Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

SUPSI

Vue.js framework

Studente/i

Zanichelli Mattia Turturiello Timothy

Data

04/05/2022

Corso di laurea

Ingegneria informatica

Modulo

Web 2

Anno

2021/2022

Introduzione

- JavaScript framework per interfacce grafiche
- Basato su HTML, CSS e vanilla JavaScript
- Modello di programmazione dichiarativo basato su componenti
- Due varianti di API
    - Options
    - Composition

Filosofia: separazione per concetti

- Componente singolo favorisce
  - leggibilità
  - manutenibilità sistema
- Rottura separazione classica
  - Separazione classica tra file: struttura, comportamento, view
  - template -> html
  - script -> javascript, …
  - style -> css

```
<template>
    <span id="example">
        <!-- here comes the muscles... -->
    </span>
</template>


<script>
    // here comes the character...
</script>

<style>
    /* here comes the beauty... */
</style>
```

Pro e Contro di Vue.js

Pro

- Dimensione ridotte ~18 KB
  - impatto positivo su UX della applicazione frontend
- Architettura basata su componenti
- Facile da usare
- Buon supporto Babel, Webpack, librerie sistemi di routing e testing

Contro

- Problemi noti reattività del componente
  - ci sono tipi di cambiamenti che Vue deve aggirare per rilevarli (watchers, computed props, …)
- Comunità principalmente asiatica
  - Forum in cinese
  - Crescente popolarità attenua questo problema
- Diverse vie, diversi possibili approcci
  - possibile codice non coeso
  - es: Options vs Composition api

Options

- Definizione della logica dei componenti usando un oggetto come *data*, *methods* o *mounted*
- Costruito a partire dalla variante Composition
- Concetto di «istanza di un componente»
- Modello mentale basato su classi simili ai linguaggi OOP
- Organizzazione del codice tramite gruppi di opzioni

- Esempio di Option API

```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>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

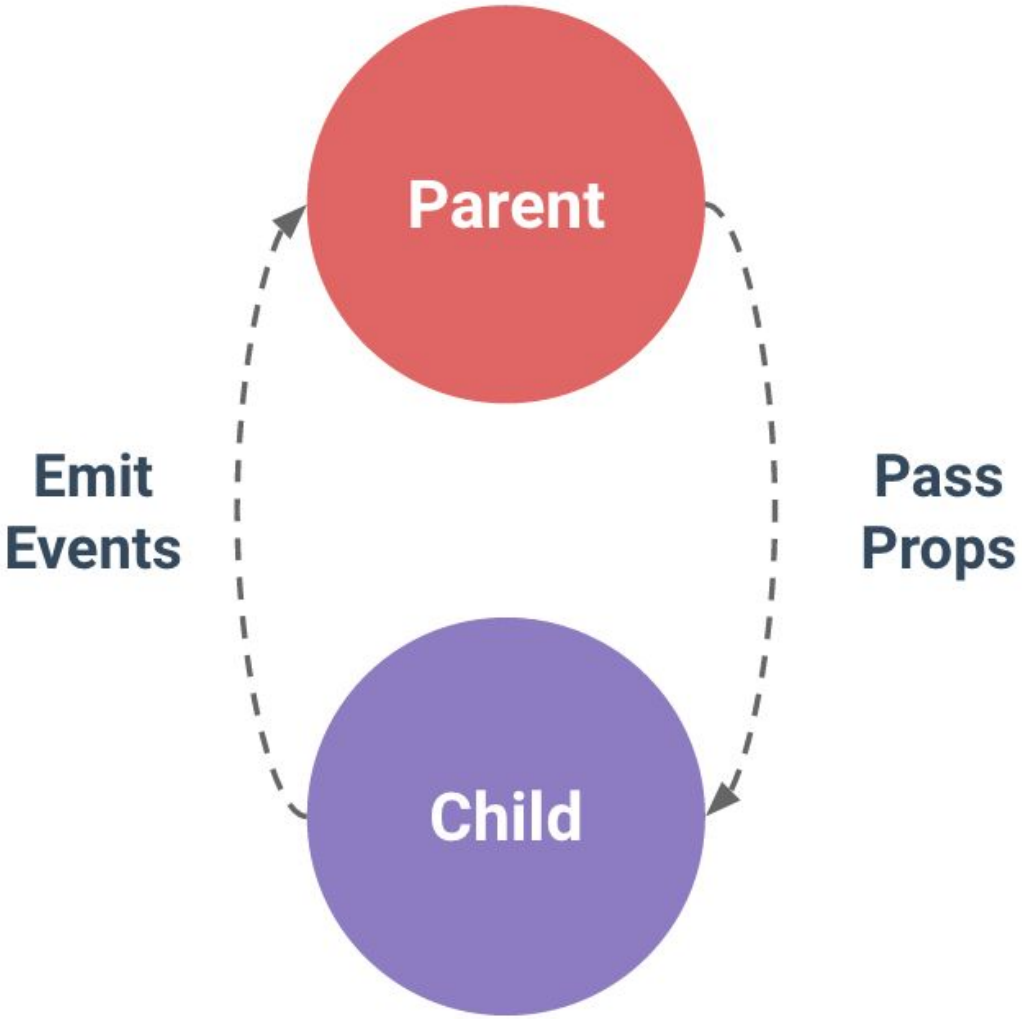
Composition

- Definizione della logica dei componenti attraverso funzioni dell'API importate
- Keyword *setup* nel tag di script per «informare» il compilatore
    - In questo modo è possibile importare variabili e funzioni da usare nel modello
- Incentrato sulla dichiarazione di variabili di stato direttamente nelle funzioni
- Forma più libera sfrutta la reattività di Vue.js in modo efficace
- Flessibilità che consente riutilizzo della logica

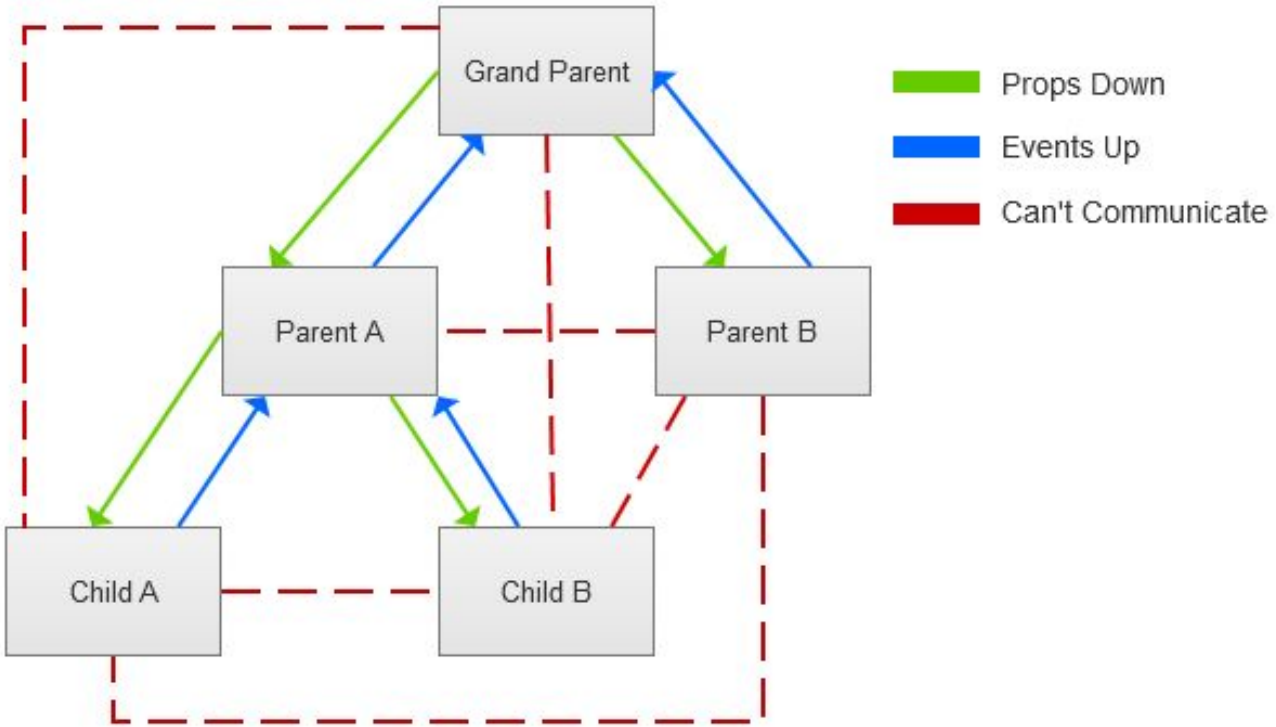
- Esempio di Composition API

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

// reactive state
const count = ref(0)

// functions that mutate state and trigger updates
function increment() {
  count.value++
}

// lifecycle hooks
onMounted(() => {
  console.log(`The initial count is ${count.value}.`)
})
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

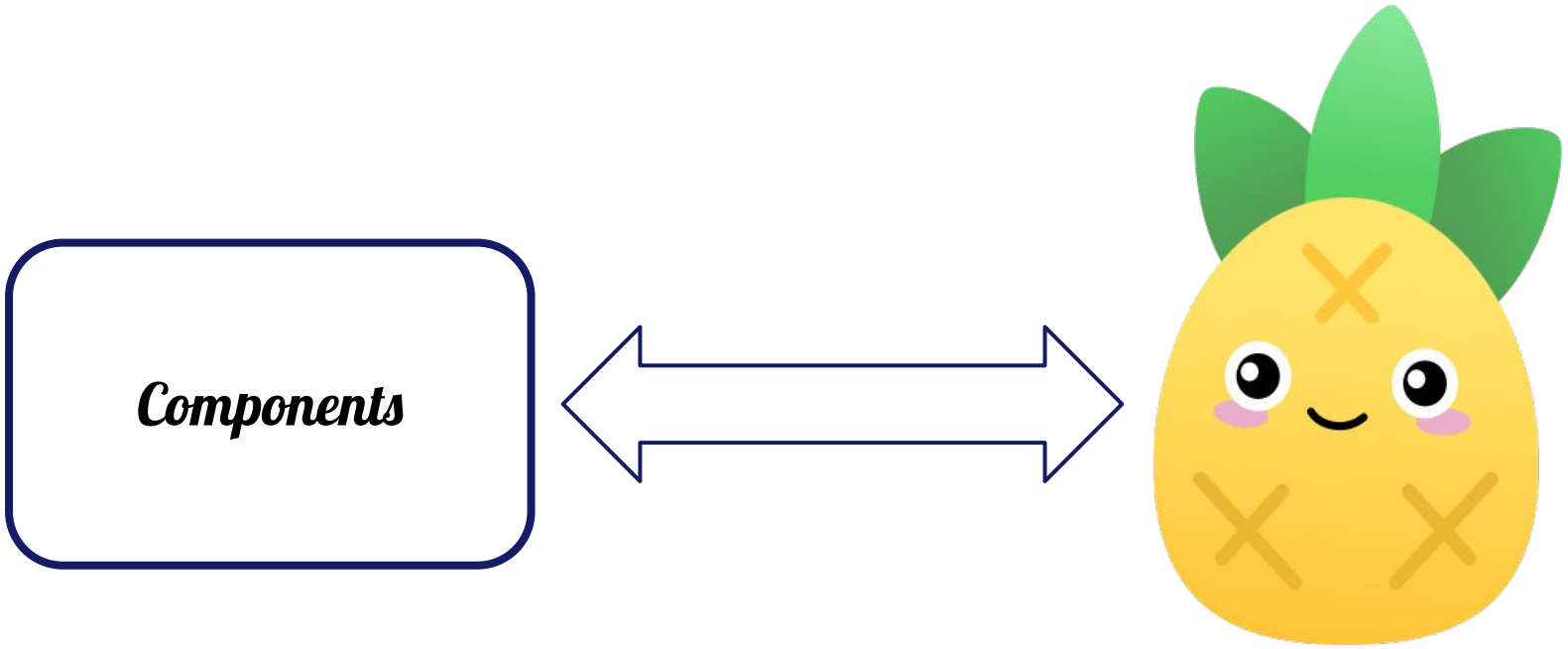
Comunicazione parent-child



Comunicazione Cross-Component

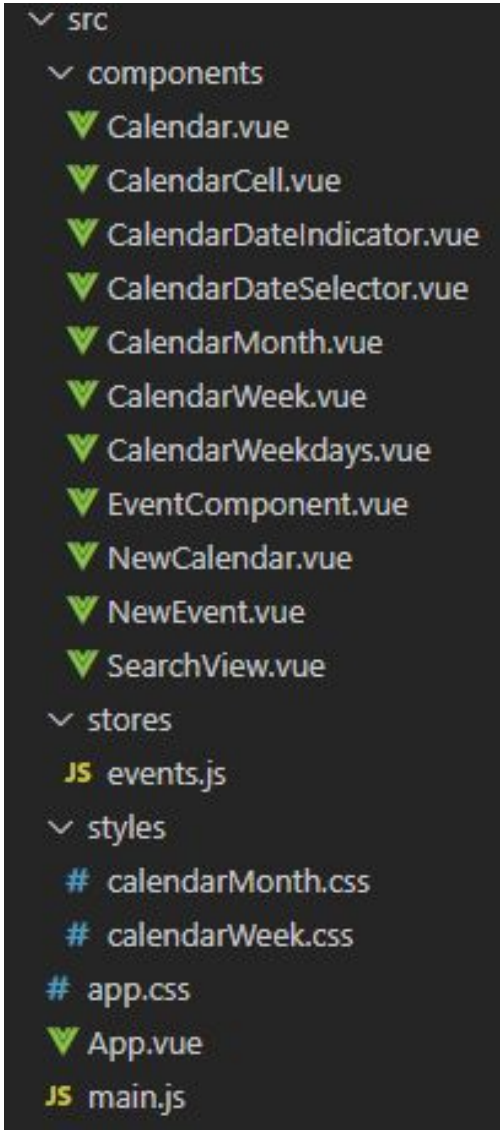


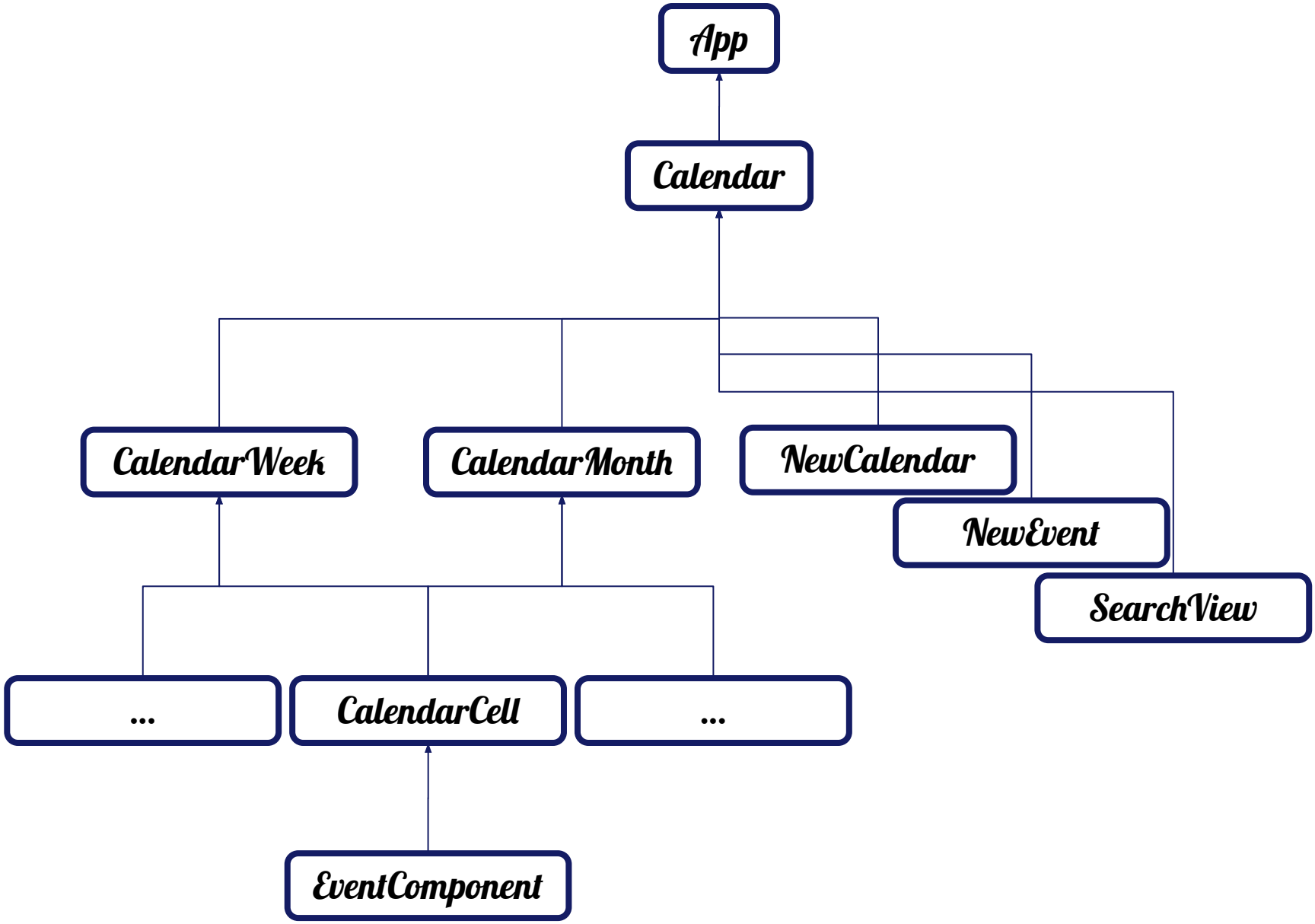
Solution: Shared Store



Calendario

- Scritto con variante Composition
- Diviso in componenti reattivi





Demo