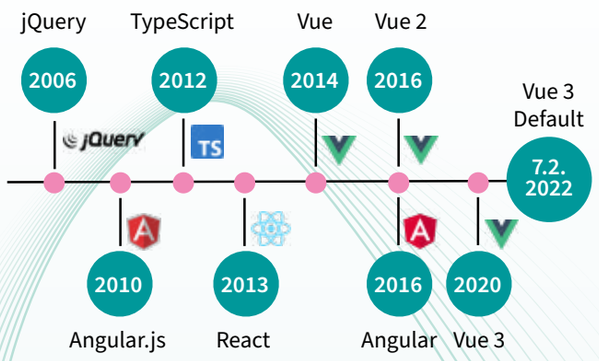
Vue 3

History



Minimal-App

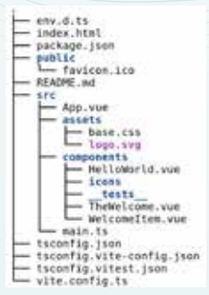
```
<script type="text/javascript">
 const AppComponent = {
  data() {
   return {
    message: "Hello Vue!"
   }
  }
 }
 Vue.createApp(AppComponent).mount('#app')
</script>
```

Selector

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

Generate projects: Create Vue

```
$ npm init vue@3
```



- env.d.ts — TypeScript typings
- index.html — Host Site
- package.json — Dependencies & scripts
- public — Folder for static dependencies
- src — Source folder
- src/App.vue — Basic app component
- src/assets — Assets used in components
- src/components — More components
- main.ts — Bootstrap the app
- tsconfig.* — TypeScript compiler configs
- vite.config.ts — Configuration of the build tool (Vite)

```
$ npm install
$ npm run dev
```

Vue 3 Overview

Vue has a long history: After React but before Angular, Vue was spearheaded by Evan You. As a former co-developer on the Angular team, his goal was to create a highly flexible library for components in web applications.

Comparable to jQuery, components can be implemented in the simplest case. Components allow isolated content styling. Components allow applications to be composed from individual modules.

Vue has grown further in version 3 and is now a mature framework thanks to recommended extensions. With create-vue, a new project is quickly set up.

Even though Vue supports different styles for implementation, the single-file component together with TypeScript is the recommended variant for more extensive, professional applications.

Thanks to Virtual DOM, Vue is extremely fast and lightweight. But at the same time, only actual updates reach the browser DOM.

Components are subject to a life cycle with extensive options to respond to condition changes.

Single File Components (SFC)

Indicator of the **Composition API**

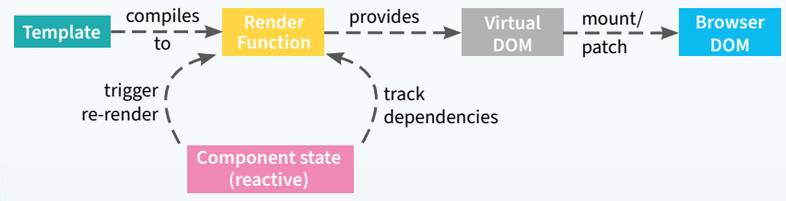
```
<script setup lang="ts">
</script>

<template>
</template>

<style scoped>
</style>
```

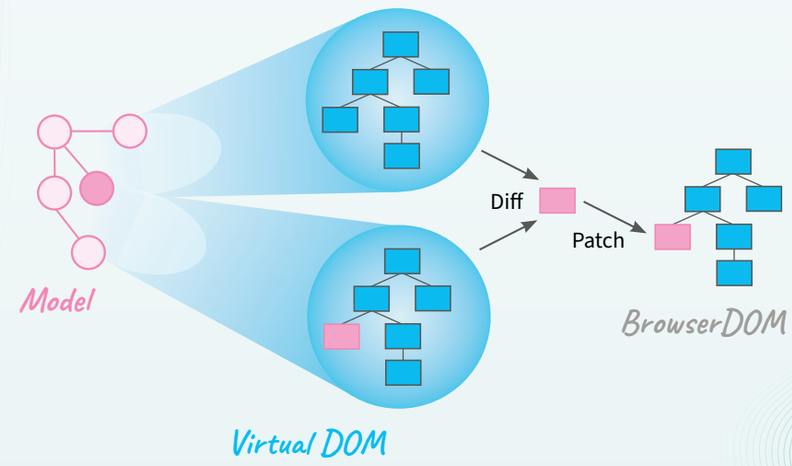
Setup Script Area:
- Imports additional components
- Defines variables, props, emits, ...

HTML template, has access to variables from the script

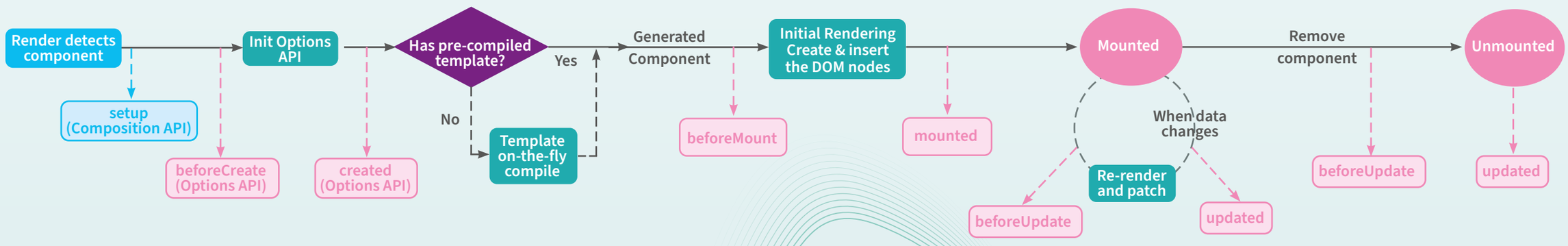
Allows CSS style (also SCSS etc. configurable)
"scoped": Styles only work in this component

Virtual DOM





Component lifecycle



Conditional Rendering

```
<div>
  <span v-if="message">Message: {{ message }}
</span>
  <p v-else-if="error">{{error}}</p>
  <div v-else>Loading</div>
</div>
```

```
<div v-show="isVisible">
  I am there - but you don't always see me.
</div>
```

Repeated rendering

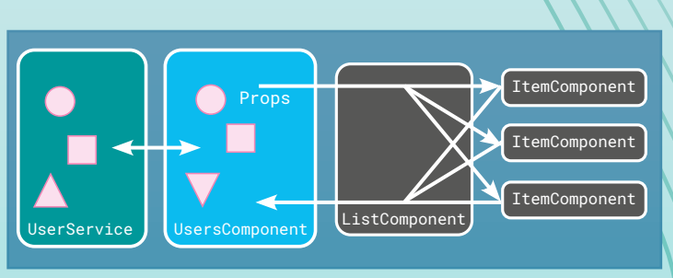
```
<ul>
  <li v-for="item in items" :key="item.id">
    {{ item }}
  </li>
</ul>
```

Connecting other components

```
<script setup lang="ts">
import FirstSample from './components/FirstSample.vue'
import SecondDemo from './components/SecondDemo.vue'
</script>

<template>
  <first-sample></first-sample>
  <SecondDemo />
</template>
```

import names in the **CamelCase** are used for referencing both in the **CamelCase** and for **snake-case**

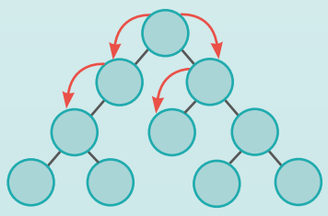
Components can be self-closing



Design: Separation of domain and view components. Domain components transform the state appropriately for reusable view components that can be reused in other contexts.

Property-Binding

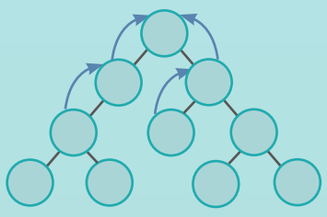
```
<template>
  <div v-bind:id="elementId">For your information: </div>

  <!-- Shorthand: omit "v-bind" -->
  <a :href="link">go to website</a>

  <!-- Binding of custom properties also possible -->
  <SecondDemo :todo-list="myList" />
</template>

<script setup lang="ts">
  import SecondDemo from "@/demo/DecondDemo.vue";

  const elementId = ref('demo-id');
  const link = ref('http://example.com');
  const myList = ref(['Clean Dishes','Mop Floor']);
</script>
```



Binding can be used to dynamically bind values to HTML attributes and to custom props. Bindings are marked with "**v-bind**" or "**:**".

Event-Handling

```
<template>
  <button v-on:click="counter += 1">Send</button>

  <!-- Shorthand: replace "v-on:" with "@". -->
  <input type="text" @click="log($event)">Send</button>

  <!-- Emits from custom events can also be handled -->
  <CountIncrementor @increment="handleChange($event)" />
</template>

<script setup lang="ts">
  import SecondDemo from "@/demo/DecondDemo.vue";

  const counter = ref(0);
  function log(event: MouseEvent): void { console.log(event); }
  function handleChange(count: number): void {
    console.log(count);
  }
</script>
```



Functions can be stored as event handlers for both DOM and custom events. Events are marked with "**v-on**" or "**@**".

Computed Properties

```
<template>
  <p v-if="todosEmpty">Success: All todos completed!</p>
</template>

<script setup lang="ts">
  const myTodos = ref(['Cook', 'Shop', 'Bake']);

  const todosEmpty = computed(() => myTodos.value?.length <= 0);
</script>
```

Will be recalculated every time **myTodos** changes

Define custom props and emits

```
<template>
  <p>Todos-Count: {{ todoList?.length }}</p>
</template>

<script setup lang="ts">
  defineProps<{ todoList?: string[] }>();
</script>
```

```
<temlate>
  <button @click="emits('increment', 42)"> Click me </button>
</template>

<script setup lang="ts">
  const emits = defineEmits<{
    (type: "increment", count: number): void
  }>();
</script>
```

Composables

```
import { ref, onMounted, onUnmounted } from 'vue'

export function useMouse() {

  const x = ref(0);
  const y = ref(0);

  function update(event: MouseEvent): void {
    x.value = event.pageX;
    y.value = event.pageY;
  }

  onMounted(() => window.addEventListener('mousemove', update));
  onUnmounted(() => window.removeEventListener('mousemove', update));
  return { x, y };
}
```

"**use**" as a characteristic of a Composable -> convention

Manage Composables **State**

...and update it.

Composables have access to **lifecycles of components** that they are integrated into

Composables should always return refs

Using the Composable in a component:

```
<script setup lang="ts">
  import { useMouse } from "@/util/mouse";
  const { x, y } = useMouse();
</script>
```


About the author

TRION

Karsten Sitterberg works as a freelance developer, trainer and consultant for web tech and Java. In his work, he is focusing on HTTP APIs, TypeScript and Angular. Karsten has a Masters degree in Physics and is an Oracle-certified Java developer. As a columnist, he writes about current trends and backgrounds of topics relevant for both developers and srchitects. He is one of the co-founders of the Frontend-Freunde Meetup series in Münster, Germany.

www.trion.de

entwickler.de

iJS International JavaScript Conference HYBRID EDITION