INFO3180 - LECTURE 7 (CONT'D)

FRONT END LIBRARIES AND TOOLS AND VUEJS

VITE

***Vite** is a build tool that aims to provide a faster and leaner development experience for modern web projects.*

https://vitejs.dev/guide/

FEATURES OF VITE

- Provides a dev server with Hot Module Replacement (HMR)
- Pre-configured to output highly optimized static assets for production deployments.
- Configured to work out of the box with TypeScript, JSX, CSS, SASS, etc.
- Handles importing static assets like images

WHAT PROBLEMS DOES VITE PRIMARILY SOLVE

- Slow Dev-Server Start
- Slow Updates

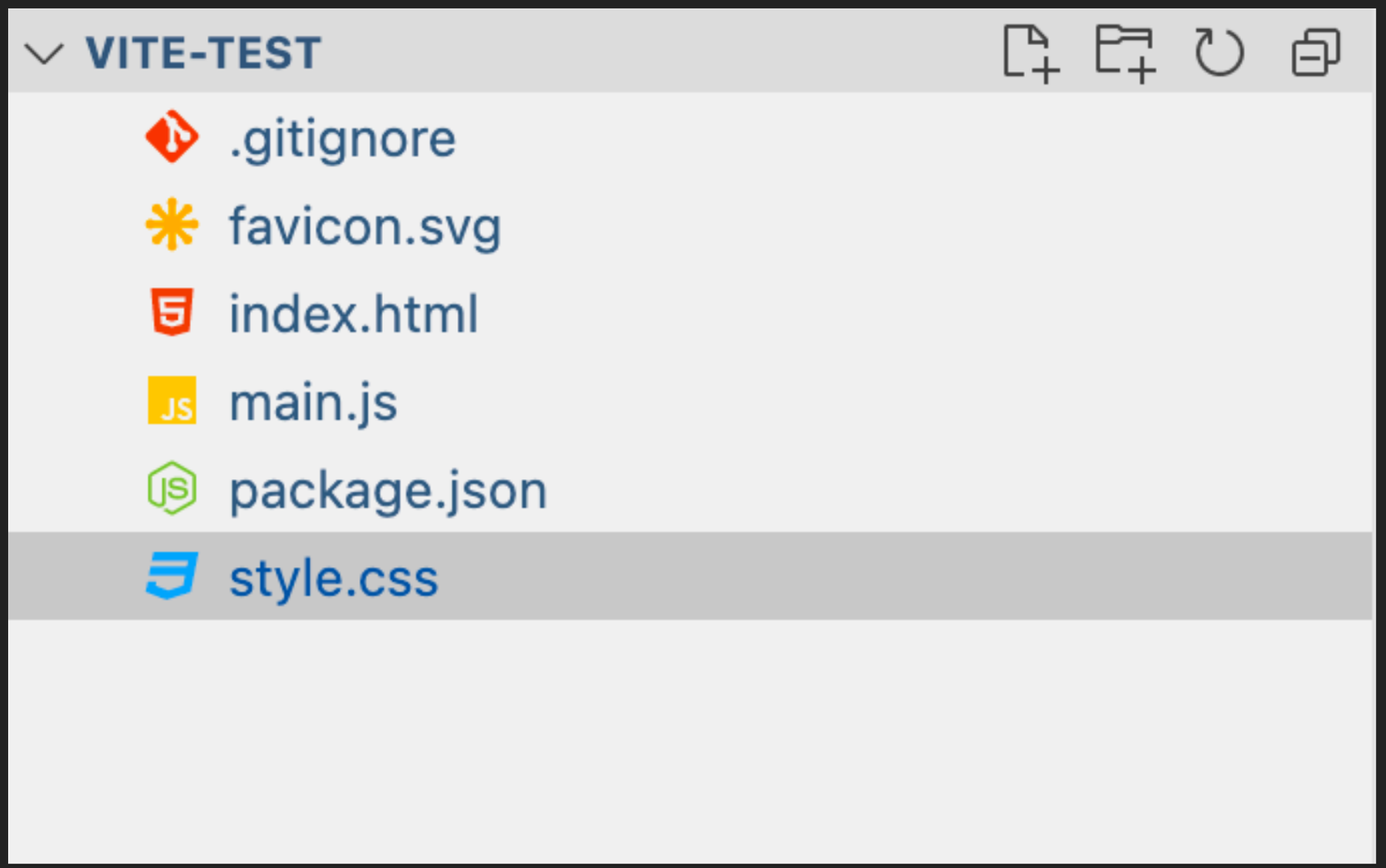
Assuming you already have NodeJS installed, to start a project with Vite, you would run the following at the command line:

```
npm create vite@latest
```

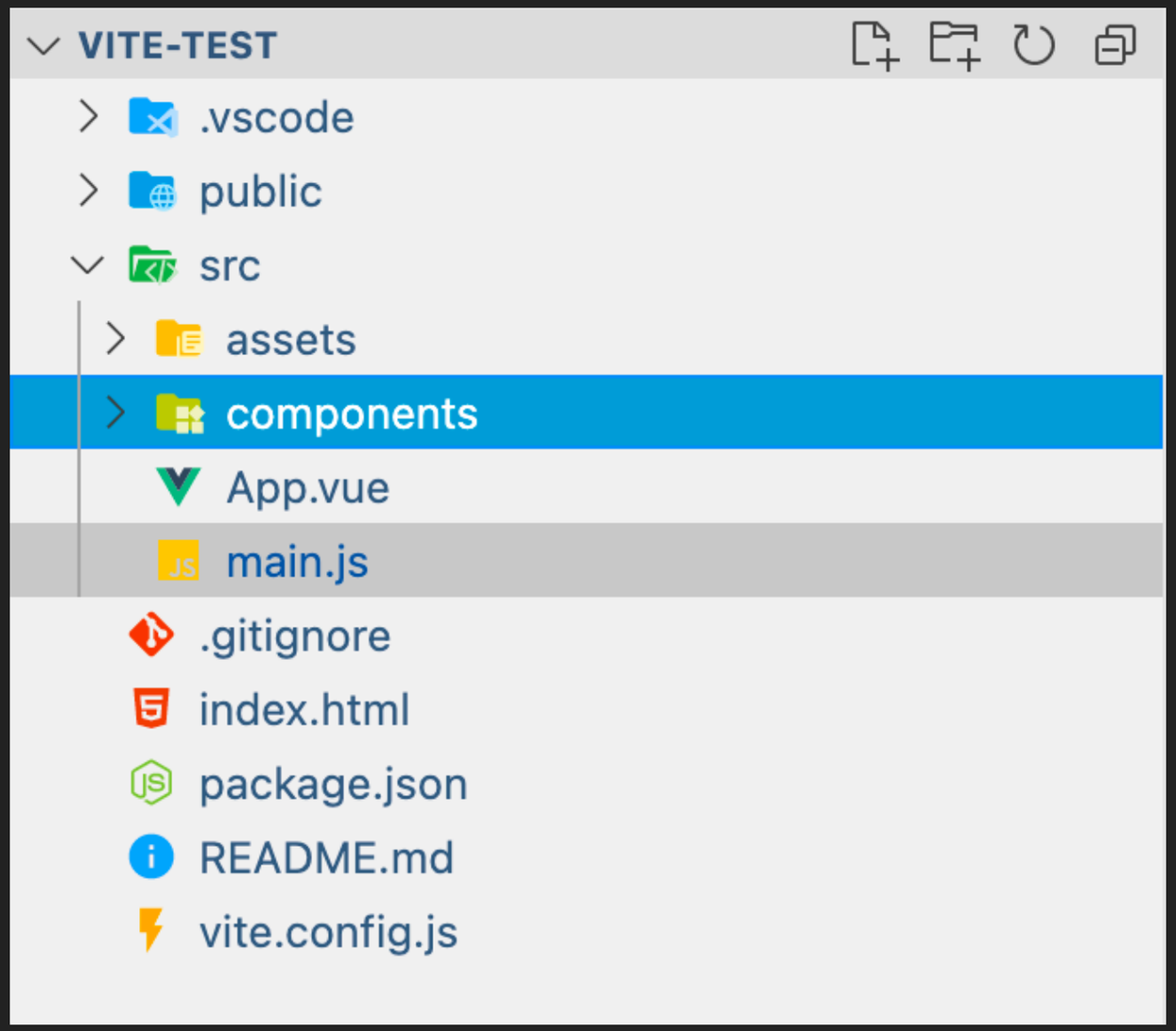
Vite will allow you to use a predefined template for some of the common frontend frameworks.

```
➜  ~ npm create vite@latest
npx: installed 6 in 2.93s
✔ Project name: … vite-test
? Select a framework: › - Use arrow-keys. Return to submit.
❯   vanilla
    vue
    react
    preact
    lit
    svelte
```

An example of structure of the 'vanilla' JavaScript template.



An example of structure of the VueJS template.



Once you have setup your project with Vite, to start the development server you would run:

```
cd your-project-name
npm install
npm run dev
```

VUE.JS

***Vue** (pronounced like view) is a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS, and JavaScript and provides a declarative and component-based programming model that helps you efficiently develop user interfaces, be they simple or complex.*

https://vuejs.org/guide/introduction.html

It has similarities to other frameworks/libraries such as Angular and ReactJS.

VUEJS FEATURES

- perfect for Single Page Applications (SPA's)
- provides Two-way Data-binding so that it synchronizes changes between the data model and the HTML view.
- Create your own reusable components
- provides routing for SPAs (via an extension called `vue-router`)
- Provides ways to apply transition effects when items are inserted, updated, or removed from the DOM
- among other features

The simplest way to begin using VueJS is to load the library in your HTML file.

```
<script src="https://unpkg.com/vue@3"></script>
```

Note: You could also download it locally and store in a folder in your project or install it via NPM.

If you prefer to use VueJS with build tools, then install NodeJS and run:

```
npm init vue@latest
cd your-project-name
npm install
npm run dev
```

At the core of Vue.js is a system that enables us to declaratively render data to the DOM using straightforward template syntax.

A BASIC VUE APP

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

<script>
const App = {
  data() {
    return {
      message: 'Hello Vue!'
    }
  }
};

Vue.createApp(App).mount('#app')
</script>
```

VueJS has special attributes called directives. Directives are prefixed with `v-` and they apply special reactive behaviour to the rendered DOM. Some common directives are `v-bind`, `v-if`, `v-for`, `v-on` and `v-model`.

CONDITIONALS

Here we use the `v-if` directive to toggle the presence of an element.

```
<div id="app">
  <span v-if="seen">Now you see me!</span>
</div>

<script>
const App = {
  data() {
    return {
      seen: true
    }
  }
};

Vue.createApp(App).mount('#app')
</script>
```

LOOPS

Let's say we have an array of todo items in our Vue instance data object.

```
<script>
const App = {
  data() {
    return {
      todos: [
        { text: 'Learn JavaScript' },
        { text: 'Learn Vue' },
        { text: 'Build something awesome' }
      ]
    }
  }
};

Vue.createApp(App).mount('#app')
</script>
```

LOOPS

Here we use the `v-for` directive to loop over the todo items that were in our data object in the Vue instance

```
<div id="app">
  <ol>
    <li v-for="todo in todos">
      {{ todo.text }}
    </li>
  </ol>
</div>
```

HANDLING USER INPUT

We can use the `v-on` directive to attach JavaScript event listeners that call methods.

HANDLING USER INPUT

```
<div id="app">
  <p>{{ message }}</p>
  <button v-on:click="reverseMessage">
    Reverse Message
  </button>
</div>
```

You could also use the shorthand **@click** instead of **v-on:click**.

HANDLING USER INPUT

```
<script>
const App = {
  data() {
    return {
      message: 'Hello Vue.js!'
    }
  },
  methods: {
    reverseMessage() {
      this.message = this.message.split('').reverse()
                     .join('')
    }
  }
};

Vue.createApp(App).mount('#app')
</script>
```

HANDLING USER INPUT

VueJS also has the `v-model` directive that makes two-way binding between form input and app state a breeze.

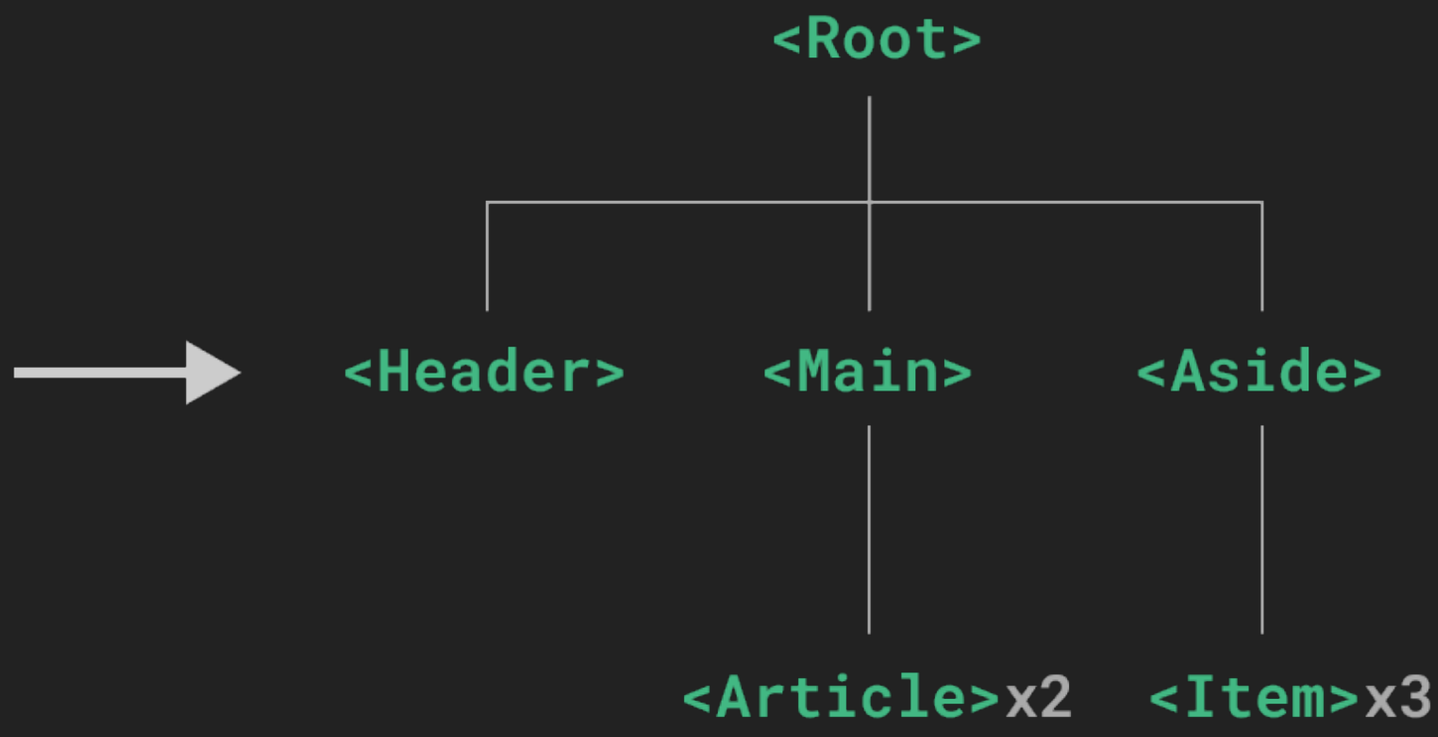
HANDLING USER INPUT

```
<div id="app">
  <p>{{ message }}</p>
  <input v-model="message" />
</div>

<script>
const App = {
  data() {
    return {
      message: 'Hello Vue.js!'
    }
  }
};

Vue.createApp(App).mount('#app')
</script>
```

COMPONENTS

The component system is another important concept in Vue, because it's an abstraction that allows us to build large-scale applications composed of small, self-contained, and often reusable components.

Almost any type of application interface can be abstracted into a tree of components


<Root>
<Header>
<Main>
<Aside>
<Article>x2
<Item>x3

When using a build step, we typically define each Vue component in a dedicated file using the .vue extension - known as a Single-File Component (SFC for short)

```
<script>
  export default {
    data() {
      return {
        count: 0
      }
    }
  }
</script>

<template>
  <button @click="count++">You clicked me {{ count }} times.</button>
</template>
```

When not using a build step, a Vue component can be defined as a plain JavaScript object containing Vue-specific options

```
export default {
  data() {
    return {
      count: 0
    }
  },
  template: `
    <button @click="count++">
      You clicked me {{ count }} times.
    </button>`
}
```

To use a child component, we need to import it in a parent component.

```vue
<script>
import ButtonCounter from './ButtonCounter.vue'

export default {
  components: {
    ButtonCounter
  }
}
</script>

<template>
  <h1>Here is a child component!</h1>
  <ButtonCounter />
</template>
```

In a larger applications, it will probably be necessary to divide the whole app into components to make development manageable.

SINGLE FILE COMPONENTS (SFC)

VueJS also has the concept of Single File Components. Each component will be defined in a file having a `.vue` extension and will contain the HTML template, JavaScript and Scoped CSS needed to make the component work bundled together.

To learn more about Single File Components , take a look at:
https://vuejs.org/guide/scaling-up/sfc.html

```
<script>
export default {
  data() {
    return {
      greeting: 'Hello World!'
    }
  }
}
</script>

<template>
  <p class="greeting">{{ greeting }}</p>
</template>

<style>
.greeting {
  color: red;
  font-weight: bold;
}
</style>
```

Vue components can be authored in two different API styles: Options API and Composition API.

OPTIONS API

```vue
<script>
export default {
  // Properties returned from data() become reactive state
  // and will be exposed on `this`.
  data() {
    return {
      count: 0
    }
  },

  // Methods are functions that mutate state and trigger updates.
  // They can be bound as event listeners in templates.
  methods: {
    increment() {
      this.count++
    }
  },

  // Lifecycle hooks are called at different stages
  // of a component's lifecycle.
  // This function will be called when the component is mounted.
  mounted() {
    console.log(`The initial count is ${this.count}.`)
  }
}
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

COMPOSITION API

```vue
<script setup>
  import { ref, onMounted } from 'vue'

  // reactive state
  const count = ref(0)

  // functions that mutate state and trigger updates
  function increment() {
    count.value++
  }

  // lifecycle hooks
  onMounted(() => {
    console.log(`The initial count is ${count.value}.`)
  })
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

COMPONENT PROPS

Props are custom attributes you can register on a component.

DEFINING PROPS

```vue
<!-- HelloWorld.vue -->
<script setup>
  defineProps(['fullname'])
</script>

<template>
  <h4>Hello {{ fullname }}!</h4>
</template>
```

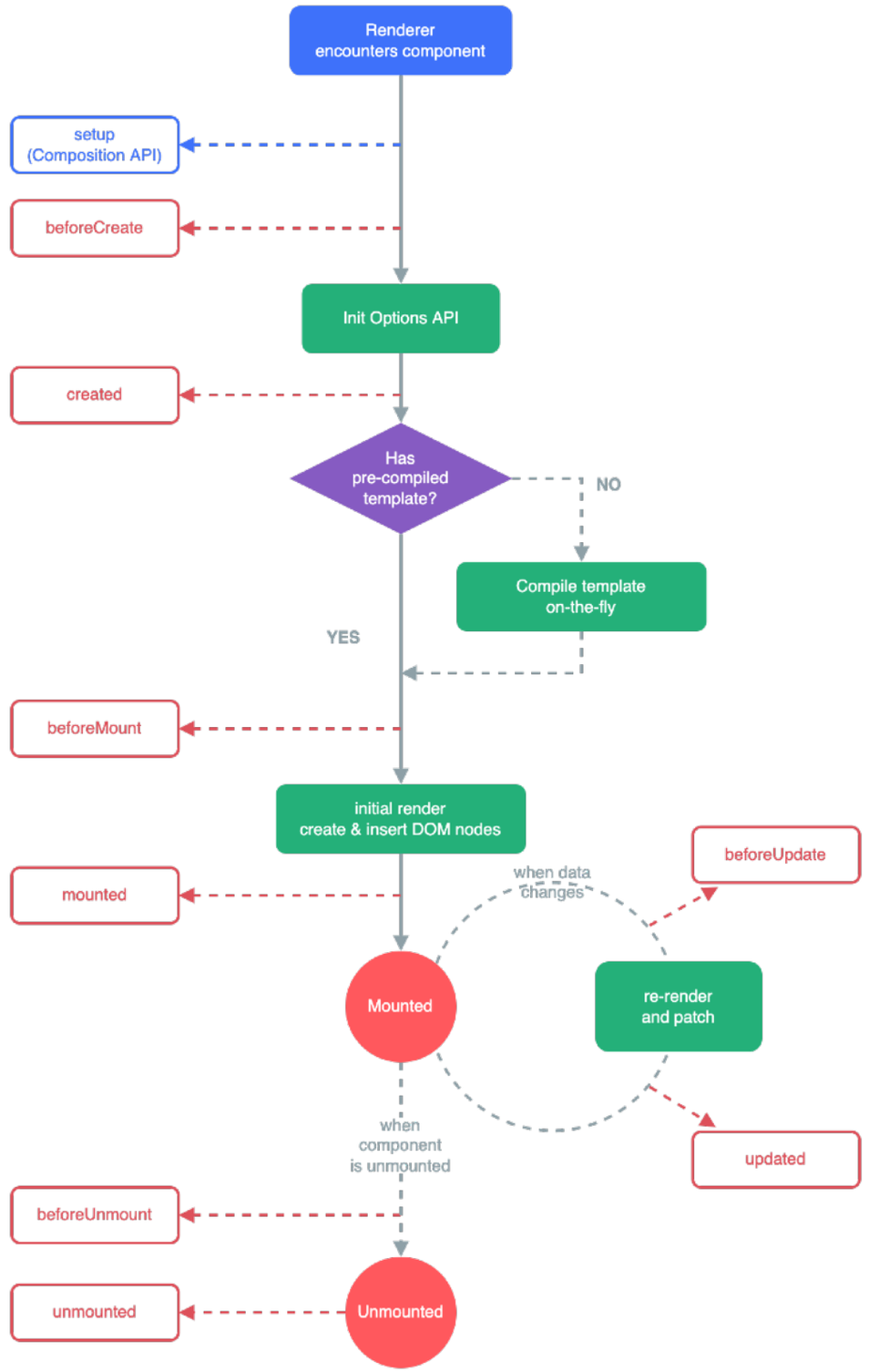
PROPS

Once a prop is registered, you can pass data to it as a custom attribute, like this:

```
<HelloWorld fullname="John Doe" />
<HelloWorld fullname="Jane Doe" />
```

LIFECYCLE HOOKS

Each Vue component also goes through a series of initialization steps when it is created. Vue therefore has some functions called Lifecycle Hooks, that will allow you to add code to do certain things at specific stages.



https://vuejs.org/guide/essentials/lifecycle.html#lifecycle-diagram

LIFECYCLE HOOKS

The most commonly used lifecycle hooks are `mounted`, `updated`, `unmounted`.

VUE DEVELOPER TOOLS

And to help you during development, VueJS has Web Browser devtools extensions for debugging Vue.js applications. You can install vue-devtools for Firefox and Chrome.

https://devtools.vuejs.org

MyApp › Components

Find components...

<NativeTypes> Filter state...

<MyApp> fragment 16 ms
  <Child>
  <NestedMore> fragment
    <Nested key='foobar'>
  <NativeTypes>
    <Anonymous Component>
  <EventEmit>
  <EventNesting>
    <EventNesting key=0>
  <AsyncComponent>
    <AsyncComponentWrapper>
  <SuspenseExample>
    <AsyncSetup> suspense
  <Provide>
    <Inject>
  <Condition>

▼ props

multiTypeProp: false

▼ data

aWeirdFunction: ƒ weird(a, b, c)
anon: ƒ anon(foo, bar)
arrow: ƒ arrow(a, b)
def: OtherWithMine (src/Other.vue)
def2: MyComponent
def3: Unknown Component
hello: ƒ foo(a, b, c)
hey: ƒ empty()
html: "Bold Italic"
html key: ƒ html key(h, t, m, l)
htmlReg: /hey<\/b>/i

Here is a link to an introductory video series on VueJS:

https://www.vuemastery.com/courses/intro-to-vue-3/intro-to-vue3

RESOURCES

- ViteJS - https://vitejs.dev
- VueJS - https://vuejs.org
- Intro to Vue 3 - https://www.vuemastery.com/courses/intro-to-vue-3/intro-to-vue3
- VueRouter - https://router.vuejs.org
- Vue DevTools - https://devtools.vuejs.org
- Vue Language Features (Volar) for VSCode - https://marketplace.visualstudio.com/items?itemName=Vue.volar

DEMO