Corso breve 4h

venerdì 6 dicembre 2019 10:47

Corso breve per la condivisione delle esperienze in Angular.

#	Titolo	Min	Descrizione
0	*Introduzione*	5	
1	Ambiente di sviluppppo (Npm, Angular CLI ..)	5	Tool da avere a disposizione per procedere con lo sviluppo di angular
2	PrimeNg	5	Esempio di struttura applicazione enterprise
3	Ciclo di vita	5	NgOnInit & NgOnChanges
4	Change detection	5	Reference e non valore o contenuto
5	Service provider (Service duplicato)	5	Service, ciclo di vita e visibilità. Evitare duplicati.
6	Debug (Chrome, vsCode, …)	5	Come procedere al debug e con cosa
7	Content projection	5	Inserire blocchi di codice all'interno di un componente riutilizzabile
8	Event emitter	5	Uso degli observable by Angular
9	Styling scoped	5	In che modo interviene lo style di un componente nell'applicazione
10	Router	5	Navigazione su più livelli
11	Guardie routing	5	Protezione della navigazione, permessi ecc… "Dati non salvati"
12	Form	5	Costruzione e validazione di una form "complessa"
13	Control value accessor	5	Creazione di un componente integrato in una form
14	Http Client	5	Chiamate API
15	Interceptor	5	Authentication e Gestione errori chiamate API
16	Global Error Handler	5	Gestione errore globale
17	Shared module e Core module	5	--
18	Logger	5	--
19	Login Service	5	--
20	Token Refresh	5	--

Introduzione

Angular è un framework front-end scritto in Javascript fortemente strutturato e modulare.

E' composto fondamentalmente da:

- Componenti
- Service
- Moduli

Configurazione ambiente di sviluppo

Per lo sviluppo in Angular son necessari I seguenti strumenti:

- ## Node & Npm

  Node.js® è un runtime JavaScript costruito sul motore JavaScript V8 di Chrome. (cit. sito node)
  Npm per noi è essenzialmente un package manager alla stregua di Nuget
  Installando Node si installa automaticamente anche npm, provare da riga di comando:

  ```
  npm --version
  ```

  La presenza di npm all'interno dell'applicazione si "limita" alla presenza del file "package.json" e la cartella "node_modules".

  ### *Il file package.json*

  E' necessario conoscere almeno 3 parti di questo file:

  - scripts
    - Per default sono già presenti alcuni script, ma possono esserne definiti altri
    - Tutti gli script presenti sono eseguiti da Node per default
    - Possono essere tuttavia eseguiti da powershel in inserendo all'inizio della stringa: "@powershell" ad esempio:

      ```
      "prova": "@powershell ls"
      ```

      Stamperà il contenuto della cartella
  - dependencies
    - Contiene tutte le dipendenze che sono necessarie per la build finale dell'applicazione
    - Indica la versione attualmente in uso per ogni singola dipendenza
  - devDependencies
    - Contiene tutte le dipendenze che sono necessarie per lo sviluppo dell'applicazione, come Angular/Cli, Typescript, Tslint …
    - Anche qui indica la versione attualmente in uso per ogni singola dipendenza

  ### *Comandi base*

  In genere l'unico comando che attualmente tendiamo ad usare più spesso è:

  ```
  npm install
  ```

  Con le sue varianti,
  per l'installazione globale:

  ```
  npm install -g
  ```

  per l'installazione di una dipendenza (--save non è strettamente necessario):

  ```
  npm install --save
  ```

  per l'installazione di una dipendenza di sviluppo:

  ```
  npm install --save-dev
  ```

- ## Angular/Cli

  Angular mette a disposizione da linea di comando template di partenza per le app, e comandi per la generazione di componenti, servizi, moduli …
  Per installare la CLI di Angular, è necessario npm ed è sufficiente eseguire il seguente comando:

```
npm install -g @angular/cli
```

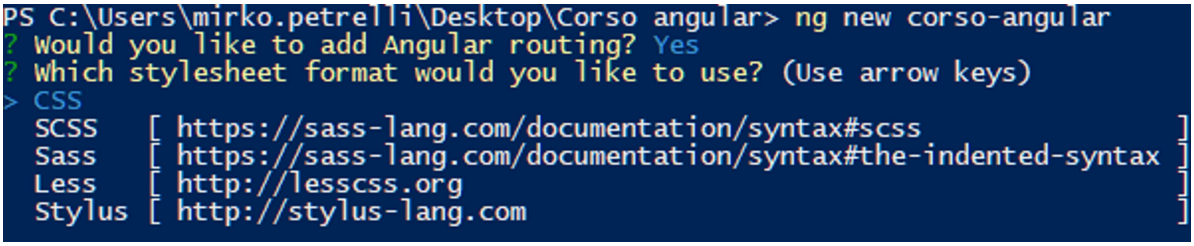
Per verificarne la corretta installazione provare da riga di comando:

```
ng --version
```

Per creare una nuova applicazione eseguire il seguente comando:

```
ng new [nome-applicazione]
```

Es. ng new corso-angular
verrà richiesta la conferma per l'aggiunta di una configurazione base per il routing e il tipo di preprocessore che si intende utilizzare per la generazione del CSS, o di utilizzare semplicemente file di tipo CSS.



Il processo di creazione dell'applicazione di partenza si occuperà anche dell'installazione delle varie dipendenze, che altrimenti sarebbero dovute essere installate tramite il seguente comando:

```
npm install
```

Per avviare l'applicazione si possono eseguire diversi comandi, che di fatto sono lo stesso comando:

`ng serve`	Comando di angular cli per l'avvio dell'applicazione
`ng s`	"s" è un alias per serve (per scrivere meno :-D) aggiungendo "-o" verrà aperto in automatico il browser
`npm start`	Comando per eseguire lo script definito in "package.json" denominato "start"
`npm run start`	Di fatto è lo stesso di prima, solo che al posto di start si può scrivere un qualsiasi altro script definito all'interno del file "package.json"

Una volta avviata l'applicazione, I messaggi sulla riga di comando ci avviseranno dell'ESITO e dell'indirizzo verso cui l'applicazione è fruibile.

Angular Cli gestisce l'applicazione facendo uso della configurazione fornita nel file "angular.json".
Ad esempio, l'aggiunta di altre librerie esterne, di fogli di stile e degli assets va indicata all'interno di questo file invece che nella "index.html".
Vedremo poi nell'esempio più avanti.

- Visual studio code
  Con le seguenti estensioni:
  - Angular language service
  - Angular schematics (Utile ma non indispensabile)
  - Scss formatter (Utile ma non indispensabile)
  - Scss IntelliSense (Utile ma non indispensabile)

- Chrome
  Con la seguente estensione:
  - Augury(Utile ma non indispensabile)

PrimeNg

Per un esempio pratico aggiungeremo all'applicazione di partenza, I componenti e lo stile di PrimeNg.

Ecco I comandi da eseguire:

```
npm install primeng --save
npm install primeicons --save
```

Dato che primeNg fa uso delle animazioni di Angular, va installato anche il seguente pacchetto:

```
npm install @angular/animations --save
```

Sembra essere necessaria anche la seguente dipendenza:

```
npm install @angular/cdk
```

Fatto ciò, vanno Inclusi I fogli di stile tramite "angular.json":

```
"styles": [
  "src/styles.scss",
  "node_modules/primeng/resources/primeng.min.css",
  "node_modules/primeng/resources/themes/nova-light/theme.css",
  "node_modules/primeicons/primeicons.css"
]
```

Component life cycle

Scattano nel seguente ordine:

Nome	Descrizione
ngOnChanges	Scatta al cambio* di uno degli input del componente
ngOnInit	Chiamato una sola volta Inizializza il componente
ngDoCheck	Aggiunge un controllo personalizzato Da non usare insieme a ngOnChanges
ngAfterContentInit	Dopo la renderizzazione dei template passati all'interno del tag del componente Ossia le proiezioni Chiamata una sola volta dopo ngDoCheck
ngAfterContentChecked	Chiamata la prima volta dopo ngAfterContentInit e ogni volta dopo ngDoCheck
ngAfterViewInit	Dopo che Angular inizializza I componenti nella view Chiamata una sola volta dopo ngAfterContentChecked
ngAfterViewChecked	Dopo il controllo dei componenti nella view Chiamata ogni volta dopo ngAfterContentChecked
ngOnDestroy	Effettua la pulizia della memoria occupata dal componente Chiamata una sola volta prima della distruzione del componente

Change detection

E' il processo che usa Angular per rendere i propri componenti reattivi al cambiamento dei dati.

1. Ad ogni occorrenza di evento asincrono viene triggerata la change detection
2. La change detection viene eseguita a partire dalla root verso il basso come flusso unidirectional top-down.
3. Per ogni view della gerarchia si esegue il check della CD che comporta le seguenti operazioni:
    - si valuta se e' necessario modificare il DOM del componente a causa dell evento accaduto al punto 1. Riducendo all'osso il concetto, per valutare le modifiche durante la CD di una view si valuta le espressioni presenti nel template e si confronta i precedenti valori con i nuovi al fine di capire se occorre ri-disegnare la view.
    - si controlla e aggiorna le input properties sui child se necessario
    - si invoca i metodi del lifecycle di un componente sul child e sul componente stesso: come ad es. onChanges su un child se almeno un input passato al figlio e' variato
    - si aggiorna il DOM se e' necessario
4. Questa operazione di check si esegue **sempre e ad ogni ciclo di CD** se si fa uso della **strategy default** in quella view (strategia di default utilizzata da Angular se nessuna strategy viene specificata dal programmatore nel componente). Cosa accade se si usa la strategy on Push ve lo dico a breve, ma per ora vi basti sapere che in determinate situazioni con la strategy on push e' possibile skippare il processo di check.

Service provider

Un providere è l'istruzione per la Dependency Injection su come ottenere un valore per una determinata dipendenza, ossia in questo caso un Service.

Per definire questa istruzioni si possono seguire 3 modi:

- All'interno dell'annotazione *@Injectable*
  per far si che il service sia visibile a tutta l'applicazion con la property
  `provideIn:'root'`
  o definendo il modulo per cui il service deve essere disponibile
  `provideIn:NomeModulo`

- All'interno dell'array *providers* del modulo in cui il service deve essere disponibile

- All'interno dell'array *providers* del componente in cui il service deve essere disponibile

Una modalità non esclude necessariamente le altre, ma usandone più di una si crea terreno fertile per futuri bug e/o comportamenti dell'applicazione imprevedibili.

Content projection

E' un modo per importare del codice HTML dall'esterno del componente e posizionarlo all'interno del componente al posto di appositi segnaposto.

```
<ng-content></ng-content>
```

Ci possono essere più segna posti all'interno dello stesso componente.
Ma se non sono distinguibili, il codice HTML passato dall'esterno, sarà posizionato sull'ultimo segnaposto incontrato.

Per rendere univoci questi segna posti ci sono diversi modi:

- Attributo	select="[nome-selettore][nome-selettore2]"
- Componente	select="component-name"
- Classe	select=".class,.class2"
- Misto	select=".class,[nome-selettore]"

Event Emitter

Wrapper degli Observable di Angular utilizzato per emettere eventi sincroni o asincroni.

Utilizzato principalmente in due occasioni

- Output dei componenti
- Eventi "Applicativi"

Router

Angular router è necessario per simulare la navigazione all'interno della SPA.

La base di partenza per il routing è indicata dal seguente tag:

```
<base href="/">
```

Il modulo da importare per configurare il routing è il seguente:

```
import { RouterModule, Routes } from '@angular/router';
```

Routes -> è la classe per la definizione di un array di Route.

Si possono definire:

- Rotte specifiche
- Redirect
- Rotte con wildCard
- Rotte con parametri, query, data, guardie

L'ordine di definizione delle rotte è fondamentale, in quanto il controllo di corrispondenza con l'URL immesso avviene sequenzialmente.
"*Ossia il primo buono che trovo uso*".

Per controllare il routing nel template, Angular predisone due oggetti:

```
<router-outlet></router-outlet>                          -> Dove avviene la navigazione
<a routerLink="/[URL]" routerLinkActive="active">Nome link</a>   -> Link, pulsante ecc per la navigazione
```

ActivatedRoute

Angular mette a disposizione un servizio richiamabile da qualsiasi componente, utile per accedere allo stato attuale del routing, *Data*, *parametri* e *query* comprese!

Per un applicazione semplice è sufficiente definire la configurazione del routing all'interno di un modulo e instanziare il RouterModule con la seguente forma all'interno dell'array di import:

```
RouterModule.forRoot(routes)
```

In applicazioni più complesse, o genericamente sempre, è un approccio migliore quello di definire un modulo per il routing. Per mantere il codice, per la pulizia del codice e addirittura per I test.
Come dice Angular: ***Scegli un modo e sii coerente!!***

Forms

Per l'utilizzo delle form template-driven va importato il modulo FormsModule, mentre per le reactive-forms va importato il modulo ReactivFormsModule.