Caso Práctico

Configuración de un Proyecto Angular con Angular CLI: Guía Paso a Paso

Índice

---

1. Introducción a Angular y Angular CLI

Angular es un framework de desarrollo web de código abierto mantenido por Google y una comunidad de individuos y empresas. Angular permite desarrollar aplicaciones web de una sola página (SPA) con una arquitectura modular, reutilizable y mantenible.

Angular CLI (Command Line Interface) es una herramienta poderosa que facilita la creación, desarrollo, pruebas y despliegue de aplicaciones Angular. Con Angular CLI, puedes generar componentes, servicios, módulos y más con simples comandos, estandarizando la estructura del proyecto y mejorando la productividad.

---

2. Requisitos Previos

Antes de comenzar, asegúrate de tener lo siguiente instalado en tu máquina:

- **Node.js**: Angular requiere Node.js. Puedes descargarlo desde nodejs.org.
- **npm (Node Package Manager)**: Se instala automáticamente con Node.js.
- **Un Editor de Código**: Se recomienda Visual Studio Code, pero puedes usar cualquier editor de tu preferencia.

---

3. Instalación de Angular CLI

1. **Verificar la Instalación de Node.js y npm:**

   Abre tu terminal o línea de comandos y ejecuta:

   ```
   node -v
   npm -v
   ```

   Deberías ver las versiones instaladas de Node.js y npm.

2. **Instalar Angular CLI Globalmente:**

   Ejecuta el siguiente comando para instalar Angular CLI de forma global:

   ```
   npm install -g @angular/cli
   ```

```

**Nota:** Puedes necesitar permisos de administrador. Si encuentras errores, intenta ejecutarlo con `sudo` (en sistemas Unix) o abre la terminal como administrador (en Windows).

3. **Verificar la Instalación de Angular CLI:**

```
ng version
```

Deberías ver información sobre la versión de Angular CLI y otras dependencias.

---

## 4. Creación de un Nuevo Proyecto Angular

1. **Crear un Proyecto Nuevo:**

   En tu terminal, navega al directorio donde deseas crear el proyecto y ejecuta:

```
ng new crud-workshop
```

   - `crud-workshop` es el nombre del proyecto. Puedes cambiarlo según tus preferencias.

   Durante la creación, Angular CLI te hará algunas preguntas:

   - **Would you like to add Angular routing?** `Yes`
   - **Which stylesheet format would you like to use?** `CSS` (o `SCSS` si prefieres Sass)

2. **Navegar al Directorio del Proyecto:**

```
cd crud-workshop
```
```

3. **Iniciar el Servidor de Desarrollo:**

```
ng serve
```

Nota: Puedes especificar un puerto diferente si es necesario, por ejemplo, `ng serve --port 4201`.

Abre tu navegador y navega a `http://localhost:4200/` para ver la aplicación en funcionamiento.

---

5. Estructura del Proyecto

Una vez creado el proyecto, la estructura básica será similar a la siguiente:

```
crud-workshop/
├── e2e/
├── node_modules/
├── src/
│   ├── app/
│   │   ├── app-routing.module.ts
│   │   ├── app.component.css
│   │   ├── app.component.html
│   │   ├── app.component.spec.ts
│   │   ├── app.component.ts
│   │   └── app.module.ts
│   ├── assets/
│   ├── environments/
│   ├── styles.css
│   └── index.html
├── angular.json
├── package.json
```

```
└── README.md
```

- `src/app/`: Contiene los componentes, servicios y módulos de la aplicación.
- `app-routing.module.ts`: Archivo para configurar las rutas de la aplicación.
- `styles.css`: Archivo de estilos globales.
- `index.html`: Punto de entrada de la aplicación.

---

6. Configuración de Rutas

Para manejar la navegación entre diferentes vistas (componentes) de tu aplicación, configuraremos rutas.

1. **Editar `app-routing.module.ts`:**

```
// src/app/app-routing.module.ts
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { UserListComponent } from './components/user-list/
user-list.component';
import { UserCreateComponent } from './components/user-cre
ate/user-create.component';
import { UserEditComponent } from './components/user-edit/
user-edit.component';

const routes: Routes = [
  { path: '', component: UserListComponent },
  { path: 'create', component: UserCreateComponent },
  { path: 'edit/:id', component: UserEditComponent },
  { path: '**', redirectTo: '' } // Redirige rutas descono
cidas a la lista de usuarios
];

@NgModule({
```

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

2. **Asegurar que `app.module.ts` Incluya los Componentes y `FormsModule`:**

```
// src/app/app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms'; // Importar
FormsModule

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { UserListComponent } from './components/user-list/
user-list.component';
import { UserCreateComponent } from './components/user-cre
ate/user-create.component';
import { UserEditComponent } from './components/user-edit/
user-edit.component';

@NgModule({
  declarations: [
    AppComponent,
    UserListComponent,
    UserCreateComponent,
    UserEditComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule // Añadir FormsModule
```

```
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

---

7. Creación de Componentes CRUD

Crearemos tres componentes principales para manejar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar):

- `UserListComponent`: Para listar usuarios.
- `UserCreateComponent`: Para agregar un nuevo usuario.
- `UserEditComponent`: Para editar un usuario existente.

a. Componente Lista de Usuarios (`UserListComponent`)

1. **Generar el Componente:**

```
ng generate component components/user-list
```

2. **Editar `user-list.component.html`:**

```
<!-- src/app/components/user-list/user-list.component.html
-->
<div class="user-list-container">
  <h2>Lista de Usuarios</h2>
  <button (click)="navigateToCreate()" class="pill">Agrega
r Usuario</button>
  <table class="user-table">
    <thead>
      <tr>
        <th>ID</th>
        <th>Nombre</th>
```

```
        <th>Email</th>
        <th>Acciones</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let user of users">
        <td>{{ user.id }}</td>
        <td>{{ user.name }}</td>
        <td>{{ user.email }}</td>
        <td>
          <button (click)="editUser(user.id)" class="pil
l">Editar</button>
          <button (click)="deleteUser(user.id)" class="pil
l">Eliminar</button>
        </td>
      </tr>
    </tbody>
  </table>
</div>
```

3. **Editar** `user-list.component.ts`**:**

```
// src/app/components/user-list/user-list.component.ts
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { UserService } from '../../services/user.service';

interface User {
  id: number;
  name: string;
  email: string;
}
```

```
@Component({
  selector: 'app-user-list',
  templateUrl: './user-list.component.html',
  styleUrls: ['./user-list.component.css']
})
export class UserListComponent implements OnInit {
  users: User[] = [];

  constructor(private userService: UserService, private ro
uter: Router) { }

  ngOnInit(): void {
    this.getUsers();
  }

  getUsers(): void {
    this.users = this.userService.getUsers();
  }

  navigateToCreate(): void {
    this.router.navigate(['/create']);
  }

  editUser(id: number): void {
    this.router.navigate(['/edit', id]);
  }

  deleteUser(id: number): void {
    this.userService.deleteUser(id);
    this.getUsers(); // Actualizar la lista después de eli
minar
  }
}
```

b. Componente Crear Usuario (`UserCreateComponent`)

1. **Generar el Componente:**

```
ng generate component components/user-create
```

2. **Editar** `user-create.component.html`**:**

```
<!-- src/app/components/user-create/user-create.component.
html -->
<div class="form-container">
  <h2>Agregar Nuevo Usuario</h2>
  <form (ngSubmit)="addUser()" class="form">
    <div class="form-group">
      <label for="name">Nombre:</label>
      <input id="name" [(ngModel)]="name" name="name" requ
ired class="input-field" />
    </div>
    <div class="form-group">
      <label for="email">Email:</label>
      <input id="email" [(ngModel)]="email" name="email" r
equired class="input-field" />
    </div>
    <div class="button-group">
      <button type="submit" class="pill">Agregar</button>
      <button type="button" (click)="cancel()" class="pil
l">Cancelar</button>
    </div>
  </form>
</div>
```

3. **Editar** `user-create.component.ts`**:**

```
// src/app/components/user-create/user-create.component.ts
import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { UserService } from '../../services/user.service';

@Component({
  selector: 'app-user-create',
  templateUrl: './user-create.component.html',
  styleUrls: ['./user-create.component.css']
})
export class UserCreateComponent {
  name: string = '';
  email: string = '';

  constructor(private userService: UserService, private ro
uter: Router) { }

  addUser(): void {
    if (this.name && this.email) {
      this.userService.addUser({ name: this.name, email: t
his.email });
      this.router.navigate(['/']);
    }
  }

  cancel(): void {
    this.router.navigate(['/']);
  }
}
```

c. Componente Editar Usuario (`UserEditComponent`)

1. **Generar el Componente:**

```
ng generate component components/user-edit
```

2. **Editar** `user-edit.component.html`**:**

```
<!-- src/app/components/user-edit/user-edit.component.html
-->
<div class="form-container" *ngIf="user">
  <h2>Editar Usuario</h2>
  <form (ngSubmit)="updateUser()" class="form">
    <div class="form-group">
      <label for="name">Nombre:</label>
      <input id="name" [(ngModel)]="user.name" name="name"
required class="input-field" />
    </div>
    <div class="form-group">
      <label for="email">Email:</label>
      <input id="email" [(ngModel)]="user.email" name="ema
il" required class="input-field" />
    </div>
    <div class="button-group">
      <button type="submit" class="pill">Actualizar</butto
n>
      <button type="button" (click)="cancel()" class="pil
l">Cancelar</button>
    </div>
  </form>
</div>
```

3. **Editar** `user-edit.component.ts`**:**

```
// src/app/components/user-edit/user-edit.component.ts
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { UserService } from '../../services/user.service';

interface User {
  id: number;
  name: string;
  email: string;
}

@Component({
  selector: 'app-user-edit',
  templateUrl: './user-edit.component.html',
  styleUrls: ['./user-edit.component.css']
})
export class UserEditComponent implements OnInit {
  user: User | undefined;

  constructor(
    private route: ActivatedRoute,
    private userService: UserService,
    private router: Router
  ) { }

  ngOnInit(): void {
    const id = Number(this.route.snapshot.paramMap.get('i
d'));
    this.user = this.userService.getUserById(id);
  }

  updateUser(): void {
    if (this.user) {
      this.userService.updateUser(this.user);
```

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

  cancel(): void {
    this.router.navigate(['/']);
  }
}
```

---

8. Implementación de un Servicio para CRUD

Para manejar los datos de usuarios, crearemos un servicio que simulará operaciones CRUD. En un proyecto real, este servicio interactuaría con una API backend.

1. **Generar el Servicio:**

```
ng generate service services/user
```

2. **Editar `user.service.ts`:**

```
// src/app/services/user.service.ts
import { Injectable } from '@angular/core';

interface User {
  id: number;
  name: string;
  email: string;
}

@Injectable({
  providedIn: 'root'
```

```
})
export class UserService {
  private users: User[] = [
    { id: 1, name: 'Juan Pérez', email: 'juan.perez@exampl
e.com' },
    { id: 2, name: 'María Gómez', email: 'maria.gomez@exam
ple.com' },
    { id: 3, name: 'Carlos López', email: 'carlos.lopez@ex
ample.com' }
  ];

  private nextId: number = 4;

  constructor() { }

  getUsers(): User[] {
    return this.users;
  }

  getUserById(id: number): User | undefined {
    return this.users.find(user => user.id === id);
  }

  addUser(user: Omit<User, 'id'>): void {
    const newUser: User = { id: this.nextId++, ...user };
    this.users.push(newUser);
  }

  updateUser(updatedUser: User): void {
    const index = this.users.findIndex(user => user.id ===
updatedUser.id);
    if (index !== -1) {
      this.users[index] = updatedUser;
    }
  }
```

```
  deleteUser(id: number): void {
    this.users = this.users.filter(user => user.id !== i
d);
  }
}
```

---

9. Estilización del Proyecto

Para mantener una apariencia consistente y profesional en tu aplicación, aplicaremos estilos globales y específicos a los componentes CRUD.

a. Variables CSS Globales

1. **Crear un Archivo de Variables:**

   Crea un archivo `variables.css` en la carpeta `src/styles/`.

```
/* src/styles/variables.css */
:root {
  --bright-blue: oklch(51.01% 0.274 263.83);
  --electric-violet: oklch(53.18% 0.28 296.97);
  --french-violet: oklch(47.66% 0.246 305.88);
  --vivid-pink: oklch(69.02% 0.277 332.77);
  --hot-red: oklch(61.42% 0.238 15.34);
  --orange-red: oklch(63.32% 0.24 31.68);

  --gray-900: oklch(19.37% 0.006 300.98);
  --gray-700: oklch(36.98% 0.014 302.71);
  --gray-400: oklch(70.9% 0.015 304.04);

  --red-to-pink-to-purple-vertical-gradient: linear-gradie
nt(
    180deg,
    var(--orange-red) 0%,
    var(--vivid-pink) 50%,
```

```
    var(--electric-violet) 100%
  );

  --red-to-pink-to-purple-horizontal-gradient: linear-grad
ient(
    90deg,
    var(--orange-red) 0%,
    var(--vivid-pink) 50%,
    var(--electric-violet) 100%
  );

  --pill-accent: var(--bright-blue);

  font-family: "Inter", -apple-system, BlinkMacSystemFont,
"Segoe UI", Roboto,
    Helvetica, Arial, sans-serif, "Apple Color Emoji", "Se
goe UI Emoji",
    "Segoe UI Symbol";
  box-sizing: border-box;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}
```

2. **Importar las Variables en `styles.css`:**

```
/* src/styles/styles.css */
@import './variables.css';

/* Otros estilos globales */
body {
  margin: 0;
  padding: 0;
  background-color: #f5f5f5;
```

```
}

.main {
  width: 100%;
  min-height: 100vh;
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 1rem;
  box-sizing: border-box;
  position: relative;
}

.content {
  display: flex;
  justify-content: space-around;
  width: 100%;
  max-width: 1200px;
  margin-bottom: 3rem;
}

.divider {
  width: 1px;
  background: var(--red-to-pink-to-purple-vertical-gradien
t);
  margin-inline: 0.5rem;
}

@media screen and (max-width: 650px) {
  .content {
    flex-direction: column;
    width: max-content;
  }

  .divider {
    height: 1px;
```

```
    width: 100%;
    background: var(--red-to-pink-to-purple-horizontal-gra
dient);
    margin-block: 1.5rem;
  }
}
```

b. Estilos Compartidos

1. **Crear un Archivo de Estilos Compartidos:**

   Crea un archivo `shared-styles.css` en la carpeta `src/styles/`.

```
/* src/styles/shared-styles.css */

/* Contenedor para formularios (crear y editar) */
.form-container {
  background: var(--gray-400);
  padding: 2rem;
  border-radius: 1rem;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
  max-width: 800px;
  margin: 2rem auto;
}

/* Formularios */
.form {
  display: flex;
  flex-direction: column;
}

/* Grupos de formulario */
.form-group {
  margin-bottom: 1.5rem;
```

```
}

/* Campos de entrada */
.input-field {
  width: 100%;
  padding: 0.5rem;
  border: 1px solid var(--gray-700);
  border-radius: 0.5rem;
  font-family: inherit;
  font-size: 1rem;
}

/* Grupo de botones */
.button-group {
  display: flex;
  gap: 1rem;
}

/* Botones con estilo de 'pill' */
.pill {
  background: var(--pill-accent);
  color: white;
  padding: 0.5rem 1rem;
  border: none;
  border-radius: 2.75rem;
  cursor: pointer;
  transition: background 0.3s ease;
  text-decoration: none;
  font-size: 0.875rem;
}

.pill:hover {
  background: color-mix(in srgb, var(--pill-accent) 15%, t
ransparent);
}
```

```css
/* Tabla de usuarios */
.user-table {
  width: 100%;
  border-collapse: collapse;
  margin-top: 1rem;
}

.user-table th,
.user-table td {
  padding: 0.75rem;
  text-align: left;
  border-bottom: 1px solid var(--gray-700);
}
```

2. **Importar los Estilos Compartidos en `styles.css`:**

```css
/* src/styles/styles.css */
@import './variables.css';
@import './shared-styles.css';

/* Otros estilos globales */
body {
  margin: 0;
  padding: 0;
  background-color: #f5f5f5;
}

.main {
  width: 100%;
  min-height: 100vh;
  display: flex;
  justify-content: center;
  align-items: center;
```

```
  padding: 1rem;
  box-sizing: border-box;
  position: relative;
}

.content {
  display: flex;
  justify-content: space-around;
  width: 100%;
  max-width: 1200px;
  margin-bottom: 3rem;
}

.divider {
  width: 1px;
  background: var(--red-to-pink-to-purple-vertical-gradien
t);
  margin-inline: 0.5rem;
}

@media screen and (max-width: 650px) {
  .content {
    flex-direction: column;
    width: max-content;
  }

  .divider {
    height: 1px;
    width: 100%;
    background: var(--red-to-pink-to-purple-horizontal-gra
dient);
    margin-block: 1.5rem;
  }
}
```

c. Aplicación de Estilos a los Componentes CRUD

Ya hemos actualizado los archivos HTML de los componentes CRUD (`user-list`, `user-create`, `user-edit`) para utilizar las clases definidas en `shared-styles.css`. Ahora, asegurémonos de que estos estilos se apliquen correctamente.

1. **Actualizar Archivos de Estilo de los Componentes (Opcional):**

   Puedes mantener los estilos específicos de cada componente en sus propios archivos CSS si necesitas personalizaciones adicionales. Sin embargo, los estilos básicos ya están cubiertos en `shared-styles.css`.

   Ejemplo: `user-list.component.css`

   ```
   /* src/app/components/user-list/user-list.component.css */

   /* Si necesitas estilos adicionales específicos para la li
   sta de usuarios, añádelos aquí */
   ```

   Repite lo mismo para `user-create.component.css` y `user-edit.component.css` si es necesario.

2. **Estilizar el Componente Principal (`app.component.html`):**

   Para integrar los componentes CRUD dentro de un diseño principal que incluye el logo y enlaces sociales, actualizaremos `app.component.html`.

   ```
   <!-- src/app/app.component.html -->
   <main class="main">
     <div class="content">
       <div class="left-side">
         <!-- Logo -->
         <svg
           xmlns="http://www.w3.org/2000/svg"
           viewBox="0 0 982 239"
           fill="none"
   ```

```
        class="angular-logo"
      >
        <!-- Contenido SVG aquí -->
      </svg>
      <h1>Hello, {{ title }}</h1>
      <p>Congratulations! Your app is running. 🎉</p>
    </div>
    <div class="divider" role="separator" aria-label="Divi
der"></div>
    <div class="right-side">
      <router-outlet></router-outlet>
      <!-- Enlaces sociales -->
      <div class="social-links">
        <a
          href="https://github.com/angular/angular"
          aria-label="Github"
          target="_blank"
          rel="noopener"
          class="pill"
        >
          <!-- SVG de Github -->
          Github
        </a>
        <a
          href="https://twitter.com/angular"
          aria-label="Twitter"
          target="_blank"
          rel="noopener"
          class="pill"
        >
          <!-- SVG de Twitter -->
          Twitter
        </a>
        <a
          href="https://www.youtube.com/channel/UCbn1OgGei
-DV7aSRo_HaAiw"
```

```
            aria-label="Youtube"
            target="_blank"
            rel="noopener"
            class="pill"
          >
            <!-- SVG de Youtube -->
            Youtube
          </a>
        </div>
      </div>
    </div>
  </main>
```

Nota: Reemplaza los comentarios `<!-- SVG de ... -->` con los SVGs correspondientes que tienes en tu diseño original.

3. **Añadir Estilos para el Logo y Enlaces Sociales:**

   Puedes agregar estilos adicionales en `shared-styles.css` para el logo y los enlaces sociales.

```
/* src/styles/shared-styles.css */

/* Logo de Angular */
.angular-logo {
  max-width: 150px;
  margin-bottom: 1rem;
}

/* Enlaces Sociales */
.social-links {
  display: flex;
  align-items: center;
  gap: 0.5rem;
  margin-top: 1.5rem;
```

```
}

.social-links a {
  display: flex;
  align-items: center;
  justify-content: center;
  width: 40px;
  height: 40px;
  background: var(--gray-700);
  border-radius: 50%;
  transition: background 0.3s ease;
}

.social-links a:hover {
  background: var(--gray-900);
}

.social-links svg {
  fill: white;
  width: 20px;
  height: 20px;
}
```

10. Integración de Enlaces Sociales y Logo

Para completar el diseño y hacerlo más atractivo, integraremos el logo de Angular y enlaces sociales en el componente principal.

1. **Agregar el SVG del Logo de Angular:**

   Copia el SVG de tu logo personalizado dentro del contenedor `left-side` en `app.component.html`.

```
<!-- src/app/app.component.html -->
```

```
<div class="left-side">
  <svg
    xmlns="http://www.w3.org/2000/svg"
    viewBox="0 0 982 239"
    fill="none"
    class="angular-logo"
  >
    <!-- Contenido SVG aquí -->
  </svg>
  <h1>Hello, {{ title }}</h1>
  <p>Congratulations! Your app is running. 🎉</p>
</div>
```

2. **Agregar Enlaces Sociales con SVGs:**

   Añade los SVGs correspondientes dentro de los enlaces sociales en `app.component.html`. Asegúrate de que los SVGs estén correctamente codificados.

```
<!-- src/app/app.component.html -->
<div class="social-links">
  <a
    href="https://github.com/angular/angular"
    aria-label="Github"
    target="_blank"
    rel="noopener"
    class="pill"
  >
    <svg
      width="25"
      height="24"
      viewBox="0 0 25 24"
      fill="none"
      xmlns="http://www.w3.org/2000/svg"
```

```
      alt="Github"
    >
      <!-- SVG Path aquí -->
    </svg>
  </a>
  <a
    href="https://twitter.com/angular"
    aria-label="Twitter"
    target="_blank"
    rel="noopener"
    class="pill"
  >
    <svg
      width="24"
      height="24"
      viewBox="0 0 24 24"
      fill="none"
      xmlns="http://www.w3.org/2000/svg"
      alt="Twitter"
    >
      <!-- SVG Path aquí -->
    </svg>
  </a>
  <a
    href="https://www.youtube.com/channel/UCbn1OgGei-DV7aS
Ro_HaAiw"
    aria-label="Youtube"
    target="_blank"
    rel="noopener"
    class="pill"
  >
    <svg
      width="29"
      height="20"
      viewBox="0 0 29 20"
      fill="none"
```

```
        xmlns="http://www.w3.org/2000/svg"
        alt="Youtube"
      >
        <!-- SVG Path aquí -->
      </svg>
    </a>
  </div>
```

Nota: Asegúrate de que los `path` de los SVGs estén correctamente definidos para cada icono.

---

11. Ejecución y Verificación del Proyecto

Después de haber configurado todos los componentes y estilos, es hora de verificar que todo funcione correctamente.

1. **Iniciar el Servidor de Desarrollo:**

   Si no está ya en ejecución, inicia el servidor:

   ```
   ng serve
   ```

2. **Abrir la Aplicación en el Navegador:**

   Navega a `http://localhost:4200/` en tu navegador.

3. **Verificar las Vistas CRUD:**

   - **Lista de Usuarios:** Deberías ver una tabla con usuarios predefinidos y botones para editar o eliminar.
   - **Crear Usuario:** Al hacer clic en "Agregar Usuario", deberías navegar a la vista de creación donde puedes añadir un nuevo usuario.
   - **Editar Usuario:** Al hacer clic en "Editar" en la lista, deberías navegar a la vista de edición con los detalles del usuario prellenados.

- **Eliminar Usuario:** Al hacer clic en "Eliminar", el usuario debería ser eliminado de la lista.

4. **Verificar Estilos y Responsividad:**

   Asegúrate de que los estilos se apliquen correctamente y que la aplicación sea responsiva. Prueba cambiando el tamaño de la ventana del navegador para ver cómo se adapta la interfaz.

---

12. Conclusiones y Recursos Adicionales

Has configurado exitosamente un proyecto Angular con Angular CLI, creado componentes CRUD, implementado un servicio para manejar datos y aplicado estilos consistentes a través de variables y estilos compartidos. Esta configuración proporciona una base sólida para desarrollar aplicaciones más complejas.

Recursos Adicionales:

- Documentación Oficial de Angular
- Angular CLI
- Guía de Estilos de Angular
- Angular Routing
- Componentes Reutilizables en Angular
- Buenas Prácticas en Angular

Consejos para el Workshop:

- **Demostraciones en Vivo:** Considera mostrar en tiempo real cómo crear componentes y aplicar estilos.
- **Ejercicios Prácticos:** Proporciona a los participantes ejercicios para crear sus propios componentes y servicios.
- **Resolución de Problemas:** Dedica tiempo para resolver preguntas y problemas comunes que puedan surgir durante la configuración.
- **Material de Apoyo:** Proporciona un repositorio de GitHub con el código fuente para que los participantes puedan seguirlo.

Apéndice: Resumen de Comandos Útiles

- **Instalar Angular CLI:**

```
npm install -g @angular/cli
```

- **Crear un Nuevo Proyecto:**

```
ng new nombre-proyecto
```

- **Generar un Componente:**

```
ng generate component ruta/nombre-componente
# o usando el alias
ng g c ruta/nombre-componente
```

- **Generar un Servicio:**

```
ng generate service ruta/nombre-servicio
# o usando el alias
ng g s ruta/nombre-servicio
```

- **Iniciar el Servidor de Desarrollo:**

```
ng serve
```

- **Compilar para Producción:**

```
ng build --prod
```