VUE MASTERCLASS

PRONOUNCED /VJUː/, LIKE VIEW

WELCOME

YOU WILL LEARN

- how to setup a Vue project and how to add Vue to existing code base
- how to split-up your project in components
- how to deal with state and reactivity
- how to write templates and connect state to templates
- how to use transitions
- how to scale up your application (SPA)

SCHEDULE

09:00 - Coffee + welcome

Introduction

09:20 - Intro to Vue

09:30 - The case (& project setup)

09:35 - Tooling

10:00 - Coffee

Components

10:25 - The Vue instance

10:30 - Creating components

10:35 - Using components

10:40 - Components naming

10:45 - Props

10:55 - Exercise

11:15 - Solution

Data

11:25 - State

11:30 - Making state reactive

11:40 - How reactivity works

11:50 - Testing reactivity

11:55 - Exercise

12:05 - Solution

12:10 - Lunch

Getting your data on screen

12:50 - What are Directives

13:00 - Most used directives

13:20 - Events

13:30 - Exercise

14:00 - Solution

14:05 - Two-way binding

14:10 - Exercise

14:25 - Solution

Computed values

14:20 - Computed props

14:30 - Exercise

14:55 - Solution

15:00 - Coffee

Slots

15:20 - What are slots

15:30 - Named slots

15:45 - Slot scope

15:55 - Exercise

16:20 - Solution

Animations

16:25 - transition

16:35 - transition-group

16:40 - Exercise

17:00 - Solution

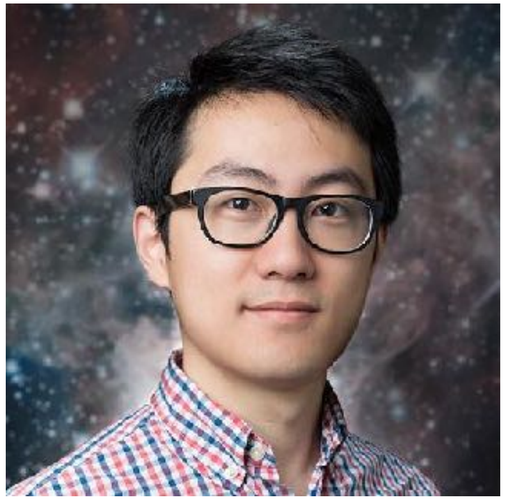
Round up

17:05 - Recap

INTRODUCTION

INTRO TO VUE

WHAT IS VUE?



Patreon: $13,453 each month

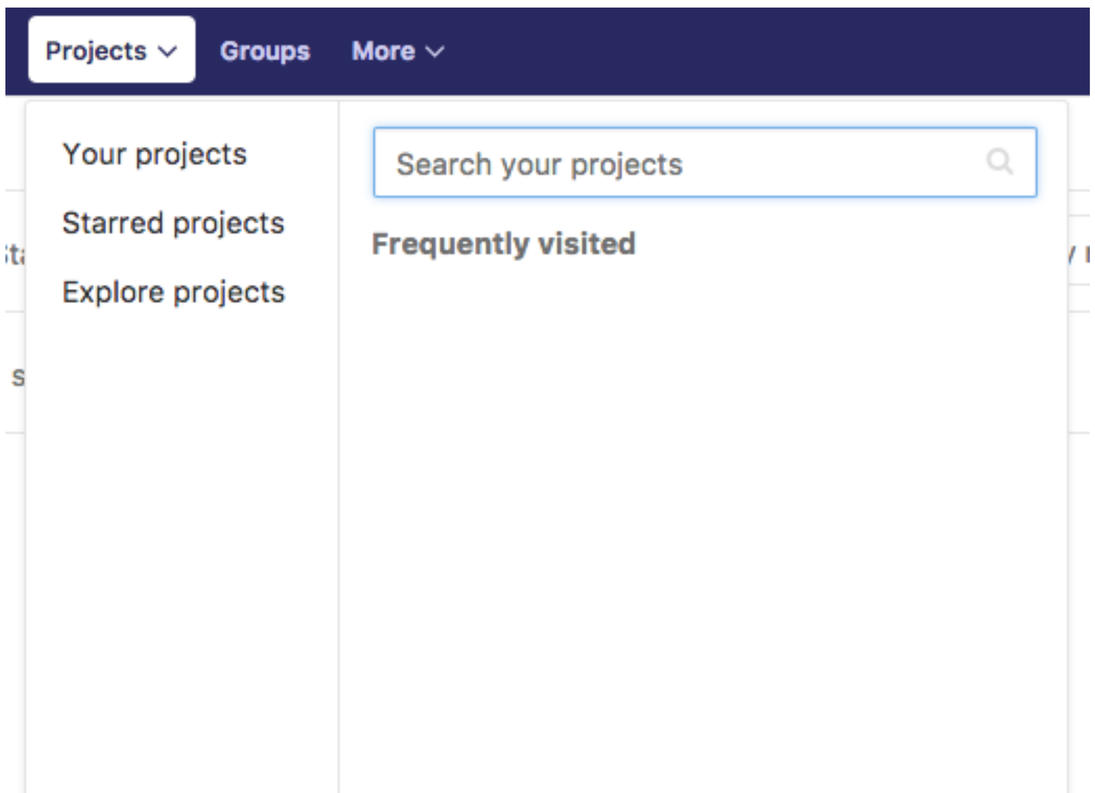
progressively adoptable UI framework

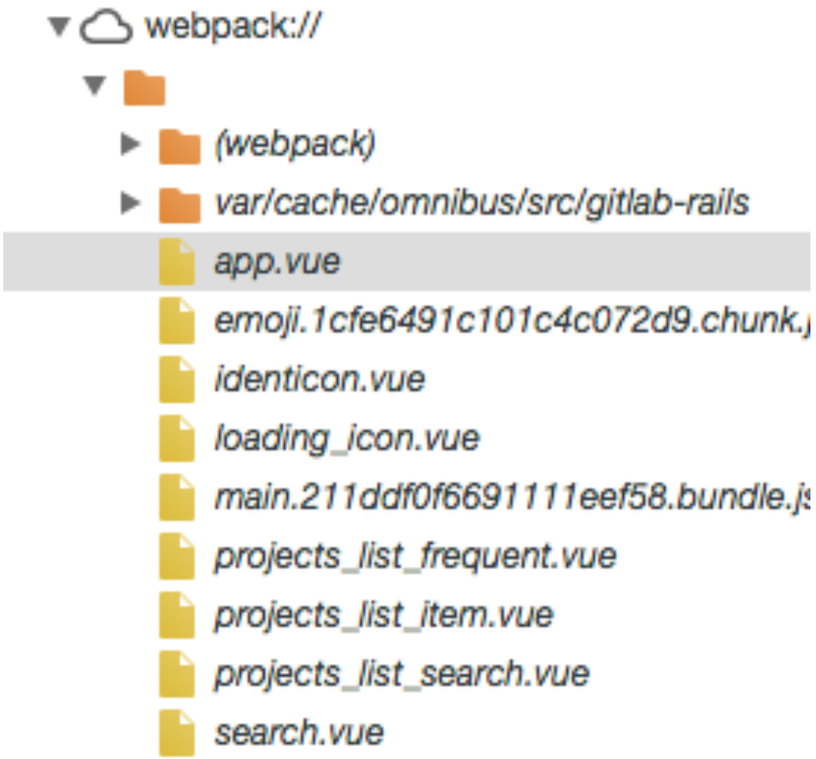
USED BY

you?

https://madewithvuejs.com

GitLab

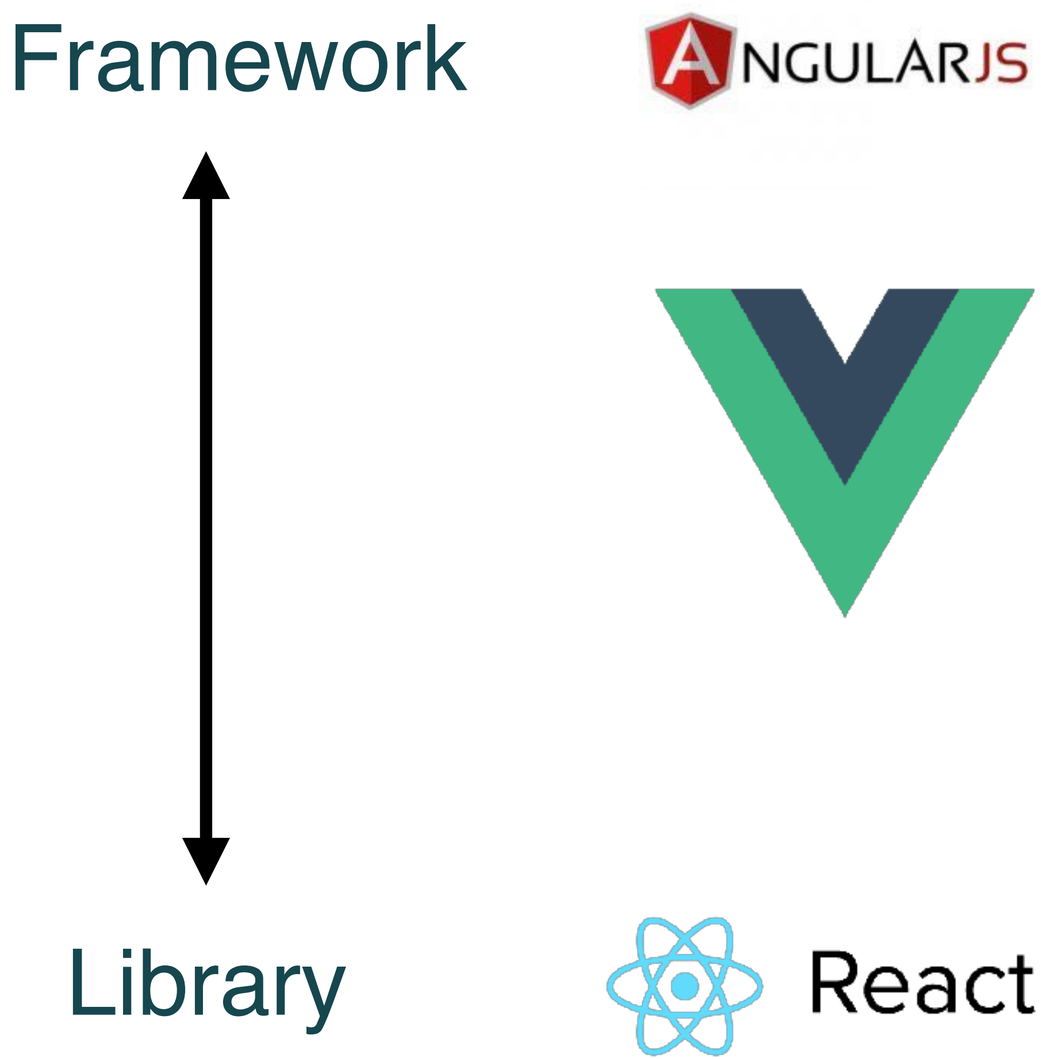

Projects
Groups
More
Your projects
Starred projects
Explore projects
Search your projects
Frequently visited


webpack://
(webpack)
var/cache/omnibus/src/gitlab-rails
app.vue
emoji.1cfe6491c101c4c072d9.chunk.
identicon.vue
loading_icon.vue
main.211ddf0f6691111eef58.bundle.
projects_list_frequent.vue
projects_list_item.vue
projects_list_search.vue
search.vue

“How we do Vue: one year later”

- don’t use jQuery in combination with Vuejs
- state management can be hard
- “Write high quality code”

COMPARISON TO OTHER FRAMEWORKS



THE CASE

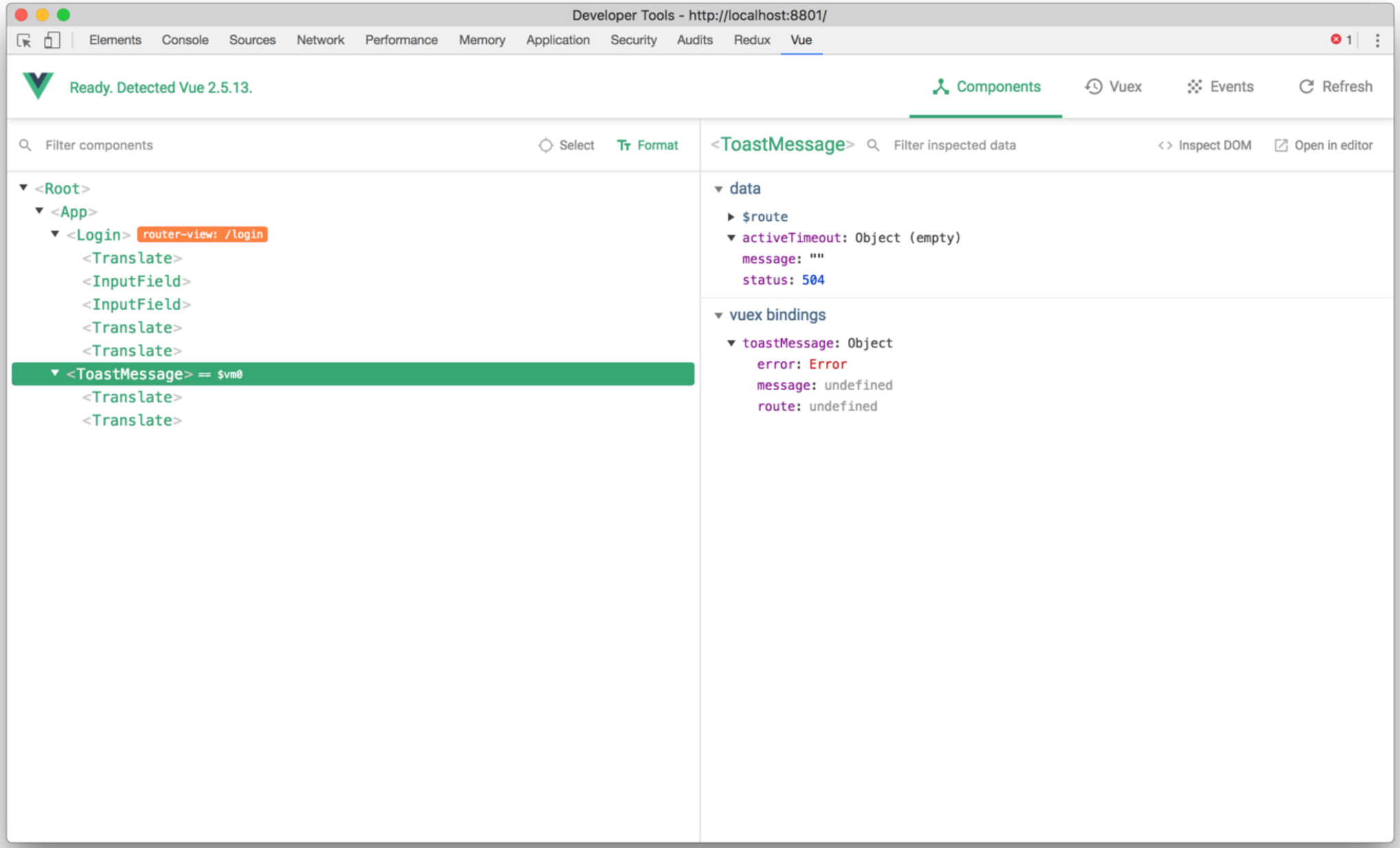
Let’s make a
simple **Slack** clone

TOOLING

```
<script src="https://cdn.jsdelivr.net/npm/vue@2.5.13/dist/vue.js"></script>
```

A browser which supports es6 modules

INSTALL THE DEVTOOLS

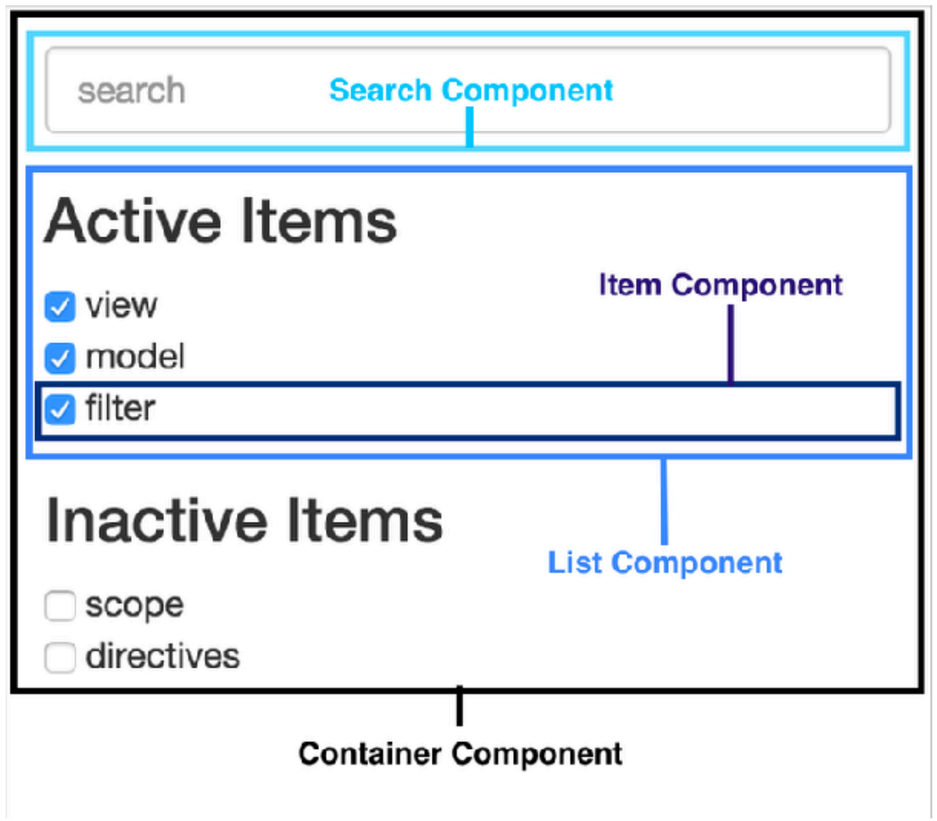


https://github.com/voorhoede/vue-masterclass

COMPONENTS

THE COMPONENT WAY OF THINKING

break apart your ui into smaller pieces



WHY COMPONENTS?

- testability
- reliability
- reusability
- extensibility

HOW USING COMPONENT-BASED DESIGN HELPS US BUILD FASTER

“With component-based design, development becomes an act of composition, rather than constantly reinventing the wheel. Using shared components as building blocks frees us up to focus on what matters, without getting bogged down in implementation details”

https://blog.twitter.com/engineering/en_us/topics/infrastructure/2019/buildingfasterwithcomponents.html

THE VUE INSTANCE

YOUR FIRST VUE APP

```
new Vue({
    el: "#app"
})
```

Mount to existing element in the DOM.

Adopts element HTML as template

```
new Vue({
    el: "#app",
    template: `<div>
        Template contents goes here
    </div>`
})
```

- Template overwrites the element
- A template always needs a **root node**!

```
new Vue({
    el: "#app",
    template: "#template"
})
```

```
<script type="x-template" id="template">
    Template content in script tag
</script>
```

All **Vue components**
are also **Vue instances**

CREATING COMPONENTS

DEFINE YOUR COMPONENT

```
const MyComponent = Vue.extend({
    template: `<div>Awesome content</div>`
})
```

returns a component **constructor**

DEFINE YOUR COMPONENT

```
const MyComponent = {
    template: `<div>Awesome content</div>`
}
```

the component options object

USING COMPONENTS

LOCAL REGISTRATION

```
const Page = {
    template: `<my-component></my-component>`,
    components: {
        MyComponent
    }
}
```

GLOBAL REGISTRATION

```
Vue.component('my-component', {
    template: `<div>Awesome content</div>`
})
```

COMPONENT NAMING

CUSTOM ELEMENT NAMING

- names separated by dash
- always use a **namespace** to prevent conflicts with existing elements.
  So **app-header** instead of header

- Vuejs does not really care…
- MyComponent <-> my-component

check if your component name is valid on:
https://mothereff.in/custom-element-name

PROPS

PROPS

```
function todo(isCompleted, text) {
    return {
        isCompleted,
        text
    }
}
```

- properties make a component customisable / reusable
- props can not be mutated inside the component

PROPS IN VUE

```
Vue.component('my-component', {
    props : [
        'isCompleted',
        'text'
    ]
})
```

Problem: no validation.

“text” can be anything!

VALIDATION

```
Vue.component('my-component', {
    props : {
        isCompleted : {type : Boolean, default : false},
        text : {type : String, required : true},
        user : {type : Object, validator : obj => "name" in obj}
    }
})
```

type can be

Boolean, String, Number, Object, Array, Function, Date

🏋️ EXERCISE TIME! 🏋️

Split the app into components.

branch: exercise1

🏋️ EXERCISE TIME! 🏋️

solution

branch: exercise1-solution

DATA

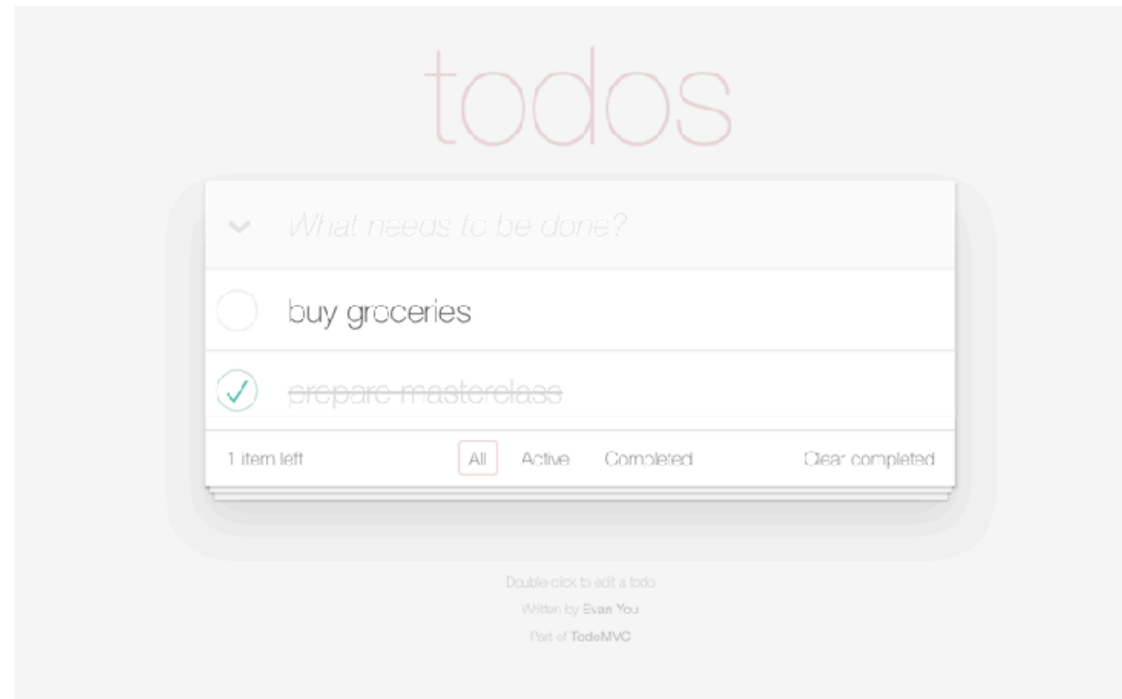
STATE

STATE IN VUE

```
{
    todos : [
        {
            text : 'Buy groceries',
            done : false
        },
        {
            text : 'Prepare masterclass',
            done : true
        }
    ]
}
```

WHAT YOU WANT

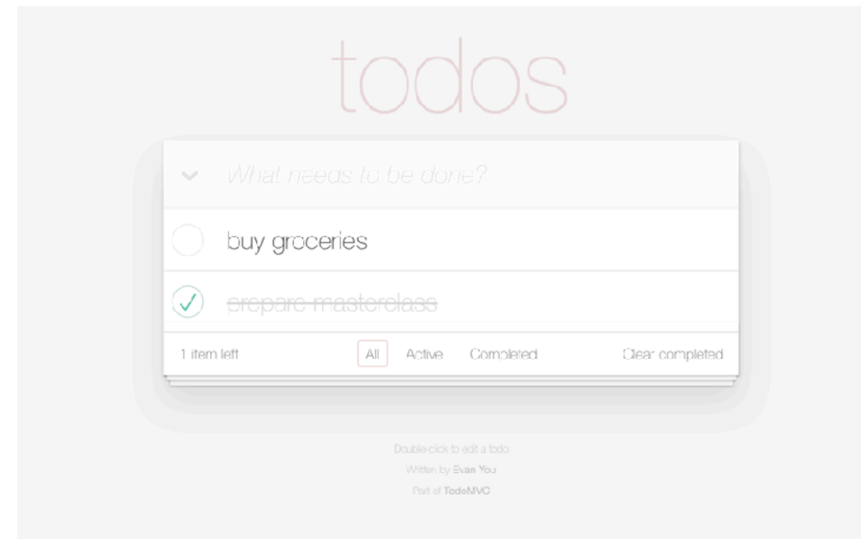
```
{
    todos : [
        {
            text : 'Buy groceries',
            done : false
        },
        {
            text : 'Prepare masterclass',
            done : true
        }
    ]
}
```



WHAT YOU NEED

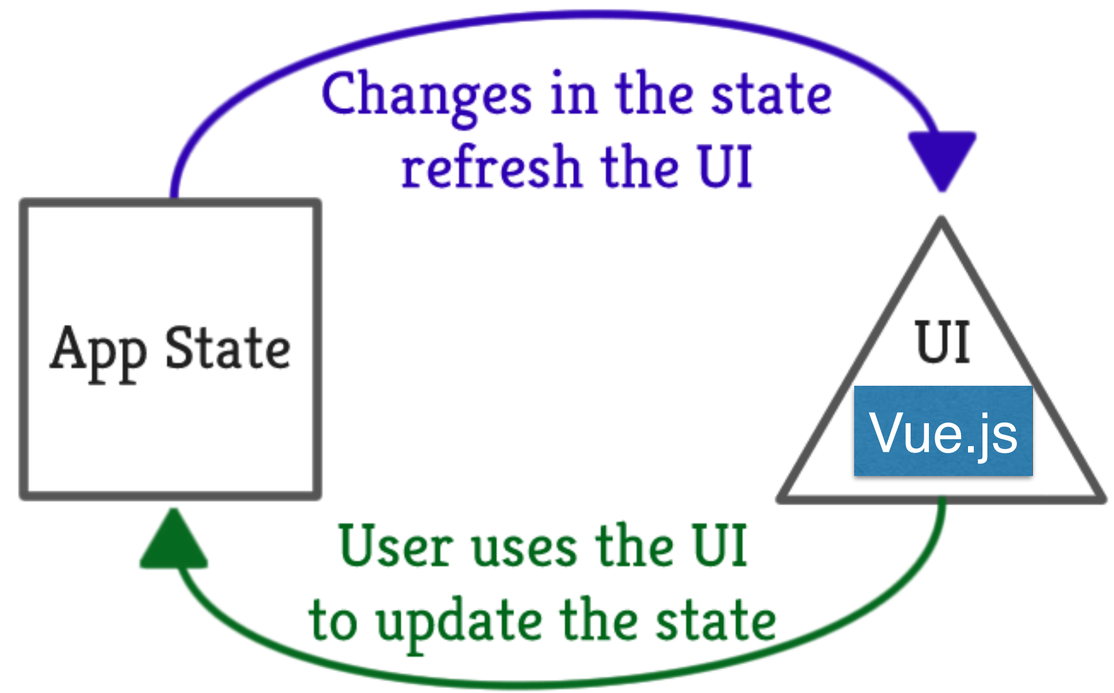
```
{
    todos : [
        {
            text : 'Buy groceries',
            done : false
        },
        {
            text : 'Prepare masterclass',
            done : true
        }
    ]
}
```

state changes
to ui changes

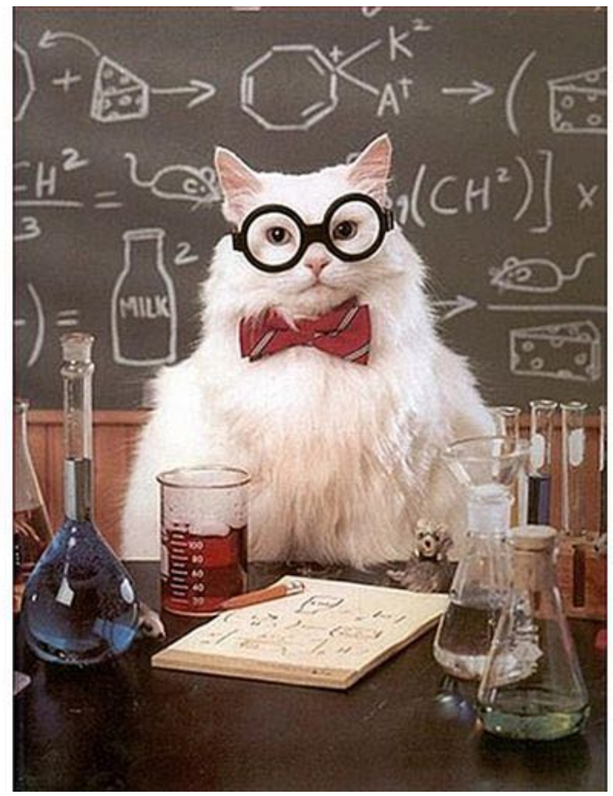


REMOVING THE MIDDLE-MAN

- less code to maintain
- better performance



REACTIVITY



the art of translating state changes
to ui changes

MAKING STATE REACTIVE

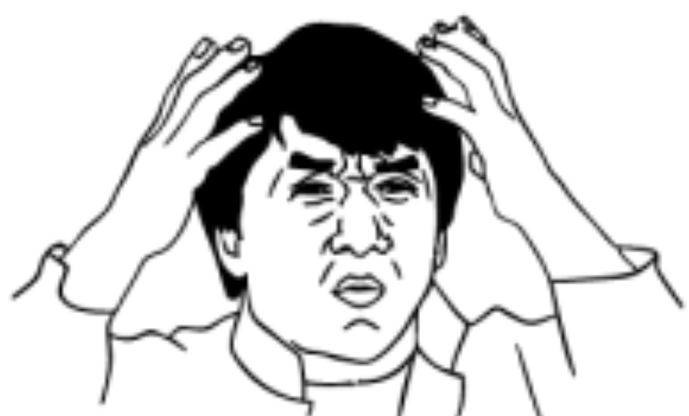
```
const appData = {
    firstName: "Remco"
}
```

MAKING STATE “REACTIVE”

appData →

```
new Vue({
  data: appData
})
```

→ reactive appData



CHANGING REACTIVE STATE

```
const app = new Vue({
    data: appData
})

app.firstName = "Jaap"

//appData.firstName == app.firstName
```

WRONG BEHAVIOUR

```
Vue.component('click-once-button', {
    data : {
        isClicked : false
    }
})
```

isClicked data is shared for all components!

USE THE FUNCTION

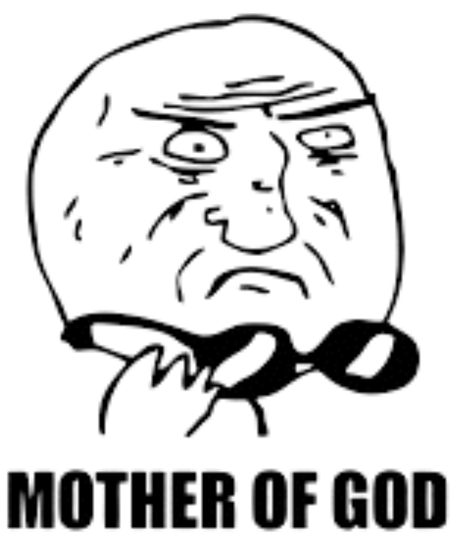
```
Vue.component('click-once-button', {
    data : function () {
        return {
            isClicked : false
        }
    }
})
```

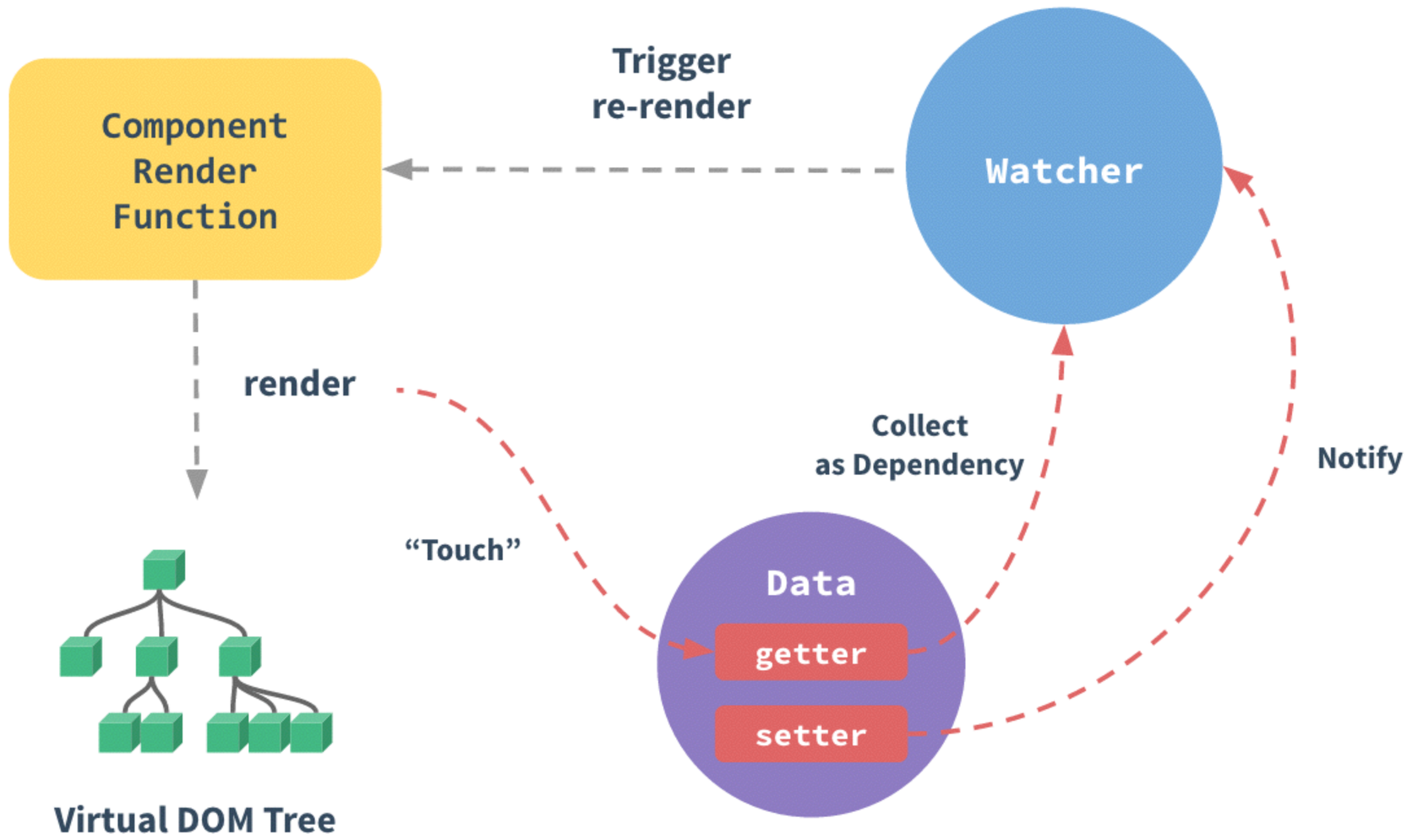
isClicked data is now unique for all components!

HOW REACTIVITY WORKS

```
console.log(someState)
```

```
▼ {__ob__: Observer} ℹ                                    (index):59
    firstName: (...)
  ▶ __ob__: Observer {value: {…}, dep: Dep, vms: Array(1)}
  ▶ get firstName: ƒ reactiveGetter()
  ▶ set firstName: ƒ reactiveSetter(newVal)
  ▶ __proto__: Object
> |
```




Component
Render
Function
Trigger
re-render
Watcher
render
Collect
as Dependency
Notify
"Touch"
Data
getter
setter
Virtual DOM Tree

TESTING REACTIVITY
(WITHOUT TEMPLATES)

```
const app = new Vue({
    data() {
        return {
            firstName: 'Remco'
        }
    },
    watch : {
        firstName(newValue, oldValue) {
            console.log("😎")
        }
    }
});

app.firstName = 'Jaap';
```

🏋️ EXERCISE TIME! 🏋️

- create the data structure for the chat app
  (containing at least a username and message array)
- mark the data as reactive
- use the devtools to change the reactive data

- try to add a new property to the reactive data
- try to change the individual items of an message array

branch: exercise2

🏋️ EXERCISE TIME! 🏋️

solution

branch: exercise2-solution

LIMITATIONS OF TRACKING

- Vue can’t detect the **replacement** of array items
- Vue can’t detect the **addition** or **removal** of properties

Solutions

- Use Vue.set or this.$set or array splice to replace array elements
- Don’t dynamically add or remove properties
- Make your data **immutable**
- Wait for **Vue-next**

GETTING YOUR STATE ON SCREEN

TEMPLATE SYNTAX

```
<div class="image-frame" :class="{'image-frame--no-animation' : !animation}">
    <img
        v-if="load"
        v-show="hasPlaceholder"
        :src="placeholderURL"
        class="image-frame__img image-frame__img--placeholder"
        @load="onPlaceholderLoad">

    <picture v-if="load">
        <source v-for="source in sources"
sizes="100vw" :type="source.type" :srcset="source.srcSet"></source>

        <img
            v-show="loaded"
            ref="img"
            :src="fallbackSrc"
            class="image-frame__img"
            :class="{'image-frame__img--loaded' : loaded}"
            @load="onImgLoad">
    </picture>
</div>
```

DIRECTIVES

optional modifiers

name of directive

```
<input type="number" v-bind:value.number="quantity" />
```

always start with

v-

optional argument

value

RULES OF DIRECTIVES

- a directive attaches **behaviour** to an component/ element
- all directives start with **v-**
- the content of a directive should be a **valid js expression**
- You can only reference reactive data (props and local state) available in the **component scope** (and methods)

VALID JS EXPRESSIONS

```
<div v-directive="someProp ? textProp : otherProp"></div>

<div v-directive="someProp || otherProp === true"></div>

<div v-directive="[1,2,3,4]"></div>

<div v-directive="{key : 'value'}"></div>

<div v-directive="someFunction()"></div>
```

What goes wrong here?

```
<div v-some-directive="some text"></div>
```

Some text is not a valid js expression!

```
<div v-some-directive="'some text'"></div>
```

wrap in quotes

V-TEXT

```
<div v-text="propWithText"></div>
```

Or

```
<div>{{ propWithText }}</div>
```

HTML is escaped! So no XSS attacks

V-BIND

bind reactive data to element props / attributes

```
<img v-bind:src="myProp">
```

accepts the property name as the argument

object syntax

```
<img v-bind="{src : 'cat.jpg', title : 'miauw'}">
```

BIND SHORTHAND

```
<img :src="myDynamicSource">
```

MANIPULATING CLASSES

string concatenation does not scale

```
<a
    v-bind:class="'link ' + (homePageActive ? 'link--active' : '') ">
    Link to homepage
</a>
```

SMART CLASS AND STYLES BINDINGS

- merge with existing class and style properties
- accepts array, objects (and combinations)

```
<a
    class="link"
    v-bind:class="[
        'link-special',
        {'link-active' : homePageActive}
    ]">
    Link to homepage
</a>
```

V-IF, V-ELSE-IF, V-ELSE

CONDITIONALLY SHOW ELEMENTS

```
<div v-if="state === 'loading'">
    Loading...
</div>

<div v-else-if="state === 'error'">
    Error!
</div>

<div v-else>
    Done!
</div>
```

CONDITIONALLY SHOW ELEMENTS

```
<template v-if="state === 'done'">
    <div>Component one</div>
    <div>Component two</div>
</template>
```

V-SHOW

```
<div v-show="empty">
    There is nothing to show
</div>
```

```
<div style="display:none">
    There is nothing to show
</div>
```

V-FOR

```
<ol>
    <li v-for="item of items"></li>
</ol>
```

you can loop over arrays and objects

```
<ol>
    <li v-for="(item, index) of items"></li>
</ol>
```

```
<ol>
    <li v-for="item of items" :key="item.id"></li>
</ol>
```

The default is to **track by index.**

Use keys so Vue can efficiently **reuse elements**

🏋️ EXERCISE TIME! 🏋️

- Show “there are no messages” when the message array is empty.
- Render a message for each item in the message array.

branch: exercise3

🏋️ EXERCISE TIME! 🏋️

solution

branch: exercise3-solution

V-ON

Handler

```
<button v-on:click="onSubmitClicked">Submit</button>
```

Event name

HANDLER NOTATIONS

```
<button v-on:click="submitted = true">Submit</button>

<button v-on:click="onSubmitClicked">Submit</button>

<button v-on:click="onSubmitClicked($event)">Submit</
button>
```

METHOD EVENT HANDLERS

```
new Vue({
    template : `
        <button v-on:click="onSubmitClicked">
        Submit
        </button>
    `,
    methods : {
        onSubmitClicked(event) {
            console.log('You clicked submit');
        }
    }
})
```

CUSTOM EVENTS

```
Vue.component('search-input', {
    template : `
        <input type="search" v-on:input="onTextInput">
    `,
    methods : {
        onTextInput(event) {
            this.$emit('input', event.target.value);
        }
    }
})
```

V-ON SHORTHAND

```
<button @click="onSubmitClicked">Submit</button>
```

🏋️ EXERCISE TIME! 🏋️

Listen to the input event of the input field
and write the entered chat text into local state

branch: exercise4

🏋️ EXERCISE TIME! 🏋️

solution

branch: exercise4-solution

V-MODEL
(2-WAY BINDING)

DEALING WITH FORM DATA

Typical scenario: saving form inputs into state to do validation

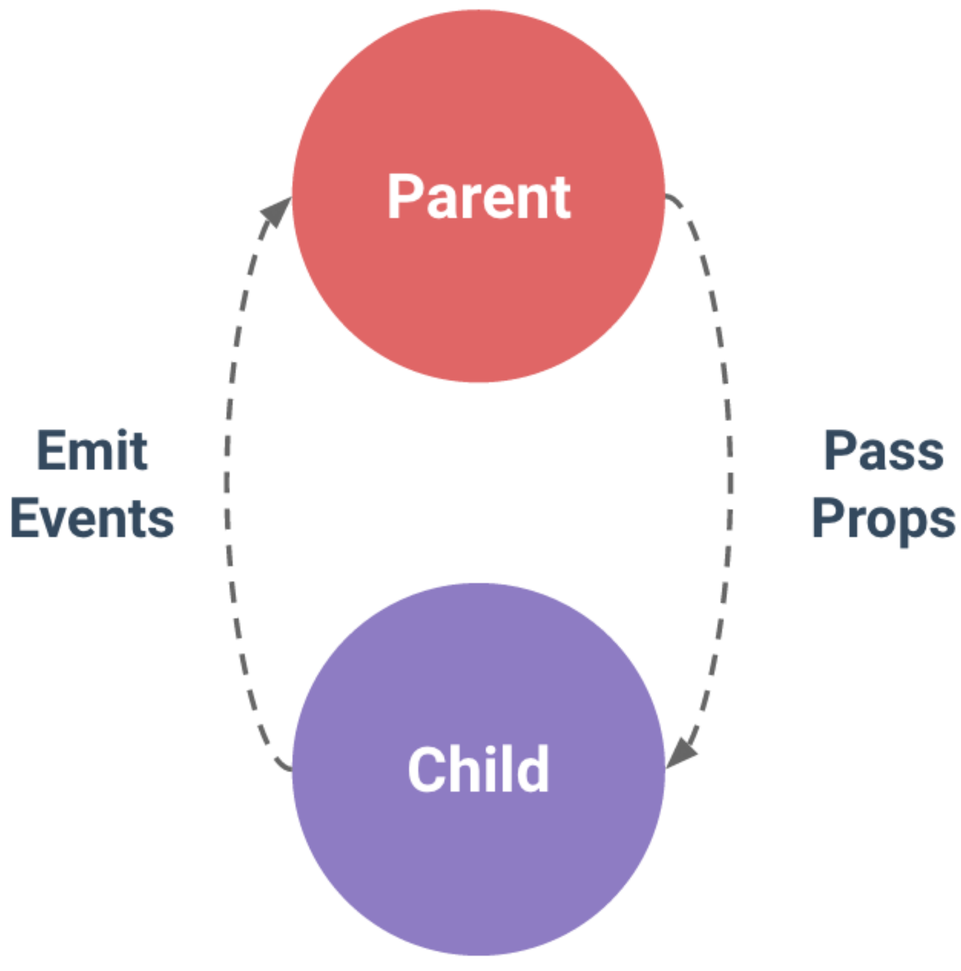
```
new Vue({
    template : `
        <input type="text"
         v-bind:value="text"
         v-on:input="text = $event.target.value">
    `,
    data() {
        return {
            text : ''
        }
    }
})
```

this is called 2-way binding

- listens to the input event
- writes value of event into the given binding
- assigns the value of the binding to the value prop

```
new Vue({
    template : `
        <input type="text" v-model="text">
    `,
    data() {
        return {
            text : ''
        }
    }
})
```

PROPS DOWN, EVENTS UP



🏋️ EXERCISE TIME! 🏋️

Change the code from the previous exercise to use v-model.

Allow the user to add a message.

branch: exercise5

🏋️ EXERCISE TIME! 🏋️

solution

branch: exercise5-solution

COMPUTED PROPS

PROBLEM

too much logic inside your templates

=

bad for maintainability

```
new Vue({
    template : `
        <div v-if="issues.some(issue => !issue.done)">
            {{ issues.filter(issue => !issue.done).length }}
            issues left
        </div>
    `
})
```

ALTERNATIVE 1

```
new Vue({
    template : `
        <div v-if="hasTodos">
            {{ todoCount }}
            issues left
        </div>
    `,
    data() {
        return {
            hasTodos  : false,
            todoCount : 0
        }
    },
    watch : {
        issues(newValue) {
            this.todoCount = newValue.filter(todo => !todo.done).length;
            this.hasTodos = this.todoCount > 0;
        }
    }
})
```

Using watchers to (re)compute data

ALTERNATIVE 2

```
new Vue({
    template : `
        <div v-if="hasTodos()">
            {{ getTodoCount() }}
            issues left
        </div>
    `,
    methods : {
        getTodoCount() {
            return this.issues.filter(issue => !issue.done).length;
        },
        hasTodos() {
            return this.getTodoCount() > 0;
        }
    }
})
```

Recomputes on demand. getTodoCount is called twice!

USING COMPUTED PROPS

```
new Vue({
    template : `
        <div v-if="hasTodos">
            {{ todoCount }}
            issues left
        </div>
    `,
    computed : {
        todoCount() {
            return this.issues.filter(issue => !issue.done).length;
        },
        hasTodos() {
            return this.todoCount > 0;
        }
    }
})
```

todoCount is **cached based on its dependencies**

🏋️ EXERCISE TIME! 🏋️

- Allow the user to search for messages
- Show the amount of search results

branch: exercise6

🏋️ EXERCISE TIME! 🏋️

solution

branch: exercise6-solution

SLOTS

```
<div class="sidebar">
    <header v-if="page === 'mail'">
        <h1>Mail</h1>
        <input type="search">
    </header>

    <header v-else-if="page === 'contacts'">
        <h1>Contacts</h1>
        <button>Add</button>
    </header>

    <header v-else-if="page === 'settings'">
        <h1>Settings</h1>
    </header>
</div>
```

COMPOSING ELEMENTS IN HTML

```
<fieldset>
    <legend>Title</legend>
    <input type="radio" id="radio">
    <label for="radio">Click me</label>
</fieldset>
```

also known as **transclusion** or **React.Children**

COMPOSING ELEMENTS IN VUE

```
<app>
    <app-header></app-header>
    <app-footer></app-footer>
</app>
```

→

```
Vue.component('app', {
    template : `<div class='app'>
        <slot>Fallback</slot>
    </div>`
})
```

- Vue will ignore child content unless you provide a slot
- Anything within the slot tag will be considered as **fallback content**

NAMED SLOTS

```
<app>
    <app-header v-slot:header></app-header>
    <app-footer v-slot:footer></app-footer>
</app>
```

```
Vue.component('app', {
    template : `<div class='app'>
        <div class='app-header'>
            <slot name='header' />
        </div>
        <div class='app-footer'>
            <slot name='footer' />
        </div>
    </div>`
})
```

you can repeat a slot with the same name

CONDITIONAL SLOTS AND LOOPS

```
<div class="list">
    <slot v-if="empty" name="emptyMessage"></slot>

    <slot v-for="item of items" name="item"></slot>
</div>
```

SLOT SCOPE

List.js

```
<ol class="list">
    <li v-for="item of items">
        <slot name="item" />
    </li>
</ol>
```

App.js

```
<list v-bind:items="items">
    <template v-slot:item>{{ item.name }}</template>
</list>
```

what goes wrong here?

SLOT SCOPE EXAMPLES

List.js

```
<ol class="list">
    <li v-for="item of items">
        <slot name="item" :item="item" />
    </li>
</ol>
```

App.js

```
<list v-bind:items="items">
    <template v-slot:item="{ item }">
        {{ item.name }}
    </template>
</list>
```

SLOT SCOPE EXAMPLES

Quiz.js

```
<div class="quiz">
    <slot name="question" :title="title" :text="text"></slot>
    <button>Next question</button>
</div>
```

App.js

```
<quiz url="http://quizdata.com/quiz.json">
    <template v-slot:question="{ title, text }">
        <div>
            <h2>{{ title }}</h2>
            <p>{{ text }}</p>
        </div>
    </template>
</quiz>
```

🏋️ EXERCISE TIME! 🏋️

I have created 2 types of messages.
Submitting "/cat" as the message text will add the cat-message instead of the text-message.

Finish the implementation of the cat-message and text-message using slots

branch: exercise7

🏋️ EXERCISE TIME! 🏋️

solution

branch: exercise7-solution

ANIMATIONS

GOALS OF MOTION DESIGN

- Guided focus between views
- Hierarchical and spatial relationships between elements
- Distraction from what’s happening behind the scenes
- Character, polish and delight

https://material.io/guidelines/motion/material-motion.html

I WANT TO ANIMATE...

- an element when it appears or disappears
- the transition from one element to another element
- the removal, addition and movement of items in a list

css alone will not help you with this

TRANSITION

TRANSITION

animate **one** component

```
<transition name="fade">
    <div v-if="someProp">I animate!</div>
</transition>
```

```
<transition name="fade">
    <div key="page-one" v-if="someProp">Page one</div>
    <div key="page-two" v-else>Page two</div>
</transition>
```

TRANSITION MODE

in-out

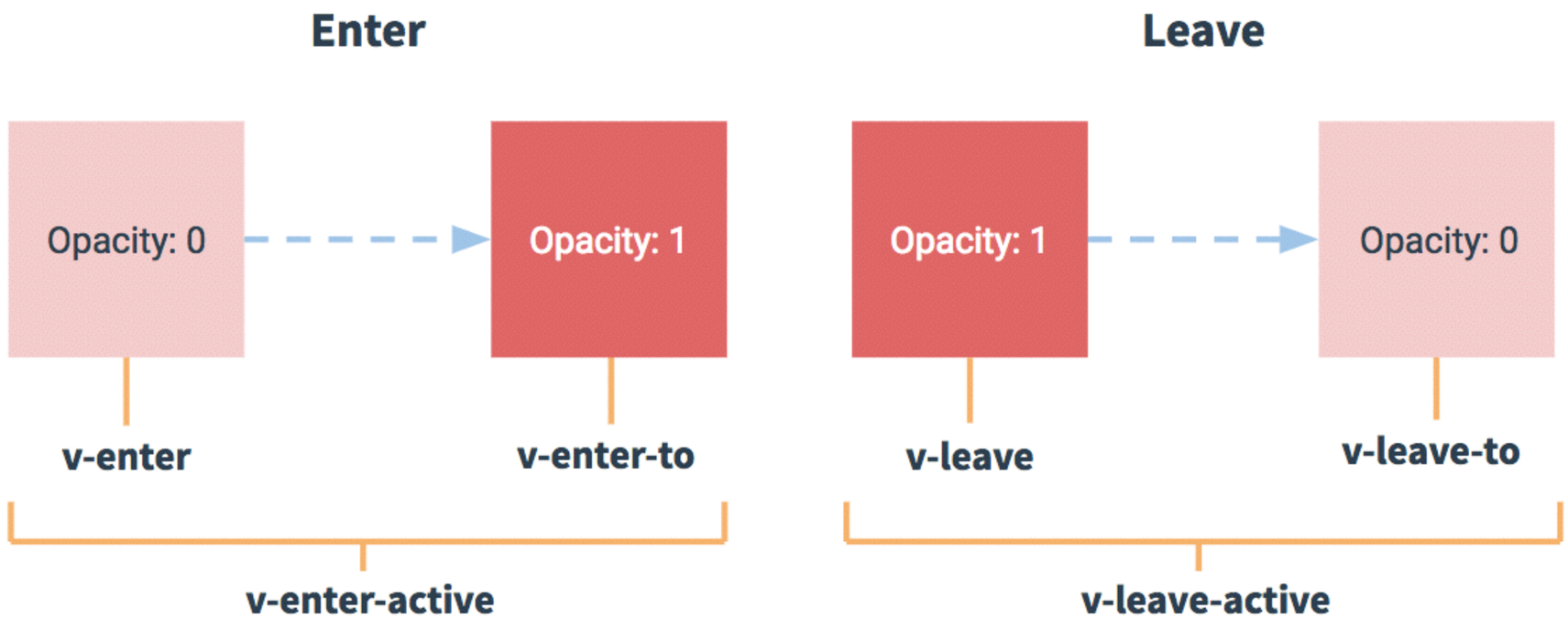
Node view

out-in

Node view

TRANSITION

Vue adds/ removes css classes during transition



USING CSS TRANSITIONS

```
.fade-enter {
    opacity: 0;
}

.fade-enter-active {
    transition: opacity .4s ease-out;
}

.fade-leave-active {
    opacity: 0;
    transition: opacity .4s ease-out;
}
```

USING CSS ANIMATIONS

```
@keyframes fade-enter {
    from { opacity: 0; }
}

@keyframes fade-leave {
    to { opacity: 0; }
}

.fade-enter-active {
    animation: fade-enter .4s ease-out;
}

.fade-leave-active {
    animation: fade-leave .4s ease-out;
}
```

TRANSITION-GROUP

TRANSITION-GROUP

deals with multiple items

```
<transition-group tag="ul" name="slide">
  <li v-for="item in items" :key="item.id">
    {{ item.text }}
  </li>
</transition-group>
```

EXAMPLE CSS

```
.slide-enter {
    transform: translateX(-100%);
}

.slide-enter-active {
    transition: transform .4s ease-out;
}

.slide-leave-active {
    transform: translateX(-100%);
    transition: transform .4s ease-out;
}

.slide-move {
    transition: transform .4s ease-out;
}
```

🏋️ EXERCISE TIME! 🏋️

Add some animations to the App!

Examples:

- Animate the message items using transition-group
- Animate the search sidebar

branch: exercise8

🏋️ EXERCISE TIME! 🏋️

solution

branch: exercise8-solution

DEALING WITH ASYNC DATA

LIFECYCLES

- beforeCreated
- **created** ← load data here
- beforeMount
- mounted
- beforeUpdate
- updated
- beforeDestroy
- destroyed

LOADING DATA ON INIT

```
new Vue({
    data() {
        return {
            items : [],
            loading : true
        }
    },
    created() {
        fetch('https://swapi.co/api/planets/1/')
            .then(res => res.json())
            .then(data => {
                this.loading = false;
                this.items = data;
            });
    }
})
```

OVER-VUE

WHAT WE LEARNED

1. We learned how to vueify our existing projects
2. We learned how to create and use components
3. How reactivity works
4. How to write templates
5. How to create customisable components using props or slots
6. How to use animations

NEXT MONDAY

1. Scaling up your Vue application using Vue CLI
2. Creating more maintainable code-bases
3. Advanced state management
4. Routing
5. Testing components

DE VOORHOEDE

front-end developers