



The Progressive JavaScript Framework



Pankaj Adhyapak



@pankaddy



@pankajadhyapak

Developer @ Acadgild.com

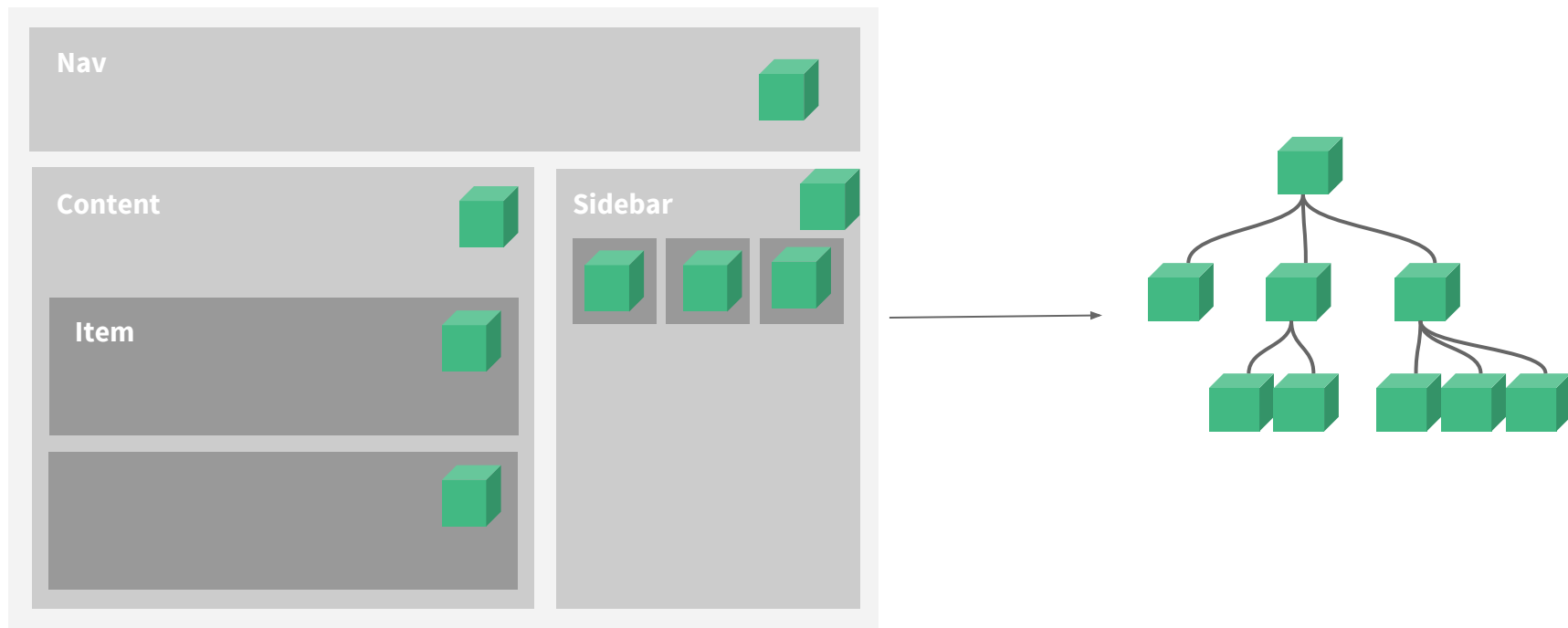
Component Communication

They help you extend basic HTML elements to encapsulate reusable code.

The entire UI can be abstracted into a tree of components

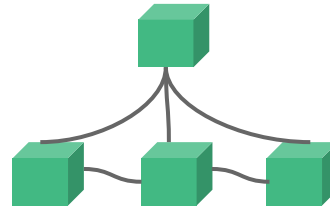
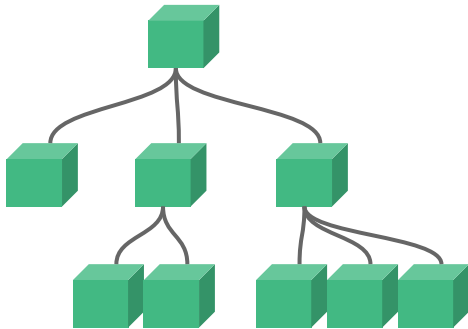
Every component is responsible for managing a piece of DOM

```
<side-bar></side-bar>  
<tabs>  
  <tab>...</tab>  
  <tab>...</tab>  
</tabs>
```

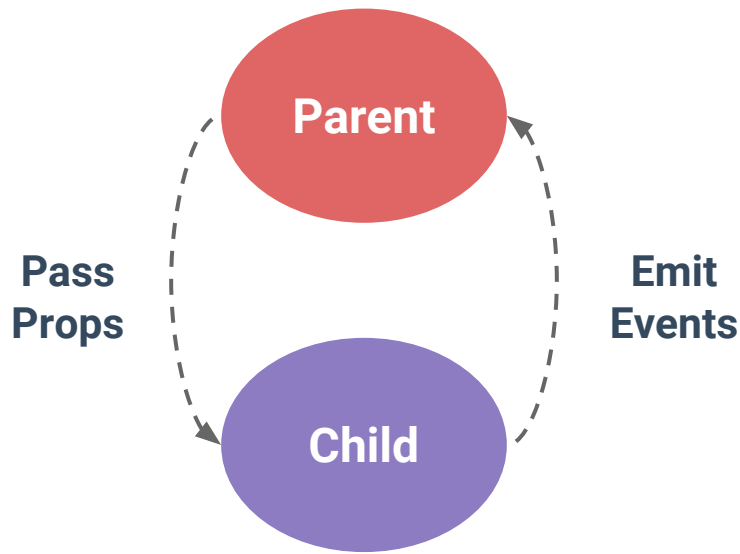


Types of communication

1. Parent to child
2. Child to parent
3. Non parent - child components



Props in, Events out



```
Vue.component('child', {  
  // declare the props  
  props: ['message'],  
  
  template: '<span>{{ message }}</span>'  
})
```

```
<child message="hello!"></child>
```

Dynamic props

```
<child :message="someVar"></child>
```

v-on with Custom Events

Every Vue instance implements an events interface, so it can:

- Listen to an event using **`$on(eventName)`**
- Trigger an event using **`$emit(eventName)`**

```
this.$emit('eventName', data)
```

```
this.$on('eventName', callback)
```



```
<div id="counter-event-example">
  <p>{{ total }}</p>
  <button-counter v-on:increment="incrementTotal"></button-counter>
  <button-counter @increment="incrementTotal"></button-counter>
</div>
```

```
new Vue({
  el: '#counter-event-example',
  data: {
    total: 0
  },
  methods: {
    incrementTotal: function () {
      this.total += 1
    }
  }
})
```

```
Vue.component('button-counter', {
  template: '<button v-on:click="increment">{{ counter }}</button>',
  data: function () {
    return { counter: 0 }
  },
  methods: {
    increment: function () {
      this.counter += 1
      this.$emit('increment')
    }
  },
})
```

Binding Native Events to Components

```
<my-component v-on:click.native="incrementTotal"></my-component>
```

```
Vue.component('button-counter', {  
  template: '<button>+1</button>'  
})
```

In root Vue Instance

```
incrementTotal: function () {  
  this.total += 1  
}
```

Form Input Components using Custom Events

```
<input v-model="something">
```

Is equal to

```
<input v-bind:value="something" v-on:input="something = $event.target.value">
```

When used with a component, this simplifies to

```
<custom-input
```

```
  :value="something"
```

```
  @input="value => { something = value }">
```

```
</custom-input>
```

So for a component to work with `v-model`, it should

- accept a **value** prop
- emit an **input** event with the new value

```
Vue.component('currency-input', {  
  template: `  
    <input  
      v-bind:value="value"  
      v-on:input="updateValue($event.target.value)">  
  `,  
  props: ['value'],  
  methods: {  
    updateValue: function (value) {  
      var formattedValue = value.trim().slice(0, value.indexOf('.') + 3)  
      this.$emit('input', Number(formattedValue))  
    }  
  }  
})
```

```
<currency-input v-model="price"></currency-input>
```

Non Parent-Child Communication

Vue instance as a central event bus

```
var bus = new Vue()
```

```
// in component A's method
```

```
bus.$emit('id-selected', 1)
```

```
// in component B's created hook
```

```
bus.$on('id-selected', function (id) {
```

```
  // ...
```

```
})
```

```
Vue.mixin({  
  data() {  
    return { eventHub: new Vue() }  
  }  
})
```

OR

```
export default new Vue() // in EventHub.js  
import eventHub from 'EventHub' // in components
```

OR

```
window.EventHub = new Vue();
```

In Components

```
//A Component
```

```
this.eventHub.$emit('eventname', data); or EventHub.$emit('eventName', data);
```

```
//B Component in created method
```

```
this.eventHub.$on('eventName', callback); or EventHub.$on('eventName', callback);
```



Examples



Questions ??



Thanks!