

# Gin Gonic/ Vue.js

Presentation of the two stunning Webframeworks in  
GoLang & JavaScript

by Alex Schübl, David Bochan, Nadin-Katrin Apel  
in SS 2018



# Table of Contents

1

Project setup: separation of backend & fronted

2

Why use a framework?

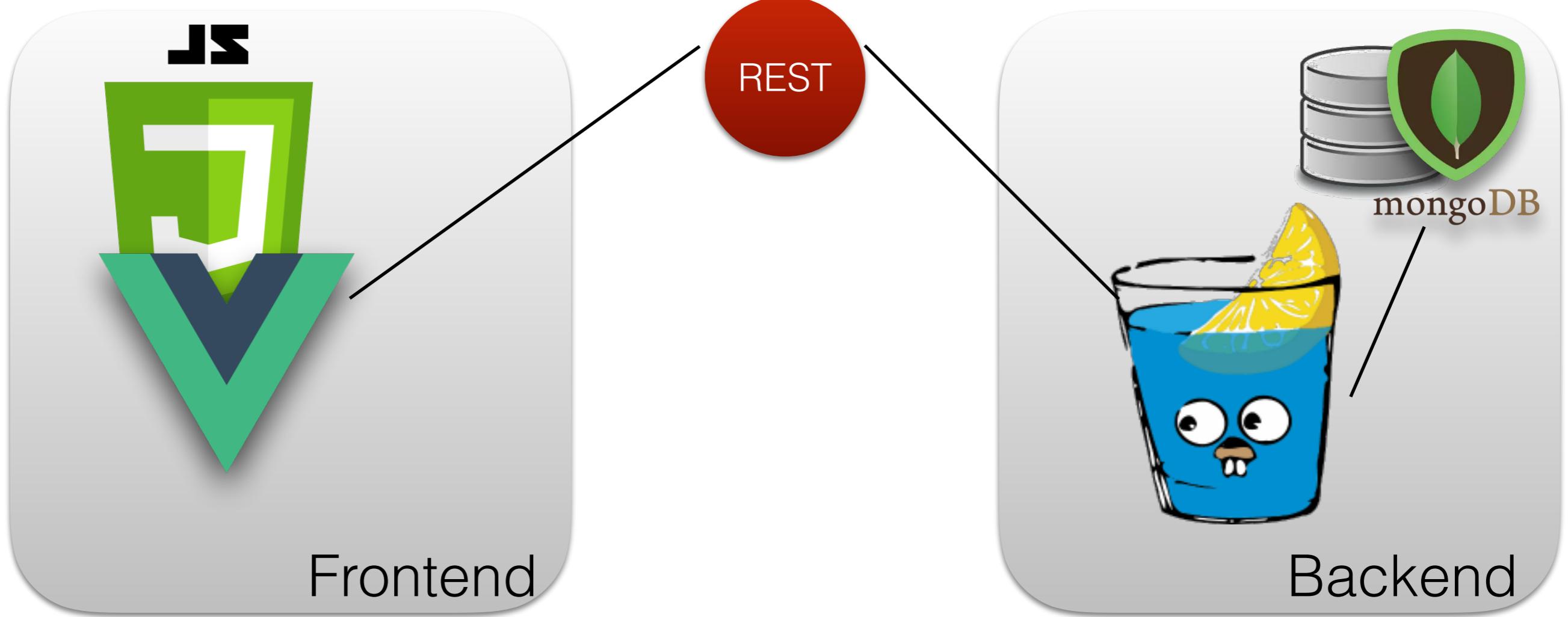
3

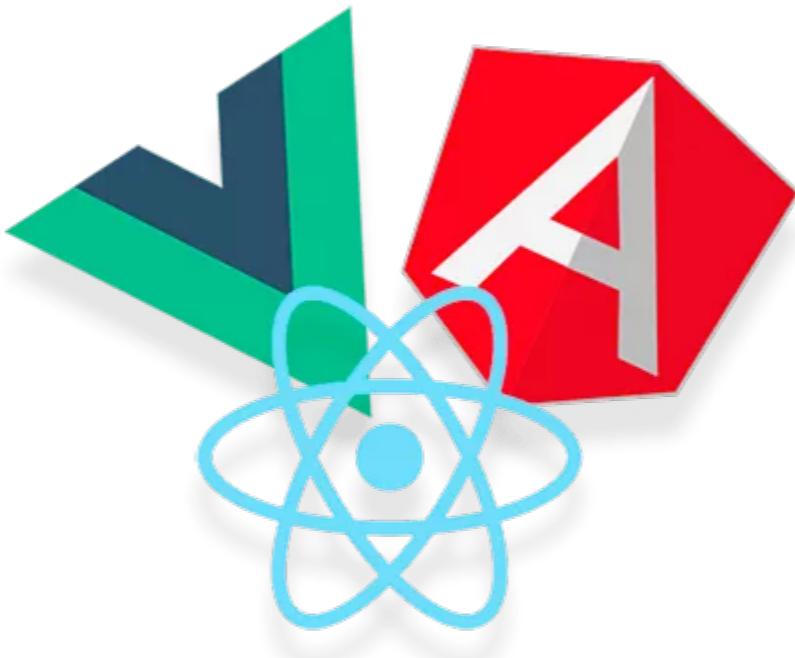
Main Concepts of Vue.js

4

Main Concepts of Gin Gonic

# Separation of Backend/ Frontend





# JS Framework - why ?

---

# HTML pages are built with complex markup

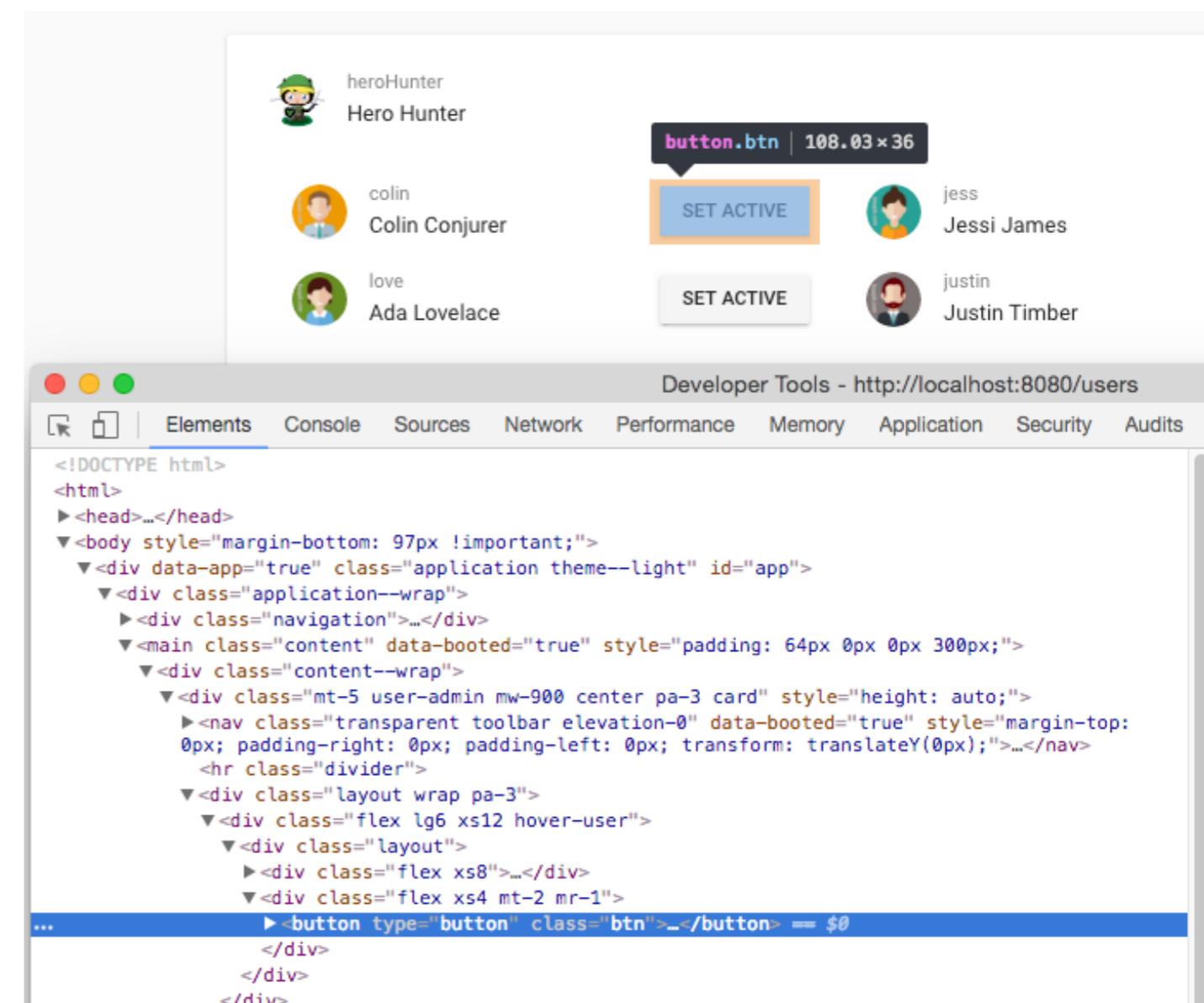
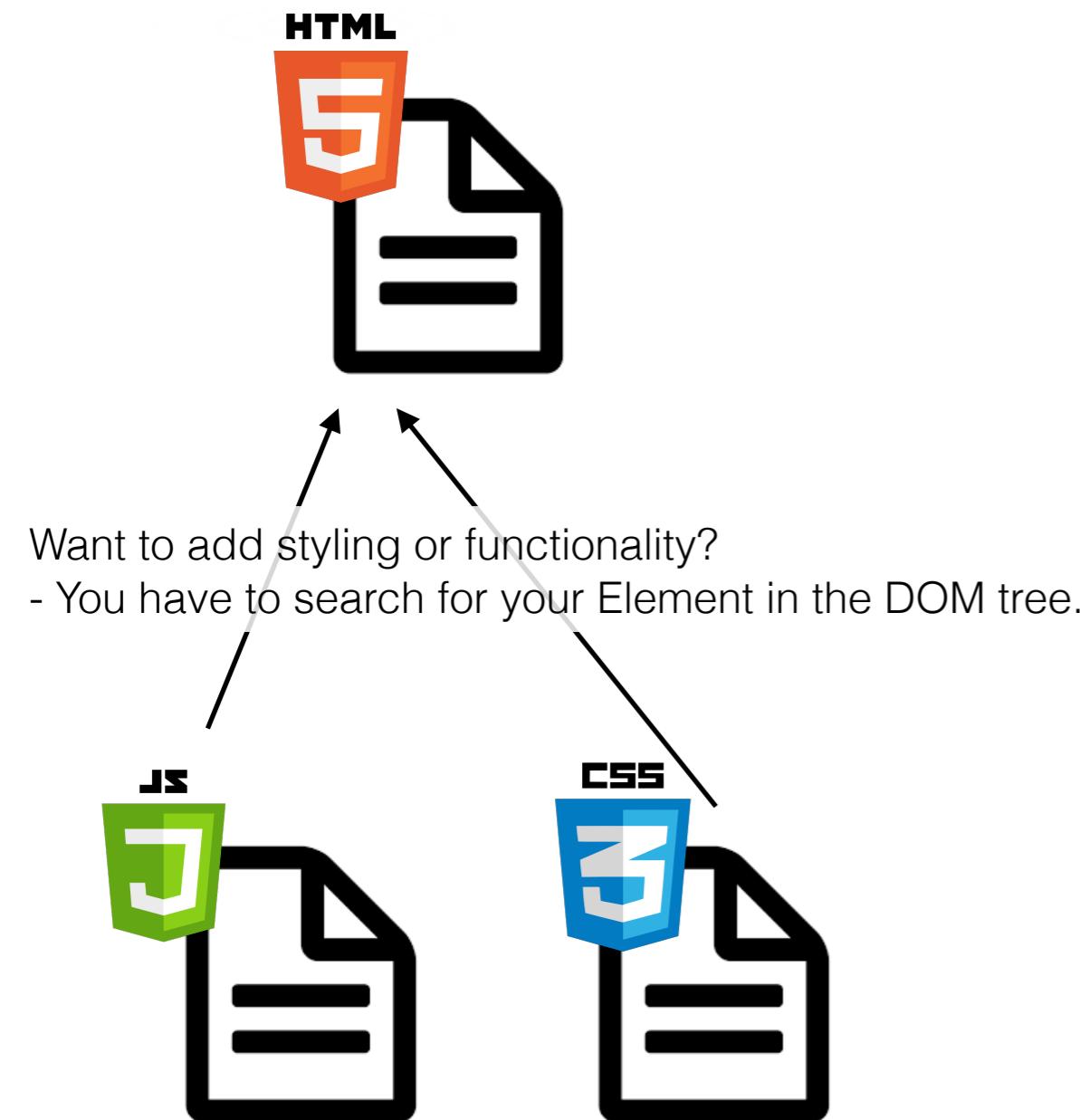
```
.mt-5.user-admin.mw-900.center.pa-3.card 36
|.list.pa-0.transparent 38
|.toolbar__content 39
|  .list__tile.list__tile--avatar 40
|  |.list__tile_avatar.avatar 41
|  |.list__tile_sub-title 42 heroH...
|  |.list__tile_title 43 Hero Hun...
.layout.wrap.pa-3 46|.flex.lg6.xs12.hover-user 47
|  .layout.flex.xs8 48
|  |.list.pa-0.transparent 49
|  ||.toolbar__content 50
|  | |.list__tile.list__tile--avatar 51
|  | | |.list__tile_avatar.avatar 52
|  | | |.list__tile_sub-title 53 colin
|  | | |.list__tile_title 54 Colin C...
|.flex.lg6.xs12.hover-user 55
|.list.pa-0.transparent 56
|.toolbar__content 57
|.list__tile.list__tile--avatar 58
|.list__tile_sub-title 59 collin
|.list__tile_title 60 Colin C...
.layout.flex.xs8 61
|.list.pa-0.transparent 62
|.toolbar__content 63
|.list__tile.list__tile--avatar 64
|.list__tile_sub-title 65 jessi
|.list__tile_title 66 Ada Lo...
.layout.flex.xs8 67
|.list.pa-0.transparent 68
|.toolbar__content 69
|.list__tile.list__tile--avatar 70
|.list__tile_sub-title 71 Justin
|.list__tile_title 72 Jessie ...
.layout.flex.xs8 73
|.list.pa-0.transparent 74
|.toolbar__content 75
|.list__tile.list__tile--avatar 76
|.list__tile_sub-title 77
|.list__tile_title 78 SET ACTIVE
.layout.flex.xs8 79
|.list.pa-0.transparent 80
|.toolbar__content 81
|.list__tile.list__tile--avatar 82
|.list__tile_sub-title 83
|.list__tile_title 84
.layout.flex.xs8 85
|.list.pa-0.transparent 86
|.toolbar__content 87
|.list__tile.list__tile--avatar 88
|.list__tile_sub-title 89
|.list__tile_title 90 JUSTIN
.layout.flex.xs8 91
|.list.pa-0.transparent 92
|.toolbar__content 93
|.list__tile.list__tile--avatar 94
|.list__tile_sub-title 95
|.list__tile_title 96 JUSTIN
.layout.flex.xs8 97
|.list.pa-0.transparent 98
|.toolbar__content 99
|.list__tile.list__tile--avatar 100
|.list__tile_sub-title 101
|.list__tile_title 102 JUSTIN
```

# Frameworks Add Structure

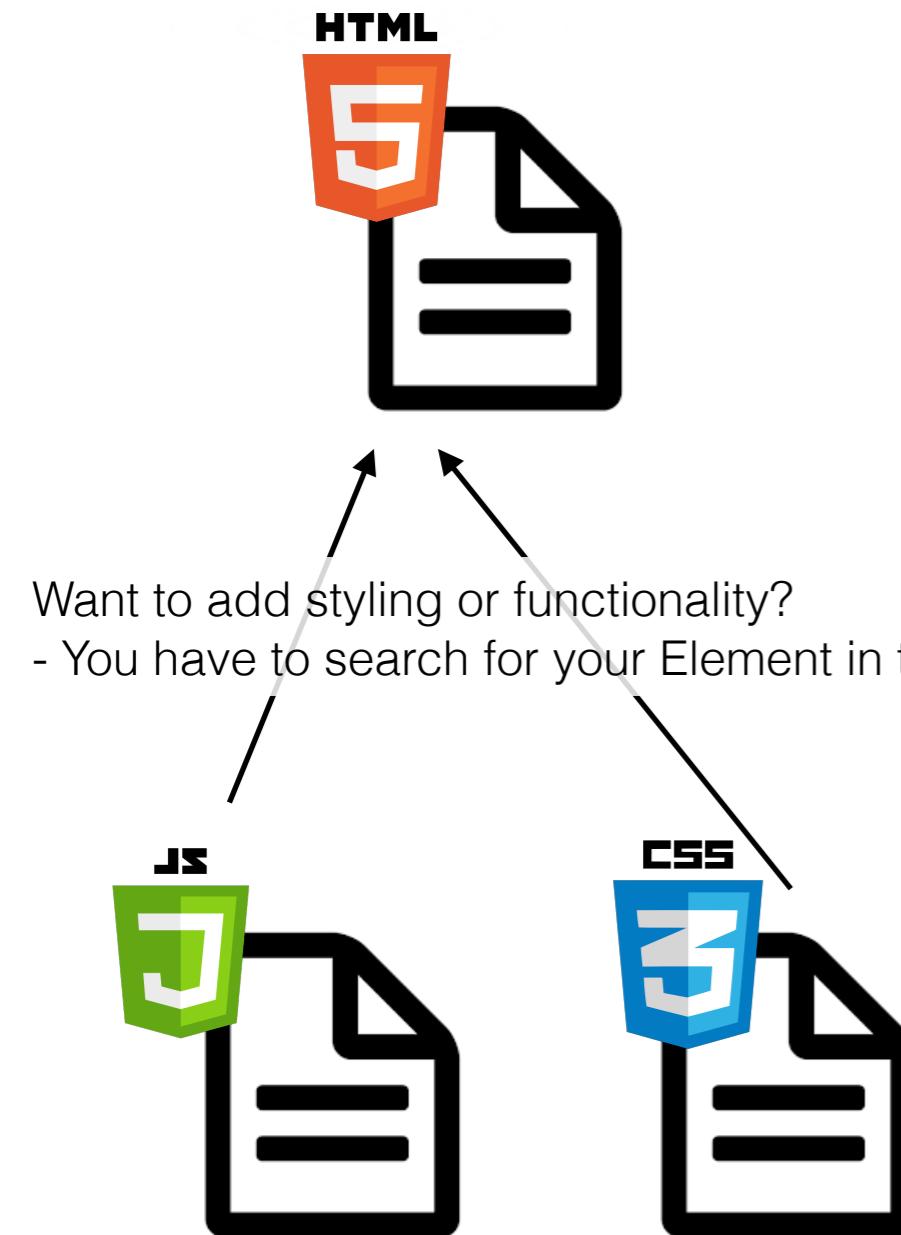
---



# Optimal traversing through the HTML DOM is a hard job



# Optimal traversing through the HTML DOM is a hard job



Developer Tools - http://localhost:8080/users

```
<main class="content" data-booted="true" style="padding: 64px 0px 0px 300px;">
  <div class="content--wrap">
    <div class="mt-5 user-admin mw-900 center pa-3 card" style="height: auto;">
      <nav class="transparent toolbar elevation-0" data-booted="true" style="margin-top: 0px; padding-right: 0px; padding-left: 0px; transform: translateY(0px);">...</nav>
      <hr class="divider">
      <div class="layout wrap pa-3">
        <div class="flex lg6 xs12 hover-user">
          <div class="layout">
            <div class="flex xs8">...</div>
            <div class="flex xs4 mt-2 mr-1">
              <button type="button" class="btn">...</button>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</main>
```

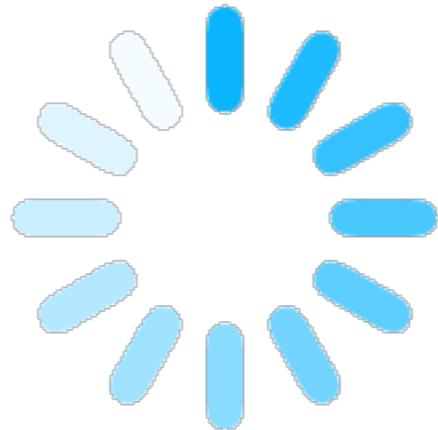
html body div#app.application.theme--light div.application--wrap div.container.app

Console

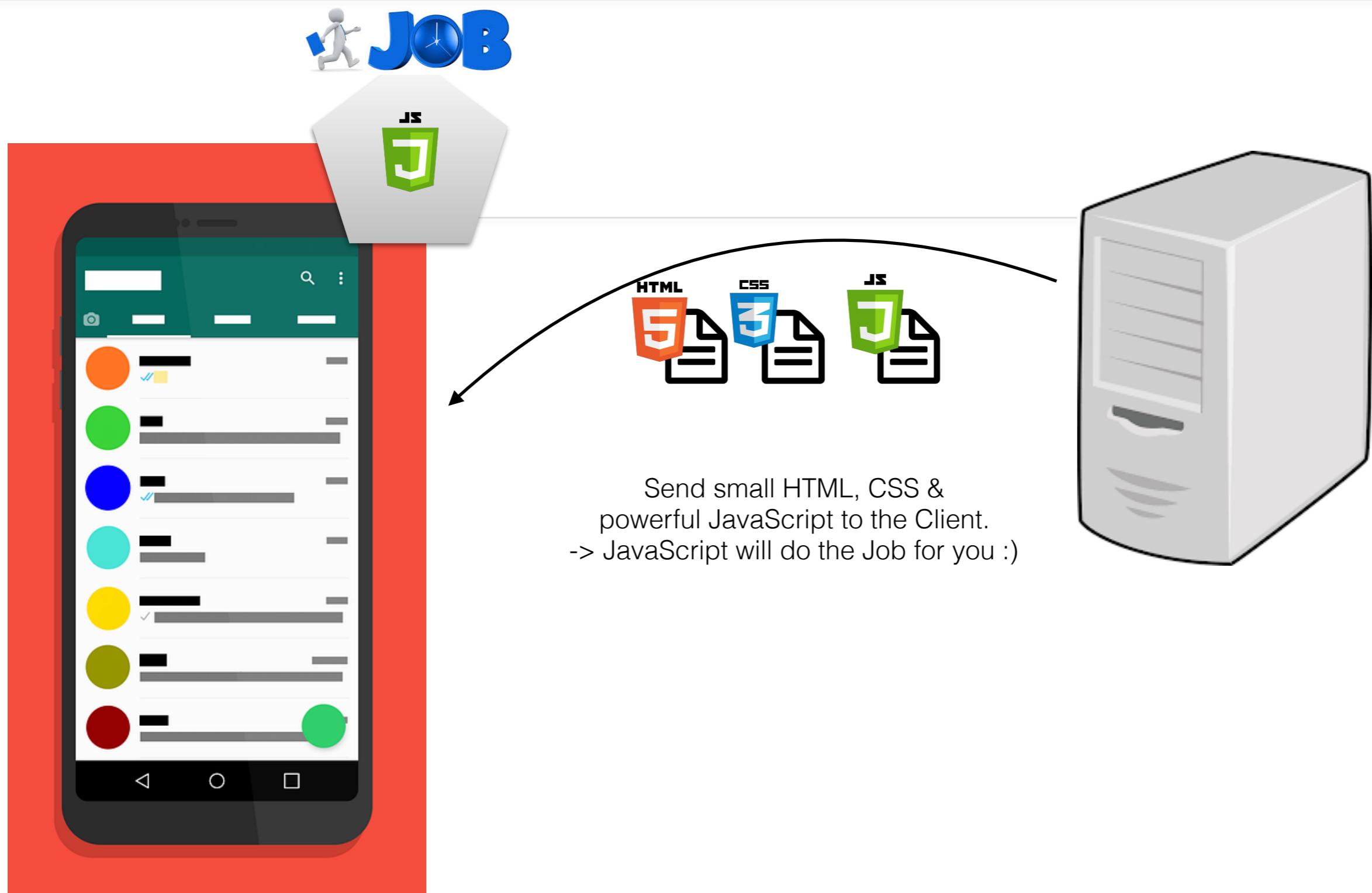
```
> document.getElementsByTagName('button')
< HTMLCollection(8) [button.toolbar__side-icon.btn.btn--icon, button.btn.btn--icon, button.btn.btn--icon, button.btn.btn--icon, button.btn.btn--icon, button.btn.btn--icon, button.btn.btn--icon, button.btn.btn--icon]
  0: button.toolbar__side-icon.btn.btn--icon
  1: button.btn.btn--icon
  2: button.btn.btn--icon
  3: button.btn.btn--icon
  4: button.btn.btn--icon
  5: button.btn.btn--icon
  6: button.btn.btn--icon
  7: button.btn.btn--icon
  length: 8
```

Build web applications with optimal performance & no  
breaks when clicking on a link ?

---



# Single Page Applications May Be Your Answer

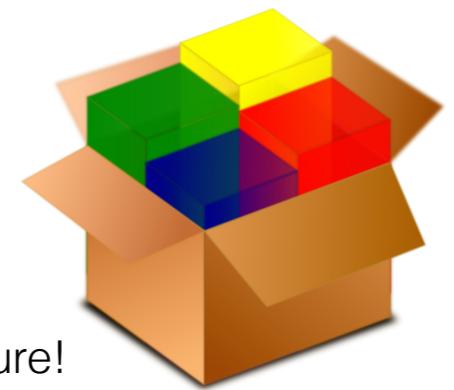


# Many Script tags for a SPA are devastating

---

```
<script type="text/javascript" src=""></script>
```

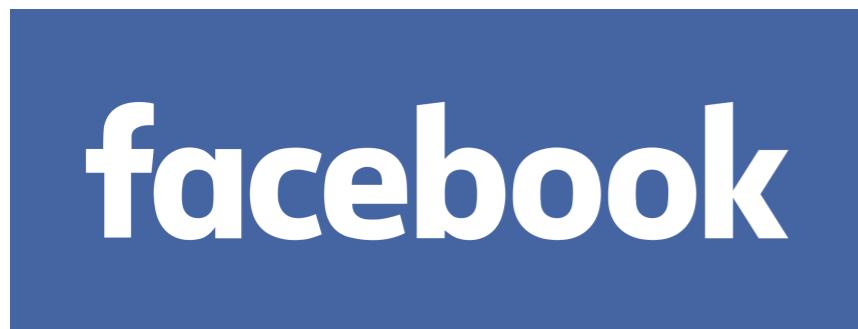
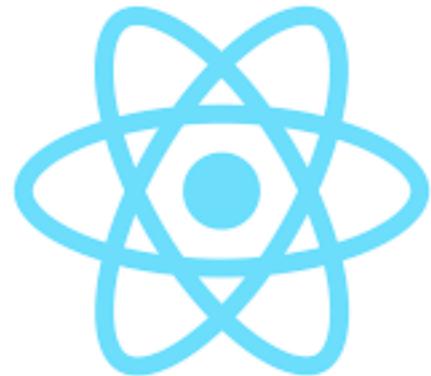
Again, Web Frameworks help you with that. 😊



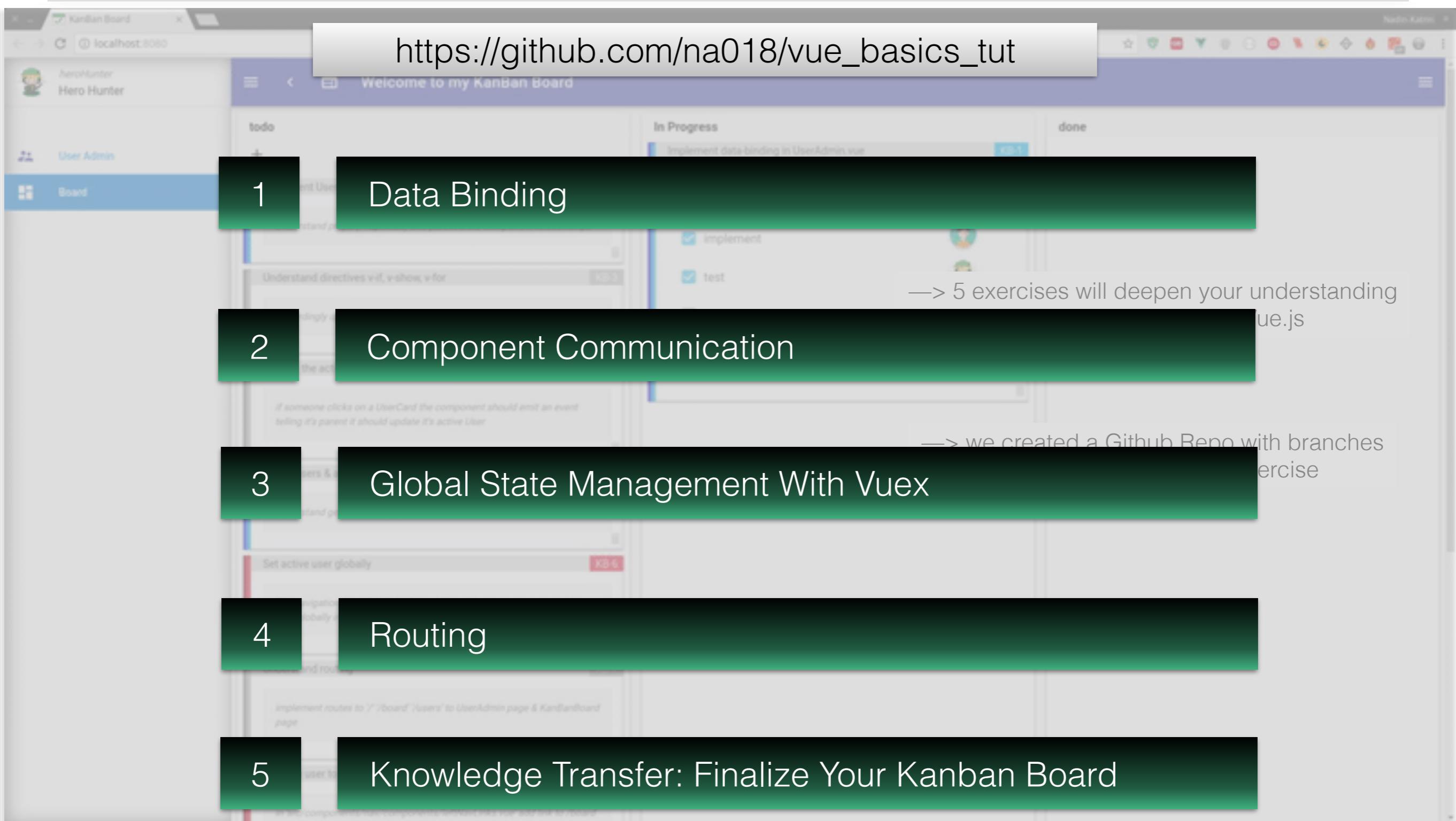
-> Think about structure!

Now let's compare the today most combating JS Frameworks

---



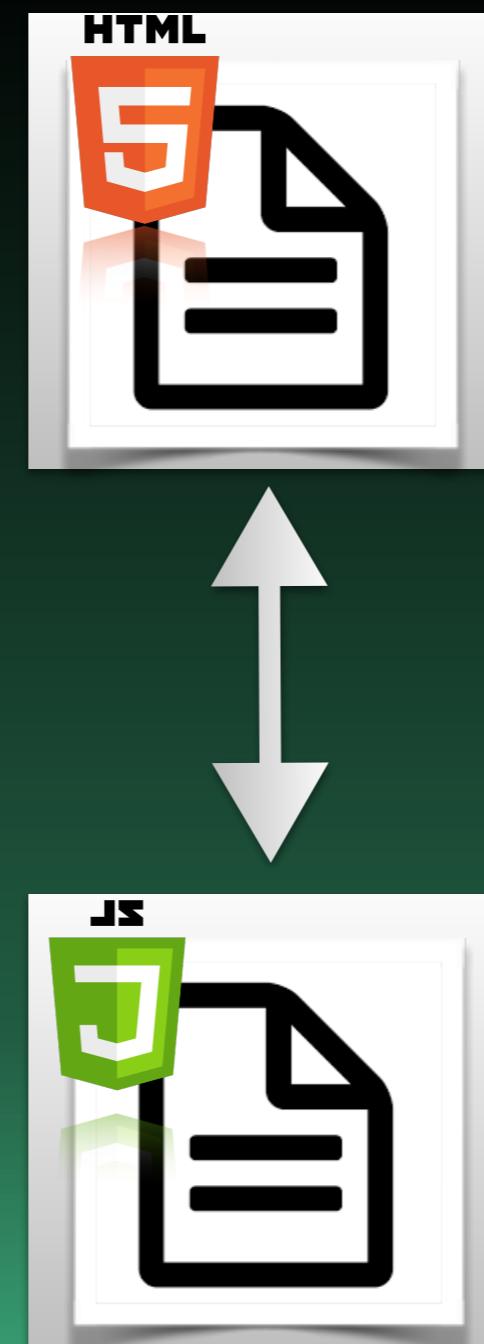
# Vue.js in Depth: Our Example Application



# Project Structure

vue_basics_tut ~/CodeProjects/vue_basics_tut	
> build	# webpack config files
> config	# main project config
> node_modules	library root # used node modules
> server	# json mock server with api
< src	# app
> components	# ui components
> plugins	# some used plugins ( <i>e.g. logger for vuex/ url for backend</i> )
> router	# application routing
> store	# vuex store ( <i>global application state management</i> )
❖ .gitrepo	
❖ App.vue	# main app component
❖ main.js	# app entry file ( <i>here does it start</i> )
> static	# pure static assets ( <i>images &amp; styles</i> )
< test	
> e2e	# end to end tests
> unit	# unit tests
❖ .babelrc	# babel config
❖ .postcssrc.js	# postcss config
❖ index.html	# only send html page
❖ package.json	# build scripts & dependencies
❖ package-lock.json	
❖ README.md	
❖ server.js	# mock server init script

# 1. Data Binding in Vue.js



# 1. Data Binding

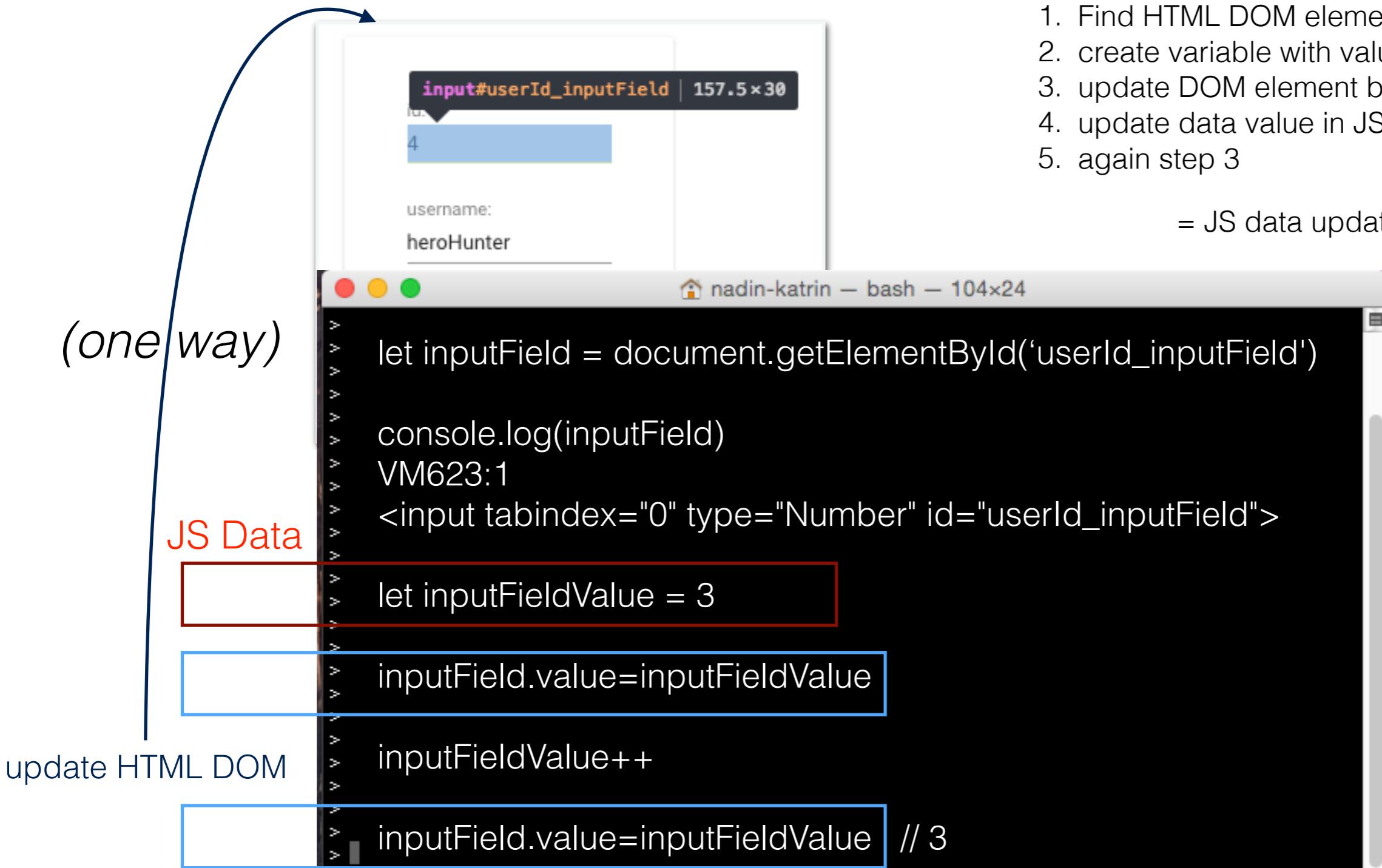
The diagram illustrates the process of data binding between JavaScript and the DOM. On the left, a screenshot of a browser's element inspector shows an input field with the ID 'userId\_inputField' containing the value '4'. To the right, a terminal window titled 'nadin-katrin — bash — 104x24' displays the following JavaScript code:

```
> let inputField = document.getElementById('userId_inputField')
>
> console.log(inputField)
VM623:1
<input tabindex="0" type="Number" id="userId_inputField">
>
> let inputFieldValue = 3
>
> inputField.value=inputFieldValue
>
> inputFieldValue++ // 4
>
> inputField.value=inputFieldValue
```

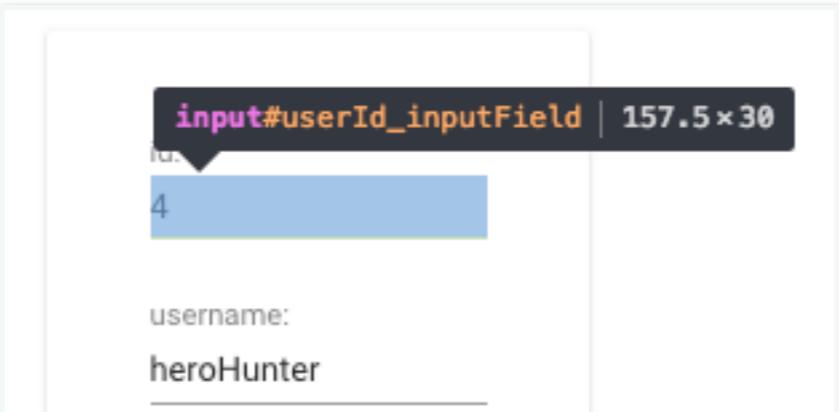
Annotations highlight specific parts of the code:

- A red box labeled "JS Data" highlights the line `let inputFieldValue = 3`.
- A blue box labeled "update HTML DOM" highlights the line `inputField.value=inputFieldValue`.
- A purple box highlights the line `inputFieldValue++ // 4`.
- A blue box highlights the final line `inputField.value=inputFieldValue`.
- A curved arrow points from the "JS Data" annotation to the input field in the browser screenshot.
- A text label `= JS data update -> update DOM` is positioned next to the browser screenshot.

# 1. Data Binding



# 1. Data Binding in Vue: v-bind (*one way*)



The screenshot shows a browser window with a light gray header. In the header, there is a button labeled "input#userId\_inputField" and its dimensions "157.5 x 30". Below the header, there is a blue input field containing the number "4". Underneath the input field, there is a text area with the placeholder "username:" followed by the text "heroHunter".



The terminal window has a title bar "nadin-katrin — bash — 104x24". The code in the terminal is:

```
<template>
  <input type="Number" id="userId_inputField" v-bind="inputFieldValue">
</template>

<script>
  export default {
    name: "test",
    data () {
      export {
        inputFieldValue: 3
      }
    }
  }
</script>
```

A red rectangle highlights the line "v-bind="inputFieldValue">". Another red rectangle highlights the line "inputFieldValue: 3".

-> *v-bind* value:  
when data updated in Vue, HTML DOM  
element will automatically update

# 1. Data Binding in Vue: v-bind (*one way*) -> Exercise 1

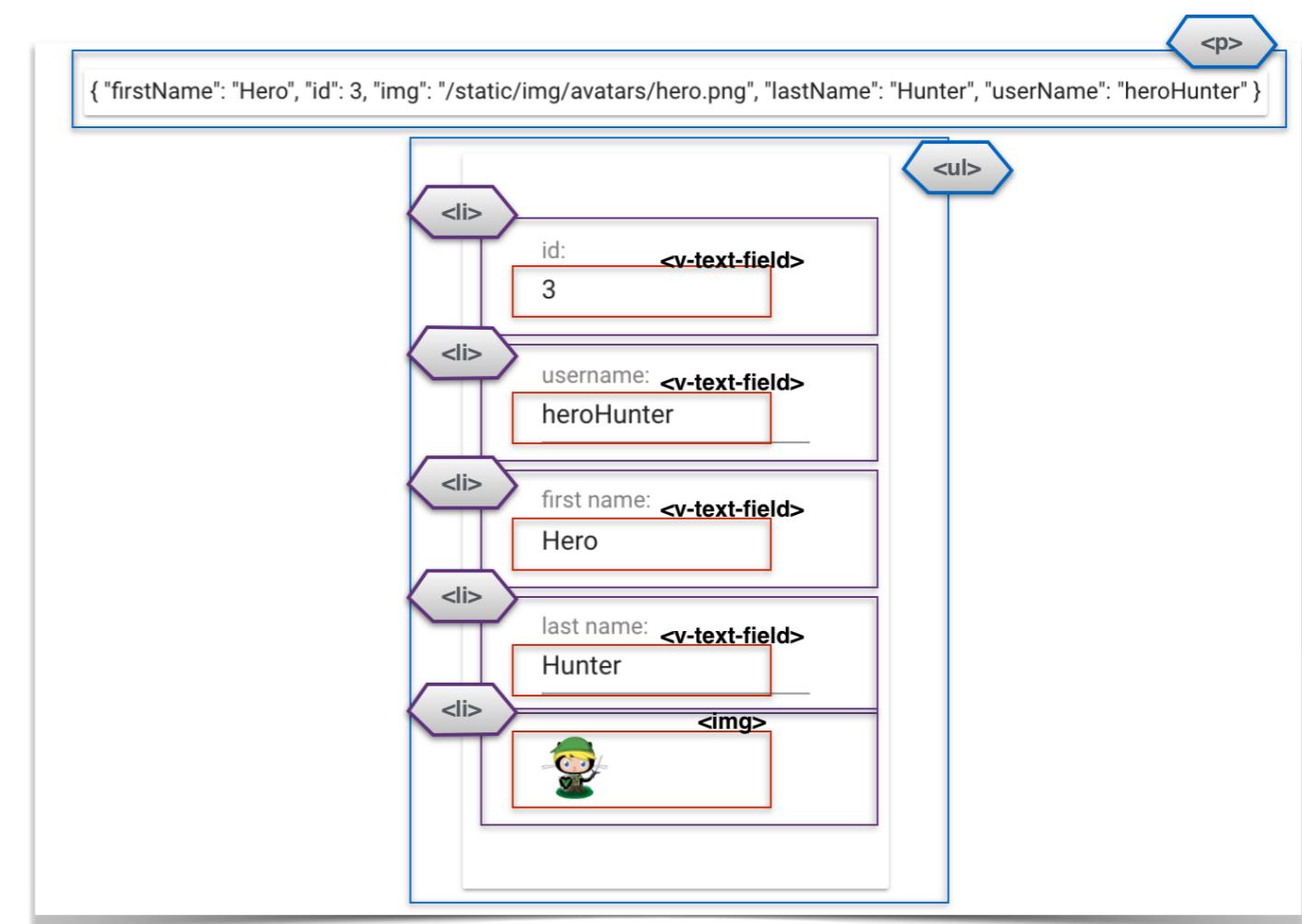
In src/components/pages/UserAdmin.vue:

- create onePerson Object with userName, firstName, lastName, img as Object attributes (in <script> data())
- display the onePerson Object in a paragraph <p> element (in <template>)
- create an unordered list <ul> & show all onePerson attributes in a (<v-text-field>|value) and the image (<img>|src) with 1-way data binding (v-bind)

(hints: classes used for styling p.centerElem.card.pa-1, ul.centerElem.card.pa-5.mt-4, img.avatarIcon)

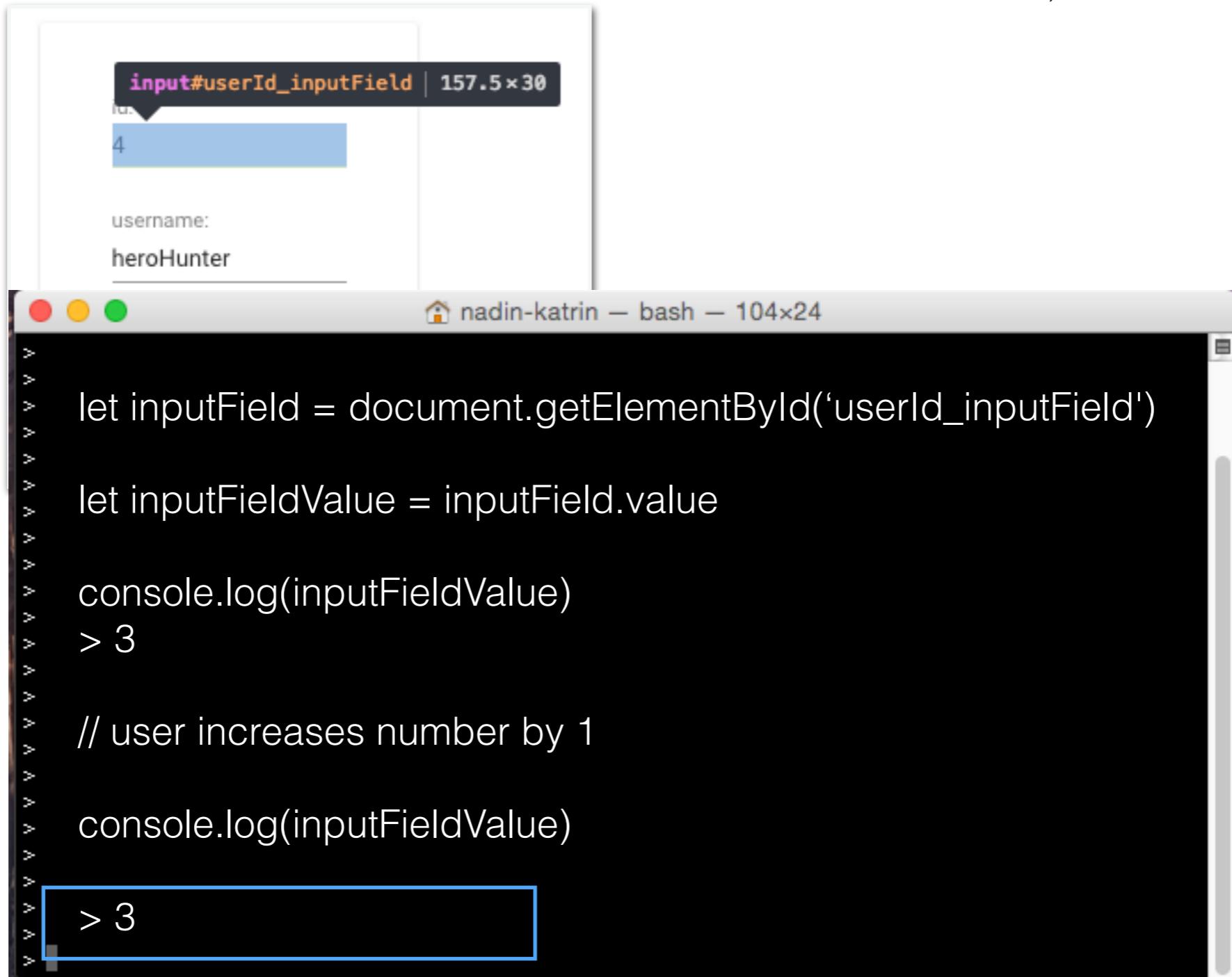
```
nadin-katrin - bash - 104x24
<template>
  <input type="Number" id="userId_inputField" v-bind="inputFieldValue">
</template>

<script>
  export default {
    name: "test",
    data () {
      export {
        inputFieldValue: 3
      }
    }
  }
</script>
```



# 1. Data Binding

User increases Number, but no change in JS Data



The screenshot shows a browser window and a terminal window side-by-side.

In the browser window, there is an input field with the ID "userId\_inputField" containing the value "4". Below the input field, the text "username:" is followed by the value "heroHunter".

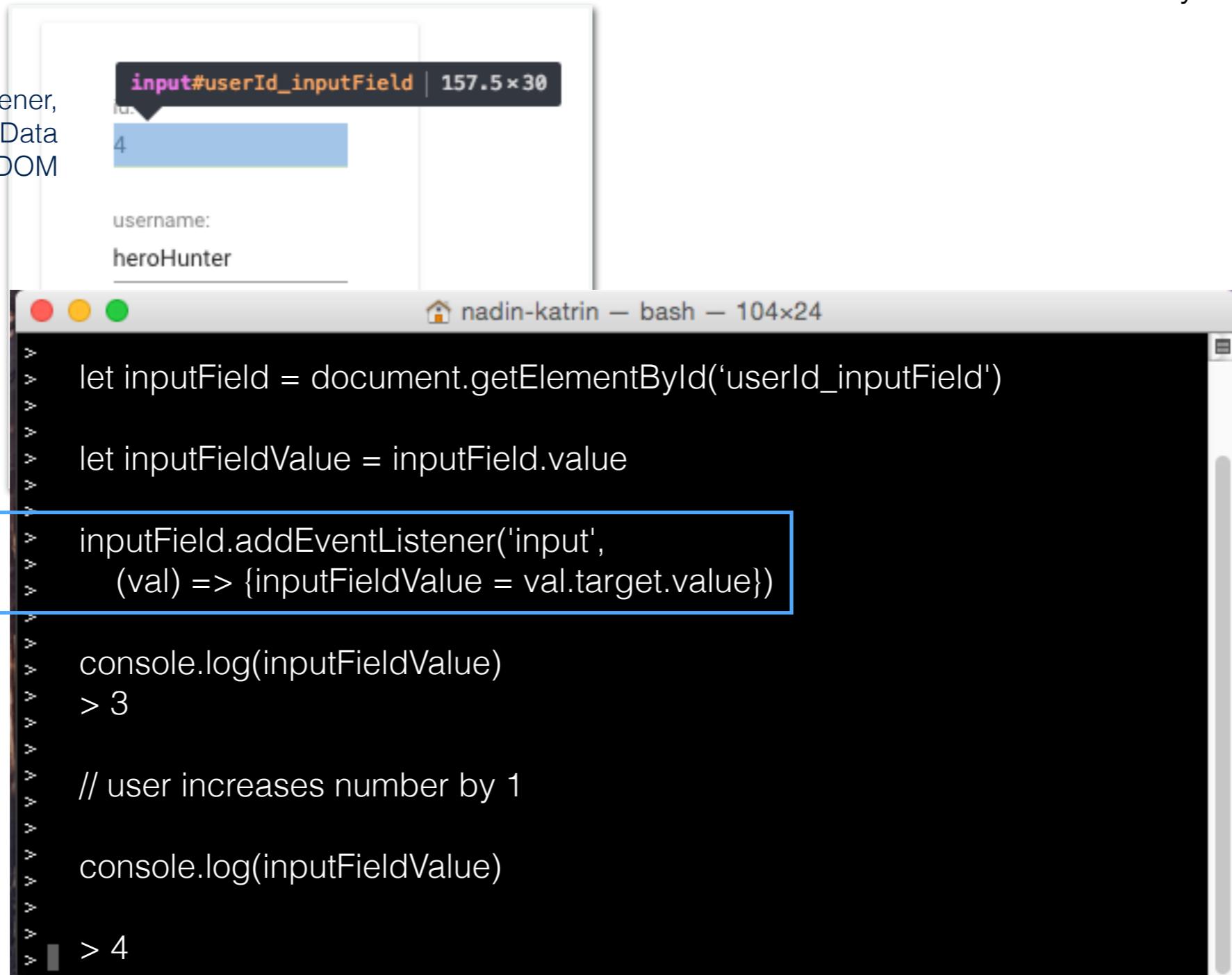
In the terminal window, the command `nadin-katrin — bash — 104x24` is running. The user has typed the following JavaScript code:

```
> let inputField = document.getElementById('userId_inputField')
>
> let inputFieldValue = inputField.value
>
> console.log(inputFieldValue)
> > 3
>
> // user increases number by 1
>
> console.log(inputFieldValue)
>
> > 3
```

The terminal output shows two log statements. The first log statement shows the value as "3", and the second log statement shows the value as "3", indicating that the value did not change despite the user's input.

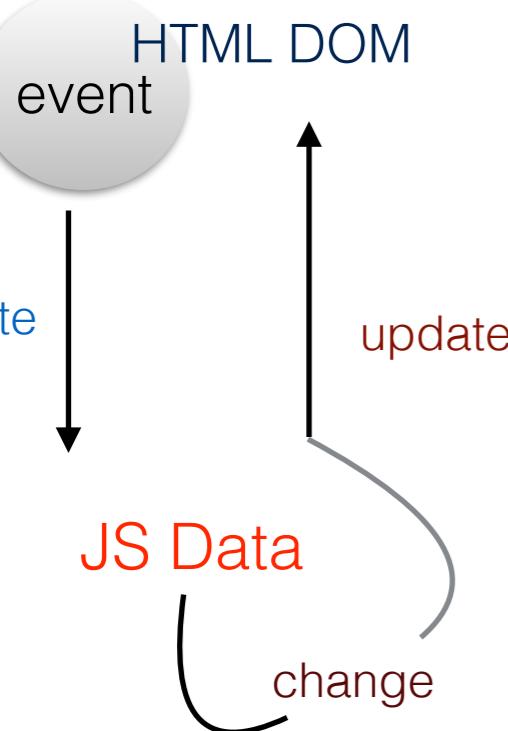
# 1. Data Binding

You need to listen to events that your users call



# 1. Data Binding: *two-way*

1. Listen two events occurring on HTML DOM site
  - update JS Data
2. changes on JS Data site
  - update HTML DOM



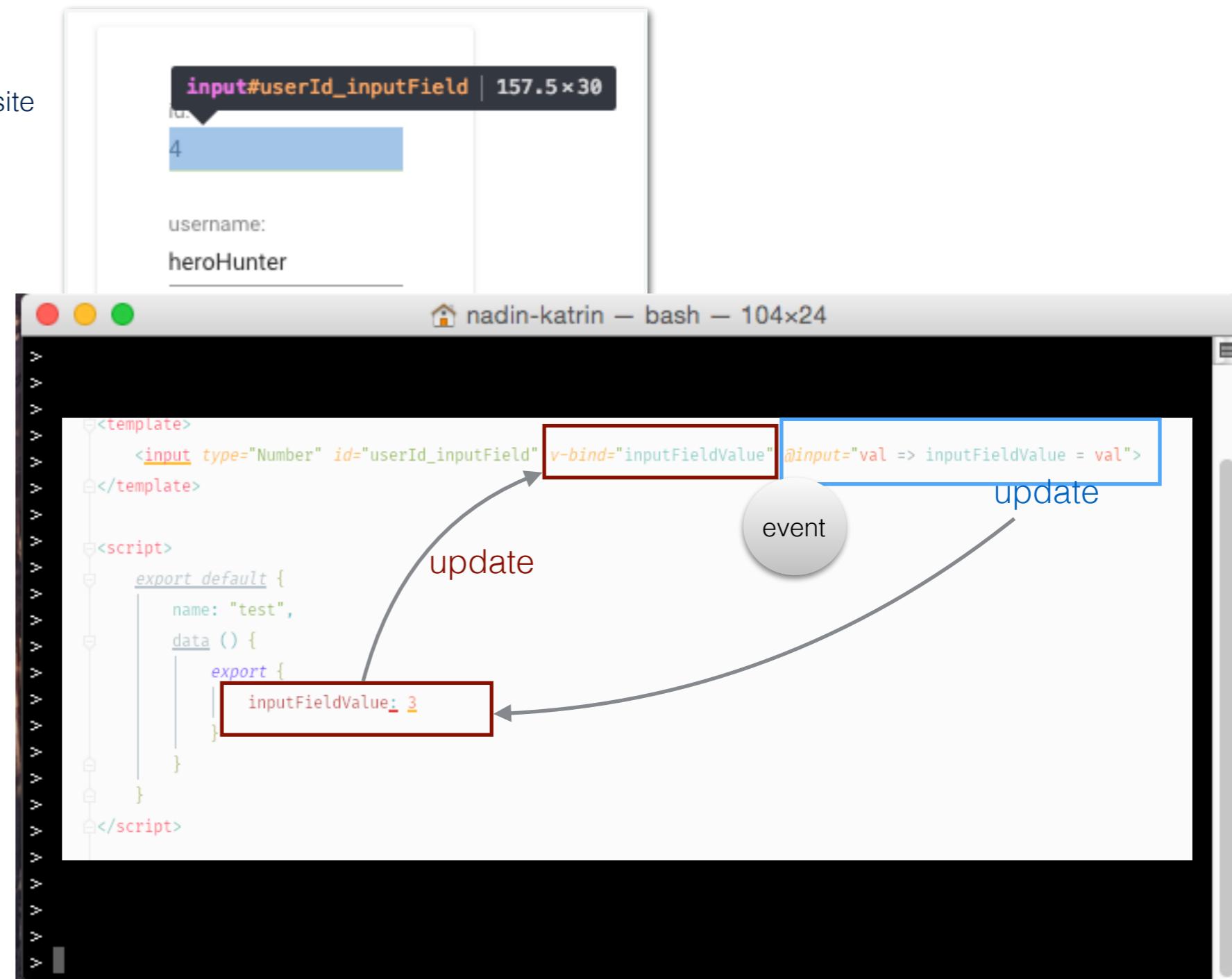
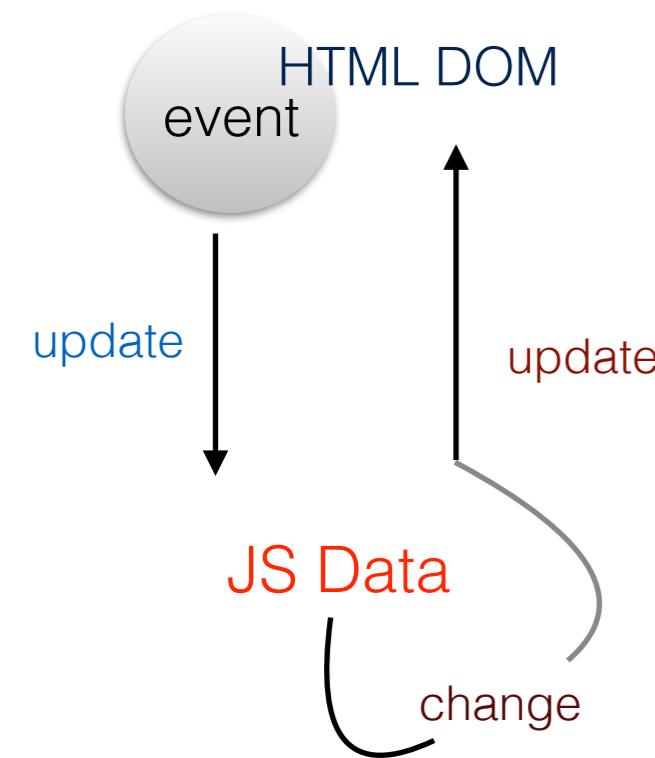
A screenshot of a terminal window titled 'nadin-katrin — bash — 104x24'. The window displays a user interface with an input field containing the value '4' and the text 'username: heroHunter'. Above the input field, a tooltip shows the element as 'input#userId\_inputField | 157.5x30'. The terminal window contains the following code:

```
let inputField = document.getElementById('userId_inputField')
let inputFieldValue = inputField.value
inputField.addEventListener('input',
  (val) => {inputFieldValue = val.target.value})
console.log(inputFieldValue)
> 3
// user increases number by 1
console.log(inputFieldValue)
> 4
inputFieldValue++
inputField.value = inputField
```

Two sections of the code are highlighted: the event listener block (blue box) and the self-updating assignment block (red box).

# 1. Data Binding: two-way in Vue

1. Listen two events occurring on HTML DOM site
  - update JS Data
2. changes on JS Data site
  - update HTML DOM



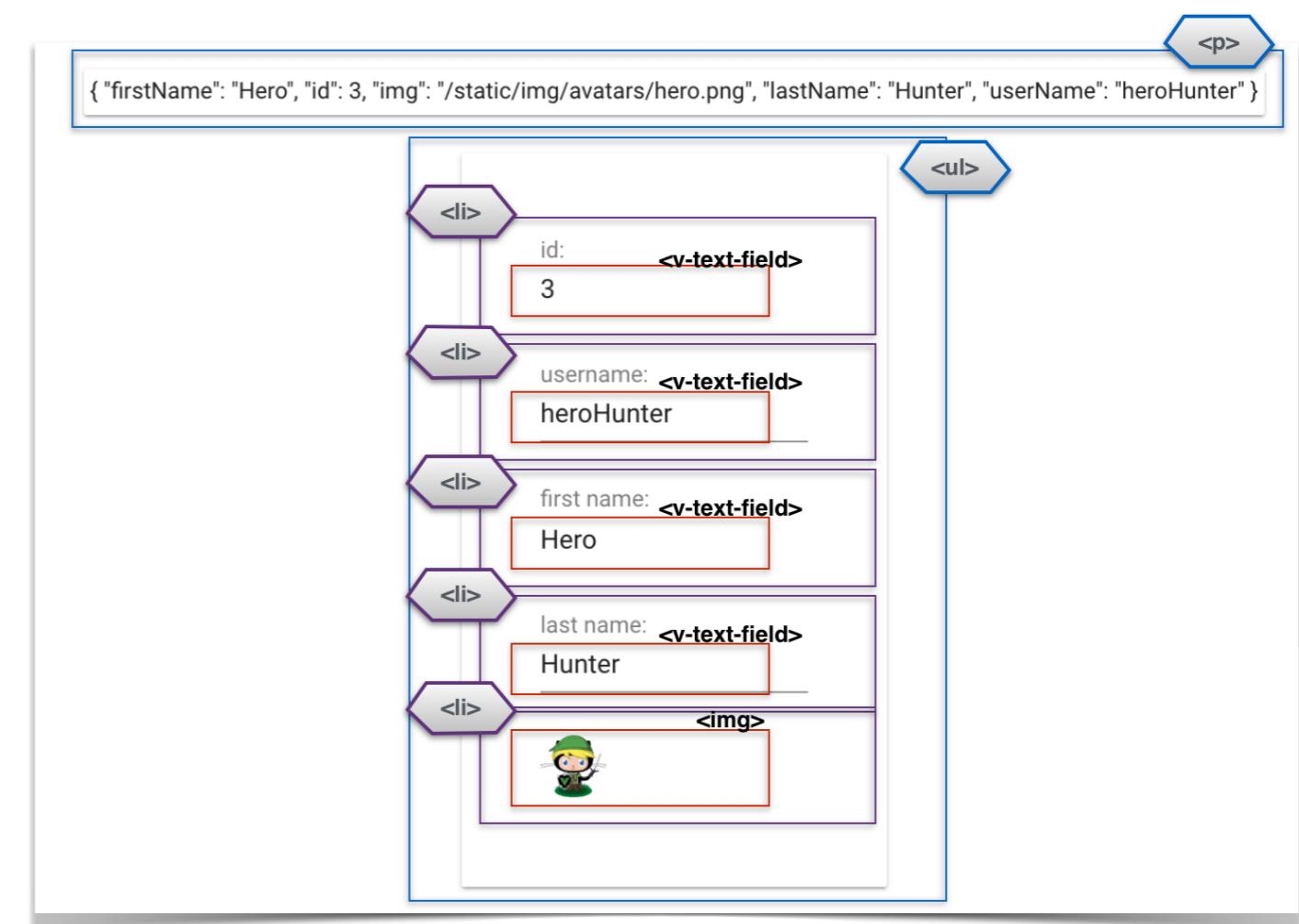
# 1. Data Binding in Vue: (*two way*) -> Exercise 1

In src/components/pages/UserAdmin.vue:

- if a user types into the <v-text-field> the event @input is fired.  
Use it for updating the onePerson's attributes (@input="changedName => onePerson.name = changedName")

```
nadin-katrin - bash - 104x24
<template>
  <input type="Number" id="userId_inputField" v-bind="inputFieldValue" @input="val => inputFieldValue = val">
</template>

<script>
  export default {
    name: "test",
    data () {
      export {
        | inputFieldValue: 3
      }
    }
  }
</script>
```



# 1. Data Binding in Vue: v-model (*two way*) -> Exercise 1

-> v-model makes your life a lot easier

- you don't have to know the name of the v-bind attribute
- you don't need to care about the event that updates your value



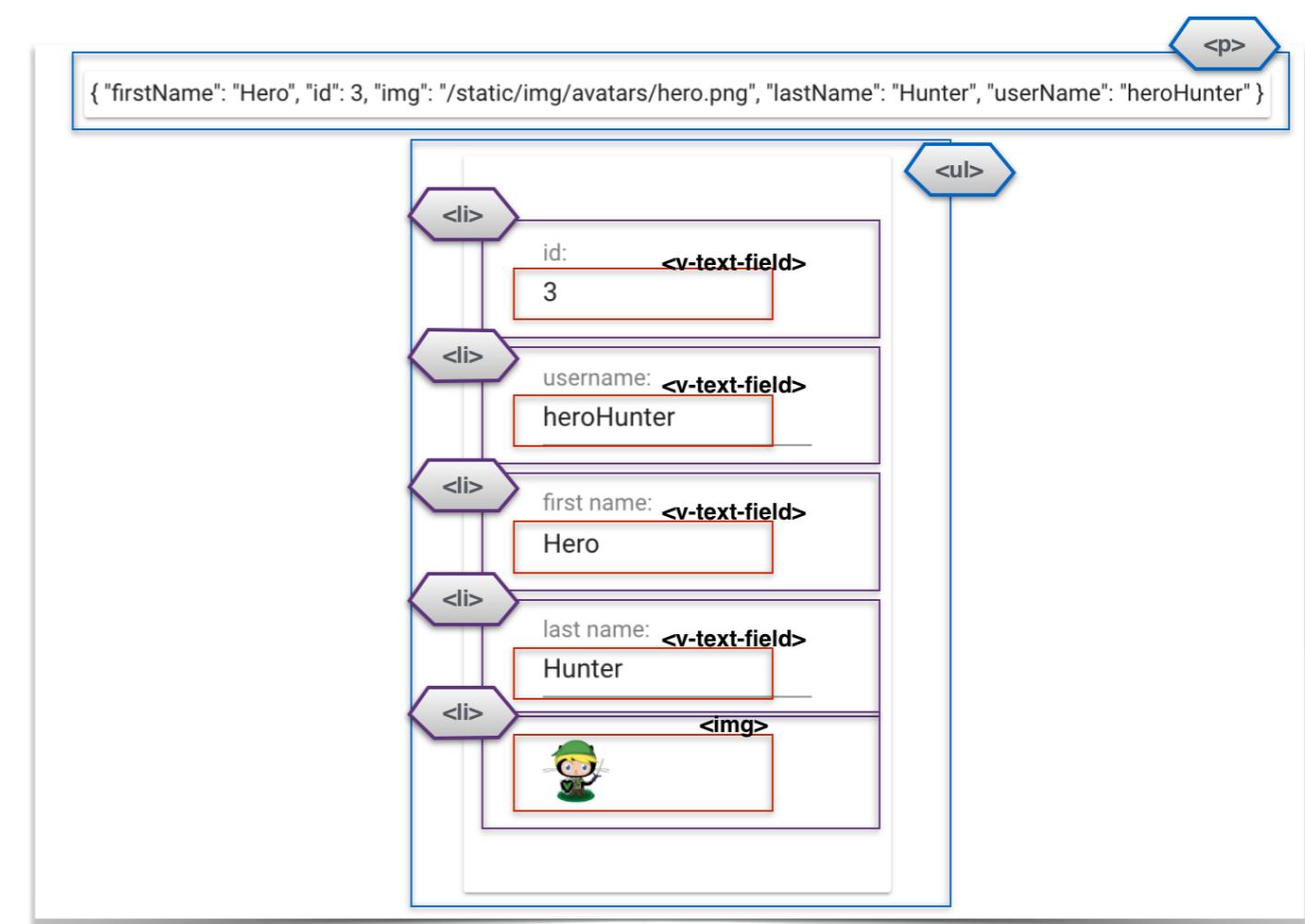
In src/components/pages/UserAdmin.vue:

- replace v-bind by v-model and remove the @input event to understand two-way data binding

```
nadin-katrin - bash - 104x24
<template>
  <input type="Number" id="userId_inputField" v-model="inputFieldValue">
</template>

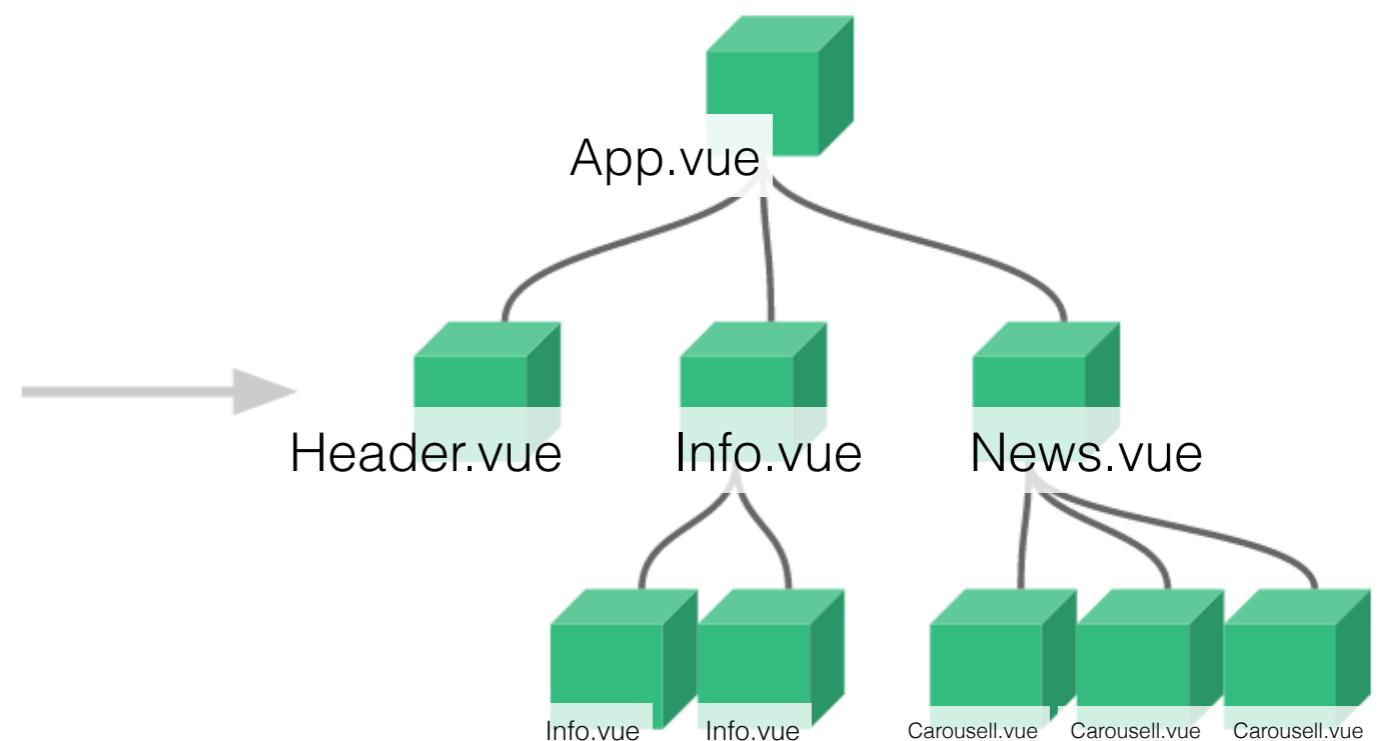
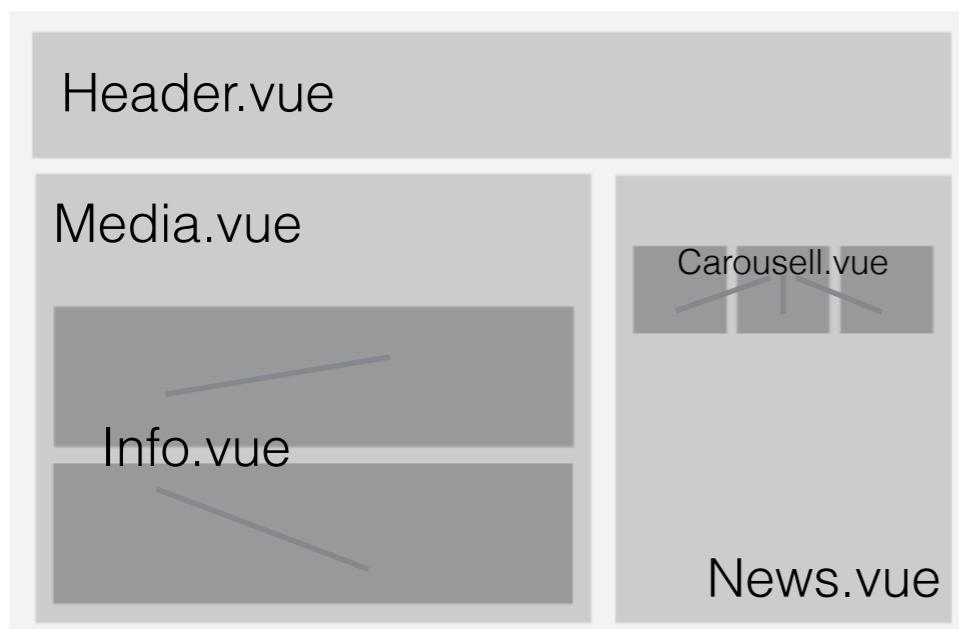
<script>
  export default {
    name: "test",
    data () {
      export [
        inputFieldValue: 3
      ]
    }
  }
</script>
```

update event



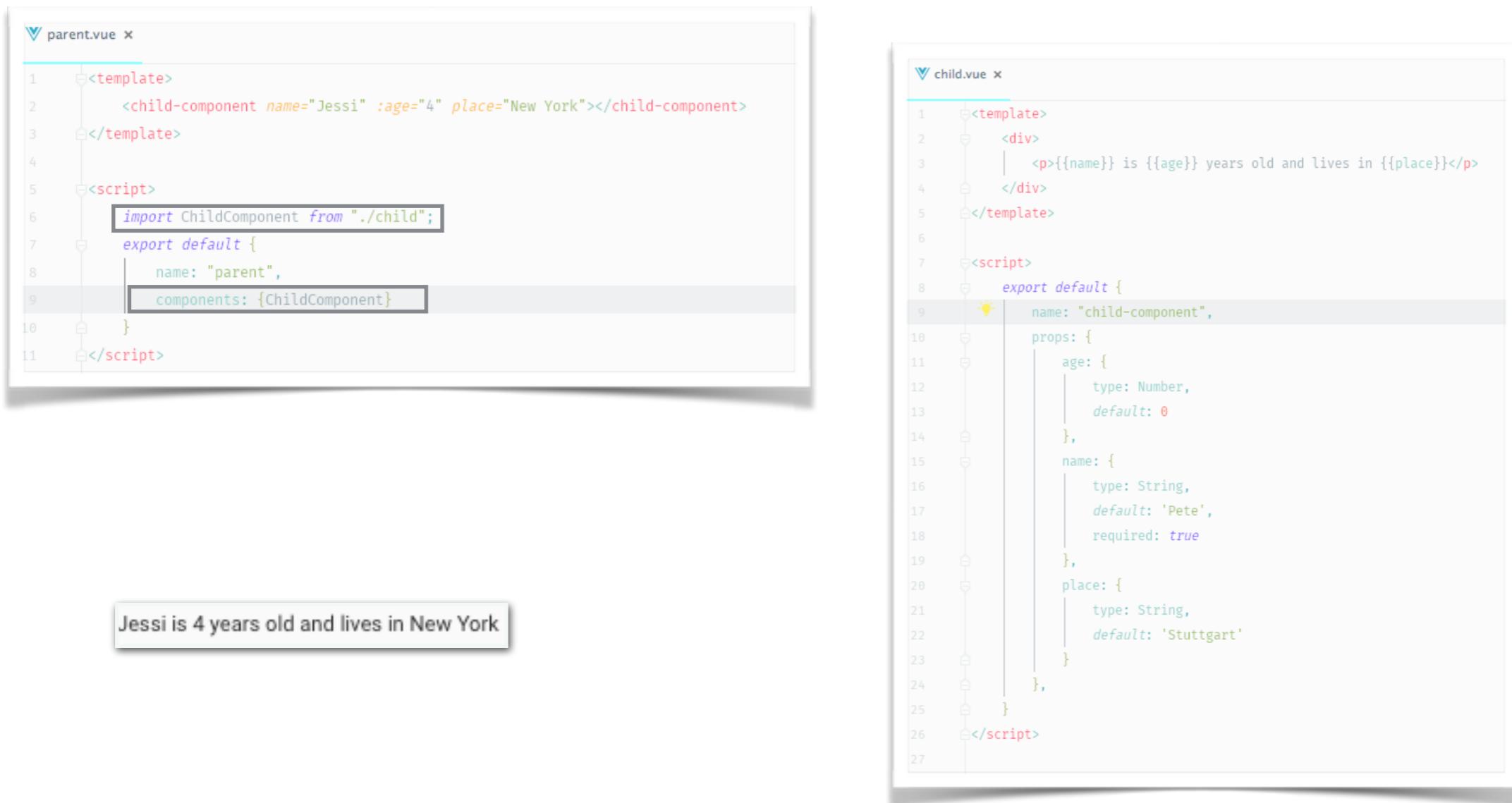
## 2. Components in Vue.js: *add modularity, reusability, structure*

App.vue



## 2. Components in Vue.js: *add modularity, reusability, structure*

---



The image shows a code editor with two files: `parent.vue` and `child.vue`.

**parent.vue:**

```
1 <template>
2   <child-component name="Jessi" :age="4" place="New York"></child-component>
3 </template>
4
5 <script>
6   import ChildComponent from "./child";
7   export default {
8     name: "parent",
9     components: {ChildComponent}
10   }
11 </script>
```

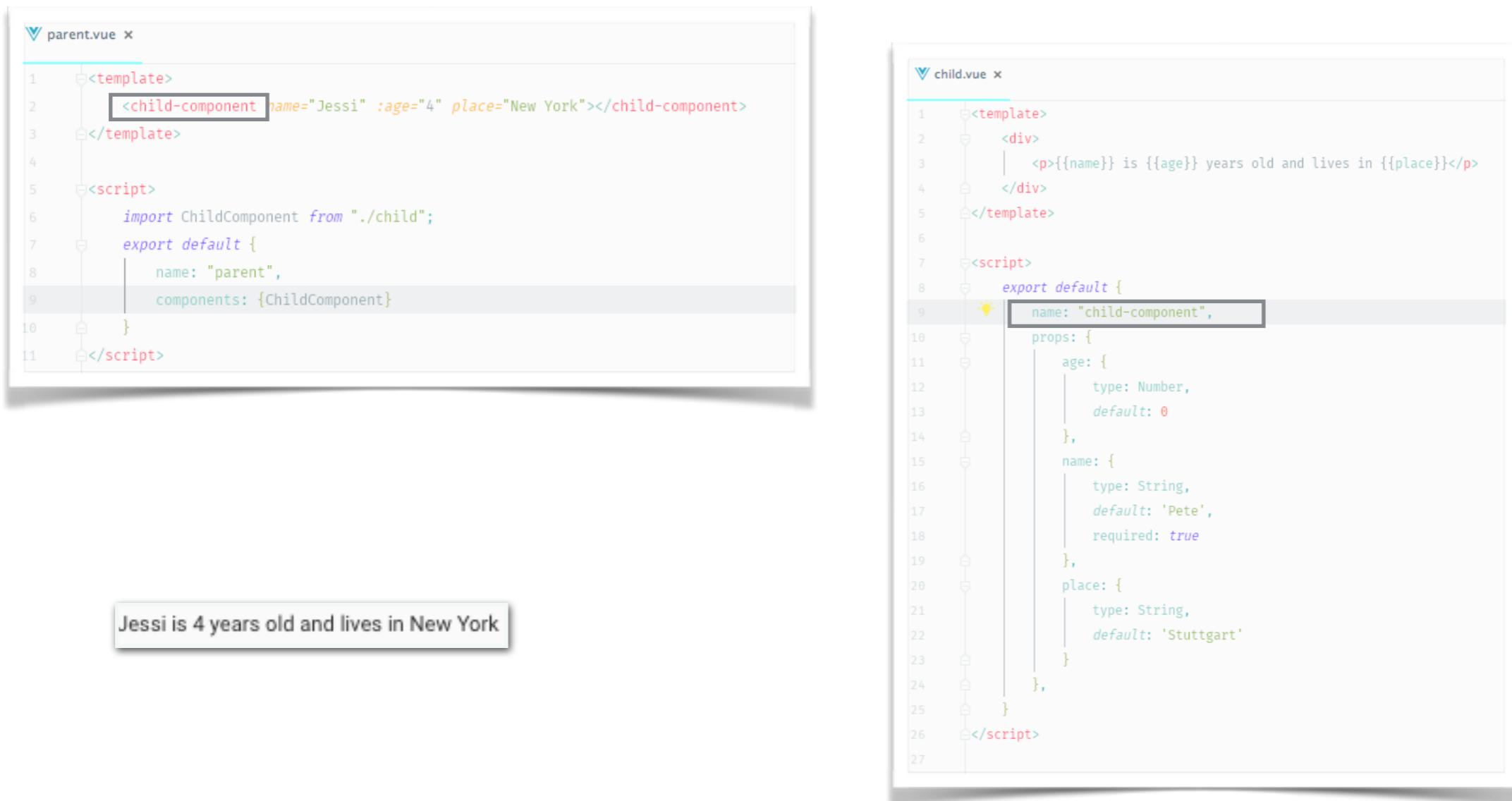
**child.vue:**

```
1 <template>
2   <div>
3     <p>{{name}} is {{age}} years old and lives in {{place}}</p>
4   </div>
5 </template>
6
7 <script>
8   export default {
9     name: "child-component",
10    props: {
11      age: {
12        type: Number,
13        default: 0
14      },
15      name: {
16        type: String,
17        default: 'Pete',
18        required: true
19      },
20      place: {
21        type: String,
22        default: 'Stuttgart'
23      }
24    }
25  </script>
26
27
```

A tooltip at the bottom left of the parent component's template area displays the rendered output: `Jessi is 4 years old and lives in New York`.

## 2. Components in Vue.js: *add modularity, reusability, structure*

---



The image shows two code editors side-by-side, illustrating the creation of a reusable component in Vue.js.

**parent.vue:**

```
1 <template>
2   <child-component name="Jessi" :age="4" place="New York"></child-component>
3 </template>
4
5 <script>
6   import ChildComponent from './child';
7   export default {
8     name: "parent",
9     components: {ChildComponent}
10   }
11 </script>
```

**child.vue:**

```
1 <template>
2   <div>
3     <p>{{name}} is {{age}} years old and lives in {{place}}</p>
4   </div>
5 </template>
6
7 <script>
8   export default {
9     name: "child-component",
10    props: {
11      age: {
12        type: Number,
13        default: 0
14      },
15      name: {
16        type: String,
17        default: 'Pete',
18        required: true
19      },
20      place: {
21        type: String,
22        default: 'Stuttgart'
23      }
24    }
25  </script>
26
27
```

A tooltip at the bottom left of the parent component editor displays the rendered output: "Jessi is 4 years old and lives in New York".

## 2. Components in Vue.js: *add modularity, reusability, structure*

The diagram illustrates the data flow between two Vue.js components: parent.vue and child.vue.

**parent.vue:**

```
1 <template>
2   <child-component name="Jessi" :age="4" place="New York"></child-component>
3 </template>
4
5 <script>
6   import ChildComponent from './child';
7   export default {
8     name: "parent",
9     components: {ChildComponent}
10   }
11 </script>
```

**child.vue:**

```
1 <template>
2   <div>
3     <p>{{name}} is {{age}} years old and lives in {{place}}</p>
4   </div>
5 </template>
6
7 <script>
8   export default {
9     name: "child-component",
10    props: {
11      age: {
12        type: Number,
13        default: 0
14      },
15      name: {
16        type: String,
17        default: 'Pete',
18        required: true
19      },
20      place: {
21        type: String,
22        default: 'Stuttgart'
23      }
24    }
25  </script>
26
27
```

A large arrow labeled "data" points from the prop declaration in child.vue back to the corresponding prop usage in parent.vue, indicating that the data is passed from the parent component to the child component via props.

**Output:**

Jessi is 4 years old and lives in New York

## 2. Components in Vue.js: *add modularity, reusability, structure*



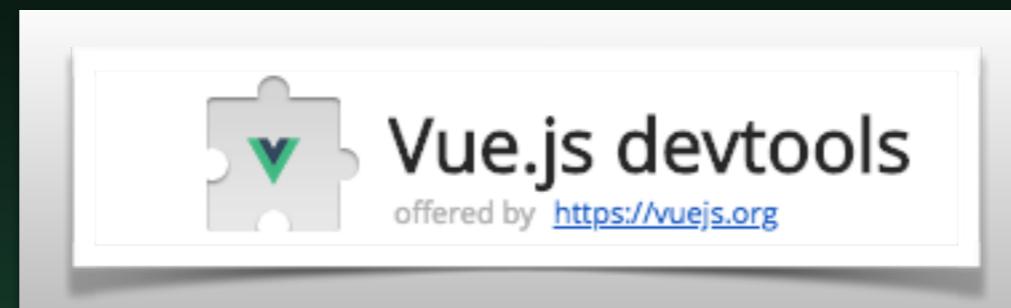
```
parent.vue
1 <template>
2   <child-component name="Jessi" :age="4" place="New York"></child-component>
3 </template>
4
5 <script>
6   import ChildComponent from './child';
7   export default {
8     name: "parent",
9     components: {ChildComponent}
10   }
11 </script>
```

```
child.vue
1 <template>
2   <div>
3     <p>{{name}} is {{age}} years old and lives in {{place}}</p>
4   </div>
5 </template>
6
7 <script>
8   export default {
9     name: "child-component",
10    props: {
11      age: {
12        type: Number,
13        default: 0
14      },
15      name: {
16        type: String,
17        default: 'Pete',
18        required: true
19      },
20      place: {
21        type: String,
22        default: 'Stuttgart'
23      }
24    }
25  </script>
26
```

Jessi is 4 years old and lives in New York

## 2. Components in Vue.js: *add modularity, reusability, structure*

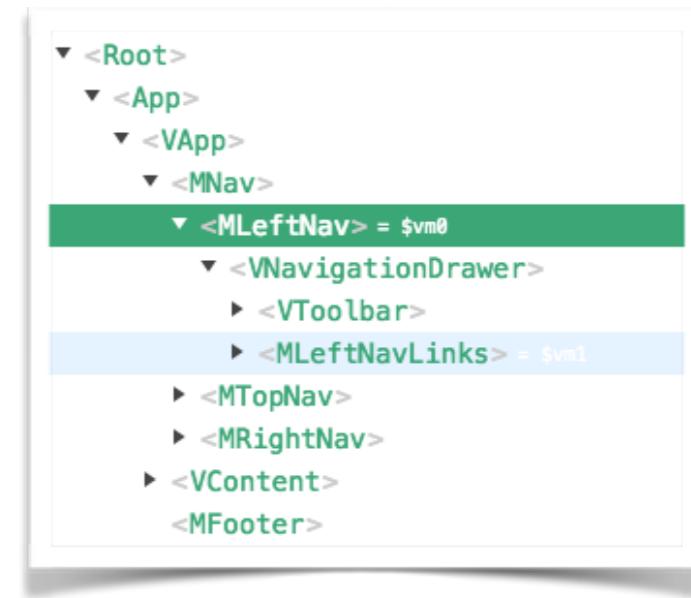
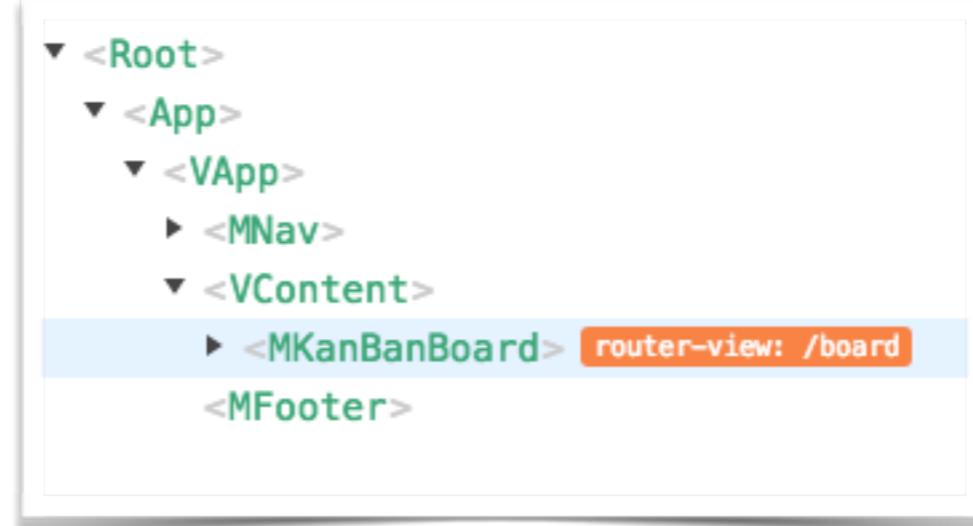
---





Vue.js devtools  
offered by <https://vuejs.org>

## 2. Components in Vue.js: *add modularity, reusability, structure*





## 2. Components in Vue.js: *add modularity, reusability, structure*

The image shows three panels from the Vue.js devtools interface. The left panel displays a component tree for a root application structure. The middle panel shows a specific component, `MKanBanBoard`, with its internal structure and a list of sub-tasks. The right panel shows a detailed view of the `MLeftNav` component and its child components.

**Component Tree (Left Panel):**

- <Root>
  - <App>
    - <VApp>
      - <MNav>
    - <VContent>
      - <UserAdmin> `router-view: /users`

## 2. Components in Vue -> Exercise 2

In src/components/userAdminComponents/UserCard.vue (*child*)

- implement UserCard with `<v-list-tile-avatar><v-list-tile-content>` as shown in the image below
- get User Object onePerson from parent component throw the props attribute

in src/components/pages/UserAdmin.vue: (*parent*)

- use 'src/components/userAdminComponents/UserCard.vue' for displaying one Person by passing the Person object as attribute to `<m-user-card>`

UserCard.vue (*child*)

```
▼ parent.vue x
1  <template>
2    <child-component name="Jessi" :age="4" place="New York"></child-component>
3  </template>
4
5  <script>
6    import ChildComponent from "./child";
7    export default {
8      name: "parent",
9      components: {ChildComponent}
10     }
11 </script>
```

```
▼ child.vue x
1  <template>
2    <div>
3      <p>{{name}} is {{age}} years old and lives in {{place}}</p>
4    </div>
5  </template>
6
7  <script>
8    export default {
9      name: "child-component",
10     props: {
11       age: {
12         type: Number,
13         default: 0
14       },
15       name: {
16         type: String,
17         default: 'Pete',
18         required: true
19       },
20       place: {
21         type: String,
22         default: 'Stuttgart'
23       }
24     }
25   </script>
26
```



### 3. Directives (conditional, loops)

---

```
1 <element  
2   prefix-directiveId="[argument:] expression [| filters...]">  
3 </element>
```

```
1 <element
2   prefix-directiveId="[argument:] expression [! filters...]">
3 </element>
```

### 3. Directive: conditional (v-if, v-show)

parent.vue

```
1 <template>
2   <v-card class="pa-3">
3     <child-component name="Jessi" :age="4" place="New York"></child-component>
4     <child-component name="Jonas" :age="24" place="Stuttgart"></child-component>
5   </v-card>
6 </template>
```

Jessi is 4 years old and lives in New York

Our child has a very cute name! ❤️

Jonas is 24 years old and lives in Stuttgart

child.vue

```
1 <template>
2   <div class="elevation-1 pa-1 ma-1">
3     <p>{{name}} is {{age}} years old and lives in {{place}}</p>
4     <p v-if="name === 'Jessi'">Our child has a very cute name! ❤️</p>
5   </div>
6 </template>
```

```
<div class="elevation-1 pa-1 ma-1"> == $0
  <p>Jessi is 4 years old and lives in New York</p>
  <p>Our child has a very cute name! ❤️</p>
</div>
<div class="elevation-1 pa-1 ma-1">
  <p>Jonas is 24 years old and lives in Stuttgart</p>
  <!-->
</div>
</div>
```

```
1 <element
2   prefix-directiveId="[argument:] expression [! filters...]">
3 </element>
```

### 3. Directive: conditional (v-if, v-show)

parent.vue

```
1 <template>
2   <v-card class="pa-3">
3     <child-component name="Jessi" :age="4" place="New York"></child-component>
4     <child-component name="Jonas" :age="24" place="Stuttgart"></child-component>
5   </v-card>
6 </template>
```

Jessi is 4 years old and lives in New York

Our child has a very cute name! ❤️

Jonas is 24 years old and lives in Stuttgart

child.vue

```
1 <template>
2   <div class="elevation-1 pa-1 ma-1">
3     <p>{{name}} is {{age}} years old and lives in {{place}}</p>
4     <p v-if="name === 'Jessi'">Our child has a very cute name! ❤️</p>
5   </div>
6 </template>
```

child.vue

```
1 <template>
2   <div class="elevation-1 pa-1 ma-1">
3     <p>{{name}} is {{age}} years old and lives in {{place}}</p>
4     <p v-show="name === 'Jessi'">Our child has a very cute name! ❤️</p>
5   </div>
6 </template>
```

```
▼<div class="elevation-1 pa-1 ma-1"> == $0
  <p>Jessi is 4 years old and lives in New York</p>
  <p>Our child has a very cute name! ❤️</p>
</div>
▼<div class="elevation-1 pa-1 ma-1">
  <p>Jonas is 24 years old and lives in Stuttgart</p>
  <!-->
</div>
</div>
```

```
▼<div class="elevation-1 pa-1 ma-1"> == $0
  <p>Jessi is 4 years old and lives in New York</p>
  <p>Our child has a very cute name! ❤️</p>
</div>
▼<div class="elevation-1 pa-1 ma-1">
  <p>Jonas is 24 years old and lives in Stuttgart</p>
  <p style="display: none;">Our child has a very cute name! ❤️</p>
</div>
```

```
1 <element  
2   prefix-directiveId="[argument:] expression [! filters...]">  
3 </element>
```

### 3. Directive: conditional (v-for)

```
data() {  
  return {  
    users: [  
      {  
        name: 'Jessi',  
        age: 4,  
        place: 'New York'  
      },  
      {  
        name: 'Jonas',  
        age: 24,  
        place: 'Stuttgart'  
      },  
      {  
        name: 'Julie',  
        age: 34,  
        place: 'Paris'  
      }  
    ]  
  },  
},
```

```
<li v-for="(user, index) in users" :key="index"> {{index}} : {{user}}</li>
```

- 0:{ "name": "Jessi", "age": 4, "place": "New York" }
- 1:{ "name": "Jonas", "age": 24, "place": "Stuttgart" }
- 2:{ "name": "Julie", "age": 34, "place": "Paris" }

```
1 <element
2   prefix-directiveId="[argument:] expression [! filters...]">
3 </element>
```

### 3. Directive: conditional (v-for)

```
data() {
  return {
    users: [
      {
        name: 'Jessi',
        age: 4,
        place: 'New York'
      },
      {
        name: 'Jonas',
        age: 24,
        place: 'Stuttgart'
      },
      {
        name: 'Julie',
        age: 34,
        place: 'Paris'
      }
    ]
  }
},
```

```
<li v-for="(user, index) in users" :key="index"> {{index}} : {{user}} </li>
```

- 0:{ "name": "Jessi", "age": 4, "place": "New York" }
- 1:{ "name": "Jonas", "age": 24, "place": "Stuttgart" }
- 2:{ "name": "Julie", "age": 34, "place": "Paris" }

```
<li v-for="(user, key, index) in users[0]" :key="key"> {{index}} : {{key}} : {{user}} </li>
```

- 0:name:Jessi
- 1:age:4
- 2:place:New York

```
1 <element  
2   prefix-directiveId="[argument:] expression [! filters...]">  
3 </element>
```

### 3. Directive: conditional (v-for)

```
data() {  
  return {  
    users: [  
      {  
        name: 'Jessi',  
        age: 4,  
        place: 'New York'  
      },  
      {  
        name: 'Jonas',  
        age: 24,  
        place: 'Stuttgart'  
      },  
      {  
        name: 'Julie',  
        age: 34,  
        place: 'Paris'  
      }  
    ]  
  },  
},
```

```
▼ parent.vue x  
1 <template>  
2   <v-card class="pa-3">  
3     <child-component v-for="(user, index) in users" :key="user.name" :name="user.name" :age="user.age" :place="user.place"></child-component>  
4   </v-card>  
5 </template>
```

Jessi is 4 years old and lives in New York

Our child has a very cute name! ❤️

Jonas is 24 years old and lives in Stuttgart

Julie is 34 years old and lives in Paris

```

1  <element
2    prefix-directiveId="[:argument:] expression [I filters...]">
3  </element>

```

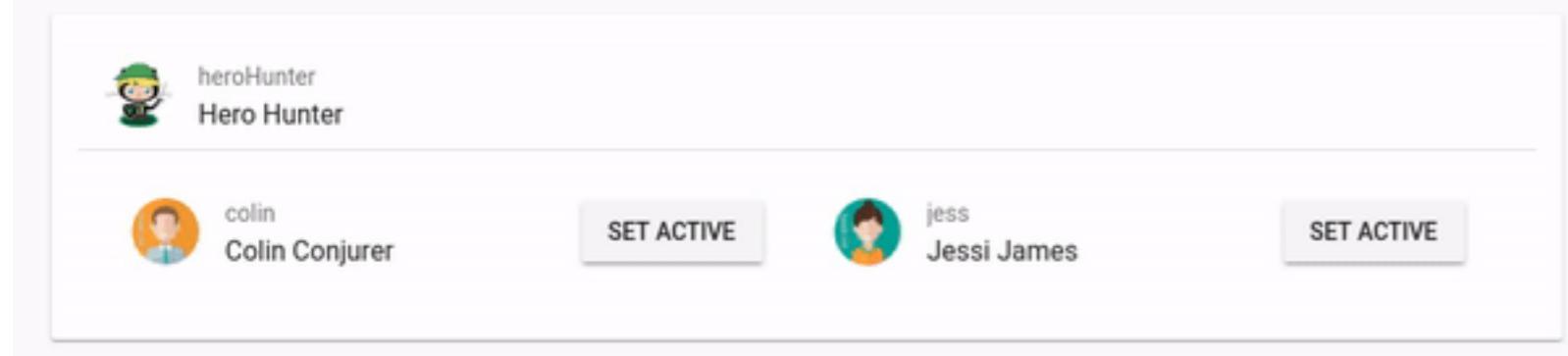
### 3. Directives (conditional, loops): Exercise 3

In src/components/userAdminComponents/ActivateUser.vue:

- use 'src/components/userAdminComponents/UserCard.vue' <m-user-card> for displaying the user nicely (reuse the component built in the previous tutorial)
- implement the method setActive(), which is activated when a user click on the btn. This method should tell the parent component to update its activeUser

in src/components/pages/UserAdmin.vue:

- define activeUser:{} Object and allUsers: {} Object & delete onePerson\*
- loop over allUsers and display them in <m-activate-user> component
- <m-activate-user> emits the event changeActiveUser() the activeUser should be updated, but only if it is not already the active User



```
<li v-for="(user, index) in users" :key="index"> {{index}} : {{user}}</li>
```

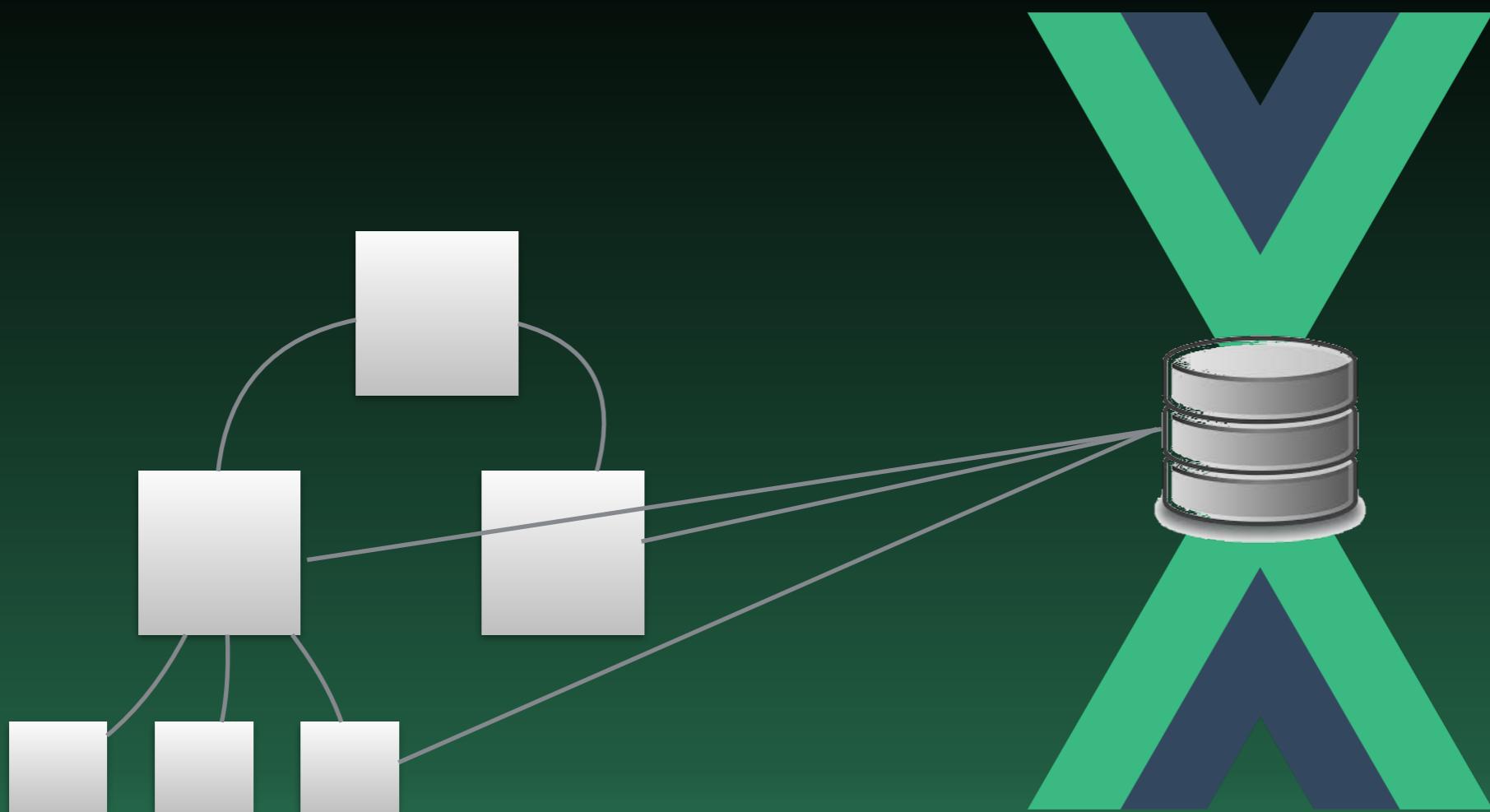
```
<li v-for="(user, key, index) in users[0]" :key="key"> {{index}} : {{key}} : {{user}} </li>
```

```

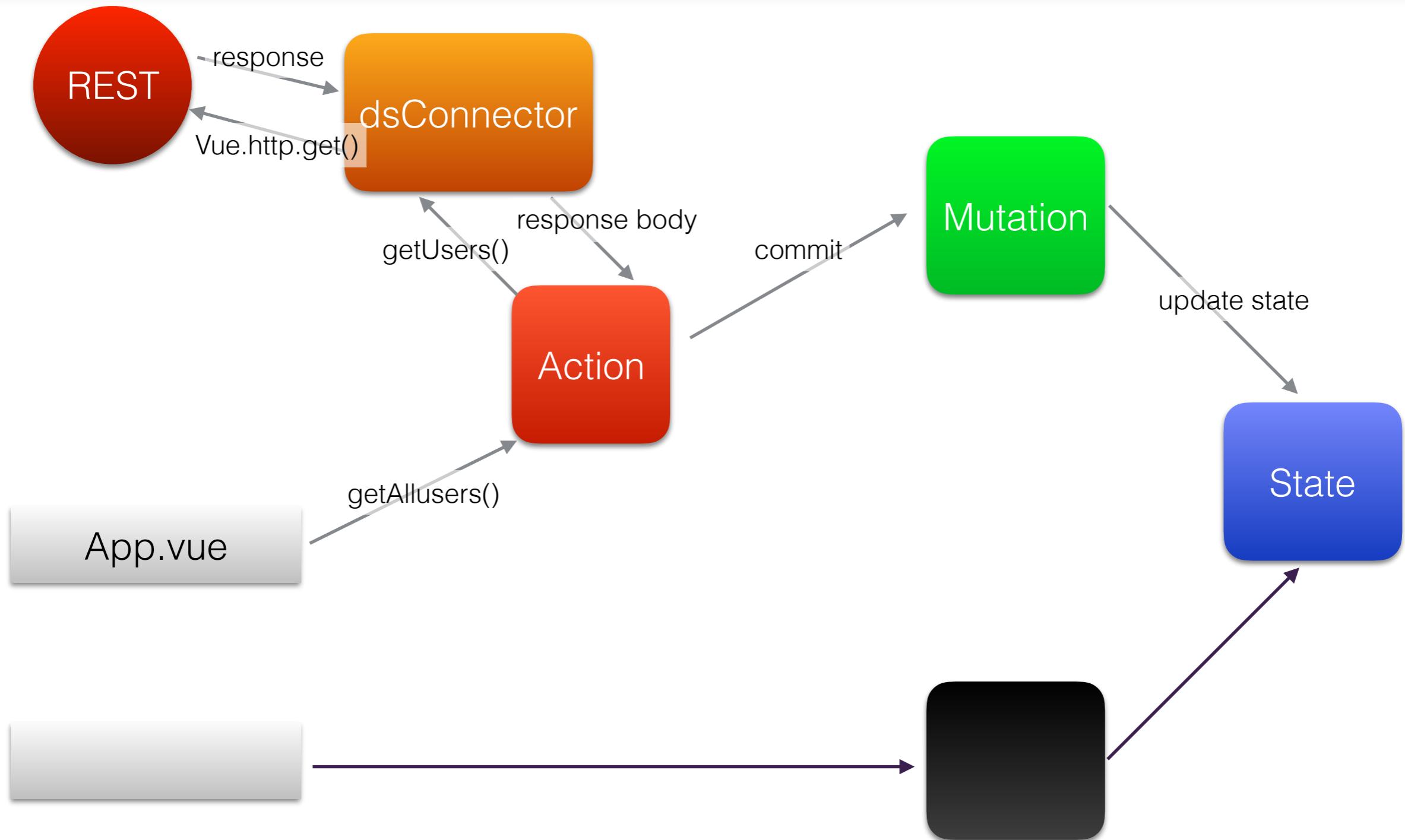
data() {
  return {
    activeUser: {
      userName: "heroHunter",
      id: 3,
      firstName: "Hero",
      lastName: "Hunter",
      img: "/static/img/avatars/hero.png",
    },
    allUsers: {
      colin: {
        userName: "colin",
        id: 1,
        firstName: "Colin",
        lastName: "Conjurer",
        img: "/static/img/avatars/avatar-male-1.png",
      },
      jess: {
        userName: "jess",
        id: 2,
        firstName: "Jessi",
        lastName: "James",
        img: "/static/img/avatars/avatar-female-1.png",
      },
      heroHunter: {
        userName: "heroHunter",
        id: 3,
        firstName: "Hero",
        lastName: "Hunter",
        img: "/static/img/avatars/hero.png",
      },
    }
  }
}

```

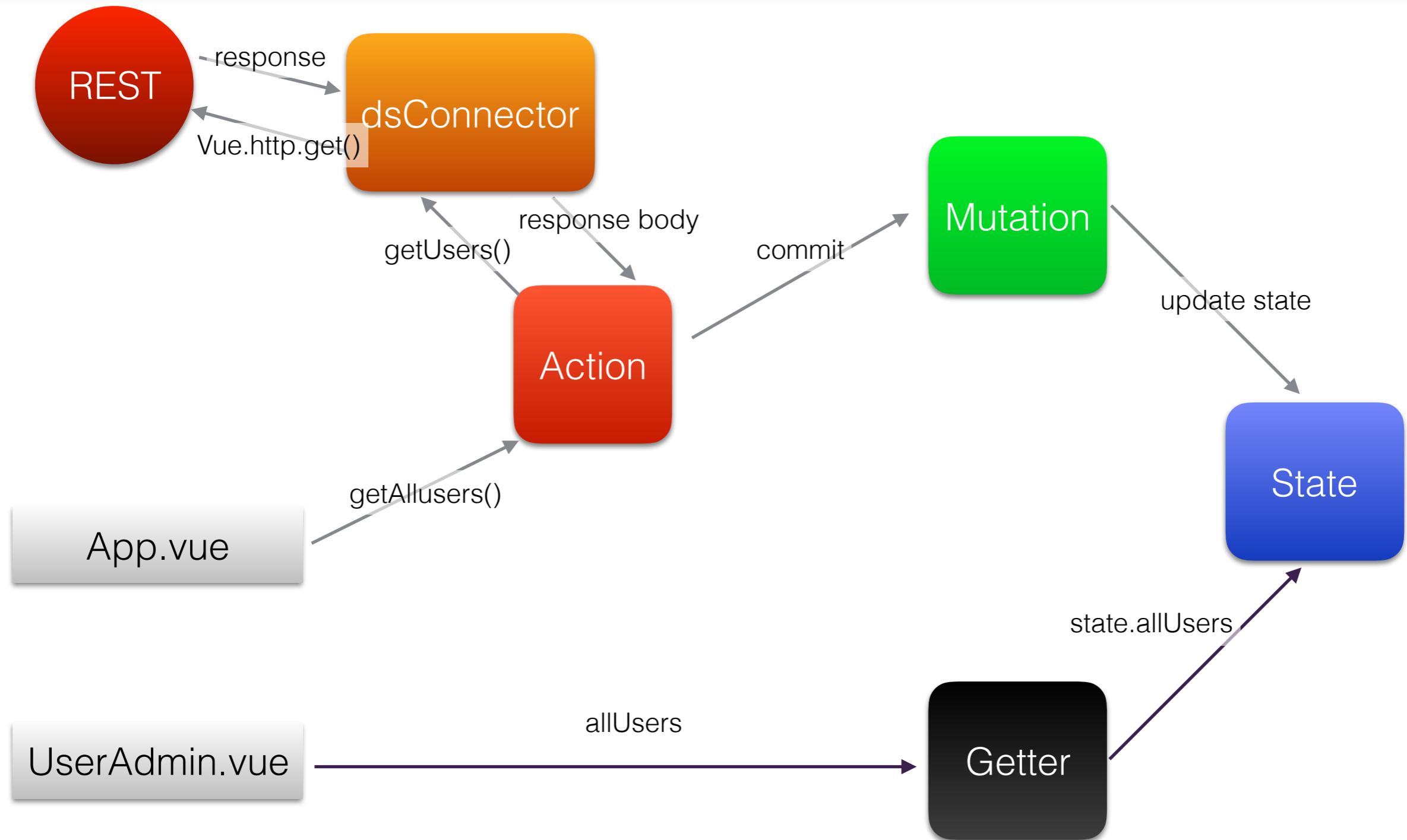
## 4. Global state management with Vuex



## 4. Global state management with Vuex



## 4. Global state management with Vuex



## 4. Global state management with Vuex



The image shows a code editor window with the file 'App.vue' open. The code is written in Vue.js. A yellow rectangular box highlights the import statement at the top. Another yellow box highlights the call to `...mapActions` in the methods section. A third yellow box highlights the three actions being mapped: `getCards`, `getAllUsers`, and `getAllChildren`. The code also includes a `name` property, component registrations, and a `created` hook with three calls to `this`.

```
App.vue x
import {mapActions} from 'vuex'

export default {
  name: 'App',
  components: { MFooter, MNav },
  methods: {
    ...mapActions(['getCards', 'getAllUsers', 'getAllChildren']),
  },
  created() {
    this.getCards({id: 1})
    this.getAllUsers()
    this.getAllChildren()
  }
}
```

## 4. Global state management with Vuex

The diagram illustrates the flow of a Vuex action from a component to a store module.

**App.vue**

```
import { mapActions } from 'vuex'

export default {
  name: 'App',
  components: { MFooter, MNav },
  methods: {
    ...mapActions(['getCards', 'getAllUsers', 'getAllChildren']),
  },
  created() {
    this.getCards({id: 1})
    this.getAllUsers()
    this.getAllChildren()
  },
}
```

**dsConnector/index.js**

```
import { dsChildren } from '../dsConnector/index'

const state = {
  all: []
}

const getters = {
  allChildren: state => state.all,
}

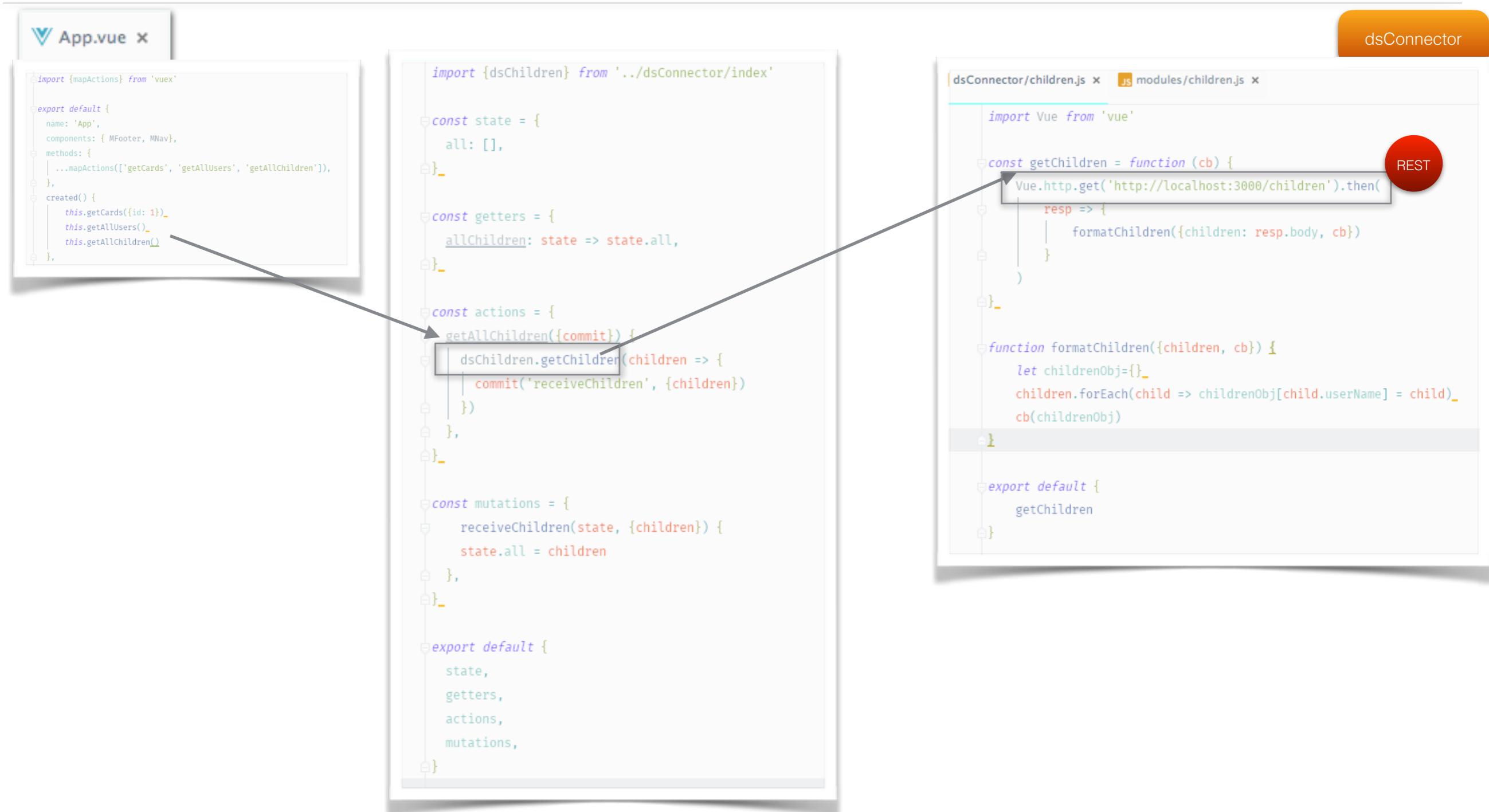
const actions = {
  getAllChildren({commit}) {
    dsChildren.getChildren(children => {
      commit('receiveChildren', {children})
    })
  }
}

const mutations = {
  receiveChildren(state, {children}) {
    state.all = children
  }
}

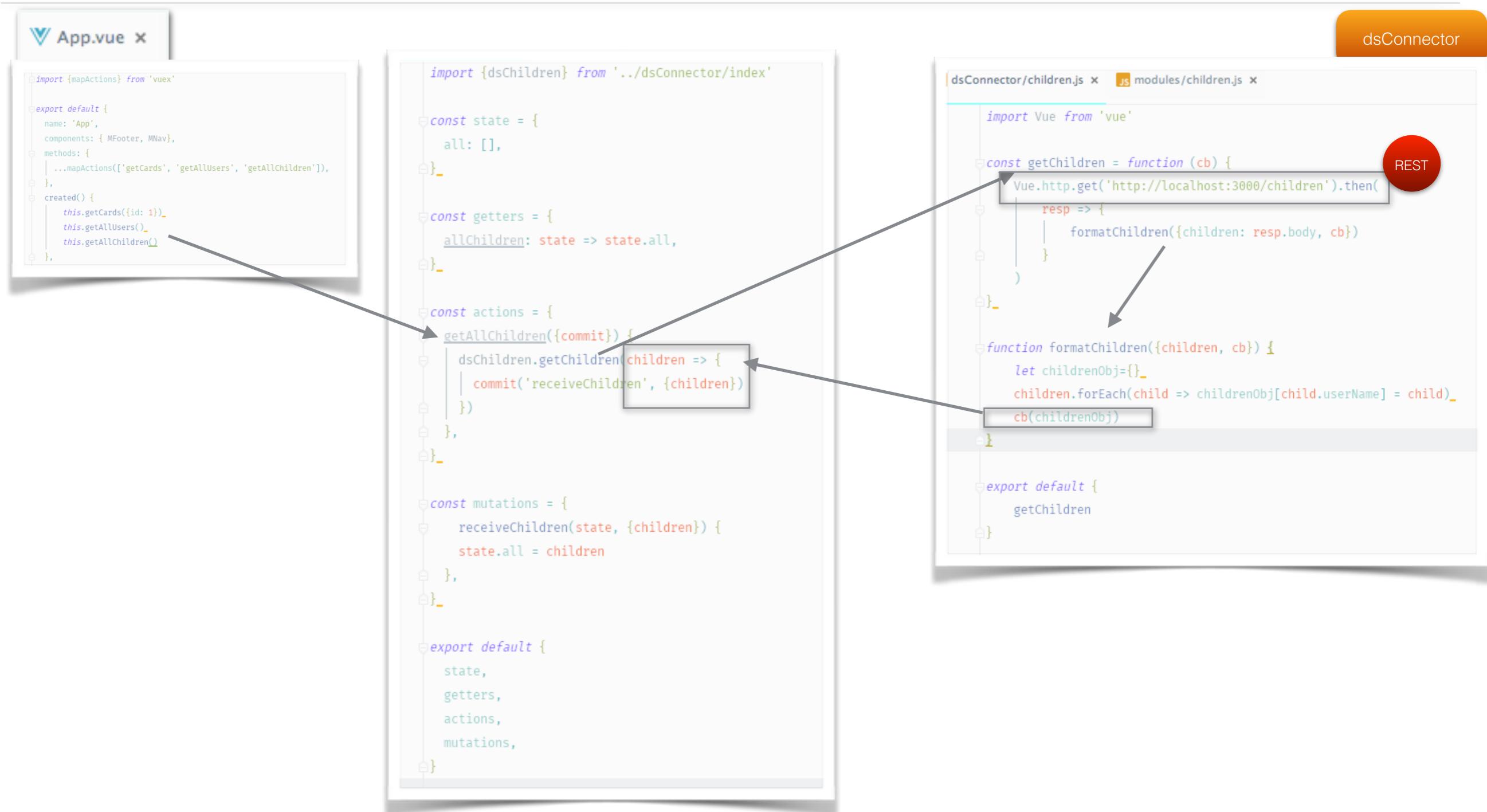
export default {
  state,
  getters,
  actions,
  mutations,
}
```

A callout arrow points from the highlighted `this.getAllChildren()` method in `App.vue` to the `getAllChildren` action in `dsConnector/index.js`.

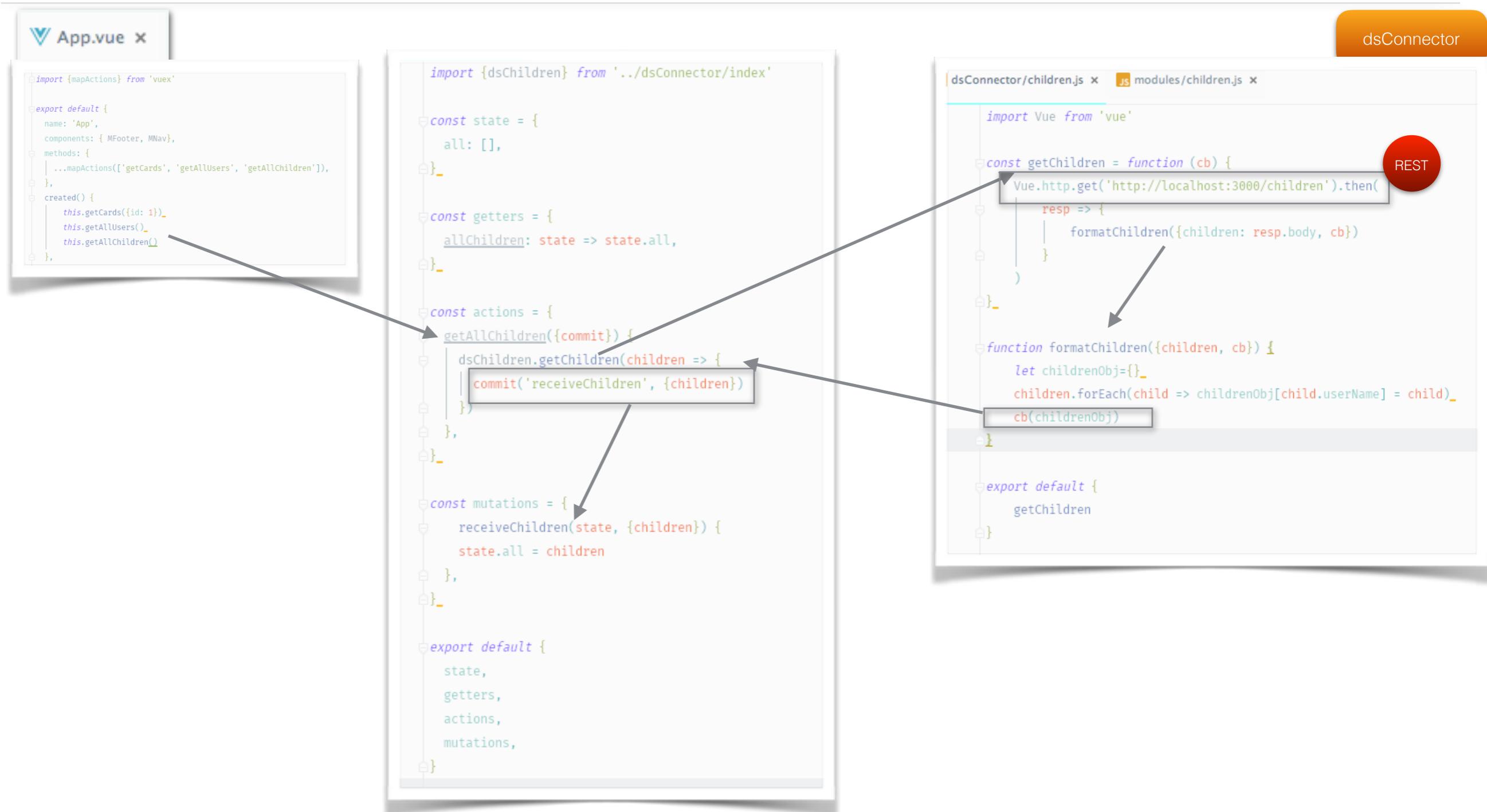
## 4. Global state management with Vuex



## 4. Global state management with Vuex



## 4. Global state management with Vuex



## 4. Global state management with Vuex

The diagram illustrates the global state management architecture using Vuex. It shows three files: `App.vue`, `dsConnector/index.js`, and `dsConnector/children.js`.

**App.vue:**

```
import {mapActions} from 'vuex'

export default {
  name: 'App',
  components: { MFooter, MNav },
  methods: {
    ...mapActions(['getCards', 'getAllUsers', 'getAllChildren']),
  },
  created() {
    this.getCards({id: 1})
    this.getAllUsers()
    this.getAllChildren()
  },
}
```

**dsConnector/index.js:**

```
import {dsChildren} from '../dsConnector/index'

const state = {
  all: []
}

const getters = {
  allChildren: state => state.all,
}

const actions = {
  getAllChildren({commit}) {
    dsChildren.getChildren(children => {
      commit('receiveChildren', {children})
    })
  },
}

const mutations = {
  receiveChildren(state, {children}) {
    state.all = children
  },
}

export default {
  state,
  getters,
  actions,
  mutations,
}
```

**dsConnector/children.js:**

```
import Vue from 'vue'

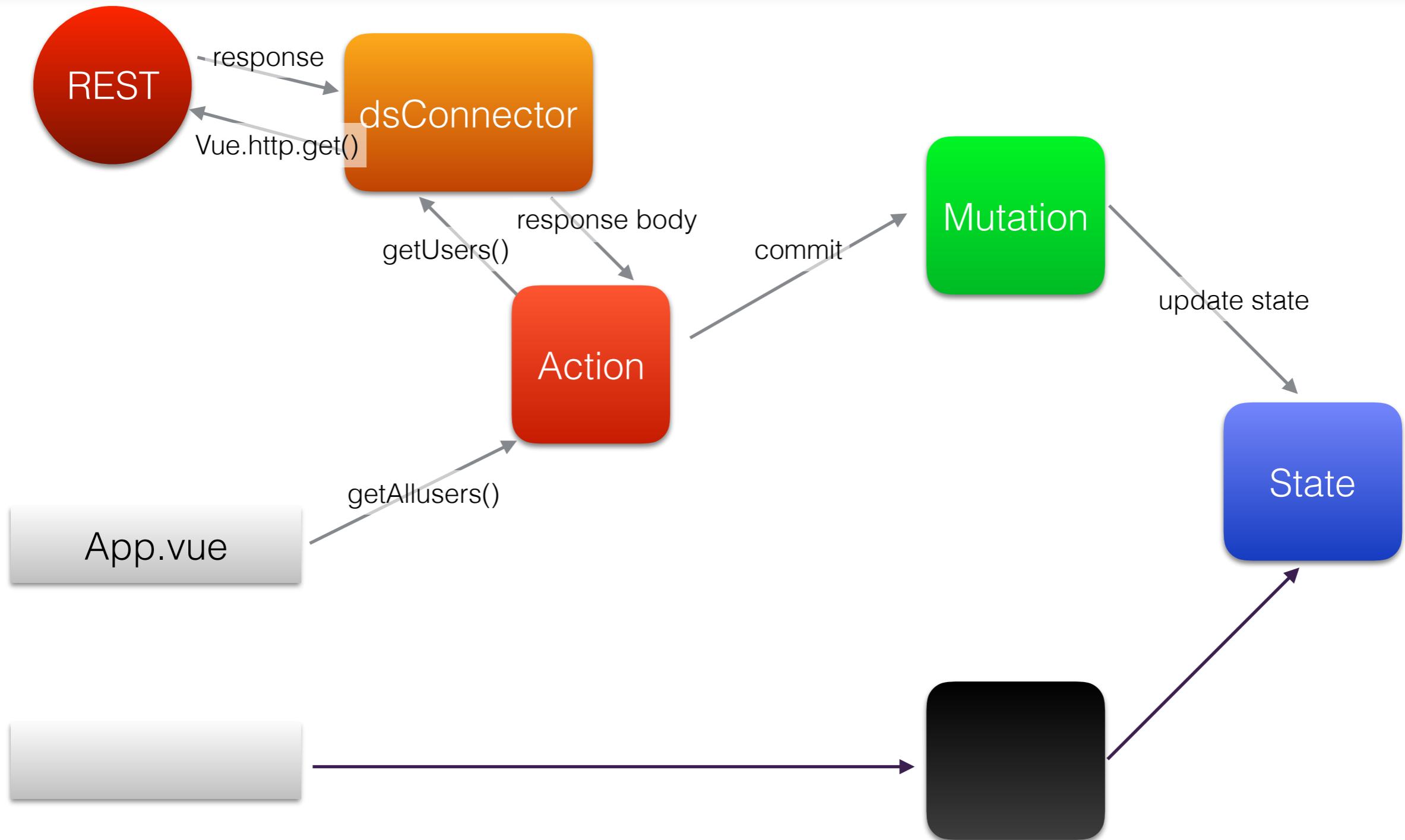
const getChildren = function (cb) {
  Vue.http.get('http://localhost:3000/children').then(
    resp => {
      formatChildren({children: resp.body, cb})
    }
  )
}

function formatChildren({children, cb}) {
  let childrenObj={}
  children.forEach(child => childrenObj[child.userName] = child)
  cb(childrenObj)
}

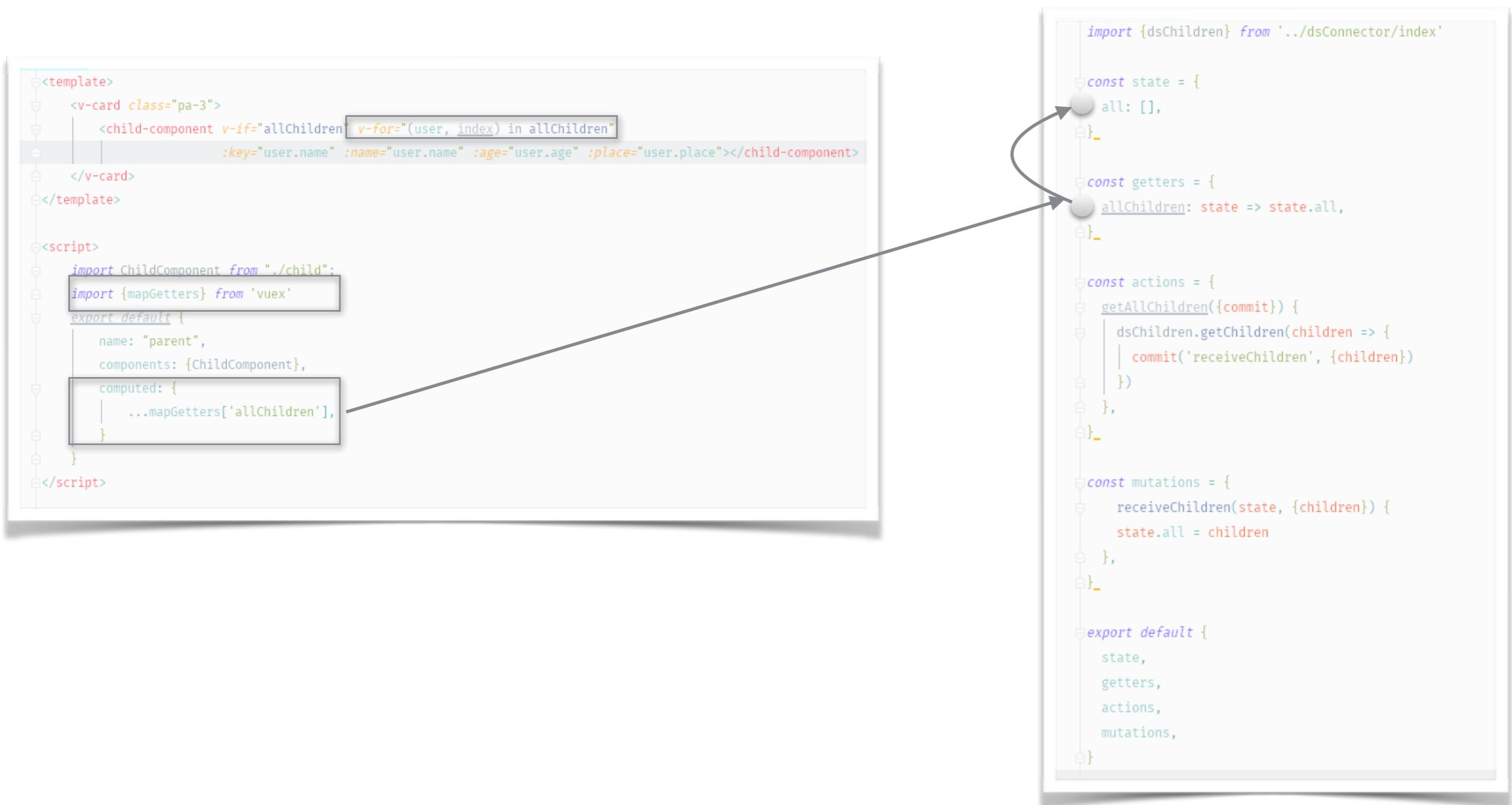
export default {
  getChildren
}
```

A red circle labeled "REST" is placed over the `formatChildren` function in `dsConnector/children.js`, indicating that this part of the code interacts with a RESTful API.

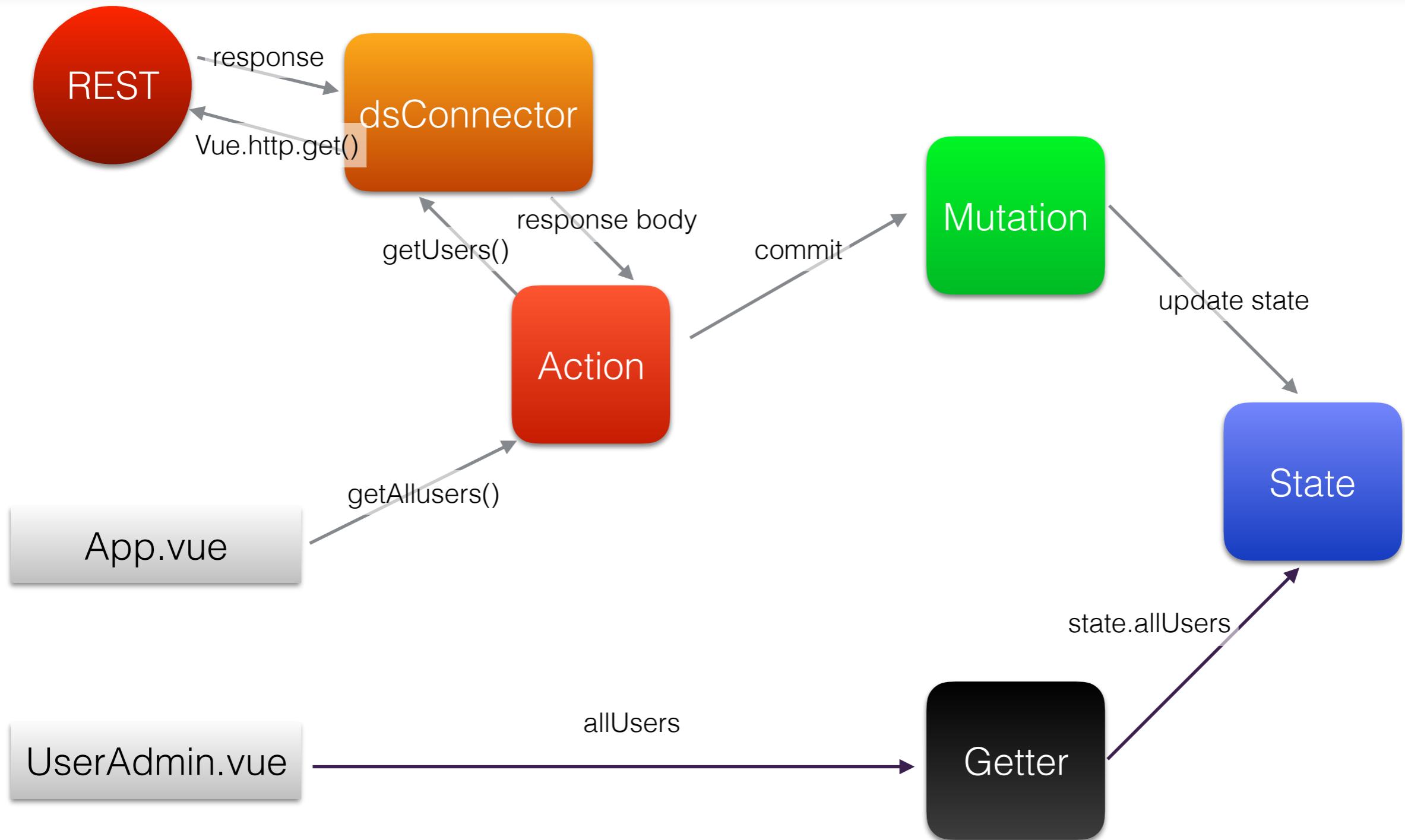
## 4. Global state management with Vuex



## 4. Global state management with Vuex



## 4. Global state management with Vuex



## 4. Global state management with Vuex: Exercise 4

in src/components/pages/UserAdmin.vue:

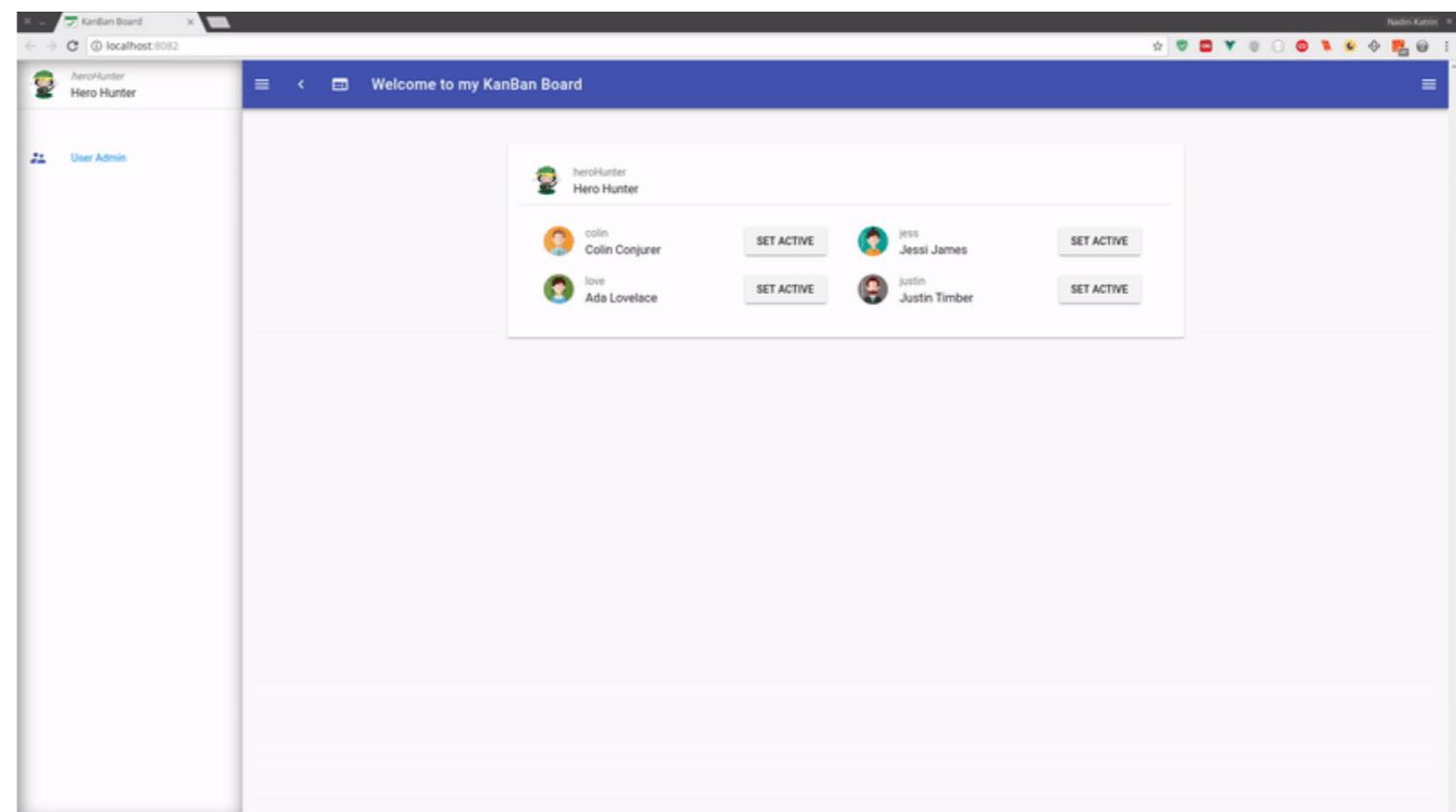
- Use the Getter allUsers & activeUser from Vuex store src/store/modules/users.js for receiving the global application state

In src/components/userAdminComponents/ActivateUser.vue:

- update the method setActive() by calling the action setActive() from src/store/modules/users.js in Vuex store, which passes the user object to the correspondent mutation method, that updated the global state and with that updates the user in leftNav.vue.

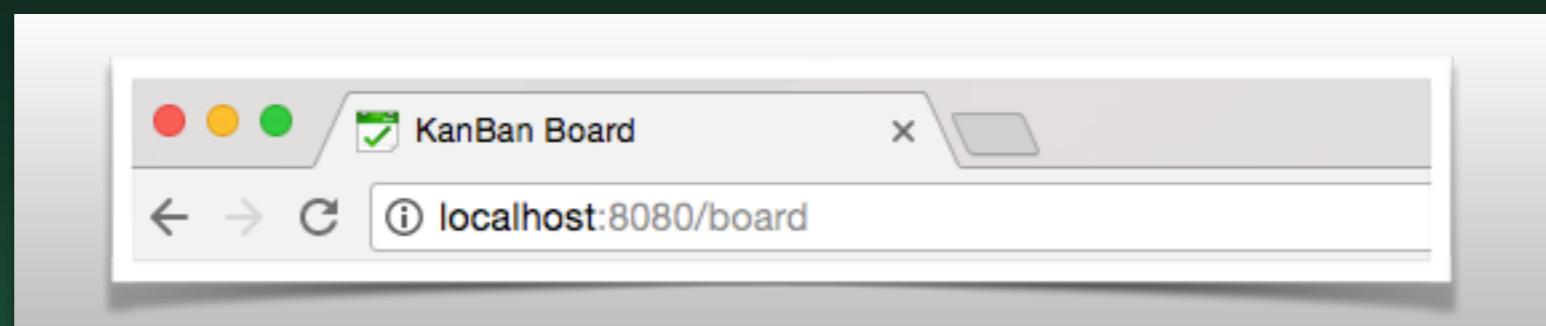
```
<template>
  <v-card class="pa-3">
    <child-component v-if="allChildren" v-for="(user, index) in allChildren"
      :key="user.name" :name="user.name" :age="user.age" :place="user.place"></child-component>
  </v-card>
</template>

<script>
  import ChildComponent from './child';
  import {mapGetters} from 'vuex'
  export default {
    name: "parent",
    components: {ChildComponent},
    computed: {
      ...mapGetters['allChildren'],
    }
  }
</script>
```

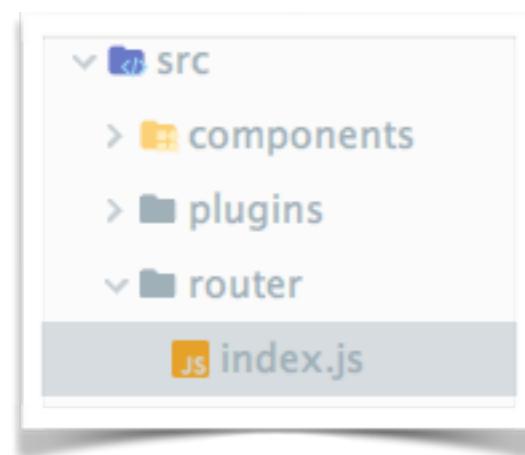
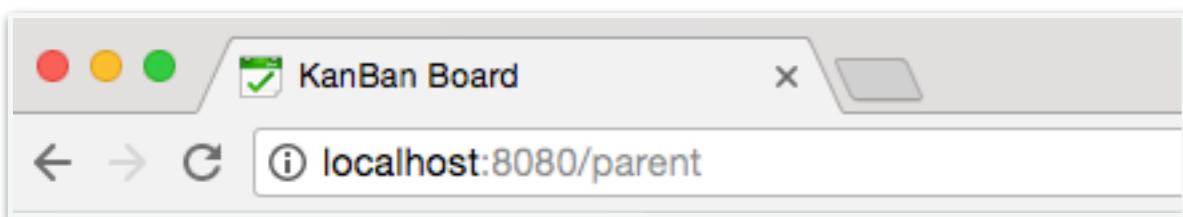


## 5. Routing in Vue

---



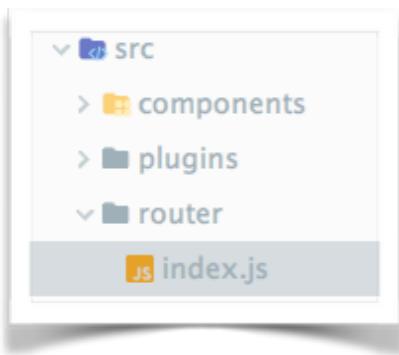
## 5. Routing in Vue



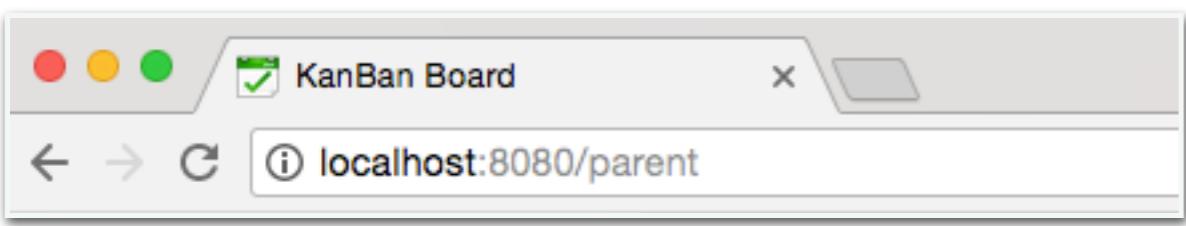
```
import Vue from 'vue'
import Router from 'vue-router'
import UserAdmin from '@/components/pages/UserAdmin'
import Parent from '@/components/pages/parent'

Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '*',
      name: 'UserAdmin',
      component: UserAdmin
    },
    {
      path: '/parent',
      name: 'Parent',
      component: Parent
    }
  ],
  mode: 'history'
})
```



## 5. Routing in Vue

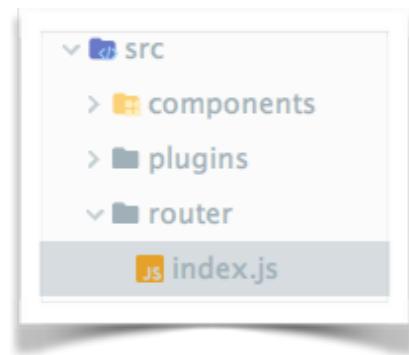


```
App.vue x
<template>
  <v-app>
    <m-nav></m-nav>
    <v-content>
      <router-view></router-view>
    </v-content>
    <m-footer></m-footer>
  </v-app>
</template>
```

```
import Vue from 'vue'
import Router from 'vue-router'
import UserAdmin from '@/components/pages/UserAdmin'
import Parent from '@/components/pages/parent'

Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '*',
      name: 'UserAdmin',
      component: UserAdmin
    },
    {
      path: '/parent',
      name: 'Parent',
      component: Parent
    }
  ],
  mode: 'history'
})
```



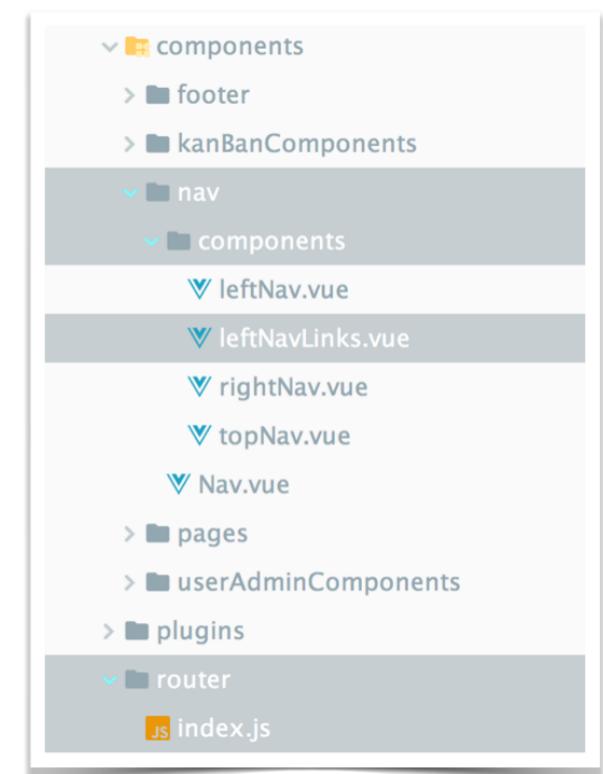
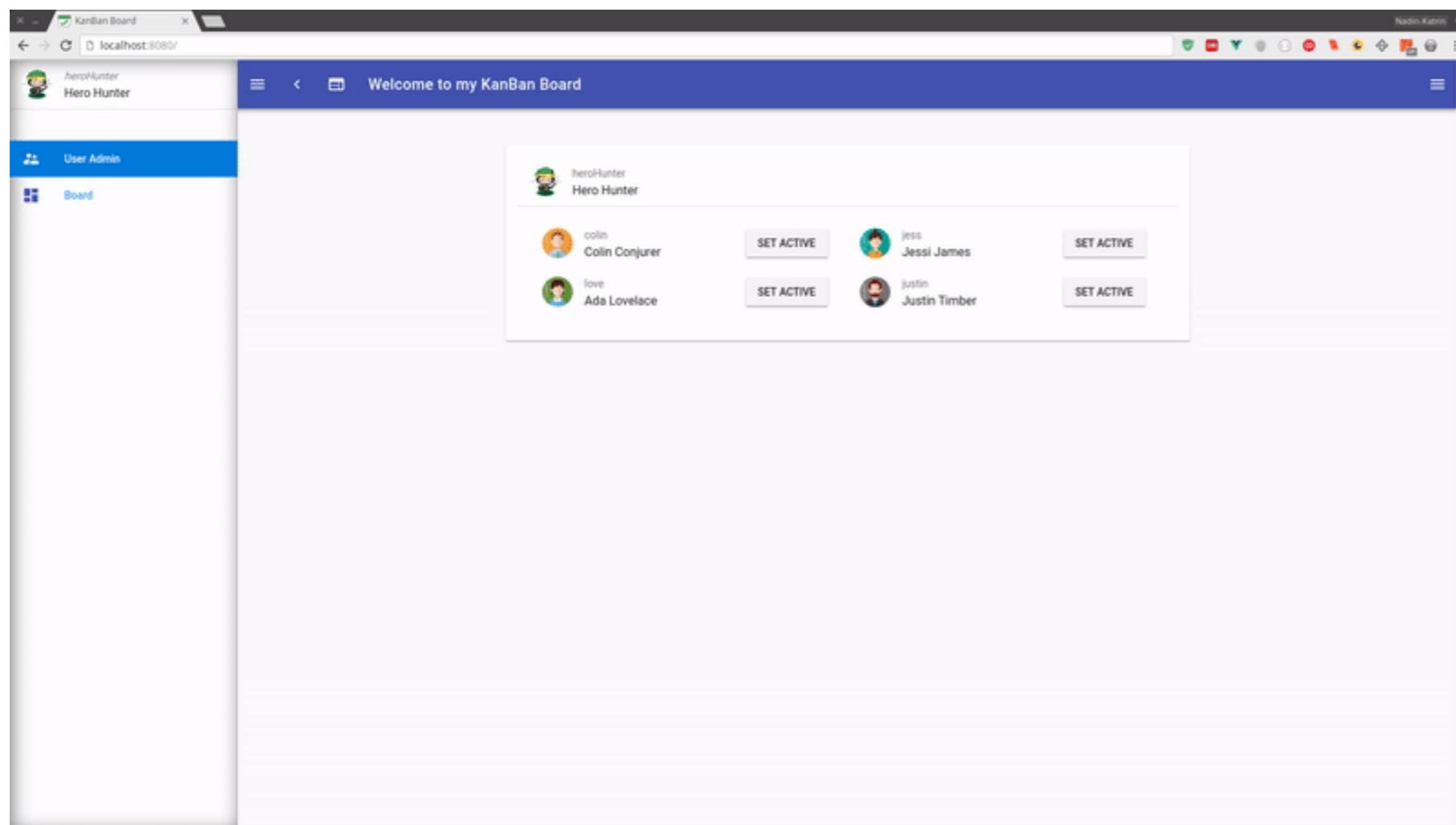
## 5. Routing in Vue: Exercise 5

in src/router/index.js:

- implement routes to '/' '/board' '/users' to UserAdmin page & KanBanBoard page

In src/components/nav/components/leftNavLinks.vue:

- add link to /board with 'dashboard' icon



# Exercise 6: Final Adjustments

in src/store/modules/kanbancards.js:

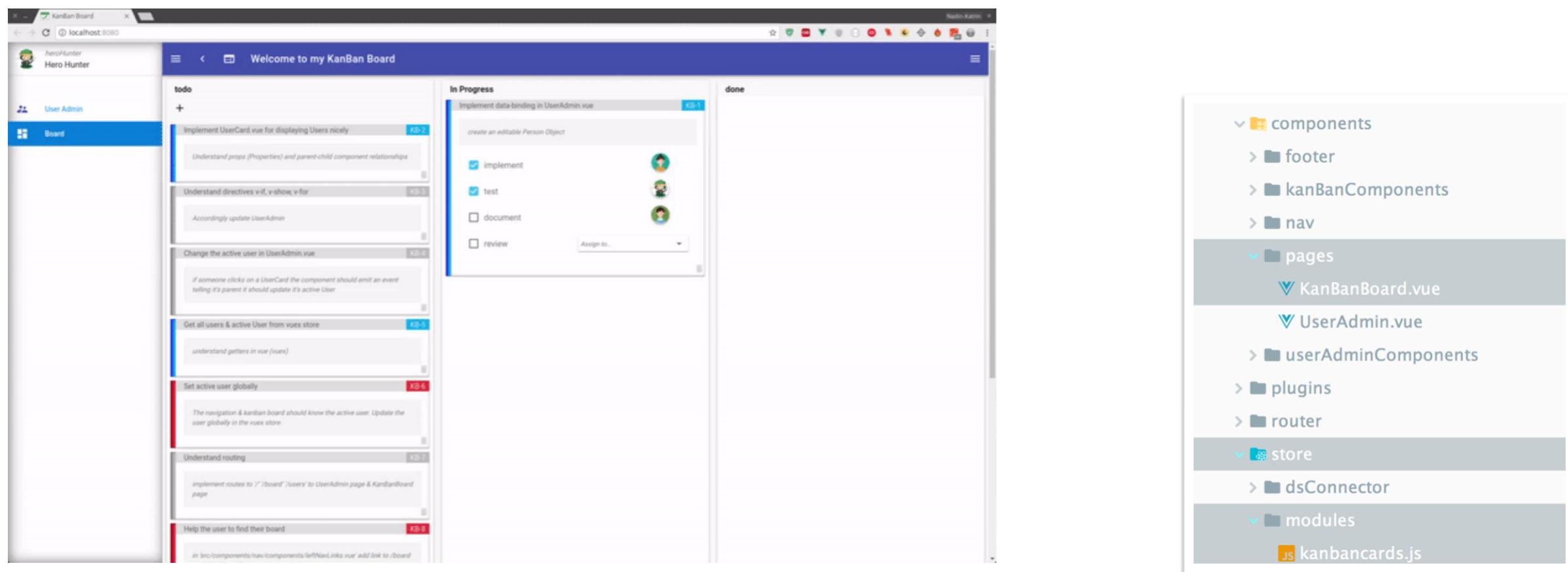
- define status type (e.g. status: ['inProgress', 'done'...])  
used in 'src/components/kanBanComponents/KanBanColumn.vue' defining the columns,

in 'src/components/kanBanComponents/KanBanCard.vue' defining the card view & in 'src/store/modules/kanbancards.js'  
receiveAllCards() for Ordering the cards by status

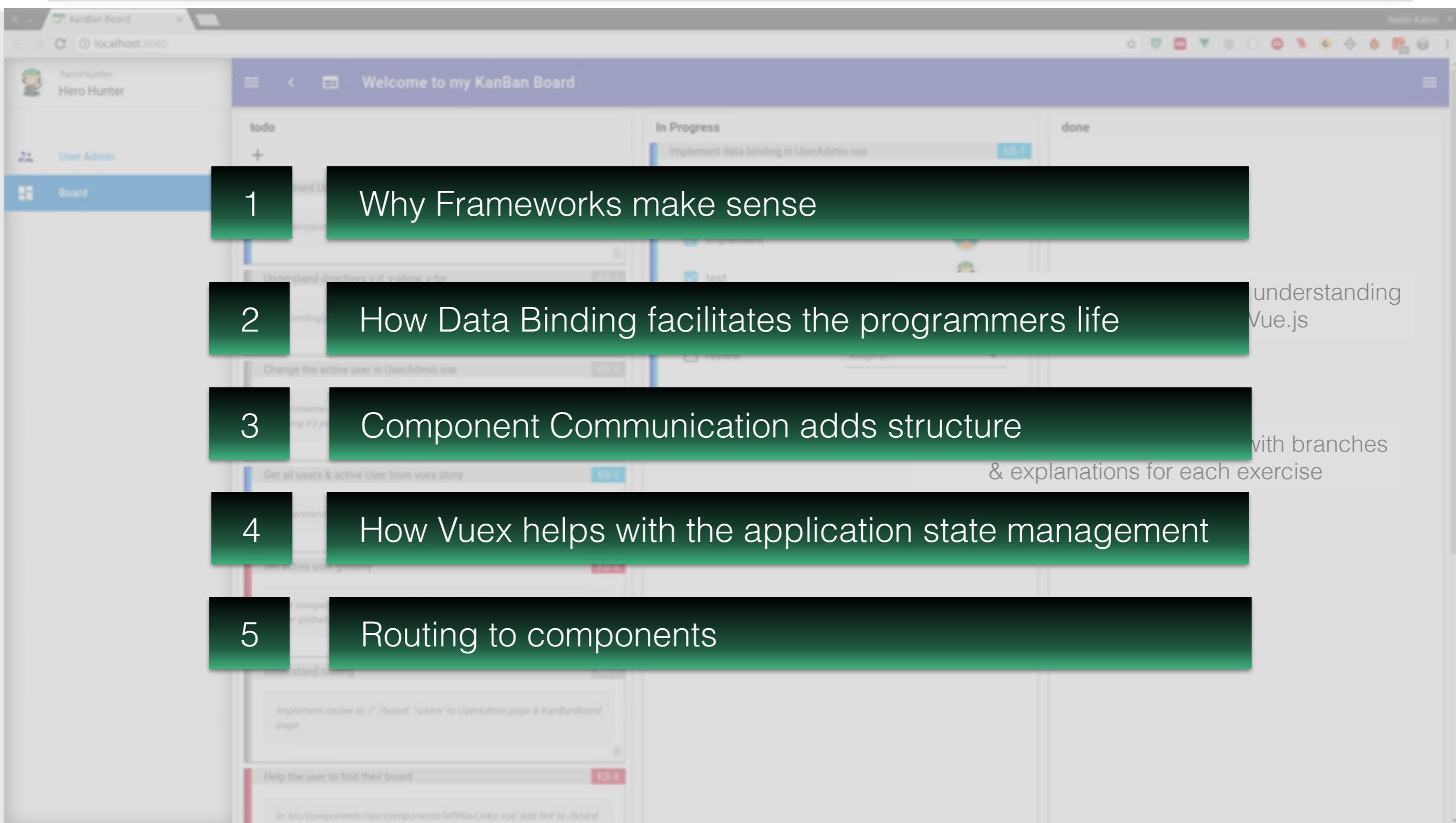
- implement getStatus getter returning the status possibilities of the kanban column & card status types (e.g. inProgress, done)

In src/components/pages/KanBanBoard.vue:

- create 'm-kan-ban-column' for each status defined in the vuex store 'src/store/modules/kanbancards.js'



# Wrap up: What did you learn?



Kanban Board

Welcome to my KanBan Board

User Admin

Board

todo

In Progress

done

Implement UserCard view for displaying Users nicely

Understand Lenses (Projected) and parent/child component relationship

Understand directives w/ isolate vs for

Accesibility update UserAdmin

Change the actions on to Disables user

If someone clicks on a UserCard, the component should emit an event containing the user id and the current state (active/inactive)

Get all Users & active User from users store

understand getters w/ user store

Set active user priority

The navigation & kanban board should know the active user. Update the user priority in the user store

Understand routing

Implementation routes in "2" board. Home to Dashboard page & KanbanBoard page

Help the user to find their board

an icon component has been recently left behind, link user and link inc closed

# Thanks for your Attention

Nadin-Katrin Apel, Alexander Schübl, David Bochan



You should now have a working kanban board



# Thanks for your Attention

A screenshot of a KanBan board application titled "Welcome to my KanBan Board". The application is running on a local host at port 8080. The interface is divided into three main columns: "todo", "In Progress", and "done".

- todo:** A list of tasks:
  - Implement UserCard.vue for displaying Users nicely (KB-2)
  - Understand directives v-if, v-show, v-for (KB-3)
  - Change the active user in UserAdmin.vue (KB-4)
  - Get all users & active User from vuex store (KB-5)
  - Set active user globally (KB-6)
  - Understand routing (KB-7)
  - Help the user to find their board (KB-8)
- In Progress:** A task titled "Implement data-binding in UserAdmin.vue (KB-1)" is shown. It includes a description: "create an editable Person Object". There are four checkboxes: "implement" (checked), "test" (checked), "document" (unchecked), and "review" (unchecked). To the right of the checkboxes are three user icons.
- done:** This column is currently empty.

You should now have a working kanban board