NUXT

# Requerimientos

## Node JS y NPM con NVM

$ git clone https://github.com/creationix/nvm.git ~/.nvm && cd ~/.nvm && git checkout `git describe --abbrev=0 --tags`  
$ echo "source ~/.nvm/nvm.sh" >> ~/.profile  
$ source ~/.profile  
$ nvm install 8.11.4  
$ nvm alias default 8.11.4  
$ node -v  
$ npm -v

## NPX y CREATE-NUXT-APP

npm install -g npx

npx create-nuxt-app <my-project>

# Configuración

## Host y Puerto

1. /package.json

"config": {  
 "nuxt": {  
 "host": "0.0.0.0",  
 "port": "3000"  
 }  
}

# Comandos

1. Forzar cambios en los componentes .vue

find -name "\*.vue" -exec touch {} +

1. Reinstalar módulos

rm -rf ../node\_modules/\* ../node\_modules/.\*

rm -f package-lock.json

npm cache clean --force

npm i

# Estructura

## assets

## components

## layouts

## middleware

## pages

## plugins

## static

## store

# Router

## Nuxt-link

Enlazar con secciones de la app mediante nuxt-link en vez de router-link

<nuxt-link to="/posts"></nuxt-link>

# Validate

# Pages

# Layouts

Dentro del directorio layout podemos declarar distintos layouts para nuestra aplicación. Por defecto siempre utiliza **default.vue**. Podemos declarar un layout para errores que se tiene que llamar **error.vue.**

1. /layouts
   * /default.vue
   * /error.vue
   * /other.vue

Para que una página utilice un layout distinto a default.vue lo indicamos mediante la etiqueta **layout:**

**export default** {  
 layout: 'other'  
}

# Assets

Se pueden incluir imagines, estilos CSS, ficheros JS, etc… Para acceder a los recursos la ruta se resuelve como **~assets**

1. /assets/images/background-image.jpg

background-image: url('~assets/images/main-background.jpg');

# AsyncData

Para optimizar SEO y que SSR devuelva un HTML con datos precargados, se utiliza **asyncData.** Los datos contenidos en asyncData serán renderizados por el servidor antes de devolver HTML al cliente.

Es importante tener en cuenta que:

1. asyncData sólo puede ser utilizado dentro de **/pages**.
2. No es posible utilizar **this** ya que el método asyncData es llamado antes de iniciar el componente.
3. Sólo se ejecuta en el servidor si se realiza una petición. Si por el contrario se navega mediante nuxt-link, asyncData se ejecuta en el cliente.
4. En caso de que exista data() une ambos resultados.

El método asyncData tiene 2 parámetros:

1. Context: contexto de la aplicación.
   1. Contiene elementos de la ruta actual :

title: 'First (ID: ' + context.params.id + ')',

* 1. Etc…

1. Callback: función con la que cargamos los datos y devuelve el error o el contenido.

## Ejemplo con callback (error, results)

asyncData (context, callback) {  
 setTimeout(() => {  
 callback(**null**, {  
 loadedPosts: [  
 {  
 id: '1',  
 title: 'First',  
 thumbnail: 'https://loscomentarios.com/files/hola/imagenes\_de\_hola-4389.gif',  
 previewText: 'Hola'  
 },  
 {  
 id: '2',  
 title: 'Second',  
 thumbnail: 'https://loscomentarios.com/files/hola/imagenes\_de\_hola-4389.gif',  
 previewText: 'Adios'  
 },  
 {  
 id: '3',  
 title: 'Third',  
 thumbnail: 'https://loscomentarios.com/files/hola/imagenes\_de\_hola-4389.gif',  
 previewText: 'Tralará'  
 }  
 ]  
 })  
 }, 1500  
 )  
}

## Ejemplo con Promise

1. asyncData sin callback para evitar que NUXT se quede esperando la respuesta.
2. Ejecutamos resolve en caso de éxito
3. Reject en caso de error

asyncData (context) {  
 **return new** Promise((resolve, reject) => {  
 // success case  
 setTimeout(() => {  
 resolve({  
 loadedPosts: [  
 {  
 id: '1',  
 title: 'First',  
 thumbnail: 'https://loscomentarios.com/files/hola/imagenes\_de\_hola-4389.gif',  
 previewText: 'Hola'  
 },  
 {  
 id: '2',  
 title: 'Second',  
 thumbnail: 'https://loscomentarios.com/files/hola/imagenes\_de\_hola-4389.gif',  
 previewText: 'Adios'  
 },  
 {  
 id: '3',  
 title: 'Third',  
 thumbnail: 'https://loscomentarios.com/files/hola/imagenes\_de\_hola-4389.gif',  
 previewText: 'Tralará'  
 }  
 ]  
 })  
 }, 1500)  
 // error case  
 // reject(new Error())  
 })  
 .then(data => {  
 **return** data  
 })  
 .catch(e => {  
 context.error(**new** Error())  
 })  
}

# Store

Para centralizar los datos de nuestra aplicación podemos utilizar VUEX. Debemos crear un fichero **/store/index.js**

**import** Vuex **from** 'vuex'  
  
**const** createStore = () => {  
 **return new** Vuex.Store({  
 state: {  
 loadedPosts: []  
 },  
 getters: {  
 loadedPosts (state) {  
 **return** state.loadedPosts  
 }  
 },  
 mutations: {  
 setPosts (state, posts) {  
 state.loadedPosts = posts  
 }  
 },  
 actions: {  
 setPosts (vuexContext, posts) {  
 vuexContext.commit('setPosts', posts)  
 }  
 }  
 })  
}  
  
**export default** createStore

## Fetch

Para cargar datos podemos utilizar el método Fetch

Es importante tener en cuenta que:

1. fetch sólo puede ser utilizado dentro de **/pages**.
2. No es posible utilizar **this** ya que el método fetch es llamado antes de iniciar el componente.

computed: {  
 loadedPosts () {  
 **return this**.$store.getters.loadedPosts  
 }  
},  
fetch (context) {  
 **return new** Promise((resolve, reject) => {  
 // success case  
 setTimeout(() => {  
 resolve({  
 loadedPosts: [  
 {  
 id: '1',  
 title: 'First',  
 thumbnail: 'https://loscomentarios.com/files/hola/imagenes\_de\_hola-4389.gif',  
 previewText: 'Hola'  
 },  
 {  
 id: '2',  
 title: 'Second',  
 thumbnail: 'https://loscomentarios.com/files/hola/imagenes\_de\_hola-4389.gif',  
 previewText: 'Adios'  
 },  
 {  
 id: '3',  
 title: 'Third',  
 thumbnail: 'https://loscomentarios.com/files/hola/imagenes\_de\_hola-4389.gif',  
 previewText: 'Tralará'  
 }  
 ]  
 })  
 }, 1500)  
 // error case  
 // reject(new Error())  
 })  
 .then(data => {  
 context.store.commit('setPosts', data.loadedPosts)  
 })  
 .catch(e => {  
 context.error(e)  
 })  
}

## Nuxt server init method

Para inicializar datos desde el servidor podemos utilizar la acción **nuxtServerInit** en el **Store**. Esta acción es ejecutada por Nuxt.

actions: {  
 nuxtServerInit (vuexContext, context) {  
 **return new** Promise((resolve, reject) => {  
 // success case  
 setTimeout(() => {  
 vuexContext.commit('setPosts', [  
 {  
 id: '1',  
 title: 'First',  
 thumbnail: 'https://loscomentarios.com/files/hola/imagenes\_de\_hola-4389.gif',  
 previewText: 'Hola'  
 },  
 {  
 id: '2',  
 title: 'Second',  
 thumbnail: 'https://loscomentarios.com/files/hola/imagenes\_de\_hola-4389.gif',  
 previewText: 'Adios'  
 },  
 {  
 id: '3',  
 title: 'Third',  
 thumbnail: 'https://loscomentarios.com/files/hola/imagenes\_de\_hola-4389.gif',  
 previewText: 'Tralará'  
 }  
 ])  
 resolve()  
 }, 1500)  
 // error case  
 // reject(new Error())  
 })  
 .then(data => {  
 context.store.commit('setPosts', data.loadedPosts)  
 })  
 .catch(e => {  
 context.error(e)  
 })  
 }

## Axios

npm i –-save axios

1. Actualizar store mediante una petición HTTP a firebase

nuxtServerInit (vuexContext, context) {  
 **return** axios.get('https://nuxt-3b4a1.firebaseio.com/posts.json')  
 .then(res => {  
 **const** postsArray = []  
 **for** (**let** key **in** res.data) {  
 postsArray.push({...res.data[key], id: key})  
 }  
 vuexContext.commit('setPosts', postsArray)  
 })  
 .catch(e => {  
 console.error(e)  
 })  
},

1. Acceder a un solo post

asyncData (context) {  
  
 **return** axios.get('https://nuxt-3b4a1.firebaseio.com/posts/' + context.params.id + '.json')  
 .then(res => {  
 **return** {  
 loadedPost: res.data  
 }  
 })  
 .catch(e => {  
 console.error(e)  
 })  
}

1. Subir datos a firebase

methods: {  
 onSubmitted (postData) {  
 // firebase.database().ref('posts').push(postData)  
 axios.post('https://nuxt-3b4a1.firebaseio.com/posts.json', postData)  
 .then(response => {console.log(response)})  
 .catch(e => {console.error(e)})  
 }  
}

1. Actualizar datos

methods: {  
 updatePost (editedPost) {  
 axios.put('https://nuxt-3b4a1.firebaseio.com/posts' + **this**.$route.params.postId + '.json', editedPost)  
 .then(res => {  
 **this**.$router.push('/admin')  
 })  
 .catch(e => {  
 console.error(e)  
 })  
 }  
}

## Vuex & axios CRUD

Firebase Docs: <https://firebase.google.com/docs/database/>

Axios: <https://github.com/axios/axios>

* Todos los cambios en el store de vuex requieren reinicio, no carga en caliente

Podemos crear un CRUD con Vuex y axios

1. **store/index.js**

**import** Vuex **from** 'vuex'  
**import** axios **from** 'axios'  
  
**const** createStore = () => {  
 **return new** Vuex.Store({  
 state: {  
 loadedPosts: []  
 },  
 getters: {  
 loadedPosts (state) {  
 **return** state.loadedPosts  
 }  
 },  
 mutations: {  
 setPosts (state, posts) {  
 state.loadedPosts = posts  
 },  
 addPost (state, post) {  
 state.loadedPosts.push(post)  
 },  
 editPost (state, editedPost) {  
 **const** iPost = state.loadedPosts.findIndex(post => post.id === editedPost.id)  
 state.loadedPosts[iPost] = editedPost  
 },  
 rmPost (state, postId) {  
 **const** iPost = state.loadedPosts.findIndex(post => post.id === postId)  
 state.loadedPosts.splice(iPost, 1)  
 }  
 },  
 actions: {  
 nuxtServerInit (vuexContext, context) {  
 **return** axios.get('https://nuxt-3b4a1.firebaseio.com/posts.json')  
 .then(res => {  
 **const** postsArray = []  
 **for** (**let** key **in** res.data) {  
 postsArray.push({...res.data[key], id: key})  
 }  
 vuexContext.commit('setPosts', postsArray)  
 })  
 .catch(e => {  
 console.error(e)  
 })  
 },  
 setPosts (vuexContext, posts) {  
 vuexContext.commit('setPosts', posts)  
 },  
 addPost (vuexContext, post) {  
 **return** axios.post('https://nuxt-3b4a1.firebaseio.com/posts.json', post)  
 .then(res => {  
 vuexContext.commit('addPost', {...post, id: res.data.name})  
 })  
 .catch(e => {  
 console.error(e)  
 })  
 },  
 editPost (vuexContext, editedPost) {  
 **return** axios.put('https://nuxt-3b4a1.firebaseio.com/posts/' + editedPost.id + '.json', editedPost)  
 .then(res => {  
 vuexContext.commit('editPost', editedPost)  
 })  
 .catch(e => {  
 console.error(e)  
 })  
 },  
 rmPost (vuexContext, postId) {  
 **return** axios.delete('https://nuxt-3b4a1.firebaseio.com/posts/' + postId + '.json')  
 .then(res => {  
 vuexContext.commit('rmPost', postId)  
 })  
 .catch(e => {  
 console.error(e)  
 })  
 }  
 }  
 })  
}  
  
**export default** createStore

1. **Crear post**

<AdminPostForm @submit="addPost"></AdminPostForm>

[…]

methods: {  
 addPost (postData) {  
 // firebase.database().ref('posts').push(postData)  
 **this**.$store.dispatch('addPost', postData)  
 .then(() => {  
 **this**.$router.push('/admin')  
 })  
 }  
}

1. **Lista de posts**

<PostList :posts="loadedPosts"></PostList>

[…]

computed: {  
 loadedPosts () {  
 **return this**.$store.getters.loadedPosts  
 }  
}

1. **Detalle de post**

asyncData (context) {  
  
 **return** axios.get('https://nuxt-3b4a1.firebaseio.com/posts/' + context.params.id + '.json')  
 .then(res => {  
 **return** {  
 loadedPost: res.data  
 }  
 })  
 .catch(e => {  
 console.error(e)  
 })  
}

1. **Eliminar post**
   1. **components/Posts/PostList.vue**

<PostPreview  
 v-for="post in posts"  
 :key="post.id"  
 :id="post.id"  
 :title="post.title"  
 :previewText="post.previewText"  
 :thumbnail="post.thumbnail"  
 @rmPost="rmPost"></PostPreview>

[…]

methods: {  
 rmPost (postId) {  
 **this**.$emit('rmPost', postId)  
 }  
}

* 1. **components/Posts/PostPreview.vue**

<div class="post-preview">  
 <button type="button" class="close" aria-label="Close"  
 v-if="$route.path.includes('/admin/remove-post')"  
 @click="$emit('rmPost', id)">  
 <span aria-hidden="true">&times;</span>  
 </button>  
 <nuxt-link :to="$route.path.includes('/admin') ? '/admin/' + id : '/posts/' + id">  
 <article>  
 <div class="post-thumbnail"  
 :style="{'background-image': 'url(' + thumbnail + ')'}"></div>  
 <div class="post-content">  
 <h1>{{ title }}</h1>  
 <p>{{ previewText }}</p>  
 </div>  
 </article>  
 </nuxt-link>  
</div>

* 1. **pages/admin/remove-post/index.vue**

<PostList :posts="loadedPosts" @rmPost="rmPost"></PostList>

[…]

methods: {  
 rmPost (postId) {  
 // firebase.database().ref('posts').push(postData)  
 **this**.$store.dispatch('rmPost', postId)  
 /\*.then(() => {  
 this.$router.push('/admin')  
 })\*/  
 }  
},

## Firebase

Ref: <https://github.com/james2doyle/nuxt-firebase-auth>

$ npm i –-save firebase

1. Crear fichero de configuración con los datos de conexión a firebase

**/firebase-config.js**

module.exports = {  
 apiKey: "APIKEY",  
 authDomain: "DOMAIN",  
 databaseURL: "URL",  
 projectId: "ID",  
 storageBucket: "BUCKET",  
 messagingSenderId: "MESSAGE\_ID"  
}

1. Crear pugin de inicialización fireinit

**plugins/fireinit.js**

**import** firebaseConfig **from** '~/firebase-config.js'  
**import** firebase **from** 'firebase/app'  
**import** 'firebase/database'  
  
**if** (!firebaseConfig) {  
 **throw new** Error('missing firebase-config.js config')  
}  
  
**export default function** ({  
 store,  
 redirect  
 }) {  
 **if** (!firebase.apps.length) {  
 firebase.initializeApp(firebaseConfig)  
 }  
  
 **return** firebase  
}

1. Añadir plugin en la configuración de NUXT

**/nuxt.config.js**

/\*  
\*\* Plugins to load before mounting the App  
\*/  
plugins: [  
 {  
 src: '~/plugins/fireinit.js',  
 ssr: **false** }  
],

1. Utilizar firebase en componente

methods: {  
 onSubmitted (postData) {  
 firebase.database().ref('posts').push(postData)  
 }  
}

# Config, plugins & modules

## Mode

1. Universal: incluye SSR (server side rendering)
2. Spa: solo para aplicaciones SPA (single page app)

## Head

Todo lo que incluye en la cabecera HTML de toda la aplicación

head: {  
 title: 'WD Blog',  
 meta: [  
 { charset: 'utf-8' },  
 { name: 'viewport', content: 'width=device-width, initial-scale=1' },  
 { hid: 'description', name: 'description', content: pkg.description }  
 ],  
 link: [  
 { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }  
 ]  
},

Para añadir otros elementos en la cabecera de las páginas hijas se puede usar la directiva head:

* **pages/posts/index.vue**

head: {  
 title: 'Posts'  
}

## Loading

Barra de progreso controlada por NUXT al navegar por la app.

Para deshabilitarla asignar **loading** a **false**:

/\*  
\*\* Customize the progress-bar color  
\*/  
loading: **false**,

### Loading indicator

En las aplicaciones cuyo modo es SPA (mode: ’spa’) se puede añadir un spinner:

loadingIndicator: {  
 name: 'circle',  
 color: '#3B8070',  
 background: 'white'  
},

## CSS

Declarar hojas de estilo CSS globales

## Build

Permite modificar el proceso de compilación

## Env

Permite inyectar variables al contexto de la aplicación.

* **store/index.js**

env: {  
 firebaseUrl: process.env.FIREBASE\_URL || 'https://nuxt-3b4a1.firebaseio.com'  
}

* **nuxt.config.js**

rmPost (vuexContext, postId) {  
 **return** axios.delete(process.env.firebaseUrl + '/posts/' + postId + '.json')

## Generate

Permite modificar el proceso de generar portales estáticos

## rootDir

Modifica la ruta principal del proyecto

## Router

Sobrescribe y extender rutas generadas por NUXT.

Por ejemplo, para todas las rutas que no estén declaradas podemos redirigir a la página principal:

* **nuxt.config.js**

router: {  
 extendRoutes(routes, resolve) {  
 routes.push({  
 path: '\*',  
 component: resolve(\_\_dirname, 'pages/index.vue')  
 })  
 },  
 linkActiveClass: 'active'  
}

## srcDir

Permite definir la carpeta raiz de todas las carpetas predefinidas de NUXT:

* nuxt-src/
  + assets
  + components
  + layouts
  + middleware
  + pages
  + plugins
  + static
  + store

srcDir: 'nuxt-src/'

## transition

Permite configurar animaciones en la transición entre páginas de la aplicación

transition: {  
 name: 'page',  
 mode: 'out-in'  
}

## Plugins

Permite ejecutar código de manera global al inicializar nuestra aplicación. Ya que no tenemos acceso al fichero main.js podemos usar plugins para añadir funcionalidad global.

Por ejemplo, podemos declarar componentes de manera global:

* **plugins/core-components.js**

**import** Vue **from** 'vue'  
**import** AppButton **from** '@/components/UI/AppButton.vue'  
**import** AppControlInput **from** '@/components/UI/AppControlInput.vue'  
**import** PostList **from** '@/components/Posts/PostList.vue'  
  
Vue.component('AppButton', AppButton)  
Vue.component('AppControlInput', AppControlInput)  
Vue.component('PostList', PostList)

* **nuxt.config.js**

/\*  
\*\* Plugins to load before mounting the App  
\*/  
plugins: [  
 '~plugins/core-components.js'  
],

## Modules

Permite añadir módulos desarrollados por terceros.

* **nuxt.config.js**

modules: [  
 '@nuxtjs/axios'  
],  
  
axios: {  
 baseURL: 'https://nuxt-3b4a1.firebaseio.com',  
 credentials: **false**},

# Middleware & Authentification

Middleware permite definir funciones que se ejecutan antes de renderizar una o varias páginas.

1. La función middleware se ejecuta antes de renderizar la página en el servidor (si existe SSR) o en el cliente.

* **middleware/log.js**

**export default function** (context) {  
 console.log(context)  
}

* **pages/posts/index.vue**

**export default** {  
 middleware: 'log',  
 computed: {  
 loadedPosts () {  
 **return this**.$store.getters.loadedPosts  
 }  
 }  
}

Tambien se puede configurar middleware de manera global en el fichero de configuracion de Nuxt:

* **nuxt.config.js**

router: {

middleware: 'log'  
},

## Firebase Auth

* Firebase Auth REST API: <https://firebase.google.com/docs/reference/rest/auth/>

Podemos implementar un servicio de autentificacion con firebase.

1. Debemos activar authentification en Firebase para nuestro proyecto

### Registro/Login / Logout

* **pages/admin/auth/index.vue**

<template>  
 <div class="admin-auth-page">  
 <div class="auth-container">  
 <form @submit.prevent="authUser">  
 <AppControlInput type="email" v-model="email">E-Mail Address</AppControlInput>  
 <AppControlInput type="password" v-model="pass">Password</AppControlInput>  
 <AppButton type="submit">{{ isLogin ? 'Login' : 'Sign Up' }}</AppButton>  
 <AppButton  
 type="button"  
 btn-style="inverted"  
 style="margin-left: 10px"  
 @click="isLogin = !isLogin">Switch to {{ isLogin ? 'Signup' : 'Login' }}  
 </AppButton>  
 </form>  
 </div>  
 </div>  
</template>  
  
<script>  
 **import** AppControlInput **from** '@/components/UI/AppControlInput'  
 **import** AppButton **from** '@/components/UI/AppButton'  
  
 **export default** {  
 name: 'AdminAuthPage',  
 components: {  
 AppControlInput,  
 AppButton  
 },  
 data () {  
 **return** {  
 isLogin: **true**,  
 email: '',  
 pass: ''  
 }  
 },  
 methods: {  
 authUser () {  
 **this**.$store.dispatch('authUser', {  
 isLogin: **this**.isLogin,  
 email: **this**.email,  
 pass: **this**.pass  
 })  
 .then(() => {  
 **this**.$router.push('/admin')  
 })  
 }  
 }  
 }  
</script>  
  
<style scoped>  
 .admin-auth-page {  
 padding: 20px;  
 }  
  
 .auth-container {  
 border: 1px solid #ccc;  
 border-radius: 5px;  
 box-shadow: 0 2px 2px #ccc;  
 width: 300px;  
 margin: auto;  
 padding: 10px;  
 box-sizing: border-box;  
 }  
</style>

* **store/index.js**

//mutations

setToken (state, token) {  
 state.token = token  
},  
clearToken (state) {  
 state.token = **null**}

[…]

// actions

authUser (vuexContext, authData) {  
 // Register  
 **let** url = process.env.firebaseSignUpUrl  
 **if** (authData.isLogin) {  
 // login  
 url = process.env.firebaseLoginUrl  
 }  
  
 **return** axios.post(url + process.env.firebaseApiKey,  
 {  
 email: authData.email,  
 password: authData.pass,  
 returnSecureToken: **true** })  
 .then(res => {  
 **const** token = res.data.idToken;  
 **const** expiresInMillis = Number(res.data.expiresIn) \* 1000;  
 **const** expiration = **new** Date().getTime() + expiresInMillis;  
  
 vuexContext.commit('setToken', token)  
 localStorage.setItem('token', token)  
 localStorage.setItem('tokenExpiration', expiration)  
 Cookie.set('token', token)  
 Cookie.set('tokenExpiration', expiration)  
 })  
 .catch(e => console.error(e))  
},  
initAuth (vuexContext, req) {  
 **let** token, tokenExpiration = **null  
 if** (req && req.headers.cookie) {  
 // server only  
 token = req.headers.cookie  
 .split(';')  
 .find(c => {  
 **if** (c.trim().startsWith('token=')) {  
 **return** c.split('=')[1]  
 }  
 }  
 )  
 tokenExpiration = req.headers.cookie  
 .split(';')  
 .find(c => {  
 **if** (c.trim().startsWith('tokenExpiration=')) {  
 **return** c.split('=')[1]  
 }  
 }  
 )  
 } **else if** (!req) {  
 // client only  
 token = localStorage.getItem('token')  
 tokenExpiration = localStorage.getItem('tokenExpiration')  
 }  
 **if** (token && **new** Date().getTime() < Number(tokenExpiration)) {  
 // load token if exists  
 vuexContext.commit('setToken', token)  
 } **else** {  
 // clear if undefined or expired token  
 vuexContext.dispatch('logout')  
 }  
},  
logout (vuexContext) {  
 vuexContext.commit('clearToken')  
 Cookie.remove('token')  
 Cookie.remove('tokenExpiration')  
 **if** (process.client) {  
 // server only  
 localStorage.removeItem('token')  
 localStorage.removeItem('tokenExpiration')  
 }  
}

### Mantener usuario autentificado al recargar la pagina

* **layouts/admin.vue**

**export default** {  
 middleware: ['check-auth', 'auth']  
}

* **middleware/check-auth.js**

**export default function** (context) {  
 context.store.dispatch('initAuth', process.client ? **null** : context.req)  
}

* **store/index.js**

initAuth (vuexContext, req) {  
 **let** token, tokenExpiration = **null  
 if** (req && req.headers.cookie) {  
 // server only  
 token = req.headers.cookie  
 .split(';')  
 .find(c => c.trim().startsWith('cToken='))  
 .split('=')[1]  
 tokenExpiration = req.headers.cookie  
 .split(';')  
 .find(c => c.trim().startsWith('cTokenExpiration='))  
 .split('=')[1]  
 } **else if** (req == **null**) {  
 // client only  
 token = localStorage.getItem('token')  
 tokenExpiration = localStorage.getItem('tokenExpiration')  
 }  
 **if** (token && **new** Date().getTime() < Number(tokenExpiration)) {  
 // load token if exists  
 vuexContext.commit('setToken', token)  
 } **else** {  
 // clear if undefined or expired token  
 vuexContext.commit('clearToken')  
 }  
}

### Limitar acceso a zona privada

* **store/index.js**

getters: {  
 isAuthenticated (state) {  
 **return** state.token != **null** }  
},

* **Middleware/auth.js**

**export default function** (context) {  
 **if**(!context.store.getters.isAuthenticated){  
 context.redirect('/admin/auth')  
 }  
}

# Producción

## Universal (SEO)

Es necesaria una versión de Node js superior a 8.x en el servidor.

Primera vista es renderizada dinámicamente en el servidor. Después funciona como una SPA.

1. Genera la carpeta dist que contiene la app con el código optimizado para producción.

npm run build

1. Sube el contenido del proyecto al servidor, incluyendo la carpeta .nuxt y node\_modules y los ficheros packege.json y nuxt.config.js

npm run start

(nuxt start)

Lanza la aplicación en el servidor

## SPA

Todo el contenido se genera en el cliente.

Es importante tener en centa que no se pueden utilizar funciones como nuxtServerInit en el Store. Como alternativa podemos utilizar created o mounted junto con asyncData o fetch.

También es necesario que el servidor devuelva la página principal /index.html para comenzar con la app.

## Estática (SEO)

Se genera las vistas pre-renderizadas por el servidor de forma que devuelve un HTML con contenido. Después se comporta como una SPA.

1. Para incluir rutas dinámicas debemos configurar nuxt.config.js/generate

Pdemos añadirlas manualmente:

generate: {  
 routes:

**‘/posts/1’,**

**‘/posts/2’,**

**[…]**

}  
}

O devolviendo una promesa y ejecutando una llamada HTTP:

generate: {  
 routes: **function** () {  
 **return** axios.get('https://nuxt-3b4a1.firebaseio.com/posts.json')  
 .then(res => {  
 **const** postsArray = []  
 **for** (**let** key **in** res.data) {  
 postsArray.push('/posts/' + key)  
 }  
 **return** postsArray  
 })  
 .catch(e => {  
 console.error(e)  
 })  
 }  
}

1. Pre-renderiza el HTML para cada ruta (no incluye rutas dinámicas)

npm run generate

1. Subir carpeta dist al servidor

### Optimize generate

Podemos optimizar la generación estática de contenido dinámico limitando las peticiones HTTP. Para ello podemos construir un objeto con la ruta y el contenido:

* **nuxt.config.js**

generate: {  
 routes: **function** () {  
 **return** axios.get('https://nuxt-3b4a1.firebaseio.com/posts.json')  
 .then(res => {  
 **const** postsArray = []  
 **for** (**let** key **in** res.data) {  
 postsArray.push({  
 route: '/posts/' + key,  
 payload: {postData: res.data[key]}  
 })  
 }  
 **return** postsArray  
 })  
 .catch(e => {  
 console.error(e)  
 })  
 }  
}

* **pages/posts/\_id/index.vue**

asyncData (context) {  
  
 **if** (context.payload) {  
 **return** {  
 loadedPost: context.payload.postData  
 }  
 } **else** {  
 **return** axios.get(process.env.firebaseUrl + '/posts/' + context.params.id + '.json')  
 .then(res => {  
 context.app.head.title = res.data.title  
 **return** {  
 loadedPost: res.data  
 }  
 })  
 .catch(e => {  
 console.error(e)  
 })  
 }

# Optimiza el tamaño de los ficheros generados

build: {  
 /\*  
 \*\* You can extend webpack config here  
 \*/  
 maxChunkSize: 300000,  
 extend (config, ctx) {  
 }  
},