TEMA 01 – Angular

Parte 01 – Ejercicios

1. Realiza los siguientes apartados de introducción a la instalación de Angular y la creación de un primer proyecto.

1. Investigación sobre Standalone vs. NgModules:

* Busca información sobre la definición de Standalone Components vs. NgModules.
  + NgModules: Agrupan componentes, directivas, pipes y servicios en unidades uniformes. Cada aplicación Angular (anterior a la versión 14) comienza con un AppModule y puede tener módulos de características, módulos compartidos, etc…
  + Standalone: Introducidos en la versión 14 los componentes permiten auto declarar sus propìas dependencias directamente, sin necesidad de pertenecer a un NgModules (una agrupación de todo) pueden importarlo todo directamente con su propiedad “imports”.
* Investiga sobre la motivación de Angular para incorporarlos.

1- Reducir la complejidad

2- Mejorar la experiencia de inicio

3- Facilitar el lazy loading in módulos: Se puede cargar un componente de forma diferida directamente

4- Alinear con tendencias modernas: Como React o Vue

5- Reducir el boilerplate: menos archivos, menos configuración.

* Ventajas e inconvenientes frente a NgModules.

Ventajas

* + Menos archivos: no se necesitan .module.ts para cada característica
  + Configuracion directa en cada componente
  + Mejor sopote\_ componentes autocontenidos son mas fáciles e reutilizar.
  + Lazy loading simplificado: Se pueden cargar un componente sin envolverlo en un modulo
  + Mejor tree.shaking: menos código innecesario incluido en el bundle

Desventajas

* + Compatitiblidad con librerías antiguas: Muchas librerías aun utilizan módulos
  + Menor encapsulamiento lógico: En proyectos grandes los módulos ayudan a unificar u organizar funcionalidades
  + Herramientas y documentación: parte de la comunidad y tutoriales aun están en NgModules.
* ¿En qué casos sigue teniendo sentido usar NgModules ?
  + Migracion progresiva: Si tienes una aplicación grande basada en módulos, no es necesario reescribirla
  + Uso de librerías: Como ya he comentado con antelación la mayoría de las librerías aun están en módulos.
  + Organización de grande equipos: Los módulos, al llevarlo todo incluido, son como “contratos” de funcionabilidad bien definidos.
* Incluye una tabla comparativa con:
  + Archivos característicos de cada enfoque (app.config.ts/app.routes.ts/main.ts con bootstrapApplication vs. app.module.ts/app-routing.module.ts con bootstrapModule).
  + Diferencias en imports (en componentes standalone vs. en módulos).
  + Impacto en el *bootstrap (arranque de la app en Angular mediante bootstrapApplication o bootstrapModule -NO confundir con el framework CSS Bootstrap-),* aprendizaje y mantenimiento.

|  |  |  |
| --- | --- | --- |
| Criterio | Standalone | NgModules |
| Archivos Caracteristicos | app.config.ts  app.routes.ts  app.component.ts | app.module.ts  app-routing.module.ts |
| Definiciones del componente raiz | @Component({ standalone: true, imports: […]}) | Declarado en declarations de AppModule |
| Gestión de rutas | En app.routes.ts usando Routes y funciones como import() para lazy loading | En AppRoutingModule o módulos de rutas, con loadChildren: () => import(…).then(m => m.FeatureModule) |
| Imports de dependencias | Directamente en el componente: imports: [CommonModule, MiComponente, MiPipe] | En el módulo: imports: [CommonModule, FetarureModule], declarations: [AppComponent] |
| Bootstrap de la app | bootstrapApplication(AppComponent, appConfig) en main.ts | platformBrowserDynamic().bootstrapModule(AppModule) |
| Lazy loading | loadComponent: () => import(‘./ruta.component’).then(c => c.RutaComponent) | loadChildren: () => import(‘./feature.module’).then(m => m.FeatureModule) |
| Curva de aprendizaje | Más baja: no se necesita entender módulos para empezar | Más alta: requiere comprender módulos, declaraciones, proveedores, etc |
| Mantenimiento | Más simple en proyectos pequeños/medianos; menos archivos | Puede volverse complejo con muchos módulos anidados, pero útil para encapsulamiento en grandes apps |
| Compativilidad con librerías | Se pueden importar módulos dentro de imports | Nativo: todas las librerías Angular clásicas están diseñadas para módulos |

b. Puesta a punto del entorno:

* Verifica que tienes Node y su gestor de paquetes npm.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

* Instala Angular CLI

Texto

El contenido generado por IA puede ser incorrecto.

* Añade estas extensiones de Google Chrome
  + [Angular DevTools](https://chromewebstore.google.com/detail/angular-devtools/ienfalfjdbdpebioblfackkekamfmbnh)
  + [JSON Viewer Pro](https://chromewebstore.google.com/detail/json-viewer-pro/eifflpmocdbdmepbjaopkkhbfmdgijcc)
  + [Postman Interceptor](https://chromewebstore.google.com/detail/postman-interceptor/aicmkgpgakddgnaphhhpliifpcfhicfo)

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

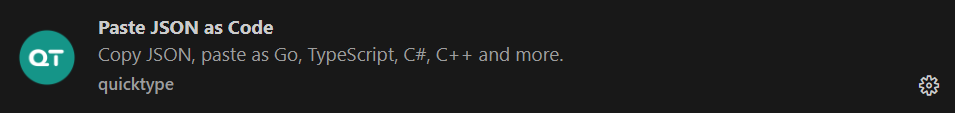
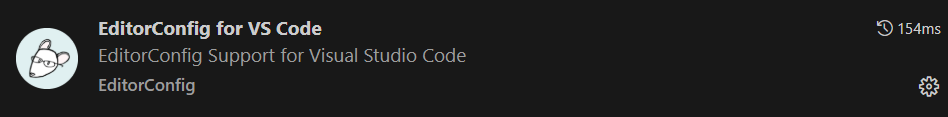
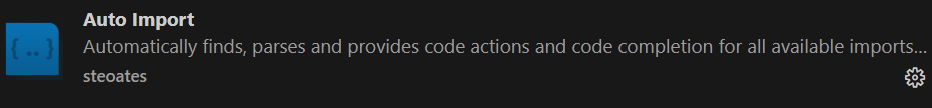
Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

* Agrega también estas Extensiones de VS Code

Forma

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

1. Crea un proyecto standalone

* Crea el proyecto con nombre fw\_proyecto1\_standalone
  + ng new fw\_proyecto1\_standalone
* Levanta el servidor con dicho proyecto.
  + ng serve
* Analiza la estructura creada (breve descripción de para qué sirve cada fichero clave):
  + main.ts → arranque con bootstrapApplication(...).
    - Para arrancar la app desde el componente raíz
  + app.config.ts → configuración/proveedores globales.
    - Define la configuración global de la aplicacion
  + app.routes.ts → definición de rutas.
    - Define las rutas principales de la app
  + app.component.ts con standalone: true e imports: [...].
    - el app.components.ts se fusiona en app.ts
      * earewComponente raíz de la app
* Realiza una pequeña modificación en la web de inicio y verifica en el navegador que carga la app.



Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Crea otro proyecto con NgModules (no standalone)

* Crea el proyecto con nombre fw\_proyecto2\_no\_standalone
  + ng new fw\_proyecto2\_no\_standalone --no-standalone
* Levanta el servidor con dicho proyecto.
  + ng serve
* Analiza la estructura (breve):
  + Presencia de app.module.ts y (si procede) [app-routing.module.ts](http://app-routing.module.ts).
    - Declara el componente raíz, importa módulos necesarios y define como arranca la app
    - Routing.module.ts: Enrutamiento de la app
  + main.ts con platformBrowserDynamic().bootstrapModule(AppModule).
    - Usa bootstrapModule para arrancar la aplicación desde el modulo raiz.
* Realiza una pequeña modificación en la web de inicio y verifica en el navegador que carga la app.



Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Comparativa técnica mediante la creación de una tabla con las diferencias clave entre:

* Bootstrap: bootstrapApplication vs bootstrapModule.
* Archivos generados y organización.
* Dónde se declaran imports (componente vs. módulo) y dónde se declara un componente (solo en módulos).
* Curva de aprendizaje y mantenimiento.
* Consideraciones de migración (por qué podría interesar migrar un proyecto con módulos a standalone y en qué casos no).

|  |  |  |
| --- | --- | --- |
| Punto | Standalone | NgModules |
| Boostrap | Usa bootstrapApplication(AppComponent, config) en main.ts . Arranca directamente desde un componente raiz | Usa platformBrowserDynamic()-bootstrapModule(AppModule) en main.ts |
| Archivos y organización | Main.ts  App.config.ts  App.routes.ts  App.ts | Main.ts  App.module.ts  App-routing.module.ts  App.component.ts |
| Imports | No se declara en ningún lado, si es standalone:true. Se usa directamente donde se importa | Debe declararse en declarations de un NgModule |
| Curva de aprendizaje | Más baja: No se necesita entender módulos, declaraciones, expot… etc | Más alta: Requiere entender la diferencia entre imports, declarations, providers y lazy loading con módulos. |
| Miracion | Si:  Reducir complejidad  Eliminar archivos de módulos innecesarios | No:  Proyectos grandes ya estables  Uso intensivo de librerías Equipos que ya sabe NgModules |

2. Realiza los siguientes ejercicios para entender los conceptos fundamentales de Angular.

1. Análisis de la “chuleta” del curso de Fernando Herrera: [**Chuleta del curso de Angular de Fernando Herrera**](https://drive.google.com/file/d/1B3k7sfw_y_KpeYfVyLsHxbvNeTJbpIZ4/view?usp=sharing)

* Enumera los distintos bloques o apartados principales definidos por el autor en dicha "chuleta".

1. Visualización y mapa mental del curso de OpenWebinars: Fundamentos de Angular

* Accede al curso: <https://academia.openwebinars.net/aprende/angular>
* Determina si el curso está enfocado a un proyecto standalone o basado en NgModules (observa cómo se importan y declaran los componentes en los ejemplos).
* Texto

  El contenido generado por IA puede ser incorrecto.
  + NgModules
    - No tiene “standalone: true”
    - No tiene imports
* A medida que avances, elabora un mapa mental (digital o a mano) que recoja los conceptos clave explicados en el curso. Puedes usar herramientas como MindMeister, Miro, Draw.io o un papel.
  + El mapa debe incluir, como mínimo, los siguientes grupos de conceptos:
  + Angular
    - Enfoque reactivo
    - Aplicaciones SPA (Single Page Application)
    - Uso de TypeScript
  + Componentes
    - Estructura y creación
    - Ciclo de vida (hooks)
    - Decoradores (@Component, @Input, @Output)
  + Plantillas (Templates)
    - Interpolación
    - Data Binding (one-way / two-way)
    - Eventos y propiedades
  + Pipes
    - Uso y creación de pipes personalizados
  + Directivas
    - Directivas estructurales (\*ngIf, \*ngFor)
    - Directivas de atributo (ngClass, ngStyle)
  + Módulos / Standalone Components
    - Organización del código
    - Importaciones y declaraciones
    - Bootstrap (arranque, no confundir con CSS Bootstrap)
  + Rutas
    - Configuración de rutas
    - Parámetros
    - Navegación y enlaces (routerLink)
  + Guards
    - Control de acceso y protección de rutas
  + Servicios
    - Inyección de dependencias
    - Comunicación entre componentes
  + Providers
    - Registro de servicios a distintos niveles (raíz o módulos/componentes)