



Vue.js

The Progressive JavaScript Framework

Ulteriori API Vue

Di seguito saranno approfondite alcune funzionalità di Vue:

- v-model
- Watchers
- Provide/Inject

v-model

La direttiva `v-model` può essere utilizzata per implementare una comunicazione **bidirezionale** tra i componenti. Ciò è possibile definendo contemporaneamente meccanismi di `props` (comunicazione parent-child) e `events` (comunicazione child-parent), sulla modifica di una medesima proprietà.

v-model

Nel componente **child** bisogna dichiarare sia la prop che l'evento relativo. Quest'ultimo dovrà avere la sintassi

`update:<nome-prop>`

example-01/model-example/MyChild.vue

```
<script>
export default {
  props: ['myProp'],
  emits: ['update:myProp'],
  methods: {
    onTextChange(event) {
      this.$emit('update:myProp', event.target.value.toUpperCase())
    }
  }
}
</script>

<template>
  <div>
    <input
      type="text"
      v-bind:value="myProp"
      v-on:input="onTextChange" />
    </div>
  </template>
```

v-model

Nel componente **parent** sarà possibile utilizzare utilizzare **prop** e **event** separati, attraverso le apposite direttave.

Altrimenti potrà essere utilizzata la direttiva `v-model`.

example-01/model-example/index.vue

```
<script>
export default {
  components: {
    MyChild,
  },
  data() {
    return {
      prop1ForChild: 'HELLO ',
      prop2ForChild: 'WORLD ',
    }
  }
}
</script>

<template>
  <div>
    <MyChild
      v-bind:my-prop="prop1ForChild"
      v-on:update:my-prop="newContent => prop1ForChild = newContent" />
    <MyChild
      v-model:my-prop="prop2ForChild" />
  </div>
</template>
```

Watchers

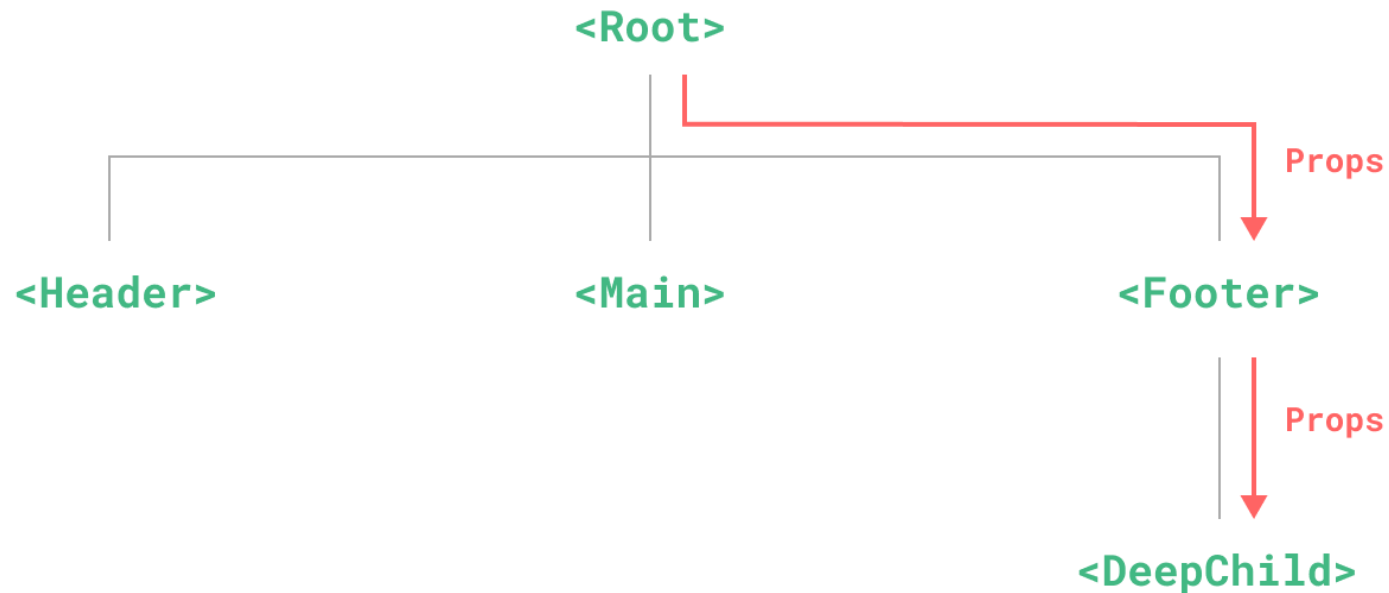
L'opzione `watch` permette di definire una computazione che verrà eseguita in reazione ai cambi di stato. Ciò abilita la realizzazione di *side-effects*.

[example-01/watchers-example/index.vue](#)

```
<script>
export default {
  data() {
    return {
      count: 0,
    }
  },
  watch: {
    count(newValue, oldValue) {
      console.log(`Changed from ${oldValue} to ${newValue}`)
    }
  },
  methods: {
    increment() {
      this.count++
    }
  }
}
</script>
```

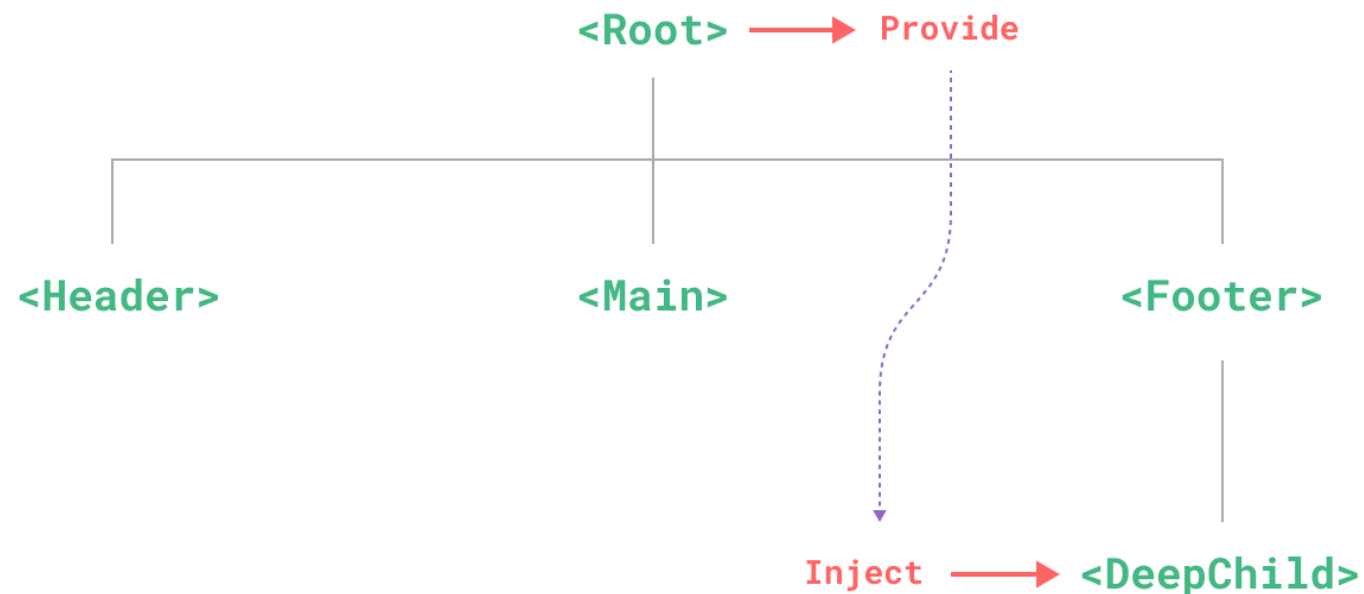
Provide/Inject

Le **props** permettono di passare i dati tra componenti, da parent a child, seguendo una discendenza diretta. Tuttavia, nel caso in cui sia necessario passare dati in profondità e i componenti nella catena non hanno interesse ad accedervi, l'utilizzo delle sole props diventa ripetitivo e dispendioso.



Provide/Inject

La funzionalità di Provide/Inject permette di evitare una catena di passaggi delle props. Il componente che mantiene lo stato, lo espone attraverso l'opzione di `provide`, mentre un qualsiasi discendente può accedervi mediante l'opzione `inject`.



Provide/Inject

Parent

Diventa provider di una proprietà

Discendente

Richiede un'ingestione della proprietà

[example-01/provide-inject-example/Grandchild.vue](#)

```
<script>
export default {
  inject: ['sharedState'],
};
</script>
```

[example-01/provide-inject-example/Parent.vue](#)

```
<script>
export default {
  data() {
    return {
      state: {
        count: 0
      }
    }
  },
  provide() {
    return {
      sharedState: this.state,
    };
  },
};
</script>
```

Options API

- Fino ad ora, abbiamo sempre creato componenti Vue utilizzando la sintassi fornita dalle **Options API**
- Il nome deriva dal fatto che i componenti Vue creati con le Options API sono appunto divisi in una serie di opzioni: data, computed, methods, ecc.
- Tuttavia, a partire dalla versione 3, Vue mette a disposizione un nuovo metodo di scrivere componenti, che prende il nome di **Composition API**
- Qual è il motivo di questa scelta?

Options API - Svantaggi

- Le Options API, per quanto efficaci, hanno alcuni svantaggi, in particolare:
 - Non forniscono molta flessibilità nell'organizzazione del codice all'interno di un componente
 - Rendono difficile condividere porzioni di codice tra componenti diversi
 - Possono risultare verbose
 - Non hanno un ottimo supporto a TypeScript
- Di seguito alcuni esempi

Options API - Organizzazione del codice

- Il contenuto dei componenti è raggruppabile solo *by type* e non *by concern*

```
import { defineComponent } from "vue"
export default defineComponent({
  data() {
    return {
      profileImage: "https://my-site.com/my-profile-image",
      name: "John",
      surname: "Doe"
    }
  },
  computed: {
    displayName() {
      return this.name + " " + this.surname
    }
  },
  methods: {
    onImgError() {
      this.profileImage = "https://my-site.com/an-error-image"
    }
  }
})
```

Options API - Riutilizzo del codice

- Condividere porzioni di codice stateful tra componenti può risultare complesso. Le opzioni sono:
 - Mixin: efficaci, ma con supporto limitato a TypeScript, e sono sconsigliati nella documentazione di Vue 3
 - State manager, come Pinia, che richiedono però un certo setup

Options API - Boilerplate

- I componenti contengono spesso varie righe di boilerplate

```
import { defineComponent } from "vue"
export default defineComponent({
  data() {
    return {
      profileImage: "https://my-site.com/my-profile-image",
      name: "John",
      surname: "Doe"
    }
  },
  computed: {
    displayName() {
      return this.name + " " + this.surname
    }
  },
  methods: {
    onImgError() {
      this.profileImage = "https://my-site.com/an-error-image"
    }
  }
})
```

Composition API

- L'idea alla base delle composition API è di vedere i componenti Vue come dei builder che producono un'interfaccia reattiva
- Ogni componente è dotato di una funzione `setup`, che viene eseguita una sola volta (per istanza del componente), creando e restituendo tutto il necessario per il suo funzionamento, prima che il componente venga inserito nel DOM (mounted)
- Le Composition API offrono una sintassi alternativa a quella delle Options API per ogni operazione di cui un componente Vue può necessitare
 - Il linguaggio di templating, e quindi il contenuto del tag `<template>` dei componenti, rimane invariato tra le due API

Options API - Esempio

- Consideriamo un componente Counter scritto con le Options API

```
<script>
import { defineComponent } from "vue"

export default defineComponent({
  data() {
    return { count: 0 }
  },
  computed: {
    doubleCount() { return this.count * 2 }
  },
  methods: {
    increment() { this.count++ }
  }
})
</script>

<template>
  <button @click="increment">{{count}}, {{doubleCount}}</button>
</template>
```


Composition API - Esempio

- Questo è lo stesso componente, scritto con le Composition API

```
<script>
import { computed, defineComponent, ref } from "vue"

export default defineComponent({
  setup() {
    const count = ref(0)
    const doubleCount = computed(() => count.value * 2)
    const increment = () => count.value++
    return { count, doubleCount, increment }
  }
})
</script>

<template>
  <button @click="increment">{{count}}, {{doubleCount}}</button>
</template>
```

script setup

- Come si può notare dall'esempio precedente, con le Composition API quasi tutta la logica del componente è gestita all'interno della funzione `setup`
 - Fanno eccezione `name`, `props`, `emits` e alcune altre opzioni che sono ancora necessarie
- **script setup** è uno zucchero sintattico introdotto di seguito alle Composition API, che permette di trasformare l'intero tag `<script>` di un componente nel corpo della funzione `setup`
 - Non è necessario fare `return` di un oggetto, tutte le variabili sono direttamente accessibili al componente
 - Non serve registrare i componenti, basta importarli
 - `name` non è necessario, Vue deduce il nome del componente dal nome del file
 - Tutte le opzioni ancora necessarie sono sostituite da apposite funzioni (vedi slide successive)

Composition API + script setup - Esempio

- E di nuovo lo stesso componente, combinando Composition API e script setup

```
<script setup>
import { computed, ref } from "vue"

const count = ref(0)
const doubleCount = computed(() => count.value * 2)
const increment = () => count.value++
</script>

<template>
  <button @click="increment">{{count}}, {{doubleCount}}</button>
</template>
```

Composition API

- Le funzionalità offerte dalle Composition API possono essere classificate in tre categorie
 - Reactivity API
 - Lifecycle hooks
 - Comunicazione tra componenti

Composition API - Reactivity

- Avete mai provato a modificare una variabile dichiarata fuori dalla funzione `data`? Funziona, ma il cambiamento non viene propagato al DOM, nè a eventuali computed properties o watcher che utilizzano quella variabile
- Questo perchè restituire una variabile dalla funzione `data` la rende **reattiva**: ogni suo cambiamento viene propagato al DOM e ad eventuali computed e watcher
- Con le Composition API, questo compito è affidato alle funzioni `reactive` e `ref`

Composition API - Reactivity

- Options API - data

```
data() {  
  return {  
    name: "John",  
    surname: "Doe"  
  }  
}  
// ...  
this.name = "Jane"  
this.surname = "Dean"
```

- Composition API - reactive o ref


```
const name = ref("John")  
const surname = ref("Doe")  
// ...  
name.value = "Jane"  
surname.value = "Dean"
```

```
const data = reactive({  
  name: "John",  
  surname: "Doe"  
})  
// ...  
data.name = "Jane"  
data.surname = "Dean"  
// ...  
data = { // Errore! È vietato riassegnare data  
  name: "Jane",  
  surname: "Dean"  
}
```

Composition API - Reactivity

- Options API - computed
- Composition API - computed

```
computed: {  
  fullName() {  
    return this.name + " " + this.surname  
  }  
}  
}
```

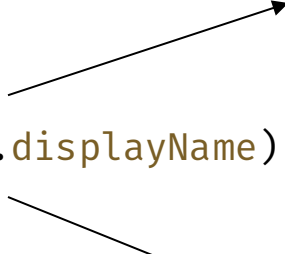


```
const fullName =  
  computed(() => name.value + " " + surname.value)
```

Composition API - Reactivity

- Options API - watch

```
watch: {  
  displayName() {  
    console  
      .log("Your new name is", this.displayName)  
  }  
}
```



- Composition API - watch o
watchEffect

```
watch(displayName, () =>  
  console.log("Your new name is", displayName.value))
```

```
watchEffect(() =>  
  console.log("Your new name is", displayName.value))
```


Composition API - Funzioni

- Options API - methods
- Composition API - una semplice funzione

```
methods: {  
  onImgError() {  
    this.profileImage =  
      "https://my-site.com/an-error-image"  
  }  
}
```

→

```
const onImgError = () => profileImage.value =  
  "https://my-site.com/an-error-image"
```

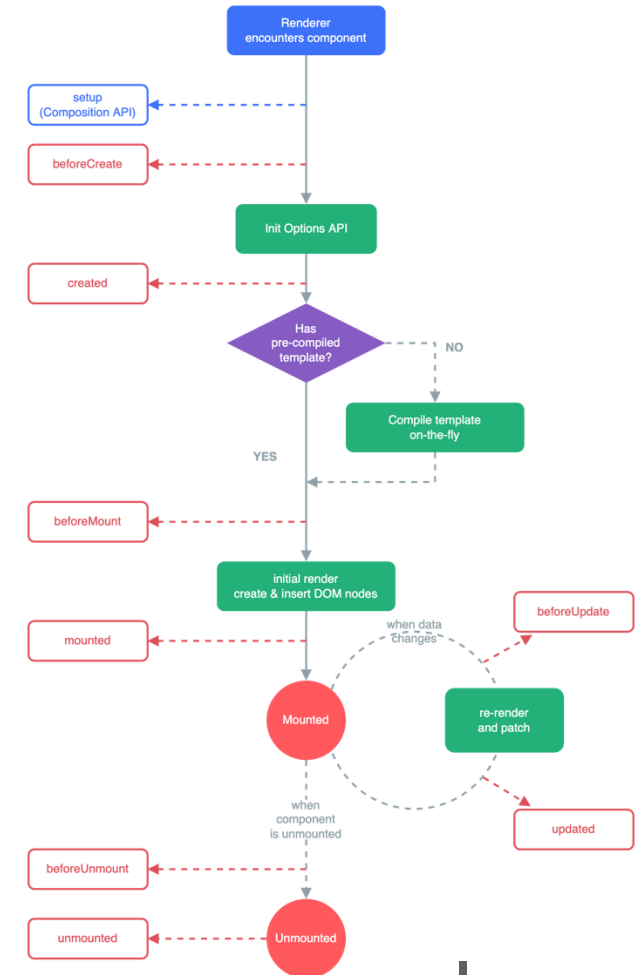
Composition API - Lifecycle Hooks

- Tutti i lifecycle hook accessibili tramite le Options API sono disponibile come funzioni nelle Composition API attraverso il prefisso on
- Esempio: mounted

```
mounted() {  
  this.getMovies()  
}
```

→


```
onMounted(() => getMovies())
```



Composition API - Props and emits

- Options API – props and emit
- Composition API – defineProps and defineEmits

```
props: ["name", "surname"],  
emits: ["click"]
```



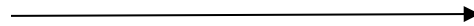
```
const props = defineProps(["name", "surname"])  
const emit = defineEmits(["click"])
```

Composition API - provide/inject

- Options API – provide ed inject
- Composition API – provide() e inject()

```
provide() {  
  return {  
    name: this.name  
  }  
}
```

```
inject: ["name"]
```

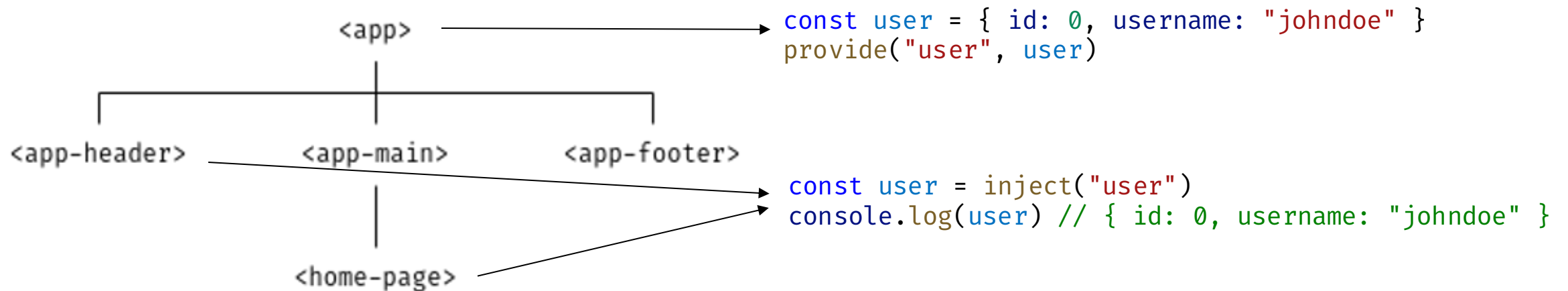


```
const name = ref("John")  
provide("name", name)
```

```
const injectedName = inject("name")
```

Composition API - Dependency injection

- Provide/Inject
- **Attenzione:** le props vanno comunque preferite per la comunicazione diretta padre-figlio



Composition API - Vantaggi

- Riepilogando, grazie alle composition API è possibile:
 - Ottenere un'organizzazione flessibile del codice all'interno dei componenti
 - Condividere facilmente porzioni di codice tra componenti diversi
 - Avere una sintassi chiara e concisa per la definizione della logica dei componenti
 - Ottenere un miglior supporto da parte di TypeScript

Composition API - Organizzazione del codice

- È possibile raggruppare il codice come meglio preferiamo (attenzione, maggior libertà non è necessariamente un vantaggio)

```
import { defineComponent } from "vue"
export default defineComponent({
  data() {
    return {
      profileImage: "https://my-site.com/my-profile-image",
      name: "John",
      surname: "Doe"
    }
  },
  computed: {
    displayName() {
      return this.name + " " + this.surname
    }
  },
  methods: {
    onImgError() {
      this.profileImage = "https://my-site.com/an-error-image"
    }
  }
})
```

```
import { computed, ref } from "vue"

const name = ref("John")
const surname = ref("Doe")
const displayName = computed(() => name.value + " " + surname.value)

const profileImage = ref("https://my-site.com/my-profile-image")
const onImgError =
  () => profileImage.value = "https://my-site.com/an-error-image"
```

Composition API - Riutilizzo del codice

- Qualsiasi variabile o funzione è facilmente condivisibile ai componenti di una gerarchia tramite provide/inject
- O, ancora più semplice, in certi casi è sufficiente importarle direttamente dove servono

```
// lib/user.js
```

```
import axios from "axios"
import { ref } from "vue"

export const user = ref(null)
export const getUser = async () => {
  user.value = (await axios.get("/api/user")).data
}
```

```
<!-- components/Cmp2.vue -->
```

```
<script setup>
import { user } from "../lib/user"
</script>
```

```
<template>
<div>{{ user?.username }}</div>
</template>
```

```
<!-- components/Cmp1.vue -->
```

```
<script setup>
import { onMounted } from "vue"
import { user, getUser } from "../lib/user"

onMounted(getUser)
</script>
```

```
<template>
<div v-if="user">{{ user.username }}</div>
</template>
```


Composition API - Boilerplate

```
<script>
import { defineComponent } from "vue"

export default defineComponent({
  data() {
    return { count: 0 }
  },
  computed: {
    doubleCount() { return this.count * 2 }
  },
  methods: {
    increment() { this.count++ }
  }
})
</script>

<template>
  <button @click="increment">
    {{count}}, {{doubleCount}}
  </button>
</template>
```

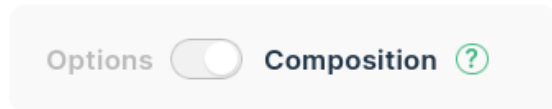
```
<script setup>
import { computed, ref } from "vue"

const count = ref(0)
const doubleCount = computed(() => count.value * 2)
const increment = () => count.value++
</script>

<template>
  <button @click="increment">
    {{count}}, {{doubleCount}}
  </button>
</template>
```

Options API vs. Composition API

- Le Options API offrono una sintassi più semplice per chi si avvicina per la prima volta a un framework frontend
- I benefici delle Composition API sono più evidenti in progetti di medie/grandi dimensioni
- Ogni snippet di codice della documentazione di Vue è consultabile in entrambe le API



- Entrambe sono valide scelte, meglio utilizzare la versione con cui si è più confidenti!

Composition API vs. React Hooks

- Le primitive offerte dalle Composition API sono molto simili a quelle dei React Hooks, semplificando per gli sviluppatori il passaggio da un framework all'altro

```
<script setup>
import { computed, ref } from "vue"

const count = ref(0)
const doubleCount = computed(() => count.value * 2)
const increment = () => count.value++
</script>

<template>
  <button @click="increment">
    {{count}}, {{doubleCount}}
  </button>
</template>
```

```
import { useMemo, useState } from "react"

export default function Counter() {
  const [count, setCount] = useState(0)
  const doubleCount = useMemo(() => count * 2)
  const increment = () => setCount(count + 1)

  return (
    <button onClick={increment}>
      {count}, {doubleCount}
    </button>
  )
}
```

Esercizio 1

- Convertire l'esercizio del laboratorio precedente dalle **Options API** alle **Composition API** con script setup
- Il materiale è fornito nella cartella `exercises/exercises-01`

Esercizio 1

- Posizionarsi nella cartella
 - `cd exercises/exercise-01/`
- Installare le dipendenze
 - `npm install`
- Eserguire il Progetto in modalità sviluppo
 - `npm run dev`

Esercizio 2

Realizzare una web app come mostrato nelle figure in seguito, utilizzando le **Composition API** con script setup.

La web app contiene una navbar con link cliccabili per cambiare il contenuto della pagina corrente. Si vuole mantenere il numero di click effettuati su ogni link e il totale dei click.

Le pagine sono da realizzare sono **Home** e **Movies**.

Esercizio 2 (Home)

La pagina **Home** mostra, in grande, i dati relativi al miglior film, ottenibili dall'endpoint `http://localhost:3000/movies/best`.

In basso, sono presenti 3 MiniCard che mostrano dati relativi a film a scelta (a partire da id fissi).

Esercizio 2 (Home)

Home (Count: 0) Movies (Count: 0)

Total click: 0

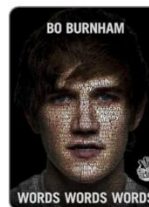
Toy Story 3

The toys are mistakenly delivered to a day-care center instead of the attic right before Andy leaves for college, and it's up to Woody to convince the other toys that they weren't abandoned and to return home.



Toy Story 3

The toys are mistakenly delivered to a day-care center instead of the attic right before Andy leaves for college, and it's up to Woody to convince the other toys that they weren't abandoned



Bo Burnham: Words, Words, Words

The internet (and soon to be movie, TV, radio, etc.) phenomenon, Bo Burnham, brings you his first one-hour stand-up special "Bo Burnham: Words. Words.



Bu jian bu san


Two people from Beijing working in Los Angeles whose paths constantly cross in a foreign city attempt to balance work with friendship/love in spite of mounting disasters

Esercizio 2 (Movies)

La pagina **Movies** mostra l'elenco di tutti i film presenti nel database e i relativi dati.

Home (Count: 2) Movies (Count: 3)

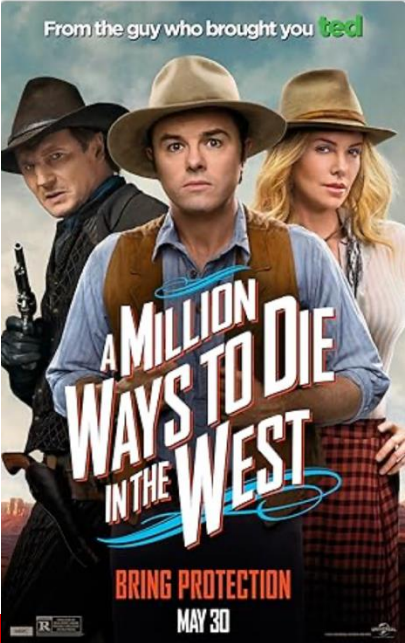
Total click: 5



Once Upon a Time in the West
8.6/10

Epic story of a mysterious stranger with a harmonica who joins forces with a notorious desperado to protect a beautiful widow from a ruthless assassin working for the railroad.

Release date: 1968-12-21



A Million Ways to Die in the West
6.1/10

As a cowardly farmer begins to fall for the mysterious new woman in town, he must put his new-found courage to the test when her husband, a notorious gun-slinger, announces his arrival.

Release date: 2014-05-30

Esercizio 2 (Esecuzione)

- Esegui il database
 - `cd exercises/exercise-02/backend/db`
 - `docker compose up`
- Esegui il backend
 - `cd exercises/exercise-02/backend`
 - `npm install`
 - `node .`
- Esegui il frontend in modalità sviluppo
 - `cd exercises/exercise-02/frontend`
 - `npm install`
 - `npm run dev`

Esercizio 2 (Suggerimenti)

- In `starting-point.html` è disponibile il materiale di partenza
- Lavorare esclusivamente sul **frontend**, cartella `src`
- Alle immagini è necessario cambiare l'endpoint.
 - `http://ia.media-imdb.com/` → `https://m.media-amazon.com/`
- Non tutte le immagini sono disponibili: utilizzare quella di default

Esercizio 2 (Challenges)

- Aggiungere una pagina per l'inserimento di nuovi film
- Permettere l'ordinamento dei film nella pagina Movies
- Effettuare paging dei film nella pagina Movies

Riferimenti

- <https://vuejs.org/guide/extras/composition-api-faq.html>
- <https://vuejs.org/api/sfc-script-setup.html>
- <https://vuejs.org/api/reactivity-core.html>
- <https://vuejs.org/api/composition-api-lifecycle.html>
- <https://vuejs.org/api/composition-api-dependency-injection.html>