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

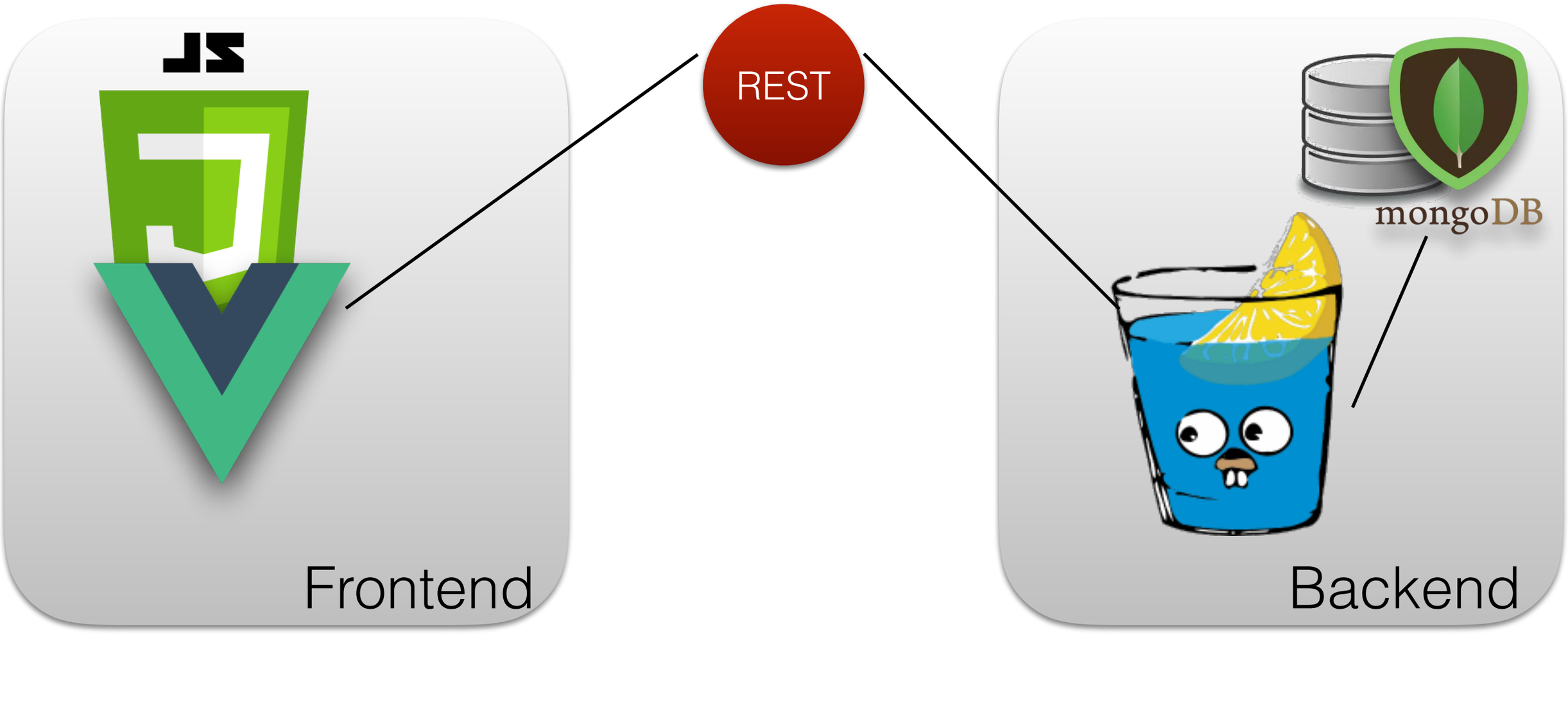
1 Project setup: separation of backend & fronted

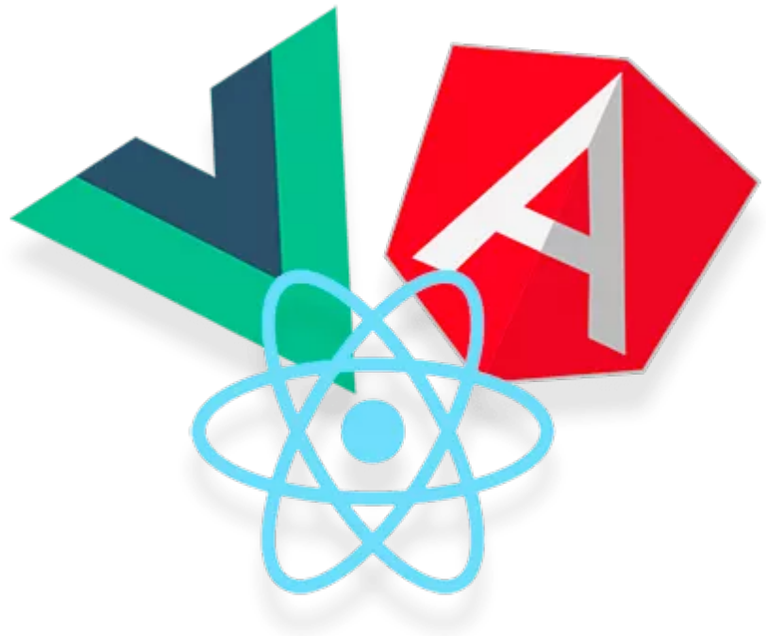
2 Why use a framework?

3 Main Concepts of Vue.js

4 Main Concepts of Gin Gonic

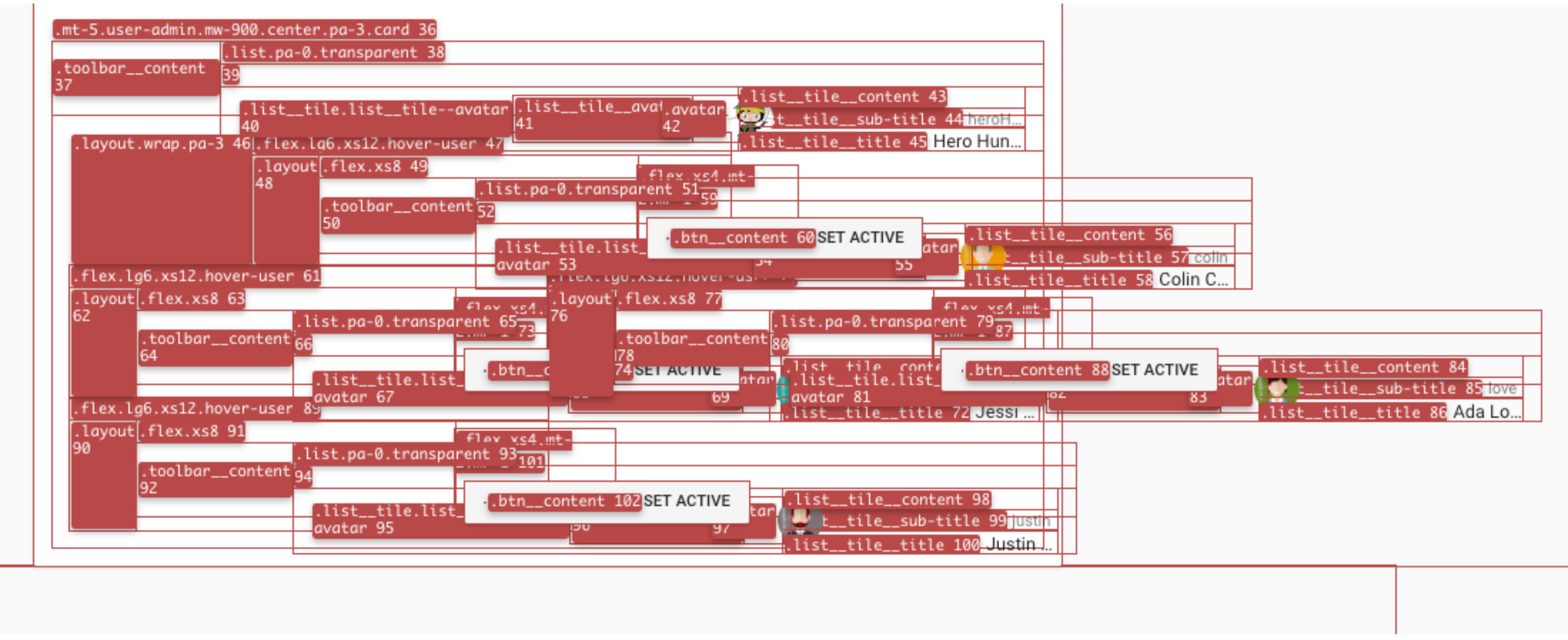
Separation of Backend/ Frontend





JS Framework - why ?

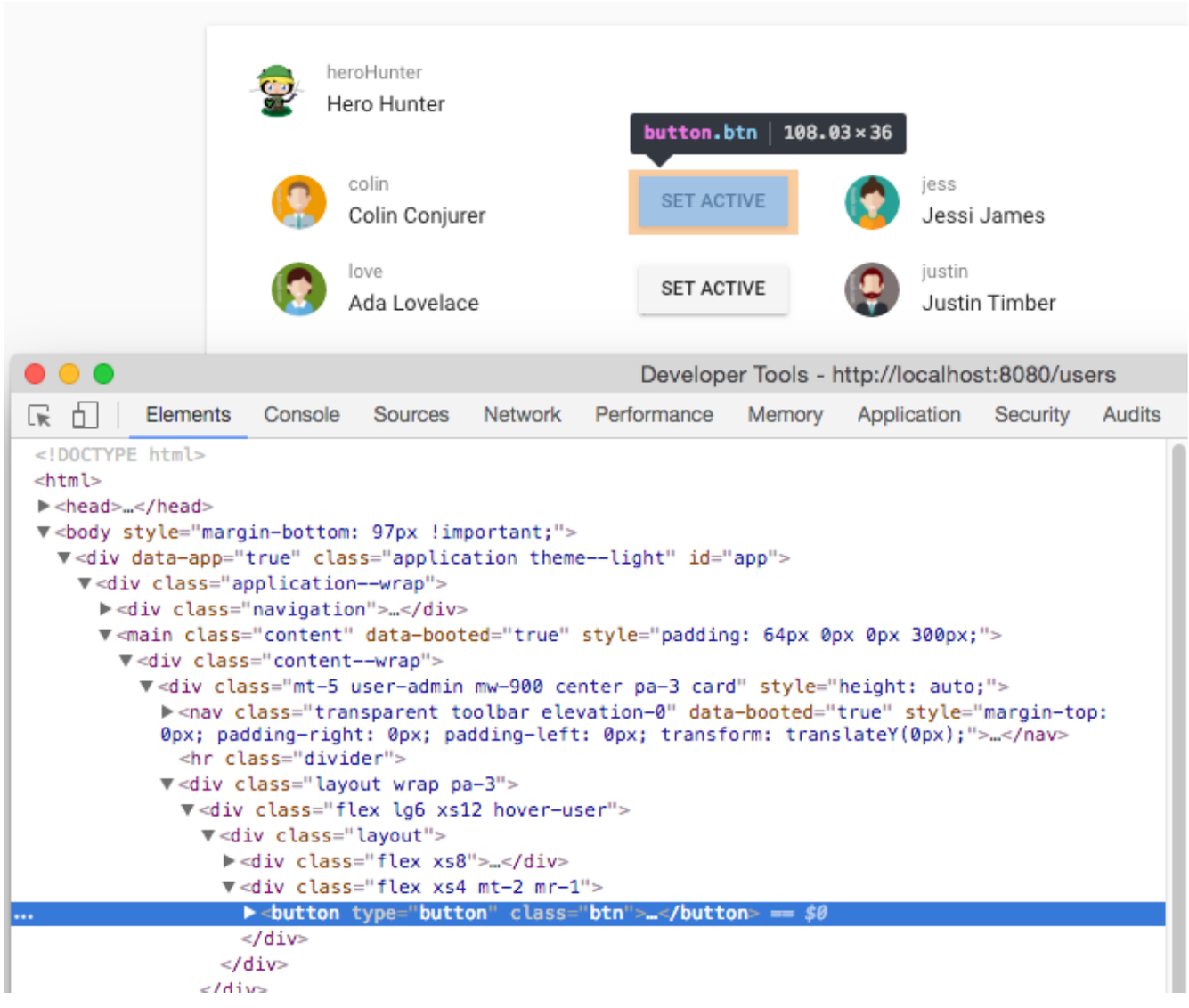
HTML pages are built with complex markup



Frameworks Add Structure

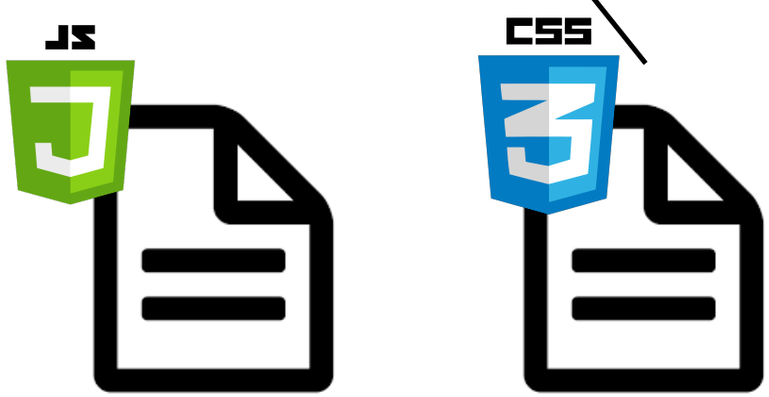
Optimal traversing through the HTML DOM is a hard job

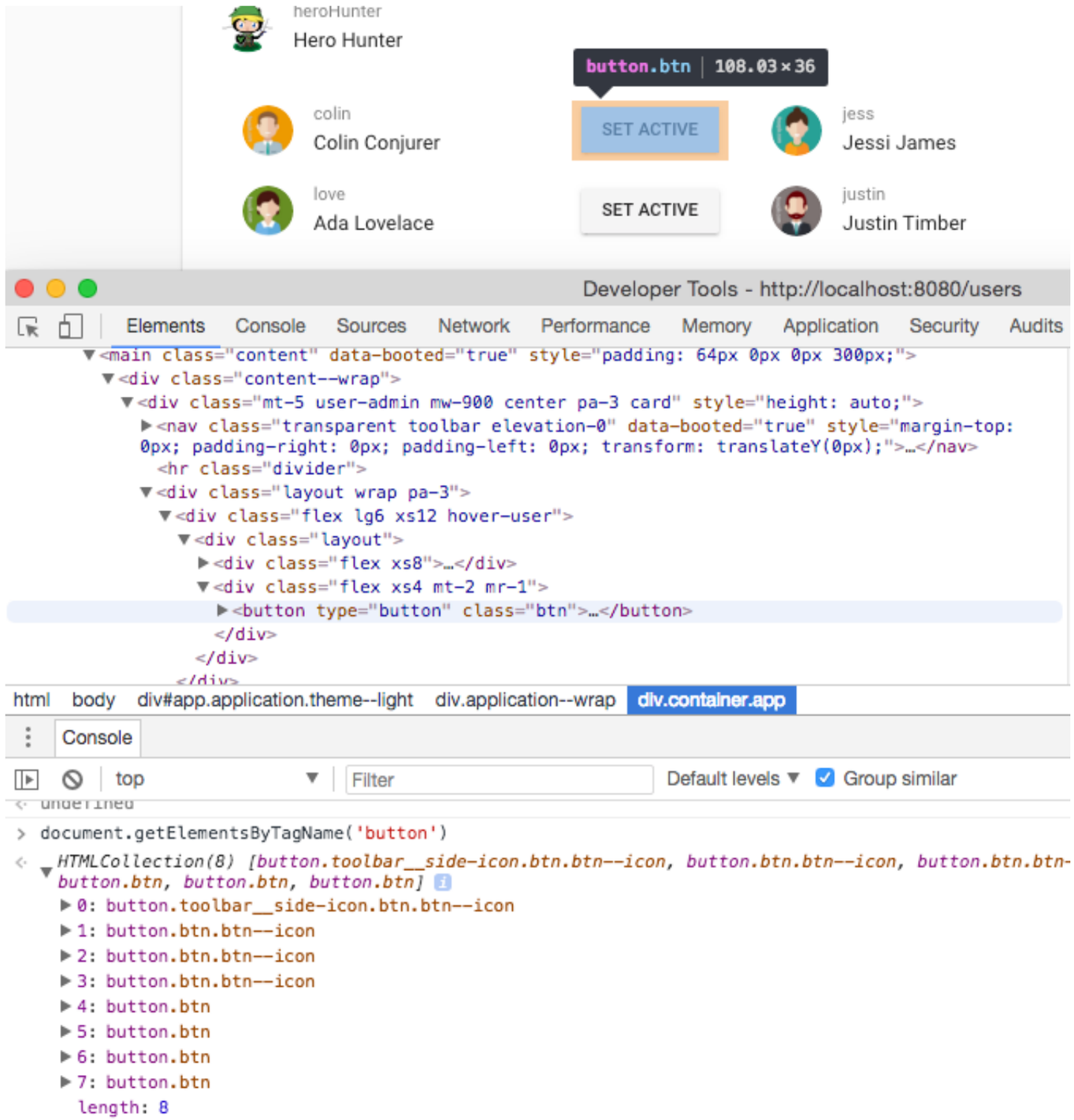
Want to add styling or functionality?
- You have to search for your Element in the DOM tree.



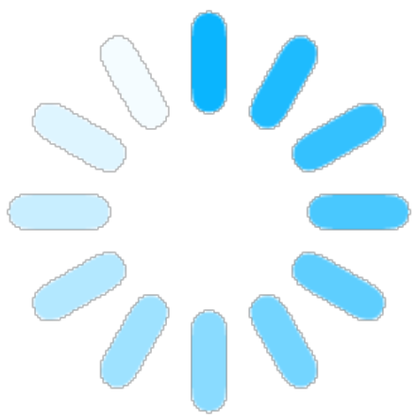
Optimal traversing through the HTML DOM is a hard job

Want to add styling or functionality?
- You have to search for your Element in the DOM tree.





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



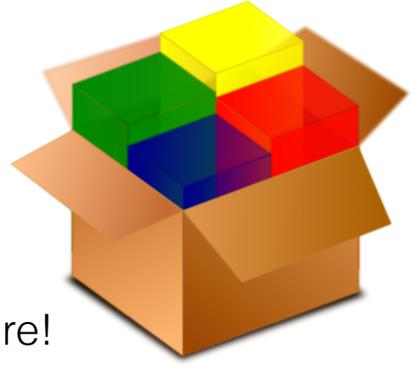
Single Page Applications May Be Your Answer

Many Script tags for a SPA are devastating

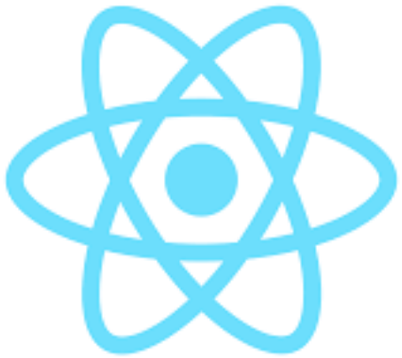
```
<script type="text/javascript" src=""></script>
<script type="text/javascript" src=""></script>
<script type="text/javascript" src=""></script>
<script type="text/javascript" src=""></script>
<script type="text/javascript" src=""></script>
<script type="text/javascript" src=""></script>
<script type="text/javascript"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



facebook

Alibaba Group
阿里巴巴集团

Vue.js in Depth: Our Example Application

https://github.com/na018/vue_basics_tut

1 Data Binding

2 Component Communication

3 Global State Management With Vuex

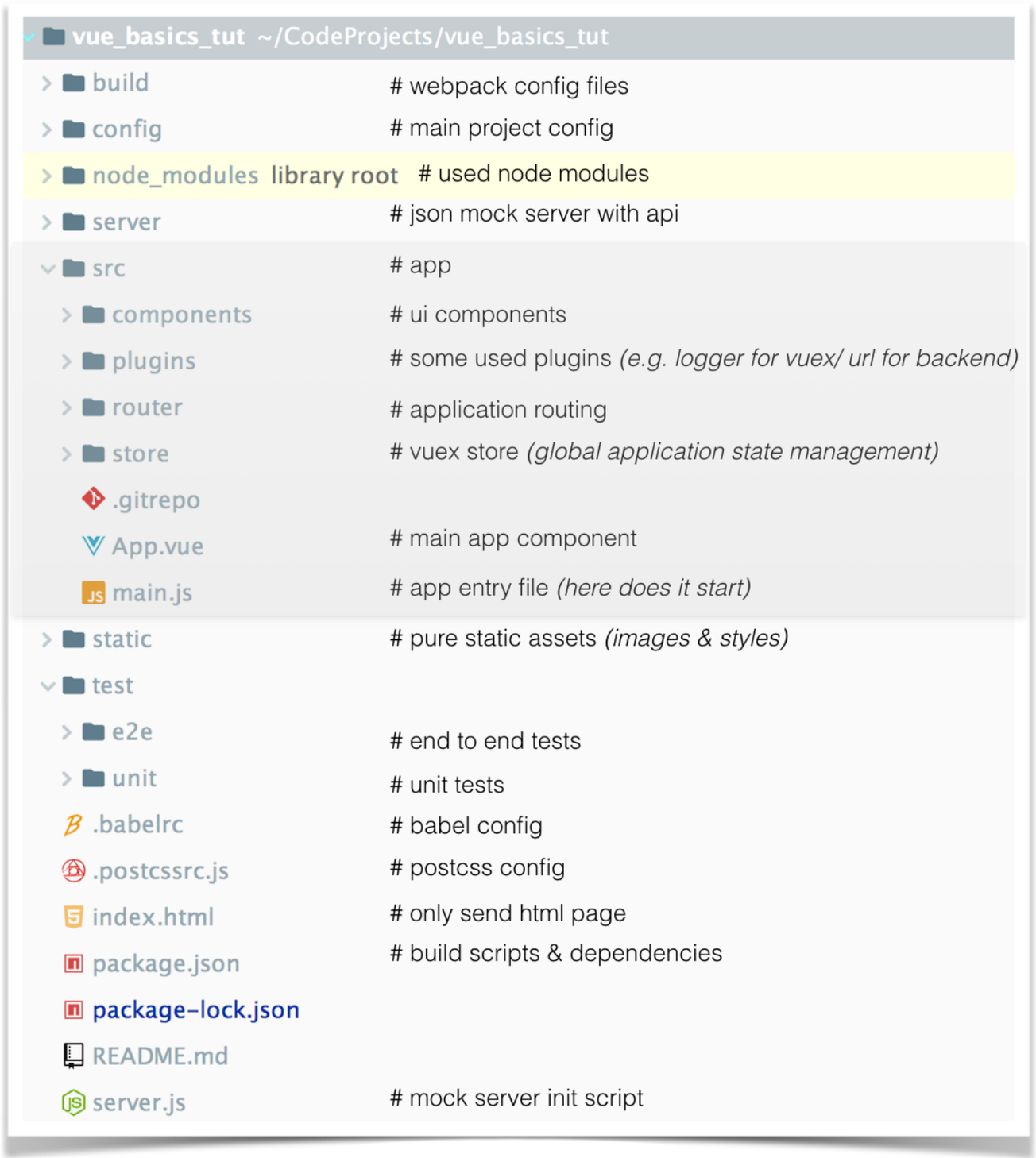
4 Routing

5 Knowledge Transfer: Finalize Your Kanban Board

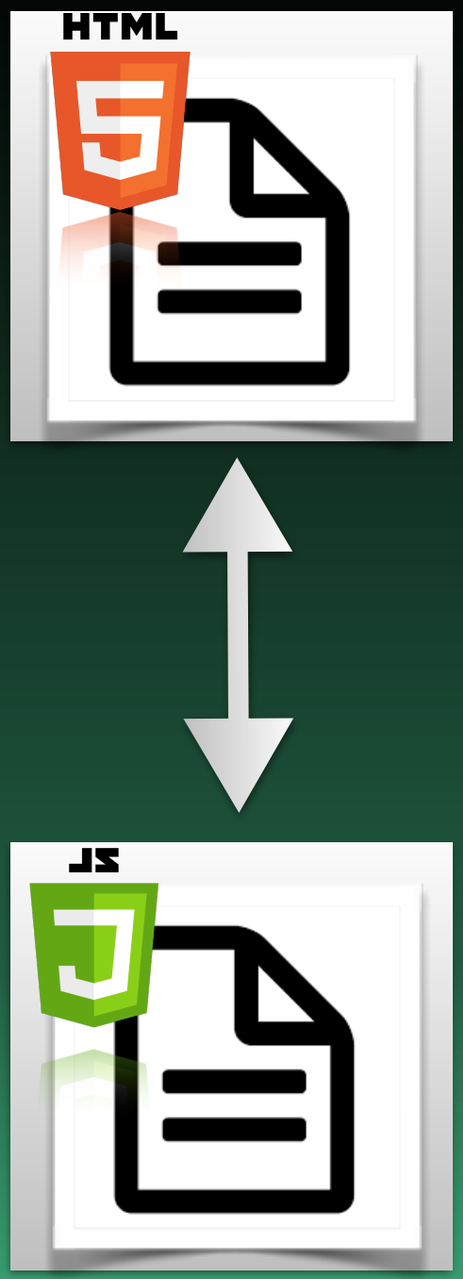
—> 5 exercises will deepen your understanding
ue.js

—> we created a Github Repo with branches
ercise

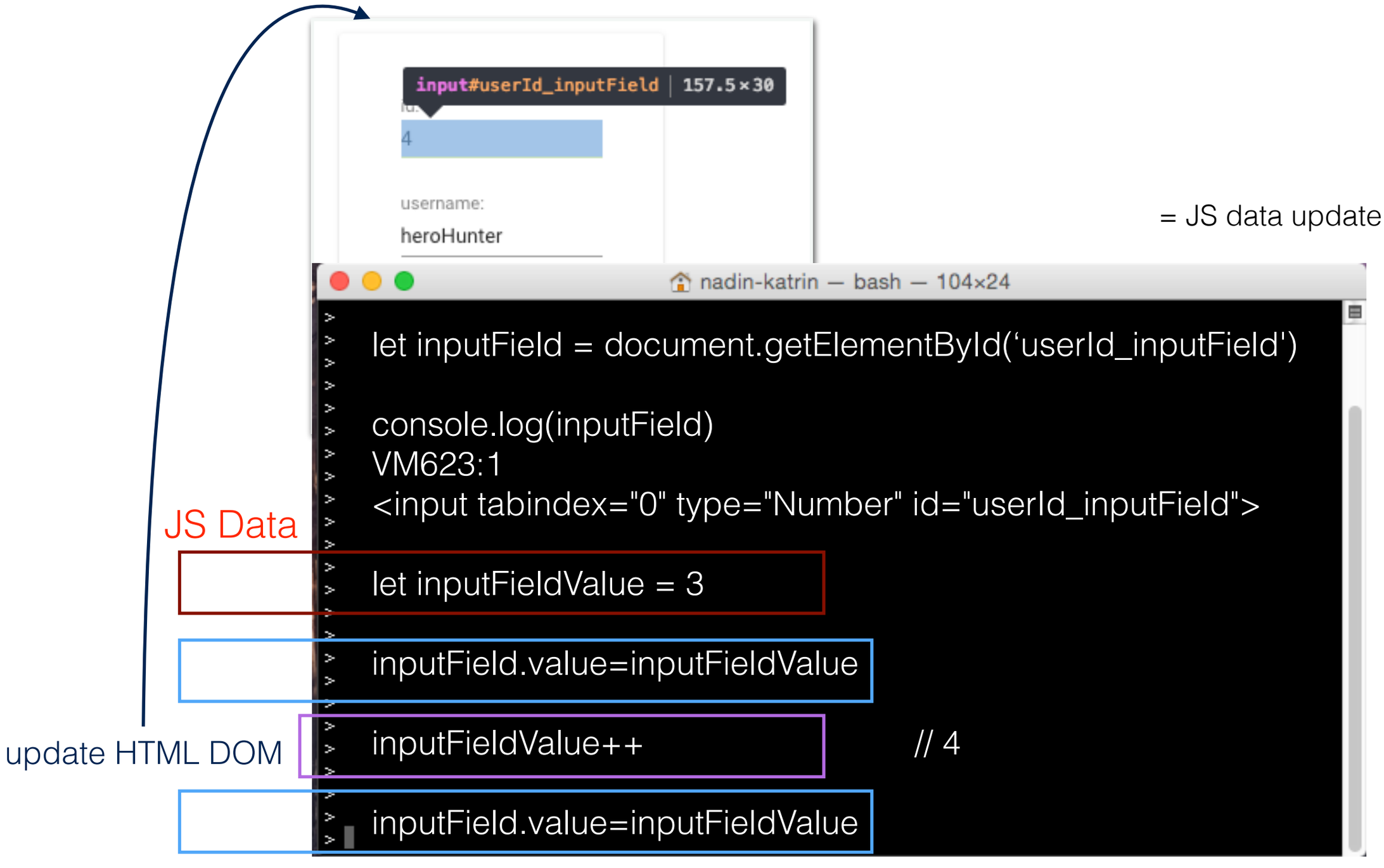
Project Structure



1. Data Binding in Vue.js



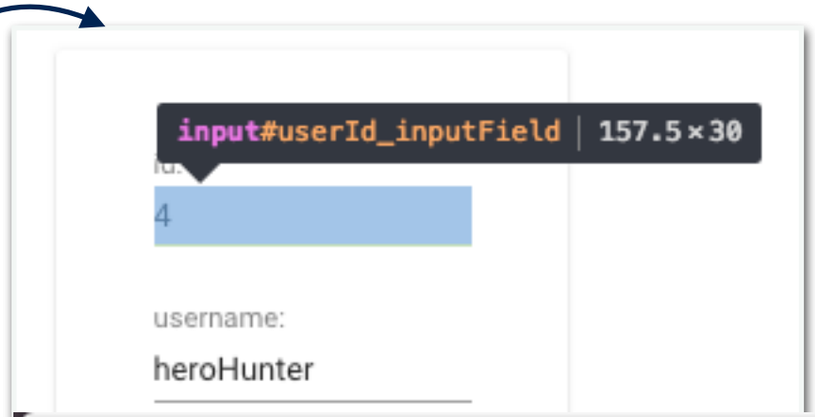
1. Data Binding



= JS data update -> update DOM

1. Data Binding

(one way)

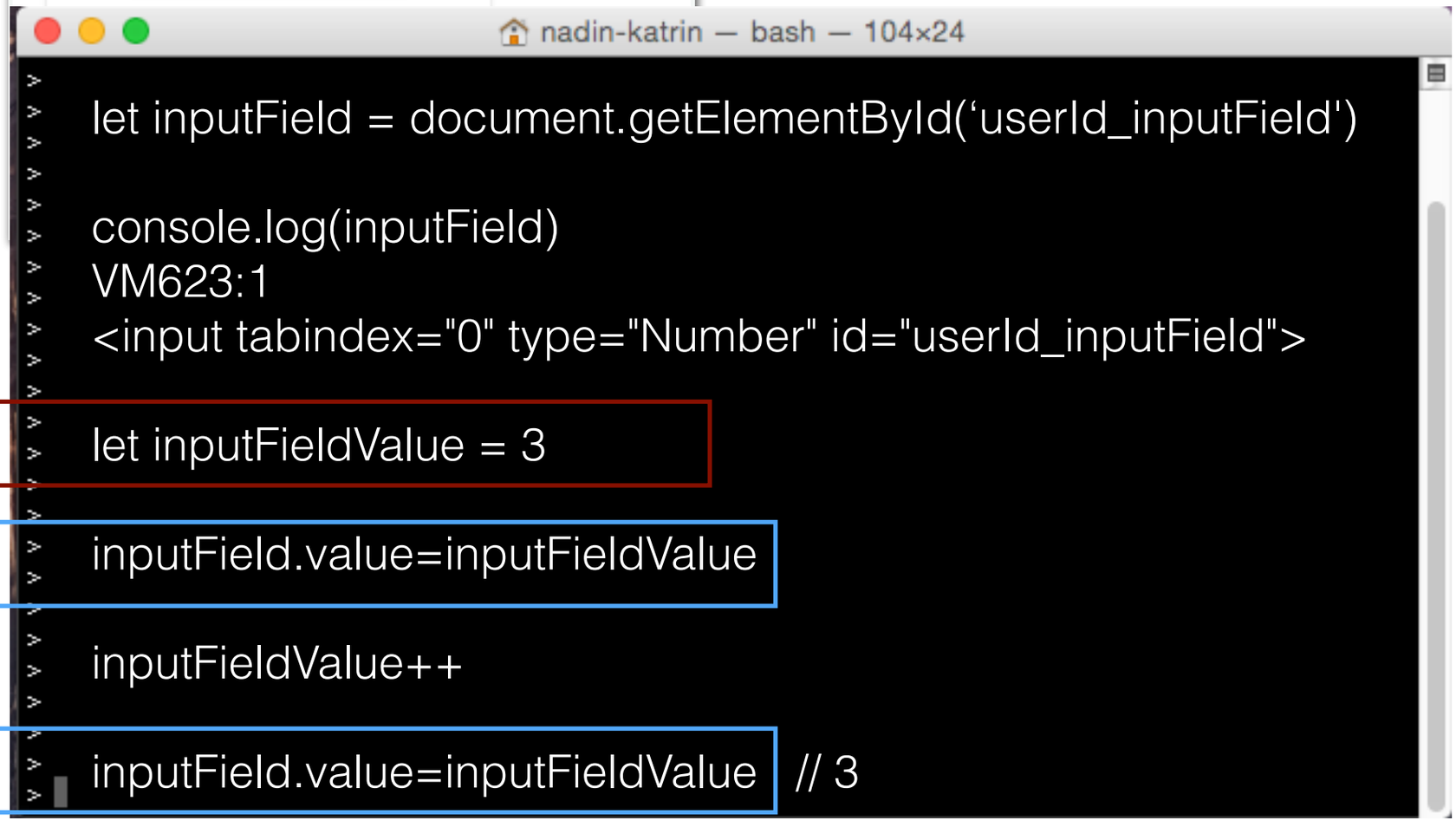


1. Find HTML DOM element
2. create variable with value *(data)* in JS
3. update DOM element by setting variable
4. update data value in JS
5. again step 3

= JS data update -> update DOM

JS Data

update HTML DOM



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

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

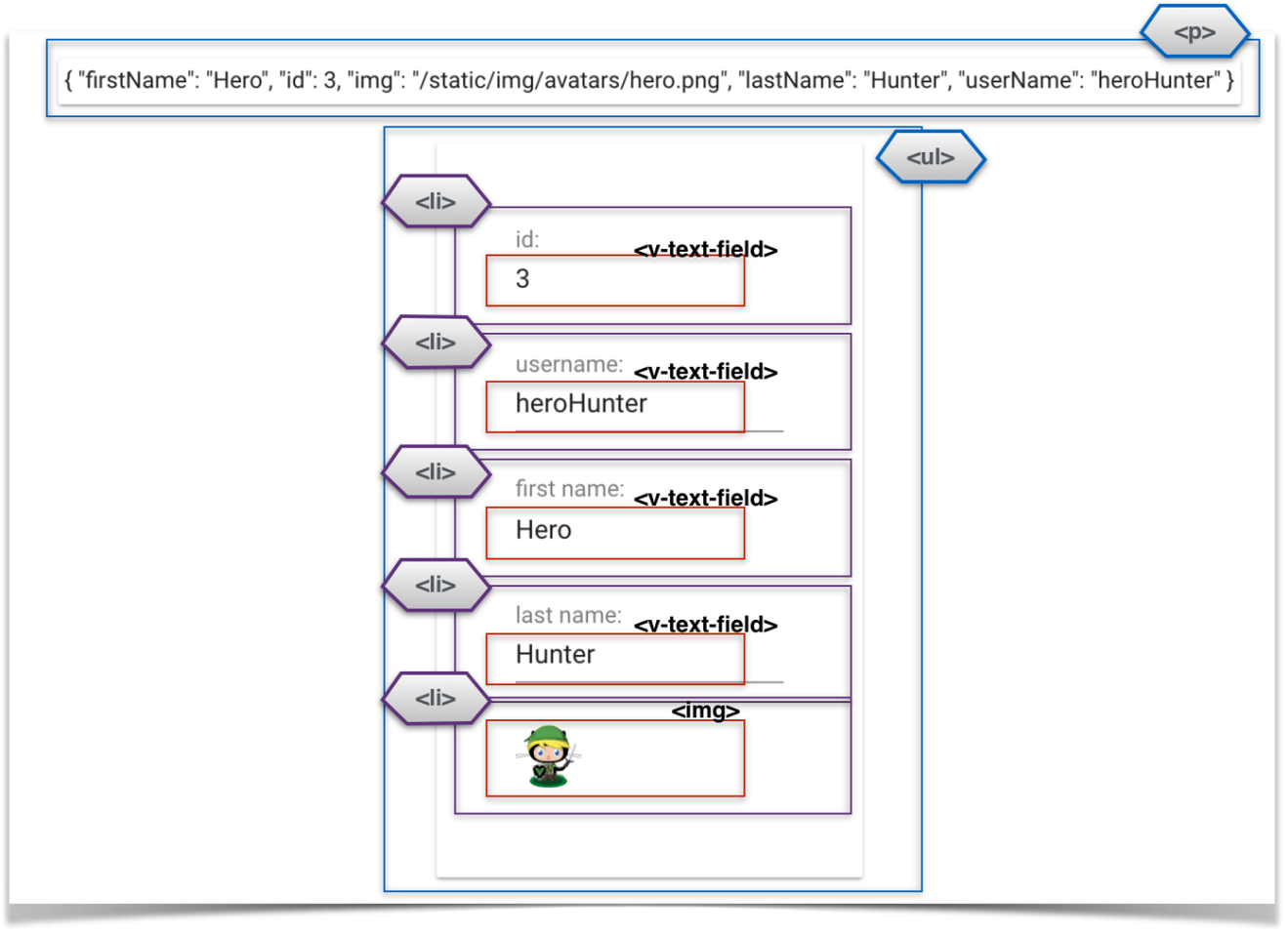
https://github.com/na018/vue_basics_tut/tree/origin/exercise/00_data_binding

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

In src/components/pages/UserAdmin.vue:

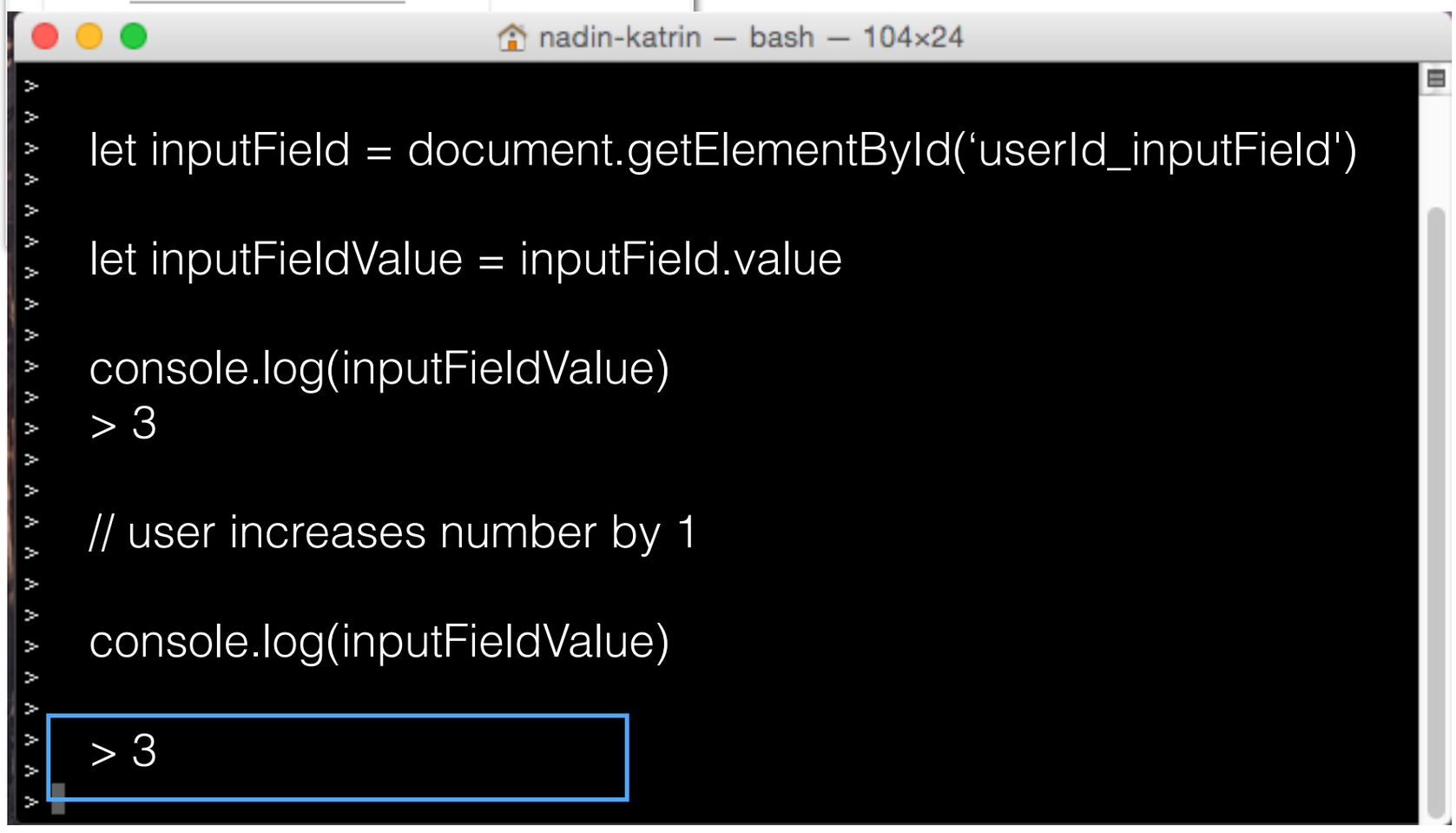
- create onePerson Object with userName, firstName, lastName, img as Object attributes (in <script> data())
- display the onePerson Object in a paragraph <p> element (in <template>)
- create an unordered list <ul> & show all onePerson attributes in a (<v-text-field>|value) and the image (<img>|src) with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



1. Data Binding

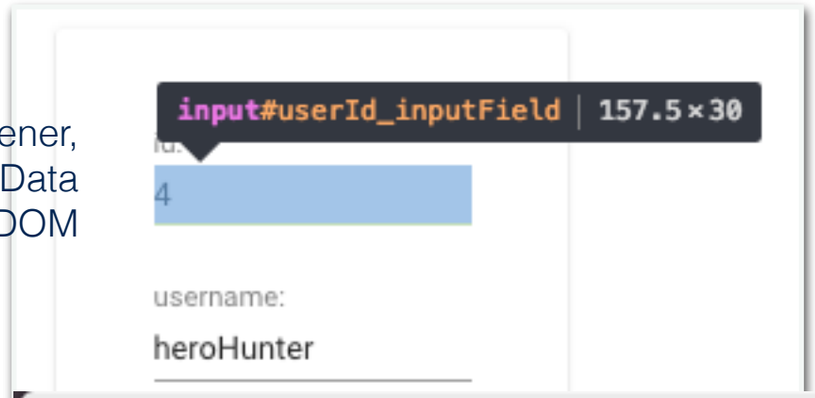
User increases Number, but no change in JS Data

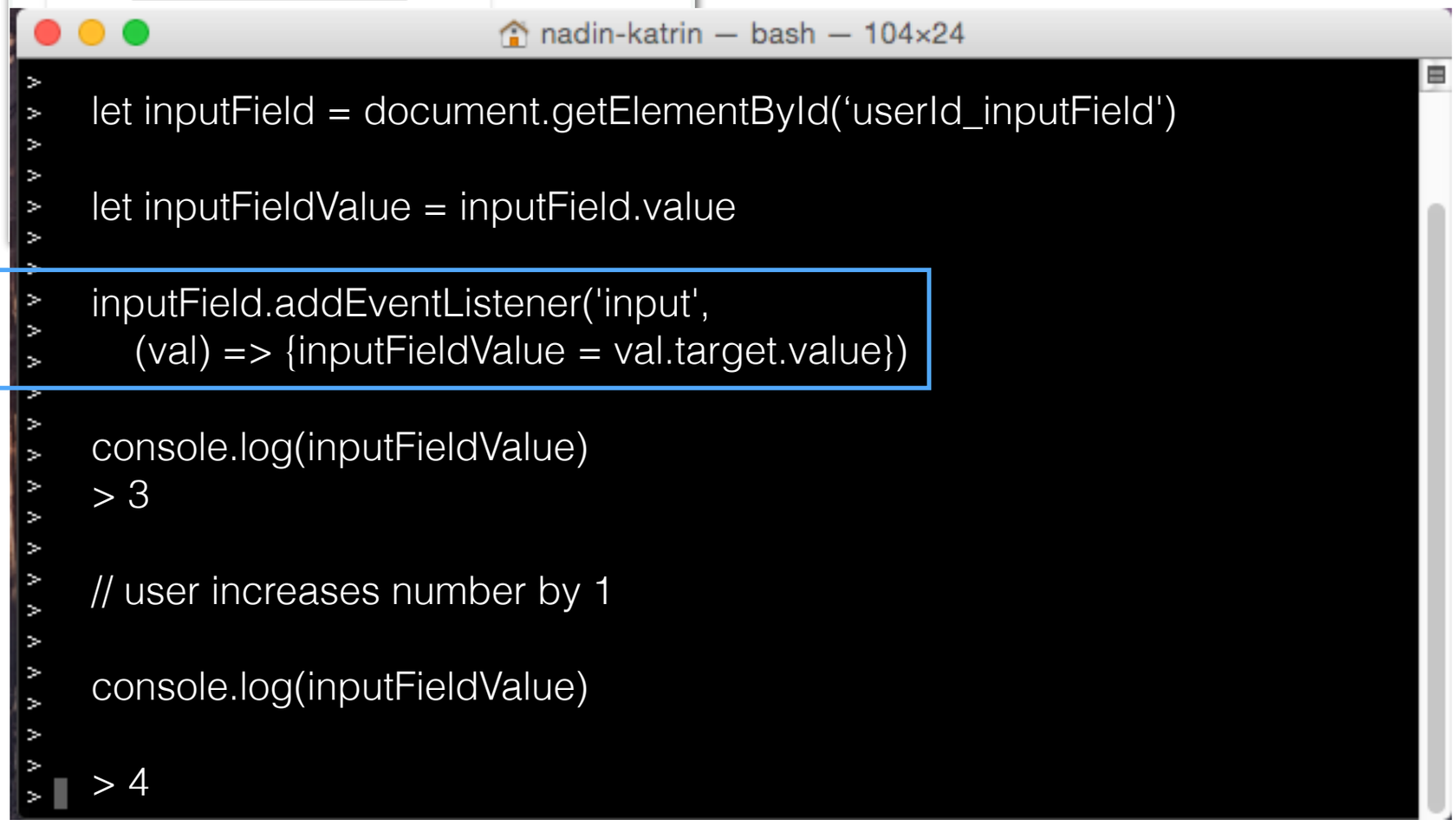


1. Data Binding

You need to listen to events that your users call

Attach event listener,
that updates your JS Data
when a user changes the DOM





HTML DOM

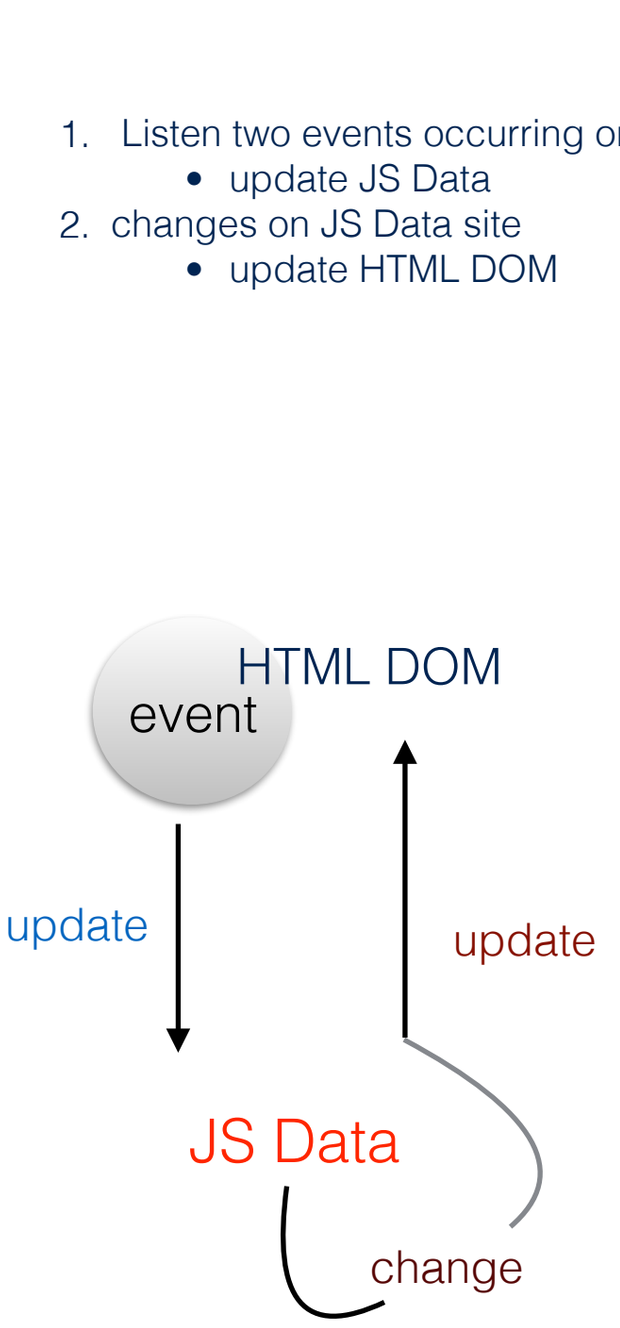
event

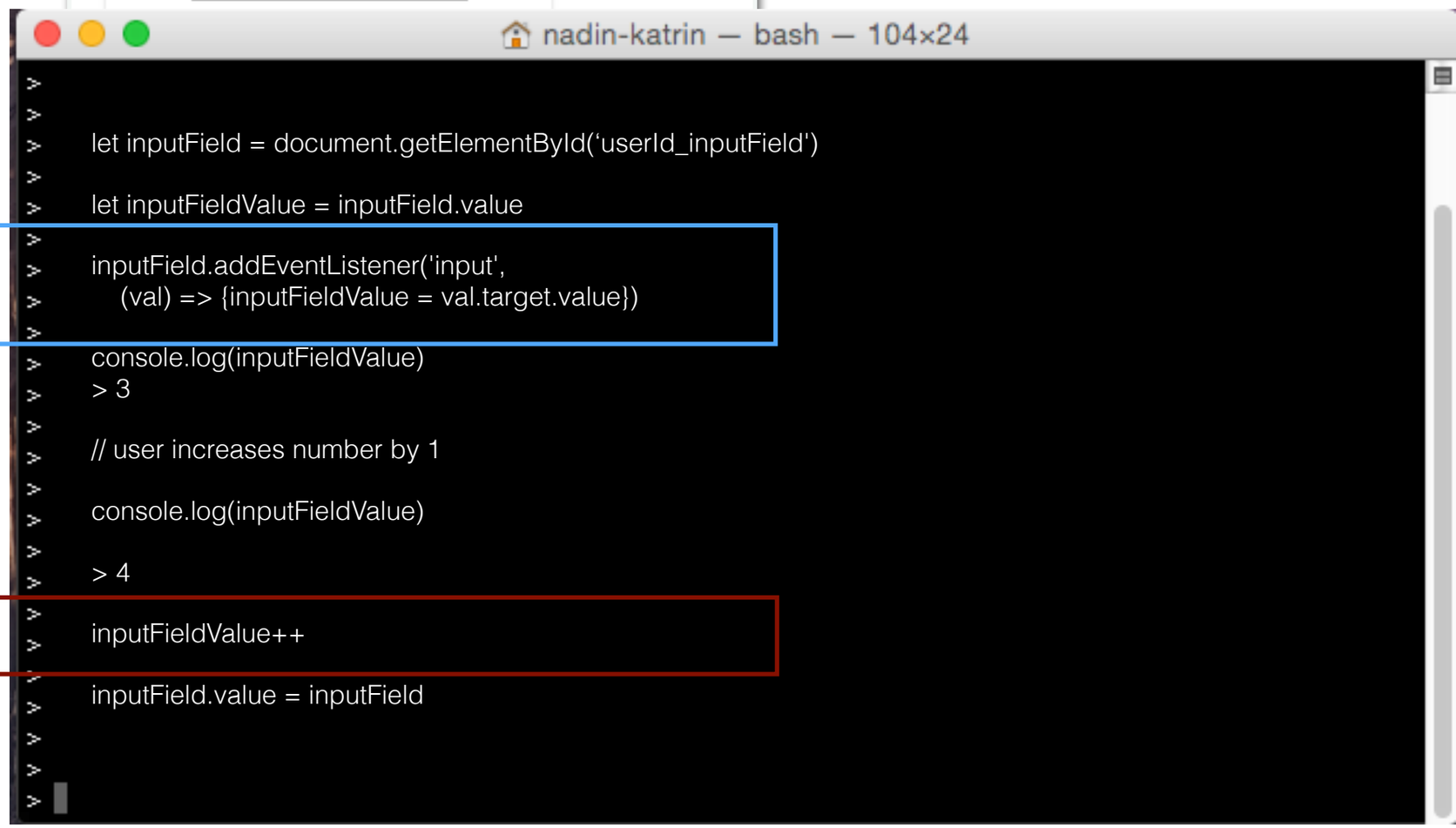
update

JS Data

1. Data Binding: *two-way*

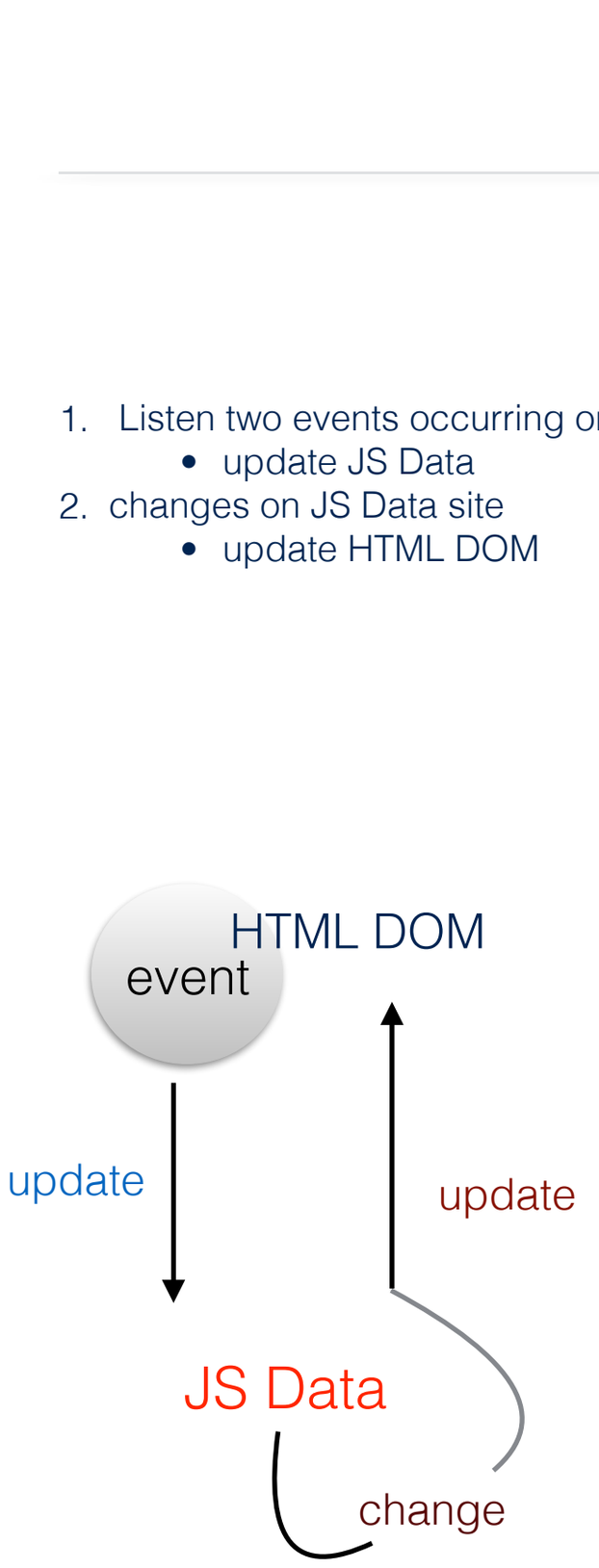
1. Listen two events occurring on HTML DOM site
    - update JS Data
2. changes on JS Data site
    - update HTML DOM

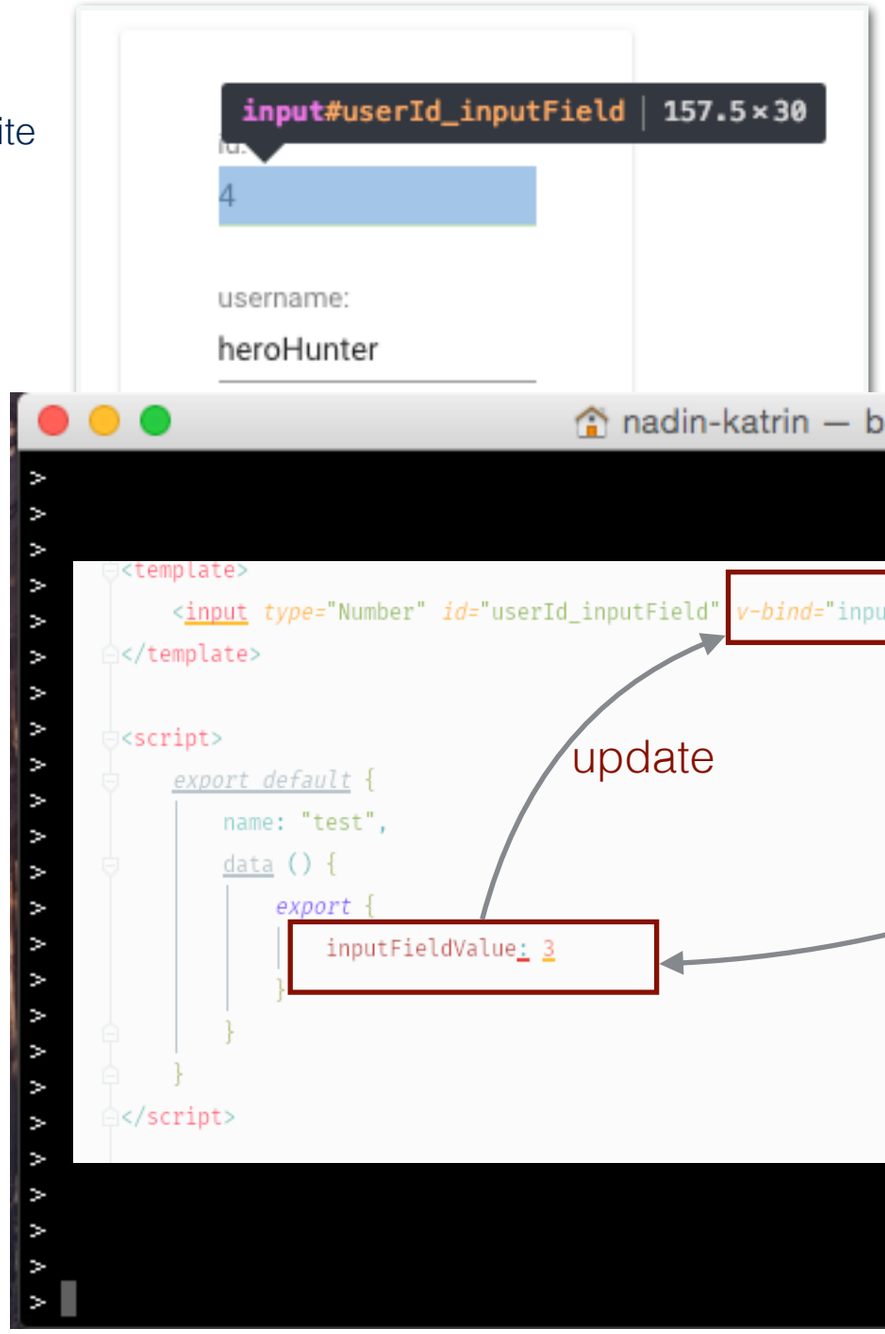




1. Data Binding: *two-way in Vue*

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



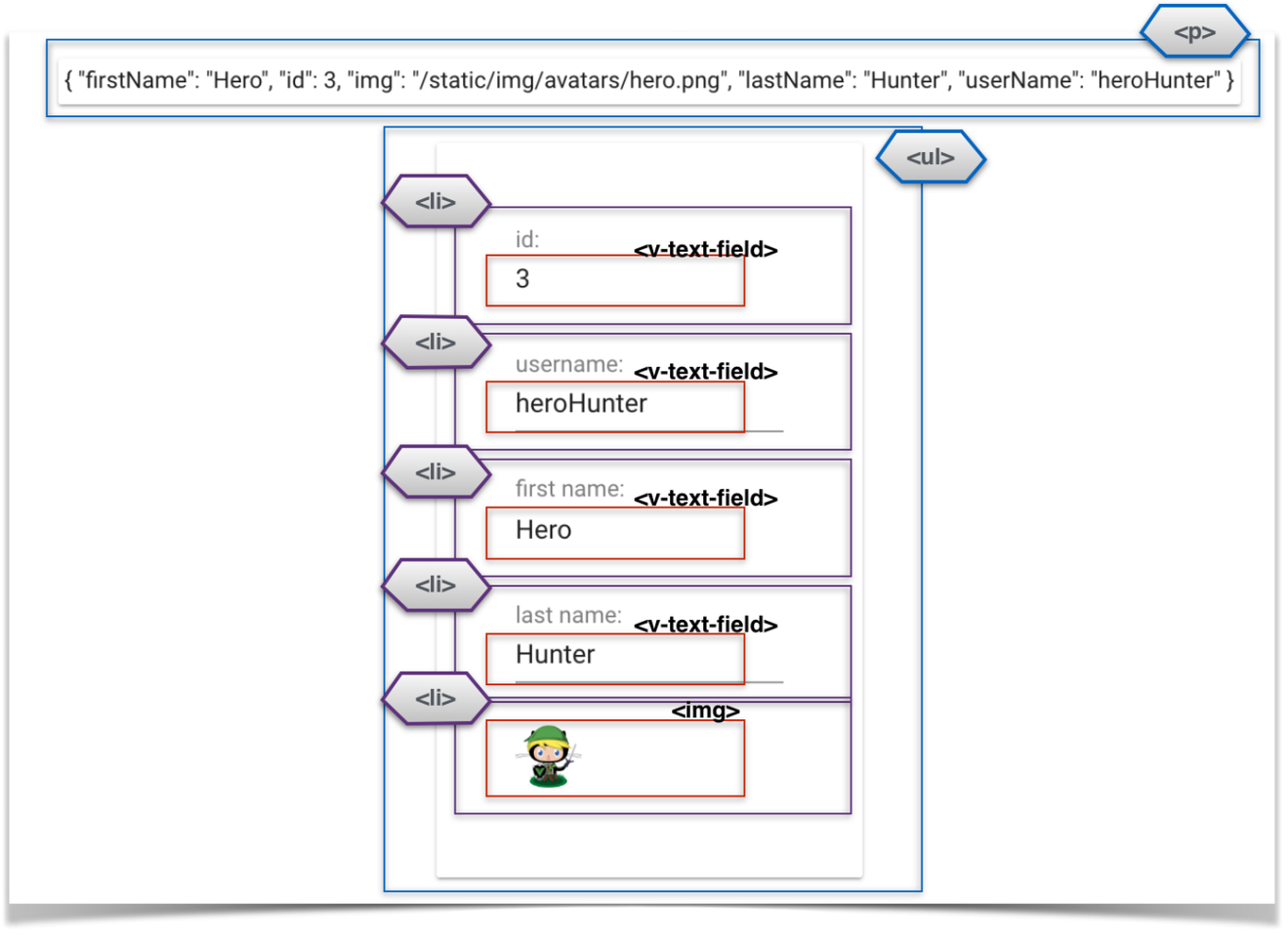


https://github.com/na018/vue_basics_tut/tree/origin/exercise/00_data_binding

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")



https://github.com/na018/vue_basics_tut/tree/origin/exercise/00_data_binding

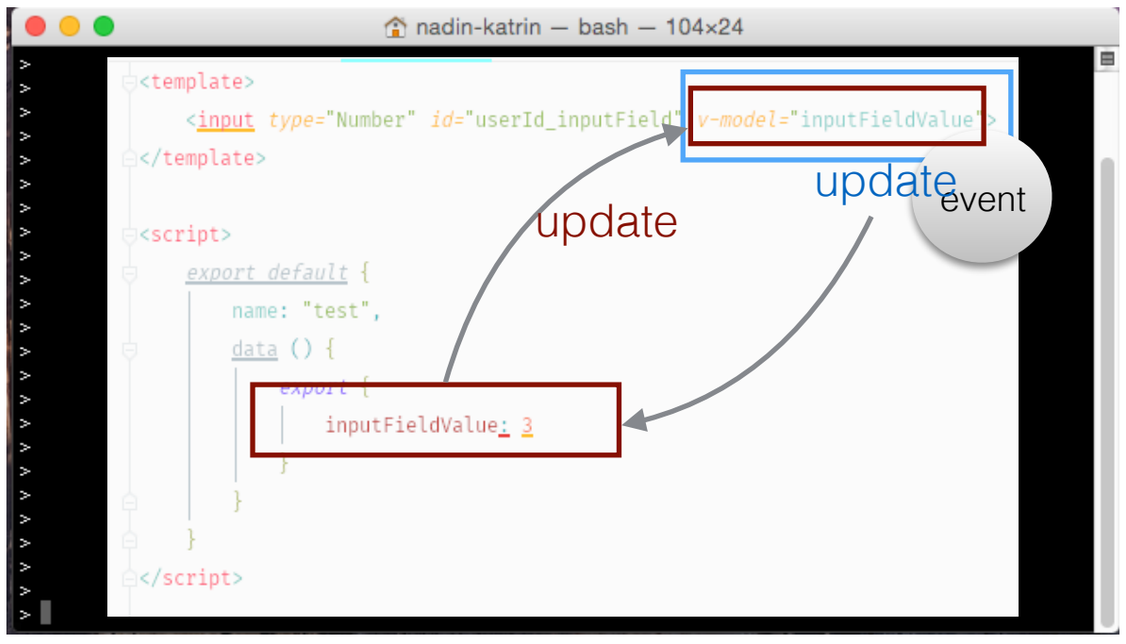
1. Data Binding in Vue: v-model *(two way)* -> Exercise 1

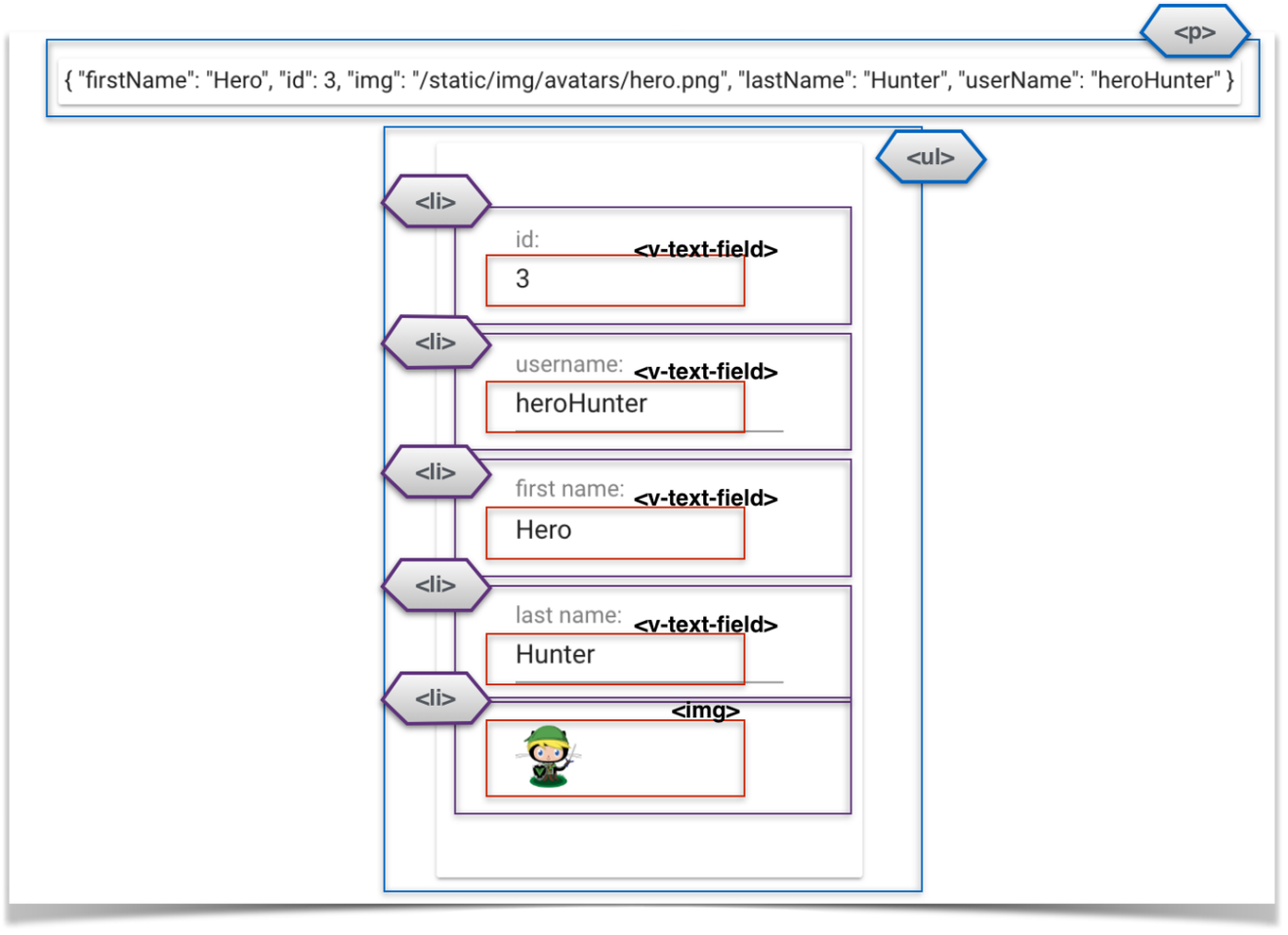
-> v-model makes your life a lot easier

- 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

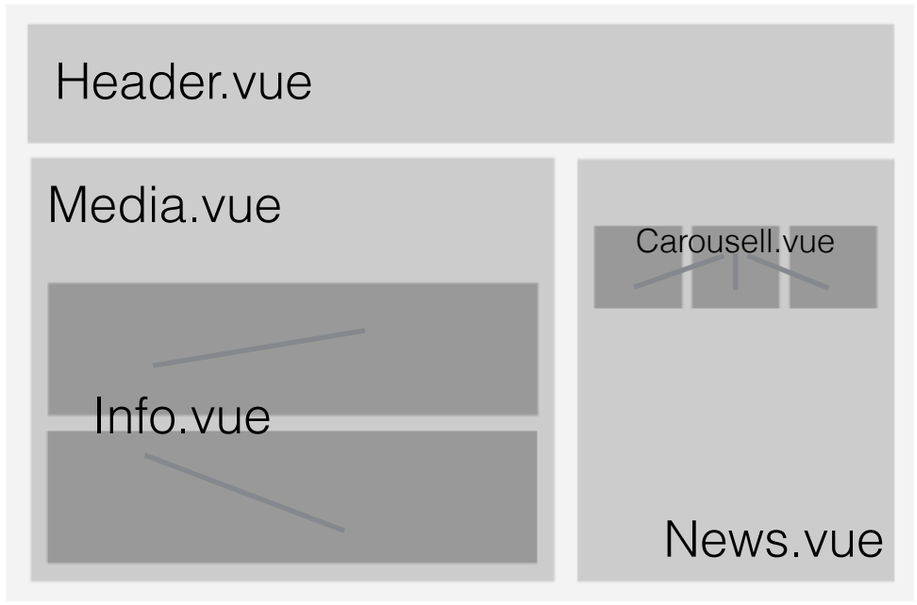


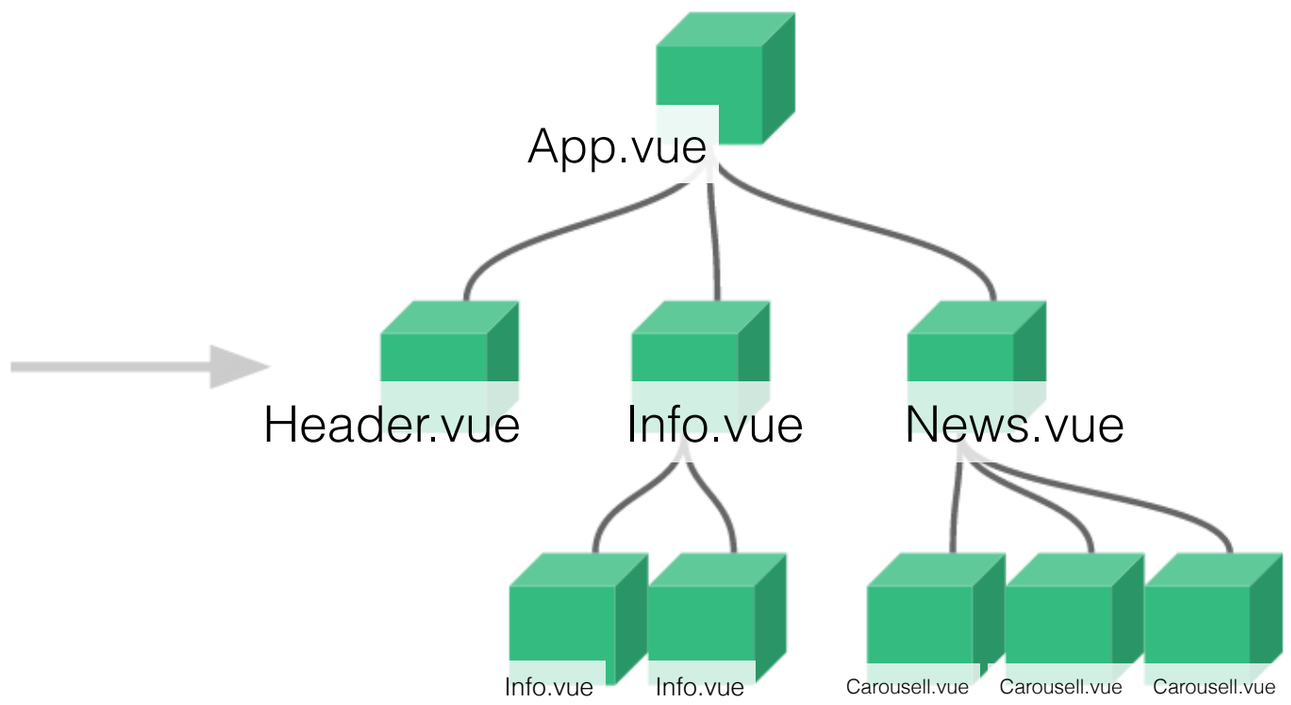


2. Components in Vue.js:

add modularity, reusability, structure

App.vue





2. Components in Vue.js:

add modularity, reusability, structure

parent.vue

```vue
<template>
    <child-component name="Jessi" :age="4" place="New York"></child-component>
</template>

<script>
    import ChildComponent from "./child";
    export default {
        name: "parent",
        components: {ChildComponent}
    }
</script>
```

Jessi is 4 years old and lives in New York

child.vue

```vue
<template>
    <div>
        <p>{{name}} is {{age}} years old and lives in {{place}}</p>
    </div>
</template>

<script>
    export default {
        name: "child-component",
        props: {
            age: {
                type: Number,
                default: 0
            },
            name: {
                type: String,
                default: 'Pete',
                required: true
            },
            place: {
                type: String,
                default: 'Stuttgart'
            }
        },
    }
</script>
```

2. Components in Vue.js:

add modularity, reusability, structure

parent.vue

```vue
<template>
    <child-component name="Jessi" :age="4" place="New York"></child-component>
</template>

<script>
    import ChildComponent from "./child";
    export default {
        name: "parent",
        components: {ChildComponent}
    }
</script>
```

Jessi is 4 years old and lives in New York

child.vue

```vue
<template>
    <div>
        <p>{{name}} is {{age}} years old and lives in {{place}}</p>
    </div>
</template>

<script>
    export default {
        name: "child-component",
        props: {
            age: {
                type: Number,
                default: 0
            },
            name: {
                type: String,
                default: 'Pete',
                required: true
            },
            place: {
                type: String,
                default: 'Stuttgart'
            }
        },
    }
</script>
```

2. Components in Vue.js:

add modularity, reusability, structure

parent.vue

```vue
<template>
    <child-component name="Jessi" :age="4" place="New York"></child-component>
</template>

<script>
    import ChildComponent from "./child";
    export default {
        name: "parent",
        components: {ChildComponent}
    }
</script>
```

data

child.vue

```vue
<template>
    <div>
        <p>{{name}} is {{age}} years old and lives in {{place}}</p>
    </div>
</template>

<script>
    export default {
        name: "child-component",
        props: {
            age: {
                type: Number,
                default: 0
            },
            name: {
                type: String,
                default: 'Pete',
                required: true
            },
            place: {
                type: String,
                default: 'Stuttgart'
            }
        },
    }
</script>
```

Jessi is 4 years old and lives in New York

2. Components in Vue.js:

add modularity, reusability, structure

parent.vue

```vue
<template>
    <child-component name="Jessi" :age="4" place="New York"></child-component>
</template>

<script>
    import ChildComponent from "./child";
    export default {
        name: "parent",
        components: {ChildComponent}
    }
</script>
```

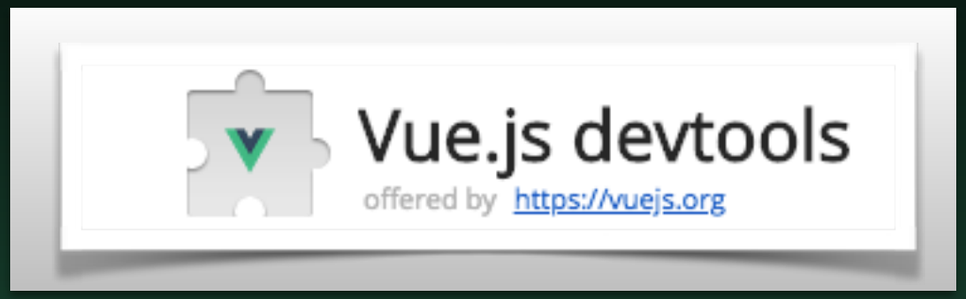
Jessi is 4 years old and lives in New York

child.vue

```vue
<template>
    <div>
        <p>{{name}} is {{age}} years old and lives in {{place}}</p>
    </div>
</template>

<script>
    export default {
        name: "child-component",
        props: {
            age: {
                type: Number,
                default: 0
            },
            name: {
                type: String,
                default: 'Pete',
                required: true
            },
            place: {
                type: String,
                default: 'Stuttgart'
            }
        },
    }
</script>
```

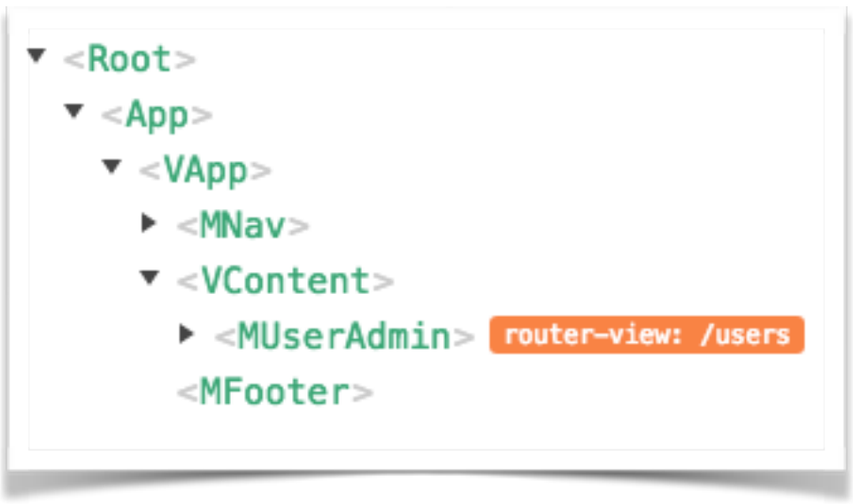
2. Components in Vue.js:

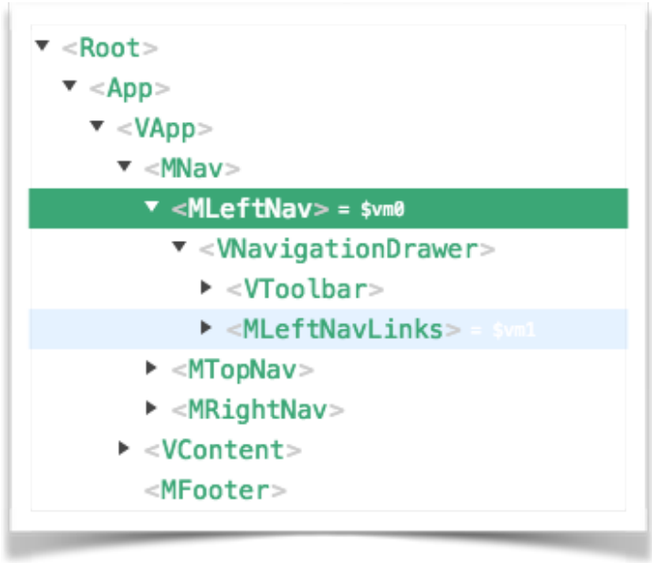
add modularity, reusability, structure



2. Components in Vue.js:

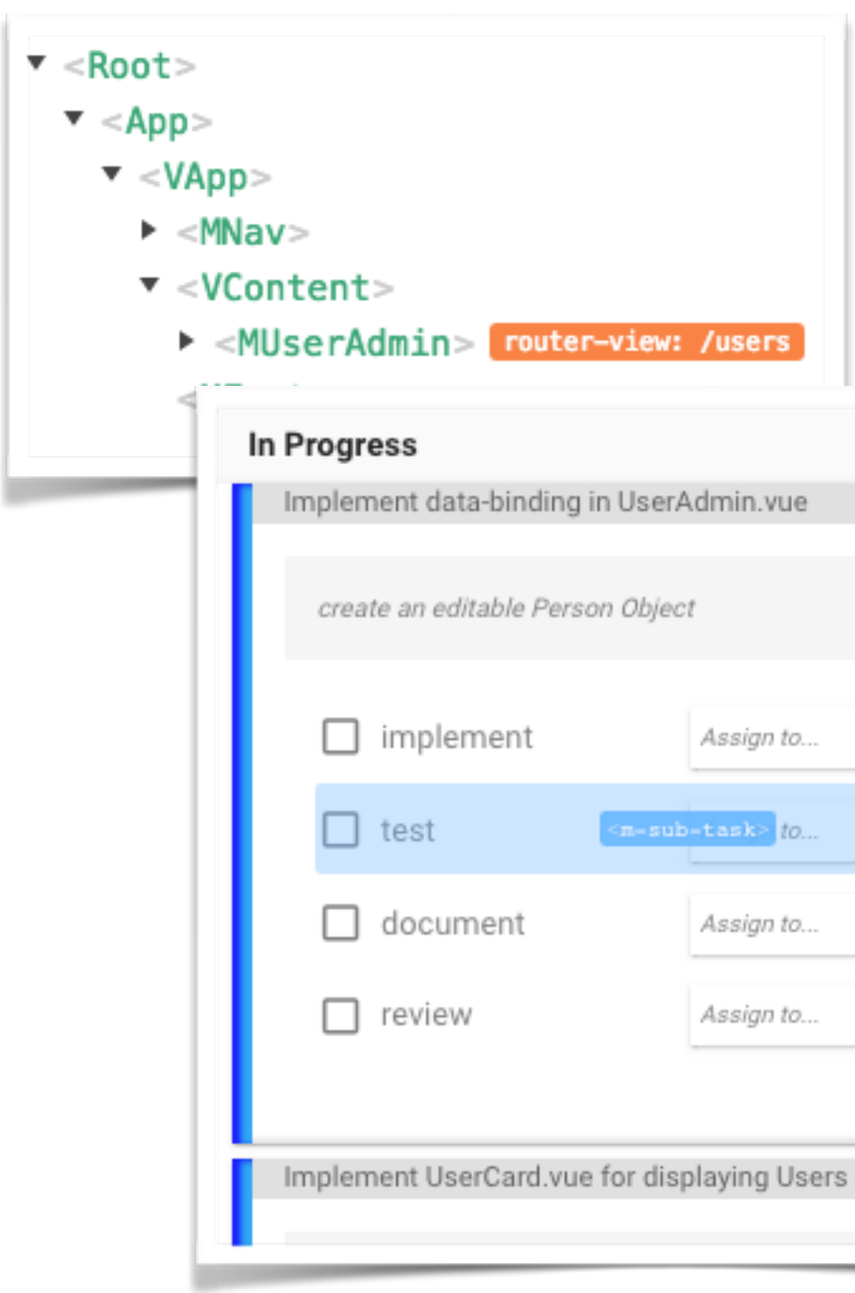
add modularity, reusability, structure

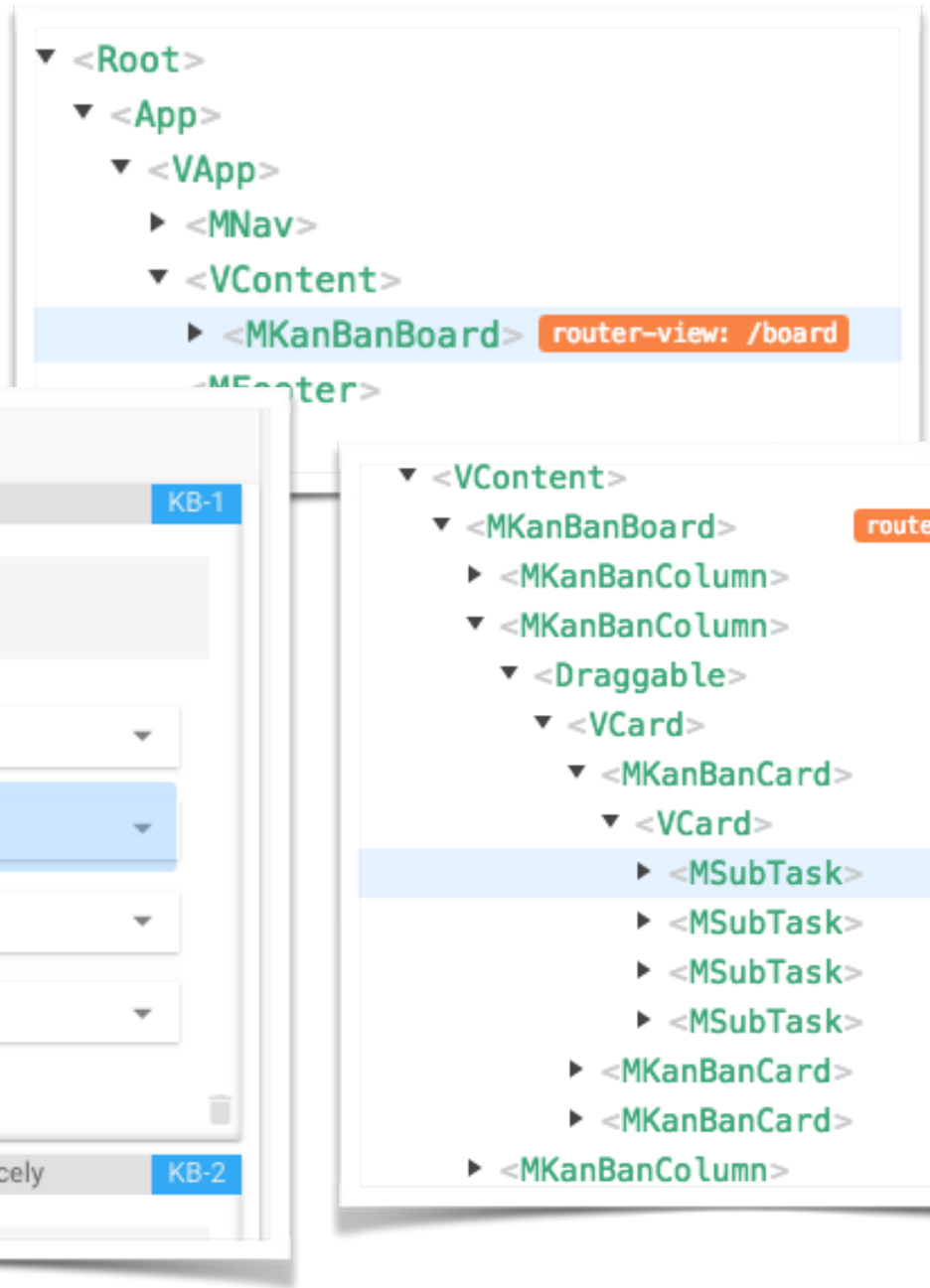


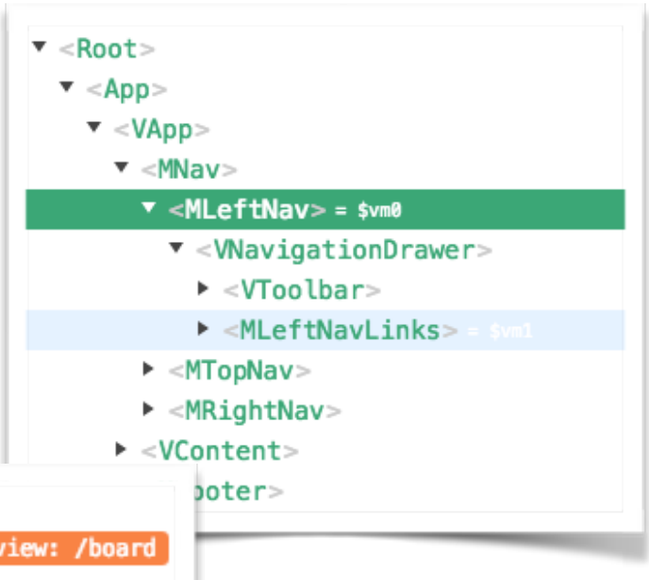


2. Components in Vue.js:

add modularity, reusability, structure







https://github.com/na018/vue_basics_tut/tree/origin/exercise/01_components

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>

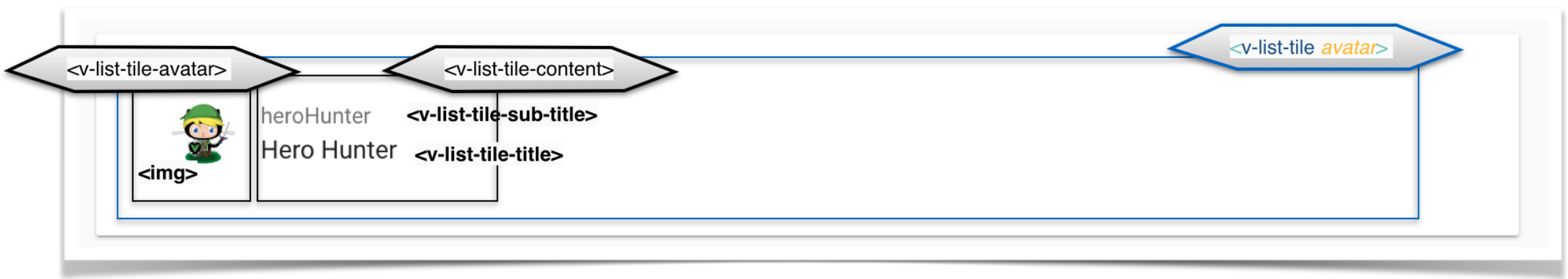
parent.vue

```vue
<template>
    <child-component name="Jessi" :age="4" place="New York"></child-component>
</template>

<script>
    import ChildComponent from "./child";
    export default {
        name: "parent",
        components: {ChildComponent}
    }
</script>
```

child.vue

```vue
<template>
    <div>
        <p>{{name}} is {{age}} years old and lives in {{place}}</p>
    </div>
</template>

<script>
    export default {
        name: "child-component",
        props: {
            age: {
                type: Number,
                default: 0
            },
            name: {
                type: String,
                default: 'Pete',
                required: true
            },
            place: {
                type: String,
                default: 'Stuttgart'
            }
        },
    }
</script>
```

UserCard.vue *(child)*



3. Directives (conditional, loops)

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

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

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

parent.vue

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

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

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

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

parent.vue

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

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

child.vue

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

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

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

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

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

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

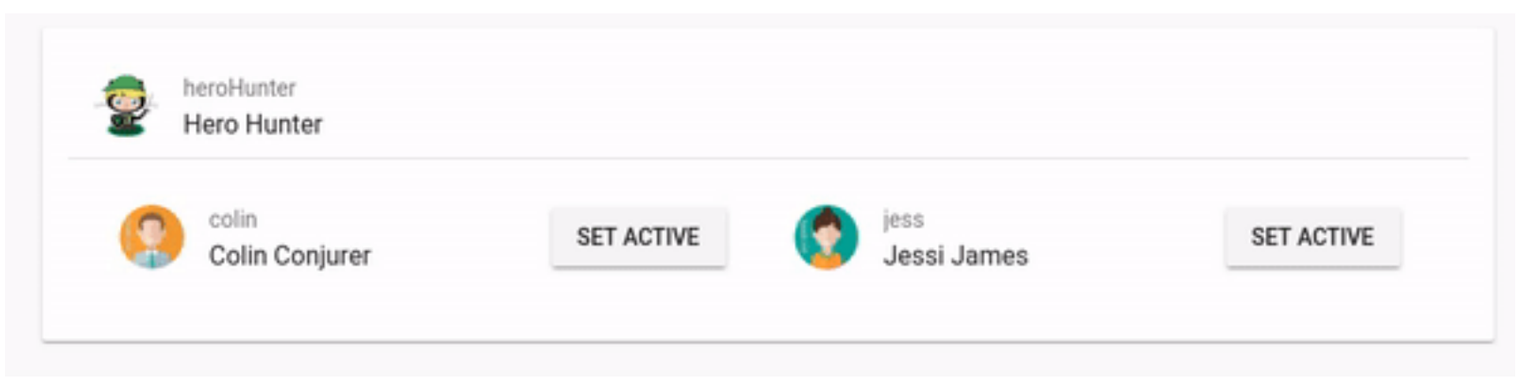
3. Directives (conditional, loops): Exercise 3

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 it's 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

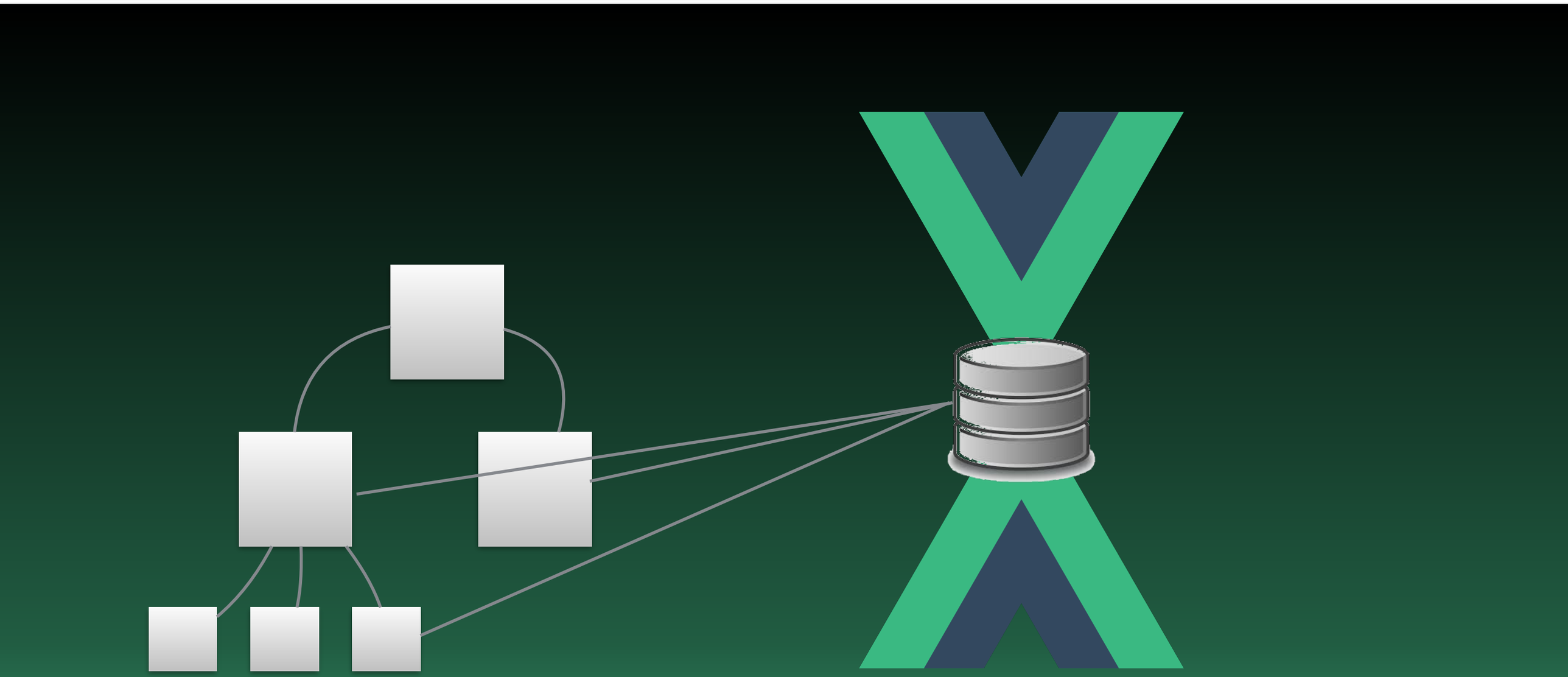



```
data() {
    return {
        activeUser: {
            userName: "heroHunter",
            id: 3,
            firstName: "Hero",
            lastName: "Hunter",
            img: "/static/img/avatars/hero.png",

        },
        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",
            },

        }
```

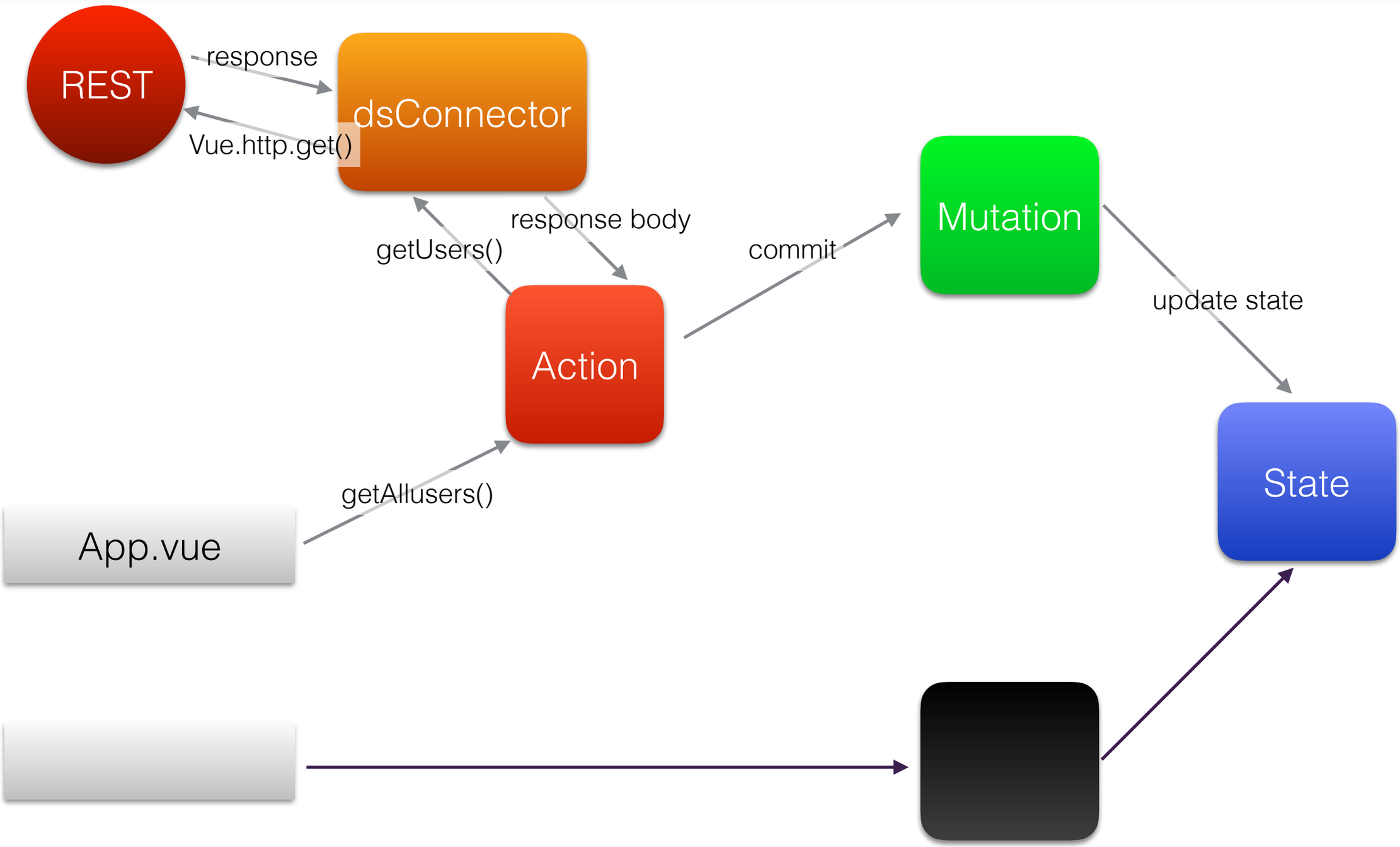
```
<li v-for="(user, index) in users" :key="index"> {{index}} : {{user}}</li>
```

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

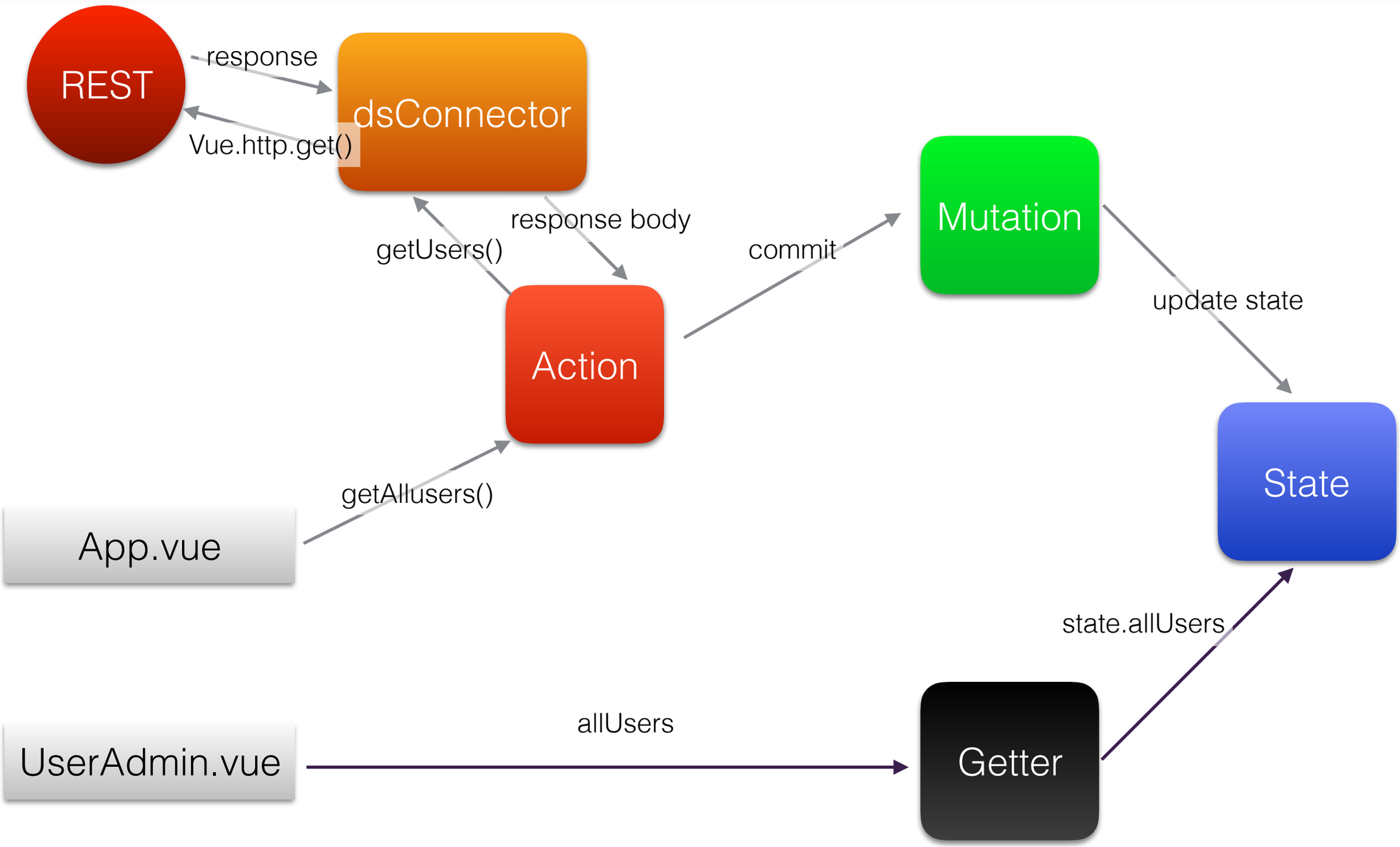
4. Global state management with Vuex



4. Global state management with Vuex



4. Global state management with Vuex



4. Global state management with Vuex

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()
  },
```

4. Global state management with Vuex

App.vue

```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()
  },
```

```vue
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,
}
```

4. Global state management with Vuex

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()
  },
```

```
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

dsConnector/children.js × modules/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
}
```

REST

4. Global state management with Vuex

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()
  },
```

```
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

dsConnector/children.js × modules/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
}
```

REST

4. Global state management with Vuex

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()
  },
```

```
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

dsConnector/children.js × modules/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
}
```

REST

4. Global state management with Vuex

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()
  },
```

```
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,
}
```

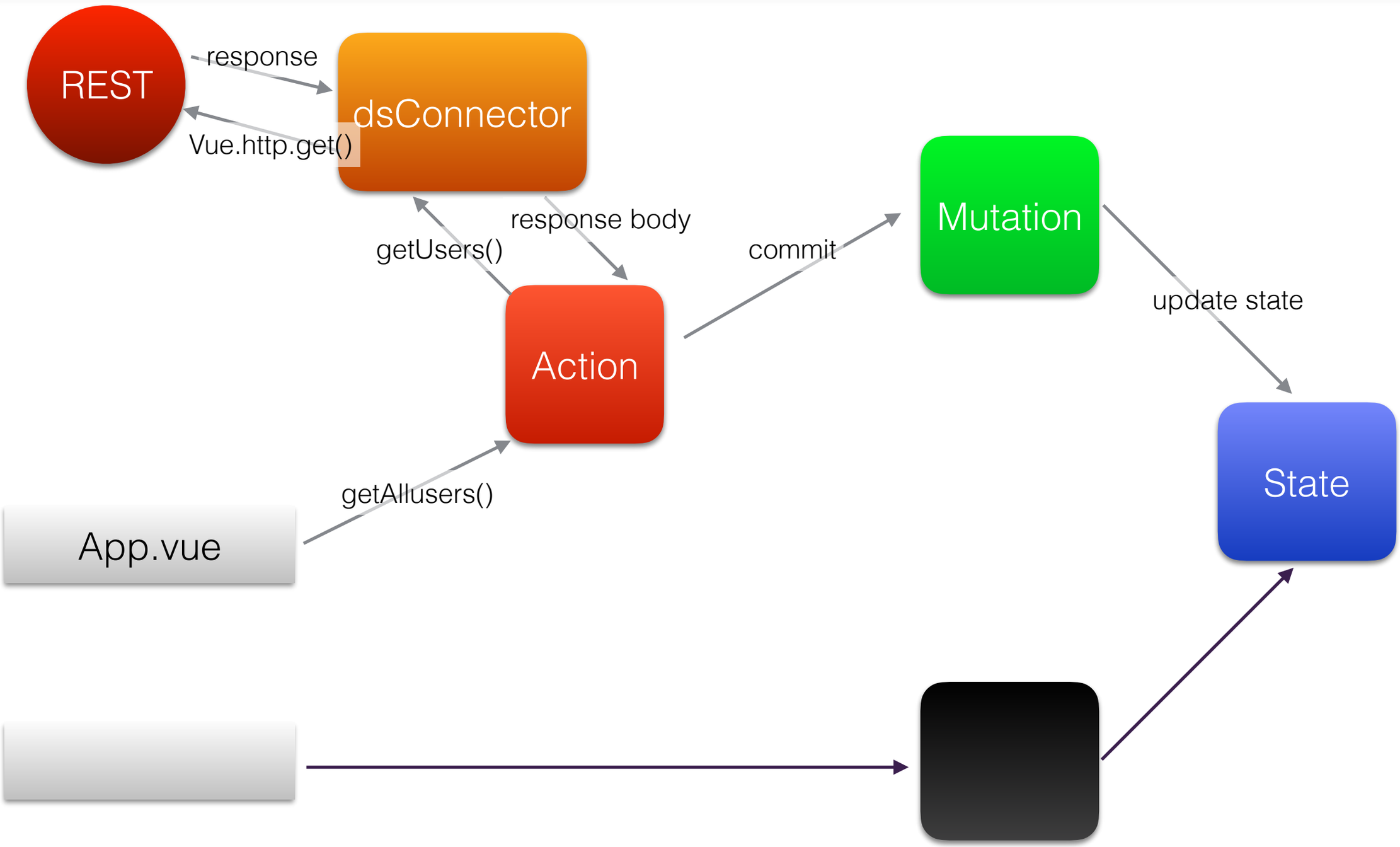
dsConnector

dsConnector/children.js × modules/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
}
```

REST

4. Global state management with Vuex

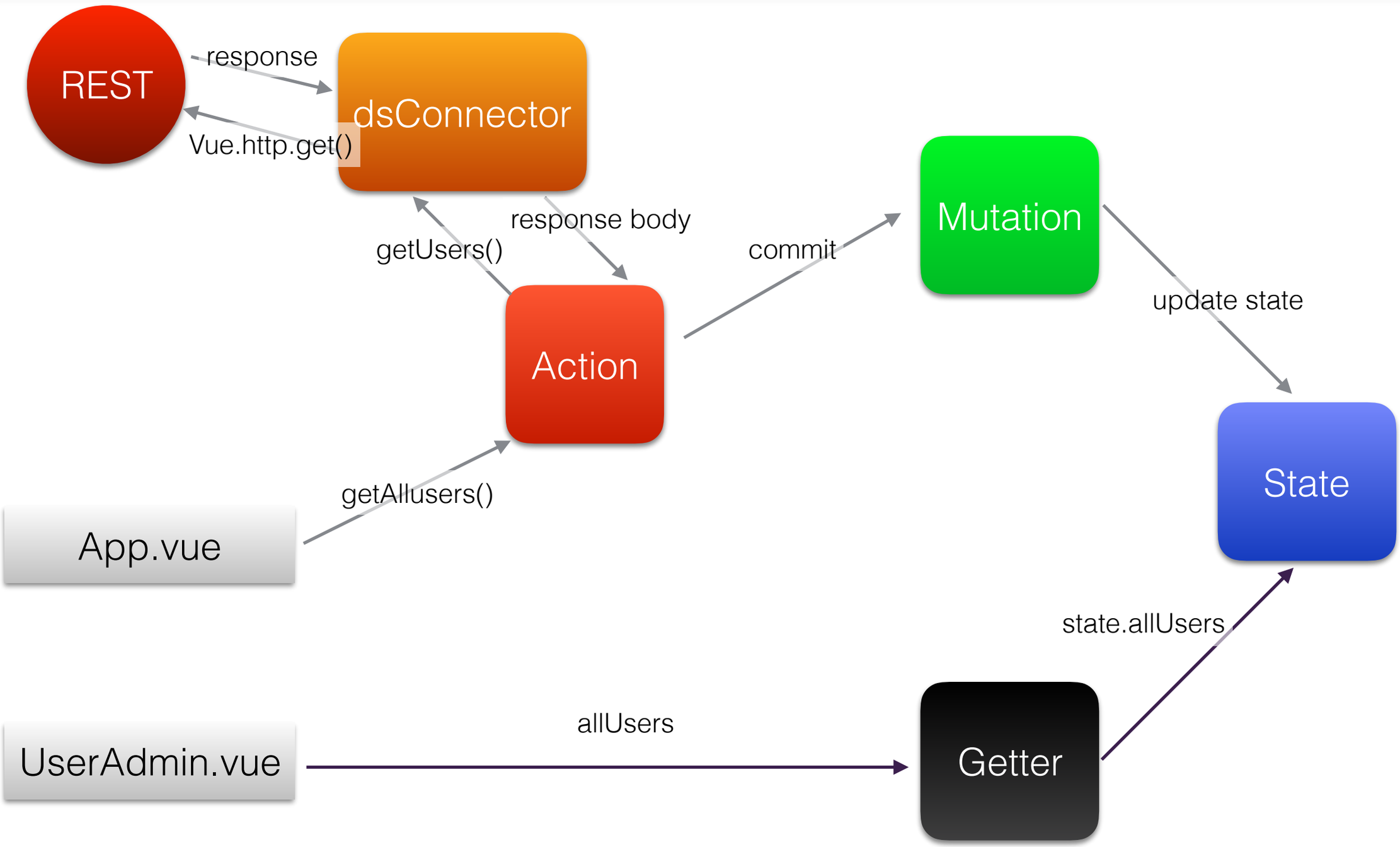


4. Global state management with Vuex

```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>
```

```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,
}
```

4. Global state management with Vuex



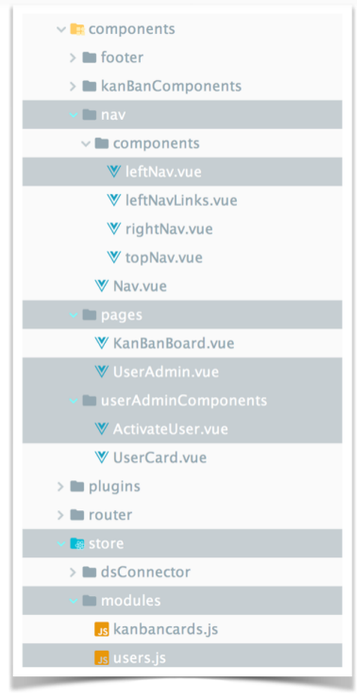
4. Global state management with Vuex: Exercise 4

in src/components/pages/UserAdmin.vue:

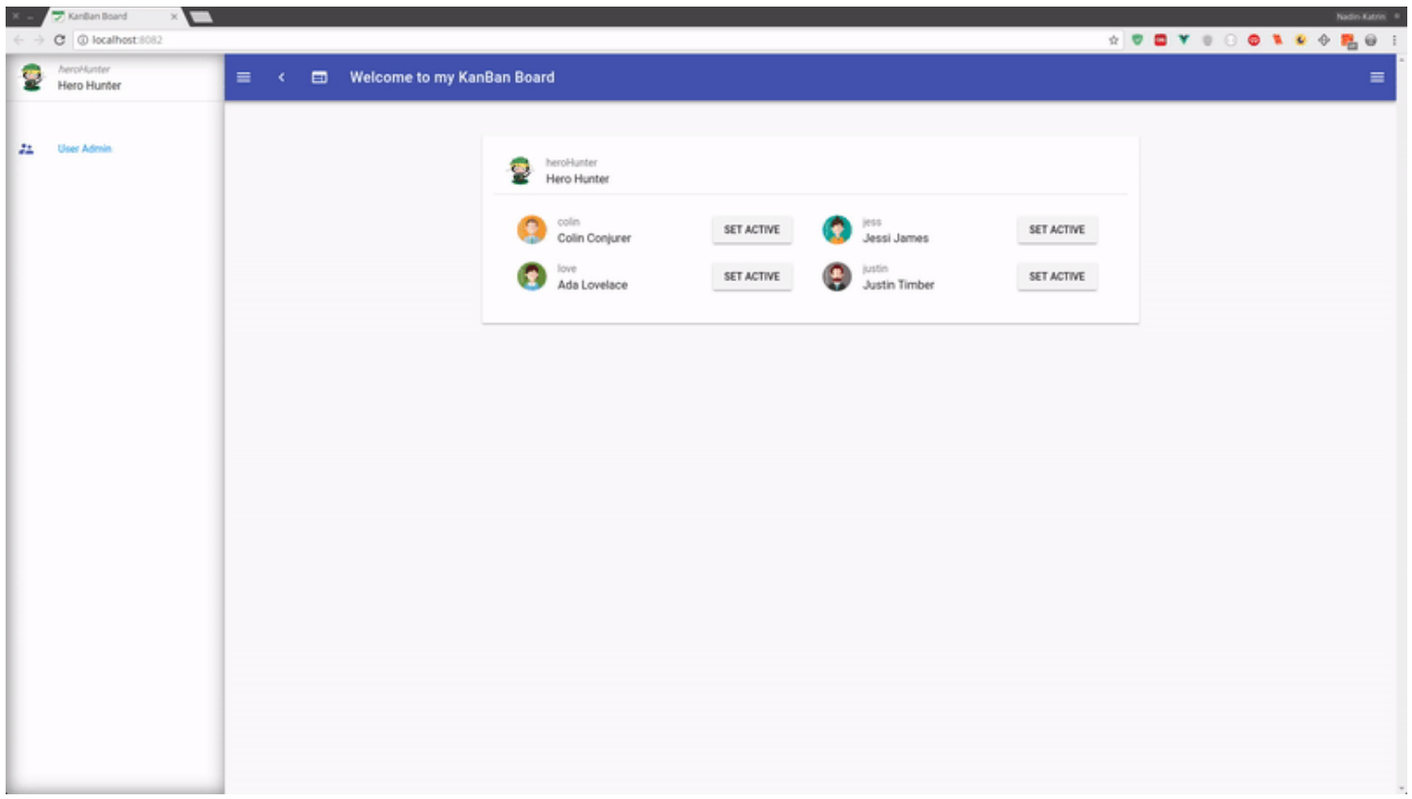
- Use the Getter allUsers & activeUser from Vuex store src/store/modules/users.js for receiving the global application state

In src/components/userAdminComponents/ActivateUser.vue:

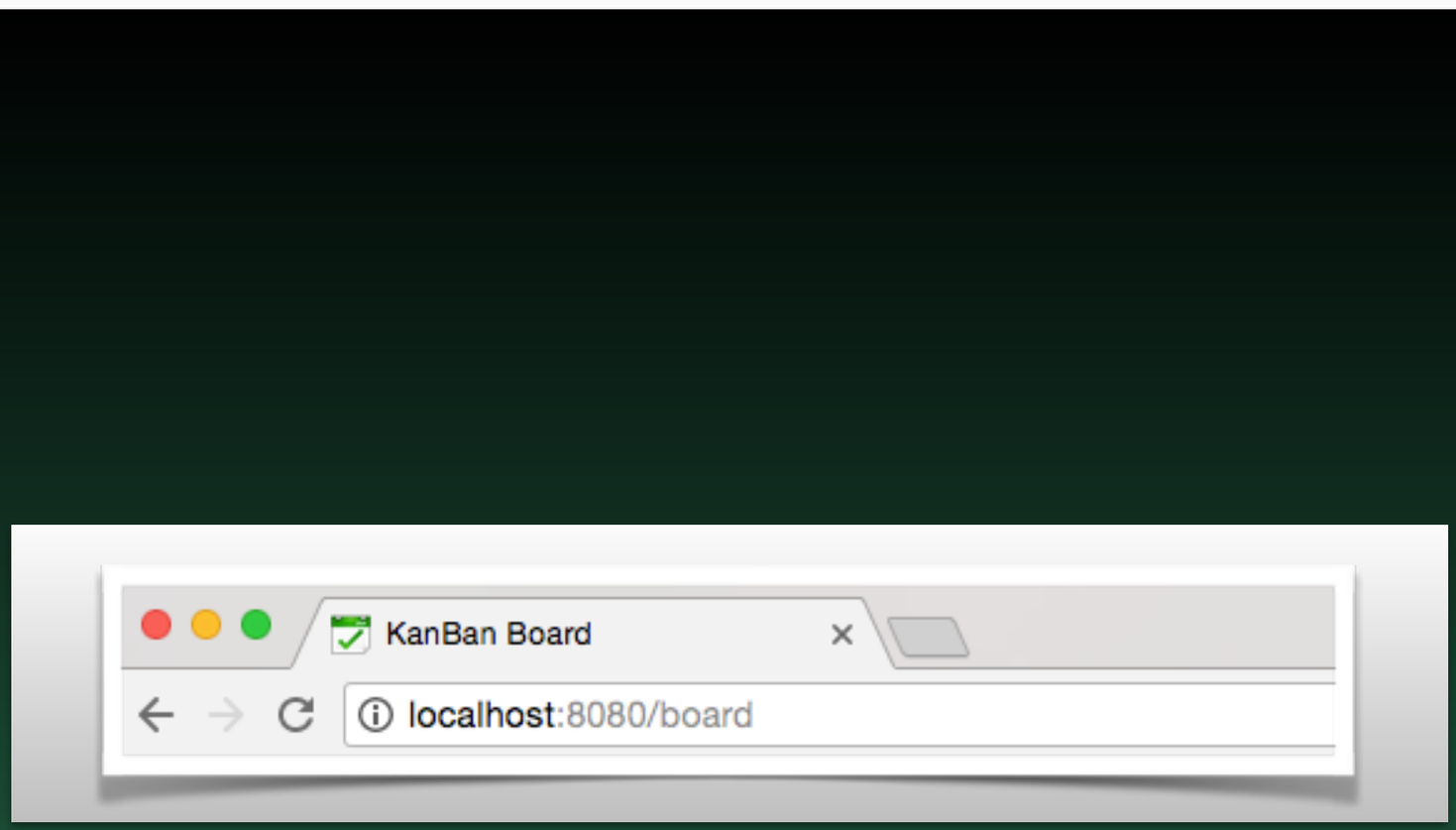
- update the method setActive() by calling the action setActiveUser() from src/store/modules/users.js in Vuex store, which passes the user object to the correspondent mutation method, that updated the global state and with that updates the user in leftNav.vue.



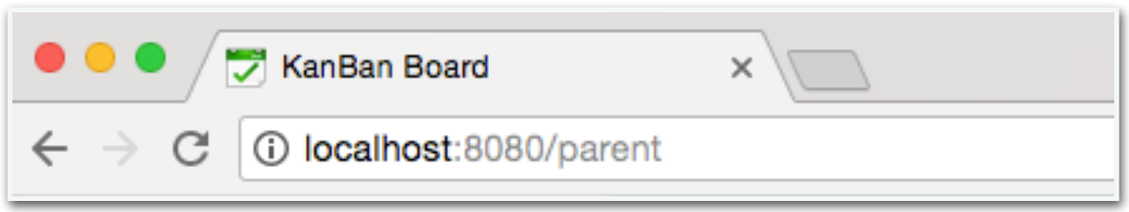

```vue
<template>
    <v-card class="pa-3">
        <child-component v-if="allChildren" v-for="(user, index) in allChildren"
                         :key="user.name" :name="user.name" :age="user.age" :place="user.place"></child-component>
    </v-card>
</template>

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

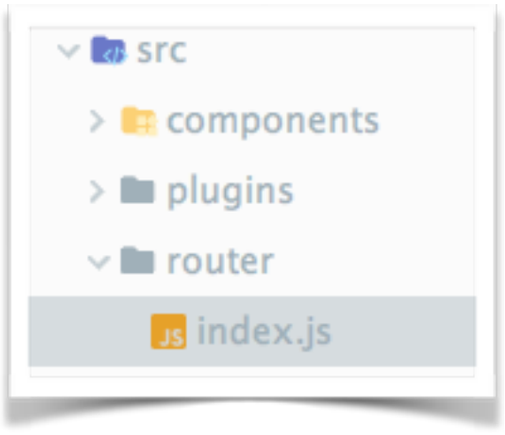


5. Routing in Vue

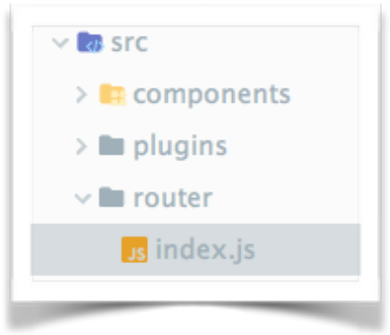


5. Routing in Vue

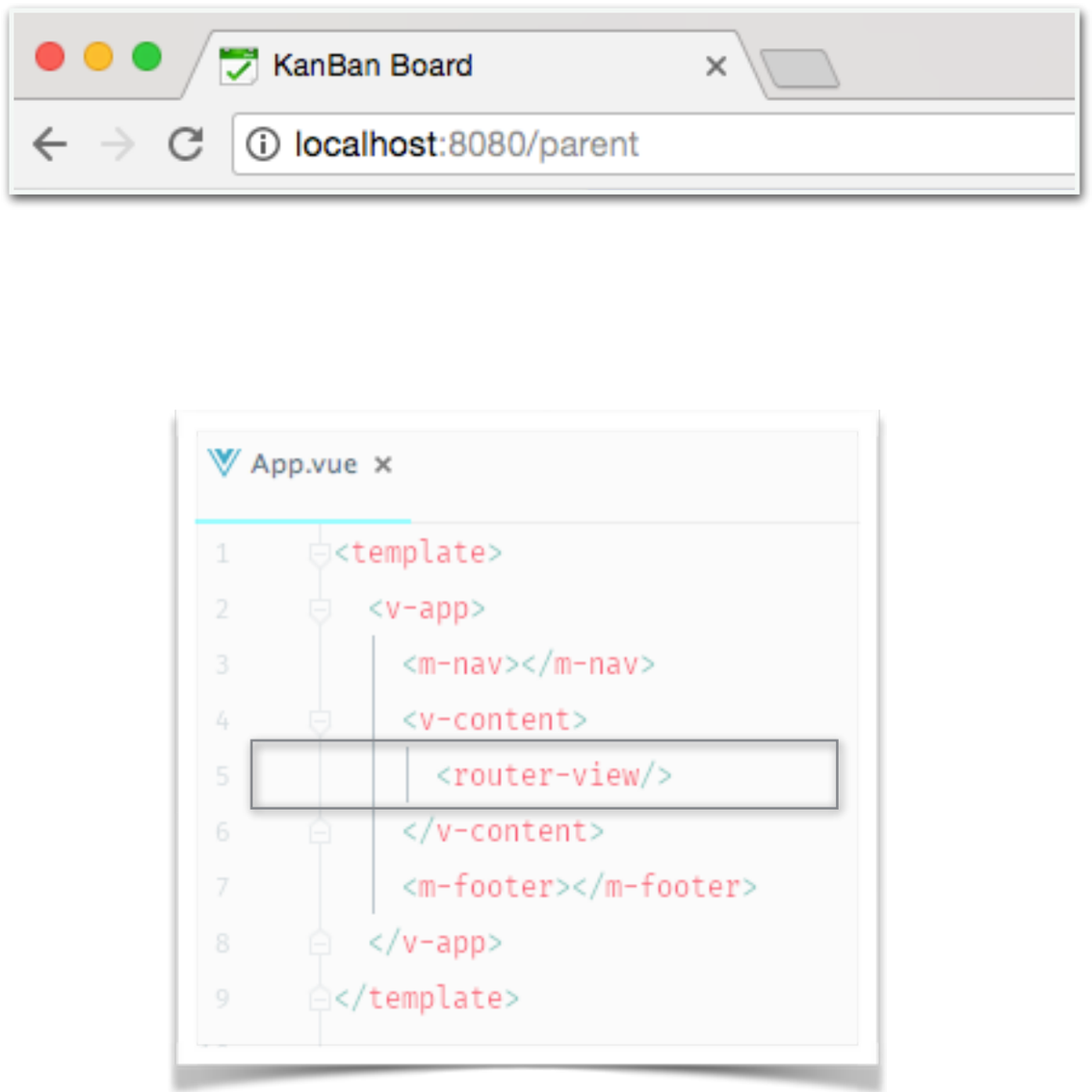




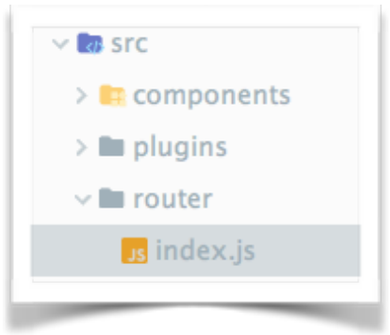

```
import Vue from 'vue'
import Router from 'vue-router'
import UserAdmin from '@/components/pages/UserAdmin'
import Parent from '@/components/pages/parent'

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



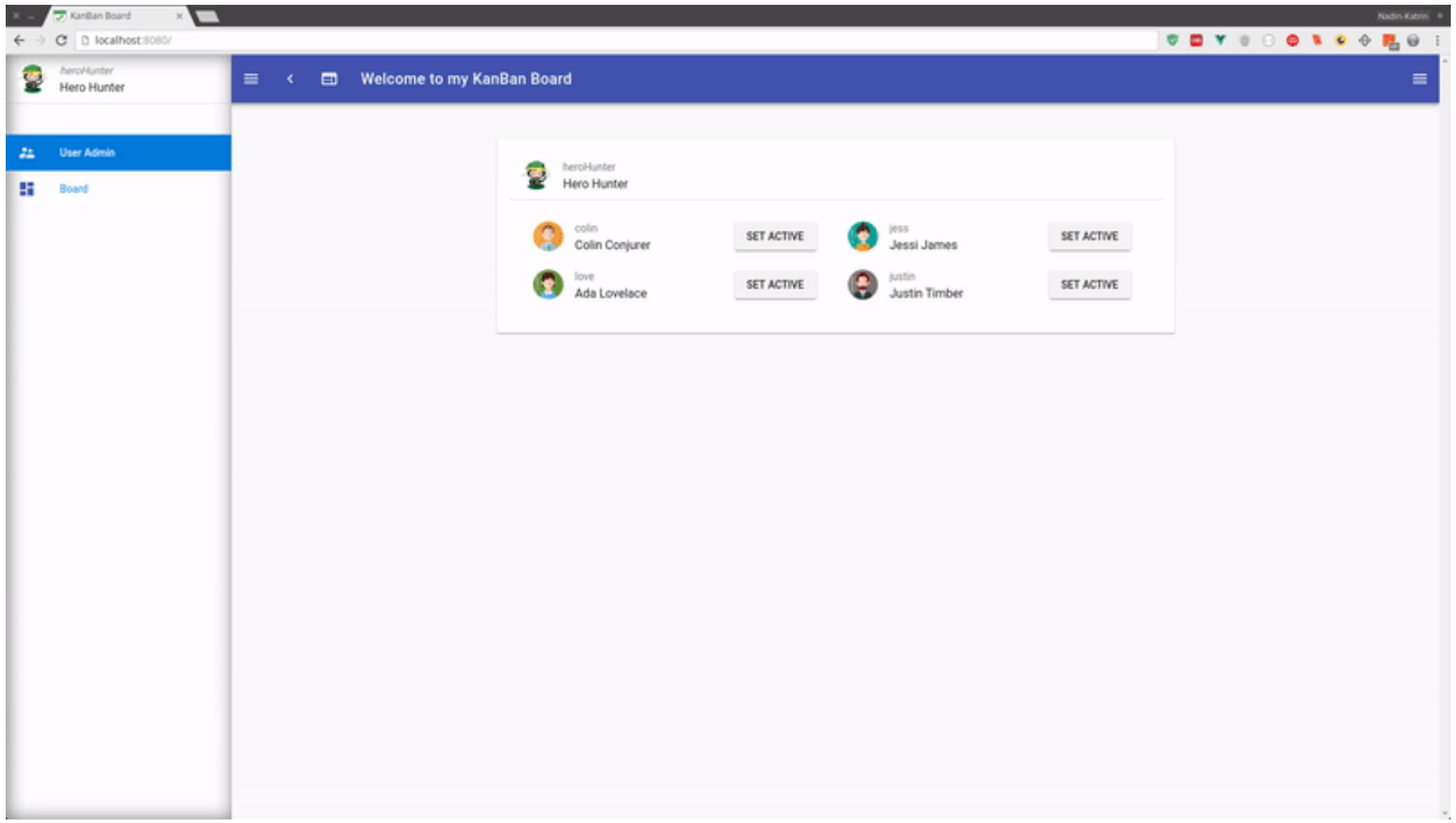

```
import Vue from 'vue'
import Router from 'vue-router'
import UserAdmin from '@/components/pages/UserAdmin'
import Parent from '@/components/pages/parent'

Vue.use(Router)

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

  ],
  mode: 'history'
})
```

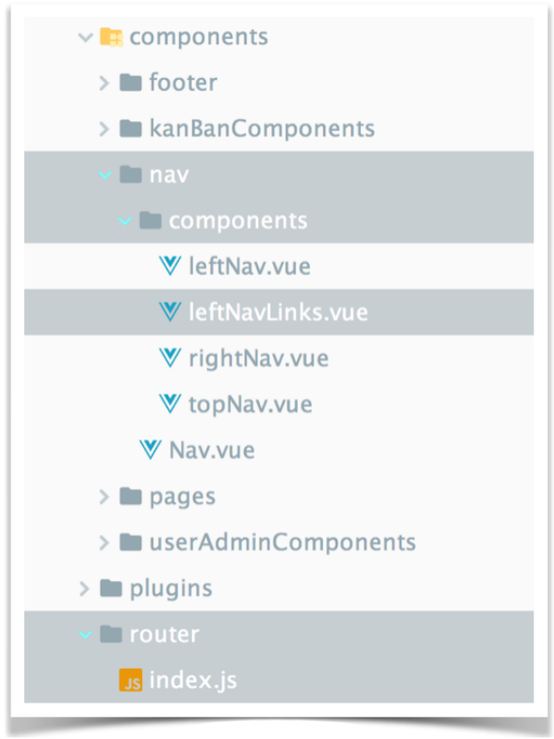


5. Routing in Vue: Exercise 5

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



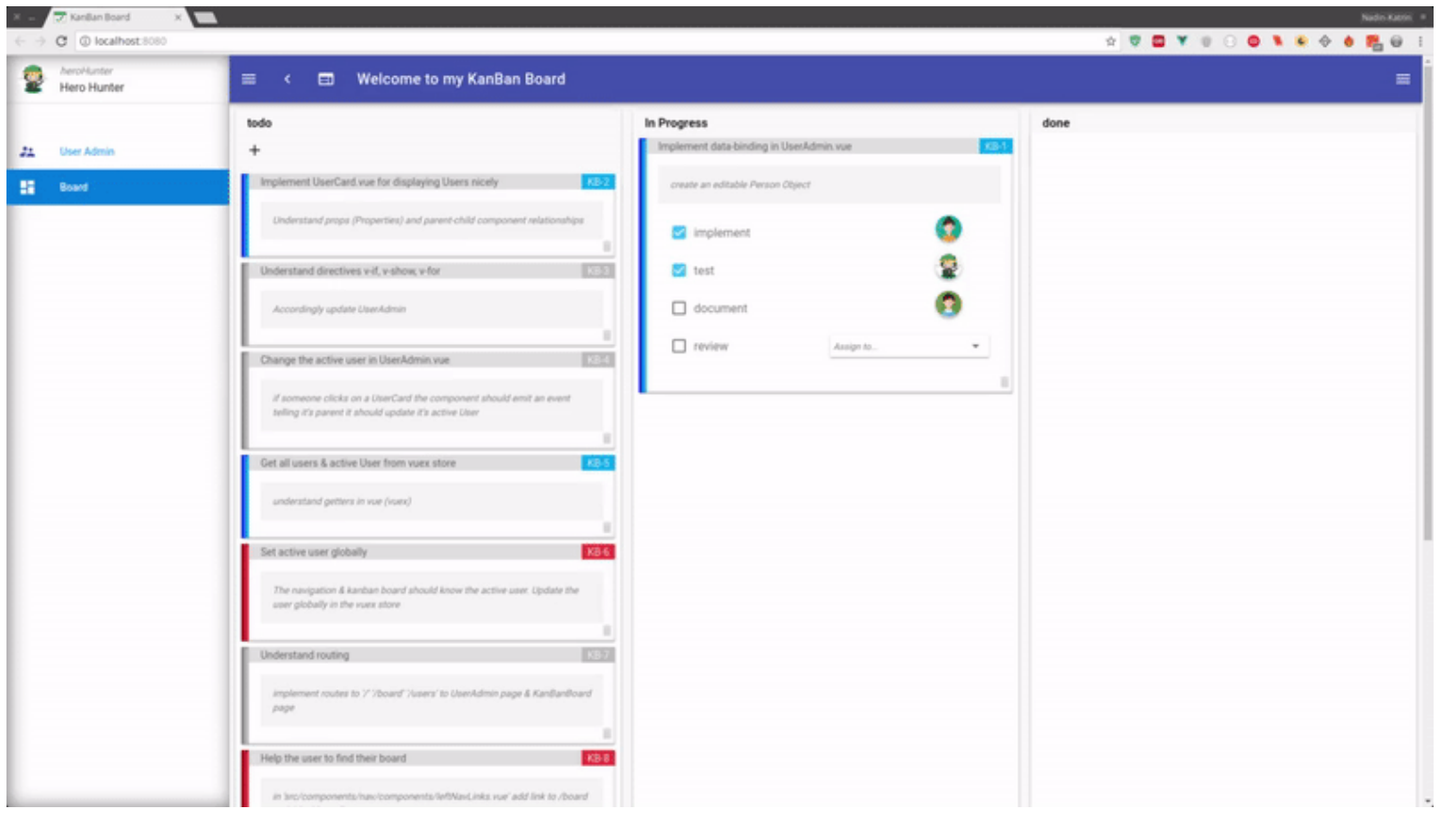


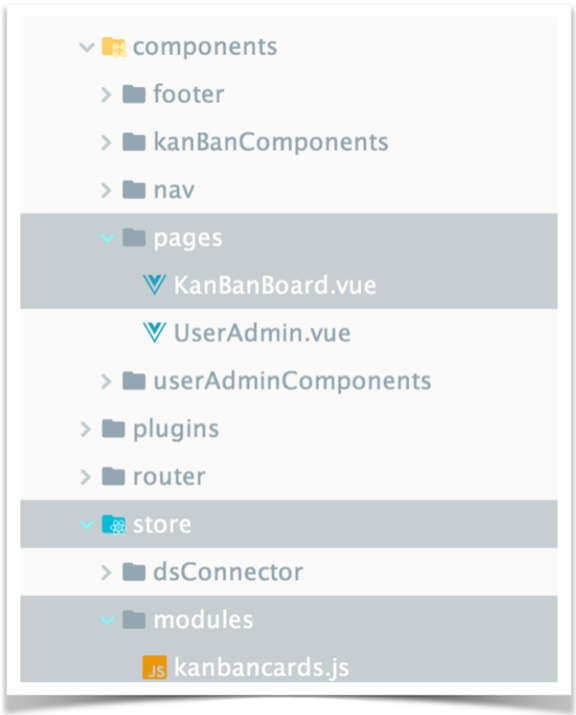
Exercise 6: Final Adjustments

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)

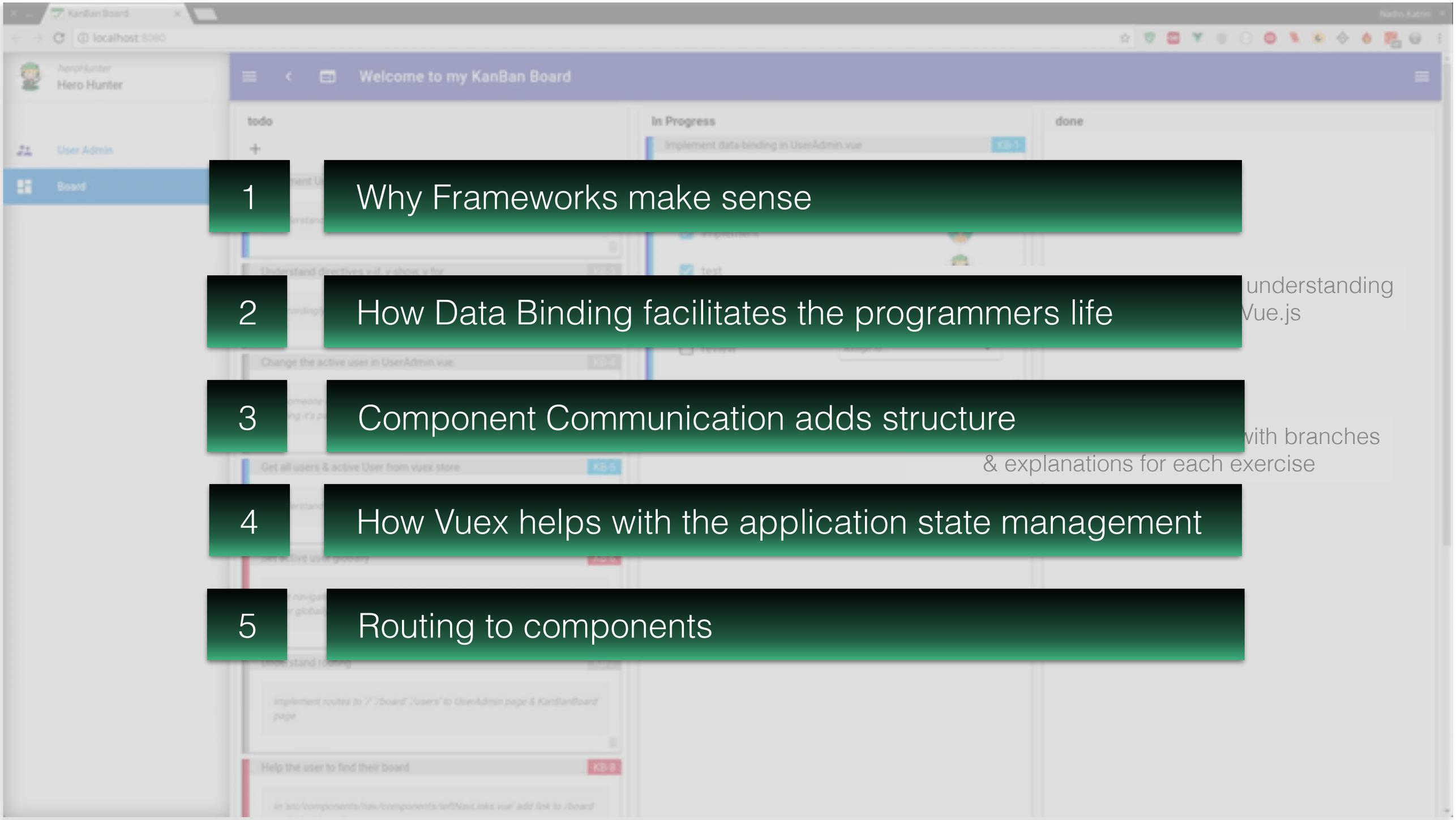
In src/components/pages/KanBanBoard.vue:
- create 'm-kan-ban-column' for each status defined in the vuex store 'src/store/modules/kanbancards.js

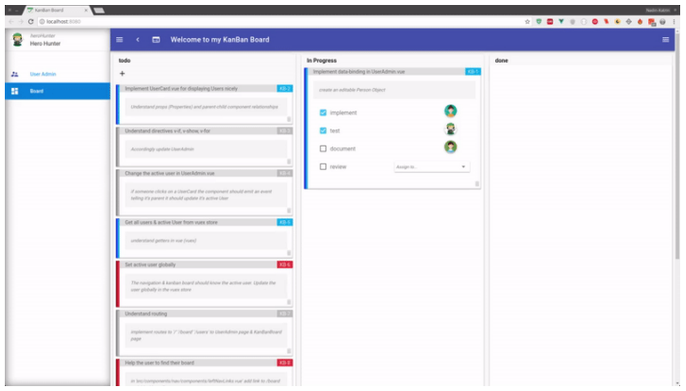




https://github.com/na018/vue_basics_tut

Wrap up: What did you learn?

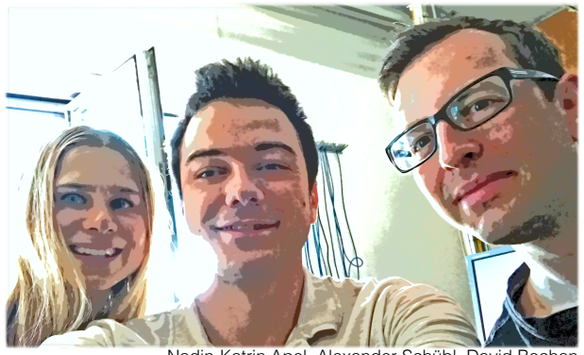




Thanks for your Attention

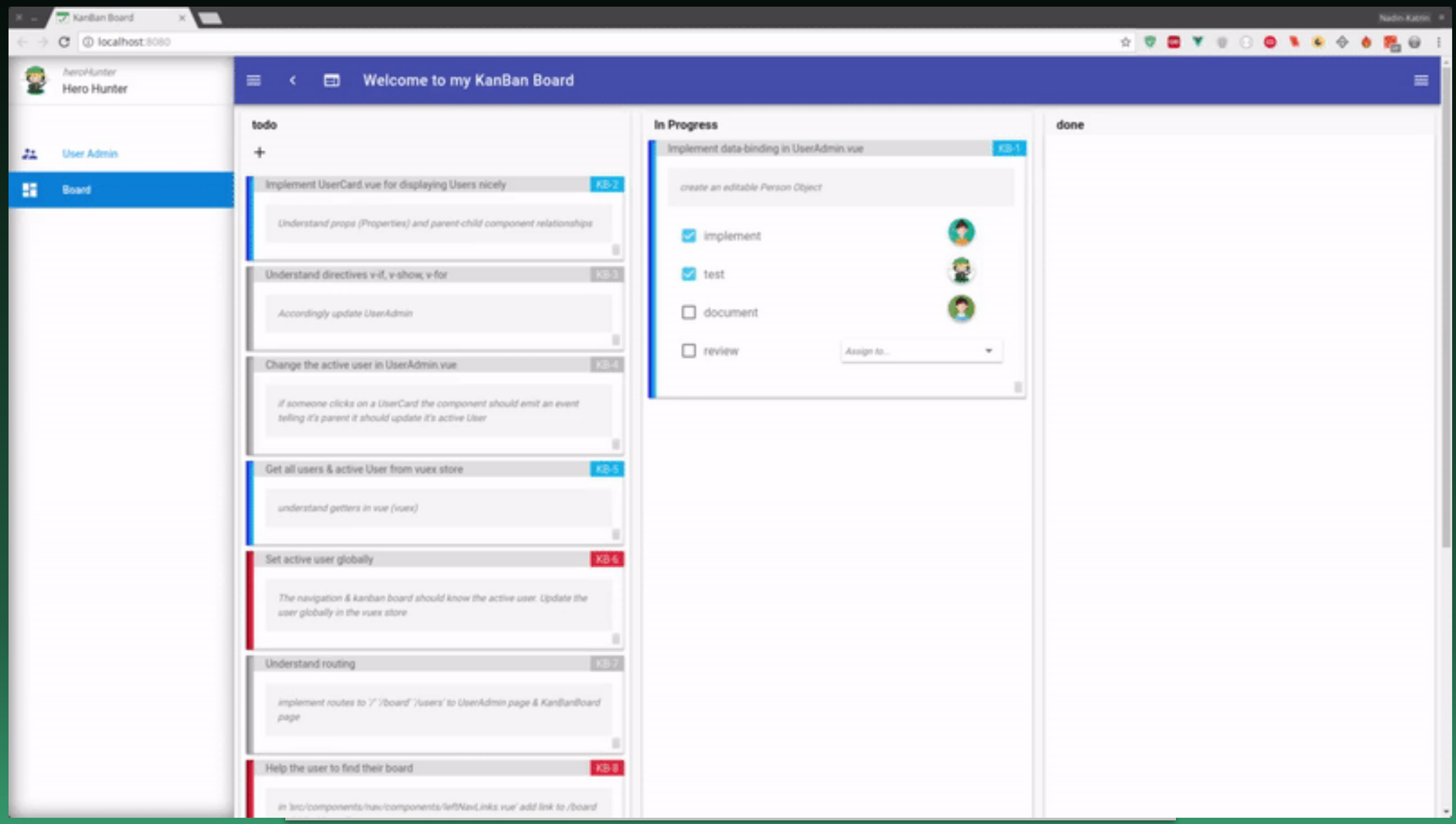
Nadin-Katrin Apel, Alexander Schübl, David Bochan

You should now have a working kanban board



Nadin-Katrin Apel, Alexander Schübl, David Bochan

Thanks for your Attention



You should now have a working kanban board