Esercitazione Angular - Lesson 1

Progetto Shop Online

Obiettivo: costruire uno shop online con le funzionalita' base come vetrina, un carrrello e la gestione dell'utente

Agenda

1. **Setup** — Angular CLI, struttura progetto
2. **Scaffolding** — `ng new`, stand-alone, routing, stili
3. **Feature: Products** — componenti, modelli TS, mock data
4. **Recap & checkpoint**

1) Setup

Prerequisiti rapidi

- **Node LTS ≥ 18** — `node -v`
- **npm** — `npm -v`
- **Editor**: VS Code (estensioni: Angular Language Service, ESLint)
- **Angular CLI**:

```
npm i -g @angular/cli
ng version
```

Crea progetto base

Attività:

- Genera il progetto `shop` (standalone, routing, SCSS) nella cartella corrente.
- Verifica versione Angular CLI (`ng version`) prima di creare il progetto.
- Dopo lo scaffold entra in `shop/` ed avvia il server di sviluppo.
- Controlla che i file chiave siano creati: `src/main.ts`, `src/app/app.config.ts`, `src/app/app.routes.ts`.
- Apri il browser e verifica che l'app giri senza errori console.

```
ng new shop --standalone --routing --style=scss
cd shop
ng serve -o
```

Scelte: Standalone Components, Routing abilitato, SCSS.

2) Scaffolding

Aggiungi cartelle per domini

```
src/app/
  core/          # servizi, interceptor, guard, modelli comuni
  shared/        # componenti/pipes riusabili
  features/
    products/    # area funzionale "prodotti"
```

Componenti iniziali

- Crea la struttura della feature `products` dentro `src/app/features/products/`.
- Genera `product-card` (presentazionale) e `product-list` (pagina navigabile). Usa standalone e, per la card, inline template/style.

```
ng generate component features/products/product-card
              --inline-style --inline-template --standalone

ng generate component features/products/product-page
              --standalone
```

Componenti iniziali

- Verifica che i nuovi file siano in: `src/app/features/products/product-card/` e `src/app/features/products/product-list/`
- Aggiorna eventuali import negli `app.routes.ts` quando la pagina sarà collegata.
- Crea un servizio dati sotto `src/app/core/` per future chiamate HTTP / mock.

```
# Servizio API (bozza)
ng generate service core/services/products
```

Modello dati & mock

Attività:

- Crea il file modello in `src/app/core/models/product.ts`.
- Campi minimi da includere (tipi + scopo):
  - `id: string` — identificatore univoco.
  - `title: string` — nome descrittivo del prodotto.
  - `price: number` — prezzo lordo in EUR.
  - `thumbnail?: string` — path relativo immagine (opzionale).
  - `tags?: string[]` — lista categorie / etichette (opzionale).
  - `createdAt: string` — ISO date string (per ordinamenti / future funzioni).
- Implementa un mock in `src/app/core/mocks/products.mock.ts` con almeno 2 prodotti.
- Usa `as const` se vuoi rendere il mock readonly.

```ts
// src/app/core/models/product.ts
export interface Product {
  id: string;
  title: string;
  price: number;
  thumbnail?: string;
  tags?: string[];
  createdAt: string;
}

// src/app/core/mocks/products.mock.ts
import { Product } from '../models/product';

export const PRODUCTS_MOCK: Product[] = [
  { id: 'p1', title: 'T-Shirt Angular',
    price: 19.9, thumbnail: 'assets/tshirt.png',
    tags: ['apparel'], createdAt: '2025-10-01T09:00:00Z' },
  { id: 'p2', title: 'Mug TypeScript',
    price: 12.5, thumbnail: 'assets/mug.png',
    tags: ['home'], createdAt: '2025-10-01T09:00:00Z' }
] as const;
```

Attività (ProductCard):

- Implementa `ProductCardComponent` in `src/app/features/products/product-card.component.ts` come standalone.
- Proprietà `@Input() product` obbligatoria: mostra `title`, `price` e opzionalmente `thumbnail`.
- Evento `@Output() add` che emette il `Product` quando si clicca il bottone.
- Usa `CurrencyPipe` per formattare il prezzo in EUR.
- Aggiungi attributi di accessibilità: `alt` sulle immagini, `aria-label` sul bottone.
- Stili minimi (bordo, padding) direttamente nel decorator (inline styles) per rapidità.

```ts
@Component({
  selector: 'app-product-card',
  standalone: true,
  template: `
    <article class="card">
      <h3>{{ product.title }}</h3>
      <p>{{ product.price | currency:'EUR':'symbol':'1.2-2' }}</p>
      <button (click)="addToCart(product)">Aggiungi</button>
    </article>
  `
})
export class ProductCardComponent {
  @Input({ required: true }) product!: Product;
  @Output() add = new EventEmitter<Product>();
  addToCart(p: Product) { this.add.emit(p); }
}
```

3) Feature: Products

Lista prodotti — pagina

Attività:

- Crea `ProductListPage` in `src/app/features/products/product-list.page.ts` (standalone component).
- Inietta il servizio prodotti (via `inject(ProductService)` o costruttore)
- Template: titolo, lista `<ul>` con `<li>` e dentro la card (`<app-product-card>`).
- Usa `@for` + `track` per efficienza
- Nel gestore `(add)` stampa su console o prepara stub per carrello (`onAdd`).
- Aggiungi stili minimi responsive con CSS Grid inline.

```ts
// src/app/features/products/product-list.page.ts
import { Component, inject } from '@angular/core';
import { AsyncPipe, NgFor, DatePipe, CurrencyPipe } from '@angular/common';
import { Product } from '../../core/models/product';
import { ProductService } from '../../core/product.service';
import { ProductCardComponent } from './product-card.component';

@Component({
  selector: 'app-product-list',
  standalone: true,
  imports: [NgFor, AsyncPipe, DatePipe, CurrencyPipe, ProductCardComponent],
  template: `
    <section>
      <h2>Prodotti</h2>
      <ul class="grid">
        <li *ngFor="let p of products$ | async; trackBy: trackById">
          <app-product-card [product]="p" (add)="onAdd($event)"></app-product-card>
        </li>
      </ul>
    </section>
  `,
  styles: [`
    .grid { display: grid; gap: 1rem; grid-template-columns: repeat(auto-fit, minmax(220px, 1fr)); }
  `]
})
export class ProductListPage {
  private service = inject(ProductService);
  products = this.service.list();
  trackById = (_: number, p: Product) => p.id;
  onAdd(p: Product) { console.log('ADD', p.id); /* TODO: integrare carrello */ }
}
```

Card prodotto — presentazionale

- Conferma che il file è `src/app/features/products/product-card.component.ts` (o in sottocartella dedicata).
- Mostra immagine se presente (`*@if` su `thumbnail`).
- Formatta il prezzo con `currency:'EUR'` .
- Emissione evento `add` su click del bottone.
- Migliora semantica: usa `<article>` e heading `<h3>` .
- Mantieni il componente privo di logica business (delegata alla pagina o servizi).

```ts
// src/app/features/products/product-card.component.ts
import { Component, EventEmitter, Input, Output } from '@angular/core';
import { CurrencyPipe } from '@angular/common';
import { Product } from '../../core/models/product';

@Component({
  selector: 'app-product-card',
  standalone: true,
  imports: [CurrencyPipe],
  template: `
    <article class="card">
      <img *ngIf="product.thumbnail" [src]="product.thumbnail" [alt]="product.title" />
      <h3>{{ product.title }}</h3>
      <p>{{ product.price | currency:'EUR' }}</p>
      <button aria-label="Aggiungi al carrello" (click)="add.emit(product)">Aggiungi</button>
    </article>
  `,
  styles: [`
    .card { padding: 1rem; border: 1px solid #e5e7eb; border-radius: .75rem; }
    img { max-width: 100%; border-radius: .5rem; }
    button { margin-top: .5rem; }
  `]
})
export class ProductCardComponent {
  @Input({ required: true }) product!: Product;
  @Output() add = new EventEmitter<Product>();
}
```

Routing + Service

Routing: aggiungi la rotta /products

Attività:

- Apri/crea `src/app/app.routes.ts` e definisci l'array `Routes`.
- Aggiungi redirect dalla root `''` a `products` con `pathMatch: 'full'`.
- Lazy-load della pagina prodotti via `loadComponent` (import dinamico).
- Aggiungi rotta wildcard `**` per una futura pagina 404 (`NotFoundPage`).
- Assicurati che nel bootstrap (`main.ts` o `app.config.ts`) sia presente `provideRouter(routes)`.
- Test: naviga a `/products` e verifica component render.

```ts
// src/app/app.routes.ts
import { Routes } from '@angular/router';

export const routes: Routes = [
  { path: '', redirectTo: 'products', pathMatch: 'full' },
  { path: 'products', loadComponent: () =>
      import('./features/products/product-list.page').then(m => m.ProductListPage) },
  { path: '**', loadComponent: () =>
      import('./shared/not-found.page').then(m => m.NotFoundPage) }
];
```

Imposta la rotta e verifica che `/products` mostri la lista.

Service con Dependency Injection

Attività:

- Usare `src/app/core/services/product.ts con` @Injectable({ providedIn: 'root' })`.
- Importa `PRODUCTS_MOCK` e restituisci un `Product[]`
- Integra nella pagina: `products = productService.list()`.
- Logga eventuali errori in console (placeholder per gestione errori futura).

```ts
// src/app/core/product.service.ts
import { Injectable } from '@angular/core';
import { Product } from '../models/product';
import { PRODUCTS_MOCK } from '../mocks/products';

@Injectable({ providedIn: 'root' })
export class ProductService {
  list(): Product[] {
    return PRODUCTS_MOCK;
  }
}
```

Filtri UI

Attività:

- Aggiungi dei filtri per Titolo
- Ricara gli elementi della lista una volta cambiato il filtro

product-list.html

```
<form class="filters" (submit)="$event.preventDefault()">
  <div appearance="outline">
      <p>Titolo</p>
      <input
          matInput
          type="search"
          name="title"
          placeholder="Cerca per titolo"
          [(ngModel)]="filters.title"
          (ngModelChange)="applyFilters()"
      />
  </div>
</form>
```

product-list.ts

```
protected filters = {
  title: ''
};

applyFilters() {
  const title = this.filters.title.trim().toLowerCase();
  const minPrice = this.parseNumber(this.filters.minPrice);
  const maxPrice = this.parseNumber(this.filters.maxPrice);

  this.filteredProducts = this.products.filter((product) => {
    const matchesTitle = !title || product.title.toLowerCase().includes(title);
    return matchesTitle;
  });
}
```

UI - Material

Attività:

- Aggiungi una libreria per la gestione della UI come Angular Material
- Adatta i componenti fino ad ora utilizzati alla nuova libreria
- Aggiungi un Header con un logo
- Effettua i miglioramenti di UI

Angular Material

https://material.angular.dev/

```
ng add @angular/material
```

```
<mat-form-field appearance="outline">
    <mat-label>Titolo</mat-label>
    <mat-icon matPrefix>search</mat-icon>
    <input
        matInput
        type="search"
        name="title"
        placeholder="Cerca per titolo"
        [(ngModel)]="filters.title"
        (ngModelChange)="applyFilters()"
    />
</mat-form-field>
```

Angular Material

```
<mat-card appearance="outlined">
  @if (product.thumbnail) {
    <img mat-card-image [src]="product.thumbnail" [alt]="product.title" />
  }
  <mat-card-header>
    <mat-card-title>{{ product.title }}</mat-card-title>
    <mat-card-subtitle>{{ product.price | currency:'EUR' }}</mat-card-subtitle>
  </mat-card-header>
  <mat-card-content>
    <ng-content></ng-content>
  </mat-card-content>
  <mat-card-actions align="end">
    <button
          mat-flat-button color="primary"
          (click)="add.emit(product)">
      <mat-icon aria-hidden="true">add_shopping_cart</mat-icon>
      Aggiungi
    </button>
  </mat-card-actions>
</mat-card>
```

Header

```
<!-- header.html -->
<mat-toolbar color="primary" class="app-toolbar">
  <button mat-icon-button class="app-toolbar__menu" *ngIf="showIcon" aria-label="Apri menu">
    <mat-icon>menu</mat-icon>
  </button>
  <span class="app-title">{{ title | titlecase }}</span>
</mat-toolbar>
```

```
<app-header [title]="title()"></app-header>

<main class="app-main">
        <section class="app-content">
                <router-outlet></router-outlet>
        </section>
</main>
```