Angular e il Mondo del Frontend Moderno

Il mercato del frontend si trova oggi diviso tra due grandi poli tecnologici: React, che adotta un approccio flessibile e componibile, e Angular, framework strutturato e opinionato sviluppato da Google.

React vs Angular

Prima di affrontare Angular è necessaria una precisazione terminologica: quando si parla di Angular non ci si riferisce ad AngularJS, la versione precedente del framework ormai deprecata e abbandonata. Si tratta di due prodotti profondamente diversi, e la distinzione è rilevante in qualsiasi contesto professionale.

TypeScript come linguaggio principale

Angular utilizza TypeScript come linguaggio di programmazione principale. TypeScript non viene eseguito direttamente dal browser, ma deve essere prima convertito in JavaScript attraverso un processo chiamato traspilazione. Questo passaggio intermedio è obbligatorio ed avviene in modo automatico durante la fase di build del progetto.

I motivi per cui TypeScript viene preferito a JavaScript standard sono molteplici. Il più rilevante è l'introduzione di un sistema di tipi statici, che consente di rilevare errori durante la fase di sviluppo prima ancora che il codice venga eseguito. A questo si aggiunge il supporto nativo per classi, interfacce e metodi statici, strumenti che aumentano la leggibilità, la manutenibilità e la produttività complessiva dello sviluppatore.

Dall'approccio page-centric all'approccio a componenti

Nello sviluppo web tradizionale si adotta un approccio cosiddetto page-centric, dove ogni pagina HTML incorpora al proprio interno il CSS e il JavaScript necessari al suo funzionamento. Questo modello presenta seri problemi di scalabilità: all'aumentare della complessità dell'applicazione, diventa sempre più difficile gestire e mantenere le risorse distribuite tra le singole pagine.

Angular risolve questo problema attraverso un approccio a componenti. Un componente non è un semplice oggetto: è una unità autonoma e riutilizzabile che rappresenta una porzione specifica dell'interfaccia grafica. Ogni componente è composto da tre elementi distinti che collaborano tra loro: un file HTML che definisce la struttura visiva, un file CSS che ne gestisce lo stile in modo isolato, e un file TypeScript che ne governa il comportamento e la logica. L'applicazione finale nasce dall'assemblaggio di questi componenti, ognuno responsabile di una precisa area funzionale dell'interfaccia.

Un aspetto fondamentale è che il CSS definito all'interno di un componente è locale a quel componente e non influenza il resto dell'applicazione. Questo comportamento, denominato isolation, rappresenta una differenza sostanziale rispetto allo sviluppo web tradizionale, dove le regole CSS hanno portata globale e possono generare conflitti difficili da tracciare.

Lo sviluppo con Angular viene tipicamente effettuato utilizzando Visual Studio Code.

Angular CLI e i comandi fondamentali

Angular mette a disposizione uno strumento da linea di comando chiamato Angular CLI, richiamabile tramite il prefisso `ng`, abbreviazione di Angular. Tramite il comando `ng new` seguito dal nome del progetto è possibile generare un nuovo workspace contenente una applicazione Angular preconfigurata, comprensiva di struttura di cartelle, file di configurazione e dipendenze. Il comando `ng generate component`, abbreviabile in `ng g c`, consente invece di creare un nuovo componente specificandone il nome: Angular CLI genererà automaticamente tutti i file necessari.

Per avviare l'applicazione in fase di sviluppo si utilizza il comando `ng serve`, che compila i file TypeScript e HTML e li serve tramite un server locale accessibile di default sulla porta 4200 all'indirizzo `http://localhost:4200`. Questo comando offre

due funzionalità particolarmente utili: il live-reload, che ricarica automaticamente il browser al rilevamento di modifiche, e il watch, che ricompila il codice in modo continuo ad ogni cambiamento dei file sorgente.

Struttura di un componente e decoratori

La logica di un componente Angular è definita all'interno di una classe TypeScript. Tutte le proprietà e i metodi dichiarati in questa classe diventano disponibili nel relativo template HTML. Per trasformare una classe ordinaria in un componente Angular è necessario importare la classe `Component` dal modulo `@angular/core` e applicare il decoratore `@Component` alla classe stessa.

Il decoratore contiene i metadati che configurano il comportamento del componente. Tra i più importanti vi è il `selector`, che definisce il tag HTML personalizzato con cui il componente verrà richiamato nei template dell'applicazione, ad esempio `app-copyright`. Il prefisso `app` è una convenzione di nomenclatura standard adottata da Angular CLI per identificare i componenti dell'applicazione. L'array `imports` consente invece di dichiarare altri moduli o componenti standalone necessari al funzionamento del componente corrente.

Template e interpolazione

Il template HTML di un componente può accedere esclusivamente alle variabili e ai metodi definiti nella classe TypeScript corrispondente. Il meccanismo con cui i dati vengono proiettati dal TypeScript verso la vista HTML si chiama interpolazione, ed è la forma più semplice di data binding disponibile in Angular.

La sintassi dell'interpolazione prevede di racchiudere un'espressione tra doppie parentesi graffe. Angular valuta quella espressione, converte il risultato in stringa e lo inserisce nel contenuto dell'elemento HTML corrispondente. Quando il valore cambia, il DOM viene aggiornato automaticamente. Si tratta di un processo unidirezionale, che va dal componente verso la vista, motivo per cui viene anche definito one-way data binding di tipo Model-to-View.

All'interno delle doppie parentesi graffe è possibile inserire proprietà del componente, chiamate a metodi, espressioni JavaScript, operatori condizionali ternari, concatenazioni di stringhe e operazioni matematiche.

Struttura del progetto

La cartella radice del progetto, che prende il nome assegnato in fase di creazione, contiene tutte le risorse dell'applicazione. Al suo interno si trovano la cartella `.angular`, la cartella `.vscode`, la cartella `node_modules` con le dipendenze npm, la cartella `public` per le risorse statiche accessibili pubblicamente e la cartella `src`, che rappresenta il cuore dell'applicazione.

All'interno di `src` si trova la cartella `app`, che ospita il componente radice e tutti i componenti custom. Ogni componente custom è organizzato in una sottocartella dedicata contenente i propri file: il CSS per gli stili locali, l'HTML per il template, il file `.spec.ts` per i test unitari e il file `.ts` per la logica TypeScript.

Nella cartella `app` sono presenti anche alcuni file fondamentali per il funzionamento dell'intera applicazione. Il file `app.config.ts` contiene la configurazione principale e registra i provider necessari, tra cui il sistema di routing tramite `provideRouter`. Il file `app.routes.ts` definisce le rotte dell'applicazione, associando i percorsi URL ai rispettivi componenti. Sono inoltre presenti il componente radice con i suoi file associati e il template principale `app.html`.

Alla radice del progetto si trovano infine numerosi file di configurazione tra cui `index.html`, punto di ingresso dell'applicazione, `main.ts` che si occupa del bootstrap del componente principale, `angular.json` per la configurazione del workspace, `package.json` per la gestione delle dipendenze npm, e i file `tsconfig.json` nella loro varie declinazioni per la configurazione del compilatore TypeScript.

Signals

Un Signal è una funzione che avvolge un valore e notifica automaticamente Angular ogni volta che quel valore cambia. I Signal rappresentano il meccanismo con cui Angular gestisce la reattività in modo preciso ed efficiente, permettendo di leggere valori che sono costantemente monitorati dal framework.

Il vantaggio principale rispetto alle proprietà ordinarie riguarda il sistema di change detection. Con le proprietà normali, ad ogni evento Angular attraversa l'intero albero dei componenti per verificare se qualcosa è cambiato, controllando anche componenti che non sono stati toccati, con evidenti inefficienze nelle applicazioni di grandi dimensioni. Con i Signal, invece, Angular conosce esattamente quali componenti e quali porzioni di template dipendono da un

determinato Signal, e aggiorna solo quelle parti quando il valore cambia. Questo comportamento prende il nome di reattività fine-grained.

I Signal si integrano in modo particolarmente efficace con la strategia di rilevamento dei cambiamenti OnPush, permettendo ad Angular di saltare interi sottoalberi di componenti a meno che un Signal rilevante non venga modificato.

Non ogni proprietà deve essere necessariamente un Signal. Una proprietà ordinaria è appropriata quando il suo cambiamento non deve produrre aggiornamenti automatici nella vista. L'uso di un Signal in questi casi introdurrebbe un costo di performance non giustificato. I Signal vanno invece utilizzati per quei valori che richiedono monitoraggio continuo e devono propagarsi automaticamente nell'interfaccia utente, come i valori numerici in un'applicazione di tracking nutrizionale dove i calcoli derivati devono aggiornarsi in tempo reale.

Computed Signals

I Computed Signal sono Signal speciali di sola lettura il cui valore viene calcolato automaticamente a partire da altri Signal. Si creano tramite la funzione `computed()`, alla quale si passa una funzione che definisce come il valore deve essere derivato. Quando uno dei Signal da cui dipende un Computed Signal cambia, Angular ricalcola automaticamente il valore del Computed Signal.

Una caratteristica importante è che i Computed Signal non possono essere modificati direttamente tramite i metodi `set()` o `update()`, poiché il loro valore è sempre il risultato del calcolo definito in fase di creazione. Le dipendenze di un Computed Signal sono inoltre dinamiche: vengono tracciati solo i Signal effettivamente letti durante l'esecuzione della funzione di calcolo, e un Signal diventa dipendenza solo quando la condizione che porta alla sua lettura risulta vera.

Vale la pena distinguere chiaramente i tre tipi di Signal disponibili in Angular. Il Computed Signal non riceve alcun input diretto e deriva il proprio valore automaticamente dagli altri Signal che legge. Il Writable Signal è un Signal scrivibile che può ricevere nuovi valori tramite i metodi `set()` e `update()`. L'Input Signal, infine, riceve il proprio valore attraverso il binding proveniente dal componente padre.

Un evento è qualsiasi cosa che accade all'interno della pagina o del componente, e può essere gestito tramite appositi meccanismi messi a disposizione dal framework.

Nel contesto della programmazione, **delta** (dalla lettera greca Δ) rappresenta generalmente una **differenza** o **variazione** tra due valori.

Una callback è una funzione che viene passata come argomento a un'altra funzione e successivamente invocata all'interno di quest'ultima. Questo meccanismo consente alla funzione chiamante di eseguire un comportamento specifico che non è necessariamente noto nel momento in cui il codice viene scritto, rendendo le callback uno strumento potente per la programmazione asincrona e la gestione degli eventi.

In JavaScript e TypeScript le funzioni sono trattate come tipi di dati a tutti gli effetti, il che significa che possono essere assegnate a variabili, passate come argomenti e restituite come valori di ritorno da altre funzioni. È proprio questa caratteristica del linguaggio che rende possibile il meccanismo delle callback.

Il binding di evento in Angular è il meccanismo che permette di ascoltare e rispondere alle azioni dell'utente all'interno del template HTML. Queste azioni possono essere di natura diversa: un click del mouse, il movimento del cursore, la pressione di un tasto e qualsiasi altra forma di interazione con l'interfaccia.