ON THIS PAGE

Activate the Store

Modules

Example folder structure

Plugins in the Store

The nuxtServerInit Action

Vuex Strict Mode

PARTNERS

Vercel

swell

SUPPORT US

NuxtJS needs you!

By whitelisting nuxtjs.org on your Ad-Blocker, you support our work and help us financially.

Store directory

the store.

> ⚠ *This directory cannot be renamed without extra configuration.*

Using a store to manage the state is important for every big application. That's why Nuxt.js implements Vuex in its core.

Activate the Store

Nuxt.js will look for the `store` directory. If it contains a file, that isn't a hidden file or a `README.md` file, then the store will be activated. This means that Nuxt will:

1. Import Vuex,
2. Add the `store` option to the root Vue instance.

Modules

Every `.js` file inside the `store` directory is transformed as a namespaced module (`index` being the root module). Your `state` value should always be a `function` to avoid unwanted *shared* state on the server side.

To get started, export the state as a function, and the mutations and actions as objects.

store/index.js

```
export const state = () => ({
  counter: 0
})

export const mutations = {
  increment(state) {
    state.counter++
  }
}
```

Then, you can have a `store/todos.js` file:

store/todos.js

```
export const mutations = {
  add(state, text) {
    state.list.push({
      text,
      done: false
    })
  },
  remove(state, { todo }) {
    state.list.splice(state.list.indexOf(todo), 1)
  },
  toggle(state, todo) {
    todo.done = !todo.done
  }
}
```

The store will be created as such:

```
new Vuex.Store({
  state: () => ({
    counter: 0
  }),
  mutations: {
    increment(state) {
      state.counter++
    }
  },
  modules: {
    todos: {

      namespaced: true,
      state: () => ({
        list: []
      }),
      mutations: {
        add(state, { text }) {
          state.list.push({
            text,
            done: false
        remove(state, { todo }) {
          state.list.splice(state.list.indexOf(todo), 1)
        },
        toggle(state, { todo }) {
          todo.done = !todo.done
        }
      }
    }
  }
})
```

And in your `pages/todos.vue`, using the `todos` module:

pages/todos.vue

```
<template>
  <ul>
    <li v-for="todo in todos" :key="todo.text">
      <input :checked="todo.done" @change="toggle(todo)" type="checkbox">
      <span :class="{ done: todo.done }">{{ todo.text }}</span>
    </li>
    <li><input @keyup.enter="addTodo" placeholder="What needs to be done?"></li>
  </ul>
</template>

<script>
import { mapMutations } from 'vuex'

export default {
  computed: {
    todos () {
      return this.$store.state.todos.list
    }
  },
  methods: {
    addTodo (e) {
      this.$store.commit('todos/add', e.target.value)
      e.target.value = ''
    },
    ...mapMutations({
      toggle: 'todos/toggle'
```

```
</script>

<style>
.done {
  text-decoration: line-through;
}
</style>
```

The module method also works for top-level definitions without implementing a sub-directory in the store directory.

Example for state: you create a file `store/state.js` and add the following.

```
export default () => ({
  counter: 0
})
```

And the corresponding mutations can be in the file `store/mutations.js`

store/mutations.js

```
export default {
  increment(state) {
    state.counter++
  }
}
```

Example folder structure

A complex store setup file/folder structure might look like this:

```
 store/
--| index.js
--| ui.js
--| shop/
----| cart/
------| actions.js
```

```
----| products/
------| mutations.js
------| state.js
------| itemsGroup1/
--------| state.js
```

Plugins in the Store

You can add additional plugins to the store by putting them into the `store/index.js` file:

store/index.js
```
import myPlugin from 'myPlugin'

export const plugins = [myPlugin]

export const state = () => ({
  counter: 0
})

export const mutations = {
  increment(state) {
    state.counter++
  }
}
```

More information about the plugins: Vuex documentation.

The nuxtServerInit Action

If the action `nuxtServerInit` is defined in the store and the mode is `universal`, Nuxt.js will call it with the context (only from the server-side). It's useful when we have some data on the server we want to give directly to the client-side.

For example, let's say we have sessions on the server-side and we can access the connected user through `req.session.user`. To add the authenticated user to our store, we

```
    nuxtServerInit ({ commit }, { req }) {
      if (req.session.user) {
        commit('user', req.session.user)
      }
    }
  }
```

⚠ Only the primary module (in `store/index.js`) will receive this action. You'll need to chain your module actions from there.

The context is given to `nuxtServerInit` as the 2nd argument in the `asyncData` method.

If `nuxt generate` is ran, `nuxtServerInit` will be executed for every dynamic route generated.

ℹ Asynchronous `nuxtServerInit` actions must return a Promise or leverage async/await to allow the nuxt server to wait on them.

store/index.js

```
actions: {
  async nuxtServerInit({ dispatch }) {
    await dispatch('core/load')
  }
}
```

Vuex Strict Mode

Strict mode is enabled by default on dev mode and turned off in production mode. To disable strict mode in dev, follow the below example in `store/index.js`:

```
export const strict = false
```

← Static directory

Nuxt configuration file →

mikhail sergienko Daniel Roe Debbie O'Brien pooya parsa Sébastien Chopin Alexandre Chopin Tom Richter Joe Mirizio Max Schweikart Krystle Salazar Benjamin Canac fgiraud Sergey Bedritsky Alba Silvente Fuentes Alexander Lichter Krutie Patel Pim Ahad Birang

Caught a mistake or want to contribute to the documentation? Edit this page on GitHub!

NuxtJS Newsletter

Get the latest Nuxt news to your inbox, curated by the NuxtJS team.

Email SUBSCRIBE