SUPSI

Human-computer interaction

HCI 2018-2019

anno accademico 2018-2019

Nicola Rizzo - nicola.rizzo@supsi.ch

Interfacce complesse

stato della GUI

- Man mano che la complessità dell'applicazione sale, diventa complesso anche gestire lo stato dell'interfaccia

- Diversi componenti si troveranno a dover condividere stato e informazioni

- Azioni da diverse parti della GUI (moduli) si troveranno a dover modificare la stessa informazione

- diventa facile dimenticare di aggiornare l'interfaccia al seguito di un cambiamento dei dati

Vue.js

- Utilizzeremo vuejs per risolvere questi problemi
- Vue è un framework per produrre interfacce utente che
  - permette di sviluppare per componenti
  - offre funzionalità di data binding
  - è (relativamente) semplice da imparare
  - ha un ricco ecosistema di plugin (eg vue router, vuex)

componenti

- Sono istanze riusabili di Vue
- Permettono di definire nuovi tag (similmente ai custom components di html5) utilizzabili nei template
- Possono essere avere un proprio stato o essere stateless grazie alle “props” (immutabili), che permettono di propagare uno stato da componenti padri ai figli

esempio (applicazione Vue)

```
let app = new Vue({
  // state
  data () {
    return {
      count: 0
    }
  },
  // view
  template: `
    <div
v-on:click="increment">{{ count }}</div>
  `,
  // actions
  methods: {
    increment () {
      this.count++
    }
  }
})
```

Stato (meglio che sia una funzione, per non essere condiviso)

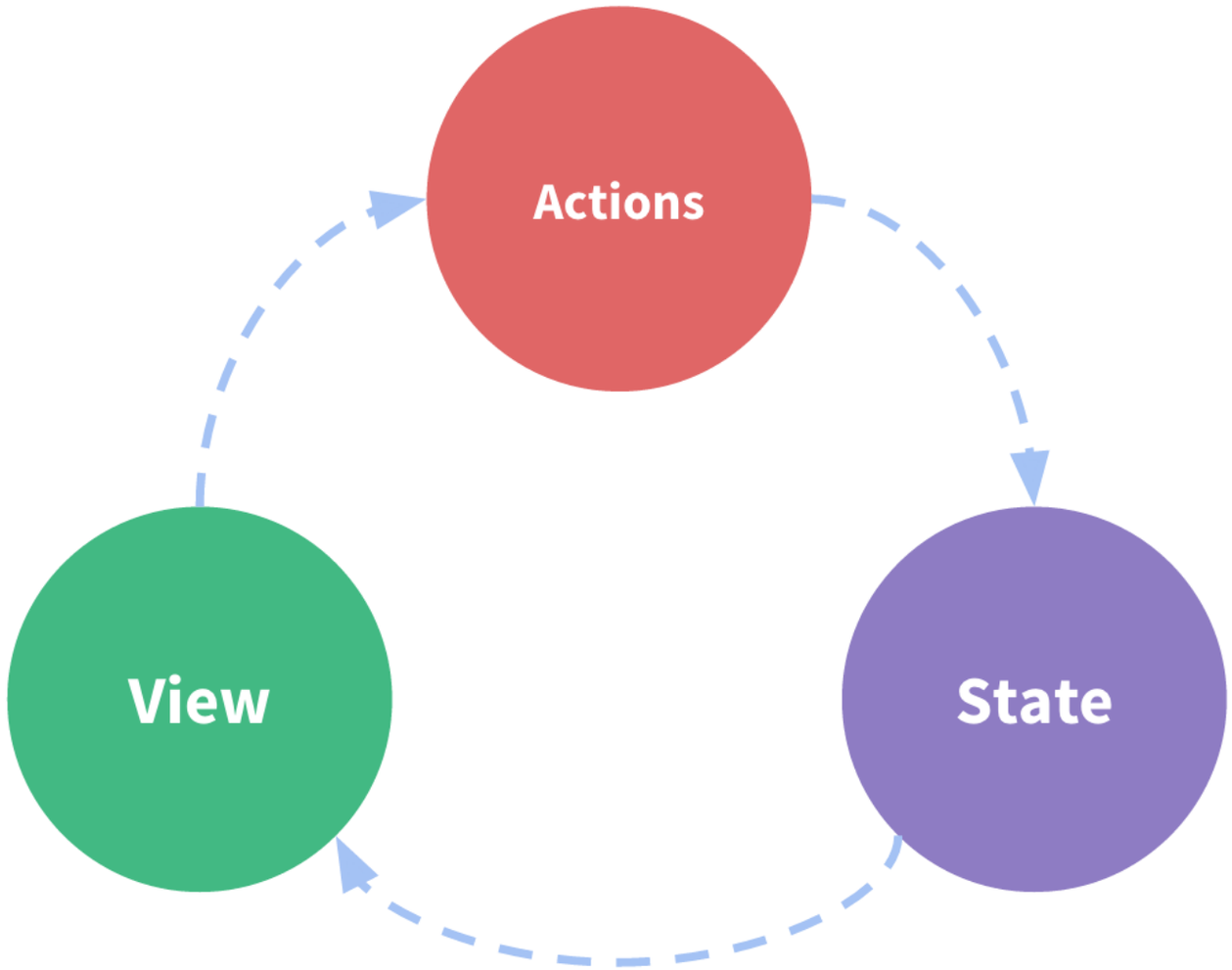
template, esprimibile anche con una altra sintassi, rappresenta la view

azioni

E la “separation of concerns”?

- Anche se la gestione dell’evento è scritta nel template, Vue garantisce che:
  - lo scope delle variabili non è globale, è tutto legato al componente corrente
  - gli handler di eventi vengono automaticamente rimossi da Vue quando serve
  - viene usata (trasparentemente) l’event delegation
  - la parte js dei componenti non è legata al DOM, quindi più “pulita” e facile da testare

One-way data flow



tuttavia

- Può diventare lungo e verboso passare in continuazione lo stato da padre in figlio (e non funziona per componenti allo stesso livello)

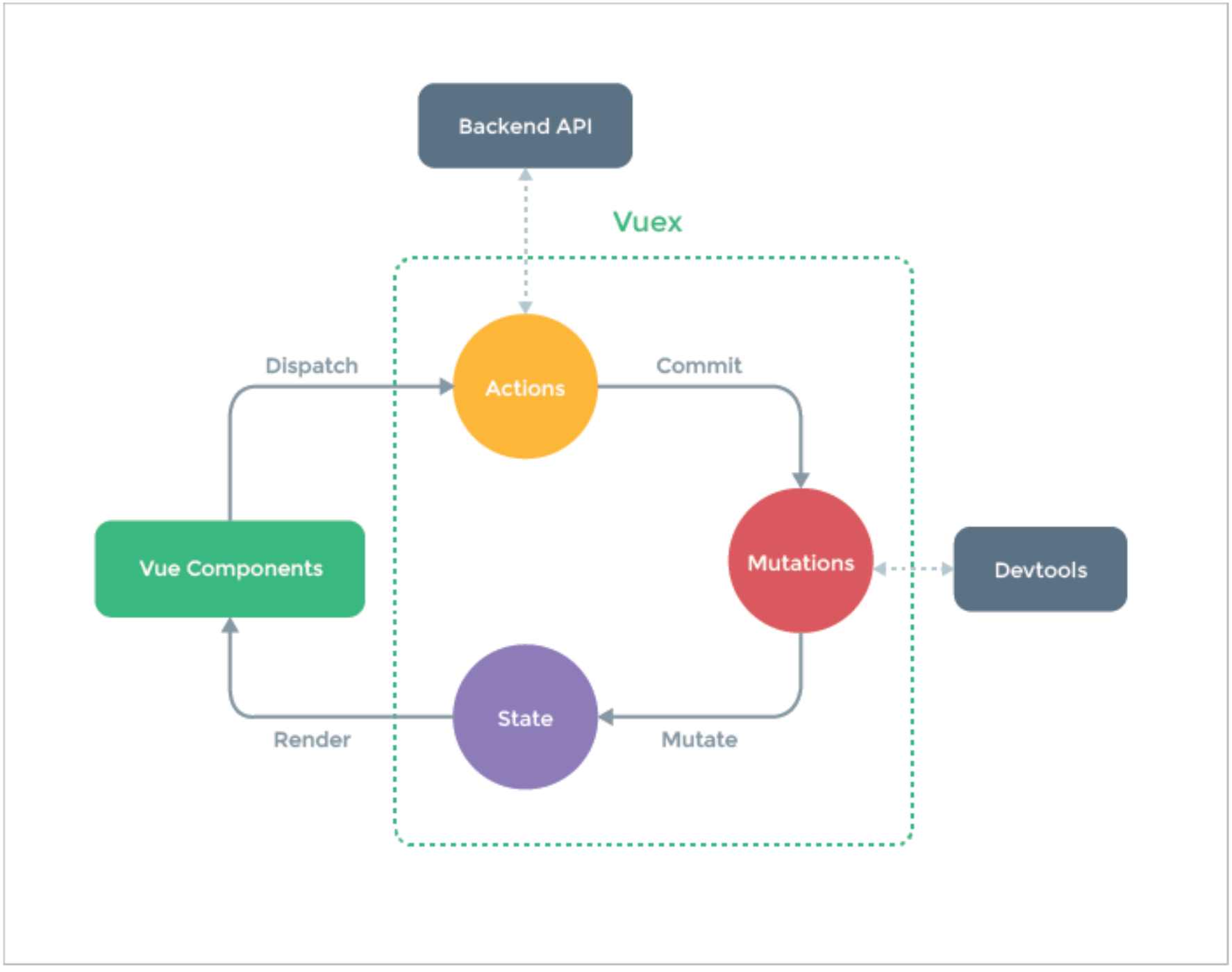
- Sincronizzare diversi componenti che condividono lo stesso stato via eventi o messaggi diventa rapidamente poco manutenibile / instabile

stato come singleton

- Una possibile soluzione è quella di **estrarre** lo stato dai componenti dalla GUI e gestirlo come un singleton, in modo che possa fare da unico valore di verità per tutta l'interfaccia

Vuex

- Vuex è definita come pattern di gestione dello stato + libreria correlata


Backend API
Vuex
Dispatch
Actions
Commit
Vue Components
Mutations
Devtools
State
Render
Mutate

Store

- Lo **store** è il contenitore che mantiene lo stato dell'applicazione
- Lo stato non può essere modificato direttamente, ma solo attraverso delle **mutations**, che possono essere tracciate e fornire varie facilitazioni (come UNDO e REDO)

State

- È l’unico punto di verità dell’applicazione.
- In Vuex è **reactive**: questo significa che basta modificare una qualsiasi parte dello stato (anche se ci sono alcune restrizioni) e l’interfaccia reagirà di conseguenza

getters

- I getter sono dei metodi dello store e possono eseguire delle elaborazioni sullo stato

- In questo modo i componenti possono estrarre l'informazione elaborata senza dover sapere come è stata calcolata e restando semplici

mutations

- Lo store contiene le mutations, che devono essere **sincrone**, per poter essere tracciabili, anche in fase di debug:
  - i tool di sviluppo devono avere uno snapshot dello stato prima e uno dopo la commit di una mutation

- Esse sono l'unico modo per modificare lo stato e possono essere invocate dai componenti o dalle actions

actions

- Anche le actions vivono all’interno dello store
- Essere permettono di richiedere la modifica dello stato o di una sua parte in maniera asincrona: un esempio può essere la chiamata a una URI remota

Link

- https://vuejs.org/

- https://cli.vuejs.org

- https://vuex.vuejs.org/