Angular

# Introduzione

La necessità di avere un metodo di sviluppo Web lato client veloce, potente e rigoroso ha reso inadeguata l’adozione canonica di HTML CSS e Javascript; con questi, risulta comunque scomodo sviluppare progetti di grandi dimensioni mantenendo una struttura del progetto chiara e ordinata.

Per risolvere tali problemi nascono nuovi framework e metodologie di sviluppo che sfruttando di fatto HTML, CSS e JavaScript propongono un nuovo modo di sviluppare, più semplice, più veloce e strutturato ai grandi progetti.

Il compito è quello di consentire lo sviluppo degli applicativi tramite nuovi linguaggi per poi compilare e pubblicare un’applicazione che nasce dalla traduzione di questi linguaggi in JavaScript con HTML e CSS.

Tra questi framework è degno di nota **Angular**.

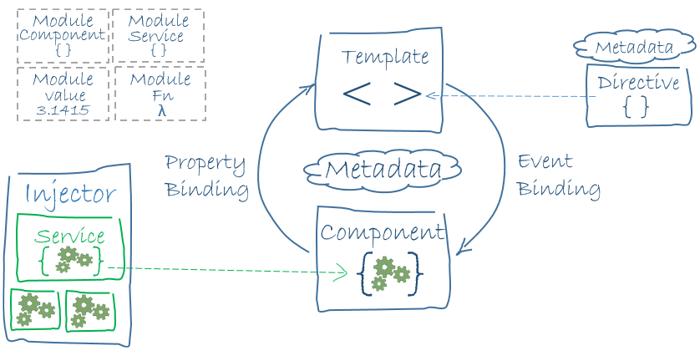
Angular si presenta come un framework **MVW** (Model-View-Whatever, anche se di fatto si parla di un **MVVM** ovvero Model View ViewModel) ed è basato su **TypeScript**, un linguaggio open source sviluppato da Microsoft che si basa su un **super-set** di JavaScript (pertanto qualsiasi programma scritto in JavaScript può funzionare con TypeScript direttamente).

Per aiutare lo sviluppatore durante il suo lavoro, Angular fornisce una shell chiamata **AngularCli** basata su **nodeJS** che consente di creare i componenti dei progetti, orchestrare le procedure di compilazione del codice TypeScript in JavaScript e fornire una console di Debug.

Il punto di forza di Angular è la possibilità di sviluppare su piattaforme diverse tra cui quelle web, mobile web, mobile native e desktop nativa garantendo ottime performance e velocità grazie anche alla possibilità di poter gestire il rendering delle applicazioni lato server.

Angular agevola lo sviluppo di applicazioni attraverso i concetti di **Data Binding** per *aggiornare in maniera bidirezionale i dati* tra le viste e i modelli, e consente di gestire il comportamento degli elementi del **DOM** attraverso i **Controllers**.

Angular consente di creare **Componenti personalizzati** che hanno la proprio logica di business e le loro viste in maniera totalmente separata dal resto dell’applicazione, con il vantaggio di evitare conflitti che nascono da diversi componenti che vengono utilizzati all’interno dello stesso progetto (ad esempio dovuti agli stili CSS) e agevolando quindi il riutilizzo di codice all’interno dell’applicazione stessa o per altri progetti.

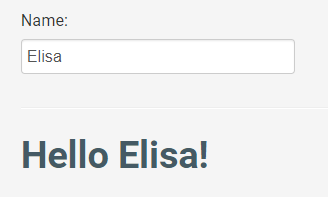
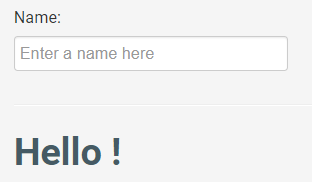


NB: framework= insieme di librerie che permette di sviluppare un applicazione concentrandosi sulla logica dell’applicazione.

# Le basi

1. <!doctype html>
2. <html ng-app>
3. <head>
4. <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
5. </head>
6. <body>
7. <div>
8. <label>Name:</label>
9. <input type="text" ng-model="yourName" placeholder="Enter a name here">
10. <hr>
11. <h1>Hello {{yourName}}!</h1>
12. </div>
13. </body>
14. </html>

Result:



Note:

ng-app: di essere attivo in questa parte della pagina, in questo caso l'intero documento

angular.min.js: carica AngularJS.

ng-model: questo collega la forma e il modello. Ciò significa che qualsiasi modifica al controller aggiorna i dati nel modello e quando si modifica il modello aggiorna il controller

{{yourName}}: le doppie parentesi sono un modo dichiarativo di specificare le posizioni di data binding nell’HTML.  
AngularJS aggiornerà automaticamente questo testo ogni volta che la variabile ‘yourName’ cambierà.

## Aggiungiamo Controllo

### Data Binding

L'associazione dati è un modo automatico per aggiornare la vista ogni volta che il modello cambia, nonché per aggiornare il modello ogni volta che la vista cambia. Questo è fantastico perché elimina la manipolazione del DOM dall'elenco delle cose di cui ti devi preoccupare.

### Controller

I controller sono il comportamento dietro gli elementi DOM. AngularJS ti consente di esprimere il comportamento in una forma leggibile e pulita senza il solito standard di aggiornamento del DOM, registrazione di callback o visualizzazione di modifiche al modello.

### JavaScript

A differenza di altri framework, non è necessario ereditare da tipi proprietari per avvolgere il modello nei metodi di accesso. I modelli AngularJS sono semplici vecchi oggetti JavaScript. Ciò rende il tuo codice facile da testare, mantenere, riutilizzare e di nuovo privo di boilerplate.

1. <!doctype html>
2. <html ng-app="todoApp">
3. <head>
4. <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
5. <script src="todo.js"></script>
6. <link rel="stylesheet" href="todo.css">
7. </head>
8. <body>
9. <h2>Todo</h2>
10. <div ng-controller="TodoListController as todoList">
11. <span>{{todoList.remaining()}} of {{todoList.todos.length}} remaining</span>
12. [ <a href="" ng-click="todoList.archive()">archive</a> ]
13. <ul class="unstyled">
14. <li ng-repeat="todo in todoList.todos">
15. <label class="checkbox">
16. <input type="checkbox" ng-model="todo.done">
17. <span class="done-{{todo.done}}">{{todo.text}}</span>
18. </label>
19. </li>
20. </ul>
21. <form ng-submit="todoList.addTodo()">
22. <input type="text" ng-model="todoList.todoText" size="30"
23. placeholder="add new todo here">
24. <input class="btn-primary" type="submit" value="add">
25. </form>
26. </div>
27. </body>
28. </html>

Note

* ng-controller: il comportamento del contenuto con questo elemento verrà gestito utilizzando la classe TodoListController definita in todo.js
* ng-click: invece di registrare gestori di eventi, dichiariamo quale metodo chiamare sul controller. Quindi in questo caso facendo click qui, si chiamerà il metodo archive()
* ng-repeat: lo utilizziamo per srotolare una raccolta. Qui, per ogni oggetto in *todos*, Angular JS creerà una nuova copia dell’elemento <li>. Quando degli oggetti vengono aggiunti alla raccolta todos, *np-repeat aggiunge automaticamente nuovi elementi* <li> nel DOM. Allo stesso modo, quando gli oggetti vengono rimossi, viene rimosso anche nel todos l’elemento corrispondente.
* ng-model="todo.done": in questo caso Angular JS copia automaticamente lo stato della casella di controllo in todo.done ; al contrario se aggiorno todo.done vedrò la casella di controllo rispondere di conseguenza.
* class="done-{{todo.done}}": per mostrare i barrati per le attività completate, abbiamo creato una classe CSS done-true. Quando todo.done è *true* il selettore CSS risultante è *done-true* che quindi applica il barrato.
* ng-submit="todoList.addTodo()": intercetta l’invio del modulo e chiama *addTodo()*. All’interno di questo metodo leggiamo la proprietà *todoText* e la inseriamo nella collezione *todos*.

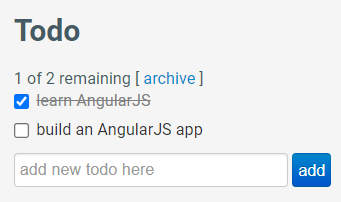
todo.js

1. angular.module('todoApp', [])
2. .controller('TodoListController', function() {
3. var todoList = this;
4. todoList.todos = [
5. {text:'learn AngularJS', done:true},
6. {text:'build an AngularJS app', done:false}];
8. todoList.addTodo = function() {
9. todoList.todos.push({text:todoList.todoText, done:false});
10. todoList.todoText = '';
11. };
13. todoList.remaining = function() {
14. var count = 0;
15. angular.forEach(todoList.todos, function(todo) {
16. count += todo.done ? 0 : 1;
17. });
18. return count;
19. };
21. todoList.archive = function() {
22. var oldTodos = todoList.todos;
23. todoList.todos = [];
24. angular.forEach(oldTodos, function(todo) {
25. if (!todo.done) todoList.todos.push(todo);
26. });
27. };
28. });

todo.css

1. .done-true {
2. text-decoration: line-through;
3. color: grey;
4. }

Risultato:



## Componenti

### Direttive

Le direttive sono una funzionalità unica e potente disponibile in AngularJS. Le direttive ti consentono di inventare una nuova sintassi HTML, specifica per la tua applicazione.

### Componenti riutilizzabili

Usiamo le direttive per creare componenti riutilizzabili. Un componente ti consente di nascondere strutture DOM, CSS e comportamenti complessi. Ciò ti consente di concentrarti su ciò che fa l'applicazione o su come l'applicazione appare separatamente.

### Localizzazione

Una parte importante delle app serie è la localizzazione. I filtri locali di AngularJS e le direttive stemming ti forniscono gli elementi costitutivi per rendere la tua applicazione disponibile in tutte le impostazioni locali.

1. <!doctype html>
2. <html ng-app="app">
3. <head>
4. <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
5. <script src="components.js"></script>
6. <script src="app.js"></script>
7. </head>
8. <body>
9. <tabs>
10. <pane title="Localization">
11. <span>Date: {{ '2012-04-01' | date:'fullDate' }}</span><br>
12. <span>Currency: {{ 123456 | currency }}</span><br>
13. <span>Number: {{ 98765.4321 | number }}</span><br>
14. </pane>
15. <pane title="Pluralization">
16. <div ng-controller="BeerCounter">
17. <div ng-repeat="beerCount in beers">
18. <ng-pluralize count="beerCount" when="beerForms"></ng-pluralize>
19. </div>
20. </div>
21. </pane>
22. </tabs>
23. </body>
24. </html>

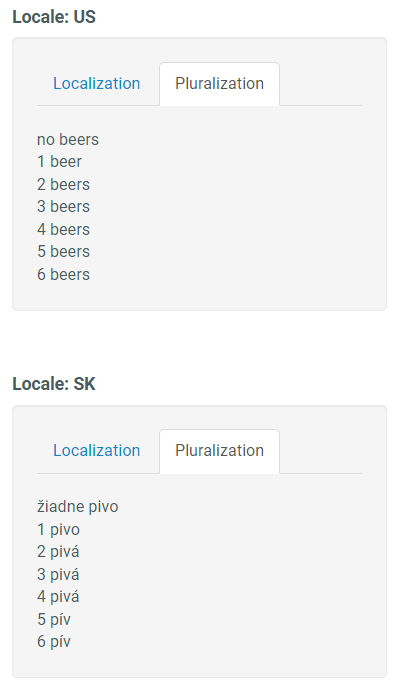
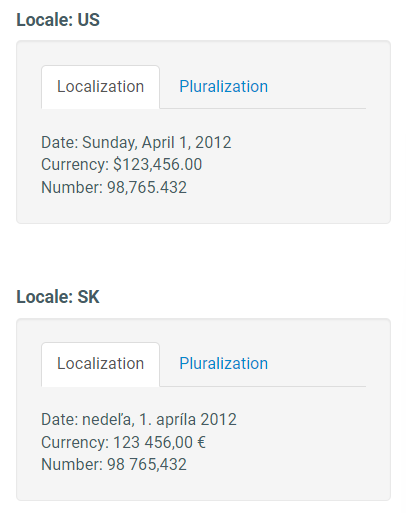
app.js

1. angular.module('app', ['components'])
3. .controller('BeerCounter', function($scope, $locale) {
4. $scope.beers = [0, 1, 2, 3, 4, 5, 6];
5. if ($locale.id == 'en-us') {
6. $scope.beerForms = {
7. 0: 'no beers',
8. one: '{} beer',
9. other: '{} beers'
10. };
11. } else {
12. $scope.beerForms = {
13. 0: 'žiadne pivo',
14. one: '{} pivo',
15. few: '{} pivá',
16. other: '{} pív'
17. };
18. }
19. });

components.js

1. angular.module('components', [])
3. .directive('tabs', function() {
4. return {
5. restrict: 'E',
6. transclude: true,
7. scope: {},
8. controller: function($scope, $element) {
9. var panes = $scope.panes = [];
11. $scope.select = function(pane) {
12. angular.forEach(panes, function(pane) {
13. pane.selected = false;
14. });
15. pane.selected = true;
16. }
18. this.addPane = function(pane) {
19. if (panes.length == 0) $scope.select(pane);
20. panes.push(pane);
21. }
22. },
23. template:
24. '<div class="tabbable">' +
25. '<ul class="nav nav-tabs">' +
26. '<li ng-repeat="pane in panes" ng-class="{active:pane.selected}">'+
27. '<a href="" ng-click="select(pane)">{{pane.title}}</a>' +
28. '</li>' +
29. '</ul>' +
30. '<div class="tab-content" ng-transclude></div>' +
31. '</div>',
32. replace: true
33. };
34. })
36. .directive('pane', function() {
37. return {
38. require: '^tabs',
39. restrict: 'E',
40. transclude: true,
41. scope: { title: '@' },
42. link: function(scope, element, attrs, tabsController) {
43. tabsController.addPane(scope);
44. },
45. template:
46. '<div class="tab-pane" ng-class="{active: selected}" ng-transclude>' +
47. '</div>',
48. replace: true
49. };
50. })

Result



Note

* tabs, pane: sono elementi personalizzati, elementi aggiuntivi HTML creati da noi, per avere una vista più leggibile e una sintassi facilmente riutilizzabile.
* title="Localization": i componenti possono accettare parametri di input sotto forma di attributi HTML. in questo caso title specifica il testo della scheda. Localization mostra: formattazione di data, numero e valuta.
* ng-controller="BeerCounter": utilizziamo il controller Beer Counter per impostare le regole di conteggio in base alle impostazioni locali.
* ng-pluralize: è la direttiva che seleziona il giusto formato plurale per ogni zona, poichè ogni lingua presenterà un diverso modo per specificare il plurale.
* count="beerCount": si lega alla proprietà number. Il count diventa il selettore per la forma plurale da scegliere.
* when="beerForms": si lega alle regole di pluralizzazione. le regole sono univoche per ogni zona e combinazione di lingua.

Accessibilità

* WCAG 2.1, contiene le linee guida che ogni sistema, che ha un interazione con persone con qualche disabilità, dovrebbe soddisfare.
* 4 principi fondamentali:
  + Interfaccia percettibile, con almeno uno dei sensi si deve percepire quello che si sta presentando.
  + Interfaccia fattibile, ovvero le componenti dell’interfaccia utente e della navigazione devono essere usabili dall’utente (grandezza che si legga, spazio giusto tra le varie componenti …)
  + Comprensibile, in modo che l’utente non debba avere particolari capacità per capire l’informazione.
  + Robusto
* la conformità di WCAG può essere classificata in tre diversi livelli, a seconda dei criteri di successo che il sistema soddisfa:
  + A
  + AA
  + AAA

Aria

è un insieme di attributi che descrivono stati o proprietà di elementi che non sono html e non sono rappresentabili in html.

Es:

Se faccio, al posto del tag botton, un bottone con span o div, siccome non esiste l’attributo alt=”” posso usare l’attributo aria per cambiare la semantica di questi .

Quindi da usare quando realizzo cose nuove.

Test automatici

Test che si agganciano ai plugin.

Esame

Database: Noleggio di DVD organizzato su diversi negozi. Un dvd può essere presente in uno o più negozi. Ciascuno ha solo una copia di dvd di un film.

Il database rappresenta tutti i noleggi fatti e correnti di tutti i negozi.

Rent: affitto

Dentro film, rental duration non è del film, ma del dvd.

Tutti i dvd che non hanno la data di ritorno sono quelli che ancora sono nel cliente.

La data di ritorno di un noleggio viene registrata quando viene ritornato.

Layer: deve essere accessibile. O come piano sovrapposto, oppure un div che si espande.

I layer dovrebbero essere usuali, quindi è riusabile.

Utente applicazione diverso dell utente del db. Se magari l’utente va a restituire il dvd in negozio, poi è l’utente del db che registra l’avvenuta ricezione.

Il database dell’applicazione (registrazione utenti) deve essere diverso dal database del dominio (dvd registrati ecc..)

Lato server: express con GraphQL

Lato client: Controller, Model, View con Angular

# 

# Angular Spiegazione prof

Typescript: tentativo di prendere javascript e renderlo tipizzato.

Poichè i linguagi tipizzati come javascript il tipo è sul valore e quindi puù continuamente cambiare valore.

Quindi associa tipi a javascript in modo che già a tempo di compilazione mi accorgo degli errori.

[ ] : tra parentesi quadre, il valore viene specificato da quello dopo l’uguale

[diabled] = “canClick”

[(ngModel)]=”message”

La componente rappresenta il model, ma non il model dell’applicazione.

Gestire gli errori sia sul service che su express .