**VUE DOCUMENTATION**

// Create a new Vue project using a webpack template ()  
/\*  
Create Vue.js application  
We will use npm to create a latest vue application template, please follow the steps below.  
1. Open your window command and cd to the folder you want to store the vue project  
2. use (npm create vue@latest) and follow the prompts  
3. run the new created vue template  
$ cd <your-project-name>  
$ npm install  
$ npm run dev  
4. Check the vue app in the browser based on the url listed in the terminal.  
  
 \*/  
//A .vue file is a custom file format that uses HTML-like syntax to describe a portion of the UI.  
// They are also known as components  
//Each file contains 3 language blocks  
//<template></template> (the HTML of your UI that defines the structure)  
//<script></script> (data and logic for the template is maintained)  
//<style></style> the css, where the style for the template markup is specified  
//Each .vue file handles a functionality with its own html, js and css markup  
//The browser does not understand what a .vue file is, so "webpack"  
// and the vue loader will parse the file  
//extract each of the 3 blocks and assemble them back into a format that browsers can understand  
  
// HOW DATA IS MAINTAINED  
//You can use Vue directly from a CDN via the script tag (<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>)  
//This makes the setup a lot simpler, and is suitable for enhancing static HTML  
// or integrating with a backend framework  
// The default object exported from the script block can contain  
// a data() / setup() property  
// Which is also a function. This function returns an object which represents  
// the data used by the template  
/\*  
 <script>  
 const { createApp, ref } = Vue // Requires createApp and ref from vue  
 createApp({  
 setup() {  
 const message = ref('Hello vue!')  
 return {  
` message,  
 }  
 },  
  
 }).mount('#app')  
 </script>  
 \*/  
// use {{ }} in the template html tag to display the return property in our data()/ setup() function e.g. {{message}}  
//When rendering using the index.html file, ensure all html is within the div tag with id "app"  
/\*  
<div id="app">{{ message }}  
</div>  
  
 \*/  
// To bind html to properties (e.g. name: '<b>Ada</b>') use the V-html  
// directive (e.g. <p v-html="name"></p>)  
// To bind a property to an element's attribute (isDisabled: false)  
// use the v-bind directive (<button v-bind:disabled = "isDisabled"> Button Disabled</button>)  
// Apply css class (e.g. <h2 class = "underline"> Underlined Text</h2>)  
  
//CONDITIONAL RENDERING  
// Show or hide html based on specific conditions  
//v-if renders the element if the condition is true e.g <h2 v-if ="num === 0"> Num is zero</h2>  
// v-else will render the element if the condition above is false  
// e.g <h2 v-else ="num === 0"> Num is not zero</h2>  
//Display more than 1 element at the same time by wrapping them in a template tag e.g  
/\* <template v-if = "display">  
 <h2> Display both elements</h2>  
 <h2> All at once</h2>  
 </template>  
  
 \*/  
// There is also the v-show directive  
  
//LIST RENDERING  
//To render a list use the v-for directive e.g. <h2 v-for ="name in eNames">{{name}}</h2>  
// where eNames is an array of names. You may need to use a key "keyword"  
  
//CONDITIONAL LIST RENDERING  
// Nest a v-if directive within a v-for directive. Wrap them in a template tag  
/\* <template v-for ="name in eNames" >  
 <h2 v-if ="name === 'sam'"> {{ name }}</h2>  
 </template>  
  
 \*/  
  
//FUNCTIONS  
// add the methods property (also an object) within the data()/ setup() function  
//include an add() function with parameters,  
/\* methods:{  
 add(a,b,c) {  
 return a+b+c  
 }  
 }  
  
 \*/  
// and call this function in the main div or template element (<h2>Add Function - {{add(2,7,9)}}</h2>)  
//Creating Functions that use data properties returned from data() or setup()  
// Add a data property to be returned (baseNumber: 5)  
// Create a new function multiply() that takes a num parameter, and returns num \* baseNumber  
// Because baseNumber is a data property, we access it in the function using this.baseNumber  
/\* multiply(num) {  
 return num \* this.baseNumber  
 }  
  
 \*/  
//Add another data property (baseValue: 4) we will pass this in as an argument in our main div or template element  
// Call the function  
  
// EVENT HANDLING  
// Listen to events using the v-on directive  
// To listen to an onClick event of a button (v-on:click = "[some code or function name]")  
// e.g. <button v-on:click ="increment(2)">Increment by 2: {{ count }}</button>  
/\*  
 increment(num) {  
 this.count+= num;  
 }  
 \*/  
// v-on: can also be replaced by @  
  
//FORM HANDLING  
//Bind form data in script to form controls in html template and vice-versa using the v-model derivative  
//Create a data property "formValues" that would keep track of the form controls/elements  
//include the form elements as formValues properties and initialize them to ''  
/\*  
 formValues: {  
 username: '',  
 password:''  
 }  
 \*/  
// bind the formValue properties to the HTML elemnts using the v-model directive  
// <input type="text" name="username" id="username" v-model="formValues.username">  
  
  
//COMPONENTS (I have used the VueProject2 to demo this)  
// create a Greet component in the components folder, export in its script tag,  
/\*  
<script>  
export default {  
 name: 'Greet', //Give the component export a name  
}  
</script>  
 \*/  
// import it in the App.vue file and use it in the App.vue file template tag  
// by specifying it as a component in the "export default", as well as a custom HTML tag in the template tag  
/\*  
 <script>  
import Greet from './components/Greet.vue'  
export default {  
 name: 'App',  
 components: {  
 Greet,  
 }  
}  
</script>  
.... Use it by adding it in App's template tag (<Greet></Greet>)  
  
 \*/  
  
//PROPS  
// Props are custom attributes we can register on a component. It allows the component's  
// content to be dynamic  
// We would pass a name from the App component to the Greet component, and render that name  
// to the browser  
// To specify props for a component we add custom attributes.  
// To specify a name prop, we specify a name attribute in the custom tag of the component,  
// and assign a value.  
// e.g <Greet name = "Ben"></Greet>  
//We have sent data to the Greet component, how do we retrieve this?  
//1).In the default export of the Greet component, specify an optional props property  
// along with the prop we want to use  
//The props property is an array of all data properties or custom attributes that  
// the components would accept  
// from the parent component (in this case Apps is parent and Greet is the child)  
/\*  
export default {  
 name: 'Greet', //Give the component export a name  
 props: ['name'],  
}  
 \*/  
//2). Bind the name prop to the Greet component template tag just like any other attribute  
// <h2> Hello {{ name }}</h2>  
// Add another prop (location)  
// To assign dynamic values (data properties) as props, create 2 data properties in the App data function  
// and bind them to the template using the v-bind directive  
  
//FETCH FROM API  
// First, Create a component GetData, add export default() with a name, add a Get Data button in template tag  
/\*  
 <template>  
 <button> Get Data</button>  
</template>  
  
  
<script>  
 export default {  
 name: 'GetData',  
 }  
</script>  
 \*/  
  
// Import the component in App.vue, add it as a component, include it in App's template as a custom tag  
  
/\*  
export default {  
 name: 'App',  
 components: {  
 Greet,  
 GetData,  
 }  
 --------------  
  
 <GetData></GetData>  
 \*/  
  
  
// Next, ensure cors is installed, required and called on node.js/ express webserver (npm install cors)  
// Next, in the GetData component, within methods, use fetch to make a get request to the node.js/ express  
// data API ("http://localhost:8080/api/employee").  
// Place this code in a new async method (fetchData)  
  
/\*  
 methods: {  
 async fetchData() {  
 const response = await fetch("http://localhost:8080/api/employee");  
 this.data = await response.json();  
 }  
 }  
 \*/  
  
// Next, execute the async fetchData() method on the Get Data button's click event  
  
// <button @click ="fetchData"> Get Data</button>  
  
//Using the v-for directive, loop through the fetched data  
  
//<h2 v-for ="emp in data">{{emp.name}}'s email is {{emp.email}}: {{emp.name}} earns {{emp.Salary}}</h2>  
  
  
//POST DATA TO API  
// First, Create a component PostData, add export default() in script with a name,  
/\*  
  
<script>  
 export default {  
 name: 'PostData',  
 }  
</script>  
 \*/  
  
// Import the component in App.vue, add it as a component, and include it in App's template as a custom tag  
  
/\*  
export default {  
 name: 'App',  
 components: {  
 Greet,  
 GetData,  
 PostData,  
 }  
 --------------  
  
 <PostData></PostData>  
 \*/  
  
// Next, ensure cors is installed, required and called on node.js/ express webserver (npm install cors)  
// Create a formValues object in the PostData component data() property return object  
// The formsValues object would hold property variables for the form elements  
  
/\*  
data () {  
 return {  
 //Return form data  
 formValues: {  
 name: '',  
 email:'',  
 Salary:'',  
 }  
 }  
 },  
 \*/  
  
//Create a form in the PostData component's Template tag with 3 fields and a submit button  
// Bind each element to its respective property variable with the v-model directive  
  
/\*  
<form v-on:click="sendData" >  
<div >  
 <label for="name" >Name </label>  
 <input type="text" name="name" id="name" v-model="formValues.name">  
</div>  
 .........  
 .........  
  
</form>  
 \*/  
  
// Next, use fetch to make a post request to the data API ("http://localhost:8080/api/employee").  
// Log the posted data to the console, as well as any errors  
  
//CREATING ROUTES  
  
// Our application only contains one \*.html file.  
// That means we cannot direct people to other html files to show them different contents on our page.  
// With routing set up appropriately in Vue, you can open the Vue application with an extension to a URL address,  
// like "/RouteA" and you will be directed to the component with the specified content  
  
//First, create 2 components to be used as routes (RouteA and RouteB)  
// Next, install the Vue Router Library (npm install vue-router@4)  
//Next, in main.js import { createRouter, createWebHistory } from 'vue-router'  
//createRouter creates a route handeler  
//createWebhistory allows you use the forward and backward button on the browser for the routes  
//Next, in main.js import RouteA and RouteB components  
  
/\*  
import RouteA from './components/RouteA.vue'  
import RouteB from './components/RouteB.vue'  
 \*/  
  
//Next, in main.js create a route handler, include the ability to navigate,  
// then, specify the routes for each component  
  
/\*  
const router = createRouter({  
 history: createWebHistory(),  
 routes: [  
 { path: '/routeA', component: RouteA },  
 { path: '/routeB', component: RouteB },  
 ]  
});  
 \*/  
  
// In main.js, ensure you create an instance of the App.vue component,  
// create a router middleware using the App component, and  
//mount the App component on the DOM element with id "app"  
  
/\*  
const app = createApp(App)  
app.use(router);  
app.mount('#app')  
 \*/  
  
//Next, in the template tag of the App.vue component, enable user navigation  
//with the router-link tag (this renders as a hyperlink)  
  
/\*  
 <router-link to="/routeA">Activities</router-link>  
 <router-link to="/routeB">Movies</router-link>  
 \*/  
  
//Next, in the template tag of the App.vue component,  
// use/display the new router component (<router-view></router-view>)