Communicating between Components

According to the *p8-start* project, there is a parent **user** component and two child components: **detail** and **edit**.

```
// App.vue template
<template>
    <div class="container">
        <div class="row">
            <div class="col-xs-12">
                // User component defined in User.vue
                <app-user></app-user>
            </div>
        </div>
    </div>
</template>
```

```
// User.vue template
<template>
    <div class="component">
        <h1>The User Component</h1>
        <p>I'm an awesome User!</p>
        <hr>
        <div class="row">
            <div class="col-xs-12 col-sm-6">
                // UserDetail sub-component
(UserDetail.vue)
                <app-user-detail></app-user-detail>
            </div>
            <div class="col-xs-12 col-sm-6">
                // UserEdit sub-component
```

```
(UserEdit.vue)
                <app-user-edit></app-user-edit>
            </div>
        </div>
    </div>
</template>
```

Communication Problems

In **User** we add a *button*, some data, and methods to change user name. Name is initiailized as 'Dome' and will be changed to 'Anna' if clicking the button. However, we want the 'name' to be display in **UserDetail** component. **How do we pass the name to the child component?**

```
<template>
    <div class="component">
        <h1>The User Component</h1>
        <p>I'm an awesome User!</p>
        <button @click="changeName">Change my
name</button>
        <hr>
        <div class="row">
            <div class="col-xs-12 col-sm-6">
                <app-user-detail v-bind:name="name">
</app-user-detail>
            </div>
            <div class="col-xs-12 col-sm-6">
                <app-user-edit></app-user-edit>
            </div>
        </div>
    </div>
</template>

<script>
    import UserDetail from './UserDetail.vue';
    import UserEdit from './UserEdit.vue';
```

```
    export default {
        data: function() {
            return {
                name: 'Dome'
            }
        },
        methods: {
            changeName() {
                this.name = 'Anna'  // want to output
in UserDetail
            }
        },
        components: {
            appUserDetail: UserDetail,
            appUserEdit: UserEdit
        }
    }
</script>
```

To pass a data from parent component, we need to *bind a variable (or user-defined attribute) with the parent's data property* that we want to pass in the client's placeholder. For example, we can bind **uName** attribute with **name** data property of *User component* and pass it to *UserDetail* component as following:

```
// User.vue: passing uName attribute to UserDetail
component
<app-user-detail v-bind:uName="name"></app-user-
detail>
```

On the client side, we can use **props** property to access the passing data. **Props** is an array of properties that will be passed from outside. We add **props** properties in child component's instance. We can get a data

by referring to the list of **attributes**, e.g. ['data1','data2']. **Probs** can be access in the child component as if they are data properties of the child.

```
// UserDetail.vue
<template>
    <div class="component">
        <h3>You may view the User Details here</h3>
        <p>Many Details</p>
        <p>User Name: {{ uName }}</p>
//interpolation
    </div>
</template>

<script>
  export default {
    props: ['uName'],   //using props property
referring to the 'uname' attribute
                        //uName can be used as normal
data property
    methods: {
      switchName() {
        return
this.uName.split("").reverse().join("");
      }
    }
  }
</script>
```

Validation probs

We can use **probs object** to validate their data. If the parent pass a number type, we will get a waring *Invalid prop: type check failed...* and the whole component is detached.

```
// UserDetail.vue - add prop validation
```

```
<script>
  export default {
    props: {
      uName: String,
      data1: {
        type: String,
        //required: true,
        default: 'default data'
      },
      data2: {
        type: Object,
        default: function() {
          return {
            text1: 'text1',
            text2: 'text2
          }
        }
      }
    },
    methods: {
      switchName() {
        return
this.uName.split("").reverse().join("");
      }
    }
  }
</script>
```

Passing event from child to parent using custom event

```
// UserDetail.vue - add prop validation
<template>
    <div class="component">
        <h3>You may view the User Details here</h3>
        <p>Many Details</p>
        <p>User name: {{ uName }}</p>
```

```
        <p>Reversed name: {{ switchName() }}</p>
        <button @click="resetName">Reset
Name</button>
    </div>
</template>

<script>
  export default {
    props: {
      uName: String
    },
    methods: {
      switchName() {
        return
this.uName.split("").reverse().join("");
      },
      resetName() {
        this.uName = 'Max';   //changing String in
child has no effect in the parent
                              //how to inform the
parent the change.
                              //but changing Object
and Array will affect the parent
      }
    }
  }
</script>
```

Changing **uName** String (a primitive type) in child has no effect in the parent but changing **Object and Array** will affect the parent. So how to inform the parent about this primitive change.

```
methods: {
  switchName() {
    return this.uName.split("").reverse().join("");
  },
```

```
  resetName() {
    this.uName = 'Max';
    this.$emit('nameWasReset', this.uName);  //emit
an event
  }
}
```

In the parent, we can listen to event name by using 'v-on' or '@'

```
// User.vue - listent to event fired by its children
<app-user-detail v-bind:uName="name"
@nameWasReset="name = $event"></app-user-detail>
```

```
// User.vue - passing fucntion to children is
possible
<app-user-detail
    v-bind:uName="name"
    @nameWasReset="name = $event"
    v-bind:resetFn="resetName">
</app-user-detail>
```

```
// UserDetail.vue - add prop validation
<template>
    <div class="component">
        <h3>You may view the User Details here</h3>
        <p>Many Details</p>
        <p>User name: {{ uName }}</p>
        <p>Reversed name: {{ switchName() }}</p>
        <button @click="resetName">Reset Name (custom
event)</button>
        <button @click="resetFn">Reset Name (callback
function)</button>
```

```
    </div>
</template>

<script>
  export default {
    props: {
      uName: String,
      resetFn: Function
    },
    methods: {
      switchName() {
        return
this.uName.split("").reverse().join("");
      },
      resetName() {
        // change only in child component
        this.name = "Dome Potikanond"
        // emit a custom event, sent data back
        this.$emit('nameWasReset', this.name);
      }
    }
  }
</script>
```

Children components can not communicate among one another directly but can be done via the parent.

Communicating with Callbacks function

Create a **callback function** as a method in the parent component. Then pass the **reference of the function** to the client component.

```
// User.vue: the parent
<template>
...
<app-user-detail
```

```
    v-bind:uName="name"
    @nameWasReset="name = $event"
    :resetFn="resetName"
    :userAge="age"></app-user-detail>
...
</template>

<script>
    ...
    methods: {
        ...
        // a callback function
        resetName() {
            // change only in child component
            this.name = "Dome Potikanond"
            // emit a custom event
            this.$emit('nameWasReset', this.name);
        }
    }
    ...
</script>
```

The client component get the function reference via the **props** then execute this method (which is defined in the parent) when the **passing data** is updated in the client.

```
//UserDetail.vue
<template>
...
<button @click="resetFn()">Reset name - Callback
Fn</button>

...
</template>
<script>
    export default {
```

```
        props: {
            uName: {
                type: String,
                default: 'John Doe'
            },
            resetFn: Function,
            userAge: Number
        },
        data: function() {
            return {
                name: this.uName,
                age: this.userAge
            }
        },
        methods: {
            switchName() {
                // split, reverse, and join
                return
this.name.split("").reverse().join("");
            },
            resetName() {
                // change only in child component
                this.name = "Dome Potikanond"
                // emit a custom event
                this.$emit('nameWasReset',
this.name);
            }
        }
    }
</script>
```

Communication between sibling

Exchange data between siblings has to be done through their parent only. This can be done using the combination of previous methods

Using Event Bus for communication

Using a central class (or object) to pass data (aka Services in Angular2) between sibling directly. Firstly, create a new **constant object of Vue instance** in the **main.js**.

```
import Vue from 'vue'
import App from './App.vue'

// 1. create a constant VueJS instance as a central
object
// All necessary methods to pass data are already
there
export const eventBus = new Vue();

new Vue({
  el: '#app',
  render: h => h(App)
})
```

Secondly, in the sender sibling component, using the **eventBus** to emit to pass data.

```
...
    methods: {
        editAge() {
            this.age += 2;
            // this.$emit('ageEdited', this.age);

            // using central object to pass data
instead
            eventBus.$emit('ageEdited',this.age);
        },
        ...
    }
...
```

Thirdly, in the receiver sibling component, using the **created** life cycle hook property to create an **eventBus's listener** that listens to the **passing data** event. Define a callback function to handle the **received data**.

```
...
    created() {
        // create a listner which run since the
instance is created
        eventBus.$on('ageEdited', (age)=> {
            this.userAge = age;
        })
    }
...
```

Passing data between parent-child and siblings using above methods is fine for small to medium sized application. There is also a better method that is more suitable for complex application.