home'">Home</a></li>
        <li><a v-link="'login'">Login</a></li>
        <li><a v-link="'signup'">Sign Up</a></li>
        <li><a v-link="'secretquote'">Secret Quote</a></li>
        <li><a v-link="'login'">Logout</a></li>
      </ul>
    </div>
  </nav>
  <div class="container">
    <router-view></router-view>
  </div>
</template>
```

To start out, our `App` component just needs a template. This top-level component has a `navbar` and exposes a `router-view` which is where our various routes will be rendered. Linking to routes is as simple as placing `v-link` on the anchor tags.

Finally, we need to be sure to place a div with an id of **app** within `index.html`, as this is where the whole app will be exposed.

```
<!-- index.html -->

...

<body>
<div id="app"></div>
  <script src="build/build.js"></script>
</body>

...
```

User Authentication - Login and Signup Components

To log users in, we'll need to make an HTTP request to our authentication endpoint and save the JWT that is returned in `localStorage`. We could place this logic right within our **Login** component, but we should really have a service to make it more reusable. Let's create a directory called **auth** and provide an `index.js` file there.

```
// src/auth/index.js

import {router} from '../index'

// URL and endpoint constants
const API_URL = 'http://localhost:3001/'
const LOGIN_URL = API_URL + 'sessions/create/'
const SIGNUP_URL = API_URL + 'users/'

export default {

  // User object will let us check authentication status
  user: {
    authenticated: false
```

```
  },

  // Send a request to the login URL and save the returned JWT
  login(context, creds, redirect) {
    context.$http.post(LOGIN_URL, creds, (data) => {
      localStorage.setItem('id_token', data.id_token)
      localStorage.setItem('access_token', data.access_token)

      this.user.authenticated = true

      // Redirect to a specified route
      if(redirect) {
        router.go(redirect)
      }

    }).error((err) => {
      context.error = err
    })
  },

  signup(context, creds, redirect) {
    context.$http.post(SIGNUP_URL, creds, (data) => {
      localStorage.setItem('id_token', data.id_token)
      localStorage.setItem('access_token', data.access_token)

      this.user.authenticated = true

      if(redirect) {
        router.go(redirect)
      }

    }).error((err) => {
      context.error = err
    })
  },

  // To log out, we just need to remove the token
```

```
    logout() {
      localStorage.removeItem('id_token')
      localStorage.removeItem('access_token')
      this.user.authenticated = false
    },

    checkAuth() {
      var jwt = localStorage.getItem('id_token')
      if(jwt) {
        this.user.authenticated = true
      }
      else {
        this.user.authenticated = false
      }
    },

    // The object to be passed as a header for authenticated requests
    getAuthHeader() {
      return {
        'Authorization': 'Bearer ' + localStorage.getItem('access_token')
      }
    }
  }
```

Our `auth` service exposes methods for logging users in and out, signing them up, and checking their authentication status. Note that "logging in" is just a matter of saving the JWT that is returned by the server. These methods and properties will all be useful throughout the app. For example, we can use the `user.authenticated` property to conditionally show elements in the app.

Implementing the Login Component

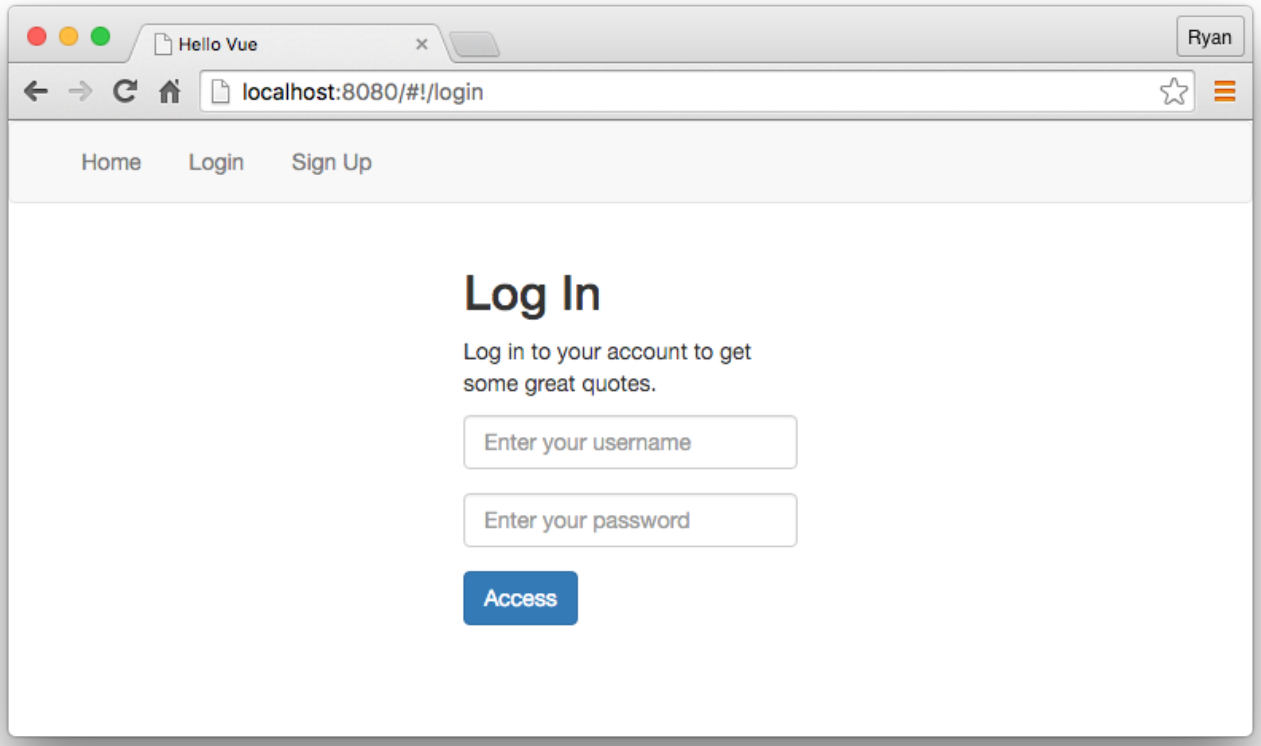
The **Login** component will need some HTML for the user inputs and a method to call our **auth** service.

```
<!-- src/components/Login.vue -->

<template>
  <div class="col-sm-4 col-sm-offset-4">
    <h2>Log In</h2>
    <p>Log in to your account to get some great quotes.</p>
    <div class="alert alert-danger" v-if="error">
      <p>{{ error }}</p>
    </div>
    <div class="form-group">
      <input
        type="text"
        class="form-control"
        placeholder="Enter your username"
        v-model="credentials.username"
      >
    </div>
    <div class="form-group">
      <input
        type="password"
        class="form-control"
        placeholder="Enter your password"
        v-model="credentials.password"
      >
    </div>
    <button class="btn btn-primary" @click="submit()">Access</button>
  </div>
</template>

<script>
import auth from '../auth'
export default {
  data() {
    return {
      // We need to initialize the component with any
      // properties that will be used in it
```

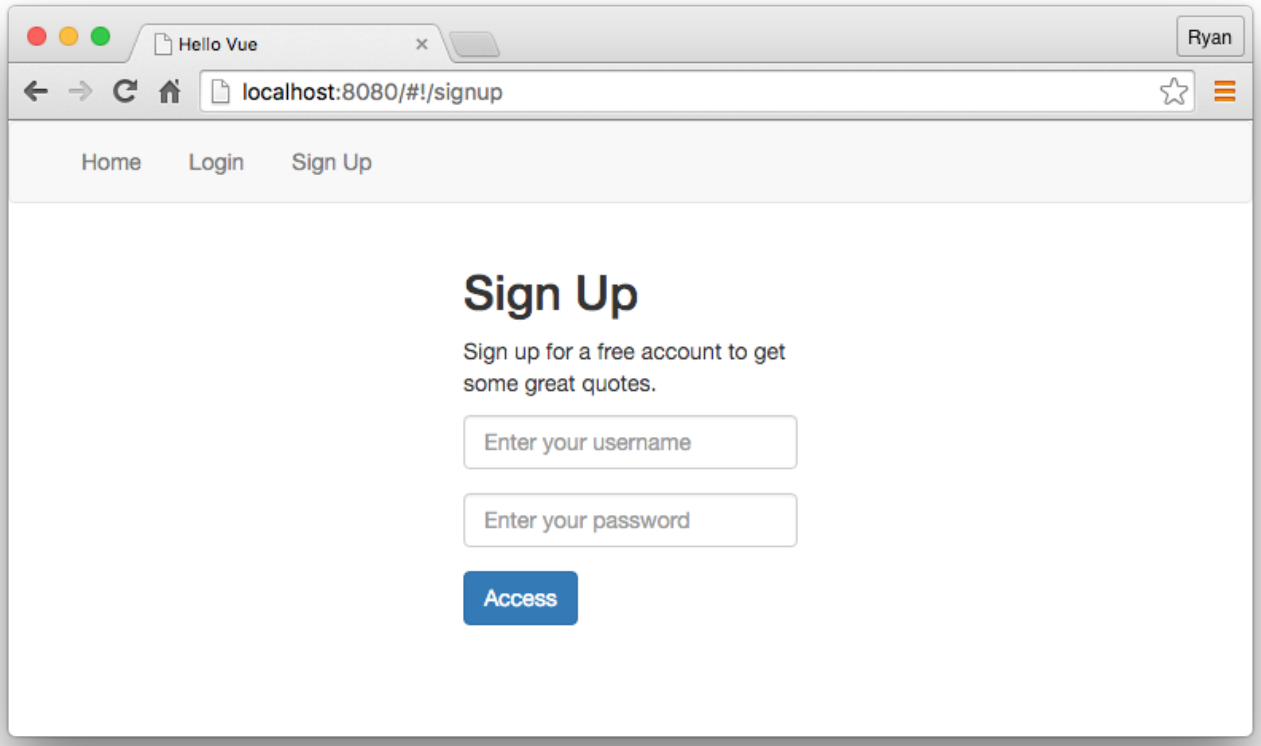
```
      credentials: {
        username: '',
        password: ''
      },
      error: ''
    }
  },
  methods: {
    submit() {
      var credentials = {
        username: this.credentials.username,
        password: this.credentials.password
      }
      // We need to pass the component's this context
      // to properly make use of http in the auth service
      auth.login(this, credentials, 'secretquote')
    }
  }

}
</script>
```

HTTP calls made with `vue-resource` require a component's context, and since we're abstracting that logic to a service, we need to pass the **Login** component's `this` context to the service. The second argument is the object with the user's credentials, and the third is the route we want to redirect to upon successfully authenticating.

Note that we're using `@click` on our submit button here. This is a shorthand alternative to `v-on:click`.



The **Signup** component is nearly identical, except it will use the `signup` method from the `auth` service to send the user's credentials to a different endpoint.



Implementing the Secret Quote Component

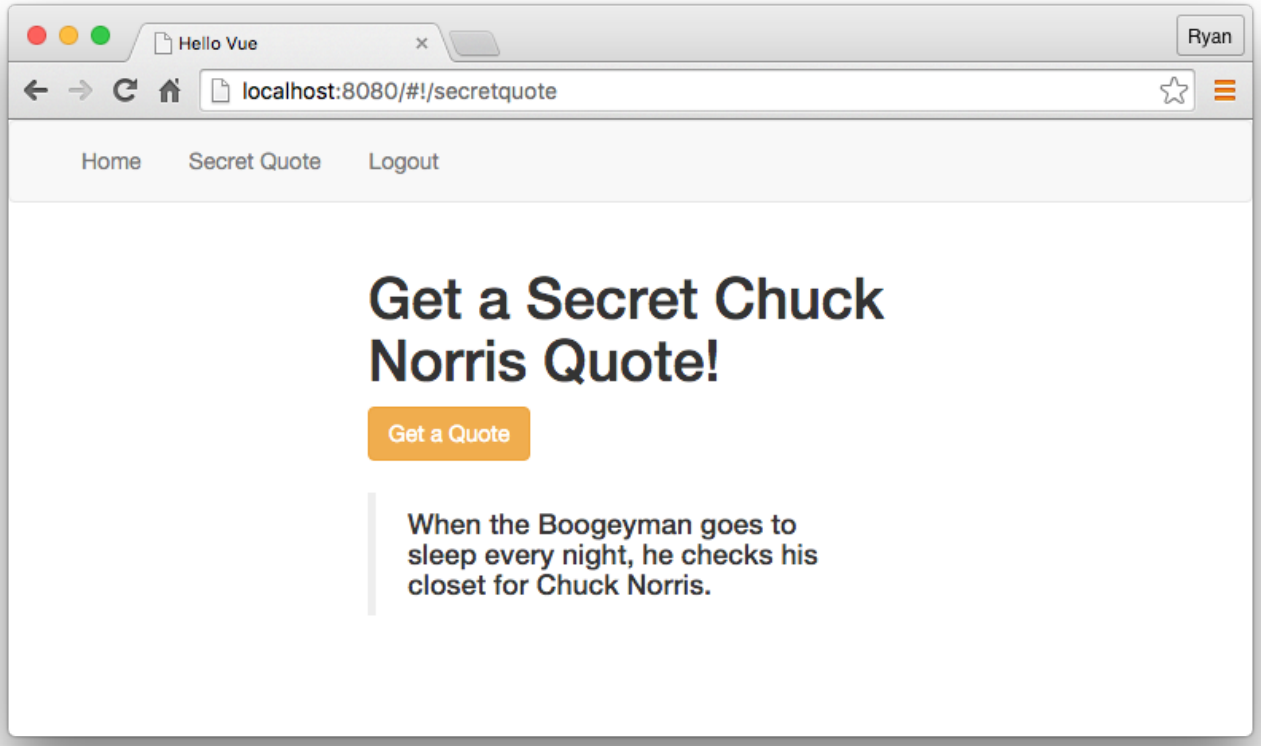
When a user successfully authenticates, they will be able to access the **secret-quote** route from the API. The **SecretQuote** component will look similar to the **Home** component, but we'll attach the user's JWT as an `Authorization` header when requests are sent.

```
<!-- src/components/SecretQuote.vue -->

<template>
  <div class="col-sm-6 col-sm-offset-3">
    <h1>Get a Secret Chuck Norris Quote!</h1>
    <button class="btn btn-warning" v-on:click="getQuote()">Get a Quote</button>
    <div class="quote-area" v-if="quote">
```

```
      <h2><blockquote>{{ quote }}</blockquote></h2>
    </div>
  </div>
</template>

<script>
import auth from '../auth'
export default {
  data() {
    return {
      quote: ''
    }
  },
  methods: {
    getQuote() {
      this.$http
        .get('http://localhost:3001/api/protected/random-quote', (data) => {
          this.quote = data;
        }, {
          // Attach the JWT header
          headers: auth.getAuthHeader()
        })
        .error((err) => console.log(err))
    }
  },
  route: {
    // Check the users auth status before
    // allowing navigation to the route
    canActivate() {
      return auth.user.authenticated
    }
  }
}
</script>
```

The header is attached by providing an options object as the third argument to the HTTP request. To get the JWT header, we call the `getAuthHeader` method from the `auth` service.

Since we don't want users to access this route if they are not authenticated, we can tap into `vue-router` 's transition pipeline. Specifically, we use the `canActivate` hook and consult the `auth` service to check if the user is authenticated. If so, the route can be navigated to.



Final Touches

We're nearly done, but there are a couple of improvements we can make before we finish out. It would be good to conditionally show and hide menu items based on the user's `auth` status. To do that, we'll use `v-if` .

```
<!-- src/components/App.vue -->

<template>
  <nav class="navbar navbar-default">
    <div class="container">
      <ul class="nav navbar-nav">
        <li><a v-link="'home'">Home</a></li>
        <li><a v-link="'login'" v-if="!user.authenticated">Login</a></li>
        <li><a v-link="'signup'" v-if="!user.authenticated">Sign Up</a></li>
        <li><a v-link="'secretquote'" v-if="user.authenticated">Secret Quote</a></li>
        <li><a v-link="'login'" v-if="user.authenticated" @click="logout()">Logout</a></li>
      </ul>
    </div>
  </nav>
  <div class="container">
    <router-view></router-view>
  </div>
</template>

<script>
import auth from '../auth'
export default {
  data() {
    return {
      user: auth.user
    }
  },
  methods: {
    logout() {
      auth.logout()
    }
  }
}
</script>
```

The `auth` service sets the user's authentication status when the `login` or `logout` methods are used, but if the page is refreshed or the app closed and reopened, that status will be lost. To get around that, let's call `checkLogin` when the app is first loaded.

```
// src/index.js

...

import auth from './auth'

// Check the users auth status when the app starts
auth.checkAuth()

...
```

Setting Global Headers

When we make a request to the protected `secret-quote` route, we pass an options object that has the `Authorization` header and user's JWT `access_token` on it. If, instead, we wanted to globally set the `Authorization` header and not worry about setting it on each HTTP request, we could set up a global header.

```
// src/index.js

...

// Optional
Vue.http.headers.common['Authorization'] = auth.getAuthHeader();

...
```

Aside: Using Auth0 With Your Vue.js App

Auth0 issues JSON Web Tokens on every login for your users. This means that you can have a solid identity infrastructure, including single sign-on, user management, support for social identity providers (Facebook, Github, Twitter, etc.), enterprise identity providers (Active Directory, LDAP, SAML, etc.) and your own database of users with just a few lines of code.

We can easily set up authentication in our Vue.js apps with Auth0's Login Page. If you don't already have an Auth0 account, sign up for one now. Navigate to the Auth0 management dashboard, select **Applications** from the navigational menu, then select the app you want to connect with **Vue.js**.

> "We can easily set up authentication in our Vue.js apps with Auth0's Login Page"
>
> TWEET THIS

Step 1: Include auth0.js

```
<!-- index.html -->

...

<!-- Auth0.js script -->
<script src="https://cdn.auth0.com/js/auth0/9.0.0/auth0.min.js"></script>

...
```

Step 2: Configure an instance of auth0.js

```
// src/index.js

...

export var webAuth = new auth0.WebAuth({
  domain: 'YOUR_DOMAIN',
  clientID: 'YOUR_CLIENT_ID',
  responseType: 'token',
  redirectUri: 'YOUR_REDIRECT_URI'
});
...
```

Step 3: Call auth0.js from a Vue.js Component

```
<!-- src/components/Login.vue -->

<template>
  <div class="col-sm-4 col-sm-offset-4">
    <h2>Log in</h2>
    <button class="btn btn-primary" @click="login()">Log in</button>
  </div>
</template>

<script>
import {webAuth} from '../index'

export default {
  ready() {
    // Parse the hash
    if (window.location.hash) {
      webAuth.parseHash({ hash: window.location.hash }, function(err, authResult) {
        if (err) {
          return console.log(err);
```

```
        }
        if (authResult) {
          webAuth.client.userInfo(authResult.accessToken, function(err, user) {
            localStorage.setItem('profile', JSON.stringify(user))
            localStorage.setItem('id_token', authResult.idToken)
          });
        }
      });
    }
  },
  methods: {
    login() {
      // Redirect to the login page
      webAuth.authorize();
    },
    logout() {
      // Remove the profile and token from localStorage
      localStorage.removeItem('profile')
      localStorage.removeItem('id_token')
    }
  }
}

</script>
```

Important API Security Note: If you want to use Auth0 authentication to authorize *API requests*, note that you'll need to use a different flow depending on your use case. Auth0 `idToken` should only be used on the client-side. Access tokens should be used to authorize APIs. You can read more about making API calls with Auth0 here.

Wrapping Up

We have many great choices for SPA frameworks these days, and this can easily cause analysis paralysis. Further, it can be fatiguing to keep up with the pace of new framework development and to learn their ins and outs.

I find Vue.js to be a breath of fresh air in this regard. The library and ecosystem are feature-rich, but they get out of your way as you develop your apps. I've found that the learning curve with Vue.js isn't as steep as it can be with other frameworks, and from my experience, it seems to always **just work**.

As we saw in this tutorial, we can easily add authentication to our Vue.js apps. Also, Vue's HTTP library, `vue-resource`, makes it trivial to send requests with an `Authorization` header.

I hope you'll consider Vue.js for your next project--it really is great to work with!

This. is. awesome.

I really appreciate how the blog posts showcases the technology not the Auth0 product. Even the backend repo is built with open source npm packages

Thank you. For sure going to look into Vue.js for my next project!

14 ^ | v • Reply • Share ›

Paweł Gościcki • 2 years ago

This tutorial is really great, but there are three problems with it. Two big and one small.

The first one is that the `getAuthHeader()` should return a String, not an object (as spotted by **@dotism**). This is easily solvable:

```
getAuthHeader () {
  return 'Bearer ' + window.localStorage.getItem('token')
}
```

The second problem is that if you would like to have globally set http Authorization headers for all `$http.*` requests, you cannot do it like presented above. The reason is that it will get set only once, with whatever value is there in the `localStorage`, on application start. If it is null on first app bootup, then after logging in, the headers won't be updated and your app will basically not be working.

The solution is to dynamically update this header on `login()` and `logout()` actions:

```
// main.js / index.js
Vue.http.headers.common['Authorization'] = auth.getAuthHeader()

// auth.js
import Vue from 'vue'
import {router} from './main'
```

see more

3 ^ | v • Reply • Share ›

ryanchenkie ➚ Paweł Gościcki • 2 years ago

Seems like things have changed with Vue since the time of writing, thanks for catching these! Would you be up for sending a PR to address these in the sample? https://github.com/auth0-bl...

Thanks :D

^ | v • Reply • Share ›

dotism • 2 years ago

This is an incredibly helpful guide, thank you for it.

One thing that I had some trouble with was setting the global authorization header with:

Vue.http.headers.common['Authorization'] = auth.getAuthHeader();

This wasn't working for me because auth.getAuthHeader returns an object when all you need is the string: 'Bearer ' + JWT. Setting things up this way works perfectly:

Vue.http.headers.common['Authorization'] = 'Bearer ' + localStorage.getItem('id_token')

I suppose one could also write a different function to handle returning the string:

```
//auth/index.js
getAuthHeader() {
return { 'Authorization': getJwtBearer() }
}
getJwtBearer() {
return 'Bearer ' + localStorage.getItem('id_token')
}

//index.js
Vue.http.headers.common['Authorization'] = auth.getJwtBearer()
```

1 ^ v • Reply • Share ›

wishinghand • 2 years ago

How do you implement Bootstrap? I don't see where it gets included.

1 ^ v • Reply • Share ›

Duncan Lock ➚ wishinghand • 2 years ago

Check out the code for this example: https://github.com/auth0-bl... - and have a look at the `index.html` file: https://github.com/auth0-bl... - it's in there.

^ v • Reply • Share ›

Duncan Lock ➚ wishinghand • 2 years ago

It's included as a dependency in the `packages.json`.

^ v • Reply • Share ›

Danil Timonin ➚ Duncan Lock • 3 months ago

It's wrong.
Must be require('node_modules/bootstrap/dist/css/bootstrap.min.css') for main js.

^ v • Reply • Share ›

Victor Kurauchi • 2 years ago

Hey, nice article, helped a lot!
I just think there's a typo in Home.vue. For me, it worked this way:
this.$http.get('http://localhost:3001/api/random-quote').then((response) => {})

1 ^ v • Reply • Share ›

YI Yang ➚ Victor Kurauchi • 2 years ago

Yes! Coz if I do not use this way to fetch data, an error occurred "context.$http.get()" is not a function. The wired part is that even I got this error, it still can send the post request. But any callback action will not be executed.

^ v • Reply • Share ›

niuniu ➚ Victor Kurauchi • 2 years ago

yes, vue-resource recommend to fetch data in this way

^ | v • Reply • Share ›

Samson Iyanda • 4 months ago

AWESOME

^ | v • Reply • Share ›

Богдан Вовчук • a year ago

Token should be verified, because I can set up to localStorage any 'token' and this logic will grant auth permition

^ | v • Reply • Share ›

Ed • a year ago

Hello Ryan, I know this blog was written a while ago.

Think I had the new version 2 webpack installed globally and ran into issues, and saw in the package.json
"webpack": "^1.15.0",
"webpack-dev-server": "^1.12.1"

So I uninstalled my global version.. I'm at the point in the blog where we go: webpack-dev-server --inline --hot

And get:
module.js:529
throw err;
^

Error: Cannot find module 'webpack'
at Function.Module._resolveFilename (module.js:527:15)
at Function.Module._load (module.js:476:23)
at Module.require (module.js:568:17)
at require (internal/module.js:11:18)
at Object.<anonymous> (/usr/local/lib/node_modules/webpack-dev-server/lib/Server.js:21:17)

I'm guessing I should try to run some localized and earlier version of webpack? Wonder if you could update your blog to clarify? Thank you

^ | v • Reply • Share ›

shiv anand • a year ago

awesome tutorial.. Learnt a lot about JWT. THANKS.. _/_

^ | v • Reply • Share ›

Mayra Olivo • a year ago

Hey.. could you help me.. I created a local rest api to consume in my vue app but it doesn´t work ... the uri works well in the browser and I´ve tried with other web api´s on vue and the data comes

right... Have any idea what happens between local apis and vue apps ???

Reply • Share ›

Joe Sestrich • 2 years ago

I already asked this of Stefan, below, but I figured I should ask more directly--

In trying out your code running, I am a bit confused: I don't see anything in server.js that would serve up index.html. When I attempt to access the website at localhost:3001 all I get is "Cannot GET /". How is server.js supposed to send out index.htmlin response to a get "/" ????

Reply • Share ›

Fola Richards • 2 years ago

Can anyone explain how to integrate Auth0 lock widget?

Reply • Share ›

adobot Mod ➚ Fola Richards • 2 years ago

The Auth0 Docs regarding Lock are probably the best way. From there you can choose your framework of choice and even download sample code that has Lock already integrated. https://auth0.com/docs/libr...

Reply • Share ›

Stefan Jarina • 2 years ago

Hello, thanks for this tutorial. It is still for Vue1 though which is kind of an obsolete right now so I have reworked it for Vue2 with Axios instead of Vue-Resource.
All is updated to newest versions including webpack and webpack file, etc.
I am using yarn though, but it should work with npm as well, just ignore the yarn.lock file I guess.
https://github.com/stefanja...
Cheers!!

Reply • Share ›

Joe Sestrich ➚ Stefan Jarina • 2 years ago

In trying out your code I am a bit confused: I don't see anything in server.js that would server up index.html. When I attempt to access the website at localhost:3001 all I get is "Cannot GET /". How is server supposed to send out index.html????

Reply • Share ›

Stefan Jarina ➚ Joe Sestrich • 2 years ago

Hello, that is actually correct - server.js is used only to expose the REST API to our frontend Vue application. To serve our frontend is used webpack development server.
so you need 2 terminals and in one run `yarn start` (or `npm start`) inside the server directory, then in second terminal from your apps root run

`yarn run dev` (or ¨npm run dev¨)
So at the end you'll have 2 servers
- one that serves REST API on port 3001 -- this is eventually a server.js

- one that serves Vue frontend on port 3000 -- this webpack dev server

Hopefully it explain it - I will try to rewrite instructions on github to explain it a bit better.

Reply • Share ›

Joe Sestrich ➚ Stefan Jarina • 2 years ago

Wow that was not obvious to me. Thanks so much for your reply!
This brings up several more questions though--
1: how do I access the site from a browser? Neither localhost:3000 nor localhost:3001 brings up the site
2: When I run dev, webpack says it is running on localhost:8080. Is that what you expect?
3: How does the vue frontend know what port to use for the REST API? It sure looks like it is hard coded; I would guess there is a more robust standard way to handle this?

Reply • Share ›

Stefan Jarina ➚ Joe Sestrich • 2 years ago

Ah yes,
1. & 2. maybe webpack starts on port 8080, my mistake - I usually specify ports myself. website should load on port 8080 then. (also webpack for me open browser automatically)

3. normally you would use something like 'my-website.com/api/v1/' or 'api.my-website.com' so harcoded address is not a big problem as it doesn't change as often -- or you simply have some config where you specify the base of your URL and then when you need to change it you just change one variable and it changes for your whole application.

This is let's say fast dirty example of Vue2 + oauth so I just hardcoded it there. for a real application you want to use a more inteligent way to handle some parts.
Best would be to find some full course/tutorial for some bigger vuejs2 application and then just use part of this my example to implement OAuth there. I can recommend this one: https://www.udemy.com/vuejs... but it is quite expensive, I have never seen better tutorial for VueJS 2 though.

Also using 'vuex' might change it a bit as well, as usually all the communication with backend is done in 'Actions' so you specify url for api just once at the top of file and you're done. but that is another and quite large topic.

Reply • Share ›

Joe Sestrich ➚ Stefan Jarina • 2 years ago

hmmm... port 8080 doesn't work either. Is there some mechanism in webpack to log accesses, or otherwise get some info about what might be

going on? The console prints out a bunch of info, but it all looks like it is working normally. I just get no response to a localhost:8080 request from a browswer.

Reply • Share ›

Stefan Jarina ➚ Joe Sestrich • 2 years ago

Write me directly to stefan@jarina.cz and send output from console maybe. I'll have more time at work tomorrow to check it up closely.

Reply • Share ›

Joe Sestrich ➚ Stefan Jarina • 2 years ago

I stumbled upon the solution to my problems-- It turns out that webpack-dev-server only listens on the loopback interface of the machine it is running on. It turns out that I am running my browser on a machine different from the one running the server. I had to add a switch to the webpack-dev-server command: --host 0.0.0.0

Reply • Share ›

Stefan Jarina ➚ Joe Sestrich • 2 years ago

Hey, this week there is a "teacher appreciation week" and a lot of courses are for 10usd/eur including the mentioned https://www.udemy.com/vuejs... Thought you might like it as it is probably most complete course on Vue 2 I have ever seen and for 10 bucks it is just awesome :-)

Reply • Share ›

adobot Mod ➚ Stefan Jarina • 2 years ago

This is awesome Stefan. Would you be interested in creating a PR and writing a blog post for this? Email me at ado@auth0.com and we can discuss :)

Reply • Share ›

egemersoz • 2 years ago

Any chance you can update this guide to use Vue 2? It was released Sept 30th 2016.

Reply • Share ›

Sebastian Peyrott Mod ➚ egemersoz • 2 years ago

We will probably take a look at Vue 2 at some point, but we cannot commit to any dates for now.

Reply • Share ›

Samuel Desconsi • 2 years ago

I'm getting the error 'localStorage' is not defined, someone knows why ?

Reply • Share ›

awwester ➚ Samuel Desconsi • 2 years ago

I had the same thing, it has to do with es6lint. If you put /* global localStorage */ at the top

of your file it will work.

1 • Reply • Share ›

Kim Maida Mod ➚ Samuel Desconsi • 2 years ago

Hello Samuel,

Are you trying to use localStorage server-side? The web storage API is only available in the browser, so attempting to access local storage in JS that runs on the server would produce this error. Can you share a repo and point to where you're getting this error?

1 • Reply • Share ›

Samuel Desconsi ➚ Kim Maida • 2 years ago

Hello Kim, hello awwester, thanks for answer me, but i'd discovered what i'm doing wrong, just missing to put 'window' in the localStorage call. Thanks again. \o

1 • Reply • Share ›

Leandro ➚ Samuel Desconsi • 2 years ago

workaround:

```
window.localStorage.getItem('api_token')
```

• Reply • Share ›

Eric Reynolds • 2 years ago

Hi,

Thanks for the article I'm learning a lot from it! However I think you're very subtly abusing Vue's rules about information flow between parent and child components.

What I couldn't get my head around is how Vue knows to re-render the main app component (i.e. show/hide nav links) after login has been performed. Then I realised that it's because, indirectly, the Login component sets the App component's data, through the auth module. In chrome dev tools I've noticed that, by the time you get to the auth.login method, the user object has been instrumented by Vue, i.e. the authenticated property has been replaced with a getter/setter functions.

Is this good practice in Vue? Or should the Login component emit a "successful login" event and allow the App component to update its own data? Sorry if this a dumb question, I'm new to Vue :)

Thanks!

Best regards,

Eric

• Reply • Share ›

ryanchenkie ➚ Eric Reynolds • 2 years ago

Good question--I think it depends on how you want to approach state management in your

application, which is largely a decision that is made after considering the tradeoffs in light of how large/complex the app is/will be. Letting the App component handle changes to its data instead of a service like Auth is probably a good idea, but at that point you should probably also be implementing something like vuex.

Reply • Share ›

Mazino S Ukah ryanchenkie • 2 years ago

nice article, however i am having some issues with user state management, i'm using vuex but on page reload the data is lost, so i decided to also store user information/state in localforage too. when the page is refreshed i then update/sync vuex with the localforage. the problem is , if for instance the user changes/updates his name/credentials on another browser, the current browser is not aware of that change as it still has the old credentials stored in its localforage. please how do i solve this issue ? do i need to send the authenticated user with each response from the server ?

Reply • Share ›

Duncan Lock • 2 years ago

Is storing your JWT in `localStorage` a good idea? This potentially makes it available to any maliciously injected JS/XSS attacks, allowing your JWT (bearer token) to be used/copied/ex-filtrated? Storing it in a secure/HTTPOnly cookie would avoid this, I think (at the price of making it inaccessible to JS altogether)?

Reply • Share ›

ryanchenkie Duncan Lock • 2 years ago

There are pros and cons to both. Storing it in a cookie isn't a silver bullet as you'd need to protect against CSRF. Ultimately the way you store the token depends on the needs of you app. In my opinion, XSS is something that should be guarded against in an application anyway.

Reply • Share ›

Subscribe to more awesome content!

your@email.com

Related Posts

Angular Authentication Tutorial

Ado Kukic

ReactJS Authentication Tutorial

Prosper Otemuyiwa

PRODUCT

Pricing

Why Auth0

10800 NE 8th St.

Suite 600

Bellevue, WA 98004

+1 (888) 235-2699

+1 (425) 312-6521

Support Center