

Advanced Web Programming

Introduction to Vue 3

Grzegorz Rogus
rogus@agh.edu.pl

Front-end foundations

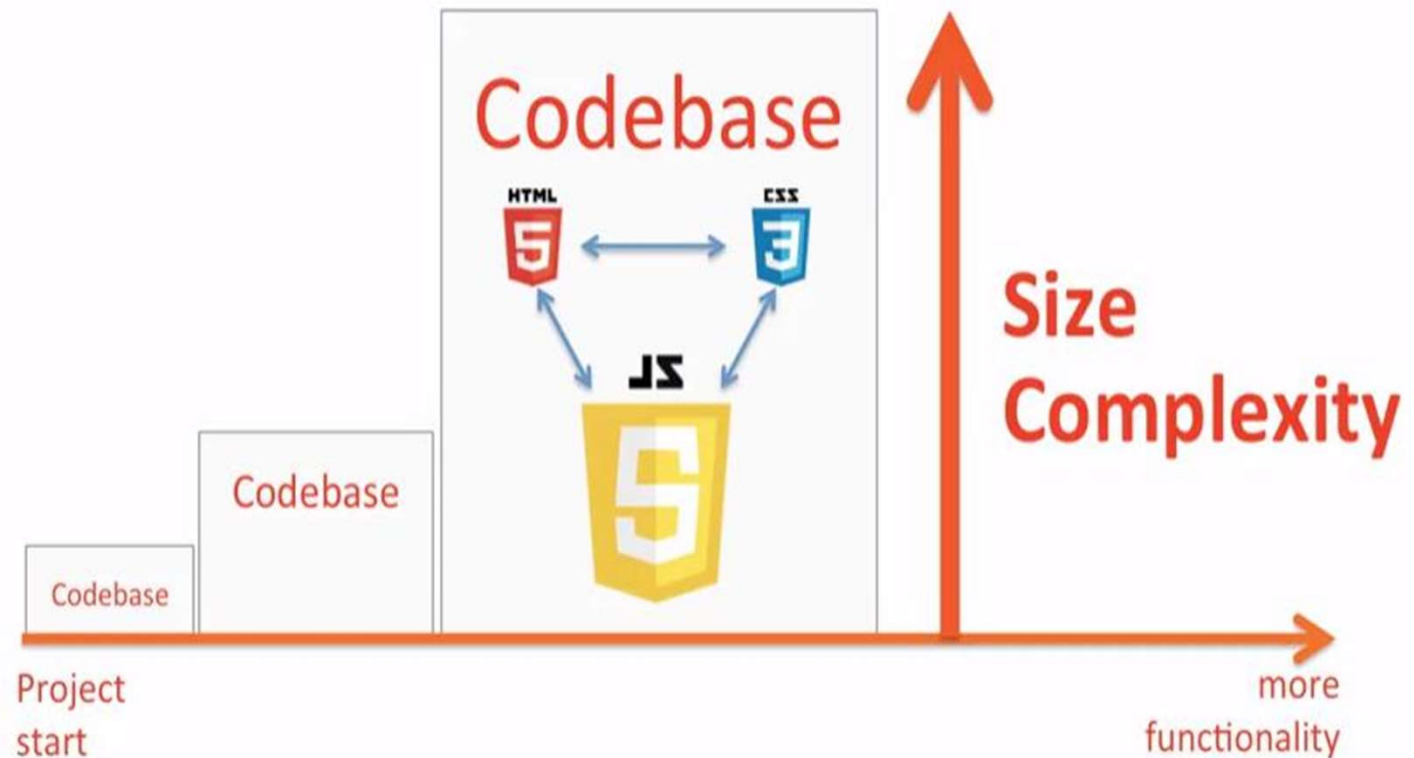


You can build ANY kind of web application with just “vanilla” JavaScript

- So why can't we stop there?
- Why use some frameworks, libraries, etc.?

Problem: Increase software complexity

- Applications are growing in size and complexity.
- Code management and software development are becoming more and more difficult.
- Applications are distributed for a long time by large teams.



Developers' expectations

- BETTER CODE ORGANIZATION ---> Speed of finding the appropriate code fragment (by any user)
- INDEPENDENT MODIFICATION OF THE CODE SECTION ---> Modification of functionality without the need to make changes to other parts of the code
- REUSING CODE ---> no need to write similar code twice
- EASY CODE TESTING ---> small code granularity

The purpose of using new technologies and frameworks is
a tackling the problem of complexity

Why frameworks – summary

No framework like Vue is needed!

You have to write all code on your own (“re-invent the wheel”)

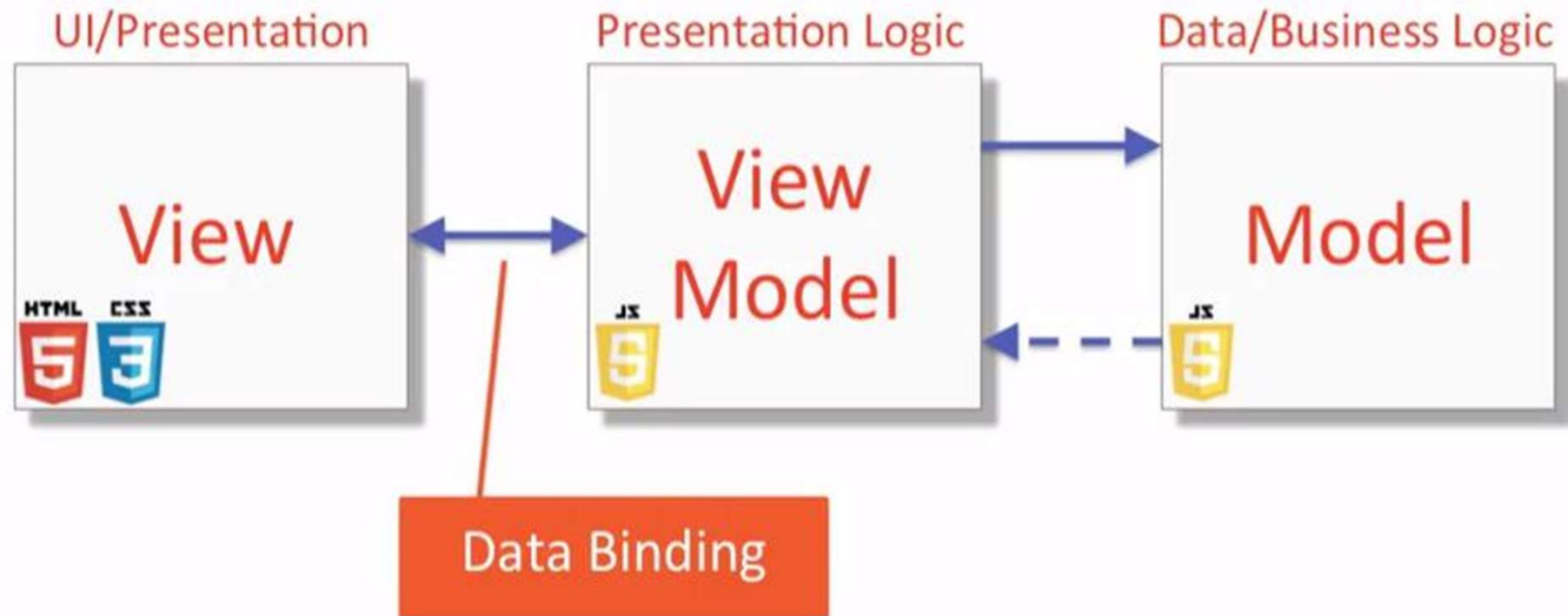
You might write suboptimal code or introduce errors & bugs

Working in a team might be harder because not everyone knows your structure and “code philosophy”

but it is better to use framework in larger projects

A pattern that solves the problem of complexity

Model-View-ViewModel



Model-View-ViewModel – implementation

Model

Represents and stores data

- Some of this data can be displayed in a view in various forms
- May contain logic used to response data from external sources (AJAX, REST API, etc.)
- It should not contain the logic associated with the display of the model

Model-View-ViewModel – implementation

View

User Interfejs

- In web applications implemented in HTML and CSS
- It only displays the data
- He should never change the data
- It is the source of events, but never manages or handles them

Model-View-ViewModel – implementation

ViewModel (kind of controler)

Represents the view state

- Stores the values of the data displayed in the view
- It responds to view events through presentation logic
- Calls other functions to implement complex business logic
- It never forces the view to display anything directly

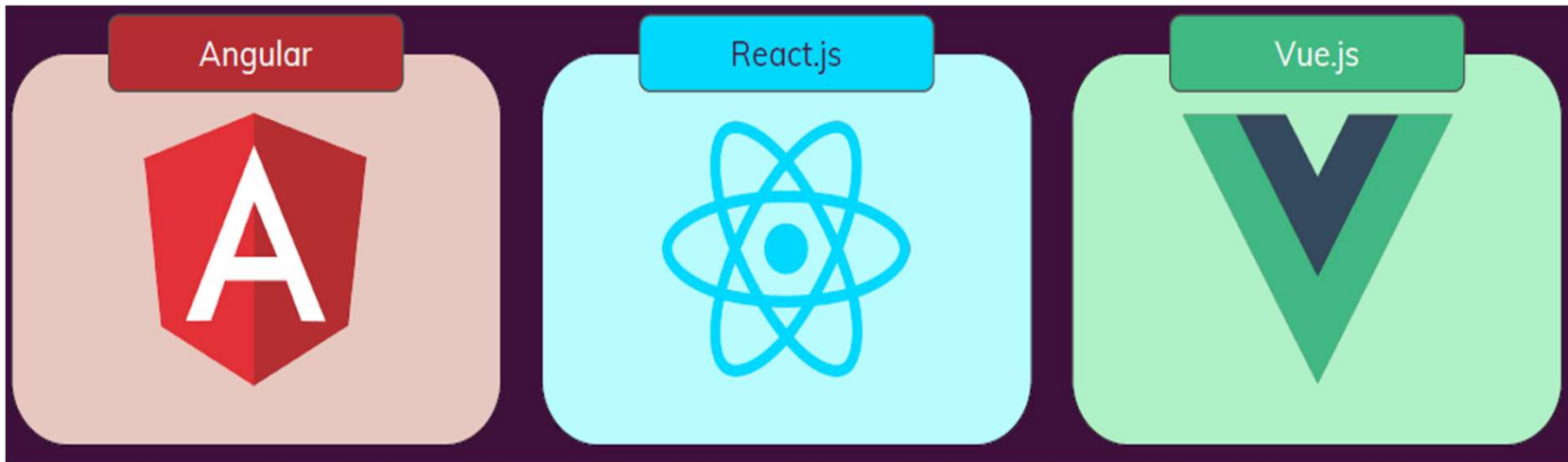
Model-View-ViewModel – implementation

Data Binder - it is magic

Connects the model from the ViewModel with the View

- It does not require any additional code - the framework does it "magically"
- The most important element of the entire MVVM pattern

Top 3 JS Frameworks



Similarity between these 3 frameworks:

- Very popular, used by top companies
- Mature, stable, funded long-term support
- Component oriented codebase
- Suitable for modern technological stacks (ES6+ / TypeScript)
- Large ecosystem of components and tools

Top 3 JS Frameworks



Lean and focused component based UI library.
Certain features (routing) are added via community packages.
The most popular, but not to all usages.



Complete component-based UI framework, includes most core features. A bit less popular than React & Angular.



Complete component-based UI framework, packed with features.
Uses TypeScript.
Can be overkill for smaller projects.
Dedicated to enterprise projects.

What is Vue.js?

Vue is an open-source model–view–viewmodel front-end JavaScript framework for building user interfaces and single-page applications.

It was created by Evan You, and is maintained by him and the rest of the active core team members.

en.wikipedia.org



Vue.js – main features?

Vue.js is a **JavaScript framework** that makes building interactive and **reactive web frontends**, which are basically web applications that run in the browser, so in the end what the user sees.

JavaScript

It allows us to manipulate the already running and that allows us to provide rich user experiences

Framework

Library that exposes utility functionalities AND a „set of rules” on how to build your jS application

Reactive

capacity to react to user input, update the screen dynamically etc.

Web frontend

What the user see

What is Vue.js - summary

- Front-end JavaScript/ TypeScript framework
- Used to create dynamic & data-driven and interactive website (SPA)
- Can also be used to create stand-alone widgets

Installation

Vue.js is built by design to be incrementally adoptable. This means that it can be integrated into a project multiple ways depending on the requirements.

There are four primary ways of adding Vue.js to a project:

1. Import it as a CDN package on the page
2. Download the JavaScript files and host them yourself
3. Install it using npm (install packages)
4. Use the official CLI to scaffold a project.

Application of Vue in WebApp

Vue can be used to **control parts** of HTML pages or entire pages.



“**Widget**” approach on a multipage-application.
(Some) pages are still **rendered on and served by a backend server**.

Vue can also be used to **control the entire frontend** of a web application



“Single-Page-Application” (SPA) approach. Server **only sends one HTML page**, thereafter, Vue takes over and controls the UI.

Vue Uses a “Declarative Approach”

You define the **result**, **NOT the way** to get there!

In “vanilla” JavaScript, you have to define all steps and control the entire way to get to a result

With frameworks like Vue, you define a template (with dynamic bindings and directives) and Vue takes care about the way to get the desired output onto the screen

Start template in Vue 3

Every Vue application starts by creating a new application instance with the `createApp` function:

```
// HTML file
<body>
  <div id="app">
    // Your app here
  </div>

  <script
    src="https://unpkg.com/vue@next">
  </script>
  <script src="app.js"> </script>

</body>
```

```
// app.js  JS file                                     Ver 1

Vue.createApp({
  // root instance definition
}) .mount("#app");
```

```
// app.js  JS file                                     Ver 2
import { createApp } from 'vue';

const app = createApp({
  // root instance definition
});

app.mount('#app');
```

The Vue Instance

You configure Vue via the Vue app object

Vue App Properties

What They Do

data

Set data that can be **used in the controlled HTML elements**. If **data changes**, Vue updates the **DOM**

methods

Define methods (functions) that can be **triggered from inside the controlled HTML elements** (and from inside the Vue instance).

If you control multiple, independent parts of HTML pages, you will often work with multiple Vue apps (you create multiple apps by calling `createApp()` more than once).

if you're building a SPA, you typically work with just one "root app" (`createApp()` is only used once in your entire codebase) and you instead build up a user interface with multiple components.

The first app in Vue – another example

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

<script src="http://unpkg.com/vue@next">
</script>

<script>
  const { createApp } = Vue;

  const App = {
    data() {
      return {
        name: "GR",
      };
    },
    template: `<h1>Hello {{ name }}</h1>`,
  };

  createApp(App).mount("#app");
</script>
```

Data Binding

Vue.js uses an HTML-based template syntax that allows you to declaratively bind the rendered DOM to the underlying instance's data.

1. Interpolations

One way binding: data -> view

The most basic form of data binding is text interpolation using the "Mustache" syntax (double curly braces):

```
<span>Message: {{ msg }}</span>
```

The mustache tag will be replaced with the value of the msg property from the corresponding Vue instance. It will also be updated whenever the msg property changes.

Expression in Data Binding

Vue.js supports the full power of JavaScript expressions inside all data bindings

EXPRESSIONS

```
<div id="app">
  <p>I have a {{ product }}</p>
  <p>{{ product + 's' }}</p>
  <p>{{ isWorking ? 'YES' : 'NO' }}</p>
  <p>{{ product.getSalePrice() }}</p>
</div>
```

These expressions will be evaluated as JavaScript in the data scope of the current active instance. One restriction is that each binding can only contain one single expression, so the following will NOT work:

`{{ var a = 1 }}` <!-- this is a statement, not an expression: -->

`{{ if (ok) { return message } }}`

Properties Binding

Mustaches cannot be used inside HTML attributes. Instead, use a **v-bind** directive.

2. v-bind

One way binding: data -> view

This is achieved with Vue's v-bind directive and would let us change the href (or any other HTML attribute) of a link or swap out an image if we need to.

```
<a v-bind:href="url">...</a>
```

shorthand

```
<a :href="url">...</a>
```

True or false will add or remove attribute:

```
<button :disabled="isActive">...
```

If isActive is truthy, the class 'active' will appear:

```
<div :class="{ active: isActive }">...
```

Style color set to value of activeColor:

```
<div :style="{ color: activeColor }">
```

event handling in Vue

- How to pass information from HTML to Biznes Logic

Ver 1

```
<input type="text" v-on:input="setName($event, 'Rogus')">
```

Ver 2

```
<input type="text" v-model="name">
```

Event Handling

Just like all other properties of the Vue instance, the methods are accessible from within the component's template. Inside a template they are most commonly used as event listeners:

```
<button v-on:click="increment">Up vote</button>
```

In the example above, the method `increment` will be called when the `<button>` is clicked. It is an example of event handling – the third data binding methods.

3. `v-on`:

One way binding: view -> data

how to respond to and handle user events with Vue's `v-on` directive

Calls `addToCart` method on component:

```
<button @click="addToCart">...
```

shorthand:

```
<button v-on:click="addToCart">...
```

Arguments can be passed:

```
<button @click="addToCart(product)">...
```

Event Handling

Calls addToCart method on component:

```
<button v-on:click="addToCart">...
```

shorthand

```
<button @click="addToCart">
```

Arguments can be passed:

```
<button @click="addToCart(product)">...
```

To prevent default behavior (e.g. page reload):

```
<form @submit.prevent="addProduct">...
```

Only trigger once:

```
<img @mouseover.once="showImage">...
```

Event Modifiers

It is a very common need to call `event.preventDefault()` or `event.stopPropagation()` inside event handlers. Although we can do this easily inside methods, it would be better if the methods can be purely about data logic rather than having to deal with DOM event details.

To address this problem, Vue provides event modifiers for `v-on`. Recall that modifiers are directive postfixes denoted by a dot.

<code>.stop</code>	<code><!-- the click event's propagation will be stopped --></code>
<code>.prevent</code>	<code><a @click.stop="doThis"></code>
<code>.capture</code>	<code><!-- the submit event will no longer reload the page --></code>
<code>.self</code>	<code><form @submit.prevent="onSubmit"></form></code>
<code>.once</code>	<code><!-- modifiers can be chained --></code>
<code>.passive</code>	<code><a @click.stop.prevent="doThat"></code>
	<code><!-- the click event will be triggered at most once --></code>
	<code><a @click.once="doThis"></code>

Key Modifiers

When listening for keyboard events, we often need to check for specific keys. Vue allows adding key modifiers for v-on or @ when listening for key events:

Keyboard entry example:

```
<input @keyup.enter="submit">
```

Call onCopy when control-c is pressed:

```
<input @keyup.ctrl.c="onCopy">
```

Key modifiers:

.tab	.up	.ctrl
.delete	.down	.alt
.esc	.left	.shift
.space	.right	.meta

Mouse modifiers:

.left	.right	.middle
-------	--------	---------

Form Input Bindings

You can use the v-model directive to create two-way data bindings on form input, textarea, and select elements. It automatically picks the correct way to update the element based on the input type. .

2. v-model

Two way binding: data <-> view

v-model internally uses different properties and emits different events for different input elements:

- text and textarea elements use value property and input event;
- checkboxes and radiobuttons use checked property and change event;
- select fields use value as a prop and change as an event.

Text

```
<input v-model="message" placeholder="edit me" />
<p>Message is: {{ message }}</p>
```

Checkbox (boolean value):

```
<input type="checkbox" id="checkbox" v-model="checked" />
<label for="checkbox">{{ checked }}</label>
```

Two way binding in Vue

```
<input type="text" v-bind:value="name" v-on:input="setName()">
```

is equivalent

```
<input type="text" v-model="name">
```

Two-way data binding:

```
<input v-model="firstName" >
```

`v-model.lazy="..."` Syncs input after change event

`v-model.number="..."` Always returns a number

`v-model.trim="..."` Strips whitespace

Vue – data binding - summary

Data binding

Interpolation `{{ name }}` `{{ funA() }}`

Property binding `v-bind:` in attribute HTML tag

Property binding `v-bind:href="vueLink"`

shorthand `:href="vueLink"`

Event binding

Event `v-on: click="eventHandler()"`

shorthand: `@click="eventHandler()"`

Two-way binding

Data Binding + Event Binding = Two-Way Binding

`<Input text="type" v-model: "name" >`

Directive: Conditional Rendering

The directive `v-if` is used to conditionally render a block. The block will only be rendered if the directive's expression returns a truthy value.

```
<h1 v-if="awesome">Vue is awesome!</h1>
```

It is also possible to add an "else block" with `v-else`:

```
<h1 v-if="awesome">Vue is awesome!</h1>
```

```
<h1 v-else>Oh no 😞</h1>
```

if we want to toggle more than one element we can use `v-if` on a `<template>` element, which serves as an invisible wrapper. The final rendered result will not include the `<template>` element.

```
<template v-if="ok">
  <h1>Title</h1>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</template>
```

Conditional Rendering

Another option for conditionally displaying an element is the `v-show` directive. The usage is largely the same:

```
<h1 v-show="ok">Hello!</h1>
```

The difference is that an element with `v-show` will always be rendered and remain in the DOM; `v-show` only toggles the `display` CSS property of the element.

`v-show` doesn't support the `<template>` element, nor does it work with `v-else`

Element inserted/removed based on truthiness:

```
<p v-if="inStock">{{ product }}</p>
```

```
<p v-else-if="onSale">...</p>
```

```
<p v-else>...</p>
```

Toggles the `display: none` CSS property:

```
<p v-show="showProductDetails">...</p>
```

List Rendering

- v-for can be used to render multiple elements dynamically
- v-for can be used with arrays, objects and ranges (numbers).

We can use the v-for directive to render a list of items based on an array. The v-for directive requires a special syntax in the form of item in items, where items is the source data array and item is an alias for the array element being iterated on:

```
<ul id="array-rendering">
  <li v-for="item in items">
    {{ item.message }}
  </li>
</ul>
```

```
Vue.createApp({
  data() {
    return {
      items: [{ message: 'Foo' },
              { message: 'Bar' }]
    }
  }
}).mount('#array-rendering')
```

List Rendering

You can extract **values**, values and **indexes** or values, **keys** and indexes

```
<li v-for="item in items" :key="item.id">
  {{ item }}
</li>
```

key always recommended

To access the position in the array:

```
<li v-for="(item, index) in items">...
```

To iterate through objects:

```
<li v-for="(value, key) in object">...
```

If you need v-for and v-if, **DON'T use them on the same element**. Use a wrapper with v-if instead.

Good practic in coding in Vue

2-way Binding

Never use v-if on the same element as v-for

Template

```
<input [(ngModel)]="searchBox">
```

✖ Bad

```
1 <ul>
2   <li
3     v-for="user in users"
4     v-if="user.isActive"
5     :key="user.id"
6   >
7     {{ user.name }}
8   </li>
9 </ul>
```

html

List Rendering for collection of objects

```
const shoppingListApp = Vue.createApp({
  data() {
    return {
      header: 'Shopping List App',
      items:[
        {id: 1, label:'product nr one'},
        {id: 2, label:' product nr two '},
        {id: 3, label:' product nr three '},
      ]
    }
  }
}).mount('#shopping-list')
```

Data model

Data Binding

View

```
<div id="shopping-list">
  <h1>{{ header || 'Welcome' }}</h1>
  <ul>
    <li v-for="item in items" :key="item.id">{{item.label}}</li>
  </ul>
</div>
```

Computed Property and Watcher

Computed properties and watchers allow you to react to data changes

Computed properties

- Use with data binding
- computed properties are cached based on their reactive dependencies.
- Computed properties are only re-evaluated if one of their “used values” changed -
- Use for data that depends on other data

Watcher

- Not used directly in template
- Allows you to run any code in reaction to some changed data (e.g. send Http request etc.)
- Use for any non-data update you want to make

For example: as long as `author.books` has not changed, multiple access to the `publishedBooksMessage` computed property will immediately return the previously computed result without having to run the function again.

COMPONENT STRUCTURE

```
export default {  
  name: 'RangeSlider',  
  mixins: [], // share common functionality with component mixins  
  extends: {}, // compose new components  
  props: {}, // component properties/variables  
  data() {}, // variables  
  computed: {},  
  components: {}, // when component uses other components  
  watch: {}, // watch variables  
  methods: {}, // methods  
  // component Lifecycle hooks  
  beforeCreate() {},  
  created() {},  
  beforeMount() {},  
  mounted() {},  
  beforeUpdate() {},  
  updated() {},  
  activated() {},  
  deactivated() {},
```

Installing & Using the Vue CLI

```
npm install -g @vue/cli
```

```
vue --version
```

Check VueCLI version

```
@vue/cli 4.5.13
```

```
vue create first-app-vue
```

```
Vue CLI v4.5.13
? Please pick a preset:
  Default ([Vue 2] babel, eslint)
> Default (Vue 3) ([Vue 3] babel, eslint)
  Manually select features
```

Runing Vue project cd.

```
DONE Compiled successfully in 4875ms 21:20:23
```

```
App running at:
```

- Local: <http://localhost:8080/>
- Network: <http://192.168.0.38:8080/>

```
Note that the development build is not optimized.  
To create a production build, run npm run build.
```

```
Vue CLI v4.5.13
```

```
? Please pick a preset:
```

```
  Default ([Vue 2] babel, eslint)
```

```
> Default (Vue 3) ([Vue 3] babel, eslint)
```

```
  Manually select features
```

- ❑ Successfully created project `first-app-vue`.
- ❑ Get started with the following commands:

```
$ cd first-app-vue
```

```
$ npm run serve
```

```
C:\Vue_project>
```

Vue project structure



Welcome to Your Vue.js App

For a guide and recipes on how to configure / customize this project,
check out the [vue-cli documentation](#).

Installed CLI Plugins

[babel](#) [eslint](#)

Essential Links

[Core Docs](#) [Forum](#) [Community Chat](#) [Twitter](#) [News](#)

Ecosystem

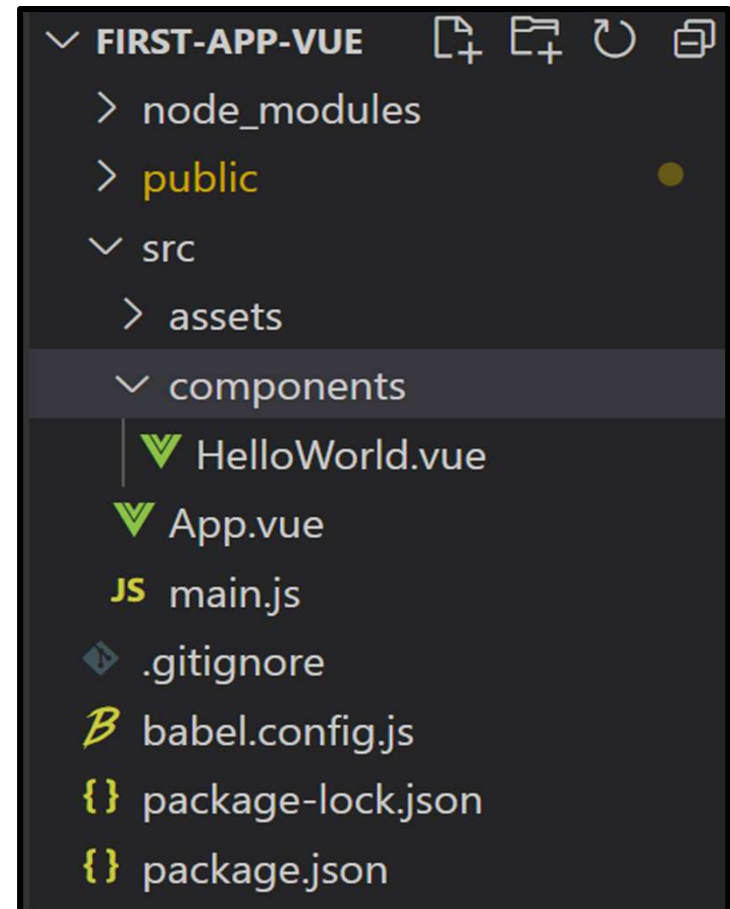
[vue-router](#) [vuex](#) [vue-devtools](#) [vue-loader](#) [awesome-vue](#)

Vue file

```
<template>
  
  <p> {{ name }} </p>
</template>
```

```
<script>
export default {
  name: 'App',
  data() { return { name : "GR"}},
}
</script>
```

```
<style>
p {
  color: blue;
  font: italic bold 5em serif;
}
</style>
```



Component Communication


“Props” is a special keyword which stands for properties. It can be registered on a component to pass data from a parent component to one of its children components.

Data in props can only flow one way – from the top, or parent component, to the bottom, or child components. This simply means you cannot pass data from a child to a parent.

Another thing to keep in mind is that Props are read-only and cannot be modified by the child component because the parent component “owns” that value.

```
<template>
  <p>Hi {{ firstName }} {{ lastName }}</p>
</template>

<script>
export default {
  props: [
    'firstName',
    'lastName'
  ],
}
</script>
```



```
props: {
  firstName: String,
  lastName: String
},
```