Angular 9

Sviluppare un'applicazione component-base

Prima di iniziare

Download repository con i laboratori

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

Hardware & Software necessari

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

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

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

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
  - Classificazione dei componenti
    - “Smart”, “Dumb” e “Stateless”

Varie ed Eventuali

- Angular CLI
- Librerie Utilizzate
- Link Utili

Laboratori del Corso

- Lab 0 - Hello world Angular Application
- Lab IC - 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

Approccio del Corso

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

Clean Code... Sempre

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

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

AngularJs vs Angular

Angular Evolution...

AngularJs 1.5
JavaScript
one-way bindings

Angular 2
Typescript
Mobile-oriented

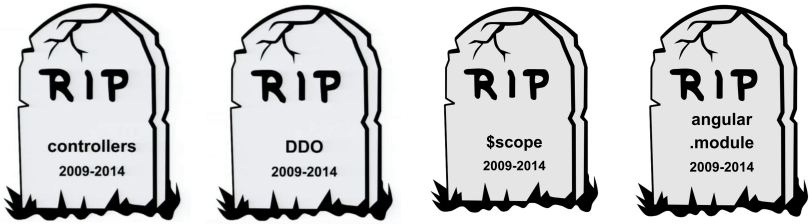
Angular 4
Typescript 2.1
HTTPClient

Angular 5
increase standardization
@angular/http deprecated

Angular 6
Angular CLI - ng-update
RxJs

Angular 8
IVY Pre-view

Concetti Eliminati



Presentazione: https://youtu.be/gNmWybAyBHI

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

I Cicli

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

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

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

Aggiungere o rimuovere dal DOM

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

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

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

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

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

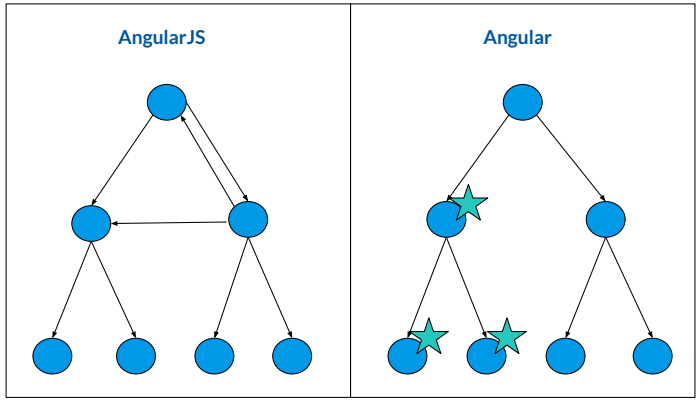
Angular: https://angular.io/guide/ajs-quick-reference

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.

Tree traversing in AngularJs vs Angular



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.

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
- Cancellabile
  - `unsubscribe()`
- Operators
  - `map, forEach, filter, ...`

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...*

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
```


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
```


Change Detection & RxJs: dal vivo

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

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.

Angular Key Concepts

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

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

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

Components

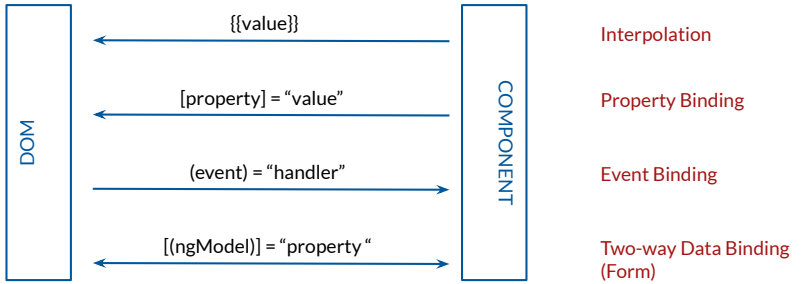
```
@Component({
  selector:    'app-hero-list',
  templateUrl: './hero-list.component.html'
  })
export class HeroListComponent implements OnInit {
/* . . . */
}
```

Metadata

Lifecycle Hooks

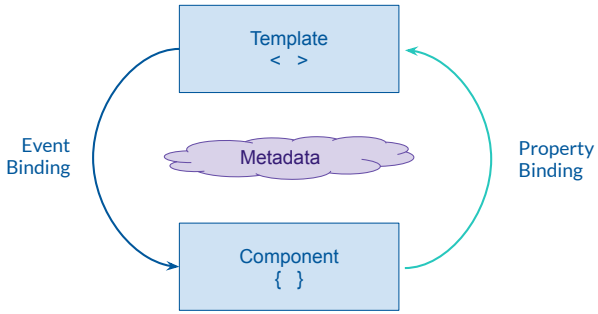
Binding

Il diagramma mostra le 4 forme di data binding



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

Data Binding



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

- ‘&’

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>
```


Lab 0 - Angular Hello word Component

Vedere project/00/helloword-base

Struttura dell'applicazione Angular

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

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 helloword-base
- funziona? cosa manca?

Services

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

Pensare a Componenti

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**

Lab IC - Identificare Componenti

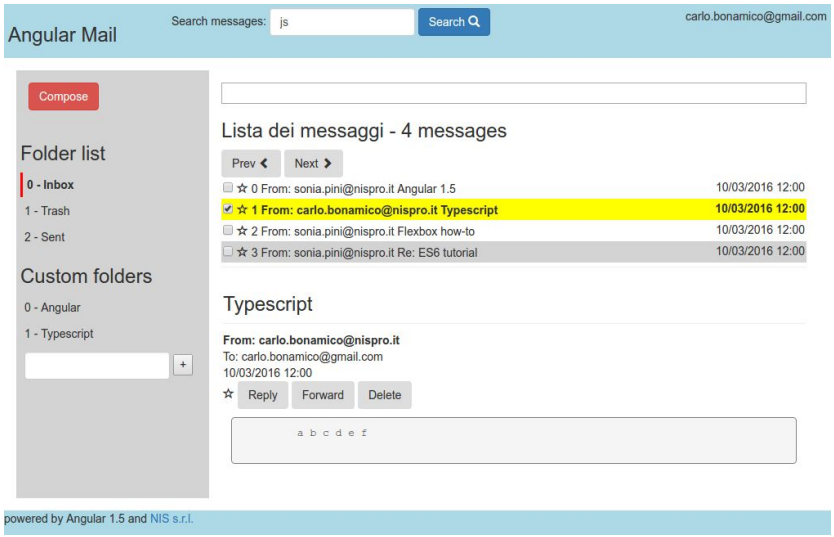
Identificare i componenti chiave in una tipica WebMail application

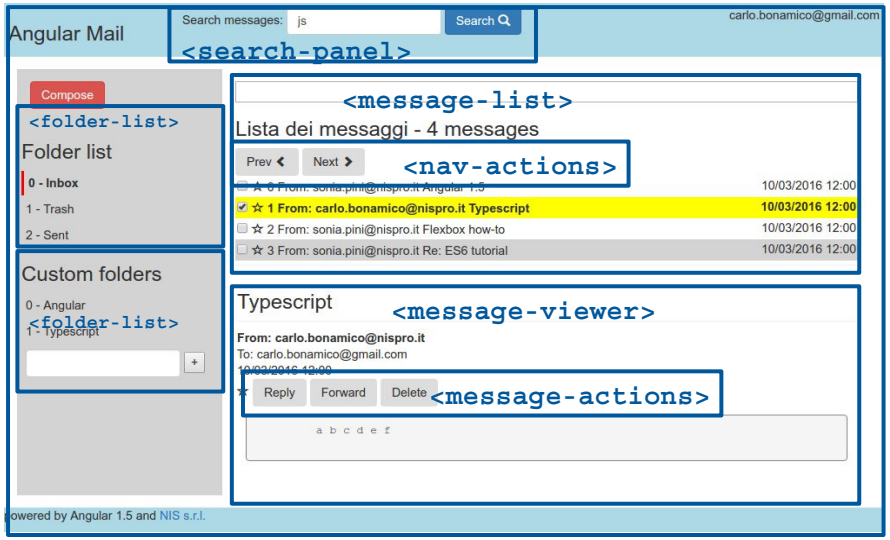
Analizzare quali componenti possono essere riutilizzati in più view

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





Vantaggi nell'uso di Componenti

Stronger Encapsulation (scope isolato + binding espliciti)

- Modificare l'implementazione interna del componente ha meno impatto sul resto dell'application
- 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

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

Component-Based UI

AngularJs

Sviluppare Applicazioni Component-Based era possibile, ma

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

Angular

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

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

Lab 01 - Mail-logo Component

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

1. Aprire l'application **`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

Lab 01 - TIPS

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

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

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

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

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

Lab 02 - Message-Viewer Component

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

1. Aprire l'application **`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

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**!

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

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.

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?

Folder-List Component in Mail-view

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

Descrizione

Componente utilizzato per visualizzare un elenco dei Folder

Deve prendere in input l'elenco dei Folder

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>;
  . . .
```

Nota

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

L'Output è un EventEmitter

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

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

</div>
```

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

Azioni e Conseguenze - Inside Component

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

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

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

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

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

@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

@Output - How to do

2. invocare la callback quando il Folder è selezionato

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

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


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

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

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

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

Lab 03 - extra

Passare al componente un parametro @Input addizionale

- allowCreate ti tipo boolean

TODO

Gestire nel Template del Component la possibilità di creare nuove Folder

Transclude Contents

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

In modo analogo alla direttiva `ng-transclude`.

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



```
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>
```

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

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

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-ngtemplateoutlet/

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

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

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

RISULTATO

Non viene disegnato nulla sullo schermo.

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

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>
```


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)

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"
]
```


Compodoc

Per generare la documentazione e lanciare il server locale

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


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

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.

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?

Message-List Component in Mail-view

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

Descrizione

Componente utilizzato per visualizzare un elenco di messaggi

Deve prendere in input l'elenco dei messaggi

Message-List Component Definition

```
@Component({
 selector: 'nis-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>
```


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

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

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

Template-Driven Forms

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

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"`

Template-Driven Forms - Avanzate

- Dare un nome alla form
  - <form #userForm="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>
```


Template-Driven Forms - Stato

- NgModel aggiorna in automatico i seguenti proprietà css, form, e model
  - form.valid → ng-valid css class
  - from.field.valid → ng-valid css class
  - form.field.invalid → ng-invalid css class
  - calcolati ricorsivamente

- Altri valori
  - valid - invalid
  - dirty - pristine
  - touched - untouched

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 */
}
```


Template-Driven Forms

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

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

- Prevenire Submit di From non valide

```
<button class="btn btn-primary" (click)="send()" [disabled]="draftForm.invalid">Send</button>
```

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

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

Angular CLI

Angular Cli - Application

Come aggiungere una nuova Application al progetto

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

Esempio

```
ng g application ProvaApp
```

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
```


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
```


Librerie Utilizzate

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"
]
```

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

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=hjEFCAAAQBAJ
- RxJs
  - https://rxjs-dev.firebaseapp.com/
- RxJs Operatori
  - https://github.com/btroncone/learn-rxjs/blob/master/operators/complete.md
- Compodoc
  - https://compodoc.app/

Link Utili

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

Contatti

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