



**ADD: MODULO MAPAS**

Carlos Andres Castaño Bustos  
2023-10-19

# Índice de contenidos

<b>1</b>	<b>Requisitos</b>	<b>2</b>
1.1	Requisitos funcionales . . . . .	2
1.2	Requisitos no funcionales . . . . .	2
<b>2</b>	<b>Motivadores de la iteración</b>	<b>3</b>
<b>3</b>	<b>Elementos a refinar</b>	<b>4</b>
<b>4</b>	<b>Conceptos que satisfacen los motivadores</b>	<b>5</b>
<b>5</b>	<b>Elementos de la arquitectura y responsabilidades</b>	<b>7</b>
<b>6</b>	<b>Diseño de vistas</b>	<b>9</b>
<b>7</b>	<b>Validación y análisis de resultados</b>	<b>16</b>

# 1 Requisitos

## 1.1 Requisitos funcionales

- El módulo debe mostrar un mapa visual del juego.
- Los pisos y pueblos deben tener nombre, descripción e imagen y estos deben ser configurables.
- El módulo debe obtener la información de los pisos y pueblos de la torre y mostrarlos en la sección de mapas.
- El módulo debe obtener la información de los pisos y pueblos del usuario, y mostrar cuales están bloqueados y desbloqueados.
- Al hacer clic en un pueblo en el mapa, se debe desplegar su nombre y una breve descripción del pueblo.

## 1.2 Requisitos no funcionales

- Se debe proporcionar una interfaz de usuario intuitiva para seleccionar los diferentes pisos y pueblos.
- La implementación debe de ser modular y escalable, permitiendo que se puedan implementar a futuro nuevos servicios de almacenamiento de información adicionales, sin que esto requiera cambios en el código principal del módulo.

## 2 Motivadores de la iteración

MOTIVADOR	DESCRIPCIÓN
Sistema modular y escalable	<ul style="list-style-type: none"><li>• Debe de ser independiente de los demás módulos del sistema.</li><li>• Debe de permitir la integración de diferentes servicios de almacenamiento de información.</li><li>• Debe de estar en la capacidad de cambiar el servicio de almacenamiento de información utilizado sin necesidad de cambiar el código principal del sistema.</li></ul>

Table 1: Componentes Motivadores. Fuente propia

### 3 Elementos a refinar

COMPONENTE	DESCRIPCIÓN
Módulo de mapas	<p>Este componente estará encargado de varios aspectos clave como:</p> <ul style="list-style-type: none"><li>• Mostrar un mapa visual del juego.</li><li>• Crear, editar y eliminar los diferentes pisos y pueblos del juego.</li><li>• Obtener y mostrar los diferentes pisos y pueblos configurados en el mapa.</li><li>• Obtener y mostrar los diferentes pisos y pueblos que se encuentran bloqueados y desbloqueados para el jugador.</li><li>• Obtener y mostrar la información relevante(Nombre, descripción e imagen) del pueblo de la torre al ser seleccionado.</li><li>• Permitir al jugador cerrar el modulo.</li></ul>

Table 2: Elementos a refinar. Fuente propia

## 4 Conceptos que satisfacen los motivadores

ESTILO ARQUITECTURA	JUSTIFICACIÓN
Cliente-Servidor	El estilo de arquitectura que se ajusta a las necesidades del proyecto es el de Cliente-Servidor, ya que este implica una separación clara entre el cliente (en este caso, las vistas del juego desarrolladas en Unity) y el servidor (donde reside la lógica de almacenamiento de la información).

Table 3: Estilo de arquitectura. Fuente propia

PATRON DE DISEÑO	JUSTIFICACIÓN
Patrón Strategy	Este patrón permite definir diferentes estrategias de almacenamiento de información (por ejemplo, el servicio de Cloud Save de Unity Services o la Realtime Database de Firebase u otro adicional) como clases separadas. De esta manera, el sistema puede cambiar dinámicamente el servicio de almacenamiento de información en tiempo de ejecución según las necesidades del usuario o del administrador del sistema. Este enfoque promueve la flexibilidad y la mantenibilidad del sistema, ya que cada estrategia puede ser modificada, reemplazada o ampliada sin afectar el resto del código.

Continúa en la siguiente página.

PATRON DE DISEÑO	JUSTIFICACIÓN
Patron Factory	Este patrón es el ideal para crear objetos del proveedor de almacenamiento de información basados en la estrategia seleccionada. La fábrica actúa como un intermediario entre el cliente y los servicios concretos de almacenamiento de información, permitiendo que el cliente solicite una instancia del proveedor sin conocer los detalles de implementación de esa instancia. Esto facilita la creación centralizada y dinámica de diferentes tipos de proveedor de almacenamiento de información (por ejemplo, una instancia del servicio de Cloud Save de Unity Services o una de la Realtime Database de Firebase) según las necesidades del sistema o del usuario. Además, el uso del patrón Factory asegura que la creación de las instancias sea coherente y estandarizada, lo que simplifica la lógica del cliente y mejora la mantenibilidad del código.
Patrón MVP (Model-View-Presenter)	En este enfoque, los modelos manejan la estructura de los datos, mientras que las vistas se centran exclusivamente en la presentación visual de la información. Los Presentadores facilitan la comunicación entre los Modelos, las Vistas y los servicios, permitiendo una manipulación precisa de la interfaz gráfica en respuesta a las acciones del usuario y los resultados de la autenticación. Esta separación de responsabilidades no solo simplifica el desarrollo y la depuración del código, sino que también mejora la flexibilidad y la mantenibilidad del sistema al permitir cambios en la lógica de negocio o en la interfaz de usuario de forma independiente.

Table 4: Patrones de diseño. Fuente propia

## 5 Elementos de la arquitectura y responsabilidades

NOMBRE	RESPONSABILIDAD	RELACIONES
Vistas	Las Vistas se encargan de mostrar la interfaz gráfica al usuario final. Son responsables de representar visualmente la información y de capturar las interacciones del usuario, como los eventos de cambio de vista, actualización de datos de usuario, obtener los logros, etc.	<ul style="list-style-type: none"><li>• Presenter</li></ul>
Presenter	El Presenter actúa como intermediario entre las Vistas y el DataManager. Se encarga de recibir los eventos del usuario desde las Vistas, procesarlos y coordinar las acciones necesarias con el AuthManager. También es responsable de actualizar las Vistas en función de los resultados de la autenticación.	<ul style="list-style-type: none"><li>• Vistas</li><li>• AuthManager</li></ul>
MapsManager	MapsManager es el componente central del módulo de mapas. Su función principal es coordinar las solicitudes de información de los pisos y pueblos y gestionar las estrategias de información proporcionadas por el DataProviderFactory.	<ul style="list-style-type: none"><li>• Presenter</li><li>• DataProviderFactory</li></ul>
DataProviderFactory	El DataProviderFactory se encarga de crear instancias de los Proveedores de información según la solicitud del MapsManager. Permite una creación centralizada y dinámica de los diferentes proveedores de información, proporcionando flexibilidad para adaptarse a diversas estrategias.	<ul style="list-style-type: none"><li>• MapsManager</li></ul>

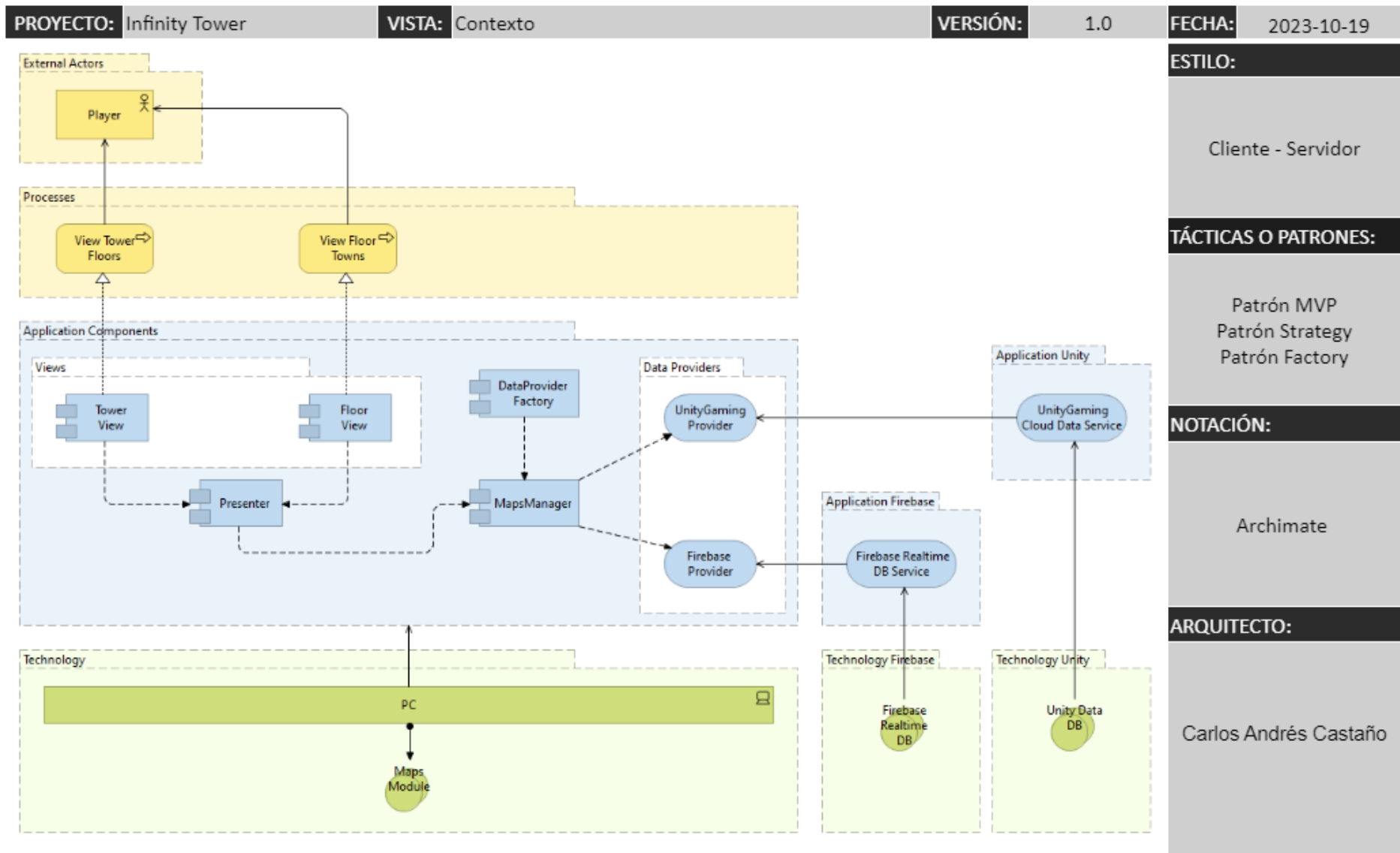
Continúa en la siguiente página.



NOMBRE	RESPONSABILIDAD	RELACIONES
Proveedor de información	El Proveedor de información representa a un proveedor de servicios como UnityGamingServices o Firebase, que ofrecen estrategias de almacenamiento de información robustas y seguras. Su función principal es proporcionar servicios de almacenamiento de información mediante servicios.	<ul style="list-style-type: none"> <li>• MapsManager</li> <li>• Base de datos proveedor de información</li> </ul>
Base de datos proveedor de información	La Base de Datos del Proveedor de información almacena y gestiona los datos. Cada instancia de la Base de Datos está asociada con un proveedor específico y es responsable de gestionar la lectura y escritura de datos relacionados con la información para esa estrategia particular.	<ul style="list-style-type: none"> <li>• Proveedor de información</li> </ul>

Table 5: Componentes del sistema. Fuente propia

# 6   Diseño de vistas



ESTILO:

Cliente - Servidor

TÁCTICAS O PATRONES:

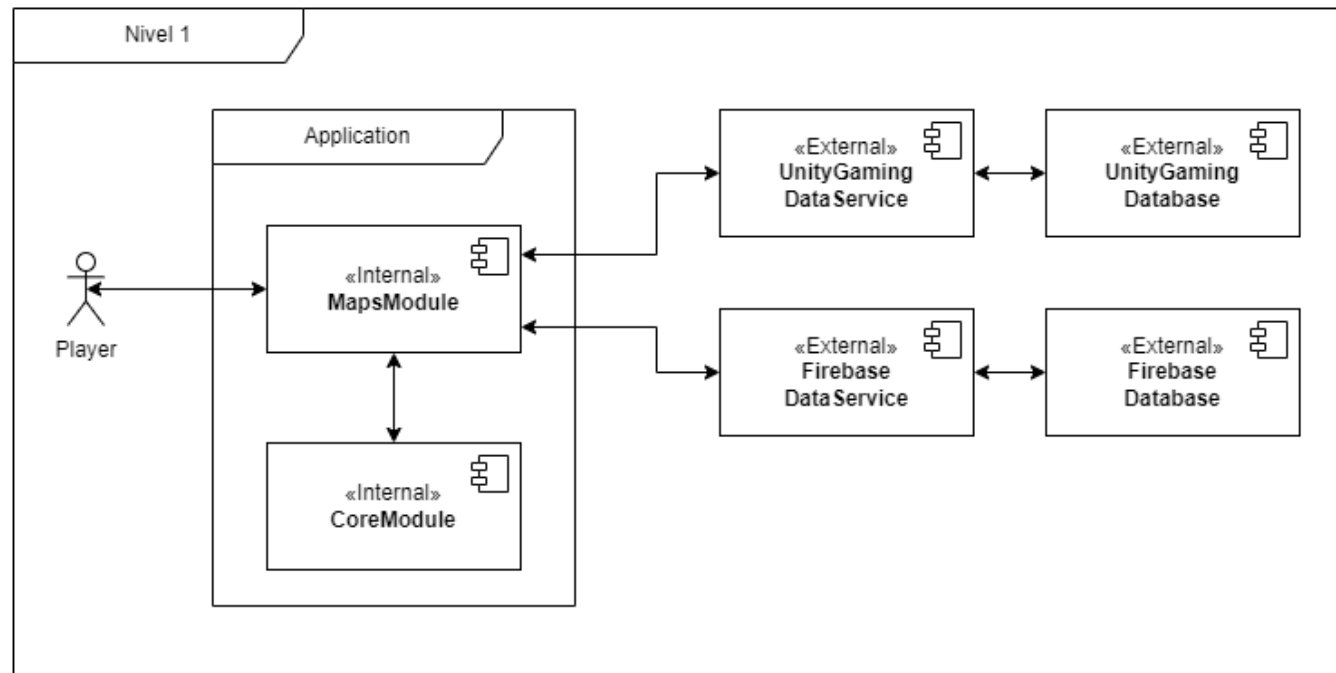
Patrón MVP  
Patrón Strategy  
Patrón Factory

NOTACIÓN:

UML

ARQUITECTO:

Carlos Andrés Castaño



**ESTILO:**

Cliente - Servidor

**TÁCTICAS O PATRONES:**

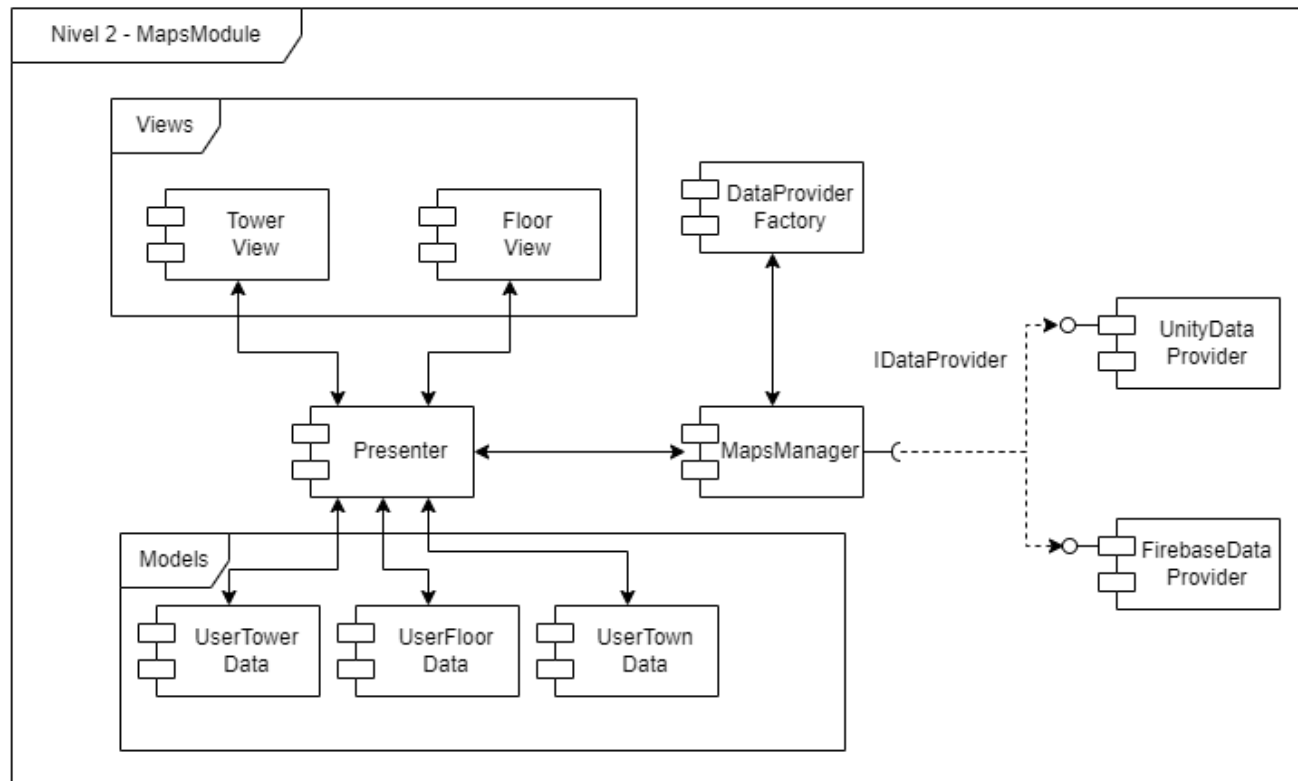
Patrón MVP  
Patrón Strategy  
Patrón Factory

**NOTACIÓN:**

UML

**ARQUITECTO:**

Carlos Andrés Castaño



PROYECTO:	Infinity Tower	VISTA:	Información	VERSIÓN:	1.0	FECHA:	2023-10-19
-----------	----------------	--------	-------------	----------	-----	--------	------------

ESTILO:

Cliente - Servidor

TÁCTICAS O PATRONES:

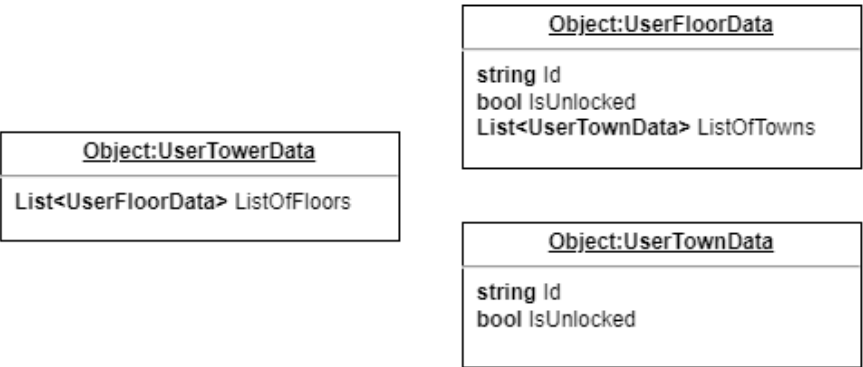
Patrón MVP  
Patrón Strategy  
Patrón Factory

NOTACIÓN:

UML

ARQUITECTO:

Carlos Andrés Castaño



**ESTILO:**

Cliente - Servidor

**TÁCTICAS O PATRONES:**

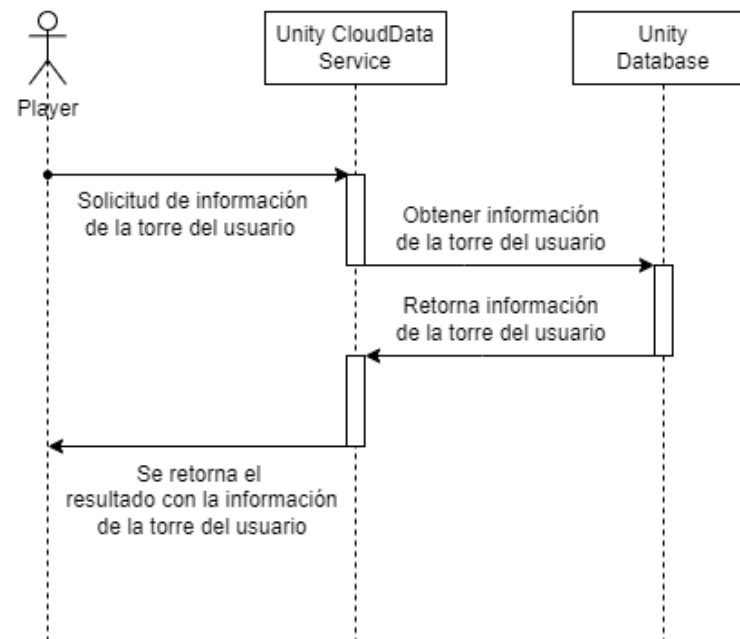
Patrón MVP  
Patrón Strategy  
Patrón Factory

**NOTACIÓN:**

UML

**ARQUITECTO:**

Carlos Andrés Castaño



**PROYECTO:** Infinity Tower

**VISTA:** Desarrollo

**VERSIÓN:** 1.0

**FECHA:** 2023-10-19

**ESTILO:**

Cliente - Servidor

**TÁCTICAS O PATRONES:**

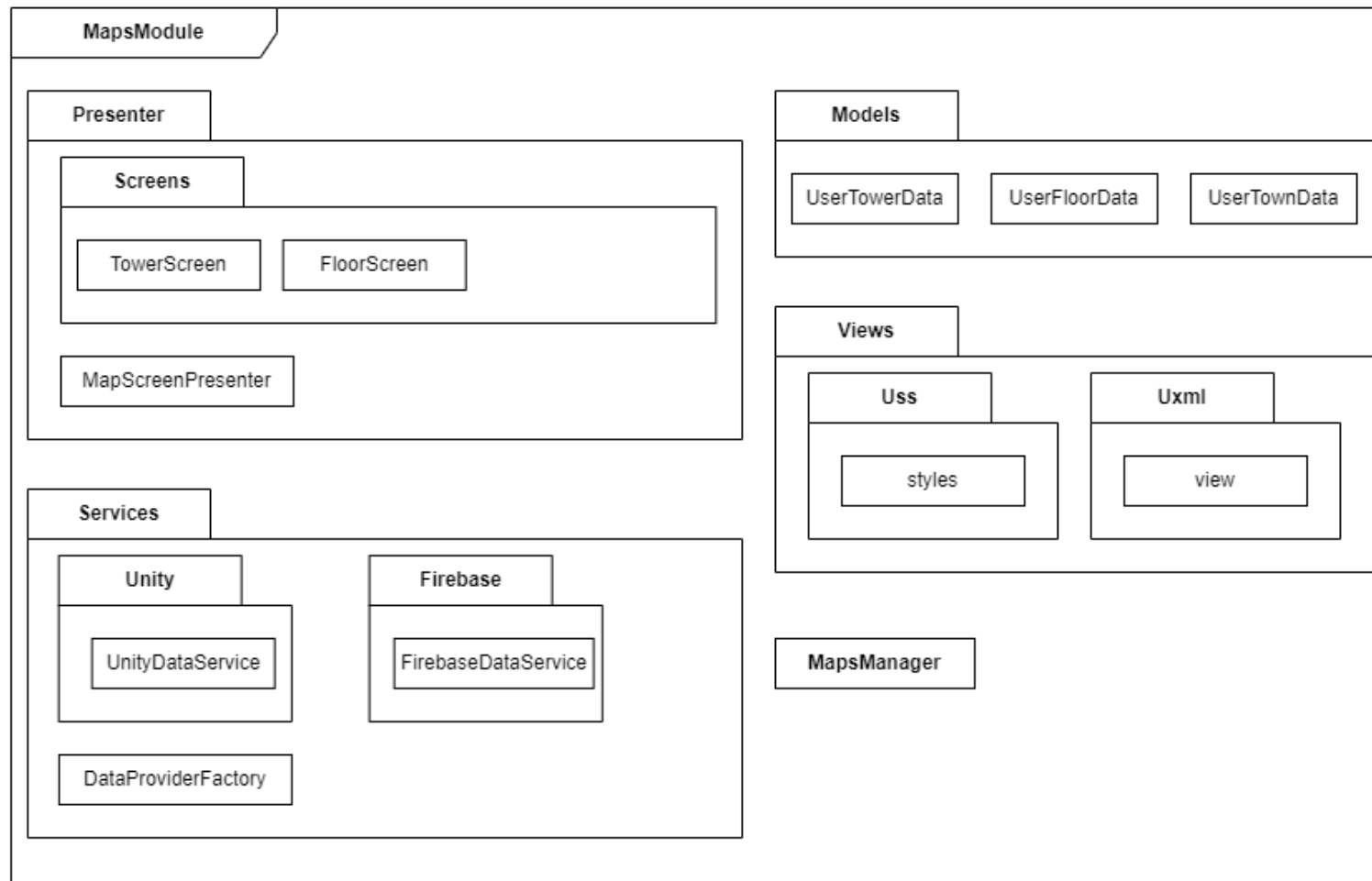
Patrón MVP  
Patrón Strategy  
Patrón Factory

**NOTACIÓN:**

UML

**ARQUITECTO:**

Carlos Andrés Castaño



**PROYECTO:** Infinity Tower

**VISTA:** Operación - Profile

**VERSIÓN:** 1.0

**FECHA:** 2023-10-19

**ESTILO:**

Cliente - Servidor

**TÁCTICAS O PATRONES:**

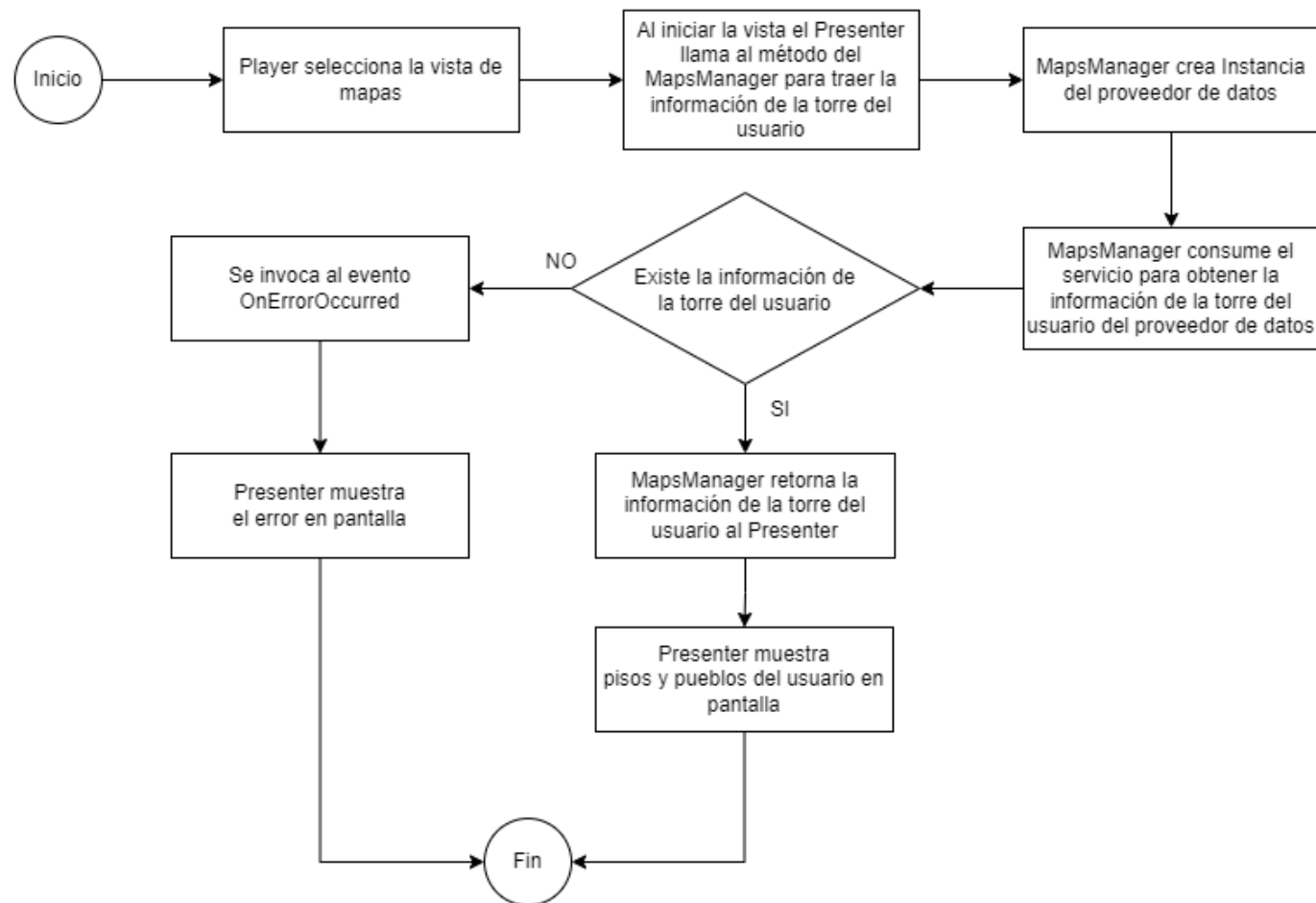
Patrón MVP  
Patrón Strategy  
Patrón Factory

**NOTACIÓN:**

UML

**ARQUITECTO:**

Carlos Andrés Castaño





## 7 Validación y análisis de resultados

	DESCRIPCIÓN
<b>RESULTADOS</b>	<p>Tomando como base el diseño de la arquitectura realizado para los modulos anteriores, este de igual forma ha producido un buen resultado transformando el código en algo comprensible, fácilmente mantenible y adaptable para la inclusión de nuevos servicios de información.</p> <p>Estos patrones han proporcionado una estructura coherente que ha simplificado la complejidad del sistema. Esta flexibilidad es crucial para garantizar que el código pueda evolucionar con el tiempo, adaptándose a nuevos requisitos y desafíos sin comprometer la estabilidad del sistema existente.</p>
<b>ACCIONES A SEGUIR</b>	<p>Dado que los resultados del diseño de la arquitectura han sido satisfactorios y han cumplido con los objetivos establecidos, la acción a seguir es llevar a cabo la implementación completa del modulo. Este paso implica traducir el diseño abstracto en código funcional, asegurando que todas las decisiones arquitectónicas se reflejen fielmente en la implementación.</p> <p>Las pruebas rigurosas también deben llevarse a cabo para validar que la implementación se ajusta a las expectativas y que todas las funcionalidades operan como se espera.</p>

Table 6: Resultados y acciones a seguir. Fuente propia