



DelphiDay
italian conference

INTRODUZIONE A VUE.JS

Costruire webapp orientate ai dati



LUCA MINUTI



luca.minuti.it



luca.minuti@gmail.com



dev.to/lminuti



github.com/lminuti



linkedin.com/in/luca.minuti/



DelphiDay

italian conference

20 Novembre 2024
Padova



wintech
italia

OPEN-SOURCE PROJECTS

github.com/Iminuti

WiRL

github.com/delphi-blocks/WiRL

Delphi SAML

github.com/EtheaDev/Delphi-SAML

OpenSSL

github.com/Iminuti/Delphi-OpenSSL



20 Novembre 2024
Padova









~~660,00 €~~

590,00 €

IVA ESCLUSA

 Durata	2 giornate
 Livello	Base
 Tipologia	Presenza/Online
 Aggiornato	2024

[Registrati](#)

Prossima Data

3-4 Dicembre 2024



AGENDA

1. Introduzione e confronto con altre librerie
2. Perché Vue?
3. Primi passi (Declarative Rendering e Reactivity)
4. Elementi principali (Template, Bind, Ref vs reactive, Events)
5. Creazione di componenti e gestione dello stato
6. Sguardo all'ecosistema VueJS



Introduzione

1



UN PO' DI STORIA

- Prototype.js (2005): Primo tra i framework JavaScript a introdurre utility avanzate per manipolazione DOM e Ajax
- jQuery (2006): Semplificazione della manipolazione del DOM e delle chiamate Ajax, con una sintassi breve e intuitiva
- Ext JS (2007): Framework completo per applicazioni web complesse, con componenti UI avanzati
- Backbone.js (2010): Struttura leggera per organizzare il codice in applicazioni basate su MVC



UN PO' DI STORIA

- AngularJS (2010): Framework MVC completo per creare applicazioni SPA, con supporto per il data binding bidirezionale
- Ember.js (2011): Soluzione strutturata per SPA, con convenzioni forti e strumenti di binding dati avanzati
- React (2013): Libreria per la creazione di interfacce utente tramite componenti riutilizzabili, concetto di Virtual DOM
- Vue.js (2014): Framework progressivo, modulare e reattivo, facile da integrare e curva di apprendimento più morbida di Angular e React



UN PO' DI STORIA

- Angular 2+ (2016): Versione riscritta di AngularJS con architettura a componenti, TypeScript e strumenti migliorati
- Svelte (2016): Trasforma il codice scritto in JavaScript vanilla ottimizzato al momento della build
- Alpine.js (2019): Micro-framework reattivo per interazioni minime simili a Vue o React, ma leggerissimo
- HTMX (2022): permette di costruire applicazioni web interattive senza bisogno di JavaScript



CARATTERISTICHE

- Framework per scrivere interfacce utente
 - Basato su HTML, CSS, JS; dichiarativo; component base
- Progressive Framework
 - Può essere integrato facilmente in progetti esistenti
 - Curva di apprendimento senza sorprese
 - Adatto sia per siti semplici che per intere webapp (es. SPA)
 - Server-Side Rendering (SSR), Static Site Generation (SSG)
 - Desktop, mobile, WebGL



CARATTERISTICHE

- Ecosistema
 - Librerie per ogni esigenza. Alcune ufficiali o semi ufficiali: vue-router, pinia, usevue, devtools
- Ottima documentazione
- Leggibilità e SFC
 - Il sistema di template e SFC (single file component) rende vue molto semplice da leggere



CONFRONTO CON EXTJS

Ext JS	Vue	Commento
Gestione componenti	Vue Core	Il modello di Vue è abbastanza diverso da quello di Ext JS. Anche perché in Ext JS ci sono vari modi di creare componenti. Ma quello che prevede <code>data + tpl</code> ricorda molto l'approccio di Vue.
MVVM - Data binding	Vue Core	Mentre su Ext JS il dati binding è una funzionalità accessoria su Vue rappresenta il cuore del sistema ed è quello che gli fornisce la sua natura Reactive .
MVC - Controller	Vue Core	Nel modello SFC (Single-File Components) di Vue ad ogni componente/view viene associato il codice che ne definisce il comportamento sulla base di uno stato (model).
Routing	Vue Router	Vue Router non fa parte del "core" di Vue ma è un modulo ufficiale.
Layout (vbox, fit, ...)	Tailwind	Anche in questo caso è possibile usare i CSS manualmente, ma una libreria permette di semplificare il codice usando dei pattern.



CONFRONTO CON EXTJS

Ext JS	Vue	Commento
Ext.Ajax	Axios	In alternativa è possibile usa <code>fetch</code> o <code>XmlHttpRequest</code> ma Axios fornisce: interceptor per richiesta e risposta, transformer per serializzazione e deserializzazione , timeout, blocco dell'esecuzione e altro.
Dipendenze (require)	npm, vite	Le dipendenze vengono importate, installate e configurate tramite npm . Nel codice vengono include tramite la direttiva (standard JavaScript) <code>import</code> (JS module) e vite si occupa di riconoscerle e creare la build che ne tenga conto sia in sviluppo che produzione.
Ext.data.Store	Array	Non c'è una vera e propria classe che sostituisce gli store. Si possiamo usare gli array o una qualche classe custom. Con PrimeVue alcune funzioni come filtri e ordinamenti, sono applicabili direttamente sui componenti (griglia, dataview).
Componenti (pulsanti, input, griglie, ecc.)	PrimeVue	PrimeVue è una libreria molto completa (esiste anche una versione per React e Angular)



CONFRONTO CON EXTJS

Ext JS	Vue	Commento
Calendario	PrimeVue?	Non c'è ma è previsto in Roadmap insieme a Gantt Chart, Flow Chart, Sheet, PDF Viewer, Calendar, Timeline, Editor.
Chart (istogrammi, ecc.)	PrimeVue, Chart.js	PrimeVue offre una serie nutrita di componenti che si appoggiano a ChartJS
Exporter	PrimeVue (solo CSV)	Gli store di Ext possono essere esportati in (XSL, CSV, XML, TSV, HTML, ...) mentre con PrimeVue solo la griglia permette l'esportazione di CSV.
Localizzazione	PrimeLocal e	PrimeVue offre un supporto alla localizzazione tramite oggetti reactive. Ci sono le localizzazioni già presenti per l'italiano come parte del progetto PrimeLocale .
Validazione del form	Custom	PrimeVue dalla 4.2.0 offre un supporto alla validazione compatibile con Zod, Yup, Joi, Valibot, e Superstruct



CONFRONTO CON EXTJS

Ext JS	Vue	Commento
Generazione del codice	vite o template	PrimeVue offre diversi template già fatti alcuni gratis altri a pagamenti. Oppure è possibile creare un nuovo progetto Vue con le dipendenze raccomandate tramite <code>npm create vue@latest</code>
Temi	PrimeVue	Per PrimeVue sono disponibili dei preset che permettono di configurare in maniera semplice alcuni aspetti del tema: font, colori, spaziatura.
Modelli	Custom	Probabilmente si potrebbe ottenere molto del concetto di modelli usando TypeScript . Oppure ci sono delle librerie come vue-mc o vee-validate .
Icone	PrimeIcon / custom	PrimeVue fornisce una serie di proprie icone denominate PrimeIcon . Ma è possibile integrarlo con praticamente qualsiasi set di icone. La documentazione fornisce esempi con FontAwesome, Material Icon e icone SVG.
Generazione del codice	vite o template	PrimeVue offre diversi template già fatti alcuni gratis altri a pagamenti. Oppure è possibile creare un nuovo progetto Vue con le dipendenze raccomandate tramite <code>npm create vue@latest</code>



CONFRONTO CON EXTJS

Ext JS	Vue	Commento
Utility Ext.Array, Ext.Date, Ext.Bind...	VueUse e librerie varie	Il progetto VueUse fornisce centinaia di plugin per Vue con le funzioni più disparate: State, Elements, Browser, Sensors, Network, Animation, Component, Watch, Reactivity, Array, Time, Utilities
Fashion: generazione CSS	vite/postcss	Vite, tramite postcss, si occupa della compilazione dei CSS
Transpile / Minify	vite/esbuild	Vite si occupa anche della transpilazione e compressione tramite il modulo esbuild .
Dynamic package Loading	vite/rollup	Nella build di produzione vite “spezza” il codice in una serie di chunks. È possibile personalizzare il meccanismo di creazione dei chunk tramite <code>build.rollupOptions.output.manualChunks</code> .



Primi passi

2



HELLO, WORLD!

```
<main id="app" class="container">
  <div>test: {{ count }}</div>
  <button @click="count++">
    click
  </button>
</main>
```

```
const { createApp, ref } = Vue

createApp({
  setup() {
    const count = ref(0);
    return { count }
  }
}).mount('#app')
```

demo time





SFC

- Modalità SFC (Single file component)
- In vue normalmente si lavora con dei file *.vue
 - Logica: JavaScript
 - Template: HTML
 - Aspetto: CSS
- Richiede un processo di build



SFC - VANTAGGI

- Template pre-compilati senza costi di compilazione a runtime
- CSS con scoping per singolo componente
- Sintassi più adatta per lavorare con la Composition API
- Supporto IDE con completamento automatico e controllo dei tipi per le espressioni nei template
- Supporto per Hot-Module Replacement (HMR) pronto all'uso



HELLO, WORLD!

```
<script setup>
import { ref } from 'vue';
const counter = ref(0);
</script>

<template>
  Counter: {{ counter }}
  <button @click="counter++">Increment</button>
</template>

<style setup> button { ... } </style>
```



SFC - Modalità

Esistono due modalità di scrivere SFC:

- Options API:
 - se non si usano build tool
 - per applicazioni semplici
- Composition API
 - per applicazioni scritte completamente con vue



OPTIONS

```
import { ref } from 'vue';  
export default {  
  data() {  
    return {  
      count: 0  
    }  
  },  
  methods: {  
    increment() {  
      this.count++  
    }  
  }  
};
```



COMPOSITION

```
<script>
import { ref } from 'vue';
export default {
  setup() {
    const counter = ref(0);
    function increment() {
      counter.value++;
    }
    return { counter, increment };
  }
};
</script>
```



COMPOSITION + SETUP

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

const counter = ref(0);

function increment() {
  counter.value++;
}

</script>
```




MOUNT

- Una applicazione vue parte dopo la chiamata a mount

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

```
const app = createApp(...);  
app.mount('#app')
```



TEMPLATE

- Text interpolation

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

- Attribute interpolation

```
<div v-bind:class="dynamicClass"></div>
```

```
<div :class="dynamicClass"></div>
```



TEMPLATE - Directives

- Le direttive sono gli attributi che cominciano con “v-”

```
<p v-if="seen">Now you see me</p>
```

```
<a v-on:click="doSomething"> ... </a>
```

```
<a @click="doSomething"> ... </a>
```

```
<li v-for="(item, index) in items" :key="item.id">  
  {{ text }} - {{ index }} - {{ item.message }}  
</li>
```

demo time





Reactive

3



REACTIVE STATE

- Per fare in modo che lo stato degli oggetti sia tracciato (reactive) è necessario usare le funzioni `ref` o `reactive`

```
import { ref } from 'vue'  
const count = ref(0)
```

- Quindi si accede al valore tramite `.value` (no nei template)

```
console.log(count) // { value: 0 }  
console.log(count.value) // 0  
count.value++
```




REACTIVE STATE

- Usando reactive non è necessario accedere tramite `.value` ma funziona solo con le proprietà degli oggetti

```
import { reactive } from 'vue'

const state = reactive({ count: 0 })
```

```
import { reactive } from 'vue'
const count = reactive(0) // NON FUNZIONA!
```

```
let state = reactive({ count: 0 })
state = { count: 1 } // NON FUNZIONA!
```



TWO-WAY BINDING

- Quando un componente può modificare lo stato c'è bisogno di usare il two-way binding:
 - Campi delle form (input, selection, textarea, ...)
 - Proprietà “visible” di una dialogbox
 - Lo stato di un pannello collassabile
 - Ecc.



TWO-WAY BINDING

- Esempi:

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

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



TWO-WAY BINDING

- Modificatori (es. `v-model.trim`) alterano il comportamento del binding:
 - `lazy`: sincronizza solo con l'evento `change` (normalmente con qualsiasi eventi)
 - `number`: esegue un cast a `number`
 - `trim`: elimina gli spazi



WATCHER

- Permettono di eseguire del codice quando viene modificato lo stato
- Supportano anche le promise

```
const question = ref('')  
  
watch(question, (newQuestion, oldQuestion) => {  
    ...  
})
```



COMPOSABLE

- I composabile permettono di “riciclare” la logica legata ad un stato (stateful logic) tra diversi componenti
 - Stateless logic: per esempio una funzione che formatta una data
 - Stateful logic: per esempio la posizione del mouse, lo stato dell'utente corrente



COMPOSABLE

```
// mouse.js
import { ref, onMounted, onUnmounted } from 'vue'

export function useMouse() {
  // <- Per convenzione iniziano per use
  const x = ref(0) // <- Lo stato
  const y = ref(0)
  function update(event) {
    // <- Funzione che modifica lo stato
    x.value = event.pageX
    y.value = event.pageY
  }
  onMounted(() => window.addEventListener('mousemove', update)) // <- gestione del ciclo di vita
  onUnmounted(() => window.removeEventListener('mousemove', update))

  return { x, y } // <- restituisce i dati reactive
}
```



COMPOSABLE

```
<script setup>
import { useMouse } from './mouse.js'

const { x, y } = useMouse()
</script>

<template>Mouse position is at: {{ x }}, {{ y }}</template>
```




REGISTRAZIONE

- Prima di essere usati i componenti devono essere registrati
 - Registrazione locale: in tutti i file dove vengono usati
 - Registrazione globale: su main.js
- Esistono tool e librerie che permettono la registrazione automatica (PrimeVue, Nuxt o manualmente via glob)



REGISTRAZIONE

```
// Globale
import { createApp } from 'vue'
import App from './App.vue'

import Counter from './components/Counter.vue'
import Message from './components/Message.vue';

const app = createApp(App);

app
  .component('Counter', Counter)
  .component('Message', Message)

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



REGISTRAZIONE

```
// Locale
<script setup>
import Counter from './components/Counter.vue'
</script>

<template>
  <Counter />
</template>
```



PROP

- Le prop, che permettono di configurare e interagire con il componente devono essere esplicitamente dichiarate
- È necessario il nome ma è possibile fornire più attributi alle prop:
 - type, default, required, ...



PROP

```
//////////////////// Solo nome
<script setup>
const props = defineProps(['text']);
</script>

//////////////////// Tipo
<script setup>
const props = defineProps({text: String});
</script>

//////////////////// Default e tipo
<script setup>
const props = defineProps({text: {
  type: String,
  default: 'default text'
}});
</script>
```



EVENTI

- I componenti possono emettere eventi tramite il metodo built-in `$emit`
- A gestore del metodo possono essere passati dei parametri



EVENTI

```
<template>
  <div>
    <button @click="$emit('rock')">Rock</button>
    <button @click="$emit('paper')">Paper</button>
    <button @click="$emit('scissors')">Scissors</button>
  </div>
</template>
```



V-MODEL

- Per usare v-model in caso di two-way binding deve essere configurato un modello nel componente
- Il modello può essere sia un tipo semplice che uno complesso (oggetti, array, ...)
- Il modello all'interno del componente è un “ref”, quindi deve essere usato tramite `.value`



V-MODEL

```
<script setup>

const choose = defineModel();
// choose.value = xxx;

</script>

<template>
  <div>
    <button @click="choose = 'rock'">Rock</button>
    <button @click="choose = 'paper'">Paper</button>
    <button @click="choose = 'scissors'">Scissors</button>
  </div>
</template>
```

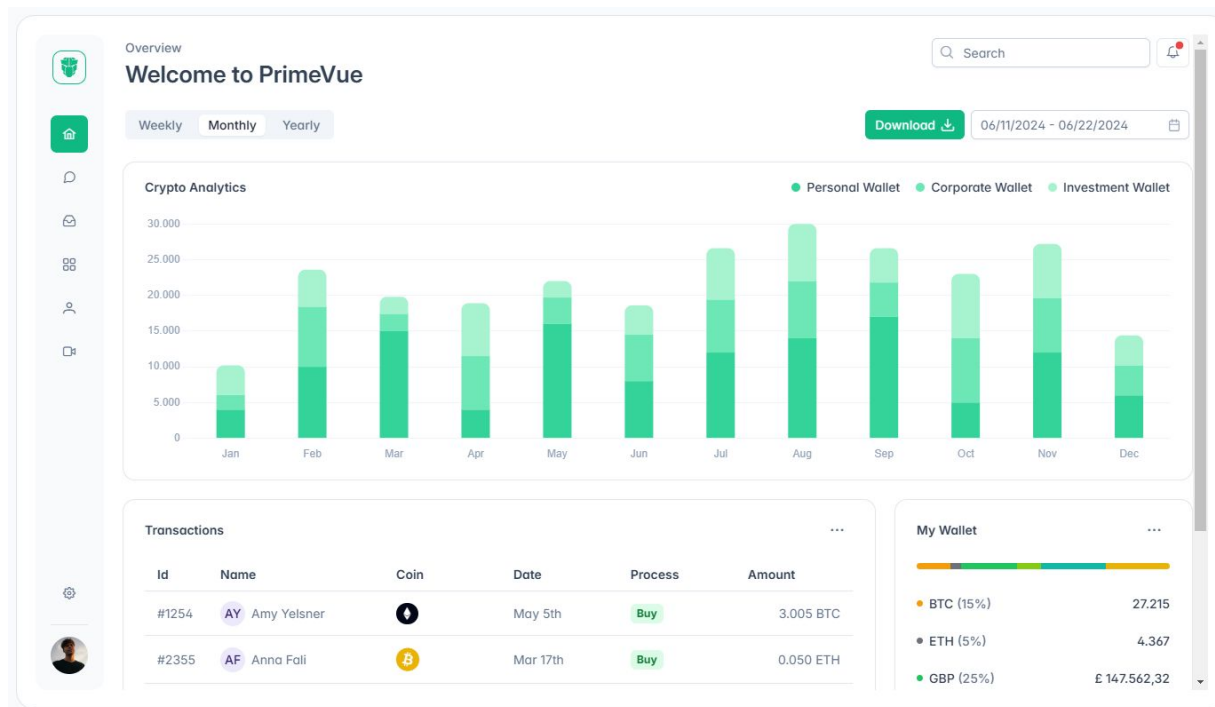


PrimeVue

4



PRIMEVUE





PRIMEVUE

- Libreria completa di componenti riutilizzabili
- Componenti complessi come griglie, treeview, chart
- Supporto flessibile per i temi (styled e unstyled)
- Accessibilità
- Template e blocchi già costruiti
- Open source (MIT/Apache)

<https://primevue.org/>





THANK YOU