Tabla de contenido

Tabla de contenido

[1 INTRODUCCIÓN 4](#_Toc185343736)

[2 OBJETIVO 4](#_Toc932193559)

[3 ALCANCE 4](#_Toc19543524)

[4 RESPONSABLES 4](#_Toc235100031)

[5 CONSIDERACIONES GENERALES 5](#_Toc1575750523)

[6 PROCEDIMIENTO 5](#_Toc370528745)

[6.1 Descarga del código fuente 5](#_Toc413364244)

[6.1.1. Creación de su rama de trabajo 5](#_Toc655813867)

[6.2 Prueba del Arquetipo sin cambios 5](#_Toc1949396921)

[6.2.1. Carga la aplicación e iniciarla 5](#_Toc2013112307)

[7 Creación de Componentes según Atomic Design 5](#_Toc1065008684)

[7.1 Átomos 5](#_Toc341979598)

[7.2 Moléculas 5](#_Toc1858876736)

[7.3 Organismos 5](#_Toc538054410)

[7.4 Plantillas 5](#_Toc936933837)

[7.5 Paginas 5](#_Toc738296221)

[8 Validaciones para la Integración de Componentes 5](#_Toc1083730149)

[9 Implementación de la Arquitectura Hexagonal 5](#_Toc1447989874)

[9.1 Capa dominio 5](#_Toc311100735)

[9.2 Capa de aplicación 5](#_Toc1881038718)

[9.3 Capa de Infraestructura 5](#_Toc614079449)

[9.4 Configuración del modulo 5](#_Toc1883482339)

[9.5 Reglas Generales de Implementación 5](#_Toc1540524162)

[10 creación de un Nuevo Proyecto utilizando el Arquetipo 5](#_Toc1125067753)

[10.1 Paso 1: Inicializar el Proyecto 5](#_Toc399046874)

[10.2 Paso 2: Estructura del Proyecto 5](#_Toc587750421)

[10.3 Paso 3: Crear Componentes 5](#_Toc1631430389)

[10.4 Paso 4: Implementar la Lógica de Negocio 5](#_Toc1587272316)

[10.5 Paso 5: Crear Servicios 5](#_Toc49512967)

[10.6 Paso 6: Integrar la UI 5](#_Toc1270779495)

[10.7 Paso 7: Pruebas y Validaciones 5](#_Toc1226211563)

[11 conclusión 5](#_Toc872101486)

# INTRODUCCIÓN

En el contexto actual del desarrollo de software, la creación de aplicaciones web escalables y mantenibles es fundamental para el éxito de cualquier proyecto. Este documento presenta un arquetipo de consumo diseñado específicamente para aplicaciones Angular, implementando una arquitectura hexagonal y principios de atomic design.

El arquetipo propuesto busca estandarizar el proceso de desarrollo, promoviendo buenas prácticas y métodos que faciliten la colaboración entre los equipos de trabajo, la reutilización de componentes y la adaptabilidad a futuros cambios tecnológicos. A través de este documento, se proporcionarán guías detalladas y ejemplos prácticos para asegurar que los desarrolladores puedan implementar este arquetipo de manera efectiva.

# OBJETIVO

El objetivo de este documento es definir un patrón de desarrollo que permita a los equipos construir aplicaciones Angular de manera eficiente y estructurada. A través de la implementación de una arquitectura hexagonal y atomic design, se busca:

* **Mitigar la complejidad del desarrollo**: Al proporcionar un marco claro y organizado, se facilitará la comprensión y el mantenimiento del código.
* **Fomentar buenas prácticas**: Se establecerán pautas que los desarrolladores deberán seguir para asegurar la calidad del software.
* **Facilitar la colaboración**: Al estandarizar procesos y estructuras, se mejorará la comunicación y el trabajo en equipo entre los diferentes miembros del proyecto.

Este documento servirá como guía para la adopción del arquetipo, asegurando que todos los involucrados en el desarrollo de la aplicación sigan un enfoque coherente y eficiente.

# ALCANCE

Este documento se centra en la definición y guía de un arquetipo de consumo para el desarrollo de aplicaciones Angular utilizando una arquitectura hexagonal y principios de atomic design. A continuación, se detallan los aspectos específicos que se cubrirán:

**Estructura del Proyecto**: Se proporcionará una guía sobre cómo organizar un proyecto Angular de acuerdo con la arquitectura hexagonal, incluyendo la separación de responsabilidades y la gestión de módulos.

**Componentes y Diseño**: Se explicará cómo aplicar los principios de atomic design para crear componentes reutilizables y escalables, asegurando una experiencia de usuario coherente y efectiva.

**Configuración del Entorno**: Instrucciones para configurar el entorno de desarrollo necesario para trabajar con Angular, incluyendo configuraciones de herramientas y dependencias requeridas.

**Patrones de Desarrollo**: Se describirán patrones de desarrollo recomendados que facilitarán la implementación de la arquitectura y el diseño, así como buenas prácticas para el manejo de datos y servicios.

**Ejemplos Prácticos**: Se incluirán ejemplos de código y casos de uso que ilustran cómo aplicar el arquetipo en situaciones del mundo real, facilitando la comprensión y la implementación.

**3.1 Exclusiones**

Es importante señalar que este documento no cubre:

**Aspectos de Infraestructura**: No se abordarán consideraciones relacionadas con la infraestructura de servidores, despliegue en la nube o configuraciones de red.

**Integraciones con Backend**: Aunque se mencionarán patrones para consumir servicios, no se detallarán las implementaciones específicas del backend ni se proporcionarán ejemplos de servidores o APIs.

**Detallado de Herramientas Específicas**: No se incluirán guías exhaustivas sobre herramientas de desarrollo o bibliotecas específicas de Angular, más allá de las necesarias para el uso del arquetipo.

Este documento está destinado a desarrolladores que busquen adoptar un enfoque estructurado y eficiente en la creación de aplicaciones Angular y que deseen seguir un conjunto de buenas prácticas para mejorar la calidad y mantenibilidad de su código.

# RESPONSABLES

|  |  |
| --- | --- |
| RESPONSABLE | DESCRIPTION |
| Área de Desarrollo | Es responsable de proporcionar y mantener el software creado |
| Área de arquitectura | Esta área se encarga de actualizar y verificar que este documento debe ser  seguido por el área de desarrollo |

# CONSIDERACIONES GENERALES

Los requisitos mínimos para el manejo de los arquetipos son los siguientes:

* + Conocimientos en Angular (preferiblemente versión 17)
  + Conocimientos en typescript
  + Un Entorno de desarrollo (IDE)
  + Conocimiento de comando git.
  + Acceso a los repositorios de arquitectura.

Como requisitos deseables, se puede leer los siguientes enlaces:

* <https://atomicdesign.bradfrost.com/table-of-contents/>
* <https://www.gluo.mx/blog/atomic-design-que-es-para-que-sirve-y-ejemplos>
* <https://www.uifrommars.com/atomic-design-ventajas/>

# PROCEDIMIENTO

# Descarga del código fuente

Se deberá descargar el repositorio:  
  
<https://github.com/carcaicedo/arch-microfront-angular-hexatomic#>

# 6.1.1. Creación de su rama de trabajo

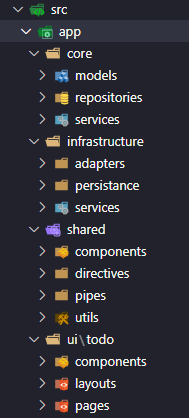
El código se encuentra una vez clonado referenciado a este repositorio de arquitectura, si se requiere crear un proyecto nuevo que deberá tener su propio repositorio, se recomienda una vez que se clona el repositorio de arquitectura, realizar la maniobra de una copia de la estructura en la carpeta destino, es decir de aquel proyecto que se va a iniciar con la estructura del arquetipo, para ello es importante indicar que en el copiado deberán no llevarse el .git de este proyecto.

Para efectos de este tutorial, se realizará sobre el mismo proyecto creando una rama de trabajo para la demostración.

# 6.2 Prueba del Arquetipo sin cambios

# 6.2.1. Carga la aplicación e iniciarla

Para efectos de este instructivos estamos usando como Visual Studio code, con este se abre el proyecto descargado en el punto anterior.



Para este proyecto, se utilizarán las siguientes tecnologías:

* Angular CLI: 19.2.14
* Node: 18.20.6

Al abrir el proyecto en Visual Studio Code, asegúrate de que el entorno esté configurado correctamente para trabajar con Angular. Visual Studio Code, al cargar el proyecto en la terminal integrada, ejecutar la instalación del proyecto se recomienda hacer uso de yarn para la instalación de dependencias, ejecutando:

**$ yarn install**

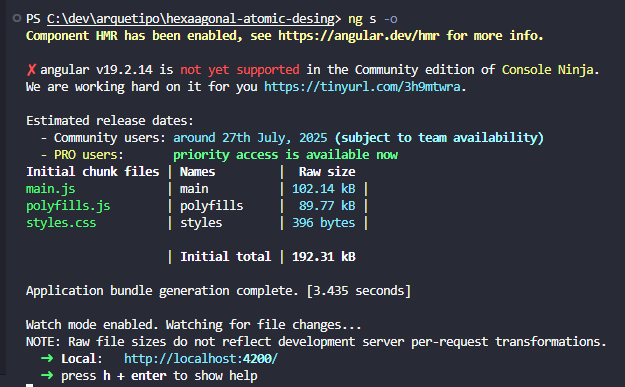
O si no se está usando yarn.

**$ npm install**

Al finalizar la instalación se debe ejecutar con el comando:

**$ ng s –o**

Esto levantara el servidor para verificar su correcto funcionamiento.



# 7 Creación de Componentes según Atomic Design

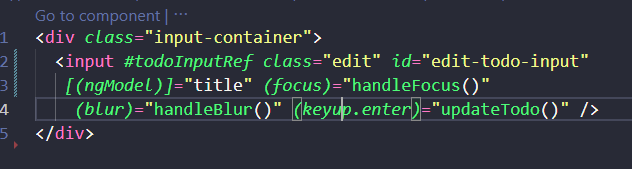
Para implementar el concepto de atomic design, se recomienda organizar los componentes en categorías: átomos, moléculas, organismos, plantillas y páginas.

# 7.1 Átomos

Los átomos son los componentes más simples y fundamentales de la interfaz de usuario. Representan elementos individuales como botones, campos de entrada, iconos o etiquetas. Cada átomo cumple una única responsabilidad y no debe tener lógica compleja ni un HTML extenso. Su propósito es ser reutilizables y servir como bloques de construcción para componentes más complejos.

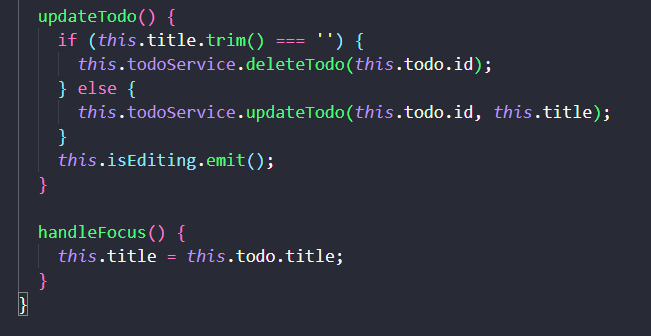
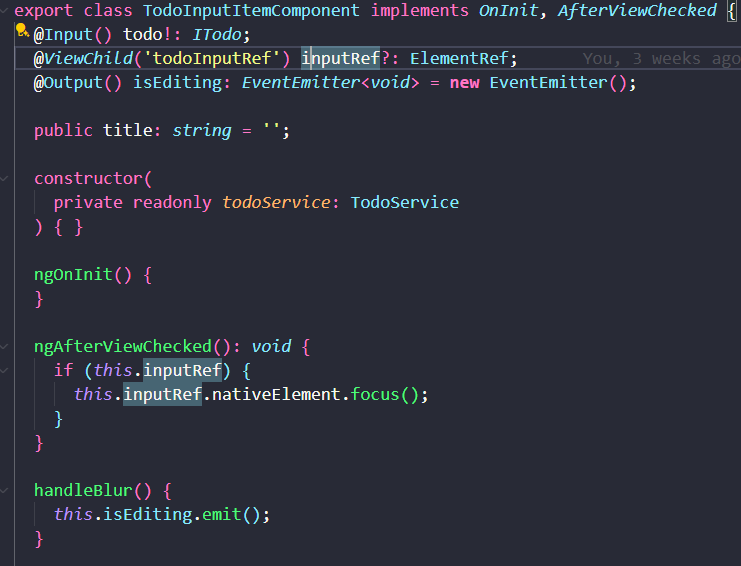
**Ejemplo de un átomo:** src\app\ui\todo\components\atoms\todo-input-item

Html:



El fragmento anterior es un claro ejemplo de lo que sería un átomo donde no es un HTML muy extenso y es un componente visual especializado en este caso es un ítem de los todos, siendo un contenedor y un input en este caso.

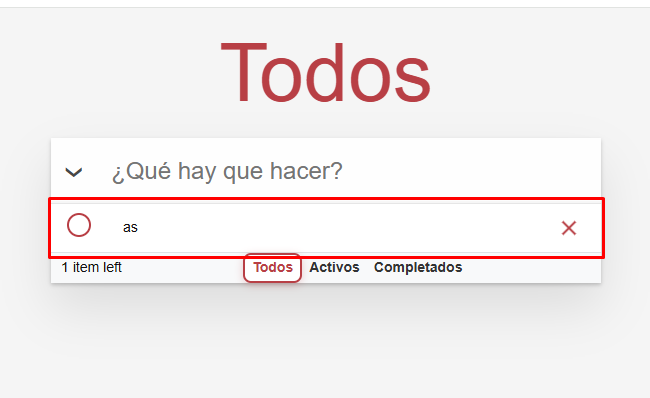
TS:



La logica del componente es referente solo al componente es sencilla y especifica en este caso este input tiene los eventos para activar el focus, para actualizar o eliminar los todos.

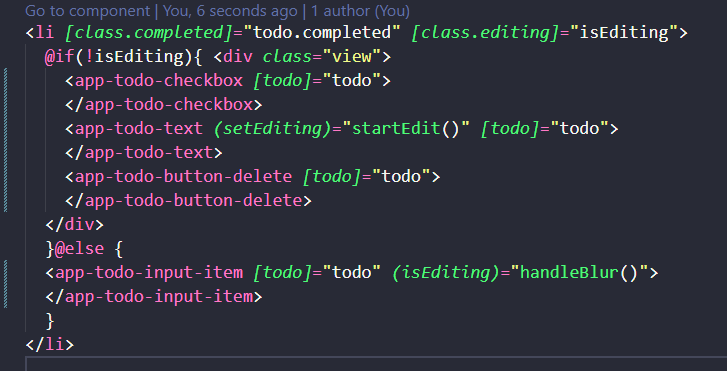
# 7.2 Moléculas

Las moléculas son combinaciones de átomos que forman componentes más complejos. Por ejemplo, un formulario que incluye varios campos de entrada. O como en este caso el item de cada todo.



Este elemento se encuentra en src\app\ui\todo\components\molecules\todo-item.

HTML:



El componente al ser una molécula podemos observar que su HTML es un poco más extenso, pero también se evidencia que está construido por varios átomos trabajando en conjunto para formar la molécula. Y no hay elementos construidos directamente acá asegurando un código reutilizable y limpio.

TS:

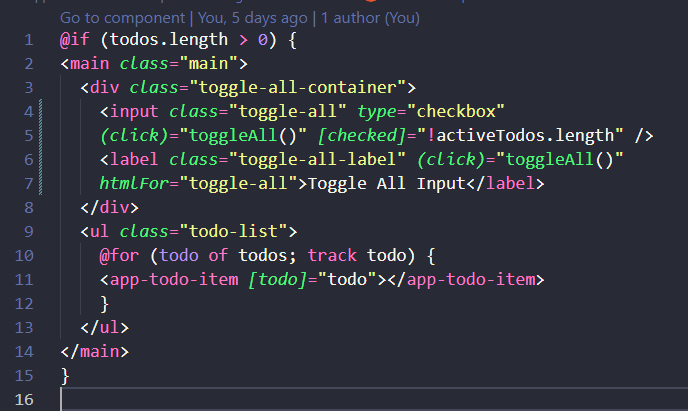


Las moléculas mantienen una única responsabilidad, pero pueden contener lógica simple de interacción entre sus átomos. Su HTML suele ser más extenso que el de un átomo, pero sigue siendo enfocado y reutilizable.

# 7.3 Organismos

Los organismos son agrupaciones de moléculas que forman secciones completas de una página. Por ejemplo, en este caso en el todo list es la lista de moléculas mostrando elementos varios de un todo. La ubicación del componente src\app\ui\todo\components\organisms\todo-list

HTML:



Los organismos son componentes relativamente complejos que agrupan varias moléculas y/o átomos para formar secciones funcionales completas de la interfaz de usuario, Los organismos representan bloques estructurales reutilizables que ya contienen lógica de interacción y disposición, y suelen ser independientes y autosuficientes dentro de la UI.

TS:

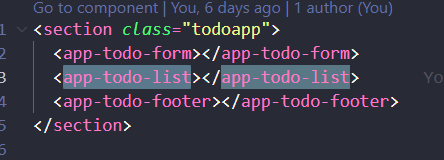


Dado que es un organismo se encarga recibir datos y pasarlos a los componentes hijos. No contiene lógica de negocio compleja, solo lógica de presentación y orquestación de componentes.

# 7.4 Plantillas

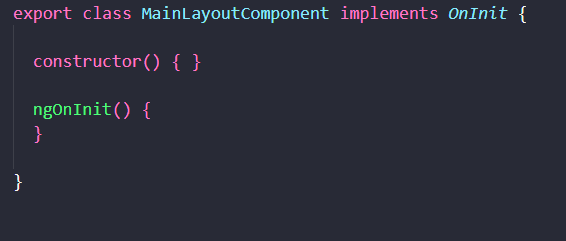
Las plantillas son estructuras que definen el diseño general de las páginas. Por ejemplo, src\app\ui\todo\layouts\main-layout este componente es la unión de varios organismos. Mostrando en conjunto la ui del manejo de los todos.

HTML:



Las plantillas o layouts definen la estructura general de una página o sección de la aplicación, organizando la disposición de los organismos, moléculas y átomos. Un layout no contiene detalles específicos de contenido, sino que sirve como marco para ubicar los bloques principales de la interfaz, como encabezados, barras laterales, áreas de contenido y pies de página. Como muestra el ejemplo se observa que se contiene varios bloques de organismos.

TS:

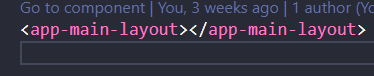


En este caso no se tiene logica si no que se tiene es el conjunto funcional de los organismos.

# 7.5 Paginas

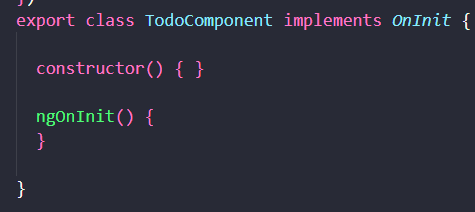
Las páginas son implementaciones específicas de las plantillas con datos reales y se encargan de la lógica de enrutamiento.

HTML:



Las páginas representan vistas completas de la aplicación, ensamblando layouts, organismos, moléculas y átomos para mostrar una experiencia de usuario final. Una página define la estructura y el contenido específico que verá el usuario en una ruta o sección concreta (por ejemplo, la página de tareas, login, dashboard, etc.).

TS:



En Atomic Design, las páginas (pages) pueden contener lógica, pero su responsabilidad principal es:

* Orquestar la composición de layouts, organismos, moléculas y átomos.
* Gestionar la lógica específica de la vista o ruta (por ejemplo, cargar datos, manejar navegación, obtener parámetros de la URL).
* Pasar datos y manejar eventos entre los componentes de la página.

# 8 Validaciones para la Integración de Componentes

Dentro de la aplicación, es fundamental definir validaciones para asegurar que cada componente se comporte como se espera. Para mantener la calidad y la mantenibilidad del código, se recomienda:

* Realizar pruebas unitarias a nivel de componente: Cada componente debe contar con sus propias pruebas unitarias, enfocadas en validar su lógica interna, renderizado y manejo de eventos. Estas pruebas deben ejecutarse de forma aislada, sin depender de la integración con otros componentes.

* Validar la carga y renderizado: Las pruebas deben asegurar que el componente se puede crear correctamente y que muestra los datos esperados en la vista.

* Verificar interacciones del usuario: Simular eventos como clics, entradas de texto o cambios de estado, y comprobar que el componente responde correctamente.

* Evitar pruebas integradas prematuras: En esta etapa, no se recomienda realizar pruebas que dependan de la integración entre varios componentes. El objetivo es garantizar que cada componente funcione correctamente de manera independiente. Las pruebas de integración entre componentes pueden realizarse en etapas posteriores, si es necesario.

* Responsabilidad única y cohesión: Verifica que cada átomo, molécula y organismo tenga una única responsabilidad clara. Los átomos no deben contener lógica de composición ni depender de otros componentes.

* Reutilización: Asegúrate de que los átomos y moléculas sean reutilizables en diferentes contextos. Valida que no haya duplicación de componentes con la misma función.

* Composición correcta: Las moléculas deben componerse de átomos y, opcionalmente, de otras moléculas simples. Los organismos deben componerse de moléculas y átomos, y orquestar su interacción. Los layouts y páginas deben ensamblar organismos y otros bloques de mayor nivel.

* Aislamiento en pruebas: Las pruebas unitarias de átomos y moléculas deben ejecutarse de forma aislada, sin depender de organismos o layouts. Los organismos pueden tener pruebas de integración de sus moléculas internas, pero sin lógica de negocio.

* Consistencia visual y de interfaz: Valida que los estilos y comportamientos de los átomos sean consistentes en toda la aplicación. Usa snapshots visuales o pruebas de regresión visual si es posible.

* Inputs y outputs claros: Verifica que los componentes expongan solo las propiedades y eventos necesarios para su función. Los átomos y moléculas no deben depender de estados globales.

* Documentación y ejemplos de uso: Cada componente debe tener documentación clara de su propósito, inputs, outputs y ejemplos de uso. Esto facilita la reutilización y el mantenimiento.

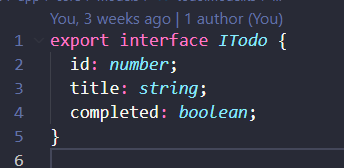
# 9 Implementación de la Arquitectura Hexagonal

La arquitectura hexagonal asegura que la lógica de negocio esté separada de la infraestructura. En el contexto de tu proyecto, esto se puede implementar de la siguiente manera:

# 9.1 Capa dominio

La capa de dominio representa el núcleo de la aplicación y contiene las entidades y la lógica de negocio pura. Esta capa no debe depender de ninguna tecnología externa.

* **Modelos:** Aquí se definen las entidades del negocio. Por ejemplo, el archivo todo.model.ts contiene la estructura del objeto "Todo".



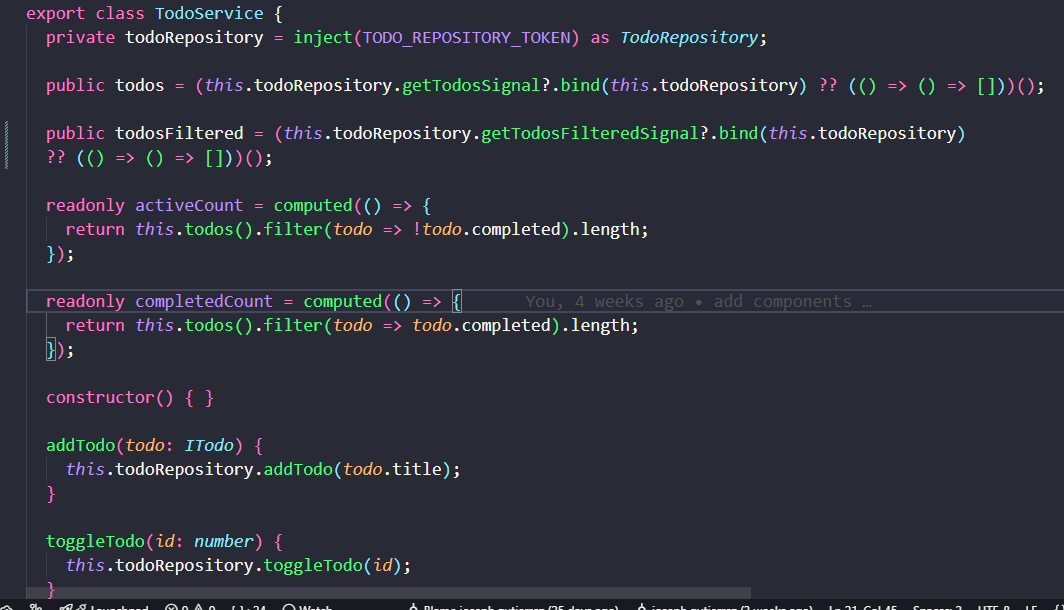
* **Repositorios:** Aquí se definen las interfaces que especifican cómo interactuar con las entidades. Esto permite que el servicio de "Todo" se mantenga desacoplado de la implementación concreta del repositorio, ya sea que use Signals, peticiones REST o almacenamiento en memoria.



# 9.2 Capa de aplicación

La capa de aplicación orquesta la lógica de negocio. Contiene los servicios que implementan casos de uso y coordina la interacción entre las entidades y las capas externas.

* **Servicios:** Aquí se implementan las reglas de negocio que interactúan con las entidades. El servicio utiliza la interfaz de repositorio para mantener un desacoplamiento.



En este servicio, TodoService interactúa con TodoRepository a través de la interfaz. Esto permite que el servicio funcione independientemente de la implementación específica del repositorio, manteniendo así un desacoplamiento efectivo. Por ejemplo, si decides cambiar de un repositorio en memoria a uno que haga peticiones REST, solo necesitas implementar la interfaz TodoRepository sin modificar el servicio.

# 9.3 Capa de Infraestructura

La capa de infraestructura implementa los adaptadores que permiten que la aplicación se comunique con el mundo exterior. Esto incluye la implementación de repositorios, servicios y otras dependencias externas.

* **Adaptadores:** Aquí se implementan las interfaces definidas en la capa de dominio. Por ejemplo, in-memory-todo.repository.ts puede ser un adaptador que almacena los "Todos" en memoria.



# 9.4 Configuración del modulo

En Angular, la inyección de dependencias debe ser configurada en el módulo correspondiente. Esto se hace generalmente en el archivo app.module.ts. Dado que en angular no existe la inyecion por interfaz y tenemos que especificar.



# 9.5 Reglas Generales de Implementación

* Cada capa debe tener una responsabilidad clara. No mezcles lógica de negocio en la capa de presentación ni lógica de presentación en la capa de dominio.
* Utiliza la inyección de dependencias para facilitar el testeo y la modularidad. Por ejemplo, el servicio de "Todo" se inyecta en el componente que lo utiliza a través de un token.
* Implementa pruebas unitarias para cada capa. Asegúrate de que la lógica de negocio en la capa de dominio esté bien testeada y que los servicios en la capa de aplicación manejen correctamente las entidades.
* Si en el futuro decides cambiar la forma en que gestionas el almacenamiento (por ejemplo, pasar de una implementación en memoria a una API REST), solo necesitarás modificar los adaptadores en la capa de infraestructura, sin afectar las capas de dominio o aplicación.

# 10 creación de un Nuevo Proyecto utilizando el Arquetipo

# 10.1 Paso 1: Inicializar el Proyecto

Para iniciar un nuevo proyecto Angular, utiliza Angular CLI. Abre una terminal y ejecuta:

**$ ng new nombre-del-proyecto**

Asegúrate de seleccionar las opciones que mejor se adapten a tu proyecto durante la configuración inicial.

# 10.2 Paso 2: Estructura del Proyecto

Una vez creado el proyecto, organiza la estructura del proyecto siguiendo la arquitectura hexagonal y los principios de atomic design. Aquí hay una sugerencia de estructura de carpetas:

**src/**

**app/**

**core/**

**models/**

**repositories/**

**services/**

**todo/**

**infrastructure/**

**adapters/**

**persistance/**

**services/**

**shared/**

**components/**

**atoms/**

**molecules/**

**organisms/**

**directives/**

**pipes/**

**utils/**

**ui/**

**todo/**

**components/**

**atoms/**

**molecules/**

**organisms/**

**layouts/**

**pages/**

**index.html**

**main.ts**

# 10.3 Paso 3: Crear Componentes

Comienza creando átomos, moléculas y organismos. Por ejemplo, para crear un átomo de botón:

* Crea un nuevo componente de átomo:

**$ ng generate component ui/components/atoms/button**

* Implementa la lógica y el HTML del botón, asegurándote de que siga los principios de atomic design.

# 10.4 Paso 4: Implementar la Lógica de Negocio

Define las entidades y la lógica de negocio en la capa de dominio. Por ejemplo, crea un modelo de Todo.

# 10.5 Paso 5: Crear Servicios

Crea servicios en la capa de aplicación que utilicen las entidades de la capa de dominio. Un servicio para manejar los Todos podría verse así

# 10.6 Paso 6: Integrar la UI

En los componentes de la UI, utiliza los servicios para manejar la lógica de negocio y renderiza los componentes correspondientes.

# 10.7 Paso 7: Pruebas y Validaciones

Asegúrate de implementar pruebas unitarias para validar el comportamiento de tus componentes y servicios.

# 11 conclusión

Este documento sirve como guía para la implementación de un arquetipo de Angular basado en atomic design y arquitectura hexagonal, así como un tutorial para crear un nuevo proyecto. Siguiendo estas recomendaciones y pasos, se espera mejorar la calidad, mantenibilidad y escalabilidad de las aplicaciones desarrolladas.