¿Qué es angular?

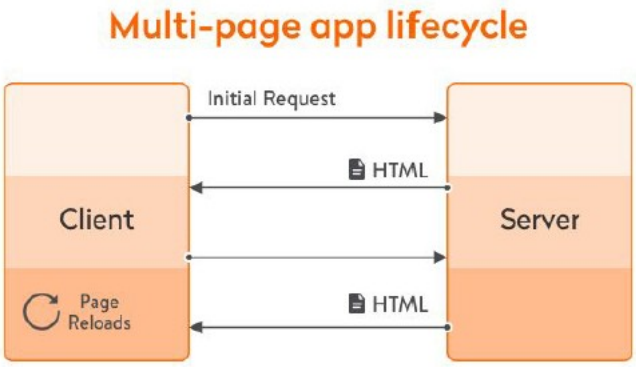
Angular es una plataforma de código abierto, que permite desarrollar aplicaciones web en la sección cliente utilizando **HTML** y **TypeScript,** descargando al servidor de buena parte del trabajo, con lo que se consigue una mayor velocidad en la ejecución y, por tanto, un mayor rendimiento.

Es un framework que permite la creación de aplicaciones web de una sola página (**SPA: *single-page application***) realizando la carga de datos de forma **asíncrona**.

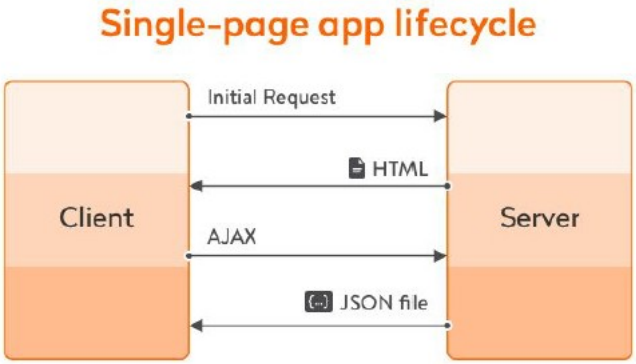
Ejemplos de empresas que utilizan Angular: Google, Microsoft, IBM, PayPal, Samsung, Deutch Bank...

SPA vs. MPA (Multiple Page Application)

- Aplicaciones cliente completas programadas con HTML, CSS y **Javascript**
- **Vistas** vs. Páginas
- Comunicación con el servidor a través de **API's**
- Frameworks para crear SPA: **Angular**, React, Vue, EmberJS, Polymer, Svelte, etc.



Cada petición a servidor, devuelve un html completo que el cliente interpretará.



Hace una petición inicial en la que el servidor devuelve un html. A partir de ahí, nunca se vuelve a cargar un html en el navegador, sino que se irá modificando por zonas.

[SPA vs MPA]

Características

Por lo tanto, es un framework para desarrollo de aplicaciones SPA utilizando HTML y Typescript.

- Ha sido desarrollado por Google.
- Angular está orientado a objetos, trabaja con clases y favorece el uso del patrón MVC (Modelo-Vista-Controlador).
- Parecía ser la continuación de AngularJS, pero, en realidad, más que una nueva versión es realmente un framework o plataforma diferente.
- Escrito y basado en Typescript. Es muy similar a Javascript pero incluye el tipado en los objetos. TypeScript ayuda a Angular a identificar eficazmente los errores en una fase más temprana del ciclo de desarrollo que muchos otros frameworks.
- Basado en módulos, componentes y servicios (clases Typescript con decoradores).
- Es modular, facilita la reutilización
- Fácil mantenimiento
- Multiplataforma
- Extiende el código HTML con etiquetas propias
- Es de código abierto.
- El desarrollo de aplicaciones es rápido, y la navegación también.
- Creciente demanda de trabajo

Release schedule

Version	Date
v19.1	Week of 2025-01-13
v19.2	Week of 2025-02-24
v20.0	Week of 2025-05-19

Las versiones de Angular siguen el formato: MAJOR.MINOR.PATCH:

- MAJOR (Versión Principal): Indica cambios importantes o incompatibles con versiones anteriores (breaking changes). Actualizar a una nueva versión principal puede requerir ajustes en tu código. Ejemplo: De Angular **15.x.x** a **16.x.x**.
- MINOR (Versión Menor): Indica nuevas características o mejoras que no rompen la compatibilidad con versiones anteriores, como adición de nuevas APIs, directivas, o funcionalidades. Puedes actualizar sin preocuparte por que tu aplicación actual deje de funcionar. Ejemplo: De Angular 16.0.x a 16.1.x.
- PATCH (Parche): Suelen llevar corrección de errores o problemas menores sin introducir nuevas funcionalidades ni romper la compatibilidad, soluciones de bugs … Es seguro aplicar actualizaciones de parche de inmediato. Ejemplo: De Angular 16.0.0 a 16.0.1.

Ejemplo: Si la versión de Angular es 16.1.4, significa lo siguiente:

- 16: Esta es la versión principal, lo que implica que podría haber cambios importantes respecto a Angular 15 o anteriores.
- 1: Indica una actualización menor que agrega nuevas funcionalidades pero sin romper la compatibilidad con Angular 16.0.x.
- 4: Indica un parche que corrige errores o problemas menores encontrados en la versión 16.1.3.

Angular utiliza una política de versiones conocida como **LTS (Long-Term Support)**:

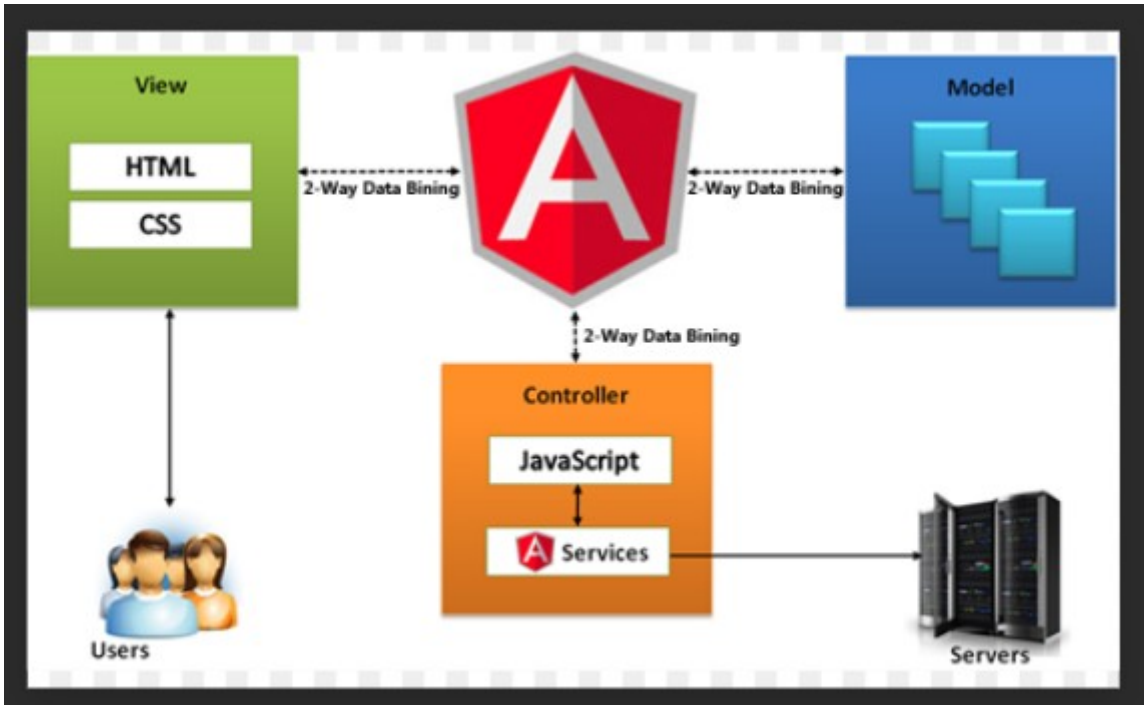
1. **Active Support:** Una nueva versión recibe actualizaciones activas durante los primeros 6 meses.
2. **LTS (Long-Term Support):** Después, recibe parches y correcciones críticas por otros 12 meses antes de quedar obsoleta.

Actively supported versions

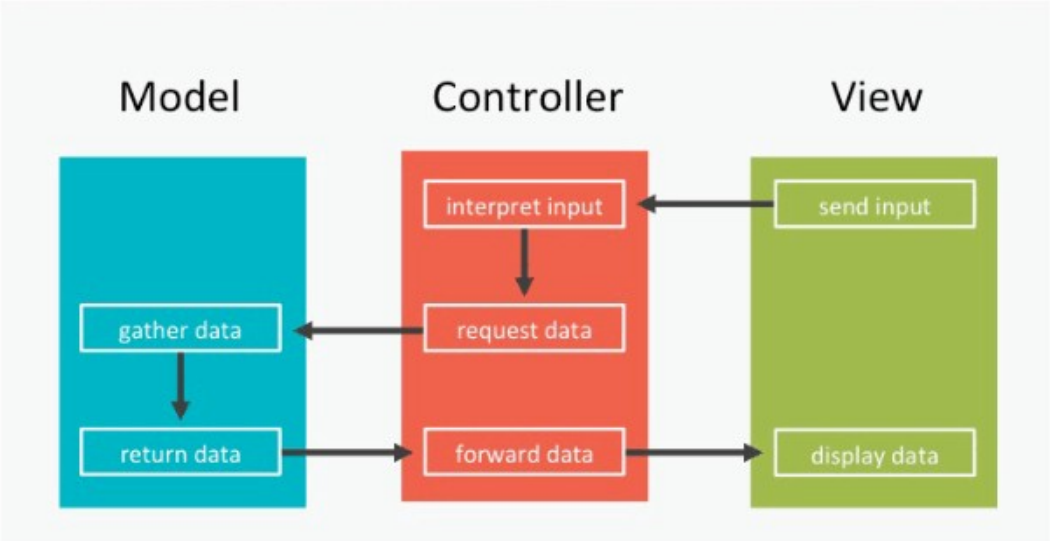
The following table provides the status for Angular versions under support.

Version	Status	Released	Active ends	LTS ends
^19.0.0	Active	2024-11-19	2025-05-19	2026-05-19
^18.0.0	LTS	2024-05-22	2024-11-19	2025-11-19
^17.0.0	LTS	2023-11-08	2024-05-08	2025-05-15

MODELO MVC (Modelo – Vista – Controlador)



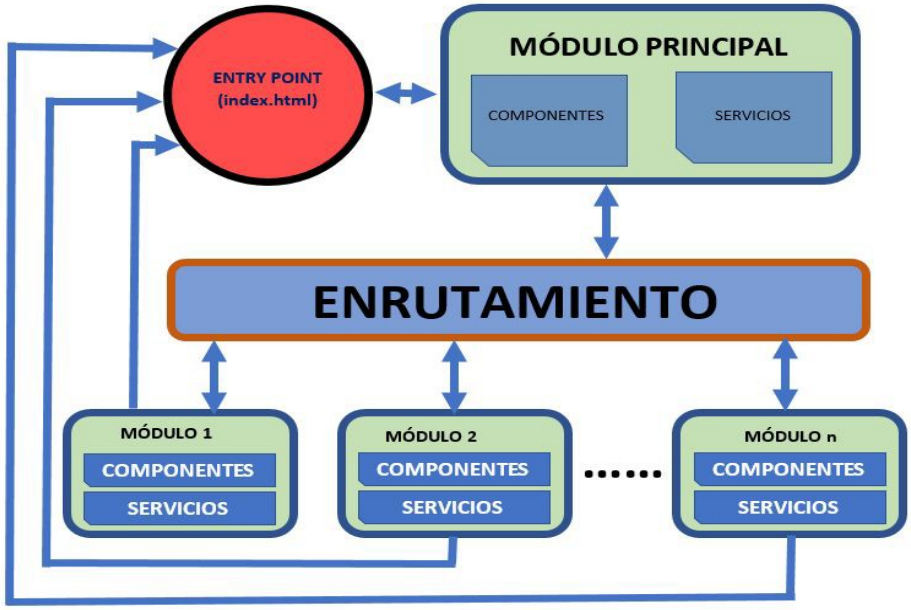
El usuario interactúa con la vista, quien hace llamadas al controlador. El controlador actúa sobre el modelo e interacciona con el servidor. Cualquier cambio del modelo actualiza la vista.



Vista: es el componente que se encarga de la presentación de la información al usuario. Es decir, es la interfaz gráfica que el usuario ve y con la que interactúa.

Modelo: es el componente que representa los datos y la lógica de negocio de la aplicación. Se encarga de interactuar con la base de datos y de manejar la información que se va a mostrar en la vista.

Controlador: es el componente que actúa como intermediario entre el modelo y la vista. Se encarga de recibir las peticiones del usuario y de coordinar la interacción entre el modelo y la vista para procesar la información.



* Permite el uso de TypeScript (lenguaje desarrollado por Microsoft) con las ventajas que supone poder disponer de un tipado estático y objetos basados en clases. Todo ello, gracias a la especificación ECMAScript 6, que es la base sobre la que se apoya TypeScript. Gracias a un compilador (transpilador) de TypeScript, el código escrito en este lenguaje se traducirá a JavaScript original.

Instalación: https://angular.io/guide/setup-local

- **Visual Studio Code:** Extensión recomendada: Angular Essentials y angular files. Editor de Microsoft

- **Nodejs:** https://nodejs.org/en. Descargamos la versión LTS

Es un entorno javaScript en tiempo de ejecución del lado del servidor.
Trae el gestor de paquete npm. Para saber la versión de Node y **npm** instalada, abrimos la consola con cmd:

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

- **Angular cli**: herramienta de línea de comandos que se utiliza para inicializar, desarrollar, estructurar y mantener aplicaciones de Angular: **ng version**

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

Puede pasar que la política de ejecución esté deshabilitada. Por defecto windows no permite la ejecución de scripts. Para solucionarlo en powerShell:

```
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned
Set-ExecutionPolicy Unrestricted
```

Una vez instalado, disponemos del **comando ng** para lanzar cualquier acción: EJ: ng --help

Creación de una aplicación angular

Vamos a crear un un proyecto:

```
ng new mi_proyecto
```

Dependiendo de la versión de angular:
Which stylesheet format would you like to use? CSS
Would you like to add Angular routing? → Decimos de momento que N
Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? → N

Lanzando este comando se creará una carpeta igual que el nombre del proyecto indicado y dentro de ella se generarán una serie de subcarpetas y archivos.

Se instalarán y se configurarán en el proyecto una gran cantidad de herramientas útiles para la etapa del desarrollo front-end.

Entramos en la carpeta creada:

```
cd mi_proyecto
```

Angular CLI lleva integrado un servidor web, lo que quiere decir que podemos visualizar y usar el proyecto sin necesidad de cualquier otro software.

Para arrancar el servidor

```
ng serve
```

Lanzará el servidor web y lo pondrá en marcha. Además, en el terminal verás como salida del comando la ruta donde el servidor está funcionando:

EJ: http://localhost:4200/

Si queremos abrirlo en otro puerto: ng serve **--port** 4201

ng serve **--port** 4201 o ng serve **-p** 4201

Para parar el servidor:

En la consola cmd: **Control C**

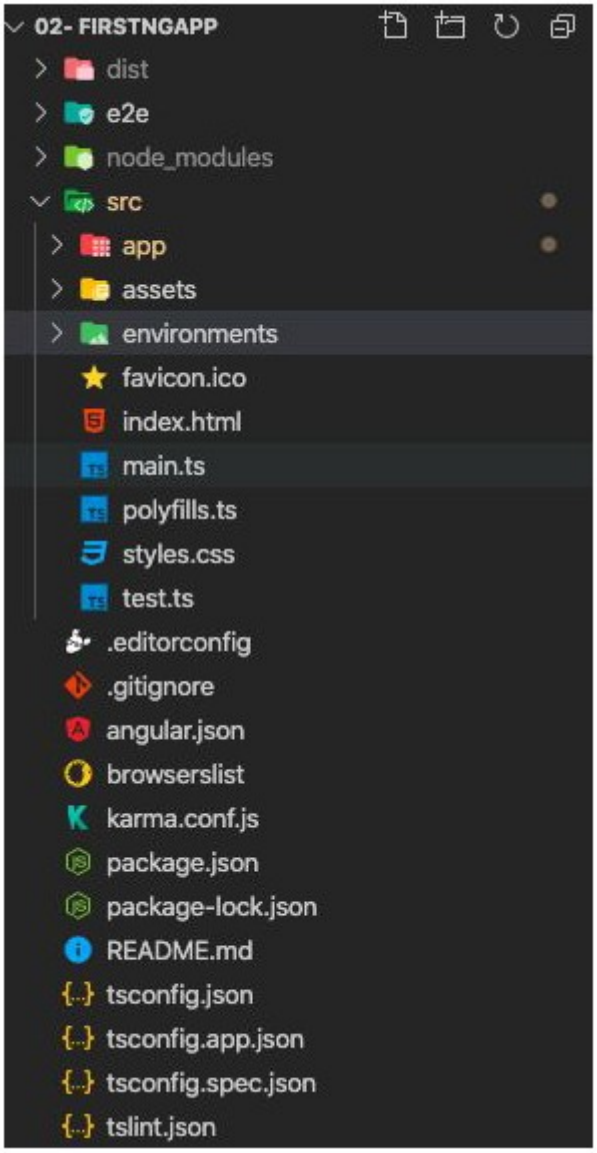
Estructura de una aplicación angular

Cuando creamos un nuevo espacio de trabajo Angular genera una estructura de carpetas y crea muchos archivos, parece algo desproporcionado para una aplicación tan sencilla y en la mayoría de los casos no modificaremos la mayoría de los archivos que forman el proyecto. Vamos a repasar por encima el propósito de algunos de estos ficheros y cómo se estructura el proyecto.

Excepto la carpeta src, el resto son archivos que se usan a la hora de desarrollar. Cuando se pase a producción un proyecto, sólo necesitamos la carpeta src.

- **dist**: La aplicación para publicar en el servidor web de producción. Se genera sólo cuando se compila para producción (ng build --prod)
- **e2e**: Ficheros para realizar pruebas end-to-end automáticas
- **node_modules**: Carpeta con las dependencias del proyecto, es decir, librerías y herramientas necesarias en el proyecto
- **src**: Fuentes de la aplicación
- **src/app**: Ficheros fuente principales de la aplicación (módulos, componentes, etc.)
- **src/assets**: Recursos estáticos que necesita la aplicación (imágenes, css, etc.)
- **src/environments**: Configuraciones y variables de entorno que se utilizarán tanto en desarrollo como en producción
- **src/favicon.ico**: Archivo icono del proyecto
- **src/index.html**: Página principal (y única) de la aplicación
- **src/main.ts**: Archivo Typescript de inicio de la aplicación. **SE DEFINE EL MÓDULO PRINCIPAL**
- **.editorconfig**: Configuración del editor VSCode

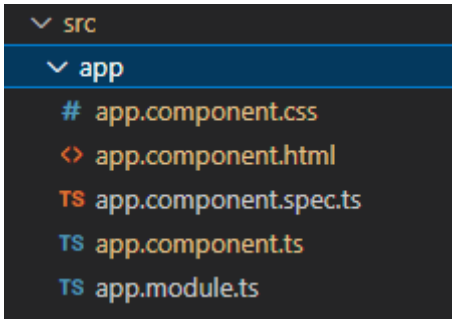
- .**gitignore**: Carpetas/ficheros que git debe ignorar
- **angular.json**: contiene la configuración del propio CLI
- **package.json**: Configuración de la aplicación y registra las dependencias de librerías y scripts necesarios para su despliegue y ejecución.
- **README.md**: Información/documentación sobre la aplicación.
- **tsconfig.json**: Contiene la configuración de TypeScript para transpilar a Javascript.
- **tslint.json**: Reglas del linter de Typescript. TsLint es una herramienta para analizar el código fuente de TypeScript: formatos, nomenclaturas, errores, etc...



Importante: La carpeta **node_modules** contienen todas las librerías del proyecto. Esta carpeta **NUNCA** se entrega a producción ni se sube a ningún repositorio. Si borramos esta carpeta y necesitamos volver a tener las librerías, lanzaremos el comando:

npm install

Vamos a ver el componente principal. Si entramos en la carpeta **src/app:**



- **app.component.css**: Estilos CSS para el componente
- **app.component.hml**: Vista del componente
- **app.component.ts**: Es una clase Typescript del componente que se decora con el decorador @Component. En el decorador se especifica:
  - **Selector**: indica la etiqueta html donde se incluirá el componente.
  - **TemplateUrl**: indica la vista del componente, la parte html.
  - **StyleUrl**: indica la hoja de estilos del componente.

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

You, 19 minutes ago | 1 author (You)
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'mi_proyecto';
}
```

- **app.component.spec.ts**: Test creado para probar el componente

Sintaxis en TypeScript

- Las variables se definen escribiendo el nombre de variable y el tipo.
  Ejemplos:

```
let nombre: string = "Juan"; // Variable que puede cambiar
const edad: number = 25; // Variable cuyo valor no cambiará
let esActivo: boolean = true;
let fecha: Date = new Date();
```

- Los métodos en TypeScript se escriben de manera similar a JavaScript, pero puedes especificar los tipos de los parámetros y el tipo de retorno. Los métodos se definen dentro de una clase o fuera de ella.
  Ejemplos:

```
class Persona {
    nombre: string;
    edad: number;

    constructor(nombre: string, edad: number) {
        this.nombre = nombre; this.edad = edad;
    }
    // Método que retorna un valor
    saludar(): string {
        return `Hola, mi nombre es ${this.nombre} y tengo ${this.edad} años.`;
    }
     // Método con parámetros y sin retorno
    cumplirAnios(): void {
        this.edad++;
    }
}
```

Tipos de datos en TypeScript:

- **string**: Cadenas de texto.
- **number**: Números (enteros o decimales).
- **boolean**: Valores true o false.
- **any**: Cualquier tipo de dato (aunque se recomienda evitar su uso por el tipo de seguridad que ofrece TypeScript).
- **void**: Representa un método que no devuelve ningún valor (usado en métodos).
- **array**: Puedes declarar un arreglo usando [] o Array<T>, por ejemplo: let lista: number[] = [1, 2, 3]; o let lista: Array<number> = [1, 2, 3];.

Realizando modificaciones

- Abre la plantilla “app.component.html”, borra el contenido y reemplazalo:

  <h1>Mi primera aplicación angular</h1>

  Ejecutamos **ng serve**

  Lanzamos http://localhost:4200

- Le damos estilos a h1, de forma inline, y ponemos el texto en rojo y el texto en negrita

- Duplicar otro mensaje con h1 pero ahora creamos mejor un estilo class=”azul”, con color de la letra azul. ¿Donde creamos el estilo?

- Añadimos en la vista del componente (html), un párrafo con un saludo con vuestro nombre.

- Añadimos al html una imagen, la que queramos, ¿dónde debemos guardar la imagen?

- Cambiar el icono favicon.ico por otro que queráis para vuestra aplicación.

- Vamos al componente “app.component.ts”. La clase AppComponent tiene una propiedad title. Vamos a cambiarle el nombre de title:

  title: string =” Clase de Angular”. ¿Cambia algo la página?

  Para mostrar el valor de una propiedad en la vista usamos {{ propiedad}}. Esto se conoce como **binding**.

  <h2> Título: {{ title}} </h2>

- Añadir una propiedad al componente **nombre** inicializada con vuestro nombre y mostrarla en la vista del componente con la etiqueta h3.

- Vamos a index.html. Aparece la etiqueta:<app-root> . Vamos a cambiarle el nombre: <app-hola>
  ¿Que pasaría?
  ¿Qué tendríamos que cambiar para que vuelva a mostrar el componente? Pista: ir al componente.

- Crear en el componente una variable numérica ‘edad’ con el valor 18 (tipo number) y un método getEdad que devuelva el campo edad. En la vista crear un párrafo que indique el mensaje “es mayor de edad” o “menor de edad” en función del valor de ‘edad’.