Mattia 06/05/2022

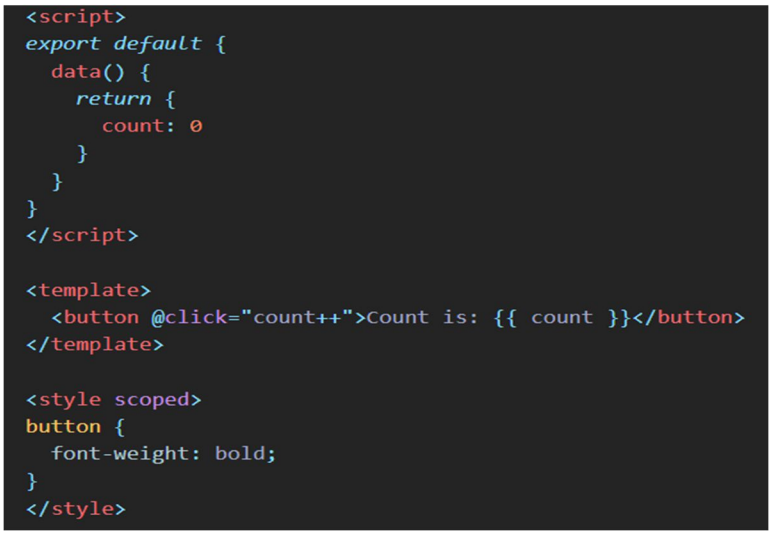
Zanini 4F

Vuejs

Vuejs, secondo me, è un framework che è molto “facile” per chi è abile nello scrivere codice funzionale, per la grafica bisogna far da sé.

Trovo davvero utile la creazione di template per suddividere le pagine in pezzi più piccoli

Dentro il tag template ci può stare dentro solo un elemento, quindi l’unico modo per sviare a questa regola è di creare un div che contenga il tutto.



In ogni pagina .vue, è possibile inserire un CSS che sarà “attivato” solo quando si visualizza la relativa pagina, per fare questo si scrive nel file della pagina il tag “<style scoped>”, inoltre con il tag “<script>”, si inserisce il codice backend della pagina (le prime 9 righe della foto di sopra), nel codice, la proprietà “data()” si occupa di contenere le variabili che si dichiarano e per separare le une dalle altre, si inserisce una virgola alla fine.

Per richiamare una variabile all’interno del codice html basta inserirlo tra 4 parentesi grafe (come nella foto di sopra), all’interno di questa parentesi si può solo invocare funzioni o utilizzare le variabili, non si può scrivere codice javascript.

```vue
<script>
export default {
  // Properties returned from data() becomes 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>
```

“Data()” non è l’unica proprietà che possiamo utilizzare in vue, c’è anche la proprietà “methods” con la quale possiamo creare le funzioni che si possono richiamare all’interno del codice html. La proprietà “mounted” è una funzione che viene eseguità quando la relativa pagina viene montata.

```vue
<div v-bind:id="dynamicId"></div>
```

La **v-bind** direttiva indica a Vue di mantenere l’attributo **id** dell'elemento sincronizzato con la variabile **dynamicId**. Se il valore associato è **null** o **undefined**, l'attributo verrà rimosso dall'elemento renderizzato.

```vue
<div class="row">
  <h1 class="title-h1">Absolute Battleship</h1>
</div>
<div v-if="show == 0" class="row d-flex justify-content-center zoomin">
  <GameMenu @hideMenu="showMenu" />
</div>
<div v-if="show == 1" class="row d-flex justify-content-center zoomin">
  <DetailGame @hideRules="showMenu" />
</div>
<div v-if="show == 2" class="row d-flex justify-content-center zoomin">
  <Grid @exitPreGame="showMenu" :rowIndicators="rIndi" :player="player" @
</div>
<div v-if="show == 3" class="row d-flex justify-content-center zoomin">
  <Game @exitGame="showMenu" :rowIndicators="rIndi" :bot="bot" :player="p
</div>
<div v-if="show == 4" class="row d-flex justify-content-center zoomin">
  <End @exit="showMenu" :winn="winner" />
</div>
```

La direttiva **v-if**, renderizza o meno gli elementi che contiene a seconda se la condizione è vera o falsa. Esiste ovviamente esistono anche **v-else** e **v-else-if**.

```
<!-- SHIPS -->
<div class="col-6 ships">
  <div class="row">
    <div class="col-1 ship-quantity">
      <h3>{{ nCarr }}x</h3>
    </div>
    <div class="col-5">
      <div class="row" id="Carrier" @click="SelectShip">
        <div v-for="shipLength in 5" :key="shipLength" cl
      </div>
    </div>
  </div>
```

La direttiva **v-on**, abbreviata come "@", ascolta gli eventi DOM, nella foto di sopra ascolta l'evento click dell'elemento div con classe row ed id "Carrier", sostanzialmente quando l'utente clicca con il cursore del mouse sull'elemento, viene eseguita la funzione "SelectShip".

```
<!-- GRID -->
<div class="col-6">
  <table class="grid">
    <tbody>
      <tr v-for="row in 11" :key="row">
        <td v-for="col in 11" :key="col" :class="[
          { 'no-border': row == 1 || col == 1 },
          { 'grid-border': row != 1 && col != 1 },
          { 'blue': CellType('-', row, col) },
          { 'busy': CellType('s', row, col) },
        ]" :id="GetPos(row, col)" @click="ClickTD">
          <!--row:{{ row }} col:{{ col }}-->
```

In Vue è possibile asegnare dinamicamente le classi ad un elemento html, come nella foto di sopra, si dichiara una classe in questa maniera (foto di sotto)

```
<div :class="{ active: isActive }"></div>
```

Secondo questa dicitura, se la condizione è vera, ovvero che la variabile active è **true**, allora al seguente div verrà assegnata la classe "isActive", altrimenti niente, o nel caso la variabile **active** dovesse passare da false a true, verrà data la classe al div, non per nulla è un modo per assegnare le classi in maniera interattiva, è possibile (come nella prima foto delle classi interattive) utilizzare le funzioni per verificare la condizione.

Mettendo una virgola alla fine della condizione, si possono concatenare le classi da assegnare dinamicamente, si può espandere la condizione in modo tale da assegnare una determinata classe

se la condizione è vera oppure un'altra classe ancora, se la condizione è falsa, in questo modo (foto di sotto, è la dicitura della condizione ternaria)

Se desideri attivare anche una classe nell'elenco in modo condizionale, puoi farlo con un'espressione ternaria:

```
template
<div :class="[isActive ? activeClass : '', errorClass]"></div>
```

Questo si applicherà sempre `errorClass`, ma `activeClass` verrà applicato solo quando `isActive` è veritiero.

Possiamo usare la **v-for** direttiva per rendere un elenco di elementi basato su un array. La v-for direttiva richiede una sintassi speciale sotto forma di item in items, dove **item** sè l'array di dati di origine ed **items** è un alias per l'elemento dell'array su cui viene ripetuto:

```
data() {
  return {
    items: [{ message: 'Foo' }, { message: 'Bar' }]
  }
}
```

```
<li v-for="item in items">
  {{ item.message }}
</li>
```

v-for può anche prendere un numero intero. In questo caso ripeterà il modello più volte, in base a un intervallo di 1...n, questa direttiva comincia da **1** e non da 0.

```
<span v-for="n in 10">{{ n }}</span>
```

Per dare a Vue un suggerimento in modo che possa tracciare l'identità di ciascun nodo, e quindi riutilizzare e riordinare gli elementi esistenti, è necessario fornire un **key** attributo univoco per ogni elemento:

```
<tr v-for="row in 11" :key="row">
  <td v-for="col in 11" :key="col" :class="[
    { 'no-border': row == 1 || col == 1 },
```

In Vue è possibile creare dei **componenti**, questi possono essere visti come dei lego che andiamo ad attaccare alla pagina principale, per crearli basta inizializzare un nuovo file .vue, successivamente bisogna importalo sulla pagina principale con questo comando nel tag **<script>**:

```
<script>
import DetailGame from "./components/VUE/DetailGame.vue";
import Grid from "./components/VUE/Grid.vue";
import GameMenu from "./components/VUE/GameMenu.vue";
import Game from "./components/VUE/Game.vue";
import End from "./components/VUE/End.vue";
```

Per poterli utilizzare poi bisogna inserirli nella proprietà **components** della pagina

```
// This function will be called when the c
mounted() {
  console.log("App Correctly mounted");
},
components: {
  DetailGame, Grid, GameMenu, Game, End,
},
```

Ogni componente può accedere, solo in lettura, alle variabili che il padre gli passa come proprietà del componente, si dichiara un array, dove vi sono i nomi delle proprietà

```
export default {
  props: ['rowIndicators', 'player'],
  data() {···
  },
```

Nella pagina principale poi gli passi le variabili che il componente **figlio** può vedere, utilizzando le variabili con i nomi delle proprietà

```
<div v-if="show == 2" class="row d-flex justify-content-center zoomin">
  <Grid @exitPreGame="showMenu" :rowIndicators="rIndi" :player="player" @update-player="updatePlayer" />
</div>
```

@exitPreGame e **@update-player** sono funzioni che la pagina principale (padre) eseguono la funzione che gli viene passata (viene passato il nome della funzione) quando il figlio gli **emette** di eseguirle, ecco un esempio:

```
<div class="col-5">
  <button class="btn btn-style" @click="$emit('exitPreGame', 0, this.RotateListener)">Esci</button>
</div>
```

nell’ **$emit** gli si passa come primo parametro la funzione da richiamare nel padre, il resto sono dei parametri classici; serve per emettere eventi personalizzati.

Nelle funzioni dichiarate nei **methods** della pagina per richiamarne un'altra bisogna usare l’espressione **this**, stessa cosa per le variabili dichiarate nel **data()** e le proprietà.

Sempre nelle funzioni dichiarate, si può utilizzare il comando **this.$emit()** per eseguire la stessa cosa di quanto parlato prima

```
CheckVictory() {
    if (!this.finished) {
        if (this.player.sunkenShips == 10) {
            console.log("YOU LOSE")
            this.finished = true
            this.winner = "bot"
        }
        else if (this.bot.sunkenShips == 10) {
            console.log("YOU WIN")
            this.finished = true
            this.winner = "player"
        }
    }
    if (this.finished) {
        this.$emit('gameFinished', this.winner)
        this.$emit('exitGame', 4)
    }
},
```