

## Step 1: Install Vue using Vue cli.

Type the following command in your terminal.

```
npm install -g @vue/cli  
  
# or  
  
yarn global add @vue/cli
```

If you find any installation error, then try command in administrator mode.

Now, create a project using the following command.

```
vue create vuexapi
```

Now, go into that project.

```
cd vuexapi
```

Open the project in your favorite editor.

```
code .
```

## Step 2: Install Vuex, Axios and vue-axios libraries.

Go to the terminal and install the **vuex**, **axios** and, a **vue-axios** library using the following command.

```
npm install vuex axios vue-axios --save
```

Install **Bootstrap 4** as well.

```
npm install bootstrap --save
```

Import this file inside **App.vue** file.

```
<template>
  <div id="app">
    <HelloWorld />
  </div>
</template>

<script>
import HelloWorld from './components/HelloWorld.vue'
import '../node_modules/bootstrap/dist/css/bootstrap.min.css';

export default {
  name: 'app',
  components: {
    HelloWorld
  }
}
</script>
```

## Step 3: Create a JSON server to serve the data.

In a real web application, we have a data coming from an API. So let's create a static **JSON** file and then we serve that file as an API via **json-server** package. So let us install that library first.

```
yarn global add json-server

# or

npm install -g json-server
```

Now we need to create a folder inside **src directory** called **data** and in that folder, create one file called **db.json**. Let us add the following data inside a **db.json** file.

```
{
  "results": [
    {
      "id": "1",
      "name": "BTC",
      "price": "1000"
    },
    {
      "id": "2",
      "name": "LTC",
      "price": "150"
    },
    {
      "id": "3",
      "name": "Ethereum",
      "price": "800"
    },
    {
      "id": "4",
      "name": "BCH",
      "price": "1500"
    }
  ]
}
```

Now, go into your terminal and type the following command to start a JSON server.

```
json-server --watch src/data/db.json --port 4000
```

Now, we have a server running that can feed the data to our React Bootstrap Application.

Our JSON server is started at port: 4000 and URL is: <http://localhost:4000/results>

## Step 4: Create a vuex store.

Inside **src** folder, create one folder called **a store** and inside that folder, create one file called **store.js**.

Write the following code inside **a store.js** file.

```
// store.js

import Vue from 'vue'
import Vuex from 'vuex'
import axios from 'axios'
import VueAxios from 'vue-axios'

Vue.use(Vuex)
Vue.use(VueAxios, axios)

export default new Vuex.Store({
  state: {
    coins: []
  },
  actions: {

  },
  mutations: {

  }
})
```

We are merely displaying the coins. So our primary state object contains **coins** array.

Our store contains **state**, **actions**, and **mutations**.

The **state** contains our whole vue application state.

The **actions** contain the function that will call our JSON Server via Axios and get the response.

The only way to change state in a Vuex store is by committing a mutation.

Vuex **mutations** are very similar to events: each mutation has a string **type** and a **handler**. The handler function is where we perform actual state modifications, and it will receive the state as the first argument.

## Step 5: Create an action.

Inside **the store.js** file, we need to create an action that will fetch the data from an API.

So, we can write the store function like this.

```
// store.js

import Vue from 'vue'
import Vuex from 'vuex'
import axios from 'axios'
import VueAxios from 'vue-axios'

Vue.use(Vuex)
Vue.use(VueAxios, axios)

export default new Vuex.Store({
  state: {
    coins: []
  },
  actions: {
    loadCoins ({ commit }) {
      axios
        .get('http://localhost:4000/results')
        .then(r => r.data)
        .then(coins => {
          console.log(coins)
        })
    }
  },
  mutations: {

  }
})
```

Inside **actions**, we have created one function called **loadCoins**. This function will take commit as an argument and then call the mutation using commit function.

So, we need to write **mutations** as well. Our final **store.js** file looks like this.

```
// store.js

import Vue from 'vue'
import Vuex from 'vuex'
import axios from 'axios'
import VueAxios from 'vue-axios'

Vue.use(Vuex)
Vue.use(VueAxios, axios)

export default new Vuex.Store({
  state: {
    coins: []
  },
  actions: {
    loadCoins ({ commit }) {
      axios
        .get('http://localhost:4000/results')
        .then(r => r.data)
        .then(coins => {
          commit('SET_COINS', coins)
        })
    }
  },
  mutations: {
    SET_COINS (state, coins) {
      state.coins = coins
    }
  }
})
```

So after the **coins** are fetched, we can commit a mutation,

In our case, the mutation is **SET\_COINS**. So we call that mutation function with an argument of **coins**.

Set the **coins** state with our fetched coins.

Now, we can use these **coins** to display the data.

## Step 6: Display the coins in table format.

In the default **Vue.js** project, we get the **HelloWorld.vue** file. So write the following code inside **HelloWorld.vue** file.



```
<template>
  <div class="container">
    <table class="table table-striped">
      <thead>
        <tr>
          <th>ID</th>
          <th>Name</th>
          <th>Price</th>
        </tr>
      </thead>
      <tbody>
        <tr v-for="coin in coins" :key="coin.id">
          <td>{{ coin.id }}</td>
          <td>{{ coin.name }}</td>
          <td>{{ coin.price }}</td>
        </tr>
      </tbody>
    </table>
  </div>
</template>

<script>
import { mapState } from 'vuex'
export default {
  name: 'HelloWorld',
  mounted () {
    this.$store.dispatch('loadCoins')
  },
  computed: mapState([
    'coins'
  ])
}

</script>
```

So, what I have done is when the **component** is mounted, we call the **store's action**.

In our case that action is **loadCoins**.

So, it will fetch the **coins** and commit the mutation called **SET\_COINS**.

This mutation set our **Vue.js** application state's coins array to the fetched coins.

Now, that **coins** we map here inside **computed** properties.

Finally, loop through all the coins and display it as a table format.