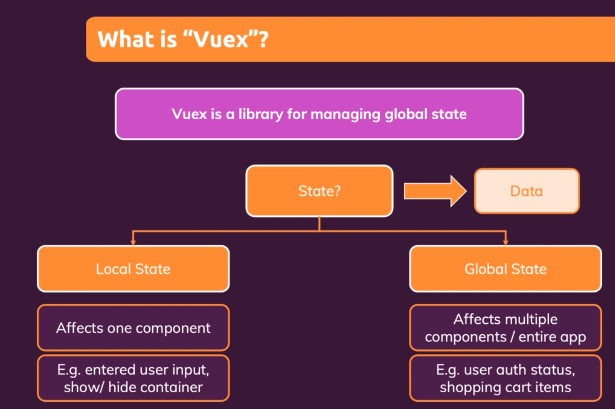
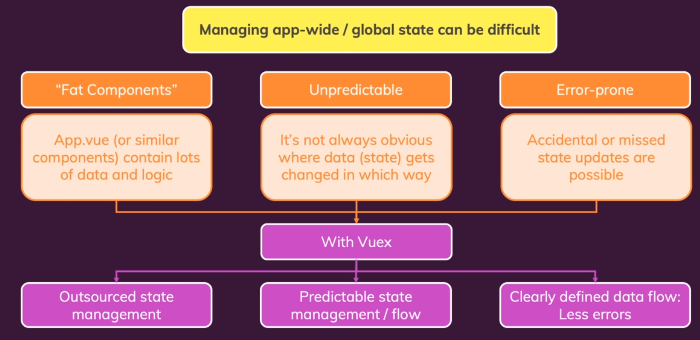
**Section 15 – Vuex**

Chapter 207 – What And Why?

1. Managing data, state, that affects different components of your app can be tricky, that is way we use Vuex.



1. Why ?



Chapter 208 – Creating And Using A Store

1. Install Vuex

npm install --save vuex

Chapter 209 – Connecting Components To State

1. AddOne to the counter

<template>

  <base-container title="Vuex">

    <the-counter></the-counter>

    <button @click="addOne">Add 1</button>

  </base-container>

</template>

export default {

  components: {

    BaseContainer,

    TheCounter

  },

  methods: {

    addOne() {

      this.$store.state.counter++;

    }

  }

};

Chapter 210 – Introducing Mutations – A Better Way Of Changing Data

1. Mutations are clearly defined methods, which have the logic to update the state. And from inside our component, we should , in the end, just trigger those mutations, instead of directly manipulating the state. By triggering mutations, all components that want to edit a state to do it in the same way and guaranteed that they do it in the same way.

const store = createStore({

  state() {

    return {

      counter: 0

    };

  },

  mutations: {

    increment(state) {

      state.counter += 1;

    }

  }

});

<template>

  <base-container title="Vuex">

    <the-counter></the-counter>

    <button @click="addOne">Add 1</button>

    <change-counter></change-counter>

  </base-container>

</template>

In the same file will be like below.

export default {

  components: {

    BaseContainer,

    TheCounter,

    ChangeCounter

  },

  methods: {

    addOne() {

      this.$store.commit('increment');

    }

  }

};

On the change counter will be like below.

export default {

    methods: {

        addOne() {

            this.$store.commit('increment');

        }

    }

}

Chapter 211 – Passing Data To Mutations With Payloads

1. Some mutations might require arguments. In this example, we want to add some value for the counter through the parameter. So, in this example, payloads extra data can be passed to our mutations.

methods: {

    addOne() {

      this.$store.commit('increase', { value: 10 });

    }

}

Or we can write like below.

methods: {

    addOne() {

      this.$store.commit({

        type: 'increase',

        value: 10

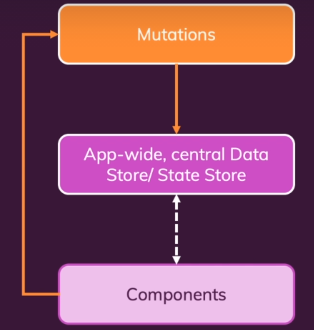
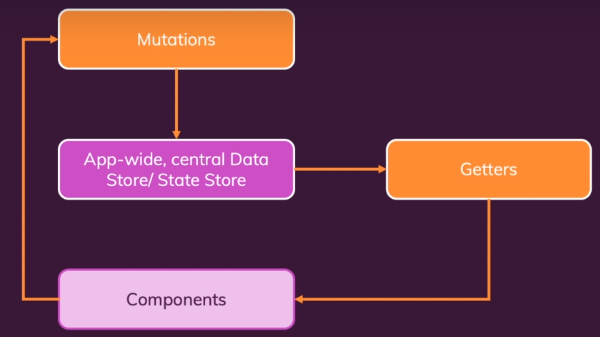
      });

    }

}

Chapter 212 – Introducing Getters – A Better Way Of Getting Data

1. We could also argue directly reading a state. For example, in this part, we have two printed value with the same variable name ‘value’, we may do duplicate function works, we must add 10, but it will add 20. Therefore, we add Getters and delete the connection between App-wide and components. We also can add normalizedCounter.

const store = createStore({

  state() {

    return {

      counter: 0

    };

  },

  mutations: {

    increment(state) {

      state.counter += 2;

    },

    increase(state, payload) {

      state.counter = state.counter + payload.value;

    }

  },

  getters: {

    finalCounter(state) {

      return state.counter \* 3;

    },

    // normalizedCounter(state) {

    //   const finalCounter = state.counter \* 3;

    normalizedCounter(state, getters) {

      const finalCounter = getters.finalCounter;

      if (finalCounter < 0) {

        return 0;

}

      if (finalCounter > 100) {

        return 100;

      }

      return finalCounter;

    }

  }

});

And to get the data we can use this method.

export default {

  computed: {

    counter() {

      // return this.$store.state.counter \* 2;

      return this.$store.getters.finalCounter;

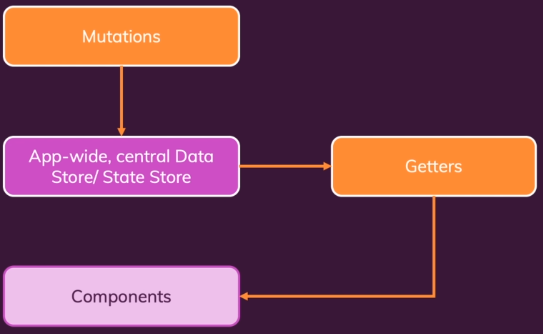
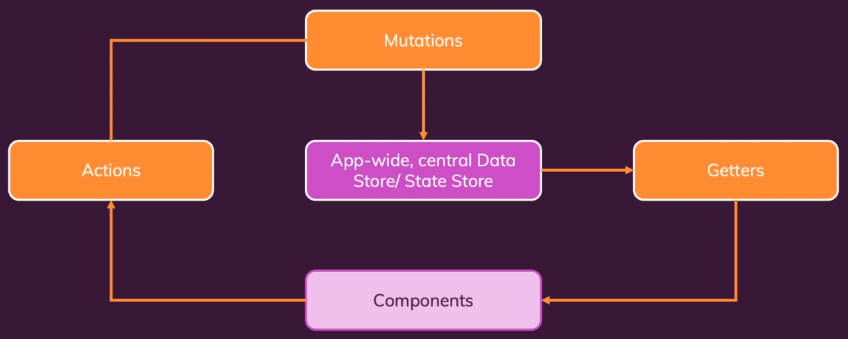
    },

  },

};

Chapter 213 – Running Async Code With Actions

1. If we want to add the value by 2 after several second, if we want to make an asynchronous result, and give the correct result. We can add actions to allow to run asynchronous code.

actions: {

     increment(context) {

        setTimeout(function() {

          context.commit('increment');

        }, 2000);

     },

     increase(context, payload) {

       context.commit('increase', payload);

     }

  },

  getters: {

    finalCounter(state) {

      return state.counter \* 3;

    },

    // normalizedCounter(state) {

    //   const finalCounter = state.counter \* 3;

    normalizedCounter(state, getters) {

      const finalCounter = getters.finalCounter;

      if (finalCounter < 0) {

        return 0;

      }

      if (finalCounter > 100) {

        return 100;

      }

      return finalCounter;

    }

}

export default {

    methods: {

        addOne() {

            // this.$store.commit('increment');

            this.$store.dispatch('increment');

        }

    }

}

Chapter 214 – Understanding The Actions “Context”

1. If we try to print using console.log the action result can be seen like below.



Chapter 215 – Using Mapper Helpers

1. mapGetters in Vuex is used to return an object which you can then spread with the free.spread operator into your computed property object. After we do this configuration, the result is same like before but now we use mapGetters, especially when you use multiple getters and not drill unnecessarily into the store all the time we might save same code.

import { mapGetters } from 'vuex';

export default {

  computed: {

    ...mapGetters(['finalCounter'])

  },

};

1. We have something similar for state and mutations, but we don’t really use those in our components. mapActions which also exists to mapActions into your components.

export default {

    methods: {

        ...mapActions(['increment', 'increase'])

    }

}

export default {

    methods: {

        ...mapActions({

            inc: 'increment',

            increase: 'increase'

        })

    }

}

Chapter 216 – Example: Adding More State

1. How if we want to deal with user authentication ?

UserAuth.vue

<template>

    <button @click="login" v-if="!isAuth">Login</button>

    <button @click="logout" v-if="isAuth">Logout</button>

</template>

<script>

export default {

    methods: {

        login() {

            this.$store.dispatch('login');

        },

        logout() {

            this.$store.dispatch('logout');

        }

    },

    computed: {

        isAuth() {

            return this.$store.getters.userIsAuthenticated;

        }

    }

}

</script>

App.vue

export default {

  components: {

    BaseContainer,

    TheCounter,

    ChangeCounter,

    FavoriteValue,

    UserAuth

  },

  computed: {

    isAuth() {

        return this.$store.getters.userIsAuthenticated;

    }

  },

  methods: {

    addOne() {

      this.$store.dispatch({

        type: 'increase',

        value: 10

      });

    }

  }

};

Chapter 217 – Organizing Your Store With Modules

1. We can separate our module into two modules like below.

const counterModule = {

  modules: {

    numbers: counterModule

  },

  state() {

    return {

      counter: 0,

    };

  },

  mutations: {

    increment(state) {

      state.counter += 2;

    },

    increase(state, payload) {

      state.counter = state.counter + payload.value;

    }

  },

  actions: {

    increment(context) {

      setTimeout(function() {

        context.commit('increment');

      }, 2000);

   },

   increase(context, payload) {

     console.log(context);

     context.commit('increase', payload);

   }

  },

  getters: {

    finalCounter(state) {

      return state.counter \* 3;

    },

    // normalizedCounter(state) {

    //   const finalCounter = state.counter \* 3;

    normalizedCounter(state, getters) {

      const finalCounter = getters.finalCounter;

      if (finalCounter < 0) {

        return 0;

      }

      if (finalCounter > 100) {

        return 100;

      }

      return finalCounter;

    }

  }

};

const store = createStore({

  state() {

    return {

      isLoggedIn: false

    };

  },

  mutations: {

    setAuth(state, payload) {

      state.isLoggedIn = payload.isAuth;

    }

  },

  actions: {

     login(context) {

       context.commit('setAuth', { isAuth: true });

     },

     logout(context) {

      context.commit('setAuth', { isAuth: false });

    }

  },

  getters: {

    userIsAuthenticated(state) {

      return state.isLoggedIn;

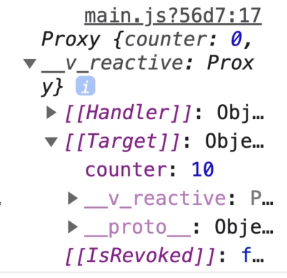
    }

  }

});

Chapter 218 – Understanding Local Module State

1. If we print console.log for increase method, the result will be like below.



Chapter 219 – Namespacing Modules

1. We can make the module in local, we can make it name spaced to make sure that multiple modules are clearly separated from each other. Name spaced is used to differentiate between one module and another module. If we do this, the module counter are no longer available on the main store.

const counterModule = {

  namespaced: true,

  state() {

    return {

      counter: 0,

    };

  },

  mutations: {

Chapter 220 – Structuring Vuex Code And Files

1. We can separate this code into some parts like below.

const store = createStore({

  state() {

    return {

      isLoggedIn: false

    };

  },

  mutations: {

    setAuth(state, payload) {

      state.isLoggedIn = payload.isAuth;

    }

  },

  actions: {

     login(context) {

       context.commit('setAuth', { isAuth: true });

     },

     logout(context) {

      context.commit('setAuth', { isAuth: false });

    }

  },

  getters: {

    userIsAuthenticated(state) {

      return state.isLoggedIn;

    }

  }

});

The result like below.

Actions.js

export default {

    login(context) {

        context.commit('setAuth', { isAuth: true });

    },

    logout(context) {

        context.commit('setAuth', { isAuth: false });

    }

}

Getters.js

export default {

    userIsAuthenticated(state) {

        return state.isLoggedIn;

    }

}

Index.js

import { createStore } from 'vuex';

import rootMutations from './mutations.js';

import rootActions from './actions.js';

import rootGetters from './getters.js';

import counterModule from './counter/index.js';

const store = createStore({

    modules: {

      numbers: counterModule

    },

    state() {

      return {

        isLoggedIn: false

      };

    },

    mutations: rootMutations,

    // {

    //   setAuth(state, payload) {

    //     state.isLoggedIn = payload.isAuth;

    //   }

    // },

    actions: rootActions,

    // {

    // },

    getters: rootGetters,

    // {

    // }

});

export default store;

Mutations.js

export default {

    setAuth(state, payload) {

        state.isLoggedIn = payload.isAuth;

    }

};