February 15, 2018

Build a Basic CRUD App with Vue.js and Node

Brandon Parise

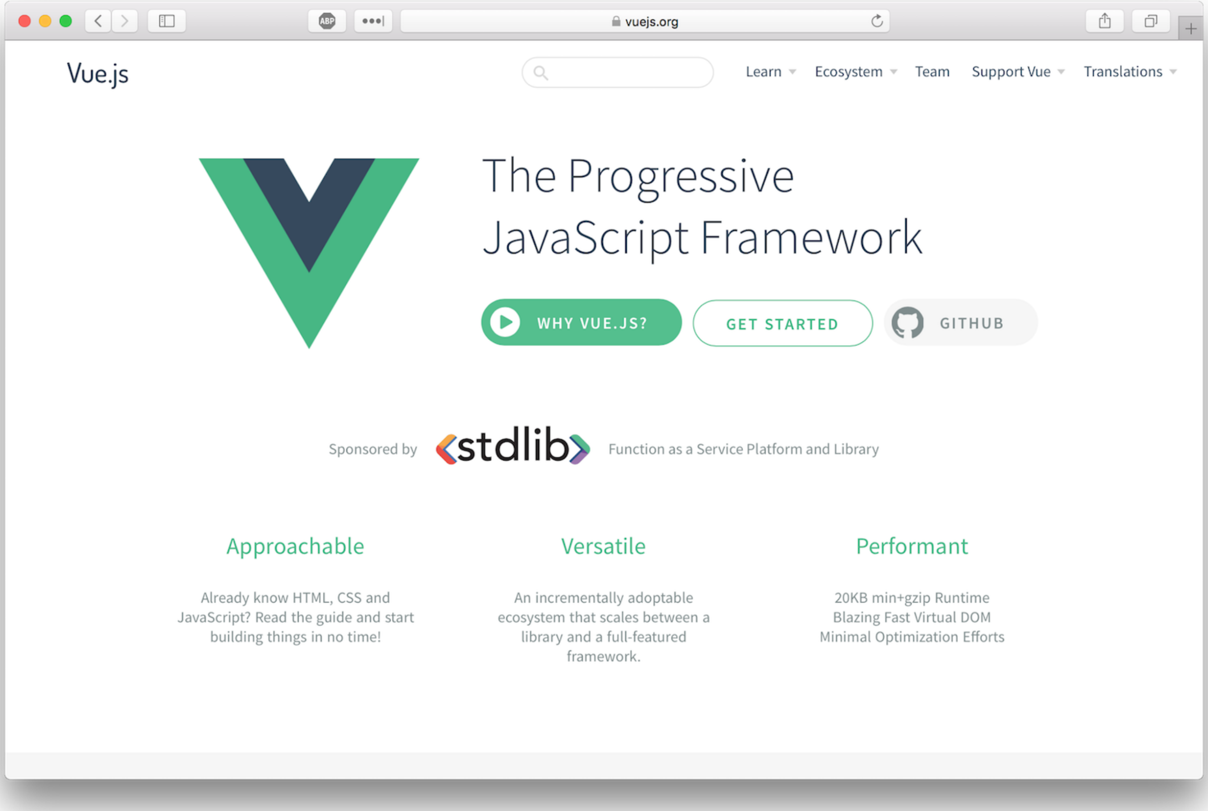
I've danced the JavaScript framework shuffle for years starting with jQuery, then on to Angular. After being frustrated with Angular's complexity, I found React and thought I was in the clear. What seemed simple on the surface ended up being a frustrating mess. Then I found Vue.js. It just felt right. It worked as expected. It was fast. The documentation was incredible. Templating was eloquent. There was a unanimous consensus around how to handle state management, conditional rendering, two-way binding, routing, and more.

This tutorial will take you step by step through scaffolding a Vue.js project, offloading secure authentication to Okta's OpenID Connect API (OIDC), locking down protected routes, and performing CRUD operations through a backend REST API server. This tutorial uses the following technologies but doesn't require intimate knowledge to follow along:

- Vue.js with vue-cli, vue-router, and Okta Vue SDK
- Node with Express, Okta JWT Verifier, Sequelize, and Epilogue

About Vue.js

Vue.js is a robust but simple Javascript framework. It has one of the lowest barriers to entry of any modern framework while providing all the required features for high performance web applications.


vuejs.org
Vue.js
Learn
Ecosystem
Team
Support Vue
Translations
The Progressive JavaScript Framework
WHY VUE.JS?
GET STARTED
GITHUB
Sponsored by stdlib Function as a Service Platform and Library
Approachable
Already know HTML, CSS and JavaScript? Read the guide and start building things in no time!
Versatile
An incrementally adoptable ecosystem that scales between a library and a full-featured framework.
Performant
20KB min+gzip Runtime
Blazing Fast Virtual DOM
Minimal Optimization Efforts

This tutorial covers two primary builds, a frontend web app and backend REST API server. The frontend will be a single page application (SPA) with a homepage, login and logout, and a posts manager.

Okta's OpenID Connect (OIDC) will handle our web app's authentication through the use of Okta's Vue SDK. If an unauthenticated user navigates to the posts manager, the web app should attempt to authenticate the user.

The server will run Express with Sequelize and Epilogue. At a high level, with Sequelize and Epilogue you can quickly generate dynamic REST endpoints with just a few lines of code.

You will use JWT-based authentication when making requests from the web app and Okta's JWT Verifier in an Express middleware to validate the token. Your app will expose the following endpoints which all require requests to have a valid access token.

```
- GET /posts
- GET /posts/:id
- POST /posts
- PUT /posts/:id
- DELETE /posts/:id
```

Create Your Vue.js App

To get your project off the ground quickly you can leverage the scaffolding functionality from vue-cli. For this tutorial, you are going to use the progressive web app (PWA) template that includes a handful of features including webpack, hot reloading, CSS extraction, and unit testing.

> If you're not familiar with the tenets of PWA, check out our ultimate guide to progressive web applications.

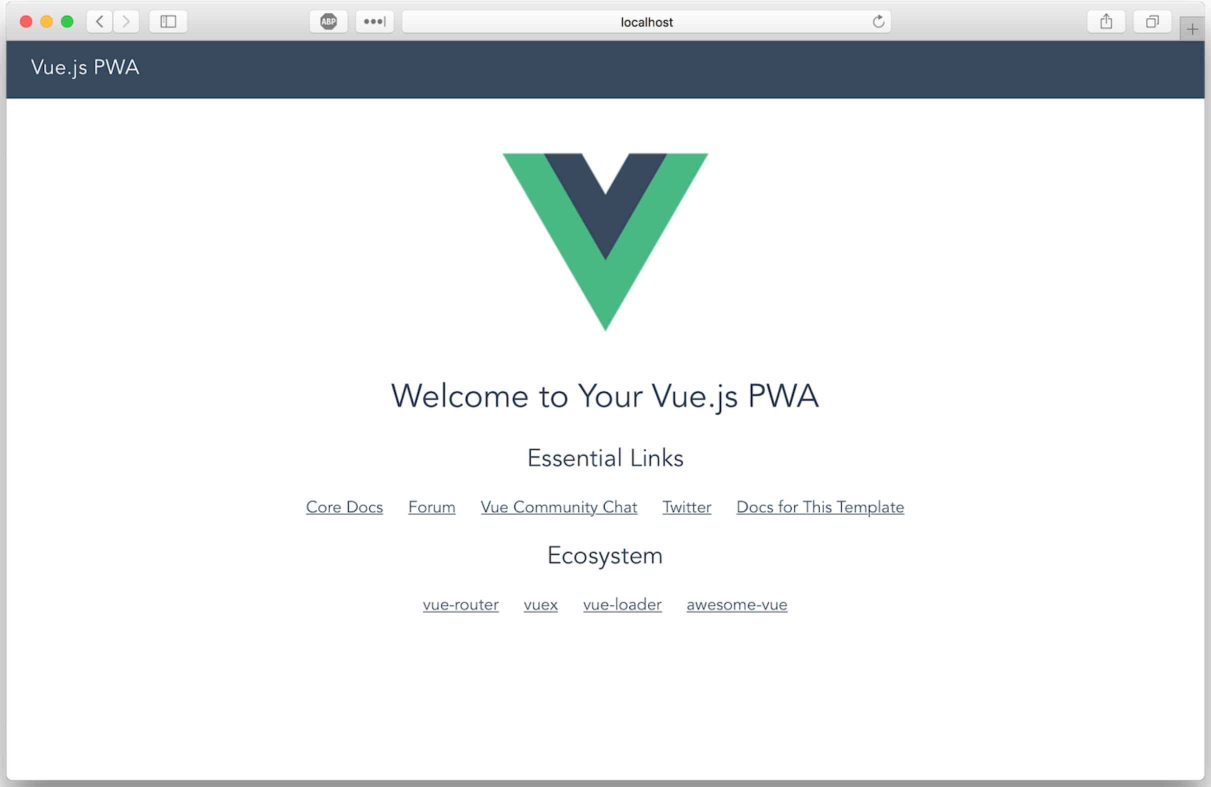
To install `vue-cli` run:

```
npm install -g vue-cli@2.9.3
```

Next, you need to initialize your project. When you run the `vue init` command just accept all the default values.

```
vue init pwa my-vue-app
cd ./my-vue-app
npm install
npm run dev
```

Point your favorite browser to `http://localhost:8080` and you should see the fruits of your labor:



Welcome to Your Vue.js PWA

Extra Credit: Check out the other templates available for `vue-cli`.

Install Bootstrap

Let's install bootstrap-vue so you can take advantage of the various premade components (plus you can keep the focus on functionality and not on custom CSS):

```
npm i bootstrap-vue@2.0.0-rc.7 bootstrap@4.1.0
```

To complete the installation, modify `./src/main.js` to include bootstrap-vue and import the required CSS files. Your `./src/main.js` file should look like this:

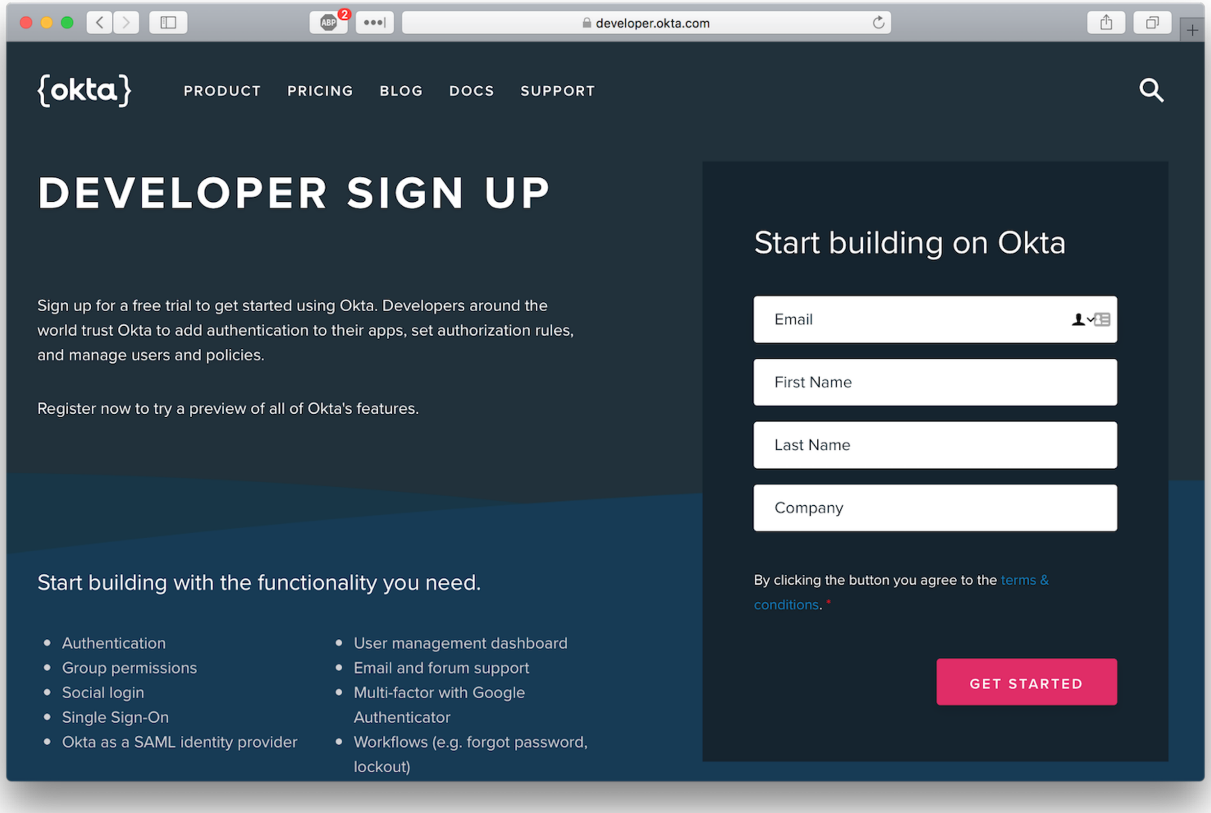
```
// The Vue build version to load with the `import` command
// (runtime-only or standalone) has been set in webpack.base.conf with an alias.
import Vue from 'vue'
import App from './App'
import router from './router'
import BootstrapVue from 'bootstrap-vue'
import 'bootstrap/dist/css/bootstrap.css'
import 'bootstrap-vue/dist/bootstrap-vue.css'

Vue.use(BootstrapVue)
Vue.config.productionTip = false

/* eslint-disable no-new */
new Vue({
  el: '#app',
  router,
  template: '<App/>',
  components: { App }
})
```

Add Authentication with Okta

Dealing with authentication in a web app is the bane of every developer's existence. That's where Okta comes in to secure your web applications with minimal code. To get started, you will need to create an OIDC application in Okta. Sign up for a forever-free developer account (or log in if you already have one).



DEVELOPER SIGN UP

Sign up for a free trial to get started using Okta. Developers around the world trust Okta to add authentication to their apps, set authorization rules, and manage users and policies.

Register now to try a preview of all of Okta's features.

Start building with the functionality you need.

- Authentication
- Group permissions
- Social login
- Single Sign-On
- Okta as a SAML identity provider
- User management dashboard
- Email and forum support
- Multi-factor with Google Authenticator
- Workflows (e.g. forgot password, lockout)

Start building on Okta

Email

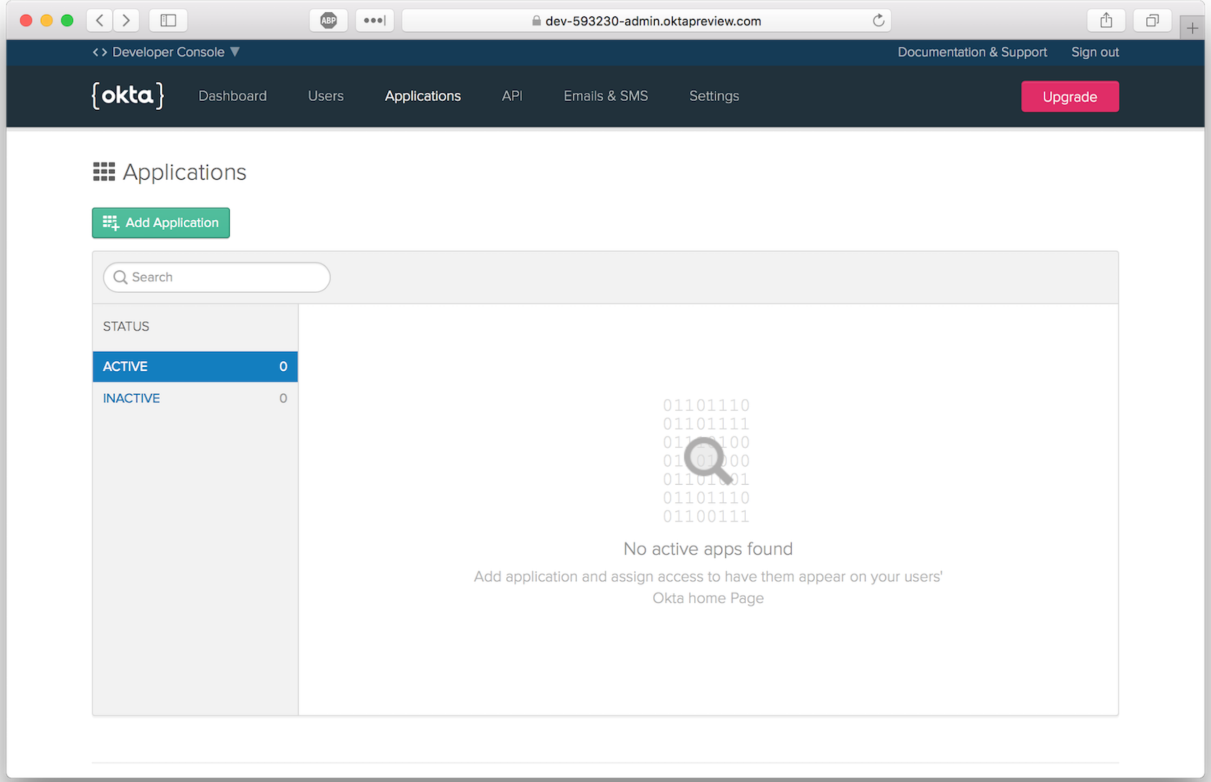
First Name

Last Name

Company

By clicking the button you agree to the terms & conditions. *

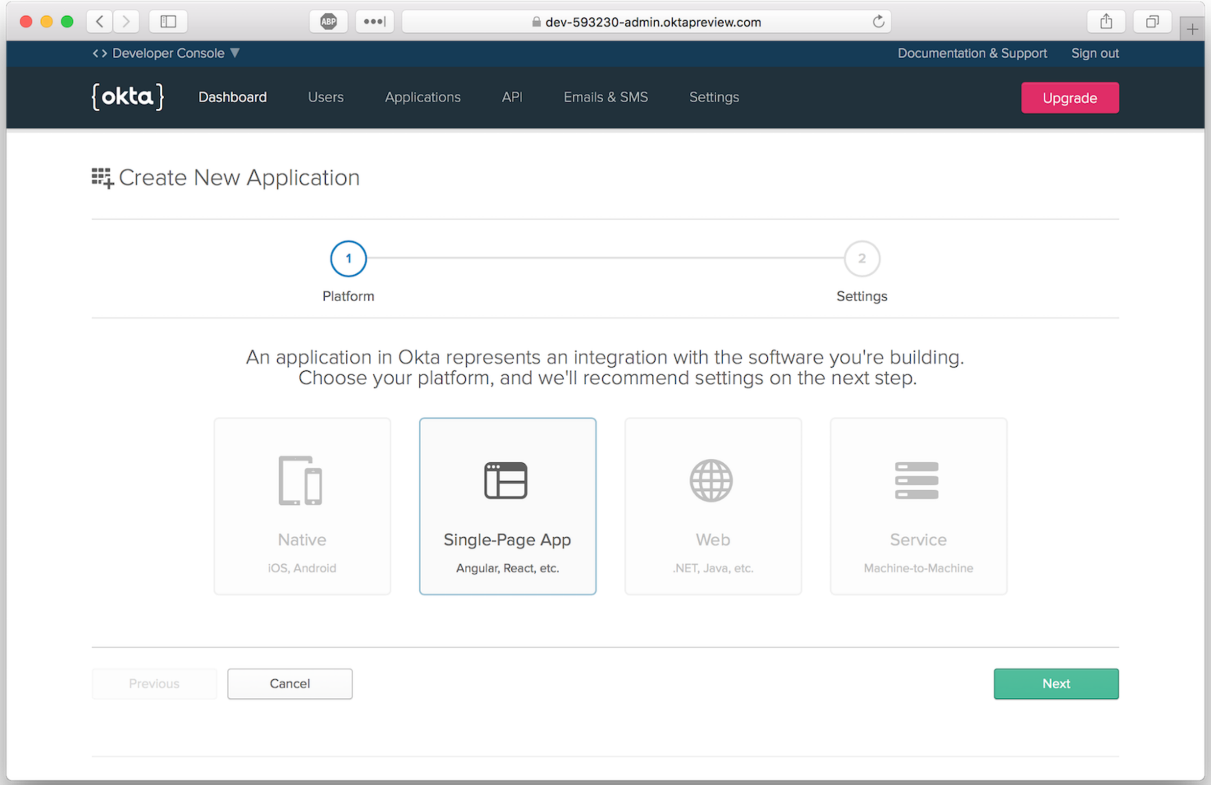
Once logged in, create a new application by clicking “Add Application”.



No active apps found

Add application and assign access to have them appear on your users' Okta home Page

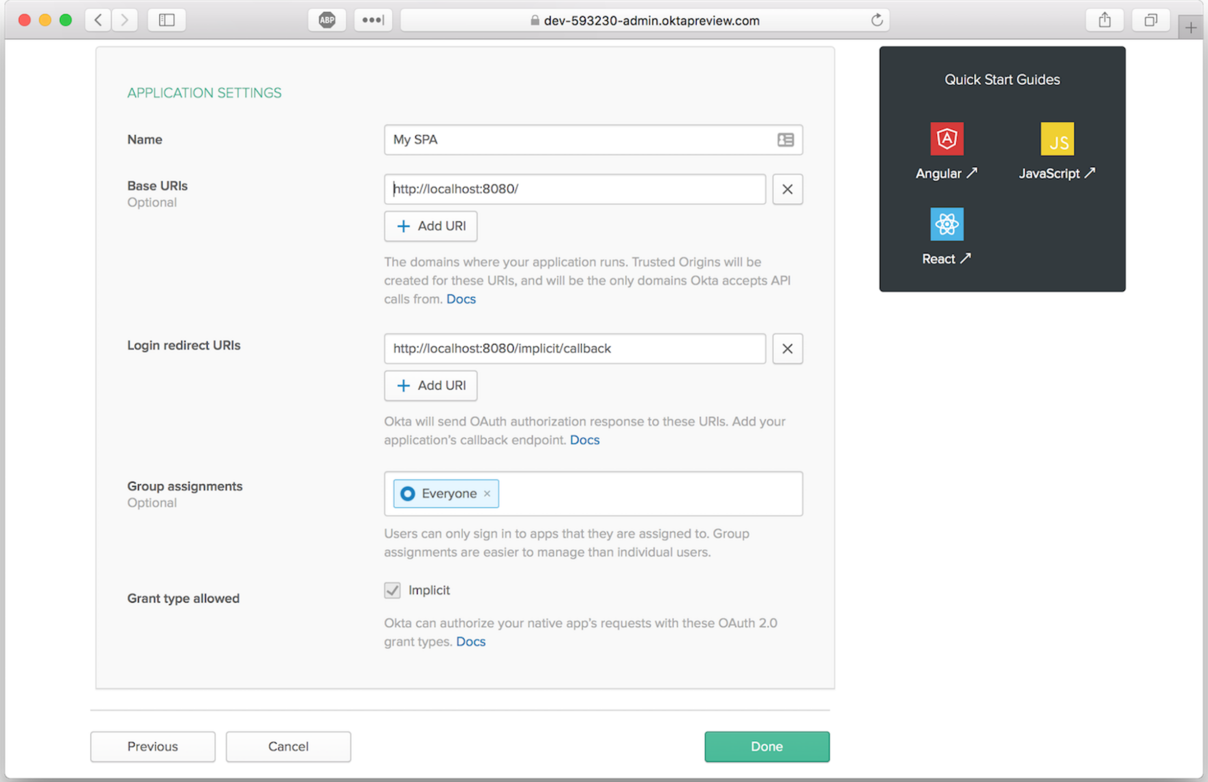
Select the “Single-Page App” platform option.



Create New Application

An application in Okta represents an integration with the software you're building. Choose your platform, and we'll recommend settings on the next step.

The default application settings should be the same as those pictured.



APPLICATION SETTINGS

Name: My SPA

Base URIs (Optional): http://localhost:8080/

The domains where your application runs. Trusted Origins will be created for these URIs, and will be the only domains Okta accepts API calls from. Docs

Login redirect URIs: http://localhost:8080/implicit/callback

Okta will send OAuth authorization response to these URIs. Add your application's callback endpoint. Docs

Group assignments (Optional): Everyone

Users can only sign in to apps that they are assigned to. Group assignments are easier to manage than individual users.

Grant type allowed: ☑ Implicit

Okta can authorize your native app's requests with these OAuth 2.0 grant types. Docs

To install the Okta Vue SDK, run the following command:

```
npm i @okta/okta-vue@1.0.0
```

Open `./src/router/index.js` and replace the entire file with the following code.

```
import Vue from 'vue'
import Router from 'vue-router'
import Hello from '@/components/Hello'
import PostsManager from '@/components/PostsManager'
import Auth from '@okta/okta-vue'

Vue.use(Auth, {
  issuer: 'https://{yourOktaDomain}/oauth2/default',
  client_id: '{yourClientId}',
  redirect_uri: 'http://localhost:8080/implicit/callback',
  scope: 'openid profile email'
})

Vue.use(Router)

let router = new Router({
  mode: 'history',
  routes: [
    {
      path: '/',
      name: 'Hello',
      component: Hello
    },
    {
      path: '/implicit/callback',
      component: Auth.handleCallback()
    },
    {
      path: '/posts-manager',
      name: 'PostsManager',
      component: PostsManager,
      meta: {
        requiresAuth: true
      }
    }
  ]
})

router.beforeEach(Vue.prototype.$auth.authRedirectGuard())

export default router
```

You'll need to replace `{yourOktaDomain}` and `{yourClientId}` which can be found on your application overview page in the Okta Developer Console. This will inject an `authClient` object into your Vue instance which can be accessed by calling `this.$auth` anywhere inside your Vue instance.

```
Vue.use(Auth, {
  issuer: 'https://{yourOktaDomain}/oauth2/default',
  client_id: '{yourClientId}',
  redirect_uri: 'http://localhost:8080/implicit/callback',
  scope: 'openid profile email'
})
```

The final step of Okta's authentication flow is redirecting the user back to your app with the token values in the URL. The `Auth.handleCallback()` component included in the SDK handles the redirect and persists the tokens on the browser.

```
{
  path: '/implicit/callback',
  component: Auth.handleCallback()
}
```

You also need to lock down protected routes from being accessed by unauthenticated users. This is accomplished by implementing a navigation guard. As the name suggests, navigation guards are primarily used to guard navigations either by redirecting or canceling.

The SDK comes with the method `auth.authRedirectGuard()` that checks matched routes' metadata for the key `requiresAuth` and redirects the user to the authentication flow if they are not authenticated.

```
router.beforeEach(Vue.prototype.$auth.authRedirectGuard())
```

With this navigation guard installed, any route that has the following metadata will be protected.

```
meta: {
  requiresAuth: true
}
```

Customize Your App Layout in Vue

The web app's layout is located in a component `./src/App.vue`. You can use the router-view component to render the matched component for the given path.

For the main menu, you'll want to change the visibility of certain menu items based on the status of the `activeUser`:

- Not Authenticated: Show only *Login*
- Authenticated: Show only *Logout*

You can toggle the visibility of these menu items using the `v-if` directive in Vue.js that checks the existence of `activeUser` on the component. When the component is loaded (which calls `created()`) or when a route changes we want to refresh the `activeUser`.

Open `./src/App.vue` and copy/paste the following code.

```
<template>
  <div id="app">
    <b-navbar toggleable="md" type="dark" variant="dark">
      <b-navbar-toggle target="nav_collapse"></b-navbar-toggle>
      <b-navbar-brand to="/">My Vue App</b-navbar-brand>
      <b-collapse is-nav id="nav_collapse">
        <b-navbar-nav>
          <b-nav-item to="/">Home</b-nav-item>
          <b-nav-item to="/posts-manager">Posts Manager</b-nav-item>
          <b-nav-item href="#" @click.prevent="login" v-if="!activeUser">Login</b-nav-item>
          <b-nav-item href="#" @click.prevent="logout" v-else>Logout</b-nav-item>
        </b-navbar-nav>
      </b-collapse>
    </b-navbar>
    <!-- routes will be rendered here -->
    <router-view />
  </div>
</template>

<script>

export default {
  name: 'app',
  data () {
    return {
      activeUser: null
    }
  },
  async created () {
    await this.refreshActiveUser()
  },
  watch: {
    // everytime a route is changed refresh the activeUser
    '$route': 'refreshActiveUser'
  },
  methods: {
    login () {
      this.$auth.loginRedirect()
    },
    async refreshActiveUser () {
      this.activeUser = await this.$auth.getUser()
    },
    async logout () {
      await this.$auth.logout()
      await this.refreshActiveUser()
      this.$router.push('/')
    }
  }
}
</script>
```

Every login must have a logout. The following snippet will logout your user, refresh the active user (which is now null), and then redirect the user to the homepage. This method is called when a user clicks on the logout link in the nav.

```
async logout () {
  await this.$auth.logout()
  await this.refreshActiveUser()
  this.$router.push('/')
}
```

Components are the building blocks within Vue.js. Each of your pages will be defined in the app as a component. Since the vue-cli webpack template utilizes vue-loader, your component source files have a convention that separates template, script, and style (see here).

Now that you've added vue-bootstrap, modify `./src/components/Hello.vue` to remove the boilerplate links vue-cli generates.

```
<template>
  <div class="hero">
    <div>
      <h1 class="display-3">Hello World</h1>
      <p class="lead">This is the homepage of your vue app</p>
    </div>
  </div>
</template>

<style>
  .hero {
    height: 90vh;
    display: flex;
    align-items: center;
    justify-content: center;
    text-align: center;
  }
  .hero .lead {
    font-weight: 200;
    font-size: 1.5rem;
  }
</style>
```

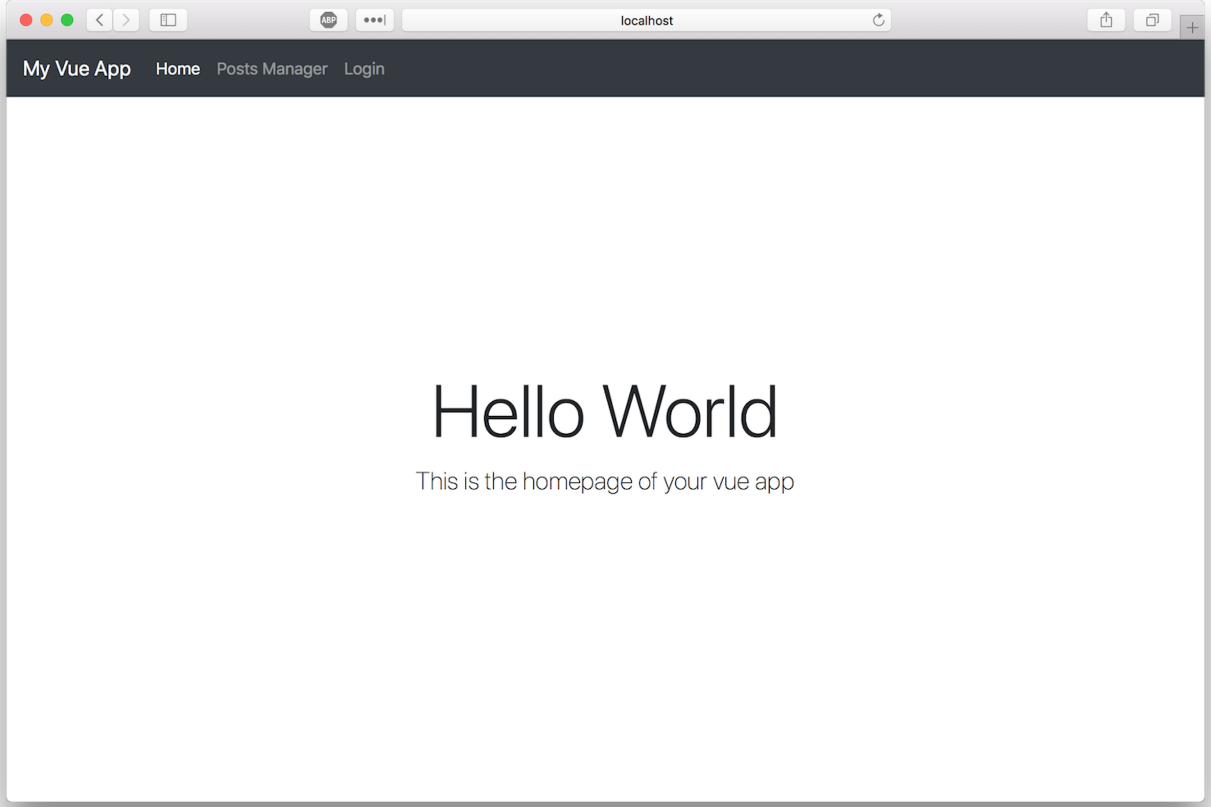
At this point you can stub out the Post Manager page to test your authentication flow. Once you confirm authentication works, you'll start to build out the API calls and components required to perform CRUD operations on your Posts model.

Create a new file `./src/components/PostsManager.vue` and paste the following code:

```
<template>
  <div class="container-fluid mt-4">
    <h1 class="h1">Posts Manager</h1>
    <p>Only authenticated users should see this page</p>
  </div>
</template>
```

Take Your Vue.js Frontend and Auth Flows for a Test Drive

In your terminal run `npm run dev` (if it's not already running). Navigate to `http://localhost:8080` and you should see the new homepage.

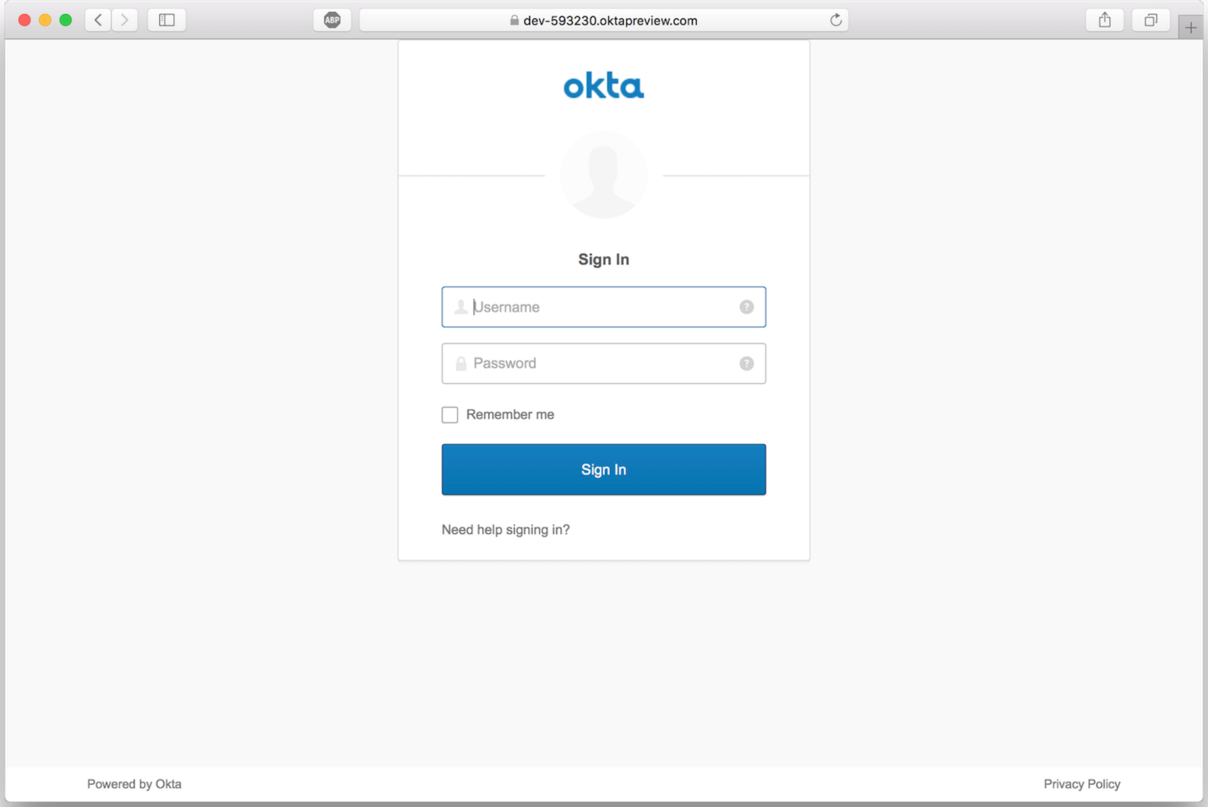


Hello World

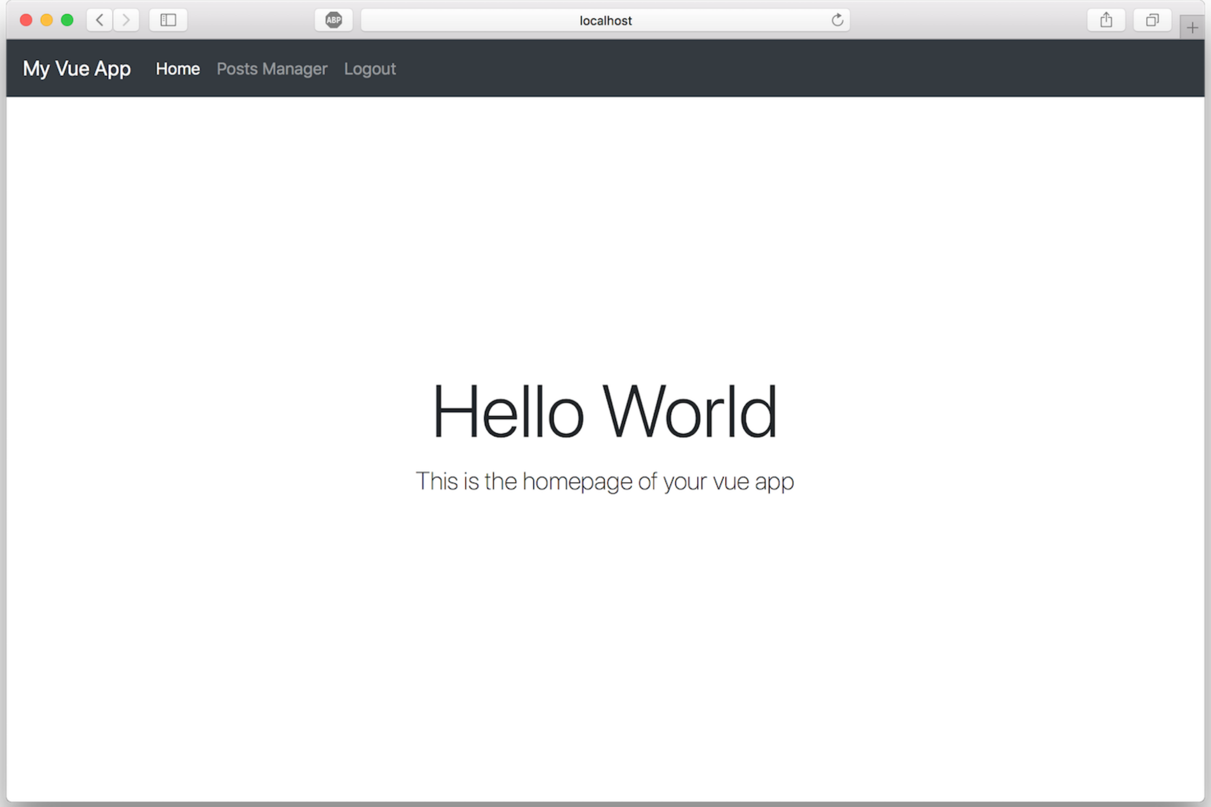
This is the homepage of your vue app

If you click **Posts Manager** or **Login** you should be directed to Okta's flow. Enter your Okta dev account credentials.

NOTE: If you are logged in to your Okta Developer Account you will be redirected automatically back to the app. You can test this by using incognito or private browsing mode.



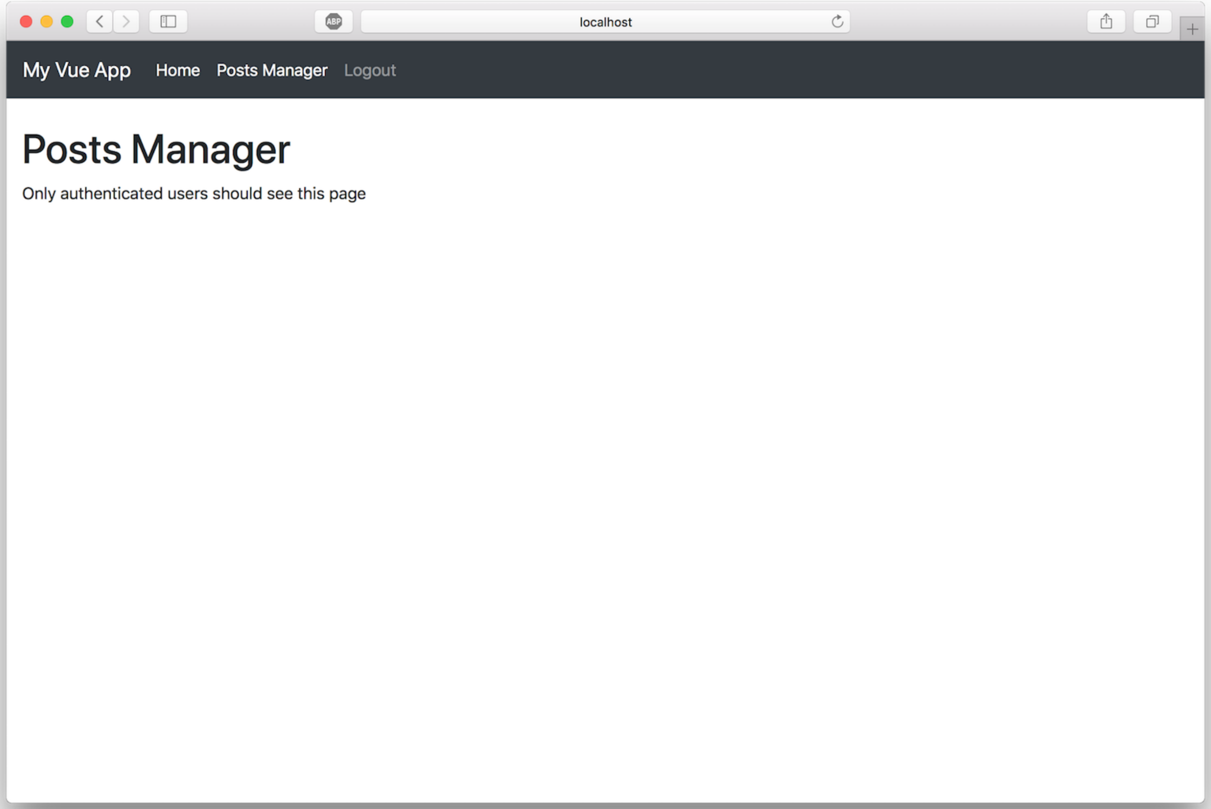
If successful, you should return to the homepage logged in.



Hello World

This is the homepage of your vue app

Clicking on **Posts Manager** link should render the protected component.



Posts Manager

Only authenticated users should see this page

Add a Backend REST API Server

Now that users can securely authenticate, you can build the REST API server to perform CRUD operations on a post model. Add the following dependencies to your project:

```
npm i express@4.16.3 cors@2.8.4 @okta/jwt-verifier@0.0.11 sequelize@4.37.6 sqlite3@4.0.0 epilogue@0.7.1 axios@0.18.0
```

Then, create the file `./src/server.js` and paste the following code.

```
const express = require('express')
const cors = require('cors')
const bodyParser = require('body-parser')
const Sequelize = require('sequelize')
const epilogue = require('epilogue')
const OktaJwtVerifier = require('@okta/jwt-verifier')

const oktaJwtVerifier = new OktaJwtVerifier({
  clientId: '{yourClientId}',
  issuer: 'https://{yourOktaDomain}/oauth2/default'
})

let app = express()
app.use(cors())
app.use(cors())
app.use(bodyParser.json())

// verify JWT token middleware
app.use((req, res, next) => {
  // require every request to have an authorization header
  if (!req.headers.authorization) {
    return next(new Error('Authorization header is required'))
  }
  let parts = req.headers.authorization.trim().split(' ')
  let accessToken = parts.pop()
  oktaJwtVerifier.verifyAccessToken(accessToken)
    .then(jwt => {
      req.user = {
        uid: jwt.claims.uid,
        email: jwt.claims.sub
      }
      next()
    })
    .catch(next) // jwt did not verify!
})

// For ease of this tutorial, we are going to use SQLite to limit dependencies
let database = new Sequelize({
  dialect: 'sqlite',
  storage: './test.sqlite'
})

// Define our Post model
// id, createdAt, and updatedAt are added by sequelize automatically
let Post = database.define('posts', {
  title: Sequelize.STRING,
  body: Sequelize.TEXT
})

// Initialize epilogue
epilogue.initialize({
  app: app,
  sequelize: database
})

// Create the dynamic REST resource for our Post model
let userResource = epilogue.resource({
  model: Post,
  endpoints: ['/posts', '/posts/:id']
})

// Resets the database and launches the express app on :8081
database
  .sync({ force: true })
  .then(() => {
    app.listen(8081, () => {
      console.log('listening to port localhost:8081')
    })
  })
```

Make sure to replace the variables `{yourOktaDomain}` and `{clientId}` in the above code with values from your OIDC app in Okta.

Add Sequelize

Sequelize is a promise-based ORM for Node.js. It supports the dialects PostgreSQL, MySQL, SQLite, and MSSQL and features solid transaction support, relations, read replication, and more.

For ease of this tutorial, you're going to use SQLite to limit external dependencies. The following code initializes a Sequelize instance using SQLite as your driver.

```
let database = new Sequelize({
  dialect: 'sqlite',
  storage: './test.sqlite'
})
```

Each post has a `title` and `body` . (The fields `createdAt` , and `updatedAt` are added by Sequelize automatically). With Sequelize, you define models by calling `define()` on your instance.

```
let Post = database.define('posts', {
  title: Sequelize.STRING,
  body: Sequelize.TEXT
})
```

Add Epilogue

Epilogue creates flexible REST endpoints from Sequelize models within an Express app. If you ever coded REST endpoints you know how much repetition there is. D.R.Y. FTW!

```
// Initialize epilogue
epilogue.initialize({
  app: app,
  sequelize: database
})

// Create the dynamic REST resource for our Post model
let userResource = epilogue.resource({
  model: Post,
  endpoints: ['/posts', '/posts/:id']
})
```

Verify Your JWT

This is the most crucial component of your REST API server. Without this middleware any user can perform CRUD operations on our database. If no authorization header is present, or the access token is invalid, the API call will fail and return an error.

```
// verify JWT token middleware
app.use((req, res, next) => {
  // require every request to have an authorization header
  if (!req.headers.authorization) {
    return next(new Error('Authorization header is required'))
  }
  let parts = req.headers.authorization.trim().split(' ')
  let accessToken = parts.pop()
  oktaJwtVerifier.verifyAccessToken(accessToken)
    .then(jwt => {
      req.user = {
        uid: jwt.claims.uid,
        email: jwt.claims.sub
      }
      next()
    })
    .catch(next) // jwt did not verify!
})
```

Run the Server

Open a new terminal window and run the server with the command `node ./src/server` . You should see debug information from Sequelize and the app listening on port 8081.

Complete the Posts Manager Component

Now that the REST API server is complete, you can start wiring up your posts manager to fetch posts, create posts, edit posts, and delete posts.

I always centralize my API integrations into a single helper module. This keeps the code in components much cleaner and provides single location in case you need to change anything with the API request.

Create a file `./src/api.js` and copy/paste the following code into it:

```
import Vue from 'vue'
import axios from 'axios'

const client = axios.create({
  baseURL: 'http://localhost:8081/',
  json: true
})

export default {
  async execute (method, resource, data) {
    // inject the accessToken for each request
    let accessToken = await Vue.prototype.$auth.getAccessToken()
    return client({
      method,
      url: resource,
      data,
      headers: {
        Authorization: `Bearer ${accessToken}`
      }
    }).then(req => {
      return req.data
    })
  },
  getPosts () {
    return this.execute('get', '/posts')
  },
  getPost (id) {
    return this.execute('get', `/posts/${id}`)
  },
  createPost (data) {
    return this.execute('post', '/posts', data)
  },
  updatePost (id, data) {
    return this.execute('put', `/posts/${id}`, data)
  },
  deletePost (id) {
    return this.execute('delete', `/posts/${id}`)
  }
}
```

When you authenticate with OIDC, an access token is persisted locally in the browser. Since each API request must have an access token, you can fetch it from the authentication client and set it in the request.

```
let accessToken = await Vue.prototype.$auth.getAccessToken()
return client({
  method,
  url: resource,
  data,
  headers: {
    Authorization: `Bearer ${accessToken}`
  }
})
```

By creating the following proxy methods inside your API helper, the code outside the helper module remains clean and semantic.

```
getPosts () {
  return this.execute('get', '/posts')
},
getPost (id) {
  return this.execute('get', `/posts/${id}`)
},
createPost (data) {
  return this.execute('post', '/posts', data)
},
updatePost (id, data) {
  return this.execute('put', `/posts/${id}`, data)
},
deletePost (id) {
  return this.execute('delete', `/posts/${id}`)
}
```

You now have all the components required to wire up your posts manager component to make CRUD operations via the REST API. Open `./src/components/PostsManager.vue` and copy/paste the following code.

```
<template>
  <div class="container-fluid mt-4">
    <h1 class="h1">Posts Manager</h1>
    <b-alert :show="loading" variant="info">Loading...</b-alert>
    <b-row>
      <b-col>
        <table class="table table-striped">
          <thead>
            <tr>
              <th>ID</th>
              <th>Title</th>
              <th>Updated At</th>
              <th>&nbsp;</th>
            </tr>
          </thead>
          <tbody>
            <tr v-for="post in posts" :key="post.id">
              <td>{{ post.id }}</td>
              <td>{{ post.title }}</td>
              <td>{{ post.updatedAt }}</td>
              <td class="text-right">
                <a href="#" @click.prevent="populatePostToEdit(post)">Edit</a> -
                <a href="#" @click.prevent="deletePost(post.id)">Delete</a>
              </td>
            </tr>
          </tbody>
        </table>
      </b-col>
      <b-col lg="3">
        <b-card :title="(model.id ? 'Edit Post ID#' + model.id : 'New Post')">
          <form @submit.prevent="savePost">
            <b-form-group label="Title">
              <b-form-input type="text" v-model="model.title"></b-form-input>
            </b-form-group>
            <b-form-group label="Body">
              <b-form-textarea rows="4" v-model="model.body"></b-form-textarea>
            </b-form-group>
            <div>
              <b-btn type="submit" variant="success">Save Post</b-btn>
            </div>
          </form>
        </b-card>
      </b-col>
    </b-row>
    </b-row>
  </div>
</template>

<script>
import api from '@/api'
export default {
  data () {
    return {
      loading: false,
      posts: [],
      model: {}
    }
  },
  async created () {
    this.refreshPosts()
  },
  methods: {
    async refreshPosts () {
      this.loading = true
      this.posts = await api.getPosts()
      this.loading = false
    },
    async populatePostToEdit (post) {
      this.model = Object.assign({}, post)
    },
    async savePost () {
      if (this.model.id) {
        await api.updatePost(this.model.id, this.model)
      } else {
        await api.createPost(this.model)
      }
      this.model = {} // reset form
      await this.refreshPosts()
    },
    async deletePost (id) {
      if (confirm('Are you sure you want to delete this post?')) {
        // if we are editing a post we deleted, remove it from the form
        if (this.model.id === id) {
          this.model = {}
        }
        await api.deletePost(id)
        await this.refreshPosts()
      }
    }
  }
}
</script>
```

Listing Posts

You'll use `api.getPosts()` to fetch posts from your REST API server. You should refresh the list of posts when the component is loaded and after any mutating operation (create, update, or delete).

```
async refreshPosts () {
  this.loading = true
  this.posts = await api.getPosts()
  this.loading = false
}
```

The attribute `this.loading` is toggled so the UI can reflect the pending API call. You might not see the loading message since the API request is not going out to the internet.

Creating Posts

A form is included in the component to save a post. It's wired up to call `savePosts()` when the form is submitted and its inputs are bound to the `model` object on the component.

When `savePost()` is called, it will perform either an update or create based on the existence of `model.id` . This is mostly a shortcut to not have to define two separate forms for creating and updating.

```
async savePost () {
  if (this.model.id) {
    await api.updatePost(this.model.id, this.model)
  } else {
    await api.createPost(this.model)
  }
  this.model = {} // reset form
  await this.refreshPosts()
}
```

Updating Posts

When updating a post, you first must load the post into the form. This sets `model.id` which will the trigger an update in `savePost()` .

```
async populatePostToEdit (post) {
  this.model = Object.assign({}, post)
}
```

Important: The `Object.assign()` call copies the value of the post argument rather than the reference. When dealing with mutation of objects in Vue, you should always set to the value, not reference.

Deleting Posts

To delete a post simply call `api.deletePost(id)` . It's always good to confirm before delete so let's throw in a native confirmation alert box to make sure the click was intentional.

```
async deletePost (id) {
  if (confirm('Are you sure you want to delete this post?')) {
    await api.deletePost(id)
    await this.refreshPosts()
  }
}
```

Test Your Vue.js + Node CRUD App

Make sure both the server and frontend are running.

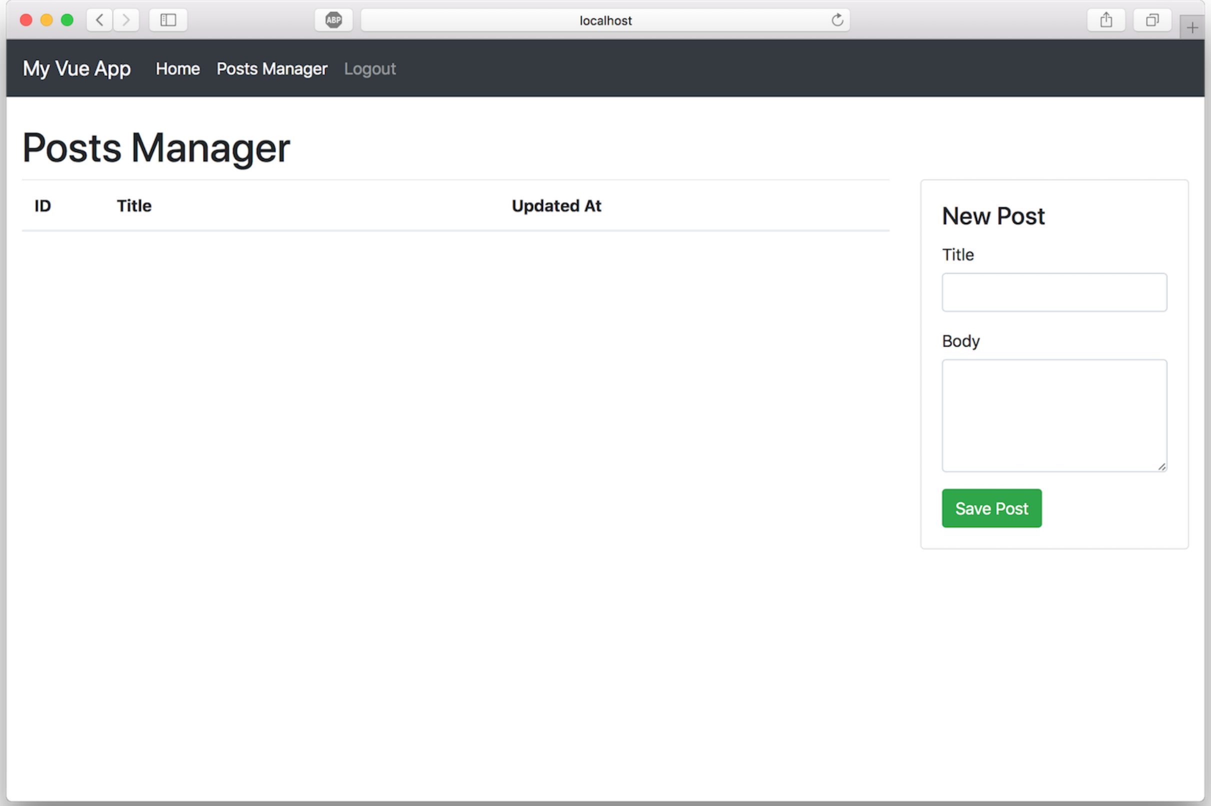
Terminal #1

```
node ./src/server
```

Terminal #2

```
npm run dev
```

Navigate to `http://localhost:8080` and give it a whirl.



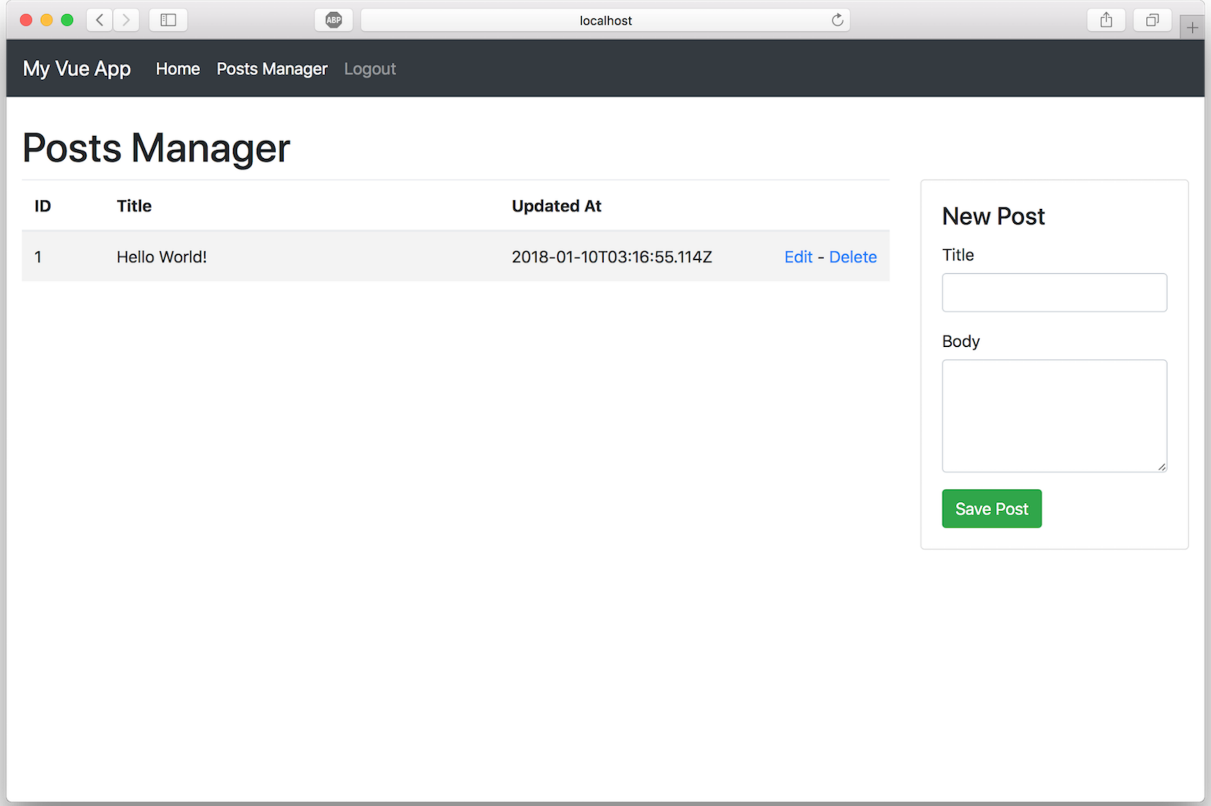
Posts Manager

ID	Title	Updated At

New Post

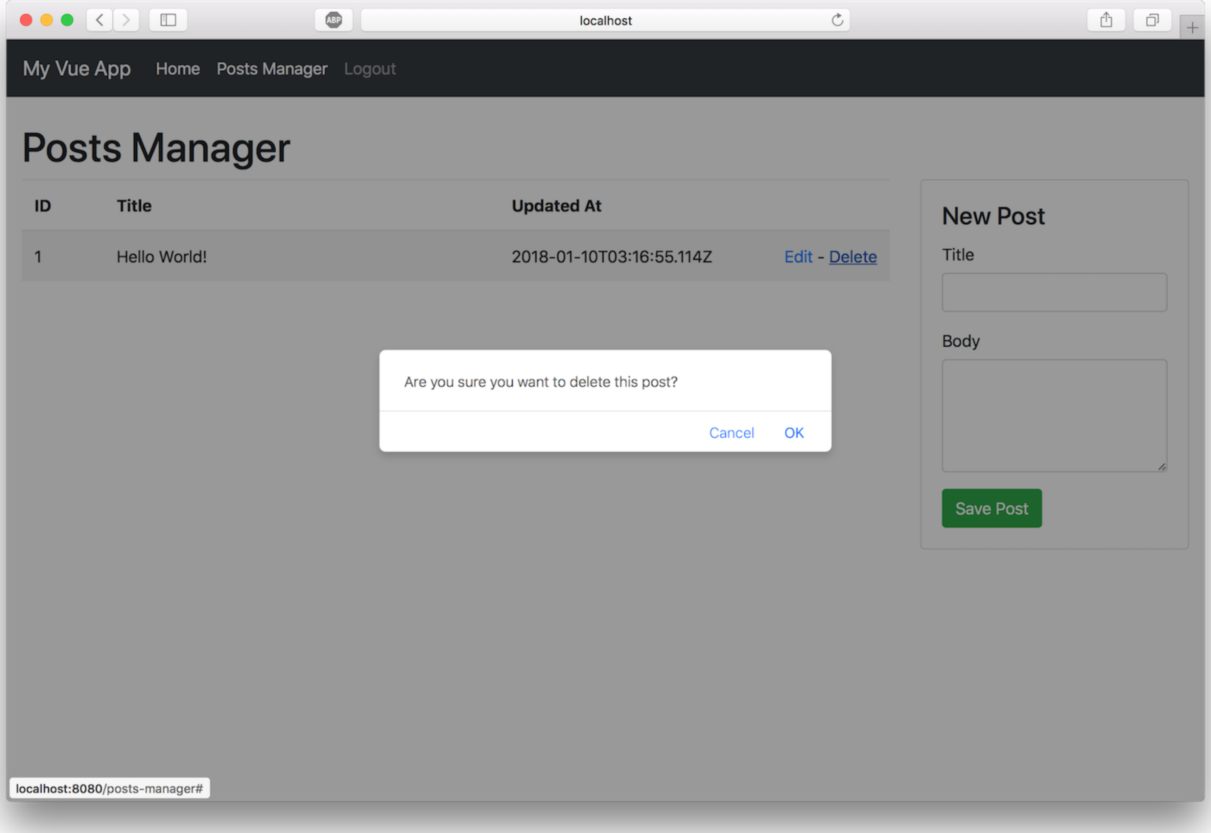
Title

Body



Posts Manager

ID	Title	Updated At	
1	Hello World!	2018-01-10T03:16:55.114Z	Edit - Delete



Posts Manager

ID	Title	Updated At	
1	Hello World!	2018-01-10T03:16:55.114Z	Edit - Delete

Are you sure you want to delete this post?

Do More With Vue!

As I said at the top of this post, I think Vue stands head and shoulders above other frameworks. Here are five quick reasons why:

- Simple component lifecycle
- HTML-based templating and native two-way binding
- Widely agreed upon ways to handle routing, state management, webpack configuration, and isomorphic web apps
- Massive community supported resources, components, libraries, and projects
- Vue feels very similar to React (without the JSX!) which lowers the barrier to entry for those with React experience. Moving between React and Vue isn't very difficult.

I covered a lot of material in this tutorial but don't feel bad if you didn't grasp everything the first time. The more you work with these technologies, the more familiar they will become.

To learn more about Vue.js head over to https://vuejs.org or check out these other great resources from the @oktadev team:

- The Ultimate Guide to Progressive Web Applications
- The Lazy Developer's Guide to Authentication with Vue.js
- Build a Cryptocurrency Comparison Site with Vue.js

You can find the source code for the application developed in this post at https://github.com/oktadeveloper/okta-vue-node-example.

Hit me up in the comments with any questions, and as always, follow @oktadev on Twitter to see all the cool content our dev team is creating.

- Apr 16, 2018: Updated to use the latest dependencies, including Okta's Vue SDK 1.0.0. See the code changes in oktadeveloper/okta-vue-node-example-example#2. Changes to this article can be viewed in okta/okta.github.io#1959.
- Mar 12, 2018: Updated to use the latest dependencies, including Bootstrap 4.0.0. See the code changes in oktadeveloper/okta-vue-node-example-example#1. Changes to this article can be viewed in okta/okta.github.io#1837.

28 Comments **Okta Developer Blog** 1 **Login**

♡ **Recommend** 5 **Share** **Sort by Best**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Allyn Becar • a month ago

Good info in this tutorial.
However I get the following error form any REST request. (I walked through the tutorial, then downloaded the code and in both cases get the same error.)

Failed to load http://localhost:8081/posts: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://localhost:8080' is therefore not allowed access. The response had HTTP status code 404.

⌃ | ⌄ • Reply • Share ›

Matt Raible ➦ Allyn Becar • a month ago

Hello Allyn,

The following line should allow anyone to call the server, from any origin:

```
app.use(cors())
```

Are you sure your server is running on 8081 and it's not a different process?

⌃ | ⌄ • Reply • Share ›

Allyn Becar ➦ Matt Raible • a month ago

Thanks for your help. Sorry it was my newbie error. 8081 was in use by a different process. Works like a champ now.

⌃ | ⌄ • Reply • Share ›

Jone Quest • a month ago

Hello, nice tutorial, but I'm getting the errors when trying to run the following:
npm i express@4.16.3 cors@2.8.4 @okta/jwt-verifier@0.0.11 sequelize@4.37.6 sqlite3@4.0.0 epilogue@0.7.1 axios@0.18.0

node-pre-gyp ERR! Tried to download(403): https://mapbox-node-binary....
node-pre-gyp ERR! Pre-built binaries not found for sqlite3@4.0.0 and node@10.4.1 (node-v64 ABI, glibc) (falling back to source compile with node-gyp)
gyp ERR! configure error
gyp ERR! stack Error: EACCES: permission denied, mkdir '/var/www/html/vuejs/my-vue-app/node_modules/sqlite3/build'
gyp ERR! System Linux 4.15.0-23-generic
gyp ERR! command "/usr/local/bin/node" "/usr/local/lib/node_modules/npm/node_modules/node-gyp/bin/node-gyp.js" "configure" "--fallback-to-build" "--module=/var/www/html/vuejs/my-vue-app/node_modules/sqlite3/lib/binding/node-v64-linux-x64/node_sqlite3.node" "--module_name=node_sqlite3" "--module_path=/var/www/html/vuejs/my-vue-app/node_modules/sqlite3/lib/binding/node-v64-linux-x64" "--napi_version=3" "--node_abi_napi=napi"
gyp ERR! cwd /var/www/html/vuejs/my-vue-app/node_modules/sqlite3
gyp ERR! node -v v10.4.1

see more

⌃ | ⌄ • Reply • Share ›

Matt Raible ➦ Jone Quest • a month ago

Can you try it with a Node LTS version? For example, v8.11.3.

⌃ | ⌄ • Reply • Share ›

Jone Quest ➦ Matt Raible • a month ago

Hey Matt, thanks for you help! I actually installed them separated and it worked! As for the Otka/JWT login authentication, I think that in a real business they don't want/allow that type of authentication. Do you know any other auth method? Thanks again.

⌃ | ⌄ • Reply • Share ›

Matt Raible ➦ Jone Quest • a month ago

I'm not sure what your requirements are. We have lots of real businesses using Okta for authentication. It's probably best to continue this discussion on our Developer Forums. Can you please post your questions there?

^ | ˅ • Reply • Share ›

Jone Quest ➦ Matt Raible • a month ago

Hi Matt, I mean, user's need to have an Okta account to be able to login in the app? Normally, companies will not agree with that! Instead, they will prefer a SSO (single sign on).

^ | ˅ • Reply • Share ›

Matt Raible ➦ Jone Quest • a month ago

Hi Jone: the good news is Okta is a popular SSO provider used by thousands of organizations around the world! You can read more at https://www.okta.com/produc.... If you're using our API Products, you can add social login too.

If that's not enough, I wrote a post recently that shows you how to enable self-registration.

^ | ˅ • Reply • Share ›

Jone Quest ➦ Matt Raible • a month ago

Thanks a lot Matt. I'll look at the links you shared with me.

^ | ˅ • Reply • Share ›

David Maksimov • a month ago

Great tutorial! Thanks for this

^ | ˅ • Reply • Share ›

ADawg • 3 months ago

Hi,
Great tutorial. I've followed this and it's working all the way up until the API tries to verify an accessToken. The console log for the node API tells me "JwtParseError: Error while resolving signing key for kid ...". Then investigating deeper, I found that my node API cannot reach my okta subdomain (or the subdomain's response is rejected) because I am behind a corporate proxy. How can I set up the requests sent by the oktaJwtVerifier to use my proxy?

^ | ˅ • Reply • Share ›

ADawg ➦ ADawg • 3 months ago

+Matt, When I look at the @okta/jwt-verifier library on npm, it has 7 dev dependencies. One of those is 'request', which is a library that makes http requests. That library has ways to configure a request to use a proxy. Since the jwt-verifier package is (presumably) using 'request' to send and receive, what I really need is a way to configure the oktaJwtVerifier (in the server.js file) so that it passes the correct proxy setup to it's instance of request.

See this
https://www.npmjs.com/packa...

and this
https://www.npmjs.com/packa...

^ | ˅ • Reply • Share ›

Robert Damphousse ➦ ADawg • 3 months ago

Hi **@ADawg** , you're seeing request on that list because it's a dev dependency (it's part of our tests, but not the actual library). The actual library delegates downward to node-openid-connect, which has these notes on configuring their request library:

https://github.com/panva/no...

I haven't tried configuring the proxy, but here is an example of configuring other aspects:

https://github.com/okta/okt...

Hope this helps!

⌃ | ⌄ • Reply • Share ›

ADawg ➦ Robert Damphousse • 3 months ago

Thank you Robert!

⌃ | ⌄ • Reply • Share ›

Alicja Franckowska • 4 months ago

Hi,
we need to keep multiple list of users that are using our app. The problem is our software is used by multiple companies that has their own users list (not customers but employees).
Is it possible to handle not one large list of users but multiple lists? So we could describe which list are for which customer?

⌃ | ⌄ • Reply • Share ›

Matt Raible ➦ Alicja Franckowska • 4 months ago

Can you put your different users into different groups and then assign groups to your apps on Okta? If that doesn't help, can you can please explain more about your issue in question on our Developer Forums?

⌃ | ⌄ • Reply • Share ›

Minjung Shin • 4 months ago

Hi!
Thanks for a great tutorial. I cloned your code and set the issuer information in src/server.js and src/router/index.js.
It works great when I run it using "npm run dev".
However, when I do a production build, it doesn't work. For example,

```
$ npm run build
$ cd dist
$ http-server(simple http server)
```

I get "page not found error" on redirect_uri.
How can I make it work from production build?
Thanks!!

⌃ | ⌄ • Reply • Share ›

Matt Raible ➦ Minjung Shin • 4 months ago

Are you using the same port and URL? I'd compare the port used by `npm run dev` versus the port used by `http-server`. If they're different, you can add a Login redirect URI to your app on Okta, or change the ports to be the same.

⌃ | ⌄ • Reply • Share ›

João José • 4 months ago

Can I use okta / okta-vue with identityserver4?

```
Vue.use(Auth, {
issuer: 'https://localhost:44392/',//My IdentityPovider
client_id: 'client',//
redirect_uri: 'http://localhost:8080/implicit/callback',
scope: 'openid profile address offline_access'
})
```

⌃ | ⌄ • Reply • Share ›

Matt Raible ➦ João José • 4 months ago

Hello João,

We're a certified OIDC provider and stick as close to the specifications as possible - in addition to addressing things like the OAuth 2.0 Threat Model. If IdentityServer4 follows the standards and is configured properly, our libraries *might* work with it.

⌃ | ⌄ • Reply • Share ›

Andrey V Maltsev • 4 months ago

Thanks a lot for this tutorial!

⌃ | ⌄ • Reply • Share ›

bubs • 4 months ago

very very nice tutorial. explains all the parts very well and how to protect your api via a JWT. thanks.

⌃ | ⌄ • Reply • Share ›

Pavel Pletmintsev • 6 months ago

Thanks for tutorial!
Got "Unable to parse a token from the url" when reload page and "The JWT was issued in the future" when click 'login' or 'Posts manager'. What's the problem?

⌃ | ⌄ • Reply • Share ›

Matt Raible ➦ Pavel Pletmintsev • 6 months ago

Do you get this error if you clone the example from GitHub and configure it to use your Okta application settings?

1 ⌃ | ⌄ • Reply • Share ›

Pavel Pletmintsev ➦ Matt Raible • 6 months ago

Yes, i tried both following the tutorial and using repo code.

⌃ | ⌄ • Reply • Share ›

Matt Raible ➦ Pavel Pletmintsev • 6 months ago

This is likely a clock skew issue. I'd suggest making sure your computer is synced to a network clock. For example, here's a guide for Windows. Also, make sure your NTP ports aren't blocked.

1 ⌃ | ⌄ • Reply • Share ›

Avatar This comment was deleted.

Matt Raible ➦ Guest • 6 months ago

We do token validation in the browser, and one of those checks is the expiration and issued at times. For the exact code that does this validation, see okta/okta-auth-js/.../clientBuilder.js#L59-L70.

⌃ | ⌄ • Reply • Share ›

ALSO ON **OKTA DEVELOPER BLOG**

What is the OAuth 2.0 Implicit Grant Type?

3 comments • 3 months ago

Joshua Chamberlain — This post says, "However, the Okta Authorization Code grant requires the client secret, so we've taken a different approach noted below."What is

Tutorial: Build a Basic CRUD App with Node.js

10 comments • 2 months ago

Matt Raible — Thanks for the heads up Pedro. You're right that a comma was missing. The following was missing from the bottom of models.js too, which causes

VISIT OKTA.COM

Social

GITHUB TWITTER FORUM RSS BLOG YOUTUBE

More Info

INTEGRATE WITH OKTA
BLOG
CHANGE LOG
3RD PARTY NOTICES

Contact & Legal

CONTACT SALES
CONTACT SUPPORT
TERMS & CONDITIONS
PRIVACY POLICY