Vue.Js Setup Part 2 :

**++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**

**links :** [**https://www.youtube.com/watch?v=MQZ4PQGw-sc&list=PL8p2I9GklV45qwTH-mdzllUuFRJO-euYn&index=13**](https://www.youtube.com/watch?v=MQZ4PQGw-sc&list=PL8p2I9GklV45qwTH-mdzllUuFRJO-euYn&index=13)

**>>>> props in vue js >>>**

**what is props >>** this is function i pass the parameter , like wise i pass props in templates.

if i pass the props in value in child components or siblling components.

props pass normally as one component to another component

but if i pass data based on name or userName then use to

**v-bind:name="userName"**

**fetch single name based >>>**

<template>

<div id="app">

<h2>Props in vue Js</h2>

<!-- <Child name="Sauarbh Shukla" /> -->

<Child v-bind:name="userName" />

</div>

</template>

<script>

import Child from './components/Child.vue'

export default {

name: 'App',

components: {

Child,

},

data()

{

return {

userName:"SAM"

}

}

}

</script>

<style>

#app {

font-family: Avenir, Helvetica, Arial, sans-serif;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

text-align: center;

color: #2c3e50;

margin-top: 60px;

}

</style>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

If i got array stored data then ...

Pass multiple data via parent to child >>>

**App.vue >>>**

<template>

<div id="app">

<h2>Props in Vue.js</h2>

<!-- Bind the first user's name to the Child component -->

<Child v-bind:name="users[0].name" />

<!-- If you want to loop through all users -->

<div v-for="(user, index) in users" :key="index">

<Child v-bind:name="user" />

</div>

</div>

</template>

<script>

import Child from './components/Child.vue';

export default {

name: 'App',

components: {

Child,

},

data() {

return {

users: [

{ name: 'Anil' ,email:"anil@gmail.com" },

{ name: 'Mukesh' , email:"mukesh@gmail.com"},

],

};

},

};

</script>

<style>

#app {

font-family: Avenir, Helvetica, Arial, sans-serif;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

text-align: center;

color: #2c3e50;

margin-top: 60px;

}

</style>

**Child.vue >>>**

<template>

<div>

<p>{{ name.name }}</p>

<p>{{ name.email }}</p>

</div>

</template>

<script>

export default {

name:"ChildPage",

props: ['name']

};

</script>

**++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**

**Send Props child to parent >>>>>**

**App.vue >>>**

<template>

<div id="app">

<h2>Send Props Child to Parent in Vue.js</h2>

**<h5>{{title}}</h5>**

**<Child v-on:changeTitle="updateTitle($event)" />**

</div>

</template>

<script>

import Child, {} from './components/Child.vue';

export default {

name: 'App',

components: {

Child,

},

**data() {**

**return {**

**title:"Props Title"**

**};**

**},**

**methods:{**

**updateTitle(title){**

**this.title=title**

**}**

**}**

};

</script>

<style>

#app {

font-family: Avenir, Helvetica, Arial, sans-serif;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

text-align: center;

color: #2c3e50;

margin-top: 60px;

}

</style>

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

**Child.vue >>>>**

<template>

<div>

**<button @click="updateTitle">Update Parent Title</button>**

</div>

</template>

<script>

export default {

name:"ChildPage",

**methods:{**

**updateTitle(){**

**this.$emit('changeTitle',"Parent Updated")**

**}**

**}**

}

</script>

child to send parent data use $emit function pass two parameter function and update value ..

in child components >>>>

in tempaltes >>>

**<button @click="updateTitle">Update Parent Title</button>**

in script >>>

**updateTitle(){**

**this.$emit('changeTitle',"Parent Updated")**

**}**

App.vue >>>>>

in tempates tag >>

<h5>{{title}}</h5>

<Child v-on:changeTitle="updateTitle($event)" />

**in script >>>**

data() {

return {

title:"Props Title"

};

},

methods:{

updateTitle(title){

this.title=title

}

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**life cycle methods (Hooks) >>>>**

**first run this process Digram Created >>>**

**new Vue() ===> (default working as main.js file)**

**|**

**init ( Events & Lifecycle )**

**beforeCreate ----------------------------- |**

**|**

**init ( injections & reactivity )**

**created ----------------------------- |**

**Has ( el ) Option? (el means external element, like main.js)**

**| ------- NO**

**Yes**

**|**

**Has "template" option?**

**| <------ vm.$mount(el) is called**

**---------- | ----------**

**Yes No**

**| |**

**Compile template into Compile el's outerHTML**

**render function \* as template \***

**|**

**beforeMount <-------- |**

**Create vm.$el and replace "el" with it**

**|**

**mounted -----------------------------> Component is now rendered and attached to the DOM**

**|**

**Watching data changes (Reactivity)**

**|**

**beforeUpdate <-------- When data changes**

**|**

**Re-render the virtual DOM**

**|**

**updated -----------------------------> Component is re-rendered with updated data**

**|**

**beforeDestroy -------------------------> The component is about to be destroyed**

**|**

**Destroying watchers, child components, event listeners, etc.**

**|**

**destroyed --------------------------> The component is completely destroyed**

Summary >>>

Key Notes:

**beforeCreate**: The lifecycle starts. No data, methods, or components are available yet.

**created**: Data, methods, and computed properties are available. However, the template is not compiled.

**beforeMount**: Before Vue inserts the compiled template into the DOM.

**mounted**: Vue inserts the template into the DOM and the component is visible.

**beforeUpdate**: If data changes, this hook is called before updating the DOM.

**updated**: Called after the data has been re-rendered and updated.

**beforeDestroy**: Triggered before the component is destroyed.

**destroyed**: The component is fully destroyed and detached from the DOM.

code >>

Child.vue >>

<template>

<div>

<h5>This is child component</h5>

</div>

</template>

<script>

export default {

name:"ChildPage",

methods: {

},

data(){

return {}

},

beforeCreate:function(){

console.log("beforeCreate");

},

beforeMount:function(){

console.log("beforeMount");

},

mounted:function(){

console.log("mounted");

},

created:function(){

console.log("created");

}

};

</script>

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**Form with checkbox and radio button >>>>**

**Create one Singup From component ( now add and call in app.vue file )**

**SingupFrom >>>>>**

<template>

<div>

<h3>Create SingUp From</h3>

**<input type="text" name="username" id="username" placeholder="Enter Your Name" v-model="singup.username"/> <br/><br/>**

**<input type="password" name="password" id="password" placeholder="Enter Your Password" v-model="singup.password"/> <br/><br/>**

**<p>Choose Hobbies</p>**

**<input type="checkbox" id="cricket" value="cricket" v-model="singup.hobbies"/>**

**<label for="cricket">Cricket</label>**

**<input type="checkbox" id="football" value="football" v-model="singup.hobbies"/>**

**<label for="cricket">FootBall</label><br/><br/>**

**<p>Select Gender</p>**

**<input type="radio" name="gender" id="male" value="male" v-model="singup.gender"/>**

**<label for="male">Male</label>**

**<input type="radio" name="gender" id="female" value="female" v-model="singup.gender"/>**

**<label for="female">Female</label>**

**<br/><br/>**

**<button @click="singupUser">Submit</button>**

</div>

</template>

<script>

export default {

name:"SingUpPage",

data(){

return{

**singup : {**

**username:null,**

**password:null,**

**hobbies:[],**

**gender:null**

**}**

}

},

methods:{

**singupUser()**

**{**

**console.log(this.singup)**

**}**

}

}

</script>

**Notes ....**

Getting all data while click on submit button getting data while click on submit button ...

I am found all data when user fill form and submit data ..

**++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**

**Make a form and get data >>>>>**

**Login.vue >>>>**

<template>

<div>

<h3>Login Page</h3>

<input v-model="loginForm.username" type="text" id="username" name="username" placeholder="Enter UserName"/><br/><br/>

<input v-model="loginForm.password" type="password" id="password" name="password" placeholder="Enter Password"/><br/><br/>

<button @click="loginData">Login</button>

</div>

</template>

<script>

export default {

name:"LoginPage",

methods:{

loginData(){

console.log("loginData",this.loginForm.username,this.loginForm.password);

}

},data(){

return {

loginForm:{

username:null,

password:null

}

}

}

}

</script>

create login form and get data while i am submit data login form i am getting data in console.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**form validation >>>>**

<template>

<div>

<h3>Login Page</h3>

<p v-if="error.length">

<b>Please correct following errors</b>

<ul>

<li v-for="em in error" :key="em.id">

{{ em }}

</li>

</ul>

</p>

<form @submit="login">

<input v-model="name" type="text" id="name" name="name" placeholder="Enter UserName"/><br/><br/>

<input v-model="password" type="password" id="password" name="password" placeholder="Enter Password"/><br/><br/>

<button @click="loginData">Login</button>

</form>

</div>

</template>

<script>

export default {

name:"LoginPage",

data(){

return{

error:[],

name:null,

password:null

}

},

methods:{

login(e){

this.error=[];

if(this.name && this.password){

console.log("no error");

return true;

}

if(!this.name){

this.error.push("User Name is required");

}

if(!this.password){

this.error.push("Password is required");

}

console.log(this.error);

e.preventDefault();

}

},

}

</script>

**validation completed for vue js ..**

**++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**

**Filters in vue Js >>>**

**Main.js >>>**

import { createApp } from 'vue'

import App from './App.vue'

**// Create the Vue app instance**

**const app = createApp(App)**

**// Define the global "filter-like" function**

**app.config.globalProperties.$filters = {**

**capitalize(val) {**

**if (!val) return ''**

**return val.toString().toUpperCase();**

**}**

**}**

// Mount the app

app.mount('#app')

**App.vue >>>>**

<template>

<div id="app">

<h2>Filters in Vue.js</h2>

**<p>{{$filters.capitalize(name)}}</p>**

<Home/>

</div>

</template>

<script>

import Home from './components/Home.vue'

export default {

name: 'App',

components: {

Home,

},

data(){

return {

name:"Saurabh Shukla",

}

}

};

</script>

<style>

#app {

font-family: Avenir, Helvetica, Arial, sans-serif;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

text-align: center;

color: #2c3e50;

margin-top: 60px;

}

</style>

**Home.js >>>**

<template>

<div>

<!-- Using the global filter-like function -->

**<p>{{ $filters.capitalize("hello world") }}</p>**

</div>

</template>

<script>

export default {

name:"homePage",

}

</script>

**use global filter function in main.js file after you call any where.**

**++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**

**beforecreate and created life cycle methods >>>>**

* when component is load on screen then beforecreate hook is called .
* when observe data & call event then created hook is called.

when beforeCreated hook called mean while component is not created.

when load component then created hook is working accordinglly.

code like this >>>

**Home.vue >>**

<template>

<div>

<h3>Home</h3>

**<p>{{ name }}</p>**

</div>

</template>

<script>

export default {

name:"homePage",

data(){

return{

name:"Saurabh Shukla"

}

},

**beforeCreate(){**

**// this.name = "Sam"; // but its not display because its when data is not loaded then display.**

**console.log("before create",this.name);**

**}**

**,created(){**

**this.name = "Sam"; // it is display because when data is render first after created is hook is called then show created property it is overwrite the data property value.**

**console.log("created",this.name);**

**}**

}

</script>

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

beforeMount & mounted hooks life cycle in vue >>>>

beforeMount >>> before html is render then before mount hooks.

mounted >>> after render html then called mounted hooks.

**Home.vue >>>**

<template>

<div>

<h3>Home</h3>

<p>{{ name }}</p>

</div>

</template>

<script>

export default {

name:"homePage",

data(){

return{

name:"Saurabh Shukla"

}

},

beforeMount(){

// its not display because its when before Html render

console.log("before mounted",this.$el); // el means ( html element define )

},

mounted(){

// its display because its when Html render

console.log("mounted",this.$el); // el means ( html element define )

}

}

</script>

**++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**

**beforeDestroy and destroyed life cycle methods >>>>**

**Before** Dom is Destroy then **beforeDestroy hooks is works.**

after Dom is Destroy then **Destroy hooks is works**

Ex..

**App.vue >>>**

<template>

<div id="app">

<h2>beforeDestroy and destroyed life cycle methods</h2>

**<div v-if="display">**

**<Home/>**

**</div>**

**<button @click="toggle">Toggle</button>**

</div>

</template>

<script>

import Home from './components/Home.vue'

export default {

name: 'App',

components: {

Home,

},

**data(){**

**return {display:true};**

**},**

**methods:{**

**toggle(){**

**this.display=!this.display;**

**}**

**}**

};

</script>

<style>

#app {

font-family: Avenir, Helvetica, Arial, sans-serif;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

text-align: center;

color: #2c3e50;

margin-top: 60px;

}

</style>

**Home.vue >>>>**

<template>

<div>

<h3>Home</h3>

<p>{{ name }}</p>

</div>

</template>

<script>

export default {

name:"homePage",

data(){

return{

name:"Saurabh Shukla"

}

},

// old vue 2 function hooks

// beforeDestroy()

// {

// // its not display because its when before Dom Removed

// console.log("before destroy");

// },

// destroyed()

// {

// // its display because after Dom Removed

// console.log("destroyed");

// }

// new vue 3 function hooks

beforeUnmount() {

// Logic before component is unmounted

console.log("before Unmount");

},

unmounted() {

// Logic after component is unmounted

console.log("before Unmounted");

}

}

</script>

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

now **using v-if="toggle" then on get condition is correct id data after send data & method while on click button then its then its show hide & show.**

**Now use beforeDestroy & Destroyed hooks code >>>**

**Old Vue 2 Code (using beforeDestroy and destroyed): >>>>>**

export default {

beforeDestroy() {

// Logic before component is destroyed

},

destroyed() {

// Logic after component is destroyed

}

}

**Updated Vue 3 Code (using beforeUnmount and unmounted): >>>>**

export default {

beforeUnmount() {

// Logic before component is unmounted

},

unmounted() {

// Logic after component is unmounted

}

}

**both working as same in vue 2 to vue3 only function hooks name is changed.**

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++