Calendario Corso

Lunedì 6

Ore 9:00 - 9:30:
Introduzione al corso Angular e TypeScript

Ore 9:30 - 11:00: Introduzione a TypeScript - tipi di dati, variabili, costrutti di controllo, funzioni.

Ore 11:00 - 12:00:
Configurazione dell'ambiente di sviluppo e installazione degli strumenti necessari

Ore 12:00 - 13:00:
Creazione di componenti in Angular

Ore 14:00 - 15:00:
Comunicazione tra componenti in Angular

Ore 15:00 - 16:00:
Uso di moduli e servizi in Angular.

Ore 16:00 - 18:00:
Esercizio pratico: creare una mini app

Martedì 7

Ore 9:00 - 10:00:
Introduzione alla componentizzazione in Angular

Ore 10:00 - 13:00:
Componenti dinamici e riutilizzabili, Material design

Ore 14:00 - 16:00:
Gestione degli eventi e delle animazioni in Angular

Ore 16:00 - 18:00:
Esercizio pratico: Creazione di un'applicazione semplice con componenti dinamici

Mercoledì 8

Ore 9:00 - 10:00:
Introduzione ai modelli di dati in Angular

Ore 10:00 - 12:00:
Utilizzo di form e validazione dei dati in Angular

Ore 12:00 - 13:00:
Utilizzo di Pipes e Direttive in Angular

Ore 14:00 - 18:00:
Esercizio pratico: creare una semplice applicazione che utilizza i form e la validazione

Giovedì 9

Ore 9:00 - 10.00:
Routing e navigazione

Ore 10:00 - 11:00:
Utilizzo di API esterne con Service

Ore 11:00 - 13:00:
Autenticazione e Guards

Ore 14:00 - 18:00:
Esercizio pratico: creare un'applicazione che utilizza API esterne

Venerdì 10

Ore 9:00 - 10:00:
Introduzione ai concetti avanzati di Angular

Ore 10:00 - 12:00:
Creazione di Directive e Pipes dinamici e riutilizzabili

Ore 12:00 - 13:00:
Debugging e risoluzione dei problemi in Angular

Ore 14:00 - 16:00:
Persistenza dei dati e state management

Ore 16:00 - 18:00:
Esercizio pratico: creazione di Directives e Pipes personalizzate e debug dell'applicazione

Lunedì 13

Ore 9:00 - 10:00:
Panoramica dei principali strumenti e librerie utilizzati con Angular

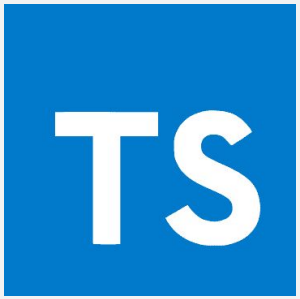
Ore 10:00 - 11:00:
Implementazione di test automatici

Ore 11:00 - 12:00:
Scrittura di test per i progetti svolti

Ore 12:00 - 13:00:
Sistemi di build e distribuzione automatizzata dei progetti

Ore 14:00 - 18:00:
Test di conoscenze Angular

Corso Angular e Typescript



ITSVIL INGEGNERIA INFORMATICA

e:learning YOUR WIDE LEARNING

FONDAZIONE SACCONE

Corso Angular e Typescript

Attilio Cianci

CTO & Full Stack Developer @ Armonia

https://github.com/zorahrel
https://www.linkedin.com/in/attiliocianci

virvelle

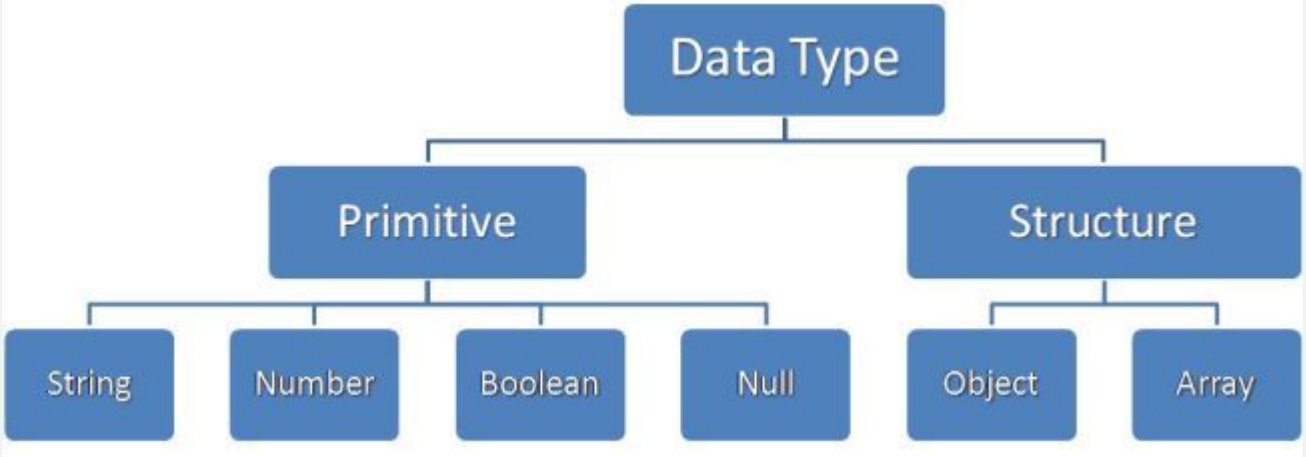
Programma del corso

- ➔ Introduzione ad Angular e TypeScript
- ➔ Struttura di un'applicazione Angular
- ➔ Visualizzazione dei dati
- ➔ Creazione di componenti dinamici e riutilizzabili
- ➔ Comunicazione tra componenti
- ➔ Directives e Pipes
- ➔ Forms e validazione dei dati
- ➔ Routing e navigazione in Angular
- ➔ Utilizzo di API esterne
- ➔ Persistenza e state management
- ➔ Implementazione di test automatici
- ➔ Autenticazione e Guards
- ➔ Sistemi di build e distribuzione automatizzata

Requisiti

- Conoscenza HTML
- Conoscenza CSS
- Conoscenza base Javascript
- Conoscenza JSON
- Conoscenza fondamenti programmazione

Tipi JSON



Cheatsheet HTML

➔ https://developer.mozilla.org/en-US/docs/Learn/HTML/Cheatsheet

Conosciamoci!

Canale Telegram per lo scambio risorse

➔ https://t.me/+qAbdIaMbTZ9iNDE8

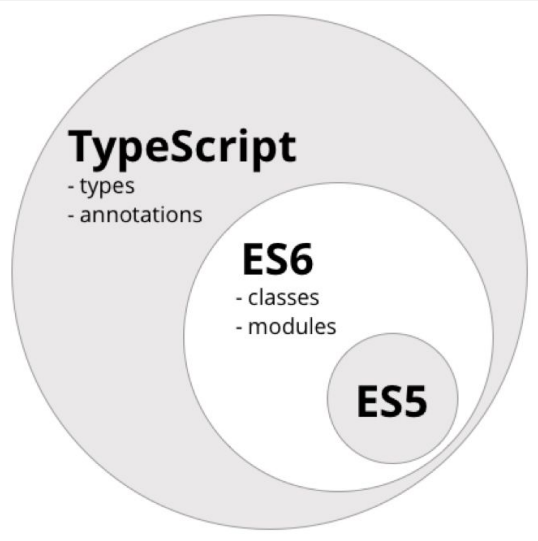
Write code for humans!
Not for machines

➔ https://slides.mattianatali.it/code-for-humans

Typescript: il super set di Javascript

- ➔ Tipi dinamici
- ➔ Funzionalità ES6
- ➔ Scalabilità del codice
- ➔ Moduli, classi, interfacce ed enumeratori
- ➔ Supporto a tools di linting e intellisense

https://ponyfoo.com/articles/es6



TypeScript Cheat Sheet

Setup

Install TS globally on your machine

```
$ npm i -g typescript
```

Check version

```
$ tsc -v
```

Create the tsconfig.json file

```
$ tsc --init
```

Set the root (to compile TS files from) and output (for the compiled JS files) directories in tsconfig.json

```
"rootDir": "./src",
"outDir": "./public",
```

Compiling

Compile a specified TS file into a JS file of the same name, into the same directory (i.e. index.ts to index.js).

```
$ tsc index.ts
```

Tell tsc to compile specified file whenever a change is saved by adding the watch flag (-w)

```
$ tsc index.ts -w
```

Compile specified file into specified output file

```
$ tsc index.ts --outfile out/script.js
```

If no file is specified, tsc will compile all TS files in the "rootDir" and output in the "outDir". Add -w to watch for changes.

```
$ tsc -w
```

Strict Mode

In tsconfig.json, it is recommended to set strict to true. One helpful feature of strict mode is No Implicit Any:

```
// Error: Parameter 'a' implicitly has an 'any' type
function logName(a) {
  console.log(a.name);
}
```

By @DoableDanny

Primitive Types

There are 7 primitive types in JS: string, number, bigInt, boolean, undefined, null, symbol.

Explicit type annotation

```
let firstname: string = 'Danny'
```

If we assign a value (as above), we don't need to state the type - TS will infer it ("implicit type annotation")

```
let firstname = 'Danny'
```

Union Types

A variable that can be assigned more than one type

```
let age: number | string;
age = 26;
age = "26";
```

Dynamic Types

The any type basically reverts TS back to JS.

```
let age: any = 100;
age = true;
```

Literal Types

We can refer to specific strings & numbers in type positions

```
let direction: 'UP' | 'DOWN';
direction = 'UP';
```

Objects

Objects in TS must have all the correct properties & value types

```
let person: {
 name: string;
 isProgrammer: boolean;
};

person = {
 name: 'Danny',
 isProgrammer: true,
};

person.age = 26; // Error - no age prop on person object

person.isProgrammer = 'yes'; // Error - should be boolean
```

Arrays

We can define what kind of data an array can contain

```
let ids: number[] = [];
ids.push(1);
ids.push("2"); // Error
```

Use a union type for arrays with multiple types

```
let options: (string | number)[];
options = [10, 'UP'];
```

If a value is assigned, TS will infer the types in the array.

```
let person = ['Delia', 48];
person[0] = true; // Error - only strings or numbers allowed
```

Tuples

A tuple is a special type of array with fixed size & known data types at each index. They're stricter than regular arrays.

```
let options: [string, number];
options = ['UP', 10];
```

Functions

We can define the types of the arguments, and the return type. Below, : string could be omitted because TS would infer the return type.

```
function circle(diam: number): string {
 return 'Circumf = ' + Math.PI * diam;
}
```

The same function as an ES6 arrow

```
const circle = (diam: number): string => 'Circumf = ' + Math.PI * diam;
```

If we want to declare a function, but not define it, use a function signature

```
let sayHi: (name: string) => void;

sayHi = (name: string) =>
 console.log('Hi ' + name);

sayHi('Danny'); // Hi Danny
```

Type Aliases

Allow you to create a new name for an existing type. They can help to reduce code duplication. They're similar to interfaces, but can also describe primitive types.

```
type StringOrNum = string | number;
let id: StringOrNum = 24;
```

Interfaces

Interfaces are used to describe objects. Interfaces can always be reopened & extended, unlike Type Aliases. Notice that `name` is `readonly`

```
interface Person {
 name: string;
 isProgrammer: boolean;
}

let p1: Person = {
 name: 'Delia',
 isProgrammer: false,
};

p1.name = 'Del'; // Error - read only
```

Two ways to describe a function in an interface

```
interface Speech {
 sayHi(name: string): string;
 sayBye: (name: string) => string;
}

let speech: Speech = {
 sayHi: function (name: string) {
  return 'Hi ' + name;
 },
 sayBye: (name: string) => 'Bye ' + name,
};
```

Extending an interface

```
interface Animal {
 name: string;
}

interface Dog extends Animal {
 breed: string;
}
```

The DOM & Type Casting

TS doesn't have access to the DOM, so use the non-null operator, !, to tell TS the expression isn't null or undefined

```
const link = document.querySelector('a')!;
```

If an element is selected by id or class, we need to tell TS what type of element it is via Type Casting

```
const form = document.getElementById('signup-form') as HTMLFormElement;
```

Generics

Generics allow for type safety in components where the arguments & return types are unkown ahead of time.

```
interface HasLength {
 length: number;
}

// logLength accepts all types with a length property
const logLength = <T extends HasLength>(a: T) => {
 console.log(a.length);
};

// TS "captures" the type implicitly
logLength('Hello'); // 5

// Can also explicitly pass the type to T
logLength<number[]>([1, 2, 3]); // 3
```

Declare a type, T, which can change in your interface.

```
interface Dog<T> {
 breed: string;
 treats: T;
}

// We have to pass in a type argument
let labrador: Dog<string> = {
 breed: 'labrador',
 treats: 'chew sticks, tripe',
};

let scottieDog: Dog<string[]> = {
 breed: 'scottish terrier',
 treats: ['turkey', 'haggis'],
};
```

Enums

A set of related values, as a set of descriptive constants

```
enum ResourceType {
 BOOK,
 FILE,
 FILM,
}
ResourceType.BOOK; // 0
ResourceType.FILE; // 1
```

Narrowing

Occurs when a variable moves from a less precise type to a more precise type

```
let age = getUserAge();
age // string | number

if (typeof age === 'string') {
 age; // string
}
```

Typescript: il super set di Javascript

Apri sul tuo computer:

➔ https://typescriptlang.org/play

Angular: un framework in Typescript

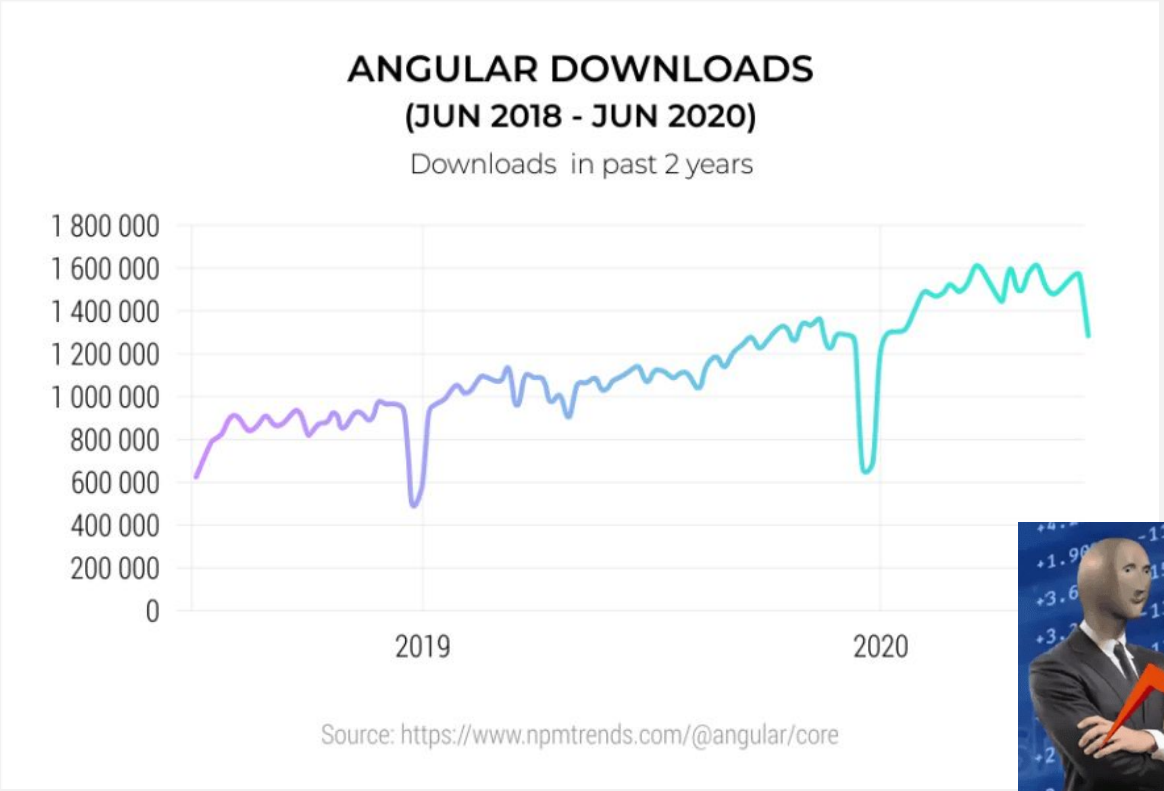
- ➔ Creazione di interfacce utente
- ➔ Generazione codice da CLI
- ➔ Web server di sviluppo
- ➔ Test runner
- ➔ Sviluppato da Google come successore di AngularJS
- ➔ Trasformazione in App Mobile
- ➔ Ecosistema e community attiva

ANGULAR VERSIONS OVER TIME

ANGULAR COMMUNITY IN A NUTSHELL

62.8K	219K	422K	362,8K	95,4K
stars	unique domains	websites built with Angular	followers	subscribers

Source: own graph



Risorse

- https://angular.io/docs
- https://github.com/PatrickJS/awesome-angular
- https://angular.io/guide/cheatsheet
- https://github.com/delprzemo/angular-cheatsheet

Strumenti di sviluppo

Node

➔ https://nodejs.org/en/

Code Editor

➔ https://code.visualstudio.com/

Si inizia!

Avviare un nuovo progetto

Dalla console spostarsi nella cartella in cui volete creare il nuovo progetto e lanciare i comandi:

- ➔ *npm -v* // per verificare l'installazione di Node e NPM
- ➔ *npm install -g @angular/cli*
- ➔ *ng new nome-progetto*

https://angular.io/guide/setup-local

https://angular.io/cli

Hands on keyboard

NgModule

- **Declarations** contiene l'elenco dei componenti, direttive e pipe che appartengono al modulo.
- **Imports** è un array contenente i nomi di altri moduli le cui classi, che sono state esportate, sono impiegate nei template dei componenti dichiarati nel modulo corrente.
- **Exports** definisce quali degli elementi del modulo devono essere visibili all'esterno in modo da renderli disponibili ed utilizzabili all'interno dei template dei componenti appartenenti ad altri moduli. Un modulo può tranquillamente esportarne un altro senza nemmeno importarlo (un modulo può infatti avere altri moduli nell'array 'exports' che non devono necessariamente essere presenti nell'array 'import'), ovviamente dovremo importare la classe richiesta attraverso la parola chiave 'import' messa a disposizione da TypeScript.
- **Providers** definisce l'elenco dei provider per la registrazione dei servizi definiti nel modulo corrente. La visibilità del servizio cambia a seconda della strategia di caricamento del modulo. (Approfondiremo questo argomento al termine della lezione)
- **Bootstrap** definisce qual è il componente root dell'applicazione. Questa proprietà è presente solo in AppModule.

Template componente

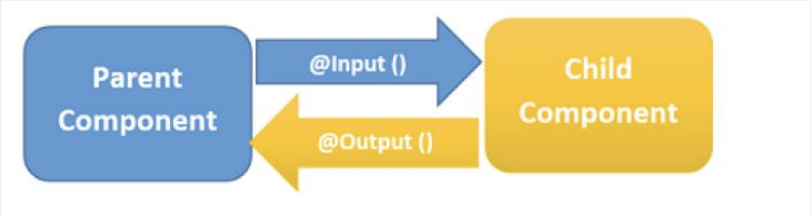
- [attributo]=”calcolo()”
- template tags
- ngFor
- ngSwitch
- ngIf
- ngStyle
- ngClass

https://angular.io/guide/built-in-directives

Logica componente

- ➔ Variabili
- ➔ Comunicazioni con l'esterno
- ➔ LifeCycle

https://angular.io/guide/inputs-outputs
https://angular.io/guide/lifecycle-hooks



Service

➔ ng generate service TestService

Esercitazione 1

- Creare un nuovo progetto Angular utilizzando il comando **ng new nome-progetto**
- Aggiungere un nuovo componente utilizzando il comando **ng generate component nome-componente**
- Creare un servizio dati demo utilizzando il comando **ng generate service nome-servizio**
- Realizzare una mini-app che usa:
  - un servizio per ottenere e mostrare i dati
  - un contatore che con 3 tasti (mostrati con stesso componente personalizzato) incrementa, decrementa o resetta il contatore, mostrando il valore del conto attuale

Esercitazione 2

- Realizzare una mini-app con Material Design che usa:
  - un servizio per ottenere e mostrare i dati:

    copiare JSON da https://dummyjson.com/products
  - una serie di componenti che si occupano di mostrare una parte delle informazioni dei prodotti a comparsa con una animazione

Animazioni in Angular

➔ Definizione animazione

```
@Component({
  selector: 'app-open-close',
  animations: [
    trigger('openClose', [
      // ...
      state('open', style({
        height: '200px',
        opacity: 1,
        backgroundColor: 'yellow'
      })),
      state('closed', style({
        height: '100px',
        opacity: 0.8,
        backgroundColor: 'blue'
      })),
      transition('open => closed', [
        animate('1s')
      ]),
      transition('closed => open', [
        animate('0.5s')
      ]),
    ]),
  ],
  templateUrl: 'open-close.component.html',
  styleUrls: ['open-close.component.css']
})
```

Animazioni in Angular

- ➔ Definizione animazione
- ➔ Associazione animazione

```
<div [@openClose]="isOpen ? 'open' : 'closed'" class="open-close-container
  <p>The box is now {{ isOpen ? 'Open' : 'Closed' }}!</p>
</div>
```

Input Dati

Proprietà dei componenti

➔ https://angular.io/guide/component-interaction

Contenuto dei componenti

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-zippy-basic',
  template: `
    <h2>Single-slot content projection</h2>
    <ng-content></ng-content>
  `
})
export class ZippyBasicComponent {}
```

```
<app-zippy-basic>
  <p>Is content projection cool?</p>
</app-zippy-basic>
```

Eventi in Angular

Built-in events

- (click): viene attivato quando l'utente fa clic su un elemento
- (dblclick): viene attivato quando l'utente fa doppio clic su un elemento
- (mouseover): viene attivato quando l'utente posiziona il cursore del mouse sopra un elemento
- (mouseout): viene attivato quando l'utente sposta il cursore del mouse fuori da un elemento
- (keyup): viene attivato quando l'utente rilascia un tasto sulla tastiera
- (keydown): viene attivato quando l'utente tiene premuto un tasto sulla tastiera
- (submit): viene attivato quando l'utente invia un modulo
- (input): viene attivato quando l'utente inserisce o modifica un valore in un campo di input

Eventi in Angular

Built-in events

- (click): viene attivato quando l'utente fa clic su un elemento

```
<button (click)="doSomething()">Clicca qui</button>
```

to

- (mouseout): viene attivato quando l'utente sposta il cursore del mouse fuori da un elemento
- (keyup): viene attivato quando l'utente rilascia un tasto sulla tastiera
- (keydown): viene attivato quando l'utente tiene premuto un tasto sulla tastiera
- (submit): viene attivato quando l'utente invia un modulo
- (input): viene attivato quando l'utente inserisce o modifica un valore in un campo di input

Eventi in Angular

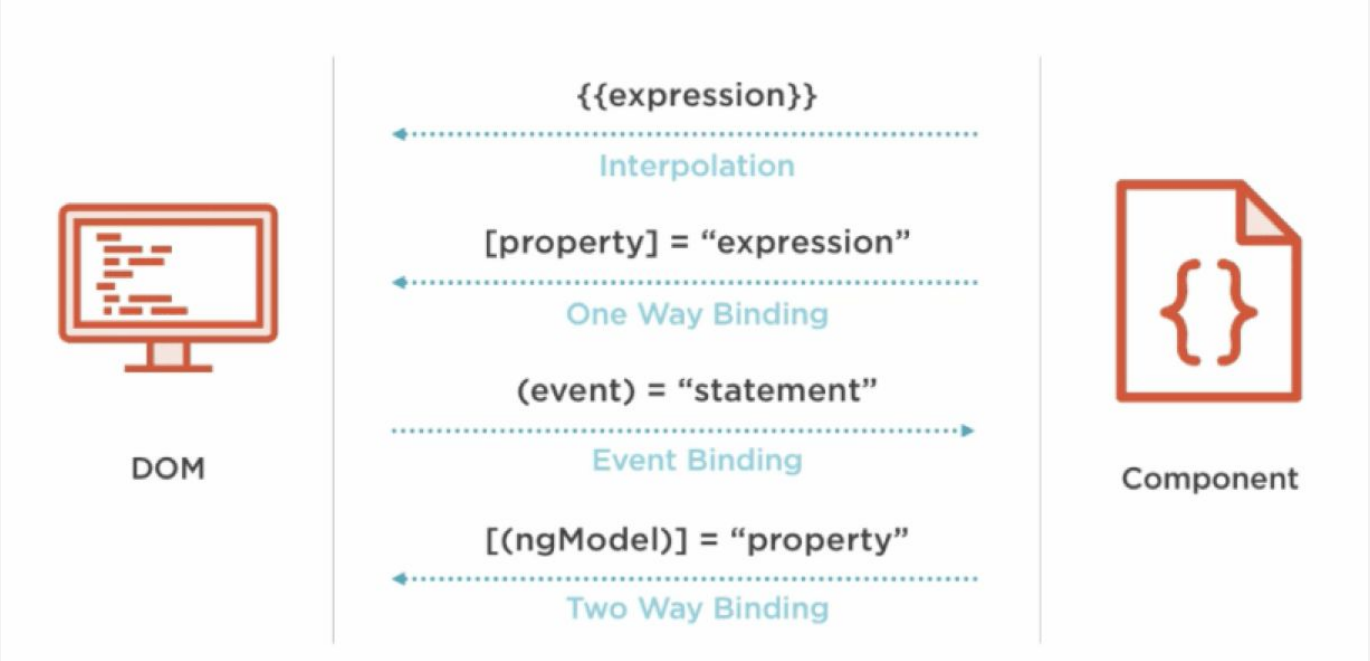
Built-in events

- (nomeEvento)="nomeMetodo()"

Custom Events

- (voted)="onVoted($event)"
- @Output() nomeEvento = new EventEmitter<TipoDati>();
  - this.nomeEvento.emit(dati)

Binding dei dati



```
import {Component} from '@angular/core';

@Component({
  selector: 'example-app',
  template: `
    <input [(ngModel)]="name" #ctrl="ngModel" required>

    <p>Value: {{ name }}</p>
    <p>Valid: {{ ctrl.valid }}</p>

    <button (click)="setValue()">Set value</button>
  `,
})
export class SimpleNgModelComp {
  name: string = '';

  setValue() {
    this.name = 'Nancy';
  }
}
```

Form in Angular

- Model based

  https://angular.io/guide/forms

- Reactive Forms

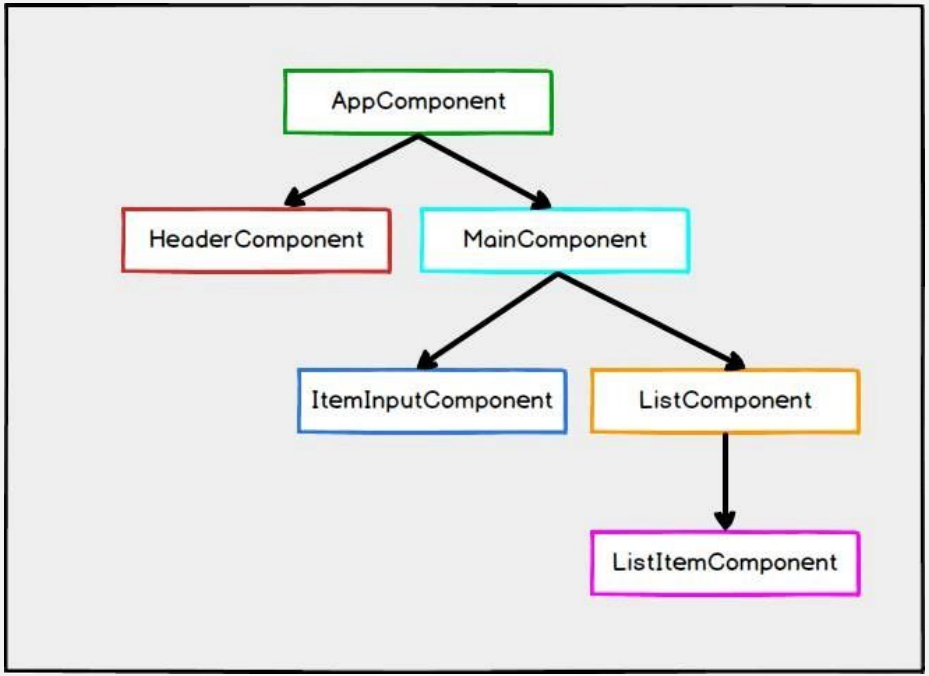
  https://stackblitz.com/edit/angular-reactive-forms

```
import {Component} from '@angular/core';
import {NgForm} from '@angular/forms';

@Component({
  selector: 'example-app',
  template: `
    <form #f="ngForm" (ngSubmit)="onSubmit(f)" novalidate>
      <input name="first" ngModel required #first="ngModel">
      <input name="last" ngModel>
      <button>Submit</button>
    </form>

    <p>First name value: {{ first.value }}</p>
    <p>First name valid: {{ first.valid }}</p>
    <p>Form value: {{ f.value | json }}</p>
    <p>Form valid: {{ f.valid }}</p>
  `,
})
export class SimpleFormComp {
  onSubmit(f: NgForm) {
    console.log(f.value);  // { first: '', last: '' }
    console.log(f.valid);  // false
  }
}
```

Componentizzazione in Angular



Librerie già pronte

- https://material.angularjs.org/
- https://ng-bootstrap.github.io/
- https://primeng.org/

Esercitazione 3

- Realizzare una mini-app ToDo List con Material Design che usa:
  - un servizio per conservare dei Task
  - un componente TaskList per mostrare i TaskItem
  - un componente TaskInput per creare un Task
  - BONUS: fai in modo che TaskInput gestisca anche la modifica dei task

Esercitazione a lezione 3

- Realizzare una mini-app ToBuy List che usa:
  - un servizio per conservare degli oggetti
  - un componente ItemList per mostrare gli Item
  - un componente Item che conserva la quantità
  - un componente ItemInput per creare un Item con un form
  - BONUS: fai in modo che ItemInput gestisca anche la modifica degli Item

Angular Router

The Angular Router enables navigation from one view to the next as users perform application tasks.

Sample routing ts file

```
const appRoutes: Routes = [
  { path: 'crisis-center', component: CrisisListComponent },
  { path: 'hero/:id',      component: HeroDetailComponent },
  {
    path: 'heroes',
    component: HeroListComponent,
    data: { title: 'Heroes List' }
  },
  { path: '',
    redirectTo: '/heroes',
    pathMatch: 'full'
  },
  { path: '**', component: PageNotFoundComponent }
];
```

Then this should be added inside Angular.module imports

```
RouterModule.forRoot(appRoutes)
```

You can also turn on console tracking for your routing by adding enableTracing

```
imports: [
   RouterModule.forRoot(
     routes,
     {enableTracing: true}
    )
],
```

Punto di partenza

```
<router-outlet></router-outlet>
```

Angular Router

The Angular Router enables navigation from one view to the next as users perform application tasks.

Sample routing ts file

```
const appRoutes: Routes = [
  { path: 'crisis-center', component: CrisisListComponent },
  { path: 'hero/:id',      component: HeroDetailComponent },
  {
    path: 'heroes',
    component: HeroListComponent,
    data: { title: 'Heroes List' }
  },
  { path: '',
    redirectTo: '/heroes',
    pathMatch: 'full'
  },
  { path: '**', component: PageNotFoundComponent }
];
```

Then this should be added inside Angular.module imports

```
RouterModule.forRoot(appRoutes)
```

You can also turn on console tracking for your routing by adding enableTracing

```
imports: [
   RouterModule.forRoot(
     routes,
     {enableTracing: true}
    )
],
```

Utilizzo

Usage

```
<a routerLink="/crisis-center" routerLinkActive="active">Crisis Center</a>
```

routerLinkActive="active" will add active class to element when the link's route becomes active

```
//Navigate from code
this.router.navigate(['/heroes']);

// with parameters
this.router.navigate(['/heroes', { id: heroId, foo: 'foo' }]);

// Receive parameters without Observable
let id = this.route.snapshot.paramMap.get('id');
```

Form in Angular

- Model based

  https://angular.io/guide/forms

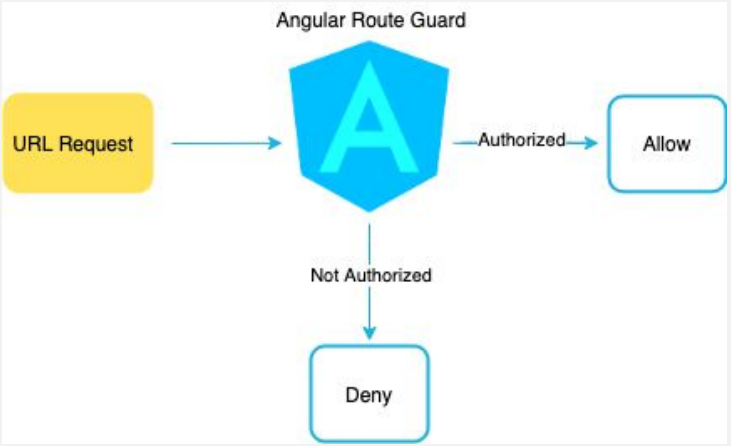
- Reactive Forms

  https://stackblitz.com/edit/angular-reactive-forms

Pipes

➔ https://cheatography.com/nathane2005/cheat-sheets/angular2-pipes/

Guards



We will simulate this via a mock `UserService` like so:

TypeScript Copy

```
class UserService {
  isLoggedIn(): boolean {
    return false;
  }
}
```

This service has one function `isLoggedIn()` which always returns `false`.

Let's create another guard called `OnlyLoggedInUsersGuard` which only allows logged in users to view a route.

TypeScript

```
@Injectable()
class OnlyLoggedInUsersGuard implements CanActivate { (1)
  constructor(private userService: UserService) {}; (2)

  canActivate() {
    console.log("OnlyLoggedInUsers");
    if (this.userService.isLoggedIn()) { (3)
      return true;
    } else {
      window.alert("You don't have permission to view this page"); (4)
      return false;
    }
  }
}
```

1. We create a new `CanActivate` guard called `OnlyLoggedInUsersGuard`
2. We inject and store `UserService` into the constructor for our class.
3. If the user is logged in the guard passes and lets the user through.
4. If the user is *not* logged in the guard fails, we show the user an alert and the page doesn't navigate to the new URL.

Esercitazione 4

- Realizzare una mini-app di User Management con Material Design che usa:
  - un servizio per scaricare e caricare user dalle API
  - una lista degli utenti
  - la creazione degli utenti
  - BONUS: fai in modo che il form di creazione gestisca anche la modifica

Esempi di Pipes

```typescript
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'capitalization'
})
export class CapitalizationPipe implements PipeTransform {
  transform(value: string, format: 'lower' | 'upper' = 'lower'): string {
    return format === 'lower' ? value.toLowerCase() : value.toUpperCase();
  }
}

@Pipe({
  name: 'truncate'
})
export class TruncatePipe implements PipeTransform {
  transform(value: string, limit: number): string {
    return value.length > limit ? value.substring(0, limit) + '...' : value;
  }
}

@Pipe({
  name: 'replace'
})
export class ReplacePipe implements PipeTransform {
  transform(value: string, searchValue: string, replaceValue: string): string {
    return value.replace(searchValue, replaceValue);
  }
}
```

Debugging con Angular

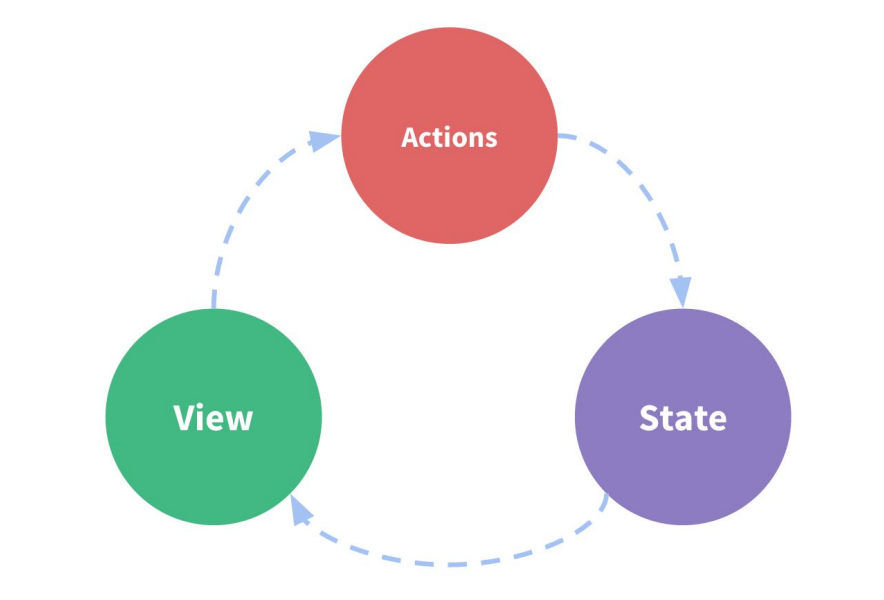
console.log o debugger?

Persistenza dei dati e state management

➔ https://ngrx.io/guide/store

➔ https://github.com/btroncone/ngrx-store-localstorage

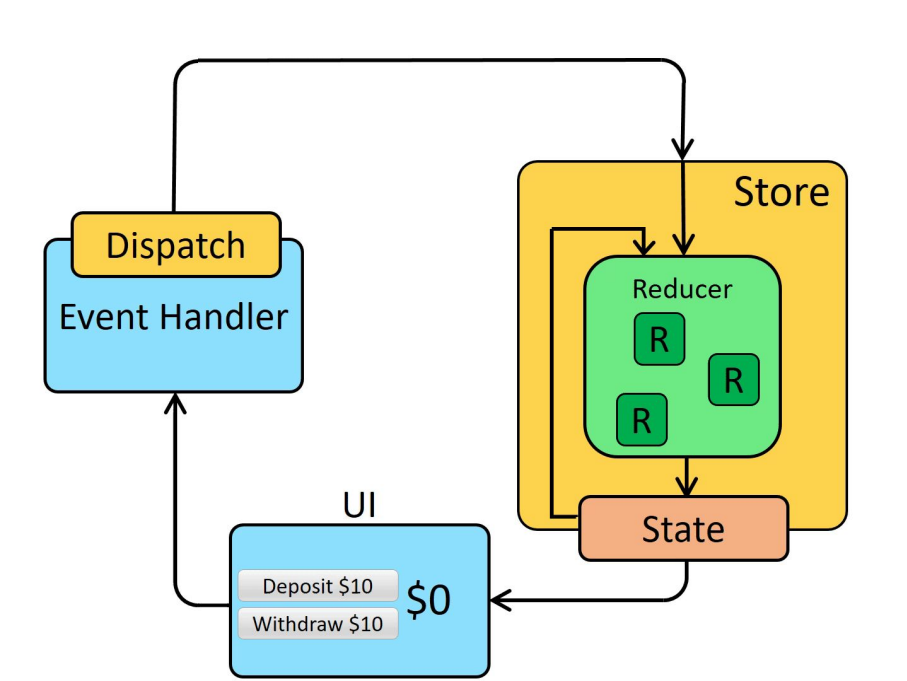
Persistenza dei dati e state management



Perchè usare lo state management?

- → Gestione centralizzata dello stato dell'applicazione
- → Immutabilità dello stato
- → Debugging Time Travel
- → Architettura basata su Redux
- → Ottimizzazione delle performance con la memoization

Perchè usare lo state management?



Cookies vs LocalStorage vs SessionStorage

	Cookies	LocalStorage	SessionStorage
Capacity	4kb	5-10 Mbs(Browser Dependent)	5 Mbs
Accessibility	All windows	All windows	Private to tab
Expiration	Manually Set	Never expires	On tab close.
Passed in request	Yes	No	No
Storage	Browser and Server	Browser Only	Browser Only

Esercitazione 5

- ➔ Realizzare e testare delle Pipes
- ➔ Realizzare e testare delle Directives
- ➔ Integrare lo storage nell'app di Todo

Test di conoscenze

- Realizzare una mini-app di gestione di un Blog con Material Design che gestisce:
  - Utenti
  - Post e commenti degli utenti