IoT swarm implementation !

Πίνακας περιεχομένων

1. Intro

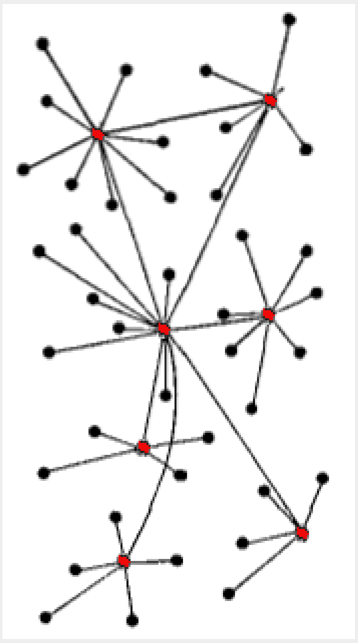
Intro IoT Swarm

We will be trying to create a swarm implementation that will allow communication between all of the members/nodes.

Imaging a swarm

To undertand this better lets look at the picture bellow and imagine that red dots are iot devices that can send and receive and black ones are clients that gather data.

Architecture of swarm communication



- Red Node: Sensor Node and Gateway Role
- Black and Red Node: Sensor Node - Client

To make our life easier at this task we will be using the following tools...

2. Prepare installation

2.1. Install docker

Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers.

Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels

2.2. Clone IoT swarm implementation example

```
git clone https://git.swarmlab.io:3000/zeus/iot-swarm-example.git
cd iot-swarm-example
```

2.2.1. Webclient tree

base dir Webclient

```
src/browser-client/src/
├── App.vue  ①
├── assets
│   ├── css
│   │   └── themify-icons.css
│   ├── fonts
│   │   ├── glyphicons-halflings-regular.448c34a.woff2
│   │   ├── glyphicons-halflings-regular.e18bbf6.ttf
│   │   ├── glyphicons-halflings-regular.f4769f9.eot
│   │   ├── glyphicons-halflings-regular.fa27723.woff
│   │   ├── themify.eot
│   │   ├── themify.svg
│   │   ├── themify.ttf
│   │   └── themify.woff
│   └── logo.png
├── components   ④
│   ├── doclive
│   │   ├── AdhocView.vue
│   │   └── runLlo.vue
│   └── DocLive.vue
├── main.js   ②
└── store
    ├── index.js
    └── modules
        └── create_pipelineLLO.js ③
```

① load App

② App config

③ Vuex and Rest calls

④ Components

2.2.2. IoT server tree

base dir IoT server

```
src/IoT/llo/
├─── bclient.js
├─── client.js
├─── iotclient.js ②
└─── iotserver.js ①
```

① IoT server

② IoT client

2.3. Control services

2.3.1. start IoT server

```
cd iot-swarm-example
./start-iotserver.sh
```

2.3.2. stop IoT server

```
cd iot-swarm-example
./stop-iotserver.sh
```

2.3.3. start IoT client

```
cd iot-swarm-example
./start-iotclient.sh
```

2.3.4. stop IoT client

```
cd iot-swarm-example
./stop-iotclient.sh
```

2.3.5. start IoT client-n

```
cd iot-swarm-example
./start-iotclient-n.sh
```

2.3.6. stop IoT client-n

```
cd iot-swarm-example
./stop-iotclient-n.sh
```

2.3.7. start IoT webclient

```
cd iot-swarm-example
./start-iotwebclient.sh
```

2.3.8. stop IoT webclient

```
cd iot-swarm-example
./stop-iotwebclient.sh
```

2.3.9. stop All services

```
cd iot-swarm-example
./stop-all.sh
```

2.4. Use webclient

open in Browser: http://localhost:8080

and

open Web Developer with ctrl+shift+K

2.4.1. send data in request body with GET

type text in *"Get iot Data"* and klick on it

See *"action"* in

- **Console** in browser
- and in Linux **Terminal**

2.4.2. Using Sockets to send and receive data

type text in editor box

See *"action"* in

- **Console** in browser
- and in Linux **Terminal**

3. Technologies

3.1. MVC

A core principle of the MVC pattern is the view layer’s ignorance with respect to the model layer. Views are dumb objects. They only know how to present data to the user. They don’t know or understand what they are presenting.

MVC: Division across three code components only: Model, View, and Controller. … Microservices: An app is divided into a set of specialized classes that interact with each other using APIs. This model is being used by companies like Netflix, Spotify, and eBay.

Model: This part manages the data on your site. Its role is to retrieve the raw information from the database, organize, and assemble it so that it can be processed by the controller.

View: This part focuses on the display. It is here where the data recovered by the model will be presented to the user.

Controller: This part manages the logic of the code and makes decisions. When the user interacts with the view, the request is processed by the controller.

It waits for the user to interact with the view to retrieve the request. Thus, it is the controller that will define the display logic, and display the next view on the screen.

3.2. Microservices Architecture

Microservices can be defined as an improvement, a kind of refinement, of what we know as service-oriented architecture (SOA).

In this architecture, a large application is made in the form of small monofunctional modules. Each microservice is autonomous.

Microservices do not share a data layer. Each has its own database and load balancer. So that each of these services can be deployed, adjusted, and redeployed individually without jeopardizing the integrity of an application.

As a result, you will only need to change a couple self-contained services instead of having to redeploy the entire application.

4. Software

4.1. Client site (PC)

4.1.1. Vue

> *vuejs*
>
> Vue.js is an open-source, progressive JavaScript framework for building user interfaces (UIs) and single-page applications.

Library modularization using a framework is common in frontend development.

What differentiates Vue.js from other alternatives is:

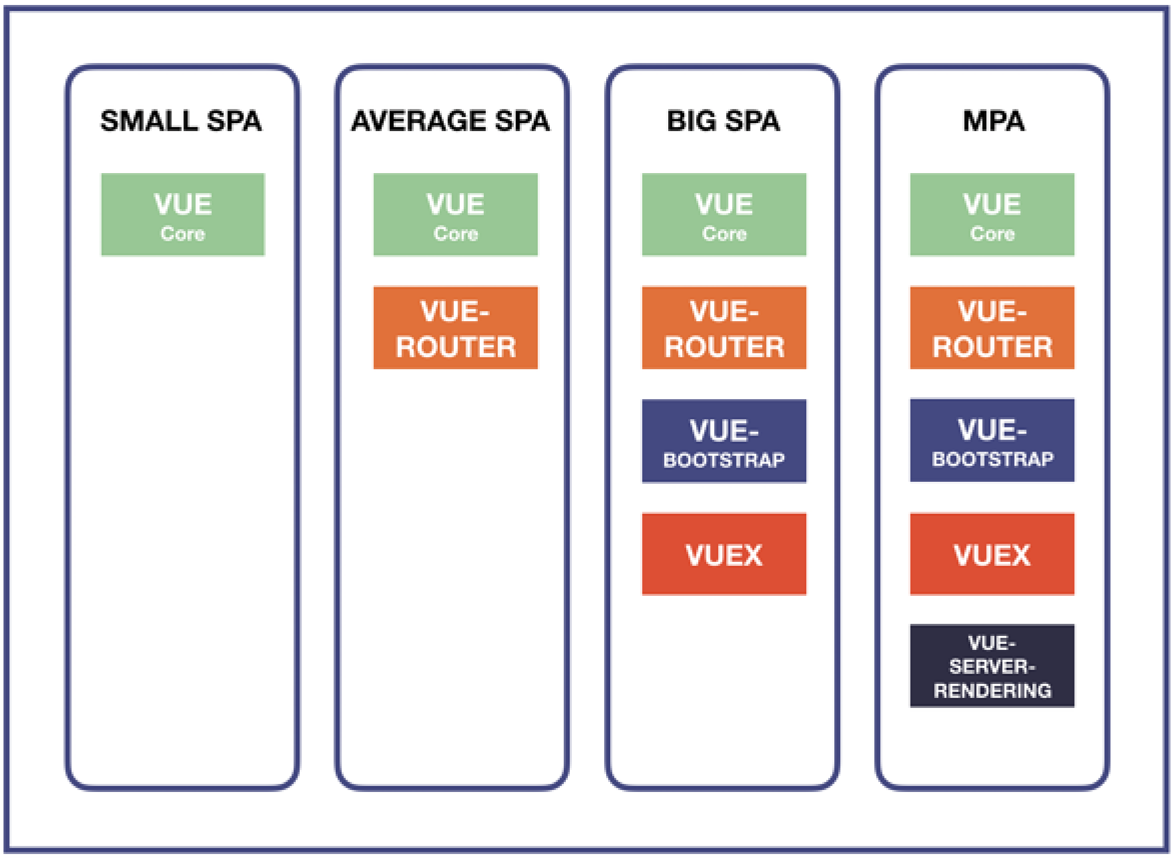
- its **“high decoupling”**, how easy it is to extend functionalities, and how well all parts work once more modules are included.

For example, if we want to organize and render small visual components, all we need is Vue.js’s ‘core’ library. It is not necessary to include additional libraries.

As the application grows,

- we have libraries to manage **routes** such as **‘vue-router’**,
- libraries to manage the global state such as **‘vuex’**
- and libraries to build responsive web applications such as **‘bootstrap-vue’**.
- Additionally, if our application needs to be optimized or needs good SEO, we can include the **‘vue-server-rendering’** library.

In the following figure, we can see how the libraries we just mentioned are progressively included, from a small SPA to multi-page applications (MPA).



The name of the framework – Vue – is the same phonetically in English as view, and it corresponds to the traditional Model-View-Controller (MVC) architecture

React and Angular are other Frameworks similar to vuejs

4.1.2. Vuex

Vuex is a state management pattern + library for Vue.js applications.

- It serves as a centralized store for all the components in an application, with rules ensuring that the state can only be mutated in a predictable fashion.
- It also integrates with Vue's official devtools extension to provide advanced features such as zero-config time-travel debugging and state snapshot export / import.

What is a "State Management Pattern"?

Let’s start with a simple Vue counter app:

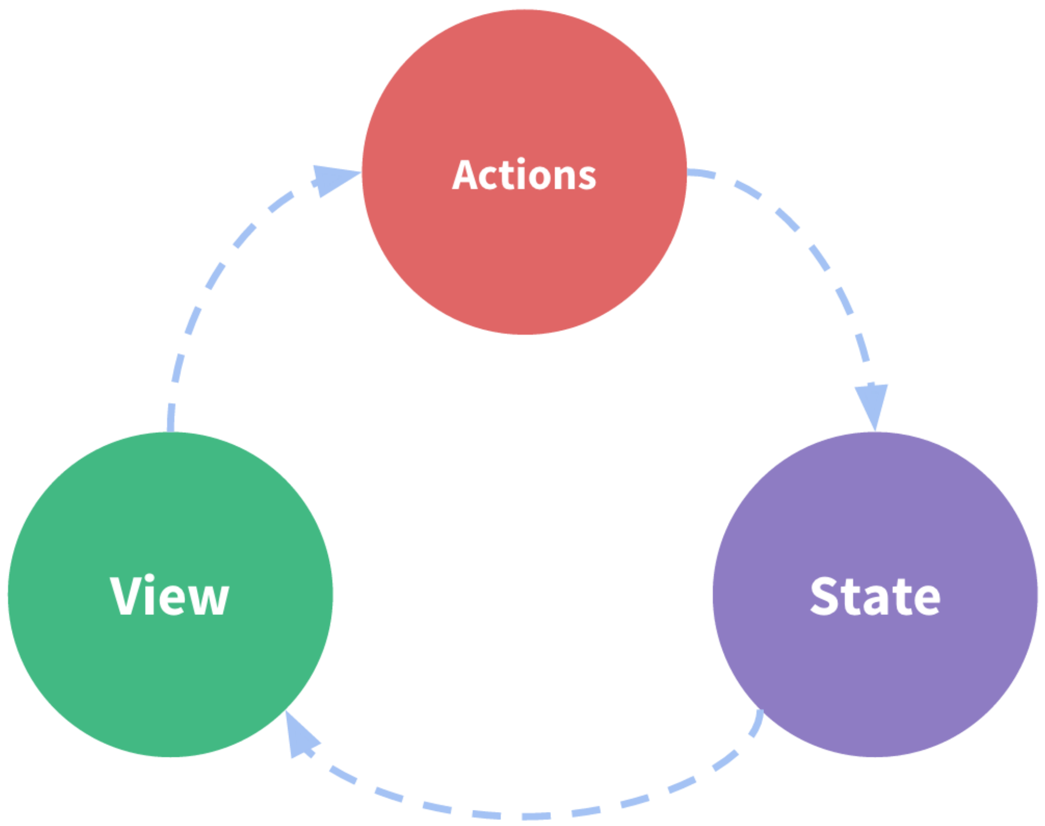
```
new Vue({
  // state
  data () {   ①
    return {
      count: 0
    }
  },
  // view
  template: `    ②
    <div>{{ count }}</div>
  `,
  // actions
  methods: {   ③
    increment () {
      this.count++
    }
  }
})
```

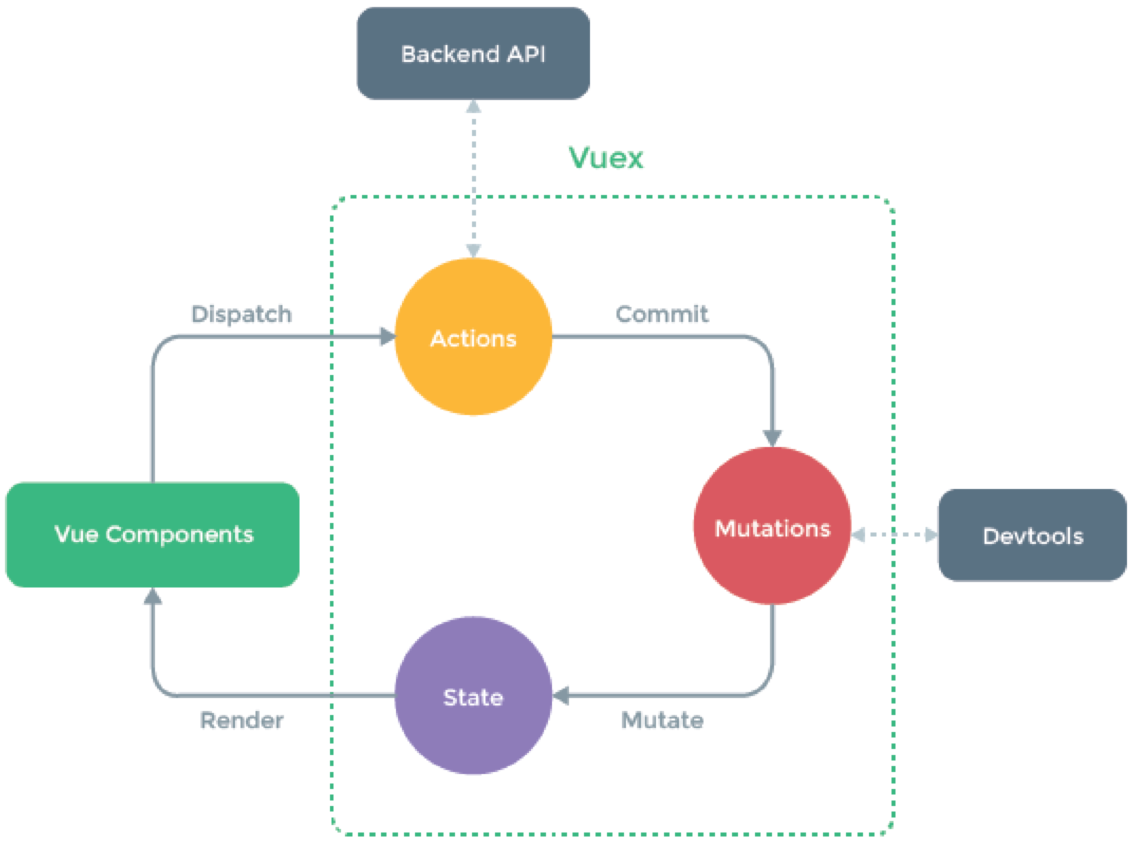
① The state, the source of truth that drives our app;

② The view, a declarative mapping of the state;

③ The actions, the possible ways the state could change in reaction to user inputs from the view.

This is a simple representation of the concept of "one-way data flow":


Actions
State
View


Backend API
Vuex
Dispatch
Actions
Commit
Vue Components
Mutations
Devtools
Render
State
Mutate

Online Vuex cources

4.1.3. Using Axios to Consume APIs

Axios is a library for http communication, making ajax requests, and so on.

See more

4.1.4. Using socket.io to Consume Websocket

Socket.IO aims to make realtime apps possible in every browser and mobile device, blurring the differences between the different transport mechanisms. It supports multiple transports, such as WebSockets, Flash sockets, long polling and more, automatically falling back when a transport fails

See more info here: Socket.io

Vue Packages

- socket.io client
- socket.io

4.2. Client/Server site (IoT device)

4.2.1. Nodejs

As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications.

Almost no function in Node.js directly performs I/O, so the process never blocks. Thanks to this, scalable systems is very reasonable to be developed in Node.js.

Overview of Blocking vs Non-Blocking

Node.js is similar in design to, and influenced by, systems like **Ruby's Event Machine** and **Python's Twisted.**

Node.js takes the event model a bit further. It presents an **event loop as a runtime construct** instead of a library.

In other systems, there is always a blocking call to start the event-loop.

Typically,

- behavior is defined through callbacks at the beginning of a script,
- and at the end a server is started through a blocking call like **EventMachine::run().**

In Node.js, there is no such start-the-event-loop call.

- **Node.js** simply **enters the event loop after executing the input script.**
- **Node.js exits the event loop** when there are **no more callbacks to perform.**

Node.js being **designed without threads** doesn't mean you can't take advantage of multiple cores in your environment.

Child processes can be spawned by using our **child_process.fork() API,** and are designed to be easy to communicate with.

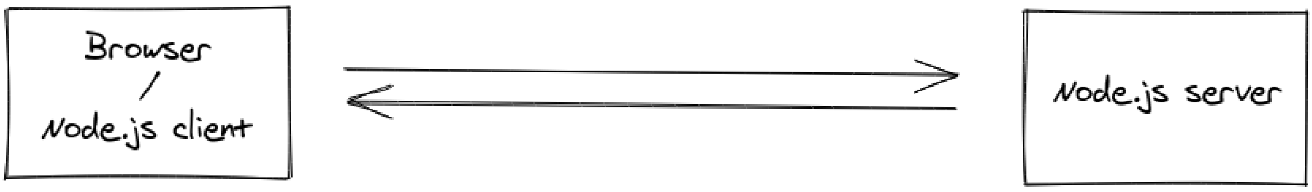
Built upon that same interface is the cluster module, which **allows you to share sockets between processes** to enable load balancing over your cores.

4.2.2. socket.io

Socket.IO is a library that enables **real-time**, **bidirectional** and **event-based** communication between the browser and the server.

It consists of:

- a Node.js server: Source | API
- a Javascript client library for the browser (which can be also run from Node.js): Source | API



See more info here: Socket.io

4.2.3. express

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

Fast, unopinionated, minimalist web framework for Node.js

4.2.4. NoSQL

A NoSQL (originally referring to "non-SQL" or "non-relational") database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.

More info

MongoDB is a general purpose, document-based, distributed database built for modern application developers and for the cloud -

5. Example Application

5.1. Client site

5.1.1. main (config)

```
import Vue from 'vue'
import App from './App.vue'
import store from '@/store/index'
import BootstrapVue from 'bootstrap-vue'
import 'bootstrap/dist/css/bootstrap.css'
import 'bootstrap-vue/dist/bootstrap-vue.css'
import VueSweetalert2 from 'vue-sweetalert2';
Vue.use(VueSweetalert2);

import VueSocketIOExt from 'vue-socket.io-extended'; ①
import io from 'socket.io-client';  ②

const socket = io('http://localhost:8084', { ③
  autoConnect: false
});

Vue.use(VueSocketIOExt, socket);



Vue.use(BootstrapVue);

Vue.config.productionTip = false
new Vue({
  el: '#app',
  store,
  render: h => h(App)
})
```

① import socket package

② import socket package

③ Connect to socket server

5.1.2. App (load)

```
<template>
  <div id="app">
                         ①
                         ...
  <div>
</template>
<script>
import DocLive from './components/DocLive.vue' ②
import "@/assets/css/themify-icons.css";


export default {
  name: 'app',
  components: {
  },
  data() {
    return {
      show: true,
      SwarmabAsciiLabTemplate:''
    }
  },
    mounted() {
  },
  methods: {
    saveDocLive() {
      this.$root.$emit('asciilive_save','save') ③
    },
  }
}
</script>
<style>
</style>
```

① your html

② import module

③ Send Message to other components

5.1.3. Create compoment

```
<template>
  <div id="app">
                         ①

     ...
    <div class="row" >

                <div :class="columnview">          ⑤
                  <run-llo
                    style="background-color: #f8f9fa"
                  >
                  </run-llo>
                </div>

                <div :class="columncode">   ⑤
                  <ad-hoc
                    style="background-color: #f8f9fa"
                  >
                  </ad-hoc>
                </div>
    </div>
  <div>
</template>
<script>
import RunLlo from "./doclive/runLlo.vue"; ②
import AdHoc from "./doclive/AdhocView.vue";

export default {
  name: 'DocLive',   ③
  props: {
  },
  components: {
    RunLlo,       ④
    AdHoc         ④
  },
  data () {
      return {
        loading: false,
        showhistory : 0,
        productIndex: 1,
        showmenou: 1,
        columnviewdefault : 0,
        columncodedefault : 0,
        columnview : 'col-7 order-first',
        columncode : 'col-5 order-last',
        tutorMenou: 'student'
      }
    },
      created: function () {
    },
  mounted() {
      this.$root.$on('LLOshowmenounotebooks', () => {
        this.showmenou = 1
      }),
    this.$root.$on('lloshowchallengehistory', (llo,active) => {
      this.showhistory = 1
    })
    },
   beforeDestroy () {
    this.$root.$off('LLOshowmenounotebooks'),
    this.$root.$off('lloshowchallengehistory') // working
   },
 methods: {                      ⑥
         fullscreen(action){     ⑦
          if(action == 'max'){
              this.columnview = 'col-11 order-first'
              this.columncode = 'col-1 order-last'
              this.columnviewdefault = 1
              this.columncodedefault = 0
              //set height iframe
              this.$root.$emit('LLOresizemenounotebooks','max')
              console.log(this.columnview)
           }
           else if(action == 'min'){
              this.columnview = 'col-7 order-first'
              this.columncode = 'col-5 order-last'
              this.columnviewdefault = 0
              this.columncodedefault = 1
              //set height iframe
              this.$root.$emit('LLOresizemenounotebooks','min')
              console.log(this.columnview)
           }
           else if(action == 'codemax'){
              this.columnview = 'col-1 order-first'
              this.columncode = 'col-11 order-last'
              this.columnviewdefault = 0
              this.columncodedefault = 1
              console.log(this.columnview)
           }
           else if(action == 'codemin'){
              this.columnview = 'col-7 order-first'
              this.columncode = 'col-5 order-last'
              this.columnviewdefault = 1
              this.columncodedefault = 0
              this.$root.$emit('LLOresizemenounotebooks','min')
              console.log(this.columnview)
           }
         },
          async onAction (action) { ⑧
              this.tutorMenou='tutor'
              //this.tutorMenou='student'
          }
    }


}

</script>
<style>
</style>
```

① your html

② import your components help files

③ Export your component

④ your component files

⑤ insert your component

⑥ your methods

⑦ method to controll window behavior

⑧ on action

5.1.4. Use your component

```
<template>
<div>

    <!--  menou -->
    <div class="row"
      v-show="showmenou == 1"
    >

...
    </div>
    <!-- menou -->



    <div class="row" >

                <div :class="columnview">
                  <run-llo                          ①
                    style="background-color: #f8f9fa"
                  >
                  </run-llo>
                </div>

                <div :class="columncode">
                  <ad-hoc                          ①
                    style="background-color: #f8f9fa"
                  >
                  </ad-hoc>
                </div>
    </div>

 </div>
</template>

<script>
import RunLlo from "./doclive/runLlo.vue";       ②
import AdHoc from "./doclive/AdhocView.vue";     ②
```

```
export default {
  props: {
  },
  components: {
    RunLlo,   ②
    AdHoc     ②
  },
  data () {
      return {
        loading: false,
        showhistory : 0,
        productIndex: 1,
        showmenou: 1,
        columnviewdefault : 0,
        columncodedefault : 0,
        columnview : 'col-7 order-first',
        columncode : 'col-5 order-last',
        tutorMenou: 'student'
      }
    },
      created: function () {
    },
  mounted() {
      this.$root.$on('LLOshowmenounotebooks', () => {
        this.showmenou = 1
      }),
    this.$root.$on('lloshowchallengehistory', (llo,active) => {
      this.showhistory = 1
    })
    },
   beforeDestroy () {
    this.$root.$off('LLOshowmenounotebooks'),
    this.$root.$off('lloshowchallengehistory') // working
   },
 methods: {
 ...
    }


}
</script>

<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>
</style>
```

① use component

② import components

5.1.5. Sockets

```
<template>
  <div>
  ...
  </div>
</template>
<script>
import Vue from 'vue'     ①
import JSZip from 'jszip'; ②
import FileSaver from 'file-saver'; ③
export default {
   components: {
   },
   data () {
    return {
      showlloedit:true,
      issocket:'close',
      socketdata:'',
      code:[],
        iotdata: [  ④
                    {
                        "iot1": {
                            "id": 1,
                            "name": "auto1",
                            "img": "rinse.png",
                            "Servicios": [
                                {"model":"Sentra", "doors":4},
                                {"model":"Maxima", "doors":4},
                                {"model":"Skyline", "doors":2}
                            ]
                        },
                     },{
                        "iot2": {
                            "id": 2,
                            "name": "auto2",
                            "img": "shirt-2.png",
                            "Servicios": [
                                {"model":"Sentra", "doors":4},
                                {"model":"Maxima", "doors":4},
                                {"model":"Skyline", "doors":2}
                            ]
                        },
                     },{
                        "iot3": {
                            "id": 3,
                            "name": "auto3",
                            "img": "iron.png",
                            "Servicios": [
                                {"model":"Sentra", "doors":4},
                                {"model":"Maxima", "doors":4},
                                {"model":"Skyline", "doors":2}
                            ]
                        },
                     },{
                          "iot4": {
                            "id": 4,
                            "name": "auto4",
                            "img": "wring.png",
                            "Servicios": [
                                {"model":"Sentra", "doors":4},
                                {"model":"Maxima", "doors":4},
                                {"model":"Skyline", "doors":2}
                            ]
                        }
                    }
                ]
   }
  },
  methods: {
   async addiot() {
      console.log(JSON.stringify(this.iotdata))
        },
   async socketopen () {
        this.$socket.client.open();     ⑤
   },
/**
 *
 * == socketclose()
 *
 * [source,javascript]
 * ----
 *       this.$socket.client.close();
 * ----
 *
 */
   async socketclose () { ⑥
        this.$socket.client.close();
   }

  },
  computed: {
    },
/**
 *
 * == Socket subscribe
     *
 * [source,javascript]
 * ----
 *            this.$root.$on('iot_add', (v) => {  ①
 *          ...
 *          })
     * sdfsf
     den to perni  sdfsf
 * ----
 * <1> EventBus is used for parent/child component communication.
 *
 */
    mounted() {
        this.$root.$on('iot_add', (v) => {
              this.iotdata.push(v);
              //this.iotdata = v
              this.addiot()
      this.$socket.client.emit('subscribe', 'iot');
        })
        this.$root.$on('socket_add', (v) => {
              this.socketdata = v;
              console.log('socket_add ' + JSON.stringify(v))
              this.$socket.client.emit('log', this.socketdata);
        })
    },
/**
 *
 * == Destroy EventBus
 *
 * See
 * https://www.digitalocean.com/community/tutorials/vuejs-component-lifecycle[Vue.js
Lifecycle Hooks^].
 *
 * *beforeDestroy*
 *
 * - beforeDestroy is fired right before teardown. Your component will still be fully
present and functional.
 *
 * [source,javascript]
 * ----
 *  this.$root.$off('iot_add') ①
 * ----
 * <1> EventBus is used for parent/child component communication.
 *
 */
  beforeDestroy () {
        this.$root.$off('iot_add')
        this.$root.$off('socket_add')
 },
 /**
 *
 * == Open a socket
 *
 * See
 * https://www.digitalocean.com/community/tutorials/vuejs-component-lifecycle[  Vue.js
Lifecycle Hooks^]
```

```
 *
 * *Created*
 *
 * - You are able to access reactive data and events that are active with the created
hook. Templates and Virtual DOM have not yet been mounted or rendered:
 *
 * [source,javascript]
 * ----
 *   this.socketopen()
 * ----
 *
 */


       created () {
     this.socketopen()
       },
 /**
 *
 * == Socket events
 *
 * [source,javascript]
 * ----
 *   this.$socket.client.emit('authenticate', 'logintoken');
 * ----
 *
 */
    sockets: {
     connect() {
       this.$socket.client.emit('authenticate', 'logintoken');

       this.$socket.client.emit('socket_id_get', 'socketdatasend');
       console.log('socket connected '+ 'socketdatasend' )
       this.issocket = 'open'
     },
 /**
 *
 * === onError
 *
 */
     error(error) {
       console.log("socket error "+JSON.stringify(error))
       this.issocket = 'close'
     },
 /**
 *
 * === connect_error
 *
 */
     connect_error(error) {
       console.log("socket connect_error "+JSON.stringify(error))
       this.issocket = 'close'
     },
 /**
 *
 * === connect_error
 *
 */
     disconnect(reason) {
       console.log("socket disconnect "+JSON.stringify(reason))
       this.issocket = 'close'
     },
 /**
 *
 * === Socket connect_timeout
 *
 */
     connect_timeout(reason) {
       console.log("socket timeout "+JSON.stringify(reason))
       this.issocket = 'close'
     },
 /**
 *
 * === Socket reconnect
 *
 */
     reconnect(attemptNumber) {
       console.log("socket reconnect attemptNumber "+JSON.stringify(attemptNumber))
     },
 /**
 *
 * === connect_attempt
 *
 */
     reconnect_attempt(attemptNumber) {
       console.log("socket reconnect_attempt "+JSON.stringify(attemptNumber))
     },
 /**
 *
 * === Socket reconnecting
 *
 */
     reconnecting(attemptNumber) {
       console.log("socket reconnecting "+JSON.stringify(attemptNumber))
     },
 /**
 *
 * === reconnect_error
 *
 */
     reconnect_error(error) {
       console.log("socket reconnect_error "+JSON.stringify(error))
       this.issocket = 'close'
```

```
      },
  /**
  *
  * === unauthorized
  *
  */
      unauthorized(val) {
        console.log("socket unauthorized "+JSON.stringify(val))
        this.issocket = 'close'
      },
  /**
  *
  * === connected
  *
  */
      socket_id_emit(val) {
        console.log("socket id from server "+JSON.stringify(val))
        console.log("socket id from serveri saved "+JSON.stringify(socketsave))
        this.issocket = 'open'
      },
  /**
  *
  * === Socket onMessage
  *
  */
      async adhocEmit(val) {
        console.log("socket from server "+JSON.stringify(val))
        this.issocket = 'open'
            this.$wait.start('myRunInstance1');
                // render begin
                this.tryLLO = 'on'
                if(this.firstbootstrap === 0 ){
                    await this.bootsrapllo();
                    this.firstbootstrap = 1
                }
                var output = log.data.out
                var mydinfunction = `
                  <div class="row">
                      <b-col  class="" cols="12" sm="12"  md="12" >
                            ${output}
                       </b-col>
                  </div>`

                  try {
                     let divascii = document.createElement('div');
                     divascii.setAttribute("class", "container-fluid w-100 p-3
llotry")
                     divascii.innerHTML = mydinfunction
                     this.addtask(divascii);
                  }catch (ex) {
                      console.log(" logi error1 "+JSON.stringify(ex))
                    return
                }
            this.$wait.end('myRunInstance1');
        },
      async iotdata(val) {
          console.log(" socket from iotdata "+JSON.stringify(val))
        },
      async message(val) {
          console.log(" socket message "+JSON.stringify(val))
        }
    }

};
</script>

<style>
.CodeMirror {


    font-family: monospace;
    height: 750px;
}

</style>
```

① import module

② import module

③ import module

④ json examples coming from IoT device

⑤ open socket

⑥ close socket

See also See Vue.js Lifecycle Hooks

5.1.6. create store

```
import { mapState, mapActions, commit } from 'vuex'  ①
import store from '@/store/index'
import axios from 'axios' ②

export default {
  namespaced: true,   ③
  state: {            ④
    llo: {},
    socketid:''
  },
  getters: {          ⑤
        getllosrc (state, container) {
```

```
       //console.log("js1 get "+JSON.stringify(state.llo))
           return state.llo
       },
       getsocketid (state, container) {
       //console.log("js1 get "+JSON.stringify(state.llo))
           return state.socketid
       }
 },
 mutations: {      ⑥
           setllo (state, data) {
       //console.log("js1 set "+JSON.stringify(data))
               state.llo=data;
       },
           setsocketid (state, data) {
       //console.log("js1 set "+JSON.stringify(data))
               state.socketid = data;
       }
 },
 actions: {    ⑦
       async get_data({commit,rootGetters}, value) {
           try {

       let p = await axios.get("http://localhost:8084/run", {   ⑧
         timeout: 45000,
         params: {
           code:  value.code
         }
       });

       //var p = value.code
       console.log("paramp "+JSON.stringify(p))
               store.dispatch('pipelineLLO/setScriptllo', p)
               return p;

           } catch (e) {
               if(e.error == "invalid_token"){
                   window.location.href = 'https://api-login.swarmlab.io:8089';
               }else{
                 var R = {
           ERROR_str: e,
           ERROR: 'yes'
         }
                 return R;
       }
           }
       },
       setScriptllo({commit}, value) {
               //console.log("container "+value)
           commit('setllo', value)
       },
       setsocketllo({commit}, value) {
```

```
                //console.log("container "+value)
            commit('setsocketid', value)
        },
        setScriptCodlogAction({commit}, value) {
                //console.log("container "+value)
            commit('setScriptCodelog', value)
        }
  }
}
```

① import vuex More info

② import axios

③ enable namespace

④ create state (store)

⑤ getters

⑥ setters

⑦ actions sync/async

⑧ Rest call

5.2. Server site

```
var path = require('path');   ①
var app = require('express')(); ①
var http = require('http').Server(app); ①
var io = require('socket.io')(http); ①

const socketAuth = require('socketio-auth'); ①


const cors = require('cors')   ②
const whitelist = [
        'http://localhost:3080',
        'http://localhost:3081',
        'http://localhost:3082'
        ]
const corsOptions = {
  credentials: true,
  methods: ['GET', 'PUT', 'POST', 'DELETE', 'OPTIONS'],
  optionsSuccessStatus: 200, // some legacy browsers (IE11, various SmartTVs) choke on
204
  allowedHeaders: [
        'Content-Type',
        'Authorization',
        'X-Requested-With',
        'device-remember-token',
        'Access-Control-Allow-Origin',
        'Access-Control-Allow-Headers',
        'Origin',
        'Accept'
  ],
  origin: function(origin, callback) {
    if (whitelist.indexOf(origin) !== -1) {
      callback(null, true)
    } else {
      callback(null, true)
    }
  }
}




app.get('/run', [   //  ③
    //check('access_token').isLength({ min: 40 }),
    //check('llo').isBase64()
  ],
cors(corsOptions),  (req, res, next) => { ④

  var RES = new Object();
  RES.code    = req.query["code"]  ⑤
      console.error('socket get from client' + RES.code);

  RES.error = false
  RES.error_msg = "ok"

      io.emit("iotdata", RES)      ⑥
      io.in('iot').emit('message', RES);  ⑦
  res.json(RES)

});

app.post('/run', [  ⑧
    //check('access_token').isLength({ min: 40 }),
    //check('llo').isBase64()
  ],
cors(corsOptions),  (req, res, next) => {
      console.error('socket post from client');
      io.emit("customEmit", 'data')

      var RES = new Object();
      RES.error = false
      RES.error_msg = "ok"
      res.json(RES)

});
```

```
io.on('connection', s => {  ⑨
      console.error('socket connection');
    var id = s.id
    s.on('pingServerEmit', obj => {
      console.error('socket.io pingServer');
      var data = obj+' testserver'
        io.emit("customEmit", data)
      io.in('iot').emit('message', data);
      console.error('from client '+ s.id + ' obj ' + obj);
    });

   s.on('subscribe', function(room) {  ⑩
        console.log('joining room', room);
        s.join(room);
    })

   io.on('unsubscribe', function(room) {
        console.log('leaving room', room);
        io.leave(room);
    })

   // when the client emits 'new message', this listens and executes
   s.on('log', (data, room) => {
      s.to('iot').emit('message', data);
      console.log('broadcast', data);

  });


});


http.listen(8084, () => console.error('listening on http://localhost:8084/'));
console.error('socket.io example');
```

① load modules

② config cors More info

③ create Get

④ use cors

⑤ get data

⑥ send with socket to all

⑦ send with socket to room

⑧ post

⑨ handle socket connections

⑩ subscribe

5.3. IoT Device

```
var path = require('path');
var app = require('express')();
var http = require('http').Server(app);
var io = require('socket.io')(http);

const socketAuth = require('socketio-auth');


const axios = require('axios');
axios.defaults.timeout = 30000

const helmet = require('helmet');


const cors = require('cors')
const whitelist = [
        'http://localhost:3080',
        'http://localhost:3081',
        'http://localhost:3082'
        ]
const corsOptions = {
  credentials: true,
  methods: ['GET', 'PUT', 'POST', 'DELETE', 'OPTIONS'],
  optionsSuccessStatus: 200, // some legacy browsers (IE11, various SmartTVs) choke on
204
  allowedHeaders: [
        'Content-Type',
        'Authorization',
        'X-Requested-With',
        'device-remember-token',
        'Access-Control-Allow-Origin',
        'Access-Control-Allow-Headers',
        'Origin',
        'Accept'
  ],
  origin: function(origin, callback) {
    if (whitelist.indexOf(origin) !== -1) {
      callback(null, true)
    } else {
      callback(null, true)
      //callback(new Error('Not allowed by CORS'))
    }
  }
}


app.use(helmet());
app.get('/run', [
    //check('access_token').isLength({ min: 40 }),
    //check('llo').isBase64()
  ],
cors(corsOptions),  (req, res, next) => {

  var RES = new Object();
  RES.tmp_token = req.query["access_token"]
  RES.datafile  = req.query["datafile"]
  RES.service   = req.query["service"]
  RES.action    = req.query["action"]



  RES.error = false
  RES.error_msg = "ok"
  res.json(RES)

});

app.post('/run', [
    //check('access_token').isLength({ min: 40 }),
    //check('llo').isBase64()
  ],
cors(corsOptions),  (req, res, next) => {
      console.error('socket post');
      io.emit("customEmit", 'data')

      var RES = new Object();
      RES.error = false
      RES.error_msg = "ok"
      res.json(RES)

});

socketoptions = {
        secure:true,
        reconnect: true,
        rejectUnauthorized : false
};


// Client
var io2 = require('socket.io-client');
var socket = io2.connect('http://localhost:8084', socketoptions);

var global = {}

socket.on('connection', s => {
    console.log(s)
      console.error('socket2 connection');
    global.id = s.id
    s.emit('log', 'client');

});

    var roomiot = 'iot'
    socket.emit('subscribe', roomiot);  ①

    socket.emit('log', 'client1');

    socket.on('message', function (data) { ②
      //console.log('from room ' + data);
      console.log("from room iot "+JSON.stringify(data))
      //io.emit("customEmit", data)
   });

io.on('connection', s => {
      console.error('socket connection');
    var roomiot = 'iot'
    var id = s.id
    s.on('pingServer', obj => {
      console.error('socket.io pingServer');
      //console.error(s);
      console.error('fromclient '+obj);
      var data = obj+' testfromclient '+id
        socket.emit("pingServerEmit", data)
        socket.emit('send', { room: roomiot, message: 'message iot' });
    });

    s.on('customEmit', obj => {
      var data = obj+' test customeEdit '+id
       //console.error('from server ' + data);
        //s.emit("pingServerEmit", data)
        socket.emit("customEmit", data)
    });


});


http.listen(3081, () => console.error('listening on http://localhost:3081/'));
console.error('socket.io example');
```

① subscribe

② event on message