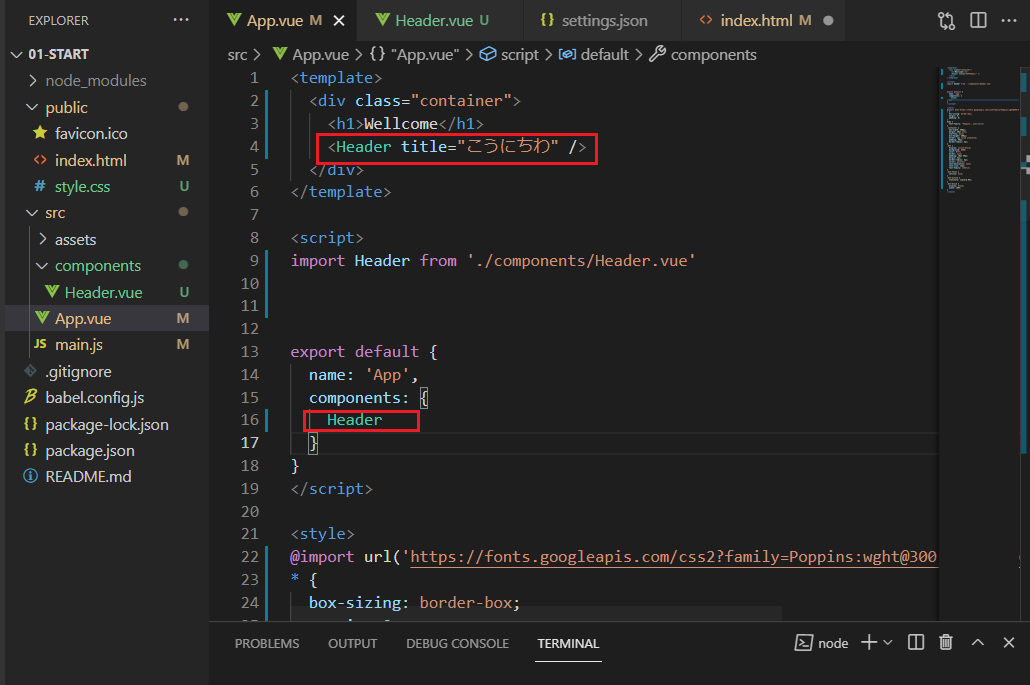
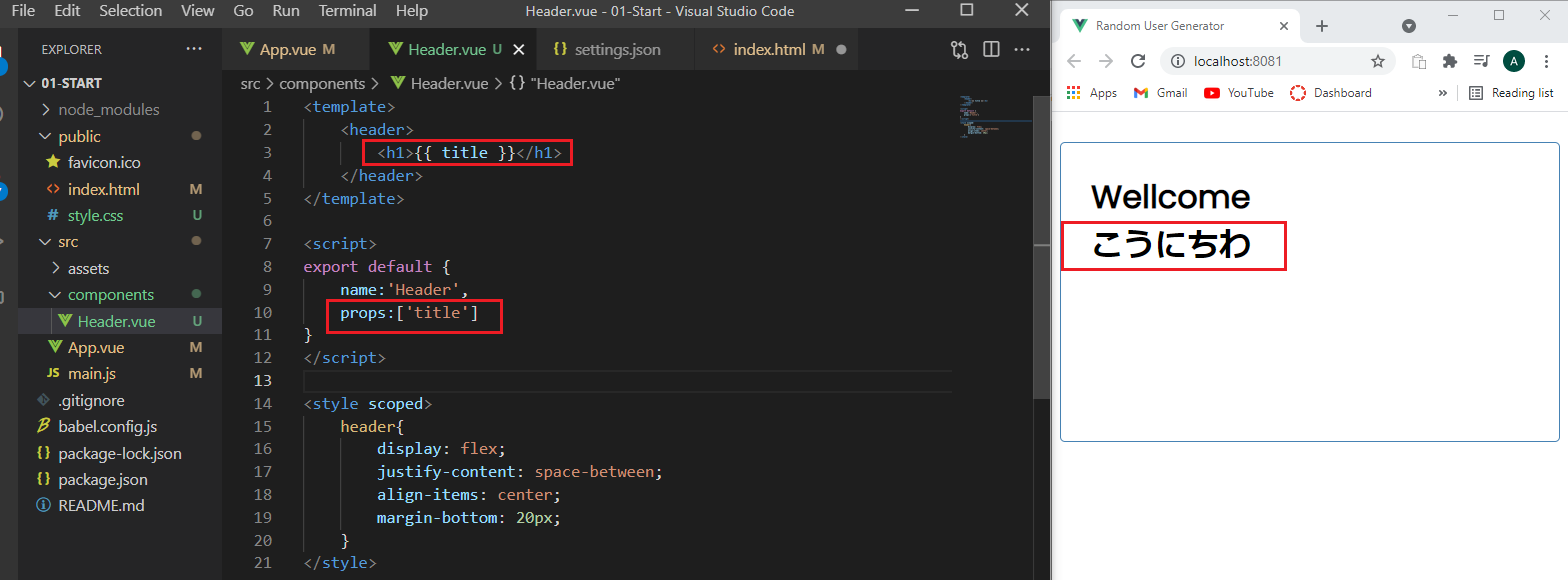
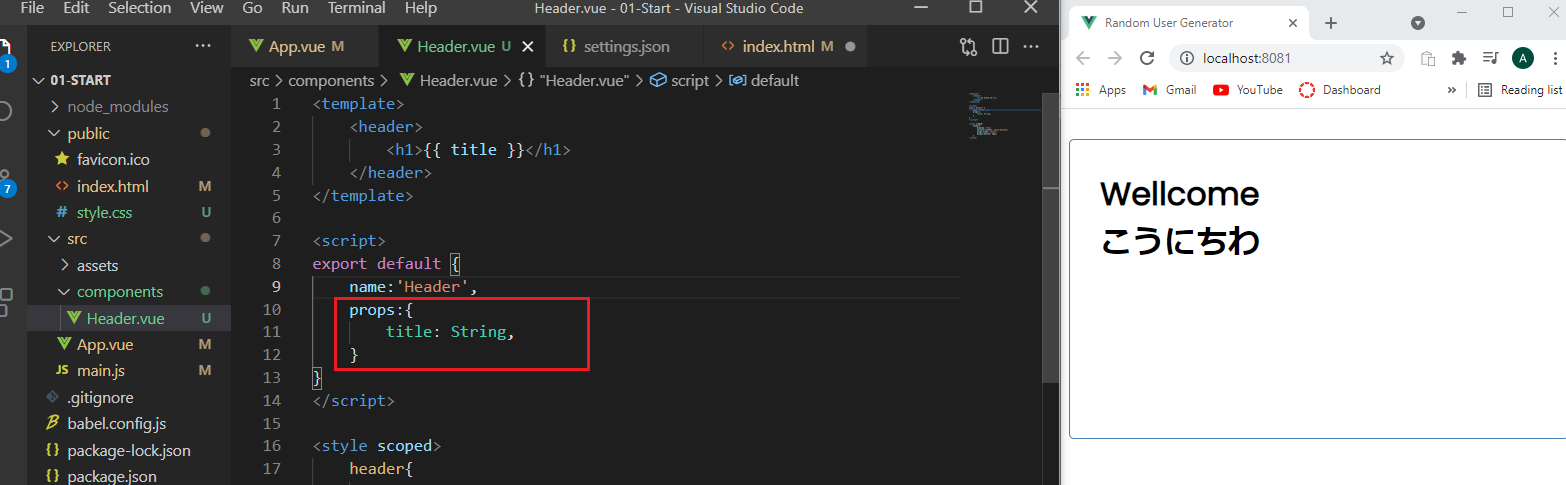
Props can be passed into components:

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


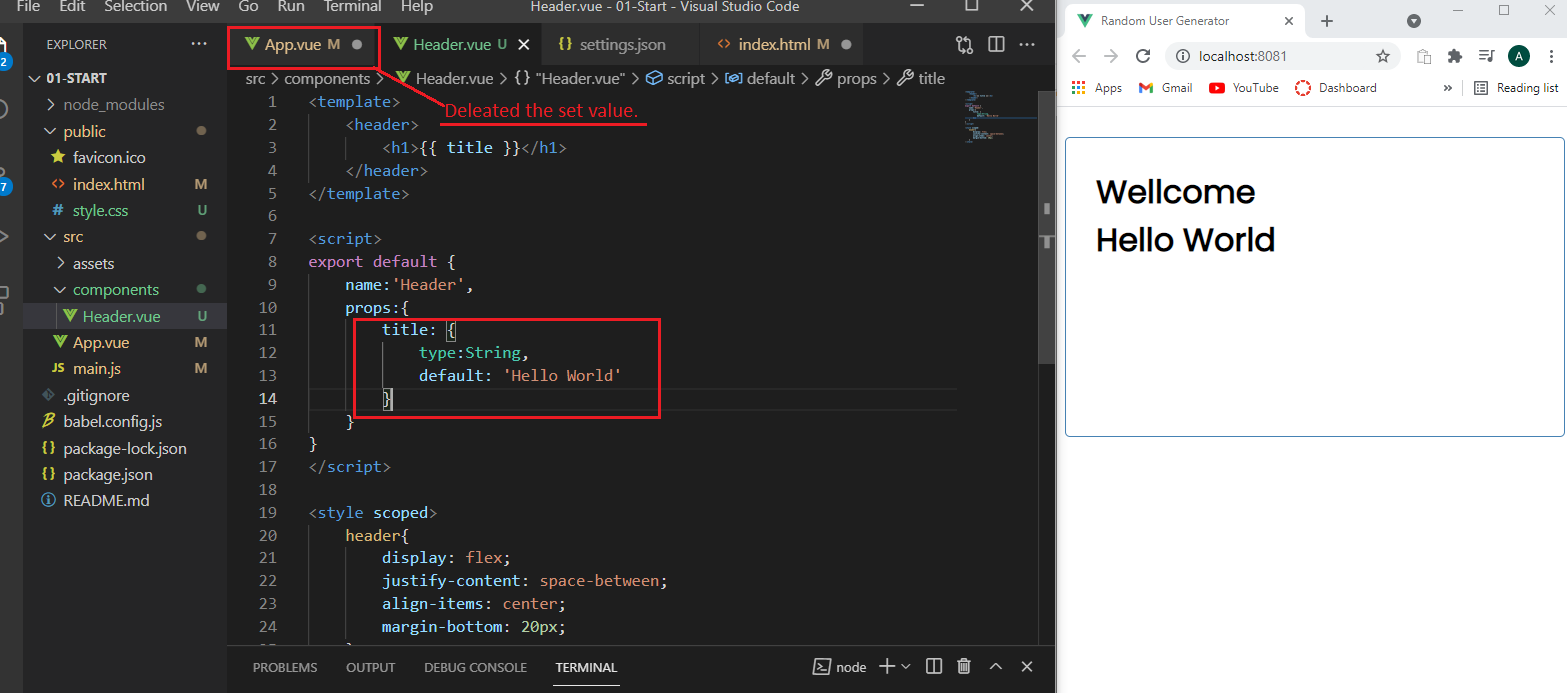
One ways is to have a prop with array name:



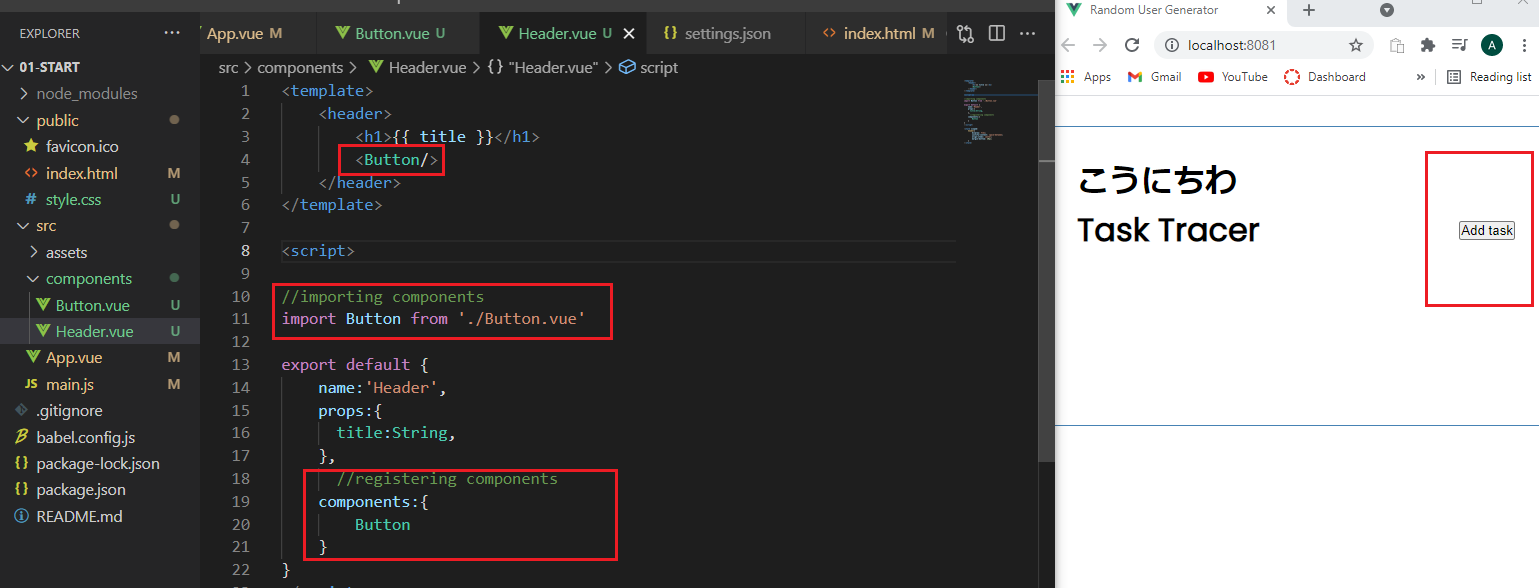
Or we can define it as an object and specify a type:



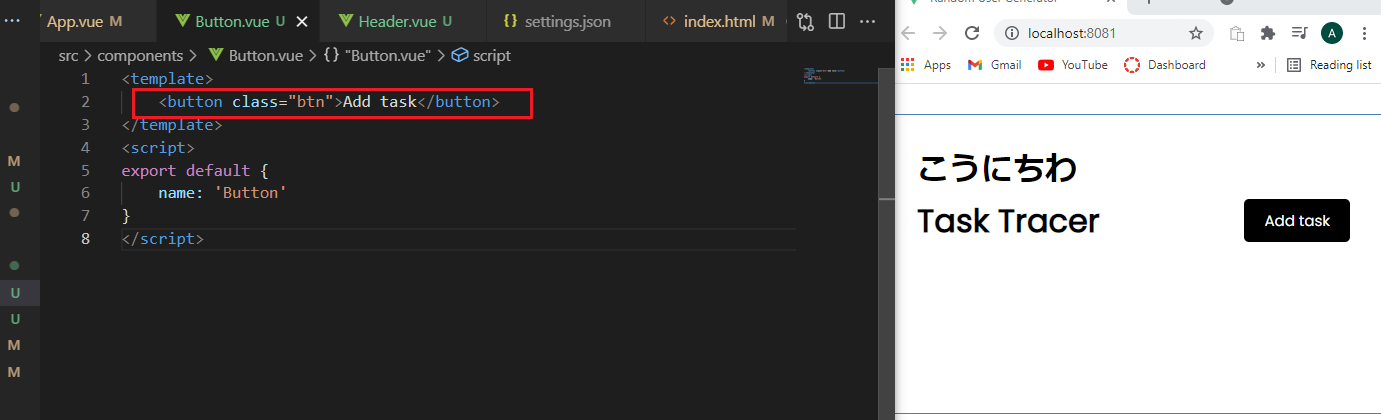
If we want a default value:



**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,

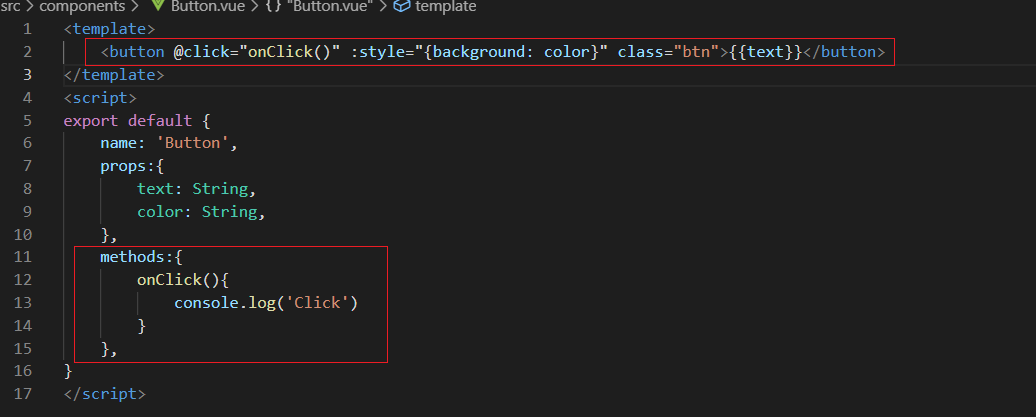
    },

}

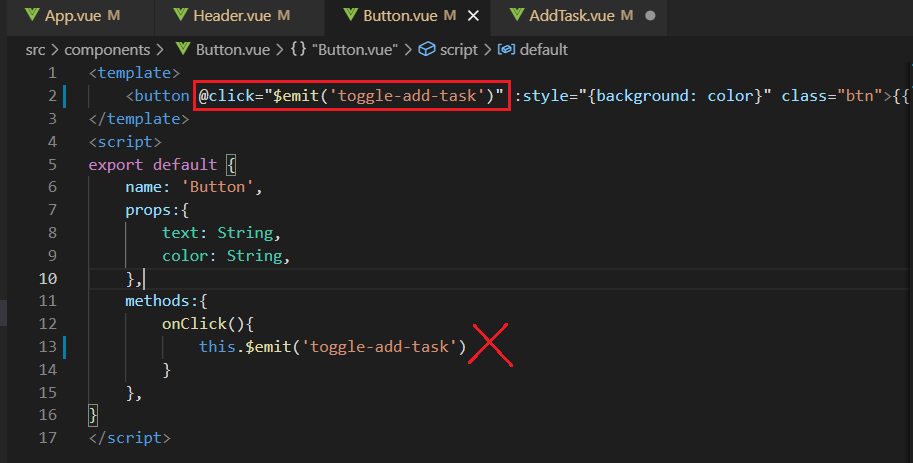


**Events**

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

Next we need to define a method.

Or we can use one line emit and don’t 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:{

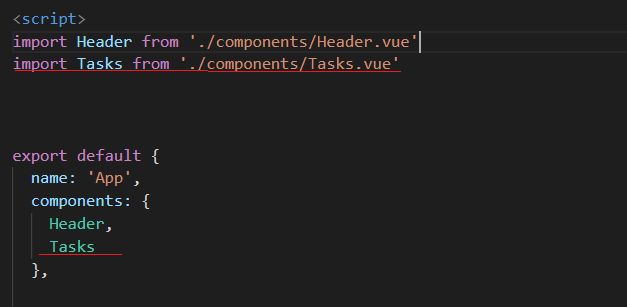
        tasks: Array

    },

}

</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

    }

}

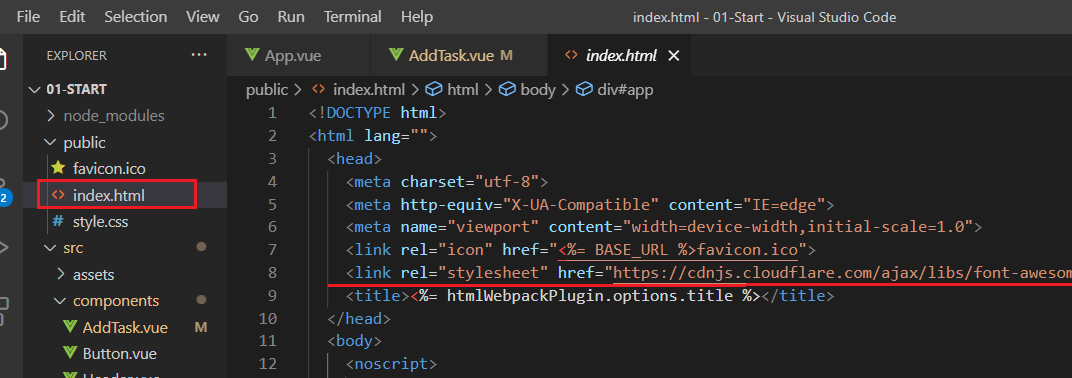
</script>

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,

    },



 <div class="form-control" >

      <label>Day & Time</label>

      <input type="text"

v-model="day"

name="day"

  placeholder="Add Day & Time" />

    </div>

 name: 'AddTask',

     data() {

    return {            //data returns an object

      text: '',          // V-MODEL binds data to input

      day: '',

      reminder: false,

    }

  },

ShowAddTask:

data(){ //function that returns object

    return {

      tasks: [] ,//an empty array

      showAddTask : false

    }

  },

Passing an argument/prop

 <Header @toggle-add-task="toggleAddTask" title="Task Tracer" :showAddTask="showAddTask" />

IF-CONDITION

 <Button @toggle-add-task="$emit('toggle-add-task')"

         :text="showAddTask ? 'Close' : 'Add Your Task'"

         :color="showAddTask ? 'red' : 'green'"

         />

[ : ] – makes it a v-bind

We can make reusable components

Generic UI Button method:

 methods:{

        onClick(){

            this.$emit('button-click')

        }

    },

Used in header to propagate specific method:

 <Button @button-click="$emit('toggle-add-task')"

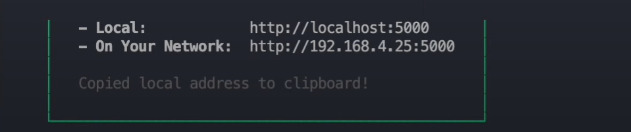
…

Building for Production

* npm run build 🡪 dist (>css , >js)

Build dist folder which we deploy to server.

* nmp i -g serve
* serve – s dist



Localhost:8081 is our dev server.

Localhost:5000 is our production server.

JASON – Server Setup

npm i jason-server

[https://www.npmjs.com/package/json-server]

Methods:

Adding to an array:

addTask(task){

     // this.tasks = [...this.tasks,task]

      this.tasks.push(task)

    },