

# Web Programming

## **Vue.js I.**

**Robert Ewald** | University of Stavanger

# Web frameworks

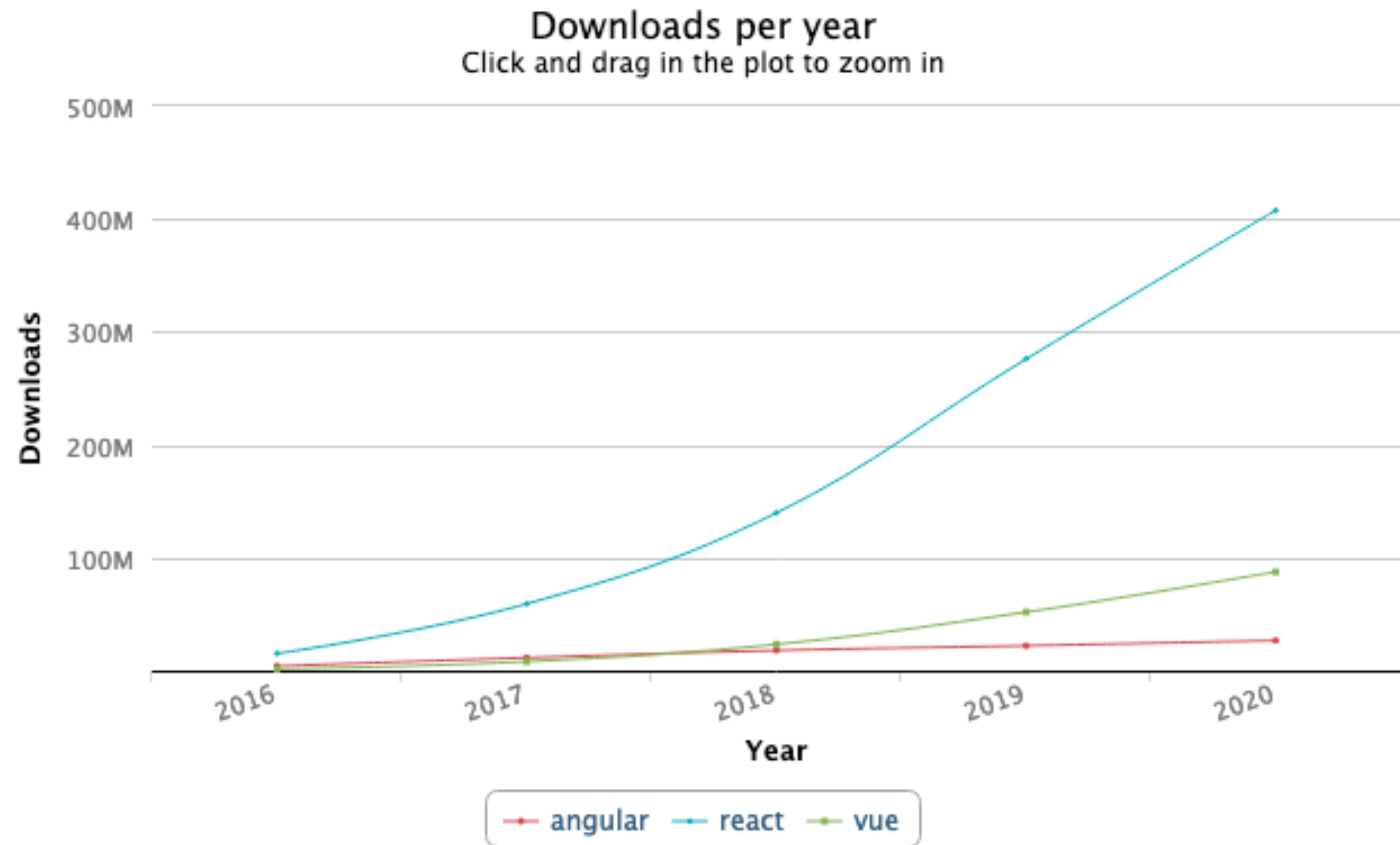
- Simplify development
- Code reuse
- Frameworks exist for all components, including
  - JavaScript
  - CSS
  - PHP
  - Python

# Simple JavaScript frameworks

- Aims
  - Consistent browser support
  - Simplified, easy-to-use functions
- Number of available options
  - jQuery UI
  - Angular
  - ReactJS
  - Vue

# The three big ones

- Angular
- React
- Vue



# **Pure JS**

## **(what is the problem?)**

- Source of truth (JS vs. HTML)
- Verbose DOM manipulation
- Difficult to separate application logic from display

# Source of truth

- Can be in the DOM or in JS

```
// This function uses the DOM (html) as source of truth
function add(){
  counter = document.getElementById("count");
  count = parseInt(counter.innerText) + 1;
  counter.innerText = count;
}

// a global variable
let count = 0;
// increment uses the global count
function increment(){
  count += 1;
  document.getElementById("count_2").innerText = count;
}
```

# Example #1

🔗 [examples/js/vue/no\\_vue/counter\\_local\\_state](#)

```
function createCounter(){
  let theCount = 0;
  let counter = document.createElement('div');
  counter.classList.add("counter");
  document.querySelector(".counter_container").appendChild(counter);

  let count = document.createElement('span');
  count.classList.add("count");
  count.innerText="0";
  counter.appendChild(count);

  let button = document.createElement('button');
  button.onclick = function(){
    theCount += 1;
    this.parentElement.querySelector(".count")
      .innerText = theCount;
  };
  button.innerText = "Add";
  counter.appendChild(button);
}
```

# Vue.js

## installation

- <https://v3.vuejs.org/guide>
- Simply include the vue library.

```
<!-- Import Vue -->  
<!-- development version, includes helpful console warnings -->  
<script src="https://unpkg.com/vue@3.0.5/dist/vue.global.js"></script>
```



# Vue.js

## My first app

- Create element to contain vue app:

```
<div id="app"></div>
```

- Create vue app in JS:

```
<script>
  var app = Vue.createApp({
    // specify a template
    template: "<strong>{{ message }}</strong>",
    // the data used in this app
  });
  // mount the app on the element #app
  app.mount("#app");
</script>
```

# Vue.js

## My first app

- Create element to contain vue app:

```
<div id="app"></div>
```

The template/application is placed inside this element.

- Create vue app in JS:

```
<script>
  var app = Vue.createApp({
    // specify a template
    template: "<strong>{{ message }}</strong>",
    // the data used in this app
  });
  // mount the app on the element #app
  app.mount("#app");
</script>
```

This tells Vue where the application lives.

# Vue.js

## My first app with data

- Create element to contain vue app:

```
<div id="app"></div>
```

- Create vue app in JS:

```
<script>
  var app = Vue.createApp({
    // specify a template
    template: "<strong>{{ message }}</strong>",
    // the data used in this app
    data: () => {return {
      message: "Hello Vue!"
    }}
  });
  // mount the app on the element #app
  app.mount("#app");
</script>
```

A function that returns an object, that is the data.

# Vue.js

## My first app with data

- Data is a function that returns an object

```
data: () => {  
  return {  
    message: "Hello Vue!"  
  }  
}
```

or

```
data: function() {  
  return {  
    message: "Hello Vue!"  
  }  
}
```

or

```
data() {  
  return {  
    message: "Hello Vue!"  
  }  
}
```

- Results in application data:

```
{  
  message: "Hello Vue!"  
}
```

# Vue.js

## Templates

- Content between `{{ }}` is a javascript expression
- The result is placed in the dom

```
{{ message }}
```

```
data: function(){  
  return {  
    message: "Hello Vue!"  
  }  
}
```

# Vue.js

## Specifying templates

- Plain JS string:

```
template: "<strong>{{ message }}</strong>"
```

- Template literal (multiline string)

```
template: `  
  <strong>  
    {{ message }}  
  </strong>  
`
```

- In the HTML (not good if component is reused)

```
<div id="app">  
  <strong>  
    {{ message }}  
  </strong>  
</div>
```

# Vue.js

## Specifying templates

- Plain JS string:

```
template: "<strong>{{ message }}</strong>"
```

- Template literal (multiline string)

```
template: /* html */ `  
  <strong>  
    {{ message }}  
  </strong>`
```

VSCode Extension: `es6-string-html` allows  
html syntax highlighting

- In the HTML template with id

```
<template id="my-component">  
  <strong>  
    {{ message }}  
  </strong>  
</template>
```

```
template: "#my-component"
```

# Vue.js

## Reactive

- Vue is **reactive**. If the data in JS changes, the DOM changes

```
<script>
  var app = Vue.createApp({
    // specify a template
    template: "<strong>{{ message }}</strong>",
    // the data used in this app
    data(){
      return {
        message: "Hello Vue!"
      }
    }
  });
  // mount the app on the element #app
  let vm = app.mount("#app");
</script>
```

- In the JS Console, we can change **message**

```
>>> vm.message = "Hello World!"
```



# v-bind attributes

- `{{ value }}` only works for text content!
- To assign a data element to an html attribute use **v-bind:**

```
// template including bound attributes
template: /*html*/`
  <img width="100"
    v-bind:src='image'
    v-bind:alt='desc'
  >`,
data: function() {
  return {
    image: "images/logo.png",
    desc: "Vue logo"
  }
}
```

To bind **attr** use **v-bind:attr**.  
No need for `{{ }}`.

# v-on and methods

- Specify methods of our app (component):

```
data(){  
  return { message: "Hello Vue!" }  
},  
methods: {  
  toggleMessage: function(){  
    if (this.message == "Hello Vue!")  
      this.message = "Can I help you?";  
    else this.message = "Hello Vue!";  
  }  
}
```

Access **message** in **data** as  
**this.message**

Can pass arguments here, e.g.  
**method(argument)**

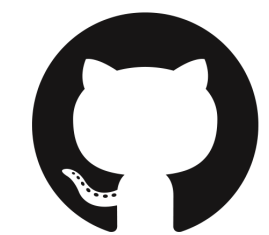
- Add event handler to template **v-on:event="method"**

```
template: /*html*/`  
  <div  
    v-on:mouseover="toggleMessage"  
    v-on:mouseout="toggleMessage"  
  >{{ message }}</div>`,
```

or

```
template: /*html*/`  
  <div  
    v-on:mouseover="toggleMessage()"  
    v-on:mouseout="toggleMessage()"  
  >{{ message }}</div>`,
```

# Exercise #1, #1b



[github.com/dat310-2022/info/tree/main/  
exercises/js/vue](https://github.com/dat310-2022/info/tree/main/exercises/js/vue)

# Expressions and computed properties

- You can specify expressions within {{ }}

```
<div id="app">
  {{ has_error ? "Oops": "OK" }}
</div>
```

- or

```
<div id="app">
  {{ count + 2 }}
</div>
```

- For an expensive expression use a computed property

```
template: "<p>{{ isEven ? 'even': 'odd' }}</p>",
computed: {
  isEven: function(){
    return (this.count%2 == 0);
  }
}
```

# Methods vs computed properties

- Methods are used with **v-on:event="method"**
  - executed on events
- Computed properties:
  - Specified as function that return value:

```
computed: {  
  isEven: function(){  
    return (this.count%2 == 0);  
  }  
}
```

- Use in template like element in data: **{{ isEven }}**
- Function is executed and value is updated reactively, i.e. when data changes

# Conditionals v-if, v-else

- Use **v-if="condition"** in the template
  - render only if **condition** is **true**
  - Can add element with **v-else**

```
template: /*html*/`
  <div v-if="easy" class="success">Have fun!</div>
  <div v-else class="error">Oh no!</div>`,
// the data used in this app
data(){
  return {
    easy: true
  }
}
```

# Render lists: v-for

- Use **v-for="item in array"** in the template
- Displays one element for each item in array

```
template: /*html*/`
  <ul>
    <li v-for="fruit in fruits">{{ fruit }}</li>
  </ul>`,
data(){
  return { fruits: ["apple", "pear", "banana", "orange"] }
}
```

- Use **v-for="item, index in array"** to get both item and index

```
template: `
  <ul>
    <li v-for="(fruit, index) in shortFruits">#{{ index }} {{ fruit }}</li>
  </ul>`,
```

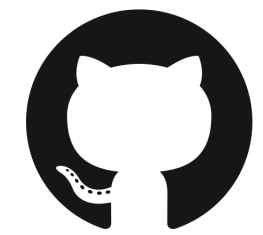
# Render lists: v-for

- Use computed property to filter array

```
template: `  
  <ul>  
    <li v-for="fruit in shortFruits">{{ fruit }}</li>  
  </ul>`,  
data(){  
  return { fruits: ["apple", "pear", "banana", "orange"] }  
},  
computed: {  
  shortFruits: function(){  
    return this.fruits  
      .filter(function(value){return (value.length <= 4)})  
  }  
}  
}
```



# Exercise #2



[github.com/dat310-2022/info/tree/main/  
exercises/js/vue](https://github.com/dat310-2022/info/tree/main/exercises/js/vue)

# Forms

- **v-bind** only creates a one way binding
  - i.e. changes in input below are not reflected in JS

```
<input id="name" v-bind:value="john.name"/>
```

- use **v-model** to create two way binding

```
<input type="text" id="nickname" v-model="john.nickname"/>
```

# Checkbox

- the model belonging to a checkbox will be true or false

```
<input id="checkbox" v-model="john.happy"/>
```

- multiple checkboxes with the same model result in an array

```
<input type="checkbox" value="CSS" v-model="skills"/>  
<input type="checkbox" value="JS" v-model="skills"/>  
<input type="checkbox" value="Python" v-model="skills"/>
```

```
data(){  
  return { skills: [] }  
}
```

# Example #2

🔄 examples/js/vue/vue6\_model/index.html

**Handling forms**

Name: John Doe

Nickname: jonni

Age: 32

☒ is happy

John knows: ☒ JS ☒ Python ☐ SQL

Johns favorite framework: ☐ Angular ☐ React ☒ Vue

Favorite database: MySQL

name John Doe

nickname jonni

skills [ "JS", "Python" ]

age 32

happy true

favorite Vue

db B

# Other

- **v-html** allows to update **innerHTML**

```
template: "<span v-html='msg'></span>",  
data: {  
  msg: "<em>Hello Vue!</em>"  
}
```

- **v-show** can be used like **v-if**, but toggles **display: none;**

```
template: `  
  <div v-show="easy" class="success">Have fun!</div>  
  <div v-show="!easy" class="error">Oh no!</div>`,
```

# Exercise #3, #4



[github.com/dat310-2022/info/tree/main/  
exercises/js/vue](https://github.com/dat310-2022/info/tree/main/exercises/js/vue)