

Angular 9

Sviluppare un'applicazione component-based

1

Prima di iniziare

Download repository con i laboratori

<https://github.com/soniapini/angular-mail-app>

2

Hardware & Software necessari

- laptop
- HTML5 Browser (Chrome, Firefox)
- text editor o IDE che supporti HTML5, CSS3, TypeScript
 - Sublime
 - Atom
 - Visual Studio Code
 - Idea
 - ...

3

Visual Studio Code

- Community molto attiva per Angular
- Estensioni utili
 - <https://medium.com/frontend-coach/7-must-have-visual-studio-code-extensions-for-angular-af9c476147fd>

4

Argomenti del Corso

- Differenze tra AngularJs e Angular
 - Angular Evolution
 - Transizione
 - Change Detection
 - Promise Vs Observable
 - TypeScript
- Concetti Chiave di Angular
 - NgModules
 - Components
 - Bindings
 - Services



5

Argomenti del Corso

- Pensare a Componenti
 - Sviluppo di un Componente Angular
 - Gestire gli @Input
 - Gestire gli @Output
 - Lifecycle dei Componenti
 - Transclude Contents
 - Ng-Template & Ng-Container
 - Come far collaborare i Componenti
 - Template Driven Form
 - Reactive Form
 - Classificazione dei componenti
 - “Smart”, “Dumb” e “Stateless”



6

Varie ed Eventuali

- Angular CLI
- Librerie Utilizzate
- Link Utili



7

Laboratori del Corso

- Lab 0 - Hello world Angular Application
- Lab 1C - Identificare i componenti
- Lab 01 - MailLogo Component
- Lab 02 - MessageViewer Component
- Lab 02 bis - MessageViewer @Output
- Lab 03 - FolderList Component
- Lab 03 Extra - AllowCreate
- Lab NgContent
- Lab NgTemplate
- Lab 04 - MessageList Component
- Lab 05 - MailComposer Component



8

Laboratori del Corso

- Lab 05 bis - Validazioni
- Lab Reactive Form



9

Approccio del Corso

1. Teoria
2. Laboratorio pratico
 - a. applicazione esempio da completare e modificare
 - b. discussione
 - c. quiz

10

<>

Clean Code... Sempre

- Clean Code: the book
 - https://books.google.it/books/about/Clean_Code.html?id=hjEFCAAAQBA



11

<>

JavaScript Concetti Avanzati

- Yakov Fain - Advanced Introduction to Javascript
 - <https://www.youtube.com/watch?v=X1J0oMayvC0>
- Enterprise WebBook
 - http://enterprisewebbook.com/appendix_a_advancedjs.html
 - <https://github.com/Farata/EnterpriseWebBook>
 - https://github.com/Farata/EnterpriseWebBook_sources

<|>

12

AngularJs vs Angular

13

AngularJs vs Angular

Angular Evolution...

AngularJs 1.5

JavaScript
one-way bindings

Angular 2

Typescript
Mobile-oriented

Angular 4

Typescript 2.1
HTTPClient

Angular 5

increase standardization
@angular/http deprecated

Angular 6

Angular CLI - ng-update
RxJs

Angular 8

IVY Pre-view

14

<> AngularJs vs Angular

Concetti Eliminati



Presentazione: <https://youtu.be/qNmWvbAvBHI>

15

<> AngularJs vs Angular

Concetti modificati

AngularJs

- Filters
- ng-controller
- ng-class
- ng-repeat
- ng-if

Angular

- Pipes
- @Component Classes
- ngClass
- *ngFor
- *ngIf
- async (pipe)

Angular: <https://angular.io/guide/ajs-quick-reference>

16

I Cicli

```
<tr ng-repeat="movie in vm.movies" >
```

```
<tr *ngFor="let movie of movies" >
```

Angular: <https://angular.io/guide/ajs-quick-reference>

17

Aggiungere o rimuovere dal DOM

```
<table ng-if="movies.length" >
```

```
<table *ngIf="movies.length" >
```

Angular: <https://angular.io/guide/ajs-quick-reference>

18

Mostrare o nascondere parti del DOM

```
<h3 ng-show="vm.favoriteHero" >  
  Your favorite hero is:  
  {{vm.favoriteHero}}  
</h3>
```

```
<h3 [hidden]="!favoriteHero" >  
  Your favorite hero is:  
  {{favoriteHero}}  
</h3>
```

La direttiva ng-hide di AngularJs non ha corrispettivo in Angular

Angular: <https://angular.io/guide/ajs-quick-reference>

19

Applicare CSS & Stili dinamicamente

```
<div ng-class="{active: isActive}" >  
<div ng-class="{active: isActive,  
  shazam: isImportant}" >
```

```
<div [ngClass]="{'active': isActive}" >  
<div [ngClass]="{'active': isActive,  
  'shazam': isImportant}" >  
<div [class.active]="isActive" >
```

- [class.active] → Class binding
- [attr.aria-label] → Attribute binding
- [style.width] → Style binding

Angular: <https://angular.io/guide/ajs-quick-reference>

20

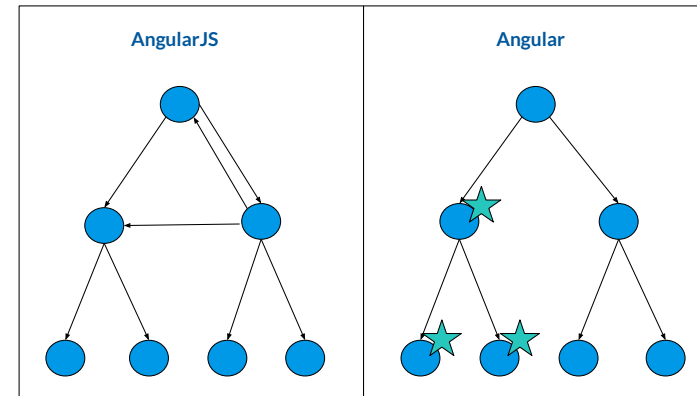
Change Detection

Angular fornisce 2 strategie:

- Standard
 - Use the default **CheckAlways** strategy, in which change detection is automatic until explicitly deactivated.
- OnPush
 - Use the **CheckOnce** strategy, meaning that automatic change detection is deactivated until reactivated by setting the strategy to Default (**CheckAlways**). Change detection can still be explicitly invoked. This strategy applies to all child directives and cannot be overridden.

21

Tree traversing in AngularJs vs Angular



22

Promise API → Observable (RxJs)

A promise is a placeholder for a future value.

Una *Promise* rappresenta un'operazione che non è ancora completata, ma lo sarà in futuro. (ES2015)

Observable

RxJS (*Reactive Extensions for JavaScript*) is a library for reactive programming using *observables* that makes it easier to compose asynchronous or callback-based code.

<|>

23

Promise API → Observable (RxJs)

- Asynchronous
- One-time operation
- Non-lazy
- Success or failure callback
- Asynchronous e Synchronous
- Stream multiple results
- Lazy
 - `subscribe()`
- Success, Failure, Complete
- Cancellable
 - `unsubscribe()`
- Operators
 - `map`, `forEach`, `filter`, ...

<|>

24

Observable: Push o Pull Model?

Push e Pull sono protocolli di comunicazione tra i data Producers e i Consumers

Pull Model: Il Consumer determina quando avere i dati. Il Producer non decide quando i dati saranno consegnati. *Pensate alle funzioni...*

Push Model: Il Produces determina quando spedire i dati al Consumer. Il Consumer non sa quando i dati arriveranno. *Pensate alle Promise...*

<>

25

Observable: Esempi pratici

```
var observable = Rx.Observable.create((observer: any) =>{
  observer.next('Hi Observable');
});

observable.subscribe((data)=>{
  console.log(data);
})
```

Producer

Consumer

output:
'Hi Observable'

D: Observable = Funzione?
R: No

```
var observable = Rx.Observable.create((observer: any) =>{
  observer.next('Hi Observable');
  observer.next('Am I understandable?');
});

observable.subscribe((data)=>{
  console.log(data);
})
```

Producer

Consumer

output:
'Hi Observable'
'Am I understandable?'

Observable restituiscono
streams

<>

26

Observable: Esempi pratici

```
var observable = Rx.Observable.create((observer: any) =>{
  observer.next('Hi Observable');
  setTimeout(()=>{
    observer.next('Yes, somehow understandable!')
  }, 1000)

  observer.next('Am I understandable? ');
});

observable.subscribe((data)=>{
  console.log(data);
})
```

Producer

Consumer

output:
'Hi Observable'
'Am I understandable?'
'Yes, somehow understandable!'.

Valori Asincroni

<>

27

Change Detection & RxJs: dal vivo

<https://stackblitz.com/edit/angular-changedetection-test-987654>

28

TypeScript

- Data Types
- Decorators
- Classes
- Interfaces
- Enum
- IDE Friendly

TypeScript 1.5 include tutte le feature necessarie ad Angular.

TypeScript diventa linguaggio principale di Angular.

<>

29

Angular Key Concepts

30

NgModules

- **NgModules**
 - Forniscono contesto di compilazione per i Componenti
 - Angular App = insieme di NgModules

Angular App è composta da:

- 1 *root module* utilizzato per il bootstrapping di solito chiamato **AppModule**
- 0 - n *feature modules*
 - Reusability
 - Caricamento dei moduli on-demand, codice startup più piccolo e partenza più veloce

Angular API: <https://angular.io/api/core/NgModule>

31

NgModules

```
import { NgModule } from
'@angular/core';
import { BrowserModule } from
'@angular/platform-browser';
@NgModule({
  imports:    [ BrowserModule ],
  providers:  [ Logger ],
  declarations: [ AppComponent ],
  exports:    [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Librerie importate

Services

Components

<>

32

Components

• Components

- sono Classi con Decoratori - metadata utilizzati da Angular
- contengono Application data e logica
- HTML Template → Views
- usano Services tramite Dependency Injection (DI)
- sono tipi speciali di Directive

Angular App è composta da:

- 1 (almeno) *Root Component* di solito chiamato **AppComponent**

Angular API: <https://angular.io/api/core/Component>

33

Components

```
@Component({
  selector: 'app-hero-list',
  templateUrl: './hero-list.component.html'
})
export class HeroListComponent implements OnInit {
  /* . . . */
}
```

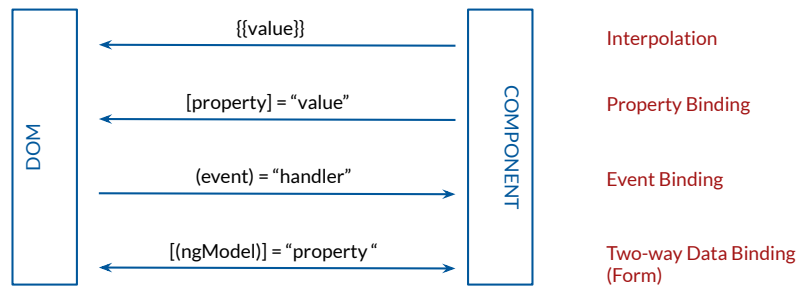
Metadata

Lifecycle
Hooks

34

Binding

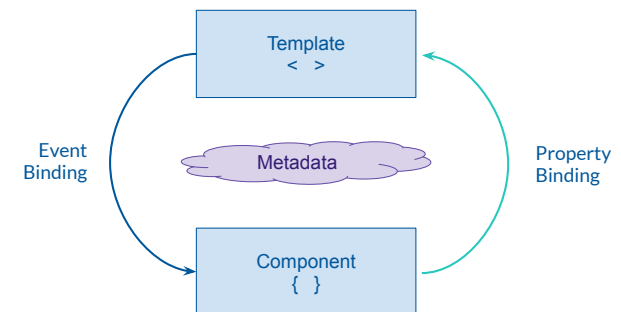
Il diagramma mostra le 4 forme di data binding



Angular: <https://angular.io/guide/template-syntax>

35

Data Binding



<|>

36

Component in AngularJs

Metodo .component()

```
angular.module('heroApp').component('heroDetail', {
  templateUrl: 'heroDetail.html',
  controller: HeroDetailController,
  bindings: {
    hero: '<',
    onDelete: '&',
    onUpdate: '&'
  }
});
```

Input Bindings

- '<' - one-way binding
- '=' - two-way binding
- '@'

Output Bindings

- '&'

<|>

37

Lab 0 - Angular Hello word Component

Vedere project/00/helloworld-base

Struttura dell'applicazione Angular

39

Component in Angular

Decoratore @Component

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

import { Hero } from './hero';

@Component({
  selector: 'hero-detail',
  template: `...`,
  styleUrls: [ './hero-detail.component.css' ]
})
export class HeroDetailComponent {
  @Input() hero: Hero;
  @Output() updateHero: EventEmitter;
}
```

Template

```
<div *ngIf="hero">
  <h2>{{hero.name}} details!</h2>
  <div><label>id: </label>{{hero.id}}</div>
  <div>
    <label>name: </label>
    <input [(ngModel)]="hero.name" placeholder="name"/>
  </div>
</div>
```

<|>

38

An Angular Application

```
import { BrowserModule } from
  '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Dal vecchio ng-app

Al nuovo NgModule

40

Lab 0

Modificare l'esempio Hello World in modo che:

1. la Classe AppComponent inizializzi una variabile con il Timestamp corrente
2. Il Template del AppComponent lo visualizzi

Provare a far partire l'applicazione:

- npm run start helloworld-base
- funziona? cosa manca?

41

Pensare a Componenti

43

Services

- **Services**
 - forniscono funzionalità non legate direttamente alle Views
 - possono essere Iniettati come dipendenze

<|>

42

Identificare i componenti

- Suddividere una "View" / "Page" in gerarchia di Components
- Sviluppare Component-based UIs
 - rende più semplice e meno costoso fare un buon design
 - spinge nella giusta direzione (best practice)

Cos'è un component

Self-contained set of UI and logic

- encapsulates a **specific behaviour**
- provides an **explicit API**

44

Lab IC - Identificare Componenti

Identificare i componenti chiave in una tipica WebMail application

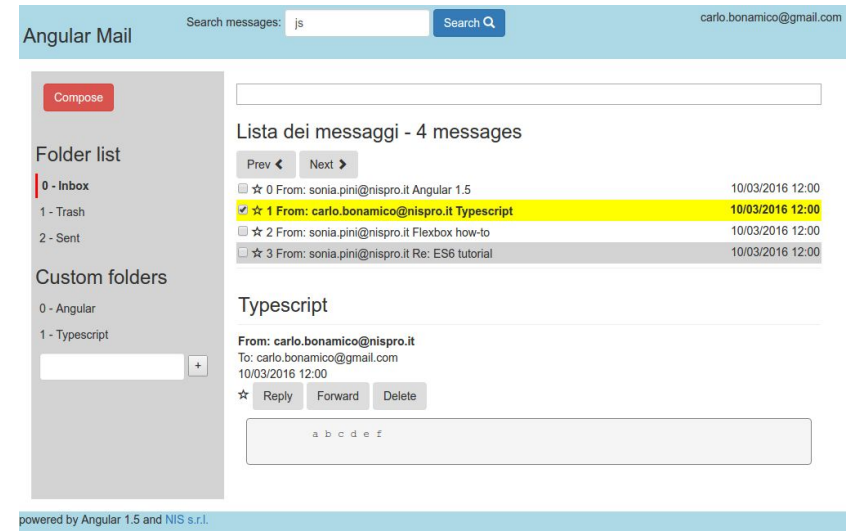
Analizzare quali componenti possono essere riutilizzati in più view

Identificare quali sono gli input e gli output per ogni componente

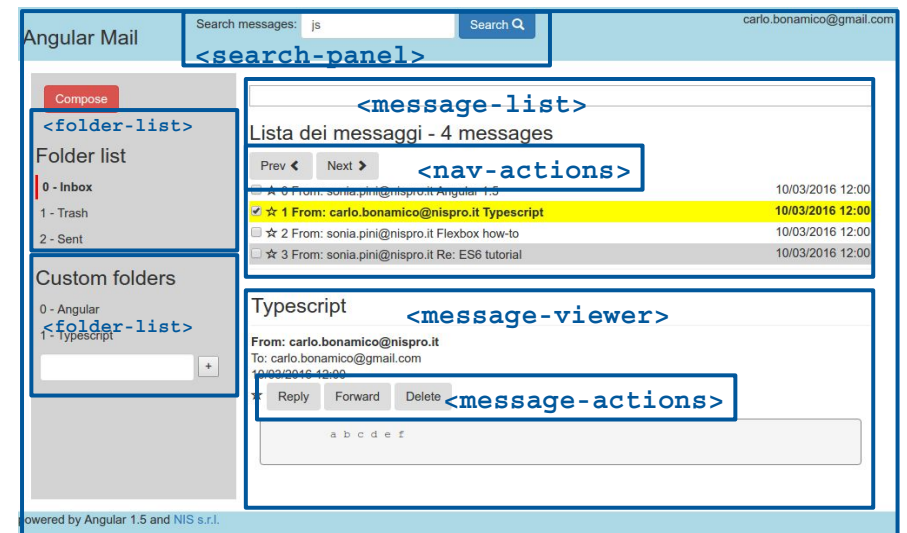
Now go find even more components

https://drive.google.com/drive/u/0/folders/0B-Bogp8tUho_bDh6SkFOMXEwa1E

45



46



47

48

Vantaggi nell'uso di Componenti

Stronger Encapsulation (scope isolato + binding espliciti)

- Modificare l'implementazione interna del componente ha meno impatto sul resto dell'applicazione
- più disaccoppiamento, meno regressioni

Reusability (con parametrizzazione)

- stesso componente utilizzato in contesti differenti
- <message-list> può visualizzare sia i messaggi presenti nei folder sia i risultati della ricerca

49

Vantaggi nell'uso dei Componenti

Better Collaboration

- Meno conflitti alla crescita del Team
- Più semplice verificare le regressioni

Clarity e readability

- Posso usare un componente conoscendone solo l'API
- Il collegamento con altri componenti è chiaro ed esplicito nel HTML

50

Component-Based UI

AngularJs

Sviluppare Applicazioni Component-Based era possibile, ma

- non semplice
- sforzo aggiuntivo
- seguire una serie di criteri

Angular

- L'unico modo è fare Componenti
- L'applicazione stessa è un componente
- Grande semplificazione della sintassi
 - migliore leggibilità
 - meno sforzo
- Typescript

51

Angular Component API

- Dichiarare il Componente `@Component`
- Definire l'interfaccia del Componente all'interno della Classe decorata con `@Component`
 - inputs → decoratore `@Input`
 - output → decoratore `@Output`
- Gestire lifecycle del componente con
 - `ngOnInit`
 - `ngOnChanges`
 - `ngOnDestroy`
- Collegare i componenti l'uno all'altro

52

Lab 01 - Mail-logo Component

Creare il componente `<nis-mail-logo>`

1. Aprire l'applicazione `01base` (/projects/01/01base)
2. Andare sul folder `mail-logo` contenente lo scheletro del componente

TODO

- Scrivete template HTML inserendo un logo/scritta a piacere
- Applicare applicare formattazione CSS a piacere
- Completare la definizione del componente `mail-logo.component.ts`

53

Lab 01 - TIPS

- Usare `templateUrl` nel decorator `@Component` per inserire il corretto template
- Importare il componente nel `@NgModule`
 - `nis.module.ts`

54

<> [Pensare a Componenti](#)

Main Page Component

Chi passa gli input agli altri Componenti?

- Il ruolo del MailView Component
 - interagire con i servizi di backend
 - fornire i dati ai componenti
 - coordinare gli elementi della pagina

TIP

Separare Layout dai componenti per incrementare il riuso

55

<> [Pensare a Componenti](#)

Il Message-Viewer Component

Componente che visualizza un messaggio di posta.

Ha bisogno di `@Input`?

Quali sono i campi che compongono un messaggio di posta?

Incorporiamo nel componente anche i pulsanti:

- Reply
- Forward
- Delete

TIP

Definiamo il `DataType Message` per semplificare gli sviluppi ed evitare errori

56

Message-View Component: @Input

```
@Component({
  selector: 'nis-message-viewer',
  templateUrl: './message-viewer.component.html',
  styleUrls: ['./message-viewer.component.scss']
})
export class MessageViewerComponent implements OnInit {
  @Input() message: Message;
```

```
export interface Message {
  subject: string;
  from: string;
  to: string;
  body: string;
}
```

Descrizione

Componente utilizzato per visualizzare un singolo messaggio.

Deve prendere in input il messaggio

57

Message-Viewer Component in Mail-View

```
<div>
  <section class="main-pane">
    <nis-message-viewer
      [message]="currentMessage">
    </nis-message-viewer>
  </section>
</div>
```

Descrizione

Componente utilizzato per visualizzare un Messaggio
Deve prendere in input un Messaggio

58

Lab 02 - Message-Viewer Component

Creare il componente `<nis-message-viewer>`

1. Aprire l'applicazione **02base** (/projects/02/02base)
2. Andare sul folder **message-viewer**

TODO

- Gestire il Messaggio @Input
- Agganciare i Click per i pulsanti Reply, Forward, Delete
- Loggare la chiamata della funzione al click sui pulsanti

59

Lab 02 - Message-Viewer Component

TODO

- Importare il componente nel @NgModule
- Completare il template del componente
- Istanziare il nuovo componente all'interno del Template del Mail-View
- Utilizzate il primo messaggio come input per il componente
 - message[0]

NOTA: TEST ad ogni passo ... - **F12** è il vostro **BBF**!

60

Eventi e Callback

I Componenti non possono fare tutto da soli.

Per implementare logiche complesse, un componente ha bisogno di interagire con:

- Child Components
- Parent Components
- Sibling Components

61

Separazione delle Responsabilità

Il Folder-List Component è responsabile di:

- Mostrare la lista dei folder
- Mostrare quale elemento è selezionato
- (Inserire un nuovo Folder)

Ma cosa fare quando un Utente seleziona un folder è un altro Use Case, un'altra Responsabilità.

Quindi teniamolo Fuori dal Folder-List component.

62

Il Folder-List Component

Vogliamo utilizzare un singolo componente per più cose, ad esempio

- per visualizzare i Folder standard sempre presenti
- per visualizzare i Folder custom creati dall'Utente
- per aggiungere nuovi Folder custom

Da dove prendiamo la lista dei Folder?

Dove è memorizzato questo elenco?

Da chi viene utilizzato e navigato?

63

Folder-List Component in Mail-view

```
<div>
  <section class="main-pane">
    <nls-folder-list
      [folders]="folders">
    </nls-folder-list>
  </section>
</div>
```

Descrizione

Componente utilizzato per visualizzare un elenco dei Folder

Deve prendere in input l'elenco dei Folder

64

Folder-List Component Definition

```
@Component({
  selector: 'nis-folder-list',
  templateUrl: './folder-list.component.html'
})
export class FolderListComponent {
  @Input() folders: Array<Folder>;
  @Output() selectedFolder: EventEmitter<any>;
  ...
}
```

```
if (this.folders.length > 0) {
  // do Something
}
```

```
<div *ngFor="let folder of folders">
</div>
```

Nota

L'input è direttamente accessibile come campo della classe e nel template del componente
L'Output è un EventEmitter

65

Azioni e Conseguenze

I Component devono Gestire Azioni con Conseguenze sia Interne sia Esterne.

Quando un Utente seleziona un Folder, avvengono due cose:

1. (Interna) Il Folder corrente deve essere evidenziato dal Folder-List
2. (Esterna) Gli altri Component devono essere notificati della selezione per
 - a. Eseguire Azioni
 - b. Abilitare Pulsanti
 - c. Aggiornare altre View

66

Azioni e Conseguenze - Inside Component

```
<div [ngClass]="{'current-folder': folder === currentFolder}"
  (click)="select(folder)">
  {{folder}}
</div>
```

Descrizione

L'evento di click sul Folder scatena l'esecuzione del metodo select(...).

select(...):

- modifica lo stato interno del componente e la sua View
- emette un evento per avvisare il Component parent dell'azione interna

```
select(selectedFolder) {
  this.currentFolder = selectedFolder;
  this.selectedFolder.emit(selectedFolder);
}
```

67

Azioni e Conseguenze - Outside Component

```
<section class="folder-list">
  <nis-folder-list
    [folders]="folders"
    (selectedFolder)="selectFolder($event)">
  </nis-folder-list>
</section>
```

Descrizione

L'evento di click sul Folder scatena l'esecuzione del metodo select(...).

select(...):

- modifica lo stato interno del componente e la sua View
- emette un evento per avvisare il Component parent dell'azione interna

68

@Output - How to do

1. Dichiarare l'output all'interno della classe del Component

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

@Component({
  selector: 'nis-folder-list',
  templateUrl: './folder-list.component.html'
})
export class FolderListComponent {

  @Output() selectedFolder: EventEmitter<Folder> = new EventEmitter<Folder>();
```

Questo inserisce un `selectedFolder` event callback nell'istanza del Component

69

@Output - How to do

2. invocare la callback quando il Folder è selezionato

```
selectFolder(folder) {
  this.currentFolder = folder;

  this.selectedFolder.emit(folder);
}
```

70

Lab 02 bis- Message-Viewer Component

TODO in Message-viewer Component

- Dichiarare gli eventi @Output nel component
 - Reply
 - Forward
 - Delete
- Gestire il click sul bottone e l'emissione dell'evento

TODO in Mail-View Component

- bindare gli eventi a metodi presenti nella classe MailViewComponent

71

Lab 03 - Folder-List Component

Implementare Folder-List Component

- Prendere la lista dei Folder dal MailViewComponent
- Visualizzarla
- Evidenziare il Folder corrente
- Abilitare la selezione di un nuovo Folder
- Notificare MailView Component ad ogni cambio di Folder così che possa caricare i messaggi del folder

72

Lab 03 - Folder-List Component

TODO

- Importare il Component nel @NgModule
- Visualizzarla
- Scrivere il Component
 - definire i metadata
 - completare il template HTML
 - attivare una class CSS al click

73

Lab 03 - Folder-List Component

TODO

- Aggiungere l'EventEmitter
- Gestire il click sul Folder
- Inizialmente semplicemente loggare qualcosa
- Emettere l'evento sul click
- Bindare l'evento nel Template del Parent Component (MailView)
- Implementare il metodo selectFolder() nel Parent Component

74

Lab 03 - extra

Passare al componente un parametro @Input aggiuntivo

- allowCreate di tipo booleano

TODO

Gestire nel Template del Component la possibilità di creare nuove Folder

75

<> [Pensare a Componenti](#)

Transclude Contents

Transclusion permette di iniettare oggetti DOM all'interno di un Component

In modo analogo alla direttiva `ng-transclude`.

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

76

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

@Component({
  selector: 'nis-card',
  templateUrl: './card.component.html',
  styleUrls: ['./card.component.scss']
})
export class CardComponent implements OnInit {

  @Input() header = 'this is header';
  @Input() footer = 'this is footer';
}
```

```
<div class="card">
  <div class="card-header">
    {{ header }}
  </div>
  <!-- single slot transclusion here -->
  <ng-content></ng-content>
  <div class="card-footer">
    {{ footer }}
  </div>
</div>
```

```
<h1>Single slot transclusion</h1>
<card header="my header" footer="my footer">
  <!-- put your dynamic content here -->
  <div class="card-block">
    <h4 class="card-title">You can put any content here</h4>
    <p class="card-text">For example this line of text and</p> <a href="#" class="btn btn-primary">This button</a>
  </div>
  <!-- end dynamic content -->
</card>
```

ng-content: select Attribute

```
<ng-content select="[card-body]"></ng-content>
```

```
<h1>Single slot transclusion</h1>
<nis-card header="my header" footer="my footer">
  <div class="card-block" card-body><!-- We add the card-body attribute here -->
    <h4 class="card-title">You can put any content here</h4>
    <p class="card-text">For example this line of text and</p>
    <button>This button</button>
  </div>
</nis-card>
```

ng-content: select Class

```
<ng-content select=".card-body"></ng-content>
```

```
<h1>Single slot transclusion</h1>
<nis-card header="my header" footer="my footer">
  <div class="card-block card-body"><!-- We add the card-body css class here -->
    <h4 class="card-title">You can put any content here</h4>
    <p class="card-text">For example this line of text and</p>
    <button>This button</button>
  </div>
</nis-card>
```

ng-content: select html-tag / component

```
<ng-content select="nis-card-body"></ng-content>
```

```
<h1>Single slot transclusion</h1>
<nis-card header="my header" footer="my footer">
  <nis-card-body><nis-card-body> <!-- We add the card-body component here -->
</nis-card>
```

ng-content: multi-slot

```
<div class="card">
  <div class="card-header"> <!-- header slot here -->
    <ng-content select="card-header"></ng-content>
  </div>
  <ng-content select="card-body"></ng-content> <!-- body slot here -->
  <div class="card-footer"> <!-- footer -->
    <ng-content select="card-footer"></ng-content>
  </div>
</div>
```

```
<h1>Single slot transclusion</h1>
<nis-card header="my header" footer="my footer">
  <div class="card-block"><!-- We add the card-body attribute here -->
    <h4 class="card-header">You can put any content here</h4>
    <p class="card-body">For example this line of text and</p>
    <button class="card-footer">This button</button>
  </div>
</nis-card>
```

81

Lab NgContent

Rifattorizzare il MessageViewer Component rimuovendo il tag h3 contenente il titolo.

Creare un nuovo MessageCard Component con:

- una sezione header che contenga il titolo
- una sezione ng-content che contenga il MessageViewer Component

82

Angular CLI - creazione Component

- Spostarsi nella cartella `/projects/03/03base/src/nis`
- eseguire il comando di generazione del componente
 - `ng generate component components/message-card --project=03base`

83

ng-content: limitazioni

- Non è utilizzabile all'interno di `*ngFor`
- Ma c'è un altro modo: `ng-template`

<https://blog.angular-university.io/angular-ng-template-ng-container-ng-templateoutlet/>

84

ng-template

Come indica il nome la direttiva ng-template rappresenta un template Angular.

- il tag <ng-content> contiene parte di un Template
- può essere combinato con altri Template
- e creare il Template finale di un Component

*ngIf e *ngFor sono esempi di ng-template

85

ng-template - Definizione

Proviamo a scrivere un template per il loading da visualizzare quando ancora i dati non sono arrivati

```
<ng-template>
<div>Loading...</div>
</ng-template>
```

Nota

Così abbiamo solo definito il template, ma non lo abbiamo utilizzato.

RISULTATO

Non viene disegnato nulla sullo schermo.

86

ng-template - Utilizzo

Proviamo a utilizzare il template ad esempio nel componente Message-List

```
<div *ngIf="message else loading">
  <div *ngFor="let message of messages">
    <!-- visualizzazione dell'elenco dei messaggi -->
  </div>
</div>

<ng-template #loading>
  <div>Loading...</div>
</ng-template>
```

Da Ricordare

Non è possibile utilizzare *ngIf e *ngFor sullo stesso elemento del DOM

87

ng-container

Per evitare di creare un tag div extra possiamo utilizzare la direttiva ng-container

```
<ng-container *ngIf="message else loading">
  <div *ngFor="let message of messages">
    <!-- visualizzazione dell'elenco dei messaggi -->
  </div>
</ng-container>

<ng-template #loading>
  <div>Loading...</div>
</ng-template>
```

88

Lab ng-template

Modificare Message-List Component

- per visualizzare la scritta loading quando non ci sono ancora messaggi
- utilizzando ng-container e ng-template

OPPURE

- Utilizzare ng-template per customizzare la visualizzazione del componente Folder-List (Custom Folder)

89

<|> [Pensare a Componenti](#)

Compodoc

Tool utilissimo per la generazione automatica della Documentazione.

<https://compodoc.app/>

- `npm i --save-dev @compodoc/compodoc`
- `npm install -g @compodoc/compodoc`

Modificare il file `tsconfig.json`

```
"include": [
  "projects/06/6xdemo/**/*"
],
"exclude": [
  "node_modules",
  "**/*.spec.ts"
]
```

90

<|> [Pensare a Componenti](#)

Compodoc

Per generare la documentazione e lanciare il server locale

```
> compodoc -p tsconfig.json -s
```

91

Lab 04 - Message-List Component

Implementare Message-List Component

- Ricevere la lista dei Messaggi dal MailViewComponent
- Visualizzarla
- Evidenziare il Messaggio corrente
- Abilitare la selezione di un Messaggio
- Abilitare la navigazione tra i Messaggi
- Notificare MailView Component ad ogni cambio di Messaggio
- Aggiungere gli opportuni Eventi @Output

92

Message-List Component

Il Message-List Component è responsabile di:

- Mostrare la lista di messaggi
- Navigare la lista di messaggi (Next, Prev)
- Mostrare quale elemento è selezionato

Ma cosa fare quando un Utente seleziona un messaggio è un altro Use Case, un'altra Responsabilità

Quindi teniamolo Fuori dal Message-List component.

93

Il Message-List Component

Vogliamo utilizzare un singolo componente per più cose, ad esempio

- per le mail presenti nella Inbox
- per le mail presenti in un singolo folder
- per mostrare i risultati della ricerca

Da dove prendiamo la lista dei messaggi?

Dove è memorizzato questo elenco?

Da chi viene utilizzato e navigato?

94

Message-List Component in Mail-view

```
<div>
  <section class="main-pane">
    <nls-message-list
      [messages]="messages">
    </nls-message-list>
  </section>
</div>
```

Descrizione

Componente utilizzato per visualizzare un elenco di messaggi

Deve prendere in input l'elenco dei messaggi

95

Message-List Component Definition

```
@Component({
  selector: 'nls-message-list',
  templateUrl: './message-list.component.html'
})
export class MessageListComponent {

  @Input() messages: Array<Message>;
  ...
}
```

Nota

L'input è direttamente accessibile come campo della classe e nel template del componente

```
if (this.messages.length > 0) {
  // do Something
}
```

```
<div *ngFor="let message of messages">
</div>
```

96

Azioni e Conseguenze

I Component devono Gestire Azioni con Conseguenze sia Interne che Esterne

Quando un Utente seleziona un Folder, avvengono due cose:

1. (Interna) Il Folder corrente deve essere evidenziato dal Folder-List
2. (Esterna) Gli altri Component devono essere notificati della selezione per
 - a. Eseguire Azioni
 - b. Abilitare Pulsanti
 - c. Aggiornare altre View

97

Azioni e Conseguenze

```
<div [ngClass]="{'message-current': message === currentMessage}"
  (click)="select(message)">
  {{message.subject}}
</div>
```

```
select(selectedMessage) {
  this.currentMessage = selectedMessage;
}
```

Descrizione

L'evento di click sul Subject della mail scatena l'esecuzione del metodo select(...).

select(...) modifica lo stato interno del componente e la sua View

98

Cambiamo insieme MailViewComponent

In MailViewComponent:

- visualizzare la sezione Compose
- quando viene cliccato Reply o Forward

Aggiungiamo la logica per il Reply nel MailViewComponent:

- Sender → To
- Subject → Re: Subject
- Body → '>' Body

99

Template-Driven Forms

- Setup veloce
- Basate sul concetto ben noto di **ngModel**
 - Simile a AngularJS
- Più difficile gestire dinamicamente aggiunta/modifica di campi

Angular DOCS: <https://angular.io/guide/forms-overview> Angular API: <https://angular.io/api/forms>

100

Template-Driven Forms - Steps

- Includere `FormsModule` nella sezione `imports` del `@NgModule`
- Aggiungere il tag `<form>`
- Includere l'attributo `name` per ogni tag `<input>`
 - es. `<input type="text" name="userName">`
- Aggiungere `DataBinding` al tag `<input>`
 - `[(ngModel)]="user.userName"`
- Opzionale: aggiungere validatori come `required`
 - `[required]="conditional expression"`

101

Template-Driven Forms - Avanzate

- Dare un nome alla form
 - `<form #draftForm="ngForm">`
- Permette di associare un identificativo alla istanza della direttiva `ngForm` all'interno del Template stesso
- Possiamo provare a stampare

```
<pre>Valid: {{draftForm.valid }}
Dirty: {{draftForm.dirty }}
Pristine: {{draftForm.pristine }}
</pre>
```

102

Template-Driven Forms - Stato

- `NgModel` aggiorna in automatico i seguenti proprietà `css`, `form`, e `model`
 - `form.valid` → `ng-valid` `css` class
 - `form.field.valid` → `ng-valid` `css` class
 - `form.field.invalid` → `ng-invalid` `css` class
 - calcolati ricorsivamente
- Altri valori
 - `valid` - `invalid`
 - `dirty` - `pristine`
 - `touched` - `untouched`

103

Template-Driven Forms - CSS

```
.ng-valid[required], .ng-valid.required {
  border-left: 5px solid #42A948; /* green */
}

.ng-invalid:not(form) {
  border-left: 5px solid #a94442; /* red */
}

form.ng-invalid {
  border: 1px solid #a94442; /* red */
}
```

104

Template-Driven Forms

- Resettare una Form allo stato iniziale
 - `userForm.reset()`
- Gestire la Submission

```
<form #draftForm="ngForm" (ngSubmit)="onSubmit()" novalidate>  
...  
</form>
```

- Prevenire Submit di Form non valide

```
<button type="submit" (click)="send()" [disabled]="draftForm.invalid">Send</button>
```

105

Lab 05 - Mail-Composer Component

Implementare Message-Composer Component

- Integrare il MailComposerComponent
- Includere in Component nel @NgModule dell'applicazione
- Dichiarare gli Input
 - draft
- Dichiarare gli Output
 - send
 - cancel
 - (opzionale) save
- Includere il componente nel Template del Parent Component

106

Lab 05 bis- Mail-Composer Component

Giocare con la Validation

- Controllare che il Subject contenga almeno 3 caratteri
- Disabilitare il pulsante Send quando la Form non è valida
- Visualizzare un messaggio di errore custom

107

Reactive Forms

- Robuste
- Scalabili
- Riusabili
- Testabili

If forms are a key part of your application, or you're already using reactive patterns for building your application, use reactive forms.

Angular DOCS: <https://angular.io/guide/forms-overview> Angular API: <https://angular.io/api/forms>

108

Reactive Forms - Steps

- Includere `ReactiveFormsModule` nella sezione `imports` del `@NgModule`
- L'elemento fondamentale è `FormControl`

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

@Component({
  selector: 'nis-message-search',
  templateUrl: './message-search.component.html',
  styleUrls: ['./message-search.component.scss']
})
export class MessageSearchComponent implements OnInit {
  ...
  searchString = new FormControl("");
```

109

Reactive Forms - Steps

- Registrare il `FormControl` nel Template

```
<div class="form-inline">
  Search messages:
  <input class="form-control" type="text" [formControl]="searchString">
  <button class="btn btn-primary"
    (click)="search()">
  Search <span class="glyphicon glyphicon-search"></span>
</button>
{{searchString.value}}
</div>
```

110

Reactive Forms - FormControl Value

Visualizzare il valore di un `FormControl`

- Nel Template:

```
Search messages:
<input class="form-control" type="text" [formControl]="searchString">
...
{{ searchString.value }}
```

- Nel Component ... Observable:

```
this.searchString.valueChanges.subscribe((value) => console.log('nuovo valore per FormControl', value));
```

111

Reactive Forms - Possibilità

- (Opzionale) Settare eventuali valori iniziali o su azioni
- (Opzionale) Aggiungere validatori

```
this.searchString.setValue(this.defaultQuery);
this.searchString.setValidators(Validators.required);
```

- Accedere allo stato della Form o dei suoi campi

```
Search messages:
<input class="form-control" type="text" [formControl]="searchString">
<button class="btn btn-primary"
  [disabled]="searchString.invalid" ←
  (click)="search()">
Search <span class="glyphicon glyphicon-search"></span>
</button>
```

112

Reactive Forms - Raggruppare Controlli

- E' possibile raggruppare i controlli utilizzando i **FormGroup**

```
import {FormGroup, FormControl, Validators} from
'@angular/forms';
...
draftForm: FormGroup;

constructor() {
  this.draftForm = new FormGroup({
    from: new FormGroup(""),
    to: new FormGroup(""),
    subject: new FormGroup(""),
    body: new FormGroup("")
  });
}
```

```
import {FormGroup, FormBuilder, Validators} from
'@angular/forms';
...
draftForm: FormGroup;

constructor(private fb: FormBuilder) {
  this.draftForm = this.fb.group({
    from: [this.accountEmail],
    to: ['', Validators.required],
    subject: [''],
    body: ['']
  });
}
```

113

Reactive Forms - Raggruppare Controlli

- Collegare il **FormGroup** nel Template

```
<form class="inline-form" novalidate [formGroup]="draftForm"> ← (ngSubmit)="send()"
  From: {{draftForm.value.from}}
  To:
  <input formControlName="to" require>
  Subject:
  <input formControlName="subject">
  <hr>
  Body:
  <textarea formControlName="body">
  </textarea>
  <hr>
  <button class="btn btn-primary" (click)="send()">Send</button> ← type="submit"
  </form> [disabled]="draftForm.invalid"

<pre>
Invalid: {{draftForm.invalid }}
</pre>
```

114

Lab Reactive Form

MessageComposer Component

- Trasformare la Template Form in una ReactiveForm

TODO

- Aggiungere il modulo **ReactiveFormsModule** negli `imports: []` del `@NgModule`
- Dependency Injection nel MessageComposerComponent
 - FormBuilder**

115

Lab Reactive Form

TODO

- Usare il **FormBuilder** per creare il **FormGroup** contenente i **FormControl**
 - from, to, subject e body
- Inserire nel tag `<form>` il collegamento al **FormGroup**
 - `[formGroup]="draftForm"`
- Inserire nei tag `<input>` i **FormControl**
 - es. `formControlName="subject"`

116

#ComponentId

Come abbiamo visto usando ngForm, possiamo assegnare un ID a un Componente e usarlo per accedere ai metodi della Form nel Template

```
<modal #errorModal>
  <div class="modal-content">
    <p>My modal content</p> </div>
  </modal>
...
<button (click)="errorModal.open()">Open Modal</button>
```

#ComponentId

117

Accesso al DOM

```
@Component({
  selector: 'my-app',
  template: '<h1>My App</h1>'
})
export class AppComponent implements AfterContentInit {
  node: string;

  constructor(private elementRef: ElementRef) {}

  ngAfterContentInit() {
    const tmp = document.createElement('div');
    const el = this.elementRef.nativeElement.cloneNode(true);
    tmp.appendChild(el);
    this.node = tmp.innerHTML;
  }
}
```

ElementRef

Questo componente stampa il suo stesso HTML

ElementRef utile quando si deve accedere direttamente al DOM

118

ElementRef - un esempio

ElementRef può essere utile nelle Direttive ... pensate proprio per modificare il DOM, aggiungere behavior.

```
import {Directive, ElementRef, HostListener} from '@angular/core';

@Directive({
  selector: '[nisHighlight]'
})
export class HighlightDirective {
  constructor(private el: ElementRef) {
    this.el.nativeElement.style.backgroundColor = 'yellow';
  }
}
```

```
<section nisHighlight>
  <nis-message-composer
    nisHighlight
    [draft]="draft"
    (saveMail)="updateDraft($event)"
    (sendMail)="send($event)"
    (cancelMail)="closeComposer()">
  </nis-message-composer>
</section>
```

119

@HostListener

E' un decoratore di Angular che:

- Dichiara un evento del DOM da ascoltare
 - click, mouseenter, mouseleave, ecc
- Fornisce un method handler da eseguire ogni volta che si verifica l'evento

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

@HostListener('mouseenter') onMouseEnter() {
  this.highlight('yellow');
}
```

120

Creiamo una Directive con Angular CLI

Spostatevi nella cartella:

- `projects/XX/xxbase/src/nis`
- eseguite il comando
 - `ng g directive directives/highlight`
- Facciamo in modo che la direttiva cambi il background color del tag a cui è applicata
- Proviamo la direttiva mettendola su un tag

121

@HostListener - un esempio

Usiamo il decoratore `HostListener` per migliorare un po' la nostra direttiva:

- cambiamo il colore di sfondo solo al passaggio del mouse

```
@Directive({
  selector: '[nisHighlight]'
})
export class HighlightDirective {
  constructor(private el: ElementRef) {
  }

  @HostListener('mouseenter') onMouseEnter() {
    this.el.nativeElement.style.backgroundColor = 'darkcyan';
  }

  @HostListener('mouseleave') onMouseLeave() {
    this.el.nativeElement.style.backgroundColor = null;
  }
}
```

122

Lab @HostListener

TODO

- Modificare la direttiva `nisHighlight` per fare in modo che:
- `@HostListener` per creare il `FormGroup` contenente i `FormControl`
 - `from`, `to`, `subject` e `body`
- Evidenzi l'elemento a cui è applicata solo al passaggio del mouse
 - Inserire `@HostListener`
 - eventi: `mouseenter`, `mouseleave`
- Applicare la direttiva agli elementi visualizzato dal `FolderList` component

123

LifeCycle Hooks

Un Component ha un ciclo di vita gestito da Angular che:

1. Crea e Renderizza i Components insieme ai loro figli
2. Controlla quando cambiano le loro proprietà
3. Li distrugge prima di rimuoverli dal DOM

Angular offre i `lifeCycle Hooks` per:

- Rendere visibili i momenti chiave della vita di un Component
- Poter agire quando si verificano

DOCS: <https://angular.io/docs/ts/latest/guide/lifecycle-hooks.html>

124

LifeCycle Hooks - ngOnChanges()

E il primo lifeCycle Hook eseguito da Angular dopo la creazione di un Component/Directive

- Scatenato quando Angular (ri)setta i valori delle proprietà di @Input
- SimpleChanges contiene
 - i valori modificati
 - current value
 - previous value

```
ngOnChanges(changes: SimpleChanges) {  
  for (let propName in changes) {  
    let chng = changes[propName];  
    let cur = JSON.stringify(chng.currentValue);  
    let prev = JSON.stringify(chng.previousValue);  
    this.changeLog.push(`${propName}: currentValue =  
    ${cur}, previousValue = ${prev}`);  
  }  
}
```

125

LifeCycle Hooks - ngOnInit()

Inizializza il Component dopo che Angular ha visualizzato per la prima volta le proprietà

- Eseguito una sola volta
- Dopo ngOnChanges

```
ngOnInit() {  
  console.log('onInit');  
}
```

Esempio: <https://stackblitz.com/angular/rmkvlejmkm?file=src%2Fapp%2Fapp.component.html>

126

LifeCycle Hooks - ngOnDestroy()

Utile per fare pulizia prima che il componente venga distrutto

- Avvisare altra parte dell'applicazione che il componente è stato distrutto
- Unsubscribe degli Observable
- Listener di Eventi

127

Lab LifeCycle Hooks

TODO

Aggiungere la callback onChange su MessageList Component

128

Lab 6

TODO

Integrare il MailComposer Component con il bottone Reply presente nel MessageViewer Component.

129

<|> [Pensare a Componenti](#)

Dependency Injection

Grande punto di forza di Angular, consente di iniettare dipendenze in diversi Component in tutta la nostra applicazione.

Dependency Injection \longleftrightarrow Providers in @NgModules

I **Service**:

- Istanziati in modo **Lazy** solo quando un componente ne ha bisogno
- Iniettati automaticamente
- Singleton

130

<|> [Pensare a Componenti](#)

Dependency Injection - Service

Usare il decoratore **@Injectable** sopra la classe

```
@Injectable()
export class MailMessageService {
  ...
}
```

Aggiungere il Service ai Provider

```
@NgModule({
  declarations: [ ... ],
  imports: [ ... ],
  providers: [MailMessageService]
})
```

131

<|> [Pensare a Componenti](#)

Dependency Injection - Optional Service

E' possibile dire a Angular che una dipendenza da un Service è opzionale, inserendo il decoratore **@Optional()**

```
constructor(@Optional() private logger?: Logger) {
  if (this.logger) {
    this.logger.log('some_message');
  }
}
```

- Se non viene registrato nessun Logger Provider allora l'Injector setterà null come valore di logger

```
{ { provide: Logger, useClass: Logger } }
oppure
{ { provide: Logger, useClass: LoggerImpService } }
```

132

Lab Servizi

TODO

Creare il FolderService e aggiungere alla classe i metodi

- getCustomFolders()
- getFolders()

Sostituire i folder mock cablati con le nuove chiamate esposte dal Service

Aggiungere il FolderService ai Provider nel @NgModule e nel costruttore del Component FolderList

133

<|> [Pensare a Componenti](#)

Injection Token

Si può creare un Token che può essere usato come DI Provider.

```
const TITLE = new InjectionToken<string>('title');
```

Registrarlo come Provider del nostro modulo

```
{ provide: APP_TITLE, useValue: 'Sonia WebApp Mail' }
```

Usarlo nei nostri componenti usando @Injector

```
constructor(@Inject(APP_TITLE) title: string, ...) {  
  this.title = title;  
}
```

134

<|> [Pensare a Componenti](#)

HttpClient

<https://angular.io/guide/http>

```
@NgModule({  
  declarations: [  
    NisMailViewComponent,  
    MailLogoComponent,  
    ...  
  ],  
  imports: [  
    BrowserModule,  
    FormsModule,  
    ReactiveFormsModule,  
    HttpClientModule  
  ],  
  providers: [  
    ...  
  ],  
  bootstrap: [NisMailViewComponent]  
})  
export class NisModule {  
}
```

Importare **HttpClientModule** nel nostro **@NgModule**

135

<|> [Pensare a Componenti](#)

HttpClient - GET

```
export class HeroService  
  
  private heroesUrl = 'api/heroes';  
  
  constructor(private http: HttpClient) {  
  }  
  
  getHeroes(): Observable<Array<Hero>> {  
    return this.http.get<Array<Hero>>(this.heroesUrl);  
  }  
}
```

Service

```
this.heroService.getHeroes().subscribe(  
  (heroes) => this.heroes = heroes,  
  (error) => // LOG ERROR ... DO SOMETHING  
);
```

Component

136

Typescript String Template

Comporre Urls e Strighe è più semplice usando String Template

```
let var1 = 1;
let var2 = 'my_var_2';
let url = `resource/${var1}/${var2}`;
```

Un esempio reale di utilizzo ...

```
import { HttpClient } from '@angular/common/http';
@Injectable()
export class HeroService {
  constructor(private http: HttpClient) {}

  getHero(id:string) {
    let url = `resource/${id}`;
    return this.http.get<Hero>({url}).toPromise();
  }
}
```

NOTA

Non usare per fare HTML Injection!

137

Lab HttpClient

TODO

- Includere HttpClient nel @NgModule
- Importare HttpClient nei Service (es MailService)
- Implementare la chiamata HTTP get per recuperare i messaggi

139

<|> [Pensare a Componenti](#)

HttpClient - POST

```
create(name: string): Observable<Hero> {
  let headers = new Headers({ 'Content-Type': 'application/json' });
  let options = new RequestOptions({ headers: headers });
  let body = { newName: name };
  return this.http.post<any>(`${this.heroesUrl}, body, options);
}
```

Service

Component

138

<|> [Pensare a Componenti](#)

Routing

Aggiungere il Base Url nel file index.html il

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

Aggiungere la dipendenza al progetto (node_modules)

```
npm install -g @angular/router --save
```

140

Routing - Definire Routes & RoutingModule

Definire le Route dell'applicazione nel file `nis-routing.module.ts`

```
const routes: Routes = [
  {path: 'messages', component: MessageListComponent},
  {path: 'message/:id', component: MessageDetailComponent},
  {
    path: 'inbox',
    component: MessageListPageComponent,
    data: {title: 'Inbox', folder: 'inbox'}
  },
  {
    path: '',
    redirectTo: '/inbox',
    pathMatch: 'full'
  },
  {path: '**', component: PageNotFoundComponent}];
```

141

Routing - Definire Routes & RoutingModule

Creare un modulo contenente le nostre **Routes**

```
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class NisRoutingModule {
}
```

Includere il modulo con le **Routes** nel modulo della nostra applicazione

142

Routing - Includere RoutingModule

Includere il modulo con le **Routes** nel modulo della nostra applicazione

```
@NgModule({
  declarations: [
    ...
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule,
    ReactiveFormsModule,
    NisRoutingModule
  ],
  providers: [
    ...
  ],
  bootstrap: [NisMailViewComponent]
})
export class NisModule {
}
```

Routing Module

143

Router-Outlet

Nel Root-Component, cioè il componente di più alto livello occorre inserire il placeholder:

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

- E' possibile avere più **RouterOutlet**
- Devono avere nome univoco → usando attributo **name**
- Il **name** non può essere settato o cambiato dinamicamente
- Se non è specificato l'attributo name il valore di default è **'primary'**

Angular: <https://angular.io/api/router/RouterOutlet>

144

Navigare tra le Route

```
<nav class="page-left-menu">
  <section class="compose-toolbar">
    <a routerLink="/compose" routerLinkActive="active">Compose</a>
  </section>
  ...
</nav>
```

```
this.router.navigate(['/inbox']);
```

145

Routed Components - Pages

Sono Component che recuperano:

- Parametri
- Stato

Dalla Route corrente

147

Passare e Leggere Parametri della Route

Passare un parametro a una Route che lo richiede:

```
{path: 'message/:id', component: MessageDetailComponent},
```

```
this.router.navigate(['/message', message.id]);
```

Recuperare il parametro da Component/Resolver:

```
ngOnInit() {
  // (*) converts string 'id' to a number
  let id = +this.route.snapshot.params['id'];
}
```

route → **ActivatedRoute**

146

Angular CLI

148

Angular Cli - Application

Come aggiungere una nuova Application al progetto

```
ng g application <application>
```

Esempio

```
ng g application ProvaApp
```

149

Angular Cli - Component

Come Aggiungere un Component al progetto demo

```
ng g component components/<nome-component>  
--project=<project>
```

Esempio

```
ng g component components/mail-logo --project=01base
```

150

Angular Cli - service

Come aggiungere un Service al progetto demo

```
ng g service services/<nome> --project=<project>
```

Esempio

```
ng g service services/mail-message --project=01base
```

151

—

Librerie Utilizzate

152

Compodoc

Tool utilissimo per la generazione automatica della Documentazione.

<https://compodoc.app/>

- `npm i --save-dev @compodoc/compodoc`
- `npm install -g @compodoc/compodoc`

Modificare il file `tsconfig.json`

```
"include": [  
  "projects/06/6xdemo/**/*"  
],  
"exclude": [  
  "node_modules",  
  "**/*.spec.ts"  
]
```

153

NVM - Node Version Manager

Tool utilissimo per tenere aggiornato Node.js e gestire più versioni contemporaneamente

<https://github.com/nvm-sh/nvm>

Guida per l'installazione di NVM su Windows

<https://docs.microsoft.com/it-it/windows/nodejs/setup-on-windows>

154

Librerie per Fake REST

- <https://github.com/typicode/json-server>
- <https://github.com/Marak/faker.js>

Link Utili

Link Utili

- Clean Code: the book
 - ◆ https://books.google.it/books/about/Clean_Code.html?id=hjEFCAAQBAJ
- RxJs
 - ◆ <https://rxjs-dev.firebaseapp.com/>
- RxJs Operatori
 - ◆ <https://github.com/btroncone/learn-rxjs/blob/master/operators/complete.md>
 - ◆ (editor on-line) <https://rxviz.com/>
 - ◆ descrizione visiva operatori <https://rxmarbles.com/>
- Compodoc
 - ◆ <https://compodoc.app/>



157

Link Utili

- Clean Code: the book
 - ◆ https://books.google.it/books/about/Clean_Code.html?id=hjEFCAAQBAJ
- Esempi @HostListener e @HostBinding
 - ◆ <https://alligator.io/angular/hostbinding-hostlistener/>
 - ◆



158

Contatti

- **E-mail**
sonia.pini@nispro.it
- **Skype**
sonia.pini

159