The Progressive JavaScript Framework

Components

Vue.js

Un’istanza minimale di Vue.js comprende un **template**, all’interno del quale possono essere utilizzare le direttive e la definizione dello **stato reattivo**, utilizzabile nelle sue proprietà (methods, computed, etc.).

```
<div id="app">
    <h1>Hello World {{ message }}</h1>
</div>

<script>
    Vue.createApp({
        data() {
            return {
                message: 'from Vue',
            }
        },
    }).mount('#app');
</script>
```

example-01/index.html

Vue.js

Il template di un'istanza può essere specificato sia all'interno del **DOM**, che mediante **proprietà** dell'istanza (il valore della proprietà ha precedenza).

```
<div id="app">
    <h1>Hello World {{ message }}</h1>
</div>

<script>
    Vue.createApp({
        data() {
            return {
                message: 'from Vue',
            }
        },
    }).mount('#app');
</script>
```

example-01/index.html

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

<script>
    Vue.createApp({
        data() {
            return {
                message: 'from Vue',
            }
        },
        template: `
            <h1>Hello World {{ message }}</h1>`
    }).mount('#app');
</script>
```

example-01/index-template.html

Vue.js

Inoltre, è possibile definire più istanze di Vue. Istanze differenti **non condividono** le proprietà e devono essere installate su **container diversi**.

```
<div id="app1"><h1>Hello World {{ msg }}</h1></div>

<div id="app2"><h1>Ciao {{ msg }}</h1></div>

<script>
    Vue.createApp({
        data() { return { msg: 'from App 1',} },
    }).mount('#app1');

    Vue.createApp({
        data() { return { msg: 'da App 2', } },
    }).mount('#app2');
</script>
```

example-01/multiple-vue.html

Componenti

I **componenti** in Vue sono istanze di riutilizzabili e modulari che consentono di suddividere l'interfaccia utente in elementi più **piccoli** e **autonomi**, ognuno con il proprio stato, logica e template. Grazie ad essi, si incoraggia un'architettura basata su piccole unità funzionali, migliorando la modularità e la riusabilità del codice.

I concetti visti finora per le istanze Vue, come le **proprietà** e le **direttive**, possono essere estese ed utilizzate dai componenti.

Componenti

Esempio di un componente che permette di cliccare ed incrementare il conteggio.

Fasi necessarie:

1. **Definizione** del componente
2. **Registrazione** del componente
3. **Utilizzo** del componente

1.

```
const CounterButton = {
    data() { return { counter: 0 } },
    template: `
        <button @click="counter++">
            You clicked me: {{ counter }} times.
        </button>`
};
```

3.

```
<div id="app">
    <counter-button></counter-button>
</div>

<script>
    Vue.createApp({
        components: {
            'counter-button': CounterButton
        }
    }).mount('#app');
</script>
```

2.

example-02/base.html

Componenti

Un componente può essere utilizzato più volte e ogni istanza mantiene il proprio stato.

You clicked me: 0 times.

You clicked me: 1315051 times.

You clicked me: 42 times.

```
<div id="app">
    <counter-button></counter-button>
    <counter-button></counter-button>
    <counter-button></counter-button>
</div>

<script>
    const CounterButton = {
        data() { return { counter: 0 } },
        template: `
            <button @click="counter++">
                You clicked me: {{ counter }} times.
            </button>`
    };

    Vue.createApp({
        components: {
            'counter-button': CounterButton
        }
    }).mount('#app');
</script>
```

example-02/multiple-components.html

Componenti (registrazione)

I componenti, a loro volta, possono **registrare ed utilizzare** altri componenti. Ciò permette di realizzare unità con **singole responsabilità**, ma anche elementi che permettono **l'aggregazione** di più componenti.

```
const StrongText = {
    template: '<strong>Strong Text</strong>'
};
```

```
const CounterButton = {
    data() { return { counter: 0 } },
    template: `
        <button @click="counter++">
            You clicked me: {{ counter }} times.
        </button>`
};
```

```
const CounterButtonWithStrongText = {
    components: {
        'strong-text': StrongText,
        'counter-button': CounterButton
    },
    template: `
        <div>
            <strong-text></strong-text>
            <counter-button></counter-button>
        </div>`
};
```

example-02/sub-components.html

Componenti (registrazione)

È possibile registrare i componenti in due modalità:

- **Globale**
- **Locale**

La **registrazione globale** permette di utilizzare il componente in tutta l'applicazione, ovvero anche in ogni suo sottocomponente.

Metodo: `app.component()`

```
const StrongText = {
    template: '<strong>Strong Text</strong>'
};

const CounterButton = {
    data() { return { counter: 0 } },
    template: `
        <button @click="counter++">
            <strong-text></strong-text>
            You clicked me: {{ counter }} times.
        </button>`
};

const app = Vue.createApp({
    components: {
        'counter-button': CounterButton
    }
});
app.component('strong-text', StrongText);
app.mount('#app');
```

`example-02/global.html`

Componenti (registrazione)

La **registrazione locale** permette di utilizzare il componente solo dove viene incluso (i sottocomponenti non possono utilizzare l'elemento incluso in modalità locale).

Proprietà `components` dentro il componente stesso.

```
const StrongText = {
    template: '<strong>Strong Text</strong>'
};

const IncreaseCounterButton = {
    data() { return { counter: 0 } },
    template: `
        <button @click="counter++">
            <strong-text></strong-text>
            You clicked me: {{ counter }} times.
        </button>`,
    components: {
        'strong-text': StrongText
    }
};

// [Vue warn]: Failed to resolve component: strong-text
const DecreaseCounterButton = {
    data() { return { counter: 0 } },
    template: `
    <button @click="counter--">
        <strong-text></strong-text>
        You clicked me: {{ counter }} times.
    </button>`
};
```

example-02/local.html

Componenti (convenzioni)

La documentazione di Vue raccomanda di usare i nomi dei tag in **`PascalCase`** per i componenti, in modo da distinguerli dai tag HTML nativi. Inoltre, è possibile utilizzare i tag **self-closing** (`<tag />`).

```
const IncreaseCounterButton = {
    data() { return { counter: 0 } },
    template: `
        <button @click="counter++">
            <strong-text></strong-text>
            You clicked me: {{ counter }} times.
        </button>`,
    components: {
        'strong-text': StrongText
    }
};
```

example-02/local.html

```
const IncreaseCounterButton = {
    data() { return { counter: 0 } },
    template: `
        <button @click="counter++">
            <StrongText />
            You clicked me: {{ counter }} times.
        </button>`,
    components: {
        'StrongText': StrongText
    }
};
```

example-02/conventions.html

Componenti (convenzioni)

Al contrario, i **tag** utilizzati direttamente nel **DOM** devono rispettare le regole di parsing di HTML, i cui nomi sono **case insensitive**. Perciò si predilige una convenzione **`kebab-case`** e i tag di chiusa espliciti (`<tag></tag>`).

```
<div id="app">
    <inc-counter-button></inc-counter-button>
</div>

<script>
    const IncreaseCounterButton = { /* ... */ };
    const app = Vue.createApp({
        components: {
            'inc-counter-button': IncreaseCounterButton
        }
    });
    app.mount('#app');
</script>
```

example-02/conventions.html

Componenti (in file separati)

Definire tutti i componenti in un unico file diventa ingestibile a lungo andare. Perciò si predilige realizzare i componenti in **file separati**.

```
example-03/
├── components
│   ├── dec-button.js
│   ├── inc-button.js
│   └── strong-text.js
└── index.html
```

```
// components/strong-text.js
const StrongText = {
    template: '<strong>Strong Text</strong>'
};
```

```
// components/inc-button.js
const IncreaseCounterButton = {
    data() { return { counter: 0 } },
    template: `
        <button @click="counter++">
            <StrongText />
            You clicked me: {{ counter }} times.
        </button>`,
    components: {
        'StrongText': StrongText
    }
};
```

`example-03/components/*`

Componenti (in file separati)

L'**entrypoint** dell'applicazione include solo gli elementi di maggiore astrazione, delegando la logica interna ai componenti stessi che, a loro volta, possono utilizzare altri sotto-componenti.

Si crea così una **struttura ad albero**.

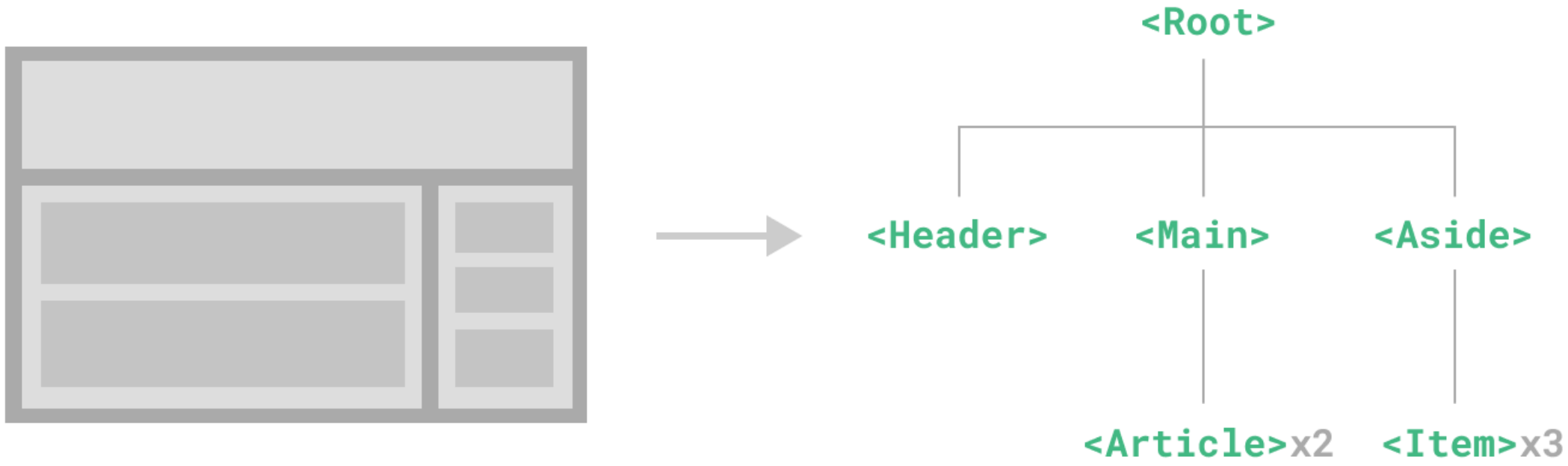
```
<script src="./components/strong-text.js"></script>
<script src="./components/inc-button.js"></script>
<script src="./components/dec-button.js"></script>

<div id="app">
    <inc-counter-button></inc-counter-button>
    <dec-counter-button></dec-counter-button>
</div>

<script>
    const app = Vue.createApp({
        components: {
            'inc-counter-button': IncreaseCounterButton,
            'dec-counter-button': DecreaseCounterButton
        }
    });
    app.mount('#app');
</script>
```

example-03/index.html

Componenti (in file separati)



Esercizio 1

Partendo dall'esercizio sui film realizzato nella lezione precedente (fornito in `exercises/exercise-01`), scomporre la struttura mediante l'utilizzo di uno o più **componenti**.

Esercizio 1

- Posizionarsi con il terminale in `exercises/exercise-01`
- Installare le dipendeze `npm install`
- Eseguire il web server `node index.js`

Esercizio 1

- Lavorare a partire dal file `www/index.html`
- I file nella cartella `static` vengono serviti staticamente

Esercizio 1

Per ogni **componente** individuato:

1. Creare un file nella cartella `static` (e.g. `component.js`)
2. Spostare le proprietà relative al componente nel file
3. Importare il componente in `index.html`
4. Registrare il componente nell'istanza Vue
5. Utilizzare il componente nel template (o nel DOM)

Esercizio 1 (suggerimento)

Esempio per componente `header`:

1. Creare il file `header.js` nella cartella `static`
2. Spostare il template da `index.html` in `header.js`
3. Importare il componente in `index.html`

```
<head>
    …
    <script src="/header.js"></script>
    …
</head>
```

Esercizio 1 (suggerimento)

4. Registrare il componente nell'istanza Vue

```
components: {
   "FilmHeader": header,
}
```

5. Utilizzare il componente nel template

```
template: `<FilmHeader />`
```

Esercizio 1 (domanda)

È possibile scomporre **ogni** parte della struttura in singoli **componenti**?

Esercizio 2 (SPA)

Partendo dal materiale fornito in `exercises/exercise-02`, realizzare una **Single Page Application** utilizzando i componenti forniti.

Esercizio 2 (componenti)

Sono stati forniti i seguenti componenti:

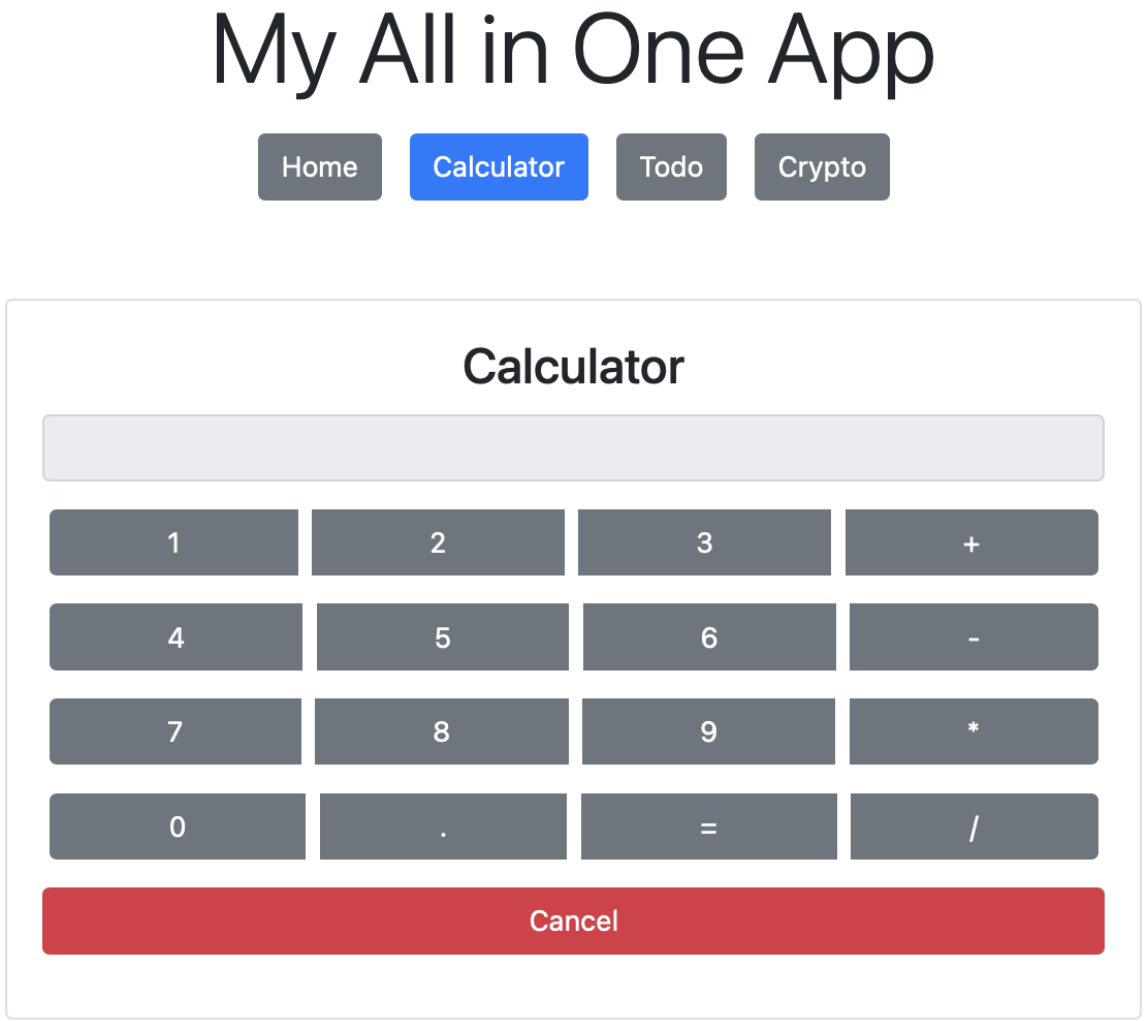
- `Homepage`
- `Calculator`
- `TodoList`
- `CryptoPrice`

Nota: ai fini dell'esercizio, non trattare la `navbar` come un componente e lasciarlo in `index.html`

Esercizio 2 (Obiettivi)

1. Implementare la logica dei componenti
2. Spostare ogni componente in un file separato
3. Mostrare solo il componente selezionato nella `navbar`

Esercizio 2


My All in One App
Home
Calculator
Todo
Crypto
Calculator
1
2
3
+
4
5
6
-
7
8
9
*
0
.
=
/
Cancel

Esercizio 2

Per **visualizzare** la pagina non è necessario l'utilizzo di `node`, dal momento l'esercizio non prevede l'utilizzo di un web server.

Aprire il file `index.html` direttamente nel browser (doppio click o terminale)

Esercizio 2 (suggerimento)

- Il componente `Home` contiene solo template.
- Il componente `Calculator` permette di scrivere espressioni e calcolarle
  - i bottoni concatenano il numero o l'operatore all'espressione
  - `Cancel` pulisce l'output video
  - il bottone = calcola il risultato

Esercizio 2 (suggerimento)

- Il componente `TodoList` permette di:
  - Aggiungere un task
  - Rimuovere un task
  - Marcare un task come completato

- Il componente `CryptoPrice` permette di:
  - Effettuare la richiesta e ricevere i valori da mostrare
  - Gestire il caricamento quando i valori non sono ancora disponibili

Esercizio 2 (domanda)

È possibile scomporre la pagina come suggerito?