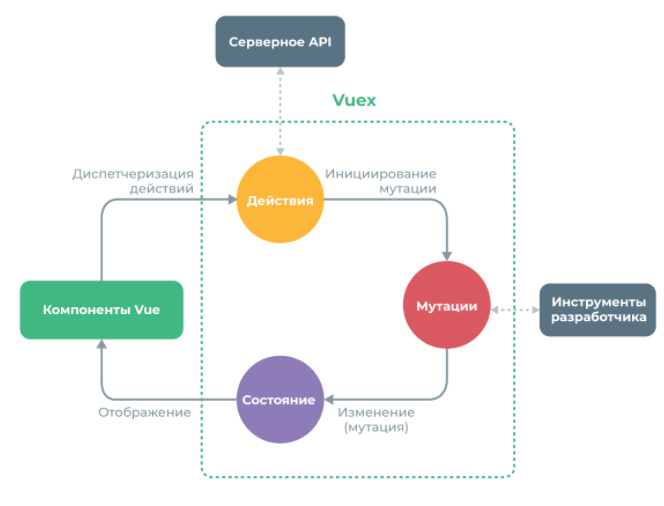
**Что такое Vuex?**



# Components -> данныйды Actions-қа жібереді

      this.$store.dispatch("increment", this.number);

# Actions-данныйды қабылдап Mutation - ға жіберді

  actions: {

    increment({ commit }, payload) {

      commit("increment", payload);

    },

  },

# Mutations ол данныйды функцияға салады, өзгертеді

# оны State - те глобаль переменныйға теңестіреді

state: {

    arr: [],

    count: 0,

  },

mutations: {

    increment(state, payload) {

      state.arr = payload;

    },

  },

# Getters - арқылы -> Components -тен көрсетеді

# (*State - тен сразу шығару кей жағдайда қате береді)*

  getters: {

    COUNT: (state) => state.arr,

  },

# *Шығаратын беттен импорт арқылы шақыру керек*

import { mapGetters } from "vuex";

  computed: {

    ...mapGetters(["COUNT"]),

  },

  <div>

    {{ COUNT  }}

  </div>

**Действия (Actions)** - данный қабылдау, серверден данный алады, ассинхронды байланыс қыла алады, мутатциядағы событяны шақыра алады, (шығарады..көрсете алады)

**Мутации (Mutations)** - данныйларды өзгерту, события, синхронды болады,

**Состояние (State)** - Data сияқты данныйлар жазылады, барлық компонентке доступ болатын переменныйлар ,

осы данный дағы заттарды Мутация өзгертеді  
**Геттеры (Getters)** - көрсету алдында filter қылу, ұзындығын, санын анықтап көрсетуге болады ,

**Модули (Modules)** - үлкен проекттерде түсінікті болу үшін бөліп тастау

Данныйлар **Actions**-қа жіберіледі

**Mutations** -та өңделеді

**Getters** -те қайта (көрсетіледі) шақырылады

----------------------------------

**#State**

## Getting Vuex State into Vue Components

`  state: {

  numberState: 10,

    posts: [1, 4, 34, 7, 3, 6],

  },

  console.log(this.$store.state.post.posts);

## The mapState Helper ???

import { mapState } from 'vuex'

computed: mapState([

// map this.count to store.state.count

'count'

])

----------------------------------

**#Getting**

## Property-Style Access ???

getters: {

// ...

doneTodosCount: (state, getters) => {

return getters.doneTodos.length

}

}

## Гетерлер сонымен қатар 2 -ші аргумент ретінде басқа гетерлерді алады:

## Method-Style Access

getters: {

   Creates: (state) => state.posts.filter((todo =>todo/2>= 3),

// Creates: (state) => (id) => state.posts.filter((todo)=>todo/2 >= id),

  },

console.log(this.$store.getters.Creates);

//console.log(this.$store.getters.Creates(10));

// console.log(this.$store.state.post.posts.filter((todo) => todo / 2 >= 3));

## The mapGetters Helper

getters: {

    COUNT: (state) => state.numberState,

    createPostes(state) {

      return state.posts;

    },

  },

import { mapGetters } from "vuex";

 computed: {

    ...mapGetters(["COUNT", "createPostes"]),

  },

    console.log(this.COUNT);

----------------------------------

mutations: {

increment (state, n) {

state.count += n

}

}

store.commit('increment', 10)

## Object-Style Commit

store.commit({

type: 'increment',

amount: 10

})

mutations: {

increment (state, payload) {

state.count += payload.amount

}

}

## Using Constants for Mutation Types

export const SOME\_MUTATION = 'SOME\_MUTATION'

import { SOME\_MUTATION } from './mutation-types'

state: { ... },

mutations: {

// we can use the ES2015 computed property name feature

// to use a constant as the function name

[SOME\_MUTATION] (state) {

// mutate state

}

}

## Mutations Must Be Synchronous

mutations: {

someMutation (state) {

api.callAsyncMethod(() => {

state.count++

})

}

}

## Committing Mutations in Components

import { mapMutations } from 'vuex'

export default {

methods: {

...mapMutations([

'increment', // map `this.increment()` to `this.$store.commit('increment')`

// `mapMutations` also supports payloads:

'incrementBy' // map `this.incrementBy(amount)` to `this.$store.commit('incrementBy', amount)`

]),

...mapMutations({

add: 'increment' // map `this.add()` to `this.$store.commit('increment')`

})

}

}

On to Actions

store.commit('increment')

---------------------------------------

state: {

count: 0

},

mutations: {

increment (state) {

state.count++

}

},

actions: {

increment (context) {

context.commit('increment')

}

}

actions: {

increment ({ commit }) {

commit('increment')

}

}

store.dispatch('increment')

actions: {

incrementAsync ({ commit }) {

setTimeout(() => {

commit('increment')

}, 1000)

}

}

// dispatch with a payload

store.dispatch('incrementAsync', {

amount: 10 ???

})

// dispatch with an object

store.dispatch({

type: 'incrementAsync',

amount: 10

})

actions: {

checkout ({ commit, state }, products) {

// save the items currently in the cart

const savedCartItems = [...state.cart.added]

// send out checkout request, and optimistically

// clear the cart

commit(types.CHECKOUT\_REQUEST)

// the shop API accepts a success callback and a failure callback

shop.buyProducts(

products,

// handle success

() => commit(types.CHECKOUT\_SUCCESS),

// handle failure

() => commit(types.CHECKOUT\_FAILURE, savedCartItems)

)

}

}

## Dispatching Actions in Components

metimport { mapActions } from 'vuex'

hods: {

...mapActions([

'increment', // map `this.increment()` to `this.$store.dispatch('increment')`

// `mapActions` also supports payloads:

'incrementBy' // map `this.incrementBy(amount)` to `this.$store.dispatch('incrementBy', amount)`

]),

...mapActions({

add: 'increment' // map `this.add()` to `this.$store.dispatch('increment')`

})

}

## Composing Actions

store.dispatch('actionA').then(() => {

// ...

})

actions: {

actionA ({ commit }) {

return new Promise((resolve, reject) => {

setTimeout(() => {

commit('someMutation')

resolve()

}, 1000)

})

}

}

--------------

actions: {

// ...

actionB ({ dispatch, commit }) {

return dispatch('actionA').then(() => {

commit('someOtherMutation')

})

}

}

actions: {

async actionA ({ commit }) {

commit('gotData', await getData())

},

async actionB ({ dispatch, commit }) {

await dispatch('actionA') // wait for `actionA` to finish

commit('gotOtherData', await getOtherData())

}

}

**Modules**

const moduleA = {

state: () => ({ ... }),

mutations: { ... },

actions: { ... },

getters: { ... }

}

const moduleB = {

state: () => ({ ... }),

mutations: { ... },

actions: { ... }

}

const store = new Vuex.Store({

modules: {

a: moduleA,

b: moduleB

}

})

store.state.a // -> `moduleA`'s state

store.state.b // -> `moduleB`'s state

## Module Local State ???