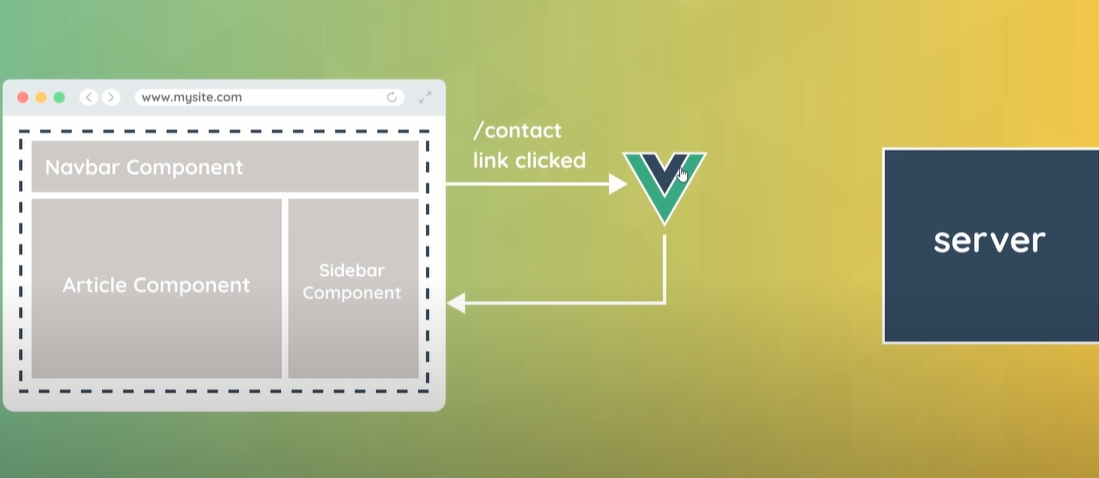
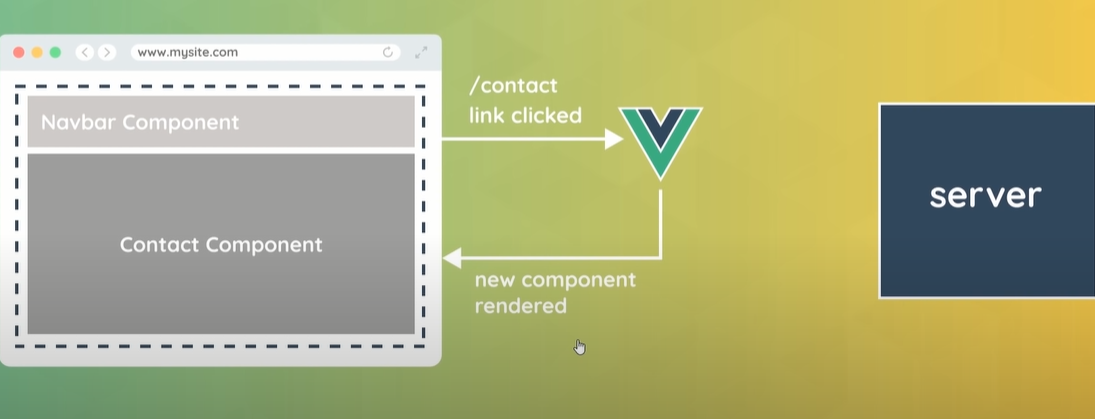
Framework for creating dynamic data driven interactive web sites.

With VueJs we can create Widgets. They are self-contained and can be reused.  
More commonly we create a Single Page Applications. All routing is done in the browser and not on the server.

Normally Browser would constantly send a request to the Server and time would be needed for each request.  
However when we use Vue to control a website browser would only send first request to the Server. After that Vue JavaScript Bundle would take control.



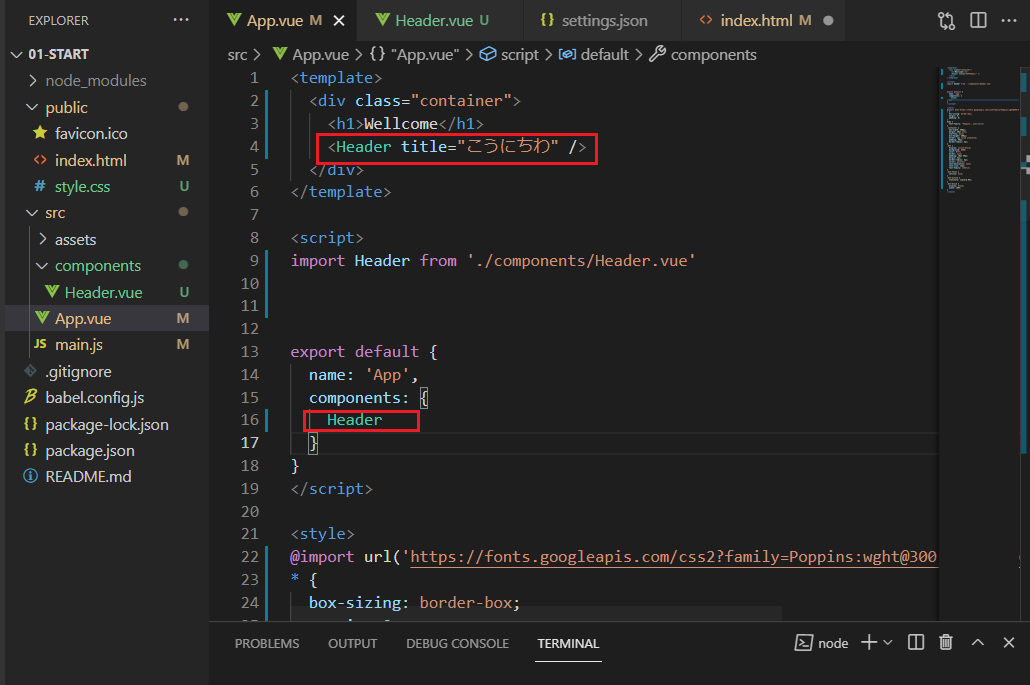
Vue intercepts a request and it doesn’t go to Server but it handles request in Browser. It is much faster.



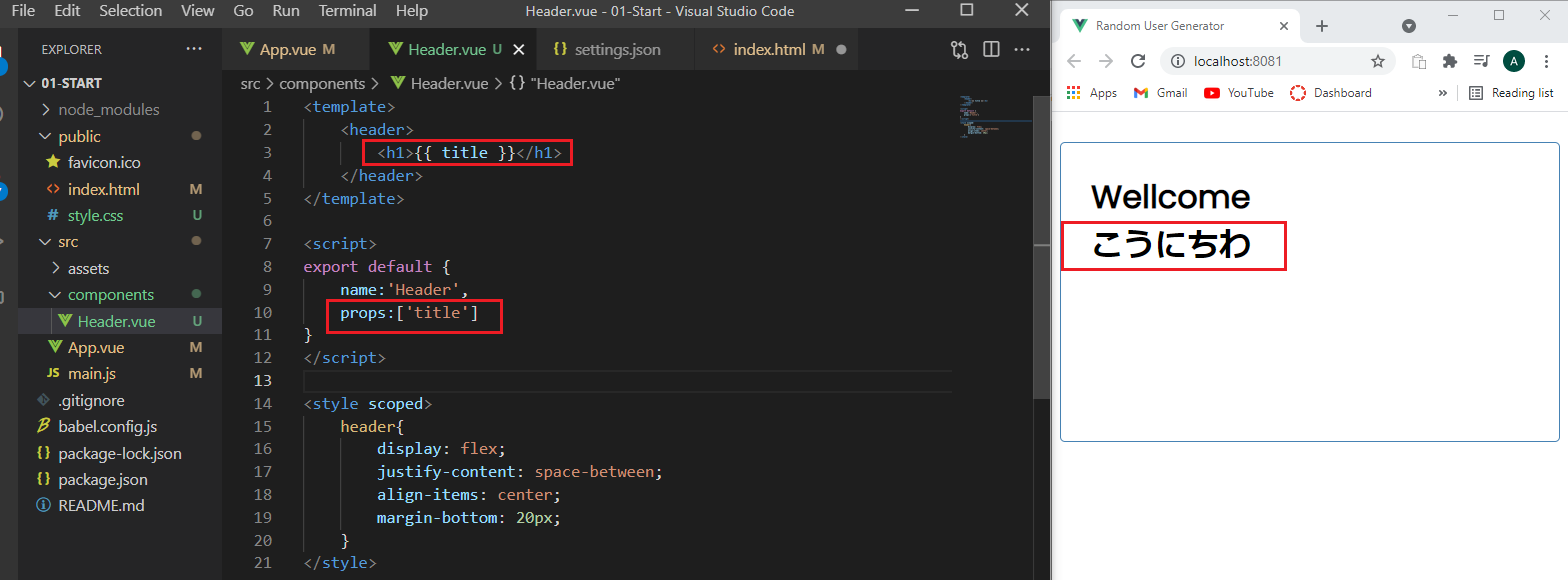
This is called a Single Page Application.

1. Only a single HTML page sent(initially) to the browser.
2. Vue intercepts subsequent requests and handles “page” changes in the browser by swapping what components are shown on the page.
3. Results in much faster and smoother website experience.

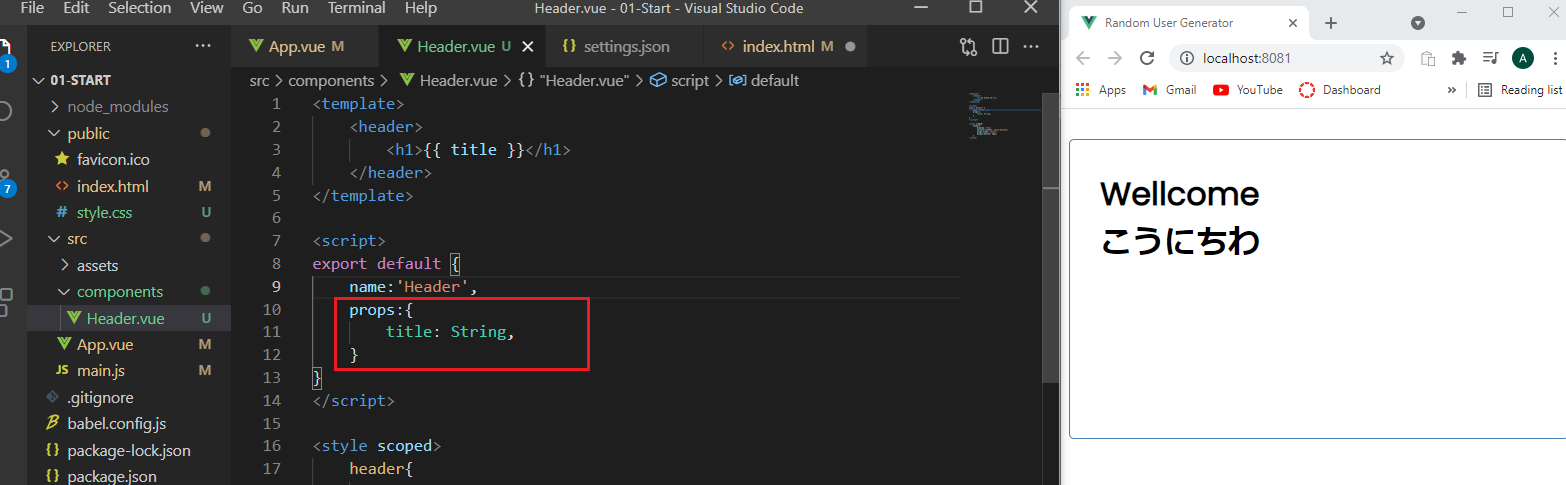
Props can be passed into components:

If we want to work with the title inside header component we need to define Props  


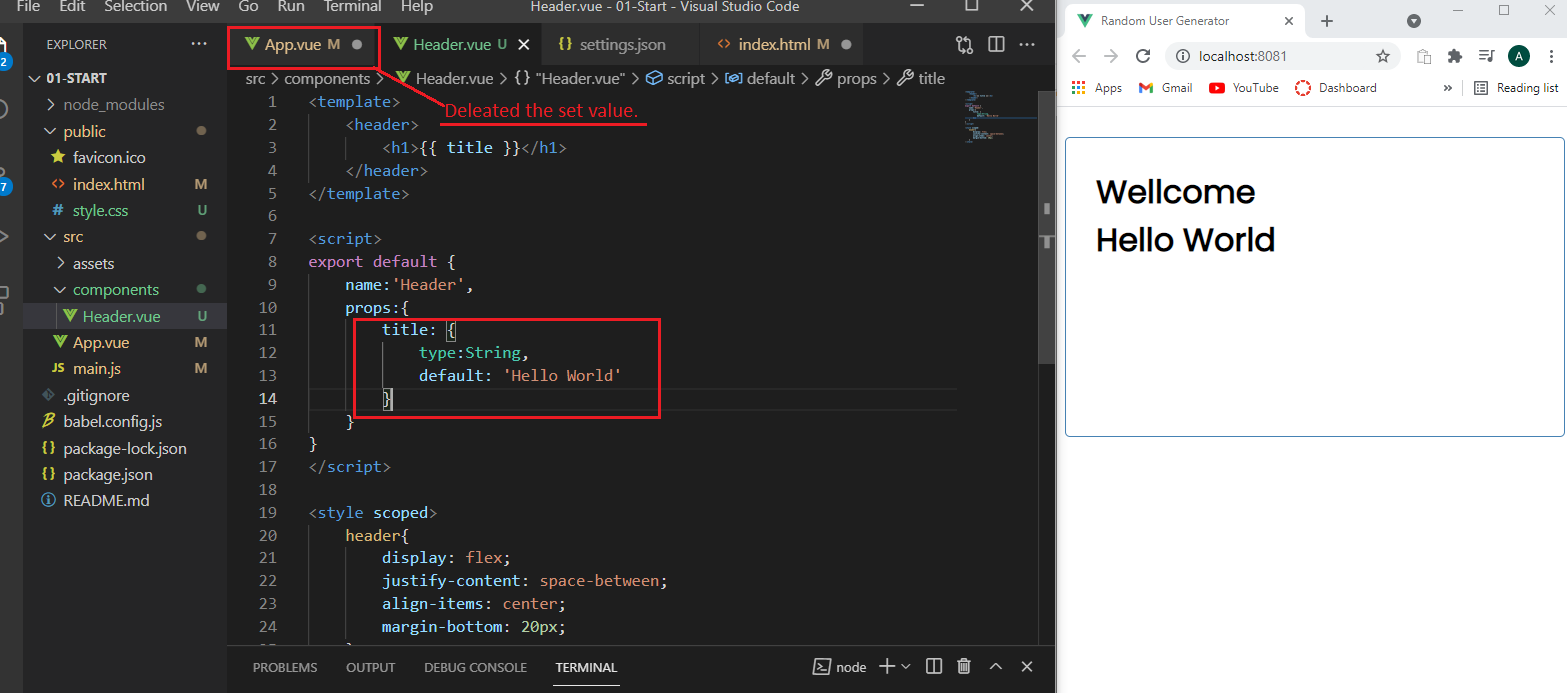
One ways is to have a prop with array name:



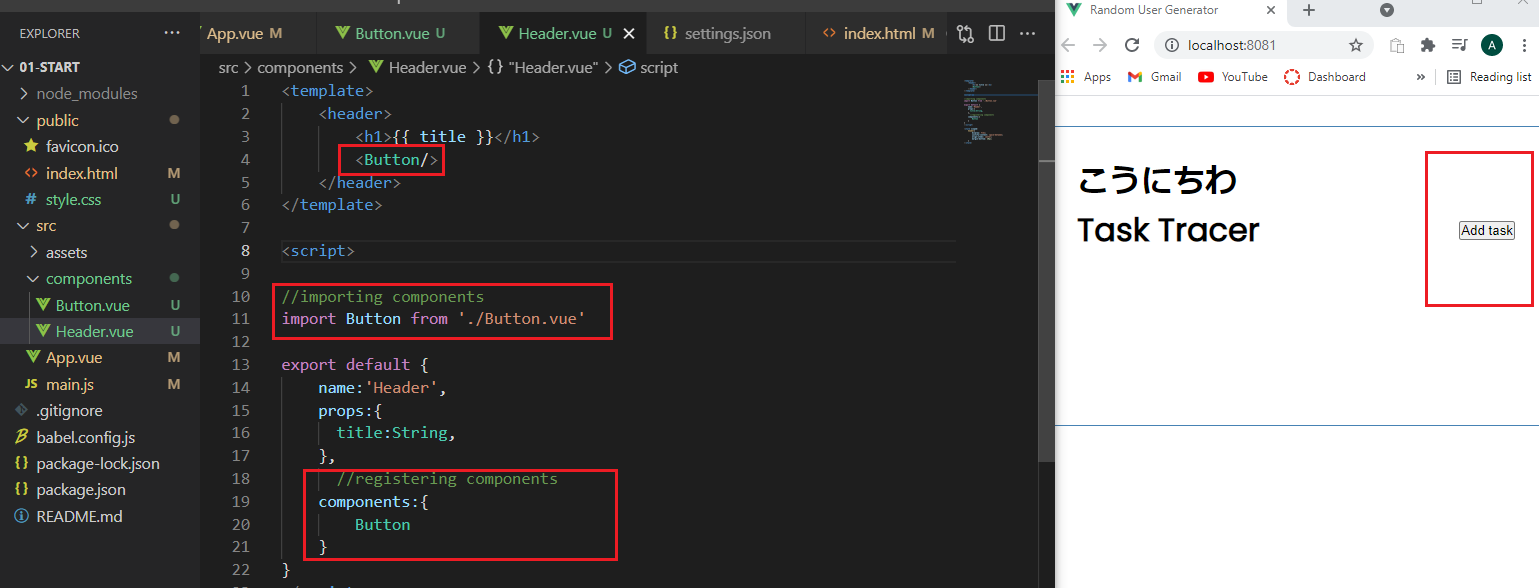
Or we can define it as an object and specify a type:



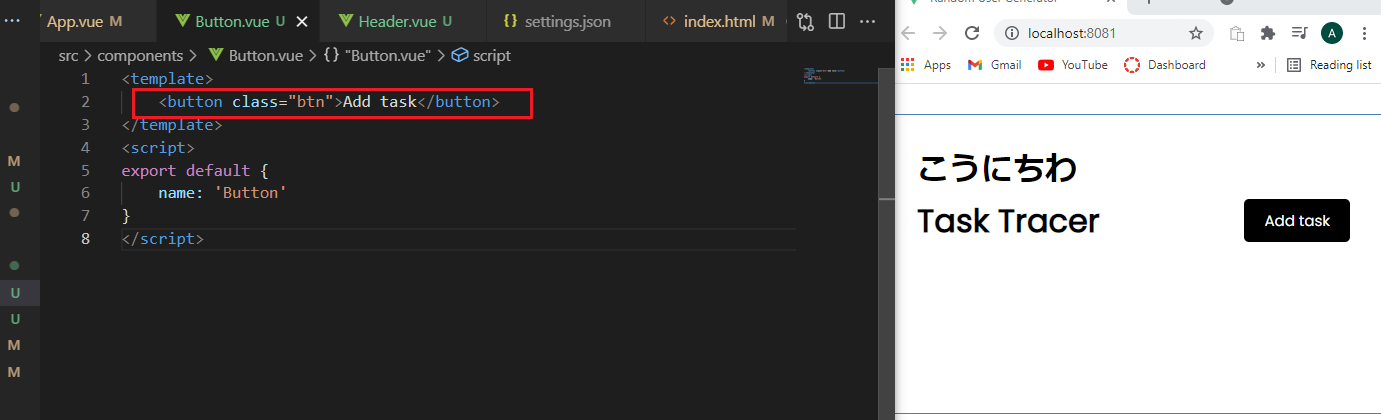
If we want a default value:



**Button component**

So I can put one component into another .  
Example putting Button.vue into Header.vue.  


We can style components using global styles in App.vue



We can style the button with inline styling.

  <Button text="Add Your Task" color="green"/>

Binding color style!:

<template>

    <button :style="{background: color}" class="btn">{{text}}</button>

</template>

<script>

export default {

    name: 'Button',

    props:{

        text: String,

        color: String,

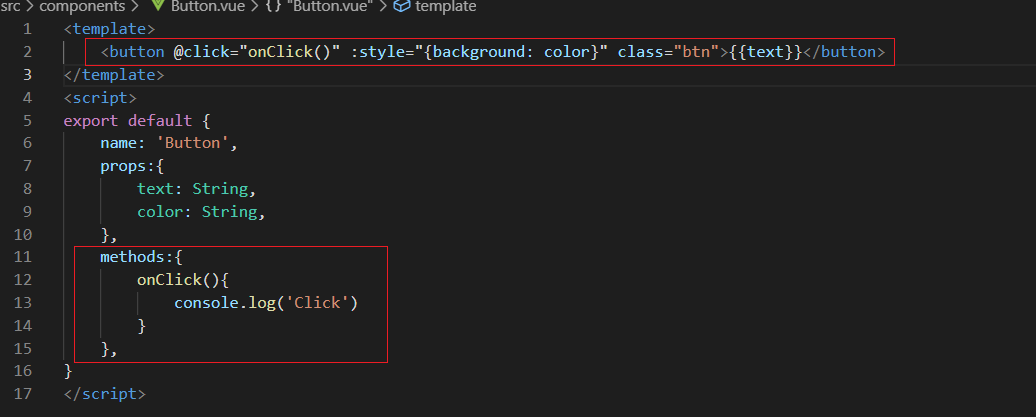
    },

}



**Events**

v-on:click=”onClick()” or @click=”onClick()”

Next we need to define a method.

**Task data & created() method**

export default {

  name: 'App',

  components: {

    Header

  },

  data(){ //function that returns object

    return {

      tasks: [] //an empty array

    }

  },

//lyfecycle method where we can make HTTP requests

  created(){     //when created runs we are gonna fill it up

    this.tasks = [

      {

        id: 1,

        text: 'Doctors apointment',

        day: 'March 1st',

        reminder: true,

      },

      {

        id: 2,

        text: 'Meeting at school',

        day: 'March 3rd',

        reminder: true,

      },

      {

        id: 3,

        text: 'Food Shopping',

        day: 'March 3rd',

        reminder: false,

      },

    ]

  }

}

</script>

Next we will need tasks component to pass this data into it so we can render it on the page.

Tasks.vue

<template>

</template>

<script>

export default {

    name:'Tasks',

    props:{

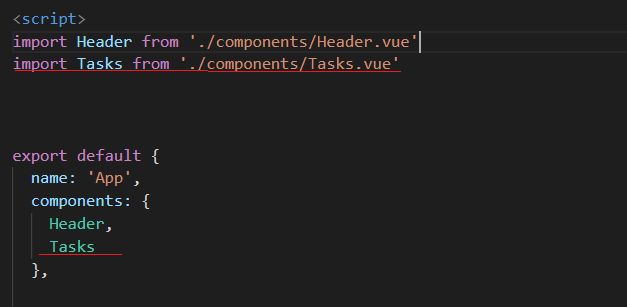
        tasks: Array

    },

}

</script>

Now we register it in App.vue so we can use it to store data.



Next we can v-bind data(One way data binding)

   <Tasks :tasks="tasks" /> <!-- Tasks.tasks = this.tasks  passing in array-->

**V-For Loop**

<template>

<div id="tasks.ids">

    <div v-for="task in tasks" v-bind:key="task.id">

        <Task :task="task" />

    </div>

**Task component:**

 this.tasks = [

      {

        id: 1,

        text: 'Doctors apointment',

        day: 'March 1st',

        reminder: true,

      },

<template>

<div class="task">

     <h3> {{task.text}} </h3>

     <p> {{task.day}} </p>

</div>

</template>

<script>

export default {

    name:'Task',

    props:{

        task:Object

    }

}

</script>

If we have scoped style we don’t need to put class to the ex: div element

<div :class="[task.reminder ? 'reminder' : '','task']">

<!-- if remindet == true then set class to reminder else don't set it to nothing but always set to task class -->

CDN – A CDN (**Content Delivery Network**) is a group of servers spread out over many locations. ... CDNs are used widely for delivering stylesheets and Javascript files (static assets) of libraries like Bootstrap, jQuery etc.

 <h3> {{task.text}}

         <i class="fas fa-times"></i>

          </h3>

<style scoped>

.fas {

    color: red;

}

**Deleting Task**

We need actions that emit up – **Emit Events**  
Task is 3 level low. Data is 3 lvls up and we need to access it in App.vue

Task.vue 🡪 Tasks.vue 🡪 App.vue

3rd Lvl

 <i @click="onDelete(task.id)" ></i>

  methods:{

        onDelete(id){

            console.log(id)

            this.$emit('delete-task',id)

        },

    }

}

Or we can just :

    <i @click="$emit('delete-task',task.id)" class="fas fa-times"></i>

          </h3>

We need to catch emit in the next lvl up.

2nd Lvl

<template>

<div id="tasks.ids">

    <div v-for="task in tasks" v-bind:key="task.id">

        <Task @delete-task="$emit('delete-task',task.id)" :task="task" />

 emits: ['delete-task']

}

</script>

1st Lvl

<Tasks @delete-task="deleteTask" :tasks="tasks" />

 methods:{

    deleteTask(id){

      if(confirm('Are you sure ?')){

        this.tasks = this.tasks.filter((task) => task.id !== id)

        //We want all back except from the task with the passed id

      }

    }

  },

Toggle reminder method:

  toggleReminder(id){

      this.tasks = this.tasks.map((task)=> task.id == id ? {...task,reminder :  !task.reminder} : task)

      // for each task check to se if task.id == id

      // true return array of objects and change remindet to oposite of current task reminder

      // false we return initial task

    }

V-MODEL

You can use the v-model directive to create two-way data bindings on form input, textarea, and select elements.

* text and textarea elements use *value* property and *input* event;
* checkboxes and radiobuttons use *checked* property and *change* event;
* select fields use value as a *prop* and *change* as an event.

[https://vuejs.org/v2/guide/forms.html]

 <input type="text" v-model="name" placeholder="Edit me" name="text"  />

      <p>Name is :{{name}}</p>

   props:{

        name:String,

    },



<!DOCTYPE html>

<html>

    <head>

        <meta charset="UTF-8">

        <title>VueJS Tutorial</title>

        <link href="styles.css" rel="stylesheet"/>

        <script src="https://cdn.jsdelivr.net/npm/vue@2.6.14/dist/vue.js"></script>

    </head>

    <body>

      <script src="app.js"></script>

    </body>

</html>

Rel (alternate, author, dns-prefetch, help, icon, stylesheet, …)

Required. Specifies the relationship between the current document and the linked document

* The Vue instance

new Vue({});

//Controls the app , app is running trough this

//We can make more vue instances for separate widgets

//How does it work?

//Vue instance takes {} - object as a paremter

//Trough this object we can pass trough properties,

//paremeters, options etc.

new Vue({

    el:'#vue-ap',

});

//el property string

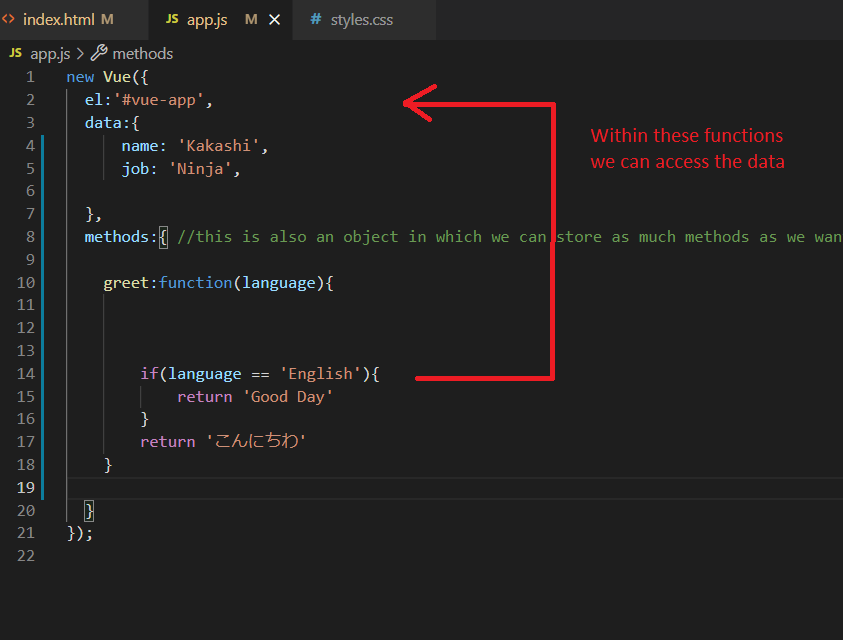
//defines which elements this vue instance is going //to control

//in this case it is the whole page

//we need the ROOT ELEMENT

// <div id="vue-app" ></div>

* The Data & Methods



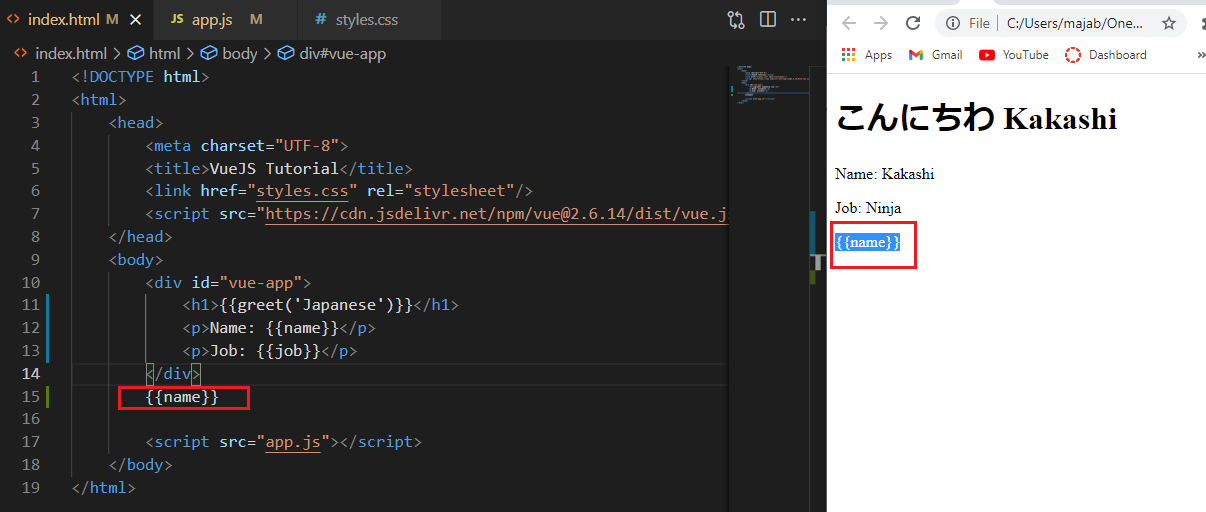
        if(language == 'English'){

            return 'Good Day ' + this.name

        }

        return 'こんにちわ ' + this.name

    }



Vue only controls div and anything outside div will not be translated by Vue.

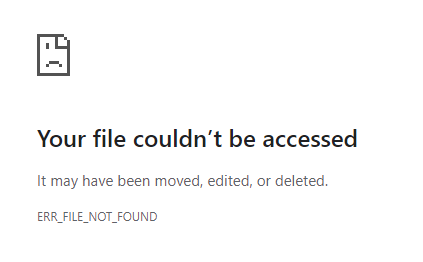
It doesn’t translate data to the DOM!

* Data Binding

  <a href={{website}}> Bio </a>

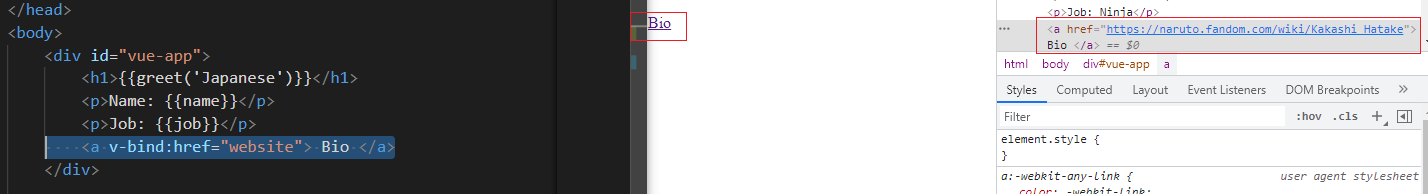
The output:

Bio



We need to use data binding

* v-bind:href=”…”
* v-html=”…”



 website: 'https://naruto.fandom.com/wiki/Kakashi\_Hatake',

 websiteTag: '<a href="https://naruto.fandom.com/wiki/Kakashi\_Hatake">Bio</a> ',

  <a v-bind:href="website"> Bio </a>

  <br>

  <a v-html="websiteTag">Bio</a>

* Events

v-on:event=”…”

 <h2>Events</h2>

            <button v-on:click="age++" >Add a year</button>

            <button v-on:click="age--">Substract a year</button>

            <p>Age {{age}}</p>

Extracting our methods to .js file:

 <h2>Events</h2>

            <button v-on:click="add" >Add a year</button>

            <button v-on:click="subtract">Subtract a year</button>

            <p>Age {{age}}</p>

We don’t need to add parentheses. We only put them when we want to display return of the function.

 <p>Age {{age}}</p>

            <br>

            <div id="canvas" v-on:mousemove="updateXY">{{x}}, {{y}}</div>

        </div>

 updateXY(event){

        this.x = event.offsetX;

        this.y = event.offsetY;

    }

* Event Modifiers

Allow us to give additional functionality. To event listeners.

Ex: listen to event once

  <button v-on:click.once="add(1)" >Add a year</button>

Ex: prevent modifier

  <a v-on:click.prevent="click" href="https://naruto.fandom.com/wiki/Icha\_Icha">Hobbies</a>

[https://vuejs.org/v2/guide/events.html]

* Keyboard Events

Keydown/up/press

  <h1>Keyboard Events</h1>

         <label>Name:</label>

         <input type="text" v-on:keyup.enter ="logName"/>

         <label>Age:</label>

         <input type="text" v-on:keyup.alt.enter ="logAge"/>

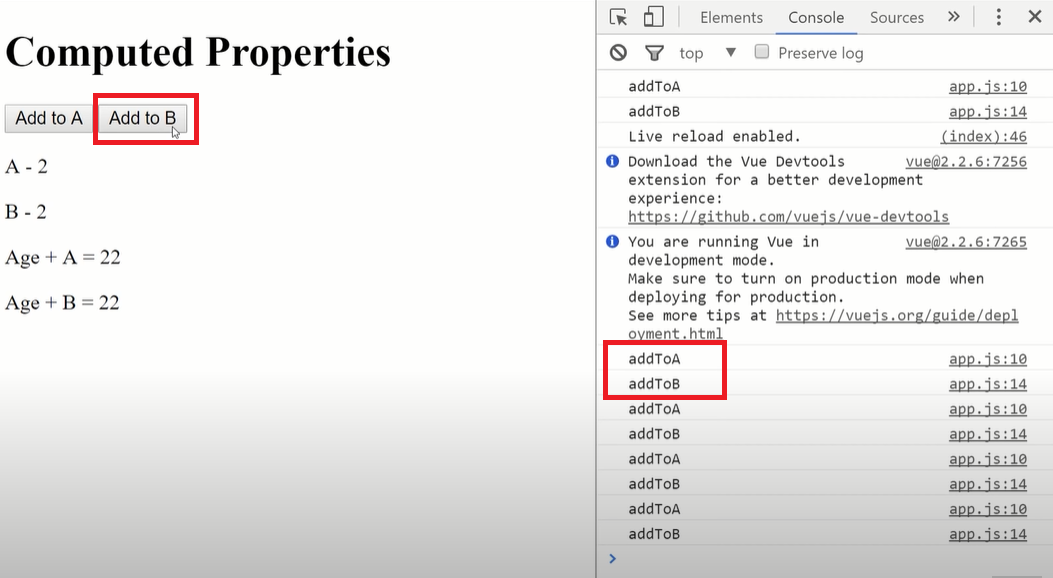
* Two way data binding [v-model]

<label>Name:</label>

        <span>{{name}}</span>

         <input type="text" v-model="name" />

* Computed Properties



Although B is changed both functions will be called.

Solution:

 /\*  methods:{

        addToA:function(){

            return this.a + this.age;

        },

        addToB:function(){

            return this.b + this.age;

        },

    },\*/

    computed:{

        addToA:function(){

            console.log('Add to A');

            return this.a + this.age;

        },

        addToB:function(){

            console.log('Add to B');

            return this.b + this.age;

        },

 <p>Age + A= {{ addToA }}</p>

 <p>Age + B= {{ addToB }}</p>

Just take off the parentheses.

"Computed values are also cached to avoid repetitively calculating a value that doesn't need to be re-calculated when it hasn't changed." and "Computed values are very valuable for manipulating data that exists on your Vue. Whenever you want to filter or transform your data, typically you will use a computed value for that purpose."