Vue

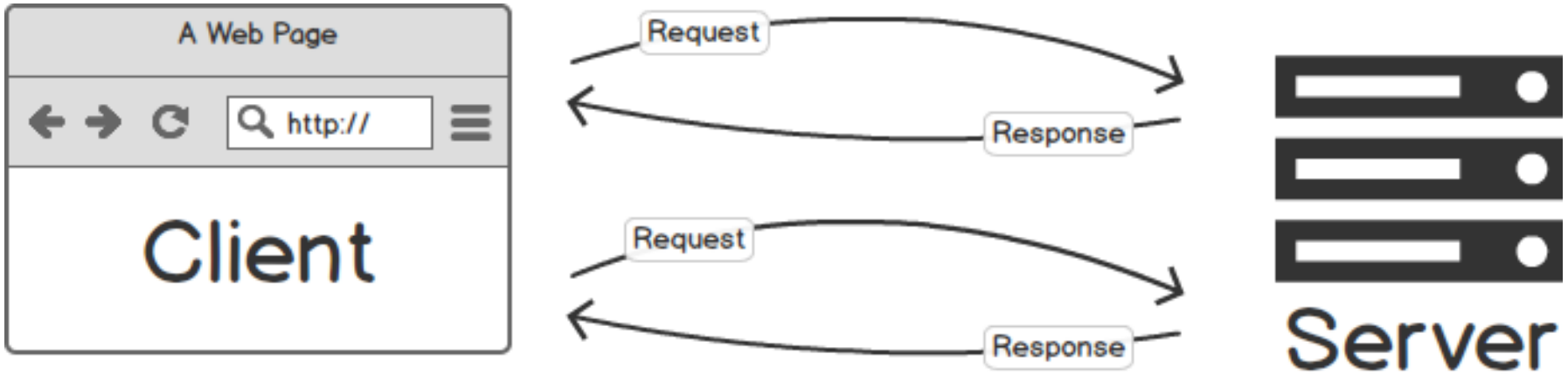
Summary

- Single Page Applications
- Framework JS
  - Introduzione
  - Confronto
- Introduzione a VUE

Siti web tradizionali - Frontend

- Riepilogo puntate precedenti …

Server-based Application



Problemi – Lato Client

- La pagina viene caricata completamente ogni volta che viene eseguita una richiesta al server.
- L'utente deve aspettare che tutti i file siano scaricati (HTML, CSS, JS, immagini).
- Il contenuto scaricato deve essere renderizzato.
- Nessuna business logic client side
- Nessun modello MVC
- Senza rete l'applicazione non funziona (funzionamento solo ***online***)

Problemi – Lato Server

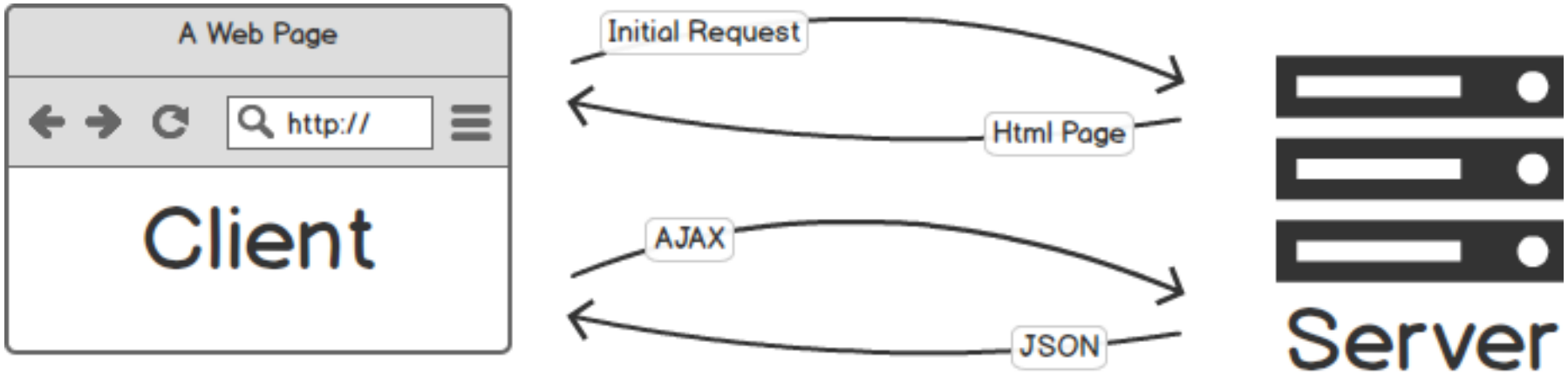
- Ad ogni richiesta la pagina deve essere generata dall'inizio.
- Banda consumata per inviare sempre gli stessi file.
- Tutte le computazioni sono eseguite lato server.
- Un cambio di tecnologia implica la riscrittura completa del codice che gestisce il rendering della pagina.

Single Page Application

Una possibile soluzione …

- «A Single Page Application is a web app in which the majority of interactions are handled on the client without the need to reach a server with a goal of providing a more fluid user experience»

Single Page Application - Interazione



Vantaggi

- Interazione App-like.
- Bottoni Indietro/Avanti funzionano.
- Possibilità di funzionamento anche ***offline***.
- Transizioni fluide tra i diversi stati delle pagine.
- CSS e JS inviati solamente nella richiesta iniziale.
- I cambiamenti alla pagina avvengo via Javascript, usando Template e la manipolazione del DOM.

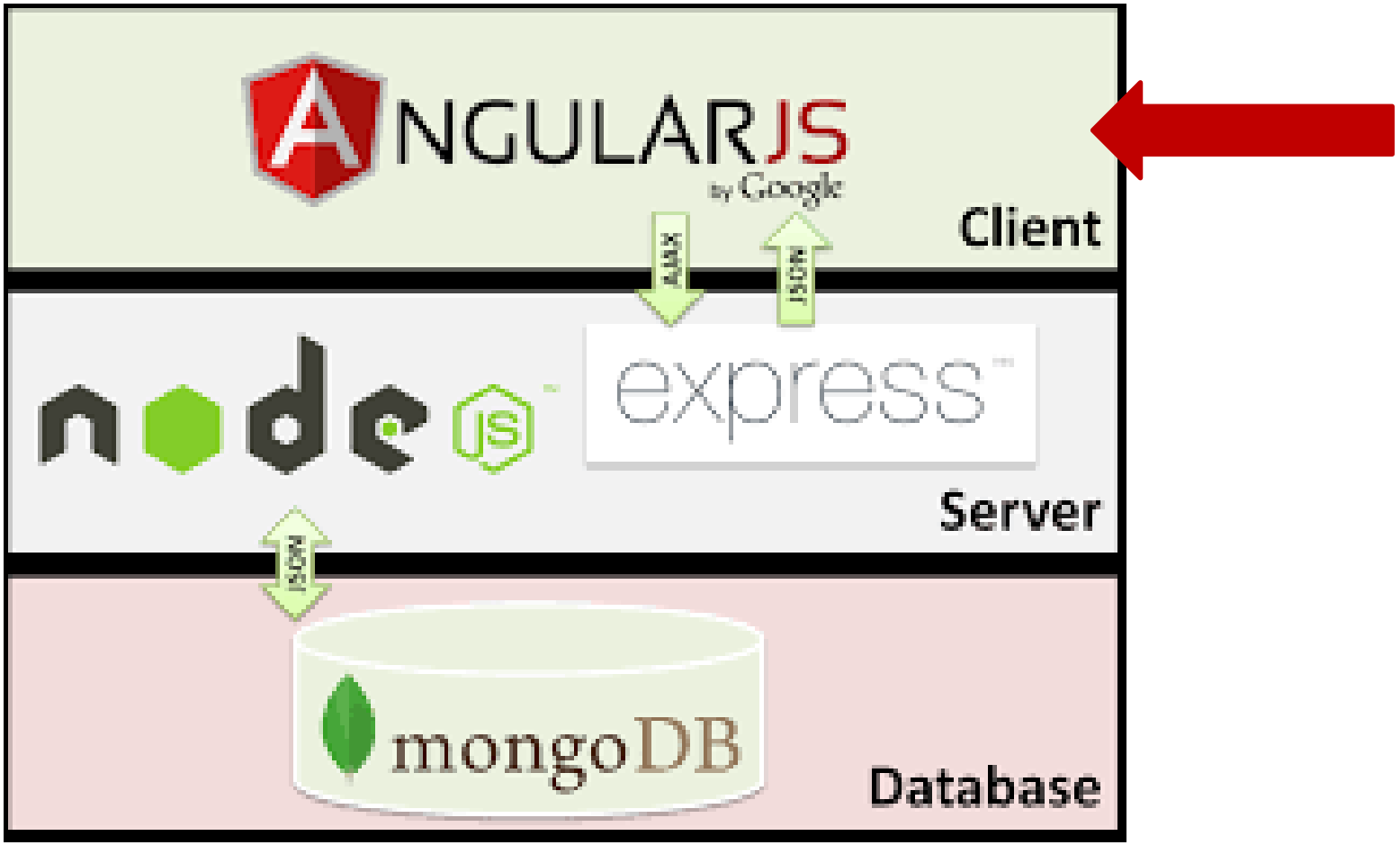
JQuery Ninja?

- Vari problemi:
  - Stato dell'applicazione risiede nel DOM
  - Lo stato guida il comportamento
  - Si accoppia logica e presentazione
  - Non è pensato per gestire la logica di un'applicazione

Storia

- Nascono diverse tecnologie (librerie/framework) per le SPA
- Conferenza Throne of JS a Toronto (2012)
  - Gli sviluppatori dei maggiori framework JS si riuniscono per discutere come affrontare il bisogno delle nuove architetture SPA

AngularJS


ANGULARJS
by Google
Client
AJAX
JSON
express
Server
JSON
mongoDB
Database

Angular(JS)

- In realtà, dalla sua versione 2, si chiama Angular (senza il js)
- È un framework javascript lato client, per questo motivo se ne può utilizzare uno diverso
- I più diffusi sono
  - Angular
  - REACT
  - VUE

AngularJS

- Release 1.0 nel 2012.
- Divenne subito molto popolare
- Nel picco della sua popolarità (2014) il team di sviluppo annunciò una nuova versione Angular 2 *incompatibile con la versione precedente* (in pratica una completa riscrittura del framework)

Altri Framework

- Nel 2013 Facebook rilascia **React** (open source) dopo averlo usato in produzione per almeno due anni
- Nel 2014 viene rilasciato **Vue.js** da Evan You, che aveva lavorato precedentemente nel team di sviluppo di AngularJS
- Nel settembre 2016 viene rilasciato **Angular 2**, mentre a marzo 2017 viene rilasciata la versione 4. L'ultima versione rilasciata è la **11** (rilasciata in data 11 novembre 2020)

Q&A

- Ci sono domande?

Quale usare?

- … Dipende!
- Vediamo le maggiori differenze in termini di:
  - Storia e longevità
  - Popolarità ed ecosistema
  - Supporto aziendale
  - Esperienza iniziale e curva di apprendimento
  - Capacità richieste
  - Completezza e performance

Storia e longevità

- **Angular**:
  - prima release nel 2010, con il nome di AngularJS. La seconda versione è stata completamente riscritta ed è stata rilasciata nel 2016
  - realizzazione di applicazioni di nuova generazione, come le applicazioni a pagina singola (SPA)

Storia e longevità

- **REACT**
  - talvolta identificato con React.js/ReactJS, rilasciato nel marzo 2013
  - creazione di interfacce utente (UI) dedicate ad applicazioni web complesse, in modo semplice, veloce e scalabile

Storia e longevità

- **Vue.js**
  - rilasciato nel febbraio 2014
  - Curiosità: Il creatore del progetto, Evan You, fu uno sviluppatore di Google che utilizzò intensivamente AngularJS per la realizzazione di svariati progetti
  - è stato prodotto per essere un'alternativa "leggera" ad Angular
    - L'autore voleva fornire agli sviluppatori un framework dalla curva di apprendimento bassa, ma che potesse crescere fino a poter gestire applicazioni molto complesse

Popolarità ed ecosistema

Framework	Stelle GitHub	Tagged Questions su StackOverflow
Angular	72 K	249 K
React	166 K	295 K
Vue.js	181 K	76 K

Framework	Numero di pacchetti npm	Download npm (nell'ultima settimana)
Angular	48.760	597 K
React	155.019	10.771 K
Vue.js	47.185	2.309 K

Supporto aziendale

Framework	**Sponsor**
Angular	Google
React	Facebook
Vue.js	Supporto da svariate organizzazioni di piccola/media dimensione, principalmente attraverso Patreon, un tool online dedicato ai creatori di servizi

Esperienza iniziale e curva d'apprendimento

- **Angular**: esperienza iniziale di utilizzo più complessa
- Per lanciare un'applicazione è necessario installare la CLI (command line interface for Angular), ed impostare alcuni parametri:

```
// installazione CLI
npm install -g @angular/cli
// creazione di un nuovo progetto
ng new my-app
// avvio del server ('open' apre automaticamente il
// browser)
cd my-app ng serve --open
```

Esperienza iniziale e curva d'apprendimento

- **Angular**: A questo punto sarà possibile lavorare con i file dei componenti generati da Angular, come *src/app/app.component.ts*

```
export class AppComponent {
      title = 'Mio titolo';
}
```

- Per iniziare, occorre seguire il tutorial: **Tour of Heroes**

Esperienza iniziale e curva d'apprendimento

REACT: offre due guide differenti per iniziare a scrivere codice avvalendosi delle sue funzionalità.

- La prima, molto semplice, racchiude lo snippet (con tanto di spiegazione) necessario per generare il classico "Hello World":

```
ReactDOM.render(
<h1>Hello, world!</h1>,
document.getElementById('root')
);
```

Esperienza iniziale e curva d'apprendimento

- **REACT**: assume che lo sviluppatore sia in grado di scrivere applicazioni Javascript conformi agli ultimi standard ECMA (6)
- Per esempio:
  - keywords **`let`** e **`const`** per definire variabili
  - la keyword **`class`** per definire le classi
  - le arrow functions, **`=>`**

Piccola Parentesi: =>

// (param1, param2, paramN) => expression

// prima

```
var multiplyES5 = function(x, y) {
     return x * y;
};
```

// ora

```
const multiplyES6 = (x, y) => { return x * y
};
```

Esperienza iniziale e curva d’apprendimento

- La seconda guida (step-by-step) mostra tutti i concetti essenziali di **React** ad un livello decisamente più alto, permettendo di creare una prima applicazione completa
- Può essere completata in un’ora di tempo dalla maggior parte degli sviluppatori

Esperienza iniziale e curva d'apprendimento

- **Vue.js**: per installare basta includere un tag **`<script>`** in un documento HTML

```
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
```

- Nella guida ufficiale sono spiegate le procedure più composite di installazione, che comprendono anche l'uso della CLI vue-cli che è però sconsigliata a chi si trova ad utilizzare Vue.js per la prima volta o che ha scarsa dimestichezza con le strutture di Node.js

Esperienza iniziale e curva d'apprendimento

- **Vue.js**: La sintassi è basata su un sistema che permette di produrre il DOM a partite da un apposito template:

```
<div id="app"> {{ message }} </div>
var app = new Vue({ el: '#app', data: { message:
'Hello Vue!' } })
```

- **Vue** è un framework progressivo: alcune componenti non sono incluse di default (ma ha librerie ufficiali che si occupano di aggiungere le varie funzionalità corredate da un'ottima documentazione)

Esperienza iniziale e curva d'apprendimento

- La curva di apprendimento migliore è quella di **Vue**, seguito da **React**, che richiede un tempo leggermente superiore da dedicare all'apprendimento. **Angular** è distanziato (il suo scopo è quello di consentire la realizzazione di applicazioni di nuova generazione anche a livelli di complessità elevati)

Q&A

- Ci sono domande?

Capacità richieste

- **React** richiede necessariamente l'utilizzo del Javascript «moderno» (ECMA script 6+)
- **React** fa uso di **JSX**, una sorta di "estensione" della sintassi Javascript utile a mescolare HTML e JS in un unico codice
- NB: **JSX** non è obbligatorio per utilizzare **React**, MA è quasi impossibile trovare un'applicazione reale che non ne faccia uso

Esempio JSX (React)

```
function formatName(user) {
   return user.firstName + ' ' + user.lastName;
}

const user = { firstName: 'Harper', lastName: 'Perez' };
const element = ( <h1> Hello, { formatName(user)}! </h1> );

ReactDOM.render( element, document.getElementById('root') );
```

Capacità richieste

- Vue.js è il framework che risulta più familiare allo sviluppatore web (che proviene da librerie come jQuery)
- Si basa su componenti quali HTML, Javascript (ECMA 5 o 6) e CSS nella loro versione tradizionale, con qualche eccezione, per la costruzione di UI e la stilizzazione
- In modo simile ad Angular, Vue utilizza attributi HTML “custom” per gestire compiti quali data-binding e controllo del flusso

Piccoli esempi in VUE

```
<a v-bind:href="url"></a>
<a :href="url"></a>

<button v-bind:disabled="someDynamicCondition">
Button</button>
<button:disabled="someDynamicCondition">Button</button>
```

Capacità richieste

- **Angular** è stato scritto utilizzando TypeScript
- Per questo motivo, occorre conoscere TypeScript per lavorare con Angular
- Uno sviluppatore Javascript “standard” avrà maggiori difficoltà, dato che dovrà prendere confidenza con un linguaggio in cui la tipizzazione è statica, mentre un programmatore che proviene da C# o Java, linguaggi d’eccellenza a tipizzazione statica, si troveranno maggiormente a loro agio

Capacità richieste in breve

Framework	**Requisiti**
React	JS (ES6+), JSX, CSS-in-JS
Vue.js	HTML, JS (ES5+), CSS
Angular	TypeScript

Completezza

Framework	Strumenti e scopo principale
Angular	All-in-one
React	UI management
Vue	Livello View, componenti aggiuntivi ufficiali

Completezza

- **Angular** è il framework nativamente più completo, con la sua struttura "***all-in-one***" che fornisce UI management, state management, routing, test end-to-end e molto altro
- **React** è improntato principalmente sulla UI, il che significa che per funzionalità avanzate è necessario estendere il suo core tramite strumenti esterni (es. Redux e MobX per la gestione dello stato delle applicazioni web)

Completezza

- **Vue.js** è un framework “progressivo”, che fornisce un core in grado di produrre applicazioni sfruttando la sua logica ed il suo motore di base
- Per le funzionalità aggiuntive è però necessario utilizzare componenti extra
  - Esistono svariati componenti di alta qualità, che coprono tutte le principali caratteristiche richieste da un’applicazione di nuova generazione, sia prodotte da terze parti, sia ufficiali, come Vuex (per lo state management centralizzato)
- Il core di Vue è infatti principalmente basato sul livello delle “View”

Completezza

- Riassumendo …
- Una soluzione full-featured come Angular potrebbe NON essere la scelta migliore nel caso in cui si stia realizzando un'applicazione che non necessiti di tutte le sue funzionalità, oppure una che deve essere realizzata in tempi brevi
- Nel caso di framework più leggeri (e nativamente meno completi) come Vue e React, abbiamo la possibilità di scegliere tra estensioni ufficiali (caso di Vue) oppure tra un ampio ecosistema di librerie aggiuntive di terze parti (caso di React)

Fattori di performance

- A causa della sua completezza “all-in-one”, Angular risulta essere la libreria più pesante, segue React e infine Vue
- Risultati di benchmark mostrano una leggera deviazione di performance tra i tre framework, dove Angular risulta il più “lento” e Vue.js il più “veloce”

VUE

- Si può apprendere rapidamente
- Vicino alla versione “classica” di HTML/CSS/Javascript
- Si può espandere progressivamente a seconda dell’espansione della complessità dell’applicazione
- Veloce e leggero

Q&A

- Ci sono domande?

Vue.js

Caratteristiche fondamentali

- Permette di sviluppare interfacce web reattive che sfruttano il **dual-binding** tra modello dati e vista
  - rende possibile implementare un'applicazione ragionando in termini di *dati, variabili e oggetti, astraendosi rispetto* all'implementazione e aggiornamento del **DOM** della pagina
- È stato definito **framework progressivo**
  - è specializzato nella realizzazione delle viste HTML, MA permette di integrarsi facilmente con componenti di altre librerie e progetti

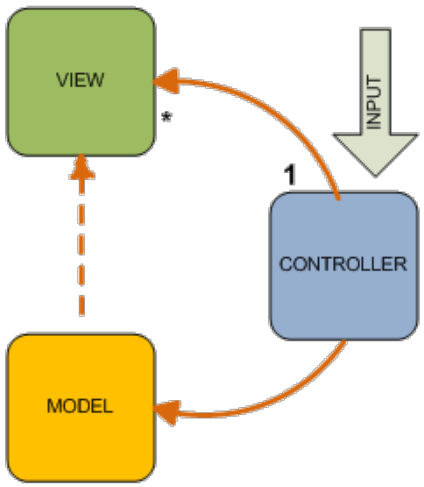
Caratteristiche fondamentali

- Implementa il **modello MVVM** (Model-View-ViewModel)
  - una declinazione del più famoso Model-View-Controller (MVC)
- I componenti di un'applicazione MVVM sono:
  - il Modello (Model) che rappresenta l'implementazione del dominio dati gestito dall'applicazione, come per la 'M' in MVC
  - la Vista (View) che rappresenta il componente grafico renderizzato all'utente, composta da HTML e CSS
  - il Modello per la Vista (ViewModel) che rappresenta il collante tra i precedenti componenti e fornisce alla view i dati in un formato consono alla presentazione e il comportamento di alcuni componenti dinamici

MVVM

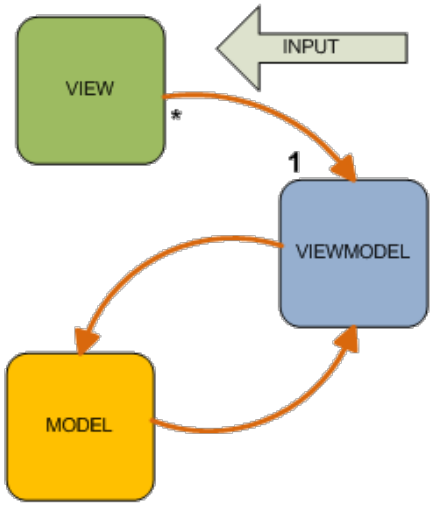
- La principale differenza è **tra Controller e ViewModel**
  - Il controller è di fatto una porzione di codice che esegue particolari logiche di business (grazie ai Model) e che ritorna una View da mostrare all'utente
  - Il ViewModel invece rappresenta di fatto un modello parallelo al Model, e che viene direttamente bindato alla View e che descrive il comportamento di quest'ultima con funzioni associate per esempio al click su un elemento.
- Il Controller esegue logiche di business *prima* del rendering della View, il ViewModel definisce il comportamento dell'applicazione *a runtime*.

MVC



- Input is directed to the Controller.
- Many-to-many relationship between View and Controller.
- View doesn't have any knowledge of the Controller.
- View is aware of the Model it is expecting to pass on to it.

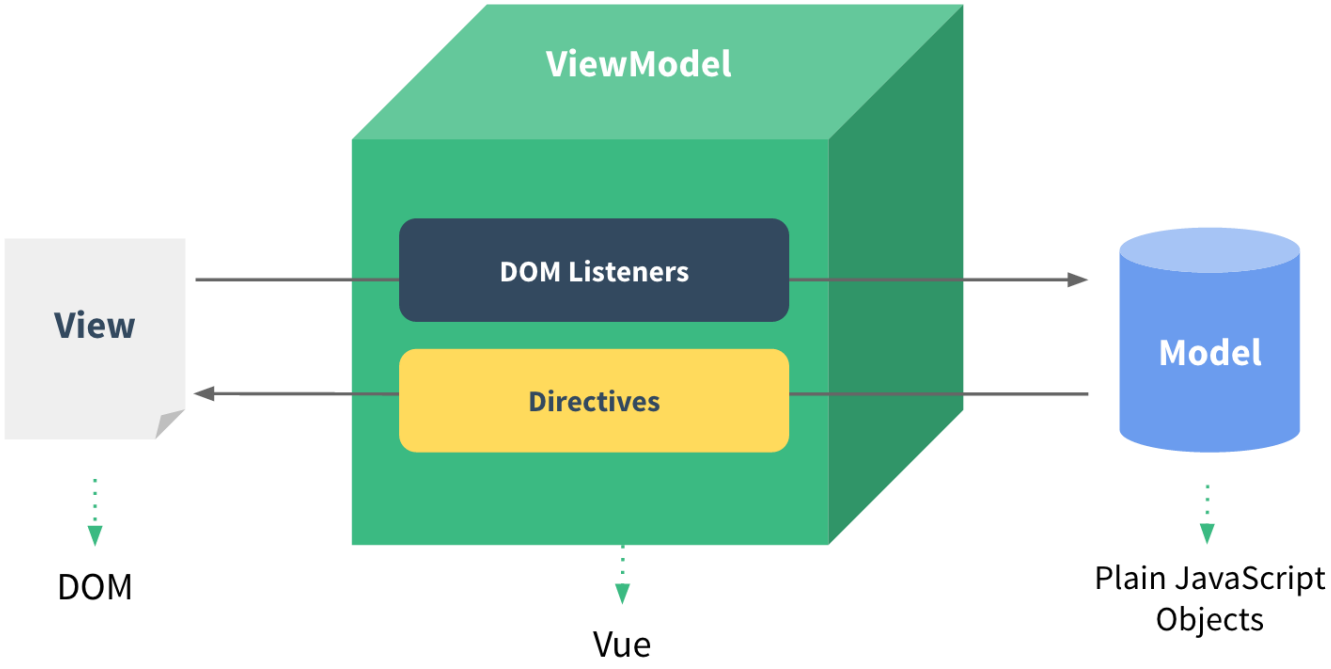
MVVM



- Input is directed to the View
- One-to-many relationship between ViewModel and View.
- ViewModel doesn't have any knowledge of it View.
- View is not aware of the Model. ViewModel updates the View.

Vue e MVVM

- Vue.js si focalizza sul ViewModel layer che connette la View con il Model attraverso un two way data bindings



Declarative rendering

- Vue utilizza un **sistema di template basato su HTML** e su particolari attributi (chiamati *direttive*) che permettono di definire il comportamento del layer di presentazione
- Il framework è in grado di interpretare il template e di compilarlo in un *Virtual DOM* che permette a Vue di
  - offrire reattività dei contenuti
  - effettuare cambiamenti alla pagina solo se davvero necessari (evitando sprechi di risorse e di tempo)

Data-binding

- Data-binding è la possibilità di *associare un elemento presentazionale con una particolare variabile o oggetto Javascript* e di essere certi che eventuali cambiamenti vengano *propagati alla view esclusivamente agendo sulla variabile*
- *Interpolazione di testi:* Binding effettuato attraverso la *Mustache Syntax*, le doppie parentesi graffe ({{...}})

Hello World!

- **Modello HTML**

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

- **JavaScript**

```
new Vue({
      el: '#app',
      data: { message: 'Hello World! '}
})
```

https://jsfiddle.net/mirrisil/fhjmg8y4/

Hello Word con viewmodel

HTML

```
<div id="app">
      {{message}}
      <input v-model="message">
</div>
```

VUE

```
new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue.js! '
  }
})
```

https://jsfiddle.net/mirrisil/sfrgn0Le/

Interpolazione di HTML

- Se volessimo mostrare tag HTML in modo dinamico, è possibile utilizzare la direttiva **`v-html`**
- Esempio:

```
<div id="app">
   My first name is
   <span v-html="firstname"></span>
   and my last name is
   <span v-html="lastname"></span>
</div>
```

Interpolazione di HTML

- E in Vue sarà

```
var vm = new Vue({
   el: '#app’,
   data: {
      firstname: ‘<strong>Mario</strong>’,
      lastname: ‘<strong>Rossi</strong>’
   }
   })
```

https://jsfiddle.net/mirrisil/2awnq51p/

Attributi HTML

- Si può fare il dual-binding anche per attributi HTML grazie alla direttiva **`v-bind`** e il nome dell'attributo come argomento della direttiva:

```
<div id="vue-app">
      <button v-bind:disabled="button1Disabled">
         Button 1
       </button>
</div>
```

Attributi HTML

- E in Vue.js

```
var vm = new Vue({
      el: '#app’,
      data: {
            button1Disabled: true,
            button2Disabled: false }
})
```

https://jsfiddle.net/mirrisil/uLo4fs0r/

Q&A

- Ci sono domande?

Direttive

- Le ***direttive*** sono speciali attributi HTML, che iniziano con prefisso **`v-`** e che permettono di aggiungere comportamenti a livello di presentazione
- Le direttive possono avere attributi e comportamenti
- Ne abbiamo già viste due
  - **`v-html`** (nessuno parametro richiesto)
  - **`v-bind`** (aveva bisogno di un parametro, ovvero «**`disabled`**») nella forma

    `v-{nomedirettiva}:{nomeargomento}`

Direttive con parametro

- Altro esempio di direttiva con attributo è **`v-on`** che permette di associare una funzione (un comportamento) a un particolare evento generato dall'interfaccia utente
- Es:

```
v-on:click
```

Modificatori

- Possiamo modificare il comportamento delle direttive grazie ai **modificatori**
  - particolari attributi che agiscono sulla parte funzionale della direttiva
- Uno dei modificatori più usati è **`prevent`** che utilizzato insieme a **`v-on`** permette di bloccare il comportamento di default del browser quando un particolare evento viene scatenato (ovvero **`event.preventDefault()`**)

```
<a href="#" v-on:click.prevent="goToHomepage">
```

Filtri

- Altra caratteristica delle direttive sono i **filtri**
- Particolari componenti dedicati alla formattazione del testo, utilizzabili tramite la direttiva **`v-bind`** o la mustache syntax

Es:

```
<div id="app">
{{firstname|uppercase|limit(3)}} </div>
```

Filtri

E nel file JS

```
var vm =
new Vue({
      el: '#app’,
      filters: {
            uppercase: function(text) {
                  return text.toUpperCase();
            },
            limit: function(text, length) {
                  return text.substring(0, length);
            }
            },
        data: { firstname: ‘Mario' } })
```

https://jsfiddle.net/mirrisil/2ozt3hpy/

Alias

- Per velocizzare la scrittura
- Ovvero
  - La direttiva **`v-bind:attributo`** può essere sostituita da **`:attributo`**
  - e **`v-on:click`** da **`@click`**

Alias

- Esempi:

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

//equivale a

```
<div :id="customId"></div>
```

```
<div v-on:click="changePage"></div>
```

//equivale a

```
<div @click="changePage"></div>
```

Precisazione

- I filtri dovrebbero occuparsi solamente di aspetti di presentazione, tant'è che Vue.js ne permette l'utilizzo solamente tramite la direttiva **`v-bind`** o con la sintassi "a graffe"
- Se però si vuole utilizzare il risultato dell'elaborazioni, allora occorre usare **Computed Properties**

Computed Properties

- Queste particolari property assumono valori dinamici in base ad altre property del nostro oggetto Vue, e reagiscono esattamente come se fossero proprietà statiche
- Esempio: In HTML

```
<div id="app">
      <input type="text" v-model="fullname">
      {{ firstname }} {{ lastname }}
</div>
```

Computed Properties

```
var vm = new Vue({
  el: '#app',
  data: {
    firstname: ‘Mario',
    lastname: ‘Rossi'
  },
  computed: {
    fullname: function() {
      return this.firstname + ' ' + this.lastname;
    }
  }
})
```

https://jsfiddle.net/mirrisil/2syceh0q/

Computed Properties

- Utilizzando le computed properties, è possibile accedere alla property **`fullname`** come se fosse una normalissima proprietà, al pari di **`firstname`** e **`lastname`**, che però rimane legata a loro
- Vue implementa questo meccanismo come un vero albero di dipendenze e di eventi: al cambio di una property, il framework sa esattamente quali computed properties rigenerare

Computed Properties

- Quest'ultima caratteristica di Vue porta con sè un secondo vantaggio, non direttamente collegato, ma fondamentale: il **caching**
- Utilizzare le computed piuttosto che i semplici property, evita di generare i valori di ritorno ad ogni accesso, così da rigenerarli solo se e quando le dipendenze dovessero cambiare
- Le computed property, di default, permettono di definire solamente il comportamento in lettura, ma è possibile anche definirne quello in scrittura

Q&A

- Ci sono domande?

Watchers

- Le computed property sono una concretizzazione di un concetto più astratto ovvero la possibilità di reagire di fronte al cambiamento dello stato del nostro componente Vue
- Questa astrazione viene implementata dai **watchers**
  - Grazie ai watchers è possibile, ad esempio, invocare un servizio locale di validazione e bloccare/sbloccare un eventuale form durante la digitazione dell'utente

Watchers

- I watchers sono strumenti comodi e funzionali, ma il loro utilizzo deve essere valutato con criterio.
- È necessario usarli solamente quando gli strumenti più specializzati (es. computed property) non sono efficaci

Rendering condizionale

- Grazie ad alcune direttive è possibile definire quali elementi del DOM della pagina mostrare in base ad alcune condizioni particolari
- Questo viene ottenuto tramite le direttive
  - **`v-if`**
  - **`v-else-if`**
  - **`v-else`**

Cicli

- La direttiva per le iterazioni è **`v-for`**
- Essa permette di ciclare una lista di elementi e di mostrare il contenuto HTML per ciascuno di essi, come un classico **`for`** in Javascript
- **`v-for`** è in grado di ciclare sia array numerici che associativi, permettendo accedere sia al valore che all'indice
- Se sullo stesso nodo HTML sono presenti sia la direttiva **`v-for`** che quella **`v-if`**, la prima avrà la priorità, in modo da poter aggiungere un controllo per ogni elemento della lista

Assegnazione di classi CSS

- Può essere fatta con **Sintassi ad oggetti** (1/2)
  - composti da una chiave che corrisponde al nome della classe da aggiungere e da un valore che determina lo stato della classe, se presente o no

Esempio:

```
<div :class="{ 'my-class-name' : hasClass }"><div>
```

Assegnazione di classi CSS ad oggetti

- Può essere fatta con **Sintassi ad oggetti** (2/2)
  - Questa modalità accetta anche oggetti con diverse proprietà:

```
<div :class="{ 'my-class-name-1' : hasClass1, 'my-class-name-2' : hasClass2 }"><div>
```

  - Ma anche la possibilità di passare direttamente un oggetto come proprietà:

```
<div :class="myCustomClasses"><div>
```

(e nel file VUE):

```
data: {
      myCustomClasses: { my-class-name: this.hasClass } }
```

Assegnazione di classi CSS con array

- Permette di passare da una lista di classi da attivare ad un nodo HTML
  - È possibile passare un array inline con diverse proprietà

```
<div :class="['my-class-name']"><div>
```

  - o direttamente una proprietà di tipo array:

```
<div :class="myCustomClasses"></div>
```

e

```
data: { myCustomClasses: ['my-class-name-1', 'my-class-name-2'] }
```

Sintassi ad oggetti per definire stili

- Utilizzando un oggetto Javascript possiamo comunicare ad un particolare nodo gli stili da applicare utilizzando la chiave dell'oggetto come nome della property CSS e come valore appunto il valore da assegnare alla regola CSS

```
<div :style="myCustomStyles"><div>
```

E

```
data: { myCustomStyles: { color: 'red',
height: '20px', fontSize: '12em' } }
```

Sintassi ad array

- Nel caso di differenti stili è possibile passare un array di oggetti alla direttiva **`style`**:

```
<div :style="[myCustomStyles,
{ width: '100px' }]"><div>
```

Q&A

- Ci sono domande?

Gestire gli eventi di base

- Per associare un particolare comportamento ad un evento nativo del browser, usiamo la direttiva **v-on** o l'alias @

```
<button v-on:click="onClickButton">
Click
<button>
```

In vue

```
methods: { onClickButton: function() {
alert('Button clicked!"); } }
```

Gestire gli eventi di base

- Possiamo anche scrivere codice Javascript inline all'interno dell'attributo:

```
<button @click="alert('Button clicked!’)”>
Click
<button>
```

(È da preferire l'altra ipotesi)

Modificatori

- Permettono di aggiungere dei comportamenti standard in maniera dichiarativa nel momento stesso in cui avviene l'associazione tra evento e la funzione callback, anche sfruttando la concatenazione di essi

Modificatori disponibili

- `.stop`: blocca la propagazione dell'evento all'interno della catena di bubbling;
- `.prevent`: previene il comportamento standard dell'elemento HTML (utile in caso di ancore o form);
- `.capture`: permette di invocare l'evento prima sull'elemento padre e solo successivamente su elementi figli;
- `.self`: permette di invocare l'evento solamente se il nodo corrente è il target (evita eventuali invocazioni nel caso l'evento scatti per esempio su elementi figli);
- `.once`: invoca la callback solamente una volta, rimuovendo il listener dopo la prima volta.

Altri modificatori

- Oltre a questi modificatori standard, esistono anche modificatori dedicati ai tasti della tastiera, da utilizzare insieme ad eventi keyboard-based come ad esempio **keyup** o **keydown**
  (Es. **`<input @keyup.enter="submit">`**)
- I modificatori dedicati agli eventi basati sulla tastiera sono: **`.enter`**, **`.tab`**, **`.delete`**, **`.esc`**, **`.space`**, **`.up`**, **`.down`**, **`.left`**, **`.right`**, **`.ctrl`**, **`.alt`**, **`.shift`** e **`.meta`**
- Sono disponibili anche modificatori speculari, ma basati sugli eventi del mouse: **`.left`**, **`.right`** e **`.middle`**

Form e input da tastiera

- La principale direttiva che Vue ci offre per la gestione degli input lato utente è **`v-model`**
- Questa direttiva permette di associare in maniera bidirezionale un modello dati ad un campo di una form.

  Per esempio:

```
<input type="text" v-model="myText"> {{ myText }}
//permette di avere il contenuto di myText, associato con il valore presente nel campo di testo
```

Form e input da tastiera

- Tramite le direttive
  - **`v-bind:true-value,`**
  - **`v-bind:false-value`**
  - **`v-bind:value`**

  è possibile modificare il valore di checkbox e radiobutton nei form

v-model e modificatori

- Anche la direttiva **v-model** ha dei modificatori
- **`.lazy`** cambia la modalità di aggiornamento del valore Javascript. Di default viene utilizzato l'evento *input* mentre se viene attivato il modificatore *lazy* viene utilizzato l'evento nativo *change*; Es: **`v-model.lazy.trim="myModel"`**
- **`.number`** converte automaticamente il valore a numero;
- **`.trim`** rimuove eventuali caratteri di spazio all'inizio e alla fine del valore inserito.

v-model e modificatori

Sono utili perché:

- `.trim` e `.number` permettono di evitare bug che potrebbero sorgere a fronte di una validazione dimenticata
- `.lazy` è spesso utile quando è necessaria l'esecuzione di task lenti o onerosi, e si preferisce limitare l'interattività dell'applicazione onde evitare problemi di performance

Q&A

- Ci sono domande?

Componenti

- Sono senza dubbio una delle funzionalità più interessanti di Vue
- Permettono di progettare applicazioni di medie/grandi dimensioni organizzando il codice in moduli riutilizzabili e ben strutturati
- Ciascun componente rappresenta una porzione di HTML arricchita da una logica di business incapsulata ed eventualmente una serie di regole CSS dedicate

Componenti

- Per utilizzare un componente possiamo utilizzare un custom tag

(**`<mio-componente></mio-componente>`**) oppure l'attributo **`is`**

(**`<div is="mio-componente"></div>`**)

- Per poter utilizzare i componenti è necessario registrarli

Registrazione componenti

- **globalmente** se è necessario renderli disponibili in tutta l'applicazione;
- **localmente** se vogliamo utilizzarli solo all'interno di un altro componente (per esempio un `<tab-panel>` potrebbe esistere solo all'interno di un `<tabs>`)

Registrazione globale e locale

- Registrazione globale:

```
Vue.component('mio-componente', { [...]
//elenco opzioni });
```

- Registrazione locale:

```
new Vue({
   [...],
   components: {
      'mio-componente': {
            [...] //elenco opzioni }
      }
   });
```

L’opzione *data*

- L’oggetto **`data`** che viene passato all’istanza di Vue è un oggetto che rappresenta il modello dati reattivo che viene reso disponibile da Vue
- NB: Utilizzando i componenti, è presente però una differenza sintattica, in quanto l’opzione non può più essere un oggetto, ma deve essere una funzione. Questo perché, in caso di diverse istanze dello stesso componente, non possiamo permetterci che i dati vengano sovrascritti

L’opzione *template*

- Ogni componente deve avere un template
- Diverse modalità
  - Template come stringa
  - Template come nodo esterno
  - Template inline

Template come stringa

- Definire il template come stringa all'interno del componente
- Esempio:

```
Vue.component('mio-componente', { template:
'<strong>Io sono il corpo del
componente</strong>' });
```

E, nel file HTML:

```
<mio-componente></mio-componente>
```

Template come nodo esterno

- In questo caso il template può essere contenuto all'interno di un tag **<template>** o di un tag **<script>** con type **text/x-template**
- Permette di avere componenti con template di una complessità notevolmente superiore

```
Vue.component('mio-componente', { template: '#mio-componente-template' });
```

E, html,

```
<mio-componente></mio-componente>
<template id="mio-componente-template">
<strong>Io sono il corpo del componente</strong>
</template>
```

Template inline

- Inline all'interno del nodo stesso del componente
- Permette di avere diverse istanze dello stesso componente ma con template differenti
- Usare l'attributo **inline-template**

```
Vue.component('mio-componente', { template:
'#mio-componente-template' });
```

E

```
<mio-componente inline-template>
<strong>Io sono il corpo del componente</strong>
</mio-componente>
```

Q&A

- Ci sono domande?

L’opzione *props*

- Le **props** rappresentano le proprietà esterne che permettono di configurare un componente
- Dato che ciascun componente ha uno scope di esecuzione isolato, *qualsiasi tipologia di dato esterno che deve essere reso accessibile dentro il componente, deve essere passato tramite appunto una prop*
- Le proprietà devono essere definite a priori tramite l’opzione **`props`**. Inoltre, se necessario, è possibile definire il loro tipo (**`String`**, **`Boolean`**, **`Number`**…), l’eventuale obbligatorietà (**`required`**) o un valore di default

L’opzione *props*

- Le **props** possono essere passate ad un componente in maniera statica o dinamica
- Questo significa che una proprietà può essere un valore definito in fase di inizializzazione di un componente oppure una variabile, controllabile da un componente più esterno.
- Ovviamente in questo secondo caso, un’eventuale modifica della variabile all’esterno del componente, produrrà una modifica anche al valore della proprietà

One way data flow

- Il motore di Vue implementa il cosiddetto **one way data flow** come modalità con la quale i componenti comunicano tra di loro
- Ovvero le **`prop`** passate ad un componente possono essere modificate dall'esterno, ma non dal componente stesso, in quanto il flusso di dati è monodirezionale, dall'alto al basso
- Se per qualche ragione cercassimo di modificare il valore di una **`prop`** all'interno di un componente, Vue lancerà un warning in console
- Soluzione… ->

Eventi custom

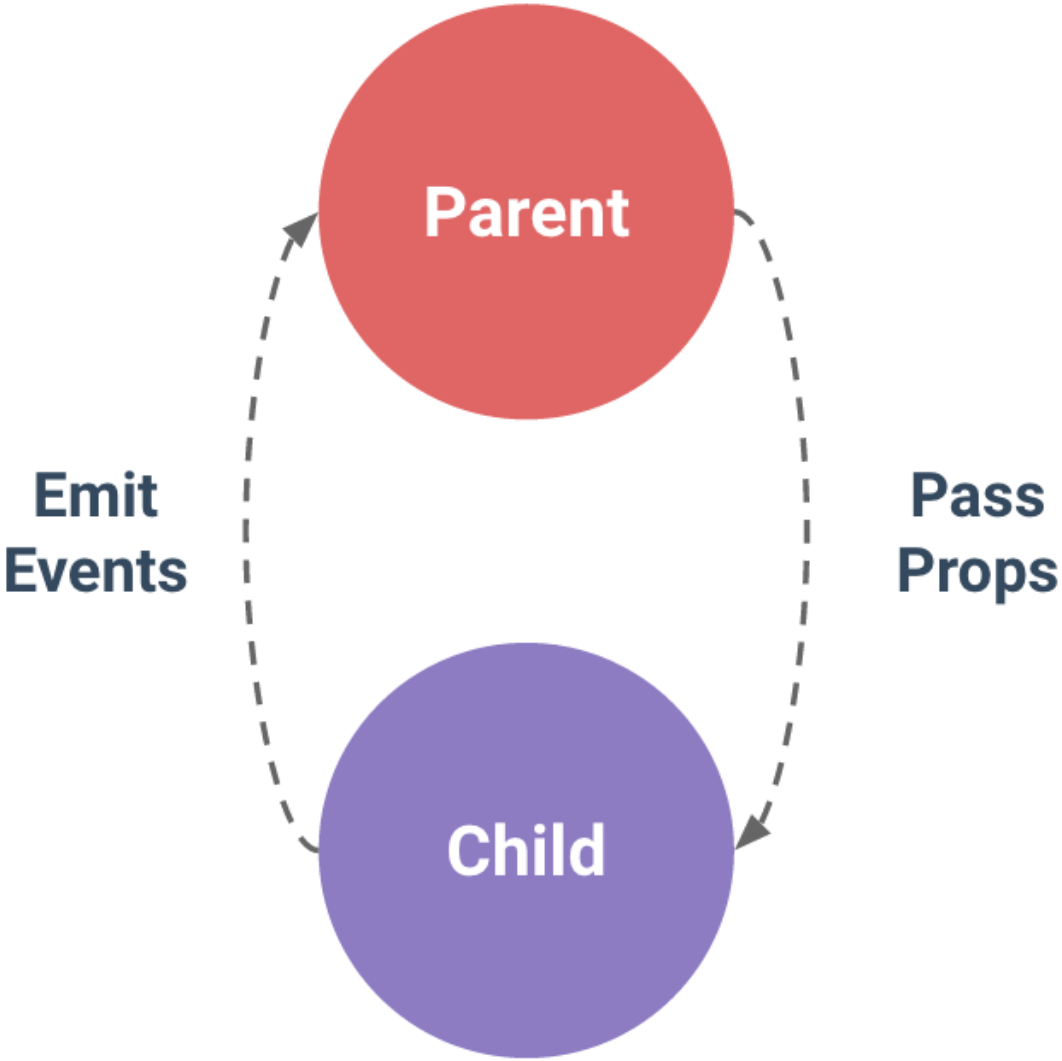
- Grazie alla definizione di **eventi custom**, è possibile comunicare qualcosa da un componente verso l'esterno
- Ogni componente presenta due metodi:
  - **`$on`**
    - per mettersi in ascolto
    - Spesso l'utilizzo del metodo **`$on`** verrà sostituito dalla direttiva **`v-on`**
  - **`$emit`**
    - per generare un evento

sync

- Dato che l'esigenza di propagare modifiche di dati da un componente verso il padre è frequente, gli sviluppatori di Vue hanno introdotto dalla versione 2.3.0 il modificatore **sync**
- E' rapido e non richiede la scrittura di metodi ad hoc

https://jsfiddle.net/mirrisil/ck8ejxw5/1/

Quindi..



Comunicazione tra componenti non correlati

- Problema: come far comunicare tra loro componenti non correlati tra loro?
- Soluzione: creare una istanza extra di Vue che funga da bus di eventi sul quale qualsiasi componente interessato ad un determinato evento potrebbe mettersi in ascolto

https://jsfiddle.net/mirrisil/w83ed4h6/

Componenti: gli slot

- Sviluppando applicazioni che fanno uso intensivo di componenti, potrebbe capitare che il template non sia sufficiente per rappresentare perfettamente le nostre esigenze, poiché *alcuni aspetti di esso potrebbero essere dinamici in base all'istanza di componente*
- Soluzione: gli **`slot`**
  - porzioni di template che possono essere configurate dall'esterno di un componente e che vengono mixate al template originale

Definire componenti riutilizzabili

- *Best practices* per creare dei componenti non solo funzionali nel breve termine, ma riutilizzabili anche in contesti differenti
- Ciò permette di creare interfacce il più possibile standardizzate e di alta qualità, nonché di aumentare la disponibilità di componenti open-source

Definire componenti riutilizzabili

- Secondo gli sviluppatori di Vue, le modalità per interfacciarsi con un componente sviluppato da terzi sono tre:
  - le **props** dovrebbero essere utilizzate per passare un set di dati al componente dal contesto esterno;
  - gli **eventi** dovrebbero essere utilizzati per influenzare il contesto esterno dall'interno del componente;
  - gli **slot** dovrebbero essere utilizzati dal contesto esterno per influenzare il contenuto del componente

Direttive personalizzate

- Spesso può capitare che sia necessario intervenire direttamente sul DOM della pagina tramite plugin esterni o per utilizzare API native del browser
- Possibilità di creare **direttive personalizzate** da richiamare all'interno dei nostri template esattamente come le direttive native di Vue (per esempio **`v-if`** o **`v-model`**)

Q&A

- Ci sono domande?

Riferimenti

- https://www.html.it/guide/vue-js-la-guida/
- https://vuejs.org/v2/guide/
- https://www.html.it/articoli/ecmascript-6-harmony-il-javascript-che-verra/
- https://angular.io/tutorial
- https://www.html.it/articoli/angular-react-vue-framework-javascript-a-confronto/
- https://reactjs.org/docs/introducing-jsx.html
- https://www.stefankrause.net/js-frameworks-benchmark6/webdriver-ts-results/table.html
- https://www.html.it/pag/69928/direttive-personalizzate-2/