Angular

- Una applicazione di esempio
- Applicazione (@Component), Modulo (@NgModule)
- Componenti, Model (relativi a Component), Direttive
- Template Driven Form, Reactive Form
- Servizio
- Routing
- Repository di riferimento
  - https://github.com/egalli64/ngi

Angular

- Framework per lo sviluppo di webapp basato su NodeJS e TypeScript
  - Google 2016
  - https://angular.io/
  - “One framework. Mobile & desktop.”
- Evoluzione di AngularJS, sviluppato da Miško Hevery (2010)
  - Definizione di elementi HTML custom
- Installazione via npm
  - Angular CLI (Command Line Interface): npm install -g @angular/cli
  - Verifica della versione dell’Angular CLI installato: ng --version

Workspace e starter app

- Angular CLI è basato su Webpack, semplifica il lavoro con Angular
- Dalla directory che intendiamo usare come workspace:
  - ng new *my-app*
  - Si possono accettare le scelte di default proposte
- Alla fine del (non breve) processo
  - Cambiare directory a quella dell'app (*my-app*, in questo caso)
  - Compilazione, esecuzione dell'app e apertura del browser
    - ng serve -o
  - Per default il server corre su
    - http://localhost:4200/
    - ng serve --port *nnnn* → il server corre sulla porta specificata

ng serve

- angular.json, proprietà projects.*my-app*.build.architect.options.main determina l'esecuzione di main.ts
- main.ts importa la classe AppModule definita in app/app.module.ts
- AppModule decorata da NgModule con le proprietà
  - declarations: lista di componenti definite nel modulo, "ng generate component" la aggiorna automaticamente
  - imports: dipendenze da altri moduli, per uso in template o per DI (dependency injection)
  - providers: servizi che devono essere disponibili via DI
  - bootstrap: componente per l'avvio dell'app – AppComponent
- AppComponent definisce l'elemento HTML 'app-root'
- Che viene usato nel body di index.html

Creazione di una component

- Nella root dell'applicazione

  ng generate component hello

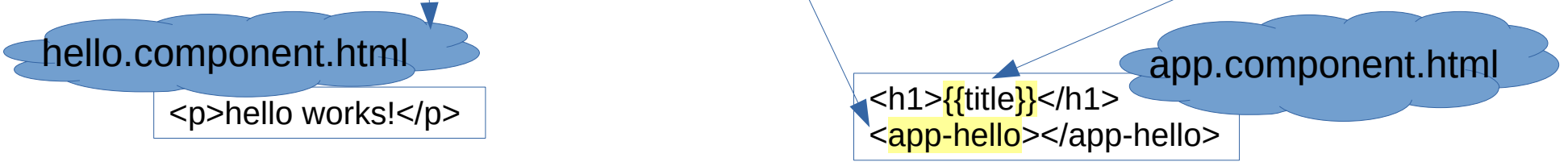
```
C:\dev\my-app>ng generate component hello
CREATE src/app/hello/hello.component.html (20 bytes)
CREATE src/app/hello/hello.component.spec.ts (621 bytes)
CREATE src/app/hello/hello.component.ts (265 bytes)
CREATE src/app/hello/hello.component.css (0 bytes)
UPDATE src/app/app.module.ts (392 bytes)
```

- .component.ts contiene la definizione di una classe decorata
  - Il decorator ***Component***
    - Meta-informazioni
  - La classe *xyz**Component***
    - Implementa OnInit

```
@Component({
  selector: 'app-hello',
  templateUrl: './hello.component.html',
  styleUrls: ['./hello.component.css']
})
export class HelloComponent implements OnInit {
  constructor() { }
  ngOnInit() {}
}
```

Il decorator Component

- Definisce le seguenti proprietà
  - selector: nome dell'elemento nel DOM
  - Il codice HTML associato è definito, a scelta, via una di queste due proprietà:
    - templateUrl: URL del file che lo contiene
    - template: inline
  - styleUrls: URL dei file in cui è specificato lo stile dell'elemento
- Si può usare il nuovo elemento con il nome definito in 'selector'
  - La sintassi {{ expression }} (template binding/mustache tag) permette di accedere proprietà di una componente



Proprietà in Component

- Nella root dell'app, creo una nuova Component
  - ng generate component user
- Aggiungo il nuovo elemento alla app Component
- Aggiungo una proprietà alla sua Component e la inizializzo nel costruttore
- Modifico il frammento HTML associato

app.component.html

```
<h1>{{title}}</h1>
<app-hello></app-hello>
<app-user></app-user>
```

```
export class UserComponent // ...
  name: string;

  constructor() {
    this.name = 'Tom';
  }

  // ...
```

user.component.ts

user.component.html

```
<span>{{ name }}</span>
```

La direttiva *ngFor

- Nella root dell'app, creo una nuova Component
  - ng generate component users
- Modifico la app Component per usare il nuovo elemento
- Aggiungo una proprietà array alla sua Component e la inizializzo nel costruttore
- Modifico il frammento HTML associato per eseguire un for each loop via direttiva *ngFor

app.component.html

```
<h1>{{title}}</h1>
<app-hello></app-hello>
<app-users></app-users>
```

```
export class UsersComponent // …
  names: string[];

  constructor() {
    this.names = ['Tom', 'Bob', 'Sid'];
  }
  // ...
```

```
let names = ['a', 'b', 'c'];
for (let name of names) {
    console.log(name);
}
```

users.component.html

```
<ul>
   <li *ngFor="let name of names">{{ name }}</li>
</ul>
```

Il decorator Input

- Modifica della component user
  - Importazione del decorator Input
  - Decorazione della proprietà name
  - Rimozione del set di name nel constructor
- Modifica della component users
  - Il template HTML accede la proprietà di user usando la sintassi [*property*]
    - [*property*] → ***property binding***

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

@Component({ /* … */ })
export class UserComponent // …
  @Input() name: string;

  constructor() { }

  // …
```

```
<ul>
   <li *ngFor="let name of names">
      <app-user [name]="name"></app-user>
   </li>
</ul>
```

Gestire i form

- ng generate component addItem
- Elemento app-add-item in app.component.html
- Form in add-item.component.html
  - Input associati a template variable (#name)
  - Attributo (click) del submit button associato ad add(), che prende le template variable come parametri
    - (*event*)="*method()*"→***event binding***
- Nella classe AddItemComponent, il metodo add() gestisce la chiamata dal form

```
<h1>{{title}}</h1>
<!-- … -->
<app-add-item></app-add-item>
```

```
<h2>Add item</h2>
<form>
        <input placeholder="enter id" #id>
        <input placeholder="name ..." #name>
        <button (click)="add(id, name);">
           OK
        </button>
</form>
```

```
add(id: HTMLInputElement, name: HTMLInputElement): boolean {
  console.log(`(${id.value}, ${name.value})`);
  return false;
}
```

Applicazione

- Albero di Component
  - La radice è la componente App, ovvero l’applicazione stessa, indicata in angular.json
  - Per default la componente root ha nome AppComponent ed è rappresentata dall’elemento HTML con nome ‘app-root’
- È una componente
  - Una applicazione può essere parte di un’altra applicazione
- Esecuzione dell’applicazione
  - ‘ng serve’ esegue main.ts, che importa (tra l’altro) l’AppModule corrente

Modulo

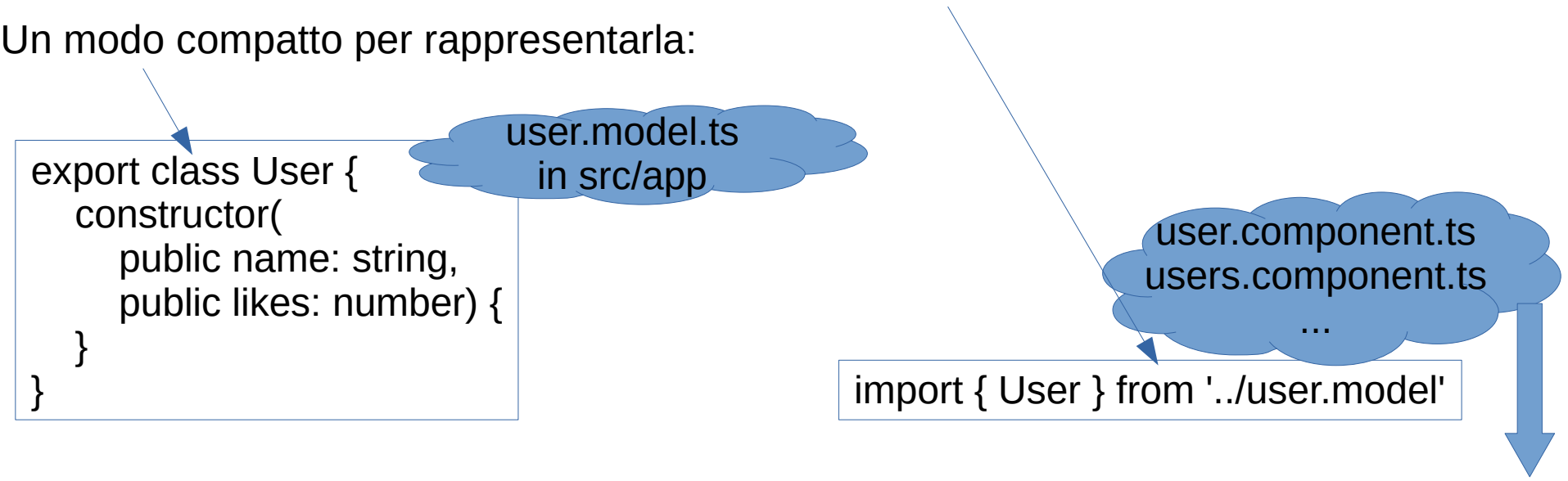
- Contenitore di funzionalità per applicazione
  - Aiuta a organizzare le parti in blocchi
- È una semplice classe
  - Nome di default AppModule
- Decorata con NgModule per specificare
  - declarations, imports, exports, providers, bootstrap
- Decorator @: al momento disponibile in JS solo via transpiling
  - Funzione che decora (annota) un elemento del linguaggio

Componente

- Blocco fondamentale di applicazioni Angular
  - ng generate component *xyz*
  - Classe TypeScript che, per convenzione, ha un nome nella forma *xyz*.component.ts
- Composto da
  - Component decorator, configurazione del componente
    - selector: nome dell'elemento (o attributo per un div) HTML
    - template/templateUrl: codice HTML associato, descrive la view
    - styles/styleUrls: CSS per il solo componente corrente ed eventuali discendenti
  - Classe decorata, *Xyz*Component
    - Descrive il controller
- Accesso al controller dalla view: template binding
  - {{ expression }} → riferimento nell'HTML a proprietà/metodi del controller
- Per il test, viene creato un file karma: *xyz*.component.spec.ts

Model per component

- È spesso utile avere una classe che rappresenta il model relativo a una component
- ng generate class User --type=model
- import nei ‘component.ts’ che la usano (ad es. User e Users)
- Un modo compatto per rappresentarla:



Model View Controller

users.component.ts

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

import { User } from '../user.model'

@Component({ /* … */ })
export class UsersComponent implements OnInit {
  users: Array<User>;

  constructor() {
    this.users = [new User('Tom', 2), new User('Bob', 1), new User('Sid', 3)];
  }

  ngOnInit() {}

  moreLikes(user: User) {
    console.log(`Likes for ${user.name} are ${user.likes}`);
  }
}
```

@Input e @Output

```
import {
  Component, OnInit, Input, Output, EventEmitter
} from '@angular/core';

import { User } from '../user.model'

@Component({ /* … */ })
export class UserComponent implements OnInit {
  @Input() user: User;
  @Output() liked: EventEmitter<User>;

  constructor() { this.liked = new EventEmitter(); }

  ngOnInit() { }

  plusOne() {
    this.user.likes += 1;
    this.liked.emit(this.user);
  }
}
```

user.component.ts

```
<ul>
   <li *ngFor="let user of users">
      <app-user [user]="user" (liked)="moreLikes($event);">
      </app-user>
   </li>
</ul>
```

users.component.html

```
<span>{{ user.name }}: {{ user.likes }}</span>
<div>
   <button (click)="plusOne();">Like</button>
</div>
```

user.component.html

Direttive

- ngIf: visualizzazione condizionale
- ngSwitch: scelta multipla
  - ngSwitchCase
  - ngSwitchDefault
- ngStyle: assegnazione di stile
- ngClass: assegnazione di classi
- ngFor: ripetizione di elementi
- ngNonBindable: esclusione dal binding
- ngForm: gestione dei form
- ngModel: two-way data binding

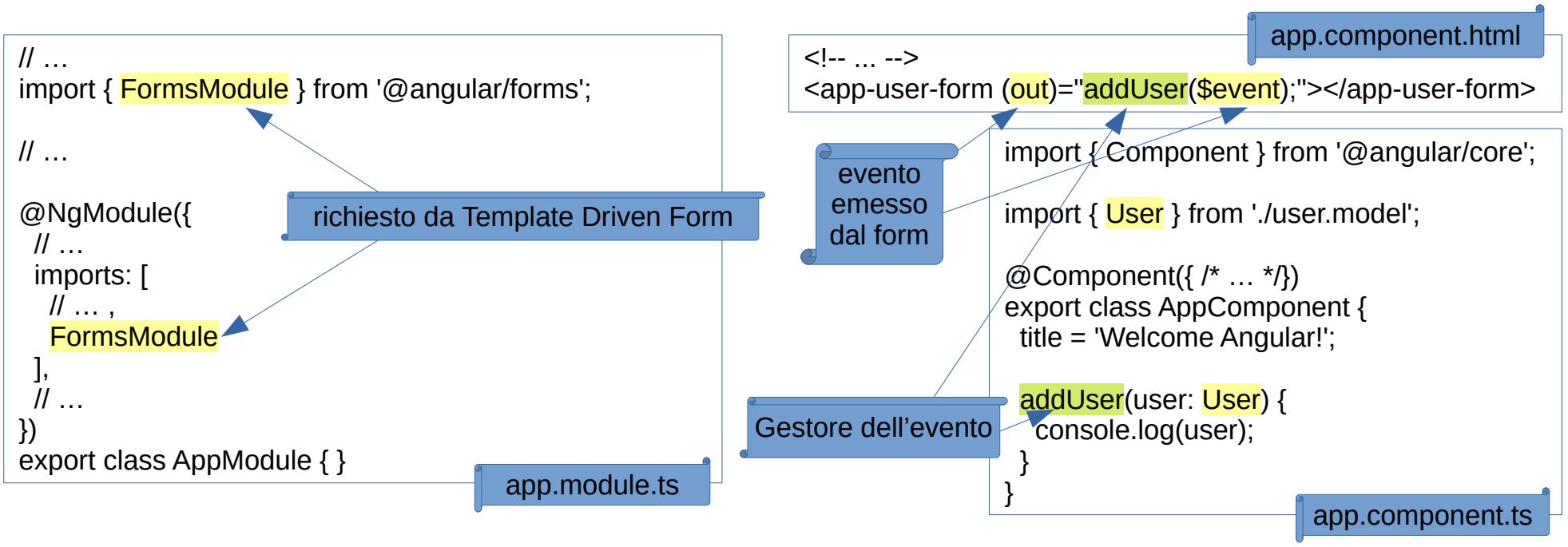
```
<span>{{ user.name }}: {{ user.likes }}</span>
<div>
   <div *ngIf="user.likes % 2" [ngStyle]="{color: 'blue'}">
      Odd number of likes
      <span ngNonBindable>{{unbound}}</span>
   </div>

   <div [ngSwitch]="user.name">
      <span *ngSwitchCase="'Tom'" [ngClass]="{zzz: true}">
         Hi
      </span>
      <span *ngSwitchCase="'Bob'" [ngClass]="{zzz: false}">
         Hello
      </span>
      <span *ngSwitchDefault>Good morning</span>
      {{user.name}}
   </div>
   <button (click)="plusOne();">Like</button>
</div>
```

user.component.html

Template Driven Form

- Form Angular definito da
  - Una classe TypeScript per gestire dati e interazioni
    - ng generate component UserForm
    - Il modulo deve importare FormsModule
  - Un template basato su HTML
    - Usa le direttive ngForm (e ngModel)

Setup per component form



Un component form

```
<h2>Template Driven Form</h2>
<form #userForm="ngForm">
  <div>
    <label for="name">Name</label>
    <input id="name" required
      [(ngModel)]="model.name" name="name">
  </div>

  <div>
    <label for="likes">Likes</label>
    <input type="number" id="likes"
      [(ngModel)]="model.likes" name="likes">
  </div>

  <button (click)="submit();">Submit</button>
</form>
```

Template variable Reference alla direttiva ngForm

bind form model

'name' richiesto da ngForm

user-form.component.html

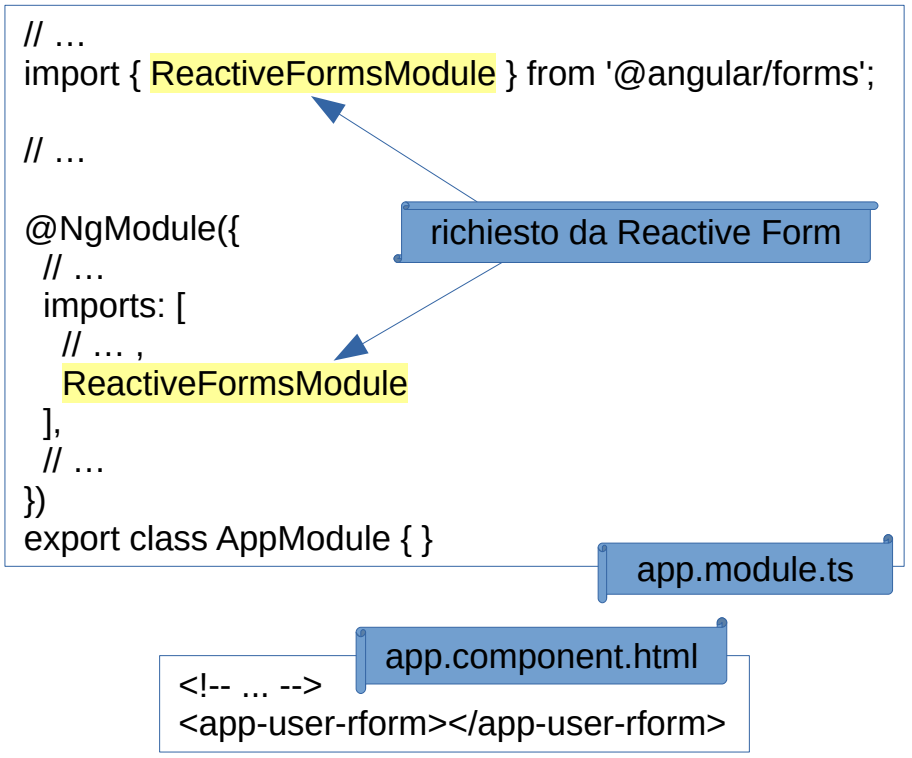
```
import { Component, OnInit, Output, EventEmitter }
from '@angular/core';

import { User } from '../user.model'

@Component({ /* … */ })
export class UserFormComponent implements OnInit {
  @Output() out = new EventEmitter<User>();
  model: User;

  constructor() {
    this.model = new User('Bill', 42);
  }

  submit() { this.out.emit(this.model); }

  ngOnInit() {}
}
```

user-form.component.ts

Reactive Form

- Più flessibile di Template Driven Form
- Form Angular definito da
  - Una classe TypeScript per gestire dati e interazioni
    - ng generate component UserRForm
    - Uso di FormBuilder, FormGroup, FormControl
    - Il modulo deve importare FormsModule
  - Un template basato su HTML
    - Usa le direttive reactive form (formGroup)

Reactive form setup



Un component reactive form

```
<h2>Reactive Form</h2>
<form [formGroup]="fUser">
    <div>
        <label for="name">Name</label>
        <input formControlName="name">
    </div>

    <div>
        <label for="likes">Likes</label>
        <input type="number" formControlName="likes">
    </div>

    <button (click)="submit(fUser.value);">Submit</button>
</form>
```

direttiva formGroup

user-rform.component.html

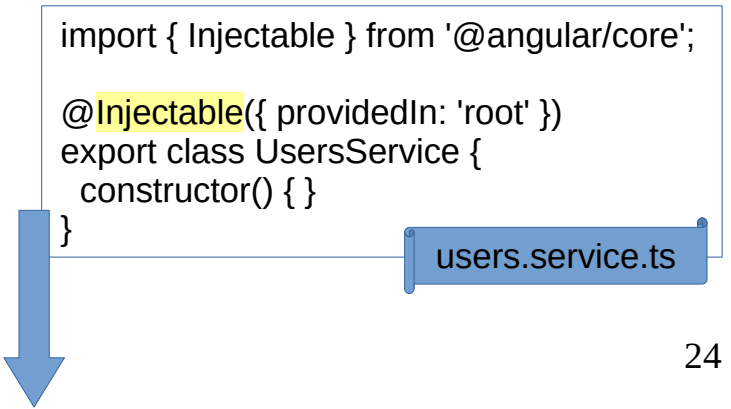
```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';

import { User } from '../user.model'

@Component({ /* … */ })
export class UserRFormComponent implements OnInit {
  fUser: FormGroup;

  constructor(fb: FormBuilder) {
    this.fUser = fb.group(new User('Kim', 12));
  }

  submit(user: User) { console.log(user); }

  ngOnInit() {}
}
```

Dependency Injection

user-rform.component.ts

Service

- Classe che implementa funzionalità condivise da elementi dell'applicazione.
  - Esempio: FormBuilder
  - Uso tipico: data source
- Gestiti da Angular via Dependency Injection
  - Supporto DI fornito al servizio via decorator Injectable
- Creazione di un nuovo servizio nell'app
  - ng generate service users



Un servizio

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

import { User } from './user.model';

@Injectable({ providedIn: 'root' })
export class UsersService {
  private users: Array<User>;

  constructor() {
    this.users = [
      new User('Bob', 1),
      new User('Tom', 2),
      new User('Sid', 3)
    ];
  }

  get(): Array<User> { return this.users; }

  add(user: User) { this.users.push(user); }
}
```

users.service.ts

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

import { UsersService } from '../users.service';
import { User } from '../user.model'

@Component({/*…*/})
export class UsersComponent implements OnInit {
  users: Array<User>;

  constructor(us: UsersService) { this.users = us.get(); }

  ngOnInit() {}

  moreLikes(user: User) {
    console.log(`Likes for ${user.name} are ${user.likes}`);
  }
}
```

users.components.ts

Routing

- Divisione dell’app in aree seguendo di solito regole basate sull’URL
- In una SPA si potrebbe avere una sola URL ma si perderebbero i vantaggi dei bookmark
- Il package Angular è @angular/router
  - supporta il client-side routing di HTML5

Esempio Routing

Routing1Component
Routing2Component
Routing3Component

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

// ...
import { Routing1Component } from './routing1/routing1.component';
import { Routing2Component } from './routing2/routing2.component';
import { Routing3Component } from './routing3/routing3.component';

// ...
const appRoutes: Routes = [
  { path: 'one', component: Routing1Component },
  { path: 'two', component: Routing2Component },
  { path: 'three', component: Routing3Component }
];

// ...
@NgModule({
  imports: [
  // ...
  RouterModule.forRoot(appRoutes)],
  // ...
```

app.module.ts

```
<!-- … -->
<div>
   <h2>Routing Example</h2>
   <nav>
      <a routerLink="one">First</a> +
      <a routerLink="two">Second</a> +
      <a routerLink="three">Third</a>
   </nav>
   <router-outlet></router-outlet>
</div>
```

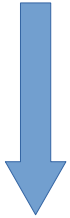
app.component.html

HTTP

- Libreria Angular per chiamate asincrone
- Tre diversi approcci supportati da JavaScript
  - Callback
  - Promise
  - Observable (preferito da Angular)

Esempio HTTP Observable

- Nuova componente MyHttp
  - Effettua una chiamata HTTP GET
  - Sottoscrive a HttpClient per la response
    - HttpClient: classe iniettabile, alternativa Angular a AJAX / fetch
- Il server deve supportare Cross-origin resource sharing (CORS)
  - Nell’esempio uso https://github.com/egalli64/nesp



Esempio HTTP Observable

```
<h2>HTTP Request</h2>
<button type="button" (click)="makeRequest()">Make Request</button>
<span *ngIf="!loaded"> loading ... </span>
<span> {{message}} </span>
```

my-http.component.html

app.component.html

```
<!-- ... -->
<app-my-http></app-my-http>
```

```
// ...
import { HttpClientModule } from '@angular/common/http';

// ...
import { MyHttpComponent } from './my-http/my-http.component';

// in @NgModule
// declarations→MyHttpComponent
// imports→HttpClientModule
```

app.module.ts

```
import { HttpClient } from '@angular/common/http';
// ...

export class MyHttpComponent implements OnInit {
  message: String;
  loaded: boolean;
  http: HttpClient;

  constructor(http: HttpClient) {
    this.http = http;
    this.loaded = true;
  }

  makeRequest(): void {
    this.loaded = false;
    this.http.get('http://localhost:3000/hello').subscribe(
      data => {
        this.message = data['message'];
        this.loaded = true;
      }
    );

    // ...
```

my-http.component.ts

Angular powered Bootstrap

- Bootstrap widgets the Angular way
  - https://ng-bootstrap.github.io/
- Usa solo Angular e Bootstrap.css (no js)
- Installazione via npm, dependencies:
  - https://ng-bootstrap.github.io/#/getting-started
- Ex: npm install --save @ng-bootstrap/ng-bootstrap bootstrap@4.3.1
- In angular.json, per il progetto, inserire tra gli styles
  - "node_modules/bootstrap/dist/css/bootstrap.min.css"
- In app.module.ts
  - import {NgbModule} from '@ng-bootstrap/ng-bootstrap';
  - imports: [NgbModule, /* … */]
- Se necessario, specifiche import nel controller

```
<ngb-alert type="success" [dismissible]="false">
    Success!
</ngb-alert>
```

```
<ngb-carousel ...>
  <ng-template ngbSlide>
  <!-- … -->
```

```
import { NgbCarouselConfig }
  from '@ng-bootstrap/ng-bootstrap';

// …

constructor(private config: NgbCarouselConfig) {
  config.showNavigationArrows = false;
  config.interval = 6000;
}
```