Práctica Angular – Parte 1

Tic-Tac-Toe

Objetivos

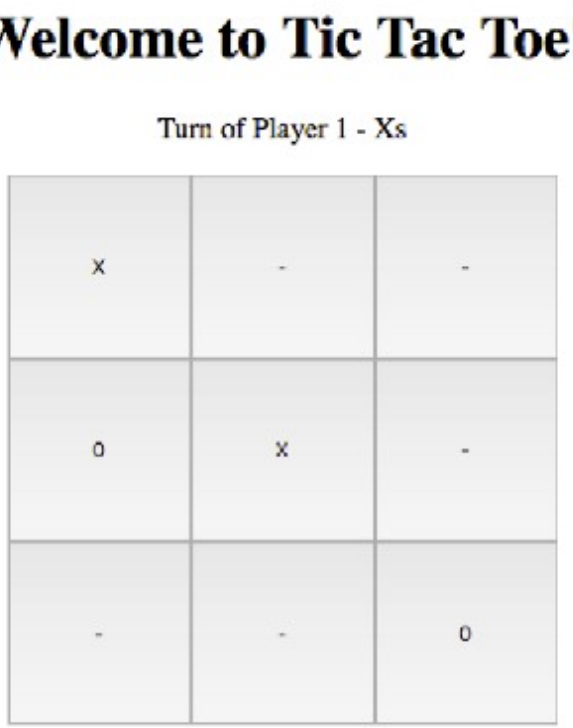
- De manera tutorizada crear una primera versión del juego de las tres en raya.
- Posteriormente, sin ayuda, el alumno deberá realizar tres actividades con tal de completar el juego.

Introducción

Usaremos conceptos vistos en las clases de teoría:

- La aplicación se dividirá en componentes, los cuales comparten información y usan pipes, directivas, data binding, …
- Encapsularemos los componentes en módulos.
- Usaremos servicios, observables y DI para gestionar el estado de la aplicación.

Funcionalidad para la versión 1.0



Como se aprecia en la figura anterior, en la parte superior del tablero se muestra qué jugador tiene el turno. El tablero consta de nueve casillas, las cuales pueden contener uno de estos tres caracteres:

- "-" indica que la casilla está vacía.
- "X" indica que la casilla contiene una ficha del jugador "X".
- "0" indica que la casilla contiene una ficha del jugador "0".

Los jugadores van alternando sus tiradas.
Gana quien consiga tres fichas en raya, tanto horizontal como vertical o diagonal.

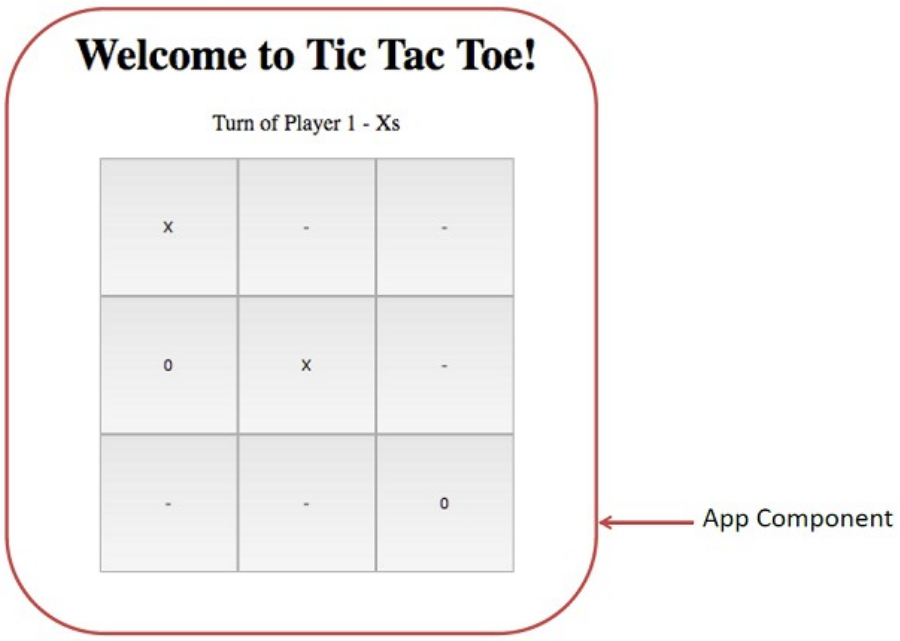
Estructura de la aplicación

Componentes

Los componentes son la estructura básica de una aplicación Angular. No en balde Angular es un framework orientado a componentes o también se dice que sigue el paradigma de componentes.
La aplicación estará formada por una serie de componentes. Veamos cuáles:

Contenedor principal (contenedor raíz)

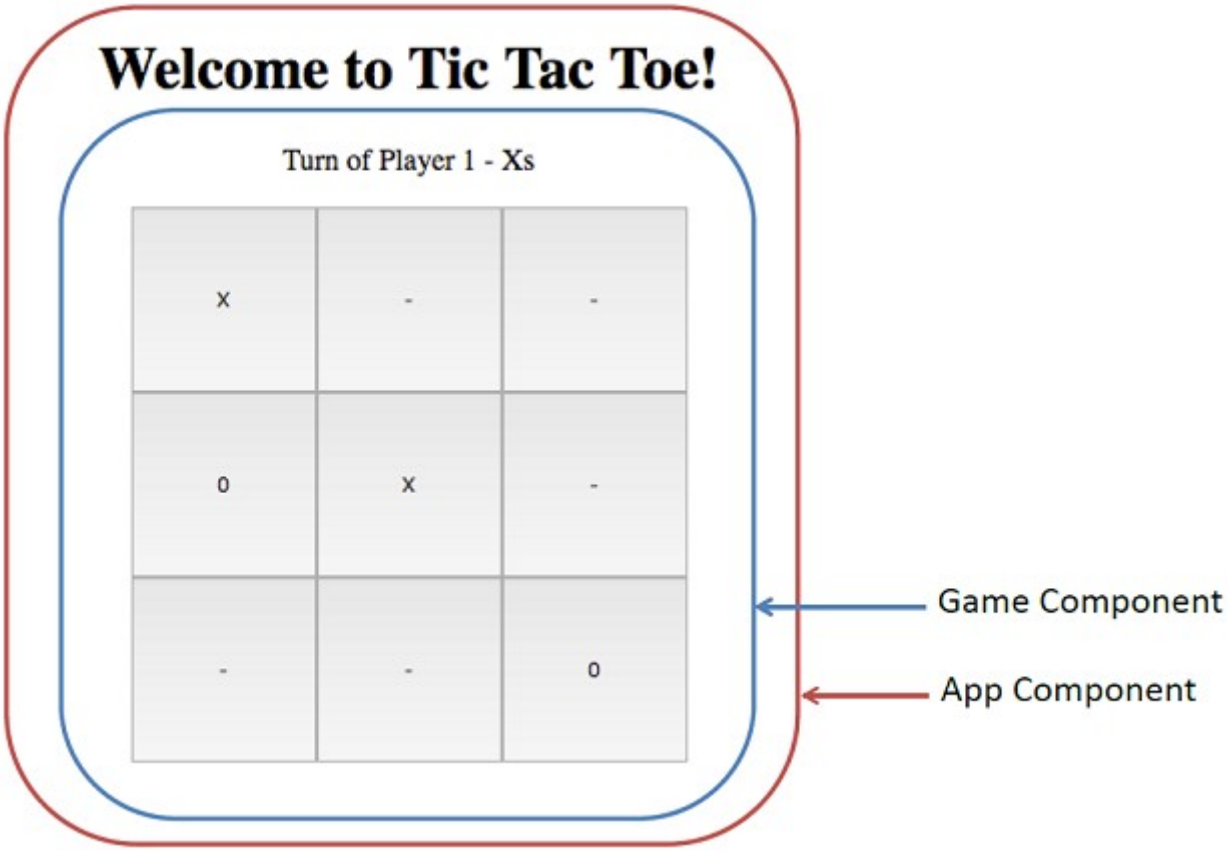
Se trata del componente predeterminado que crea Angular CLI al crear un nuevo proyecto. Este componente será el componente raíz o base, esto es, contendrá a cualquier otro componente de la aplicación. El nombre que Angular le da por defecto es **AppComponent**:



Contenedor para el juego

Crearemos un nuevo componente con el nombre **GameComponent.** Se trata de un componente que contendrá aquellos componentes específicos para el juego de las tres en raya. Esto incluye todas las funcionalidades y vistas necesarias para este juego.

Notad en la siguiente figura que GameComponent será un componente hijo de AppComponent:



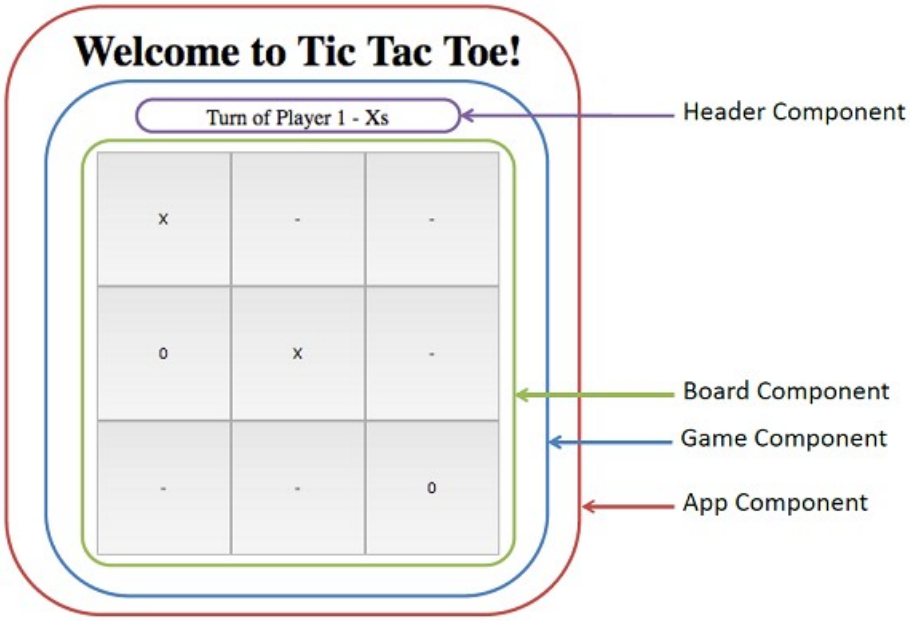
Se ha decido establecer esta distinción entre AppComponent y GameComponent para facilitar el mantenimiento de la aplicación si en un futuro se decide ampliar la funcionalidad de la misma, por ejemplo, añadiendo otro tipo de juegos, otros menús, etc, ya que de esta manera el impacto de estas modificaciones será menor.

En cualquier caso, notad que la estructura de los componentes es una decisión de diseño que toma el desarrollador de la aplicación.

Continuemos viendo el resto de la estructura de la aplicación...

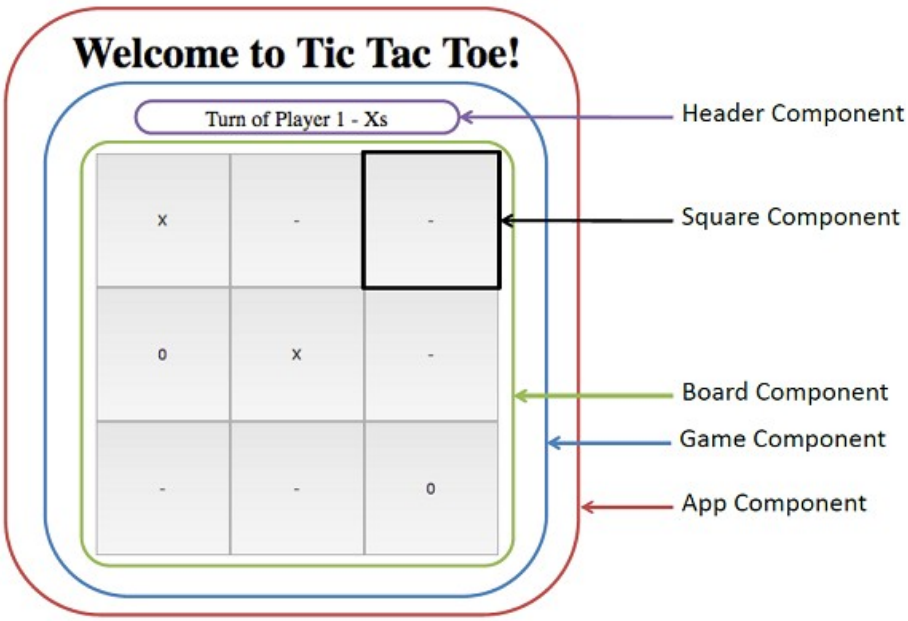
Componentes integrantes de GameComponent

Por otro lado, dentro de GameComponent tendremos un componente para mostrar el turno, llamado **HeaderComponent**, y otro componente para mostrar el tablero, llamado **BoardComponent**.



Componentes que conforman BoardComponent

Dentro de BoardComponent tendremos nueve casillas, las cuales pertenecerán a otro componente llamado **SquareComponent**.

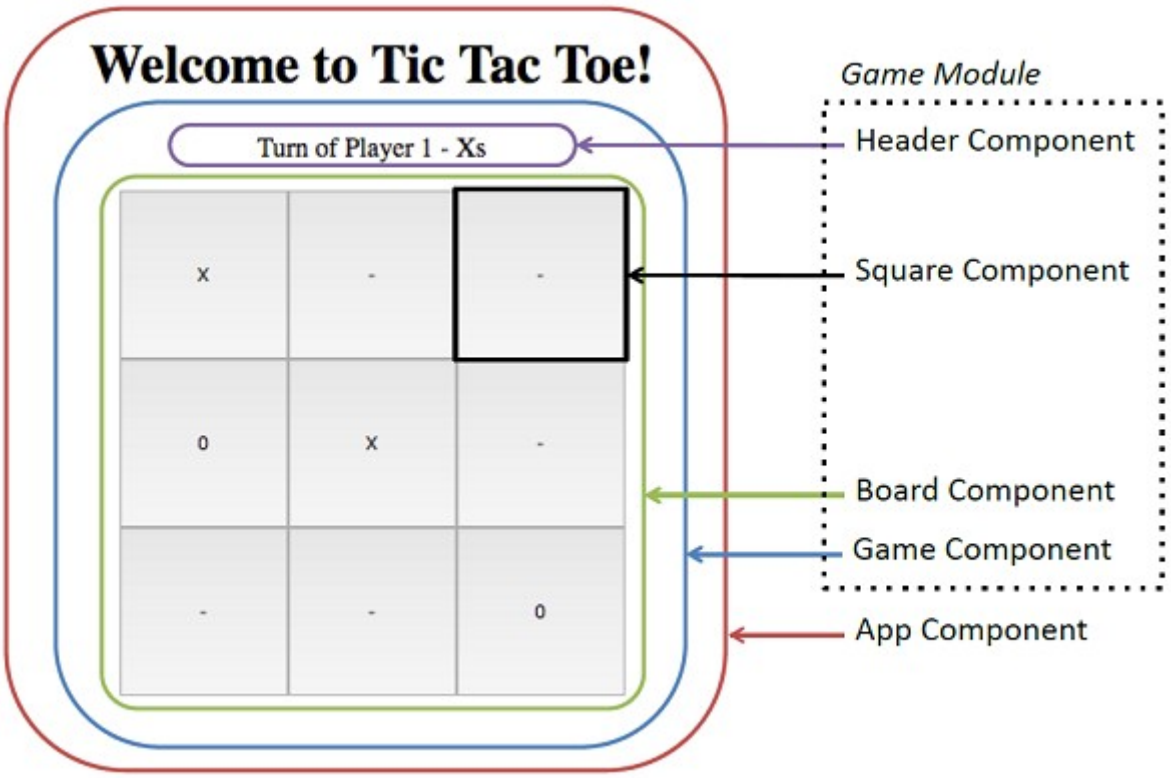


Cada uno de las casillas, esto es, cada SquareComponent, será un botón que el usuario podrá presionar si el valor de la casilla es un “-”, causando que cambie su valor a “X” o “0”, dependiendo del turno en juego.

Módulos

Para una mejor organización y encapsulación de componentes, Angular nos proporciona los módulos. Gracias a los módulos podemos agrupar componentes, obteniendo así módulos con alta cohesión y bajo acoplamiento, principios fundamentales del buen diseño.

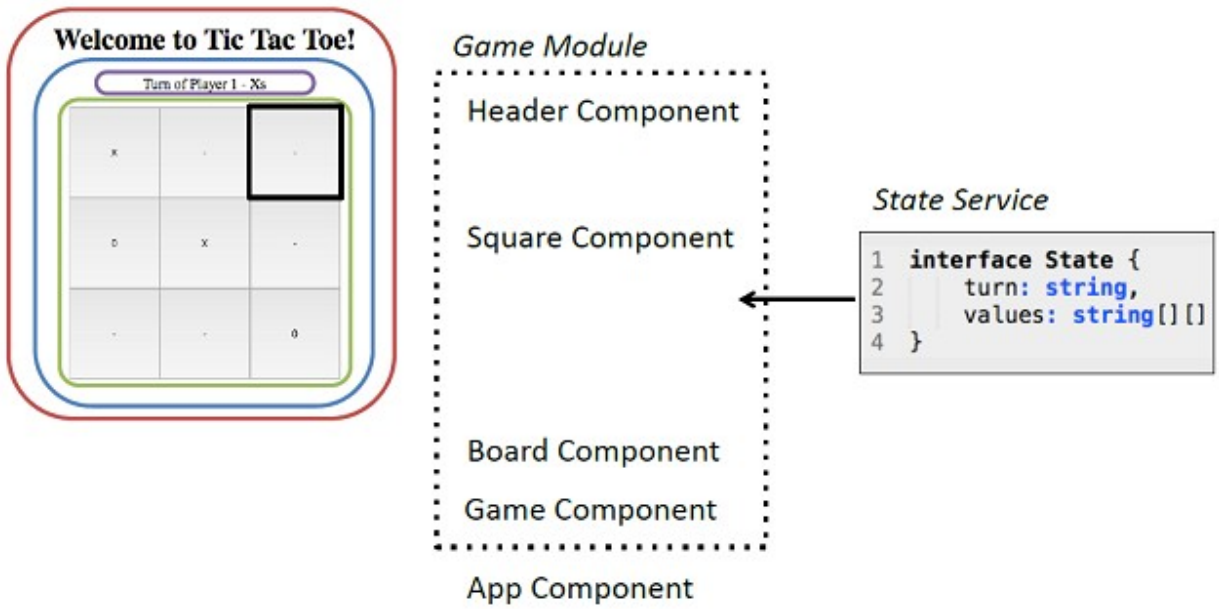
Aunque en nuestra aplicación, por tratarse de un programa pequeño, podríamos prescindir de crear un módulo adicional, vamos a crear uno llamado **GameModule**, el cual contendrá a todos los componentes de la aplicación, a excepción de AppComponent, ya que éste pertenecerá al módulo predeterminado, es decir a AppModule.



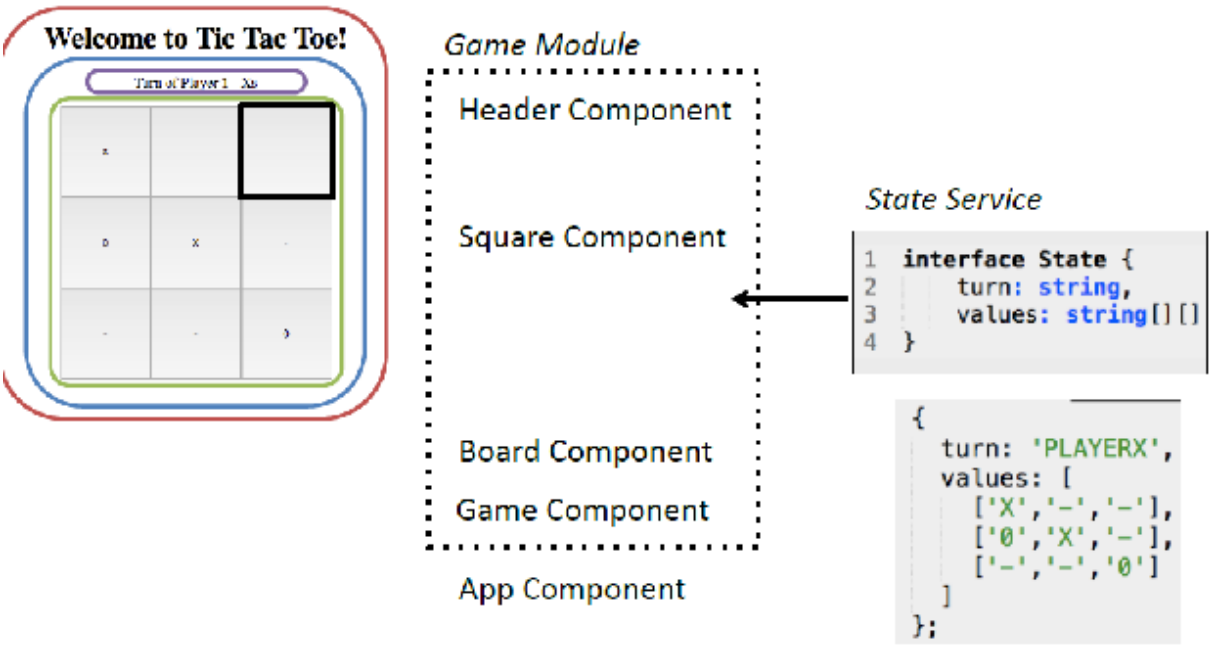
Es una buena práctica agrupar en módulos componentes estrechamente relacionados, así como otros elementos: pipes, directivas, clases, interfaces, etc. Si más adelante cambian los requisitos podremos afrontar las modificaciones creando nuevos módulos de componentes.

Servicios

Finalmente, para gestionar adecuadamente el estado de la aplicación, crearemos un servicio, cuyo nombre será **StateService**. Este servicio contendrá un interfaz llamada **Service** que representará el turno en curso y estado actual del tablero (cómo están las casilla en todo momento), utilizando para esto último una matriz de 3x3:

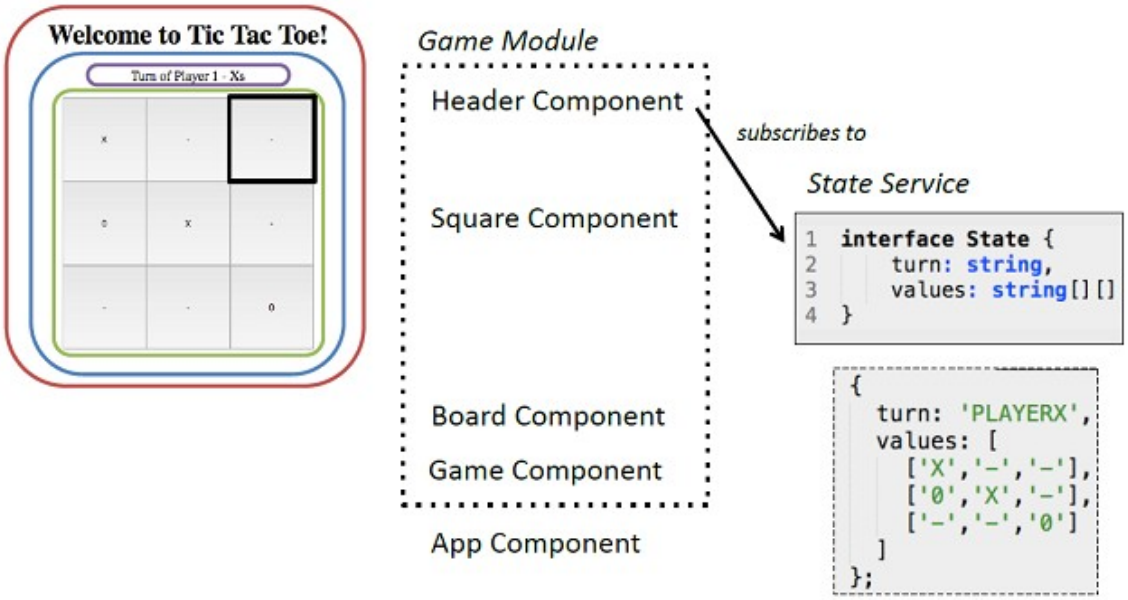


Ejemplo del estado de la aplicación en un momento dado:

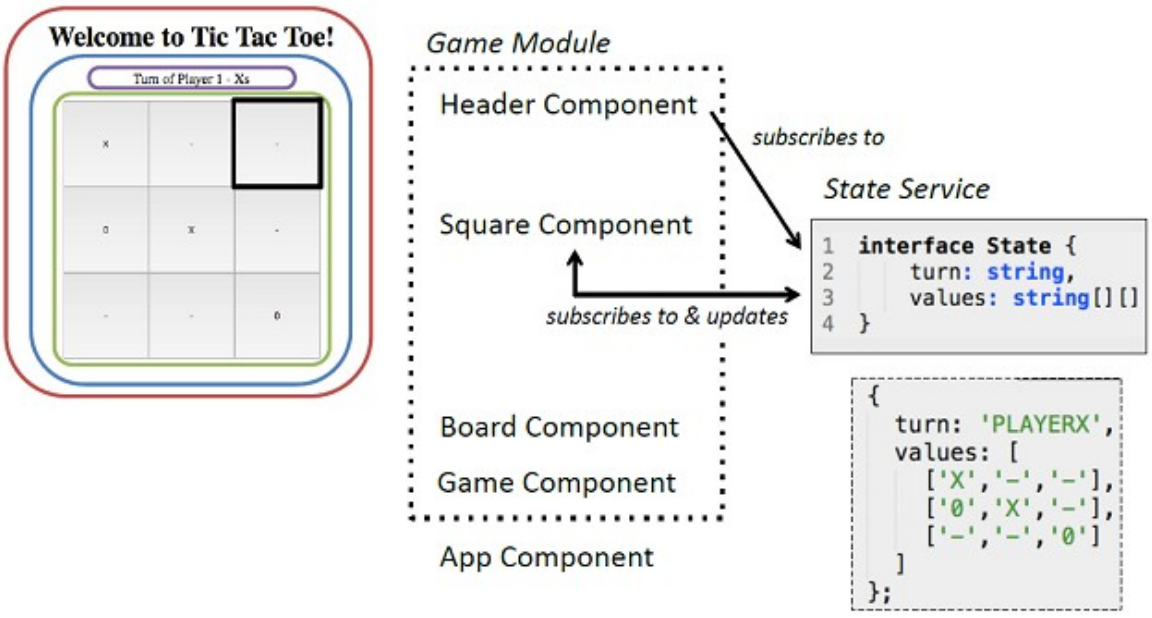


Suscripción a eventos

HeaderComponent estará suscrito a los cambios en **StateService**. De esta manera podrá mostrar adecuadamente el turno actual en todo momento



De la misma manera, **SquareComponent** necesita suscribirse a los cambios en **StateService**. Con esto se consigue que cada casilla del tablero muestre en todo momento su valor correcto. No obstante aquí tendremos un doble *binding (binding bidireccional)*, ya que cuando el usuario pulse sobre la casilla (que no es más que un botón), necesitaremos modificar el modelo asociado al SquareComponent en cuestión.



Conclusión

Hasta aquí hemos visto como se ha conceptualizado la aplicación en términos de componentes, módulos y servicios. También hemos visto qué componentes están suscritos a cambios en el estado de la aplicación o que incluso intervienen modificando el estado de la misma.

En la siguiente parte de esta práctica comenzamos a crear el proyecto.