



ADD: MODULO AUTENTICACIÓN

Carlos Andres Castaño Bustos
2023-08-30

Índice de contenidos

1	Requisitos	2
1.1	Requisitos funcionales	2
1.2	Requisitos no funcionales	2
2	Motivadores de la iteración	3
3	Elementos a refinar	4
4	Conceptos que satisfacen los motivadores	5
5	Elementos de la arquitectura y responsabilidades	7
6	Diseño de vistas	9
7	Validación y análisis de resultados	17

1 Requisitos

1.1 Requisitos funcionales

- El módulo debe permitir al usuario registrarse.
- El módulo debe permitir al usuario iniciar sesión.
- El módulo debe permitir al usuario cerrar sesión.
- El módulo debe permitir al usuario cambiar el servicio de autenticación utilizado.

1.2 Requisitos no funcionales

- La interfaz de usuario debe ser intuitiva.
- La implementación debe de ser modular y escalable, permitiendo que se puedan implementar a futuro nuevos servicios de autenticación 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">• Debe de ser independiente de los demás módulos del sistema.• Debe de permitir la integración de diferentes servicios de autenticación.• Debe de estar en la capacidad de cambiar el servicio de autenticación utilizado sin necesidad de cambiar el código principal del sistema.

Table 1: Componentes Motivadores. Fuente propia

3 Elementos a refinar

COMPONENTE	DESCRIPCIÓN
Módulo de autenticación	<p>Este componente estará encargado de varios aspectos clave como:</p> <ul style="list-style-type: none">• Registro de usuarios.• Inicio de sesión.• Cierre de sesión.

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 autenticación y los datos sensibles).

Table 3: Estilo de arquitectura. Fuente propia

PATRON DE DISEÑO	JUSTIFICACIÓN
Patrón Strategy	Este patrón permite definir diferentes estrategias de autenticación (por ejemplo, autenticación por Unity Services o por Firebase u otro adicional) como clases separadas. De esta manera, el sistema puede cambiar dinámicamente el servicio de autenticación 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.
Patron Factory	Este patrón es el ideal para crear objetos de autenticación basados en la estrategia seleccionada. La fábrica actúa como un intermediario entre el cliente y los servicios concretos de autenticación, permitiendo que el cliente solicite una instancia de autenticación sin conocer los detalles de implementación de esa instancia. Esto facilita la creación centralizada y dinámica de diferentes tipos de autenticación (por ejemplo, una instancia del servicio de autenticación de Unity Services o una 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.

Continúa en la siguiente página.

PATRON DE DISEÑO	JUSTIFICACIÓN
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

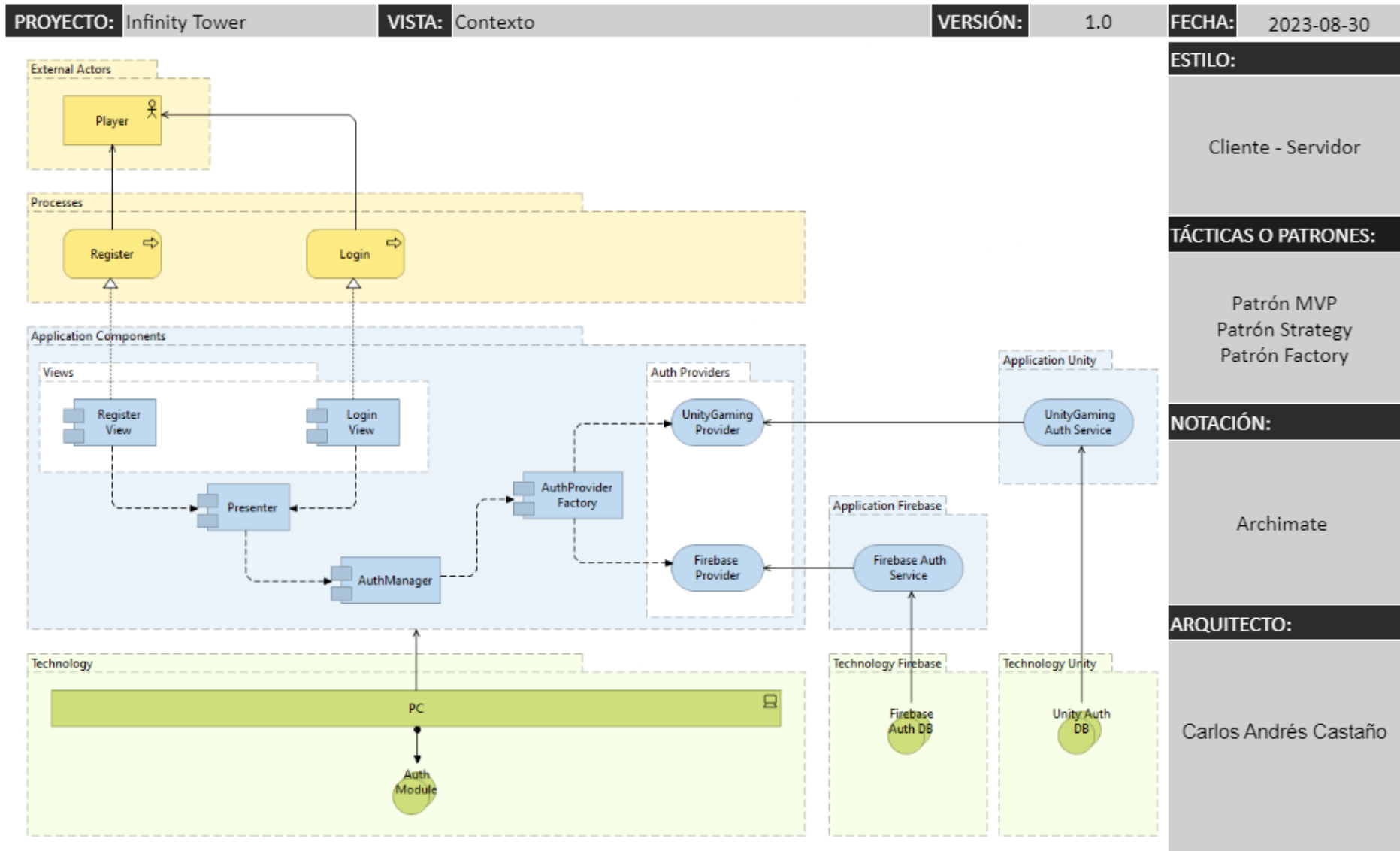
NOMBRE	RESPONSABILIDAD	RELACIONES
Vistas	Las Vistas se encargan de mostrar la interfaz gráfica al usuario final. Son responsables de representar visualmente la información y de capturar las interacciones del usuario, como los eventos de inicio de sesión y registro.	<ul style="list-style-type: none">• Presenter
Presenter	El Presenter actúa como intermediario entre las Vistas y el AuthManager. 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">• Vistas• AuthManager
AuthManager	AuthManager es el componente central del sistema de autenticación. Su función principal es coordinar las solicitudes de autenticación y gestionar las estrategias de autenticación proporcionadas por el AuthProviderFactory.	<ul style="list-style-type: none">• Presenter• Proveedor de autenticación
AuthProviderFactory	El AuthProviderFactory se encarga de crear instancias de los Proveedores de autenticación según la solicitud del AuthManager. Permite una creación centralizada y dinámica de los diferentes proveedores de autenticación, proporcionando flexibilidad para adaptarse a diversas estrategias.	<ul style="list-style-type: none">• AuthManager• AuthProviderFactory

Continúa en la siguiente página.

NOMBRE	RESPONSABILIDAD	RELACIONES
Proveedor de autenticación	El Proveedor de Autenticación representa a un proveedor de servicios como UnityGamingServices o Firebase, que ofrecen estrategias de autenticación robustas y seguras. Su función principal es proporcionar servicios de autenticación como el registro, inicio de sesión o cierre de sesión.	<ul style="list-style-type: none"> • AuthProviderFactory • Base de datos proveedor de autenticación
Base de datos proveedor de autenticación	La Base de Datos del Proveedor de Autenticación almacena y gestiona los datos sensibles requeridos, como las credenciales específicas de los usuarios para UnityGamingServices o Firebase. 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 autenticación para esa estrategia particular.	<ul style="list-style-type: none"> • Proveedor de autenticación

Table 5: Componentes del sistema. Fuente propia

6 Diseño de vistas



ESTILO:

Cliente - Servidor

TÁCTICAS O PATRONES:

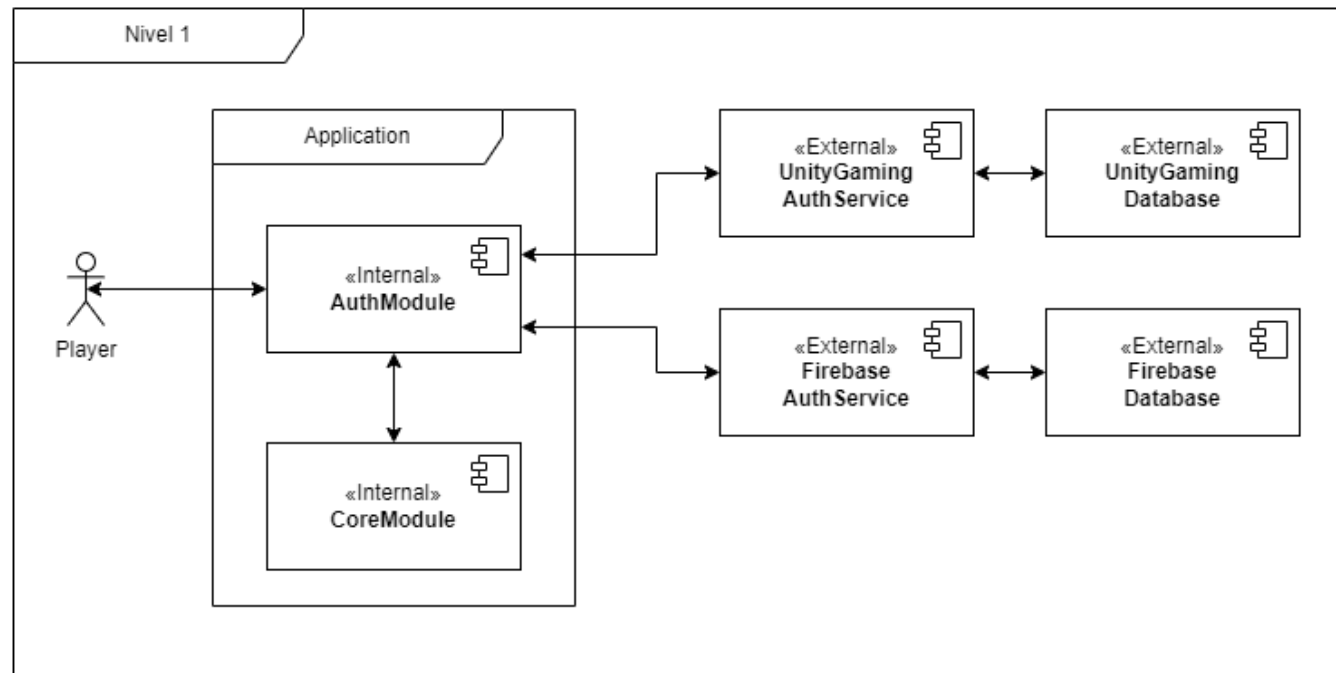
Patrón MVP
Patrón Strategy
Patrón Factory

NOTACIÓN:

UML

ARQUITECTO:

Carlos Andrés Castaño



ESTILO:

Cliente - Servidor

TÁCTICAS O PATRONES:

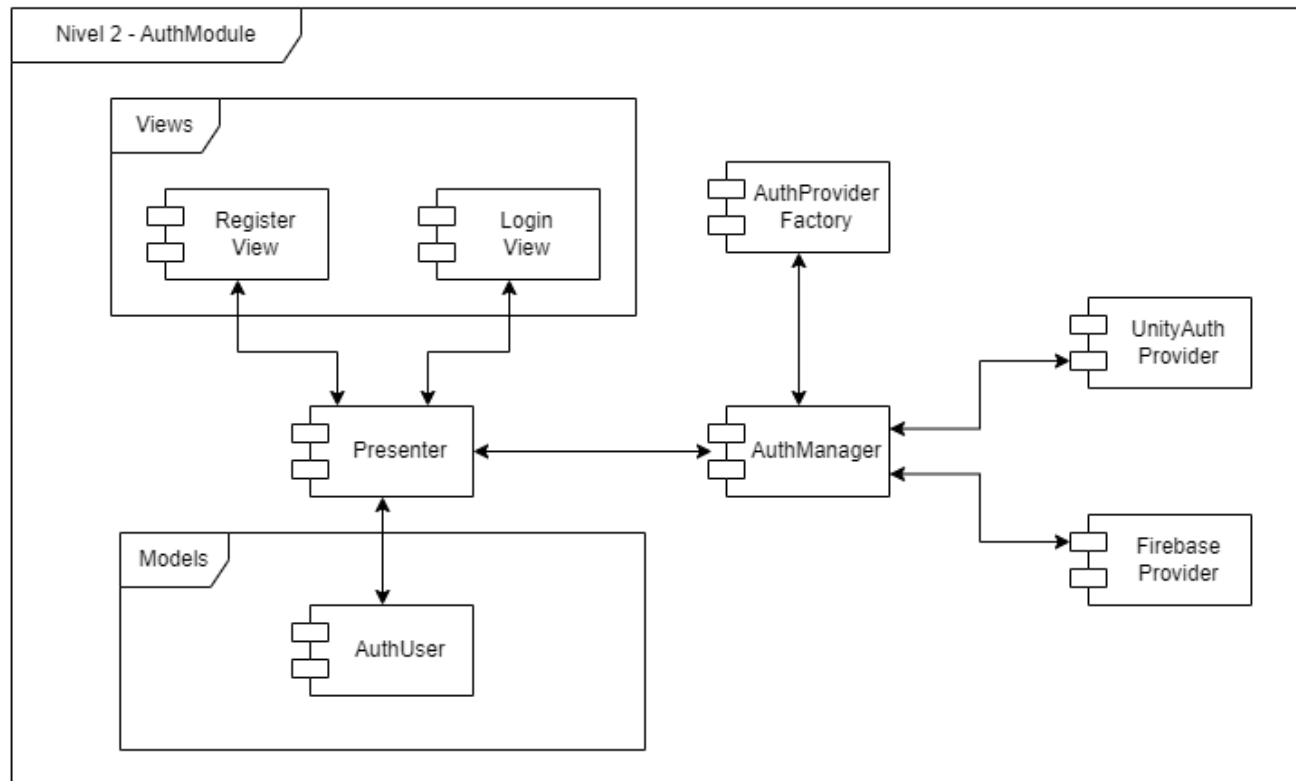
Patrón MVP
Patrón Strategy
Patrón Factory

NOTACIÓN:

UML

ARQUITECTO:

Carlos Andrés Castaño



ESTILO:

Cliente - Servidor

TÁCTICAS O PATRONES:

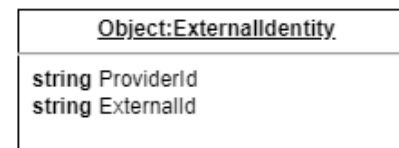
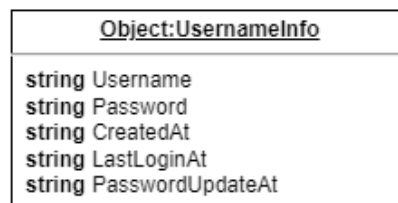
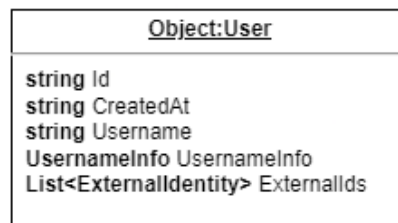
Patrón MVP
Patrón Strategy
Patrón Factory

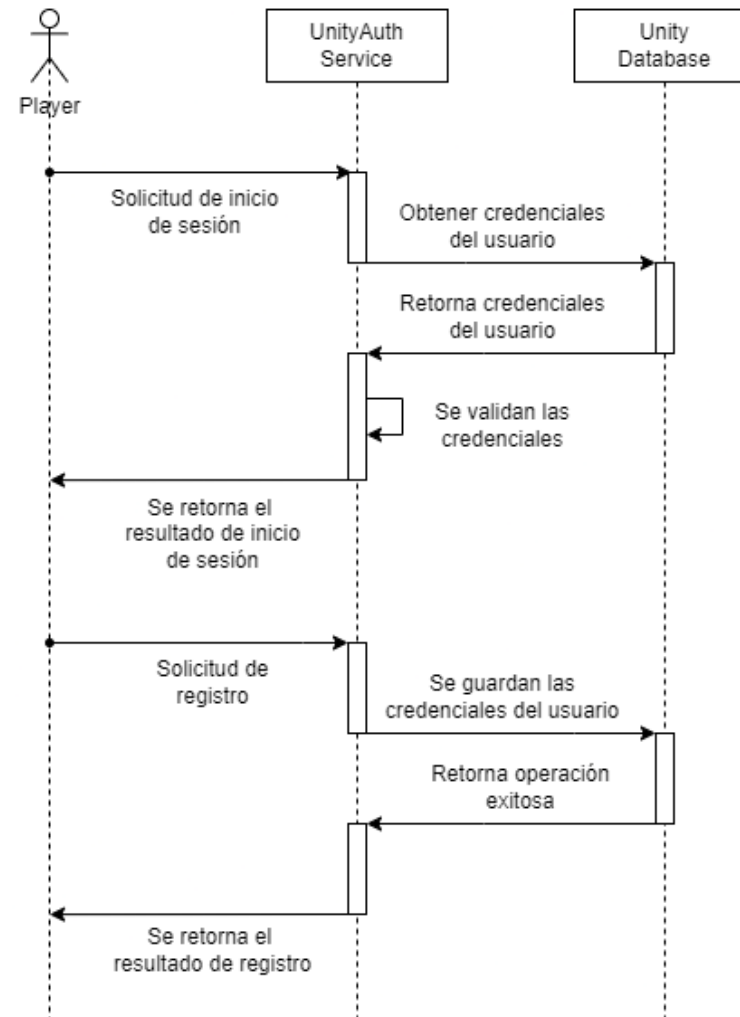
NOTACIÓN:

UML

ARQUITECTO:

Carlos Andrés Castaño





ESTILO:

Cliente - Servidor

TÁCTICAS O PATRONES:

Patrón MVP
Patrón Strategy
Patrón Factory

NOTACIÓN:

UML

ARQUITECTO:

Carlos Andrés Castaño

ESTILO:

Cliente - Servidor

TÁCTICAS O PATRONES:

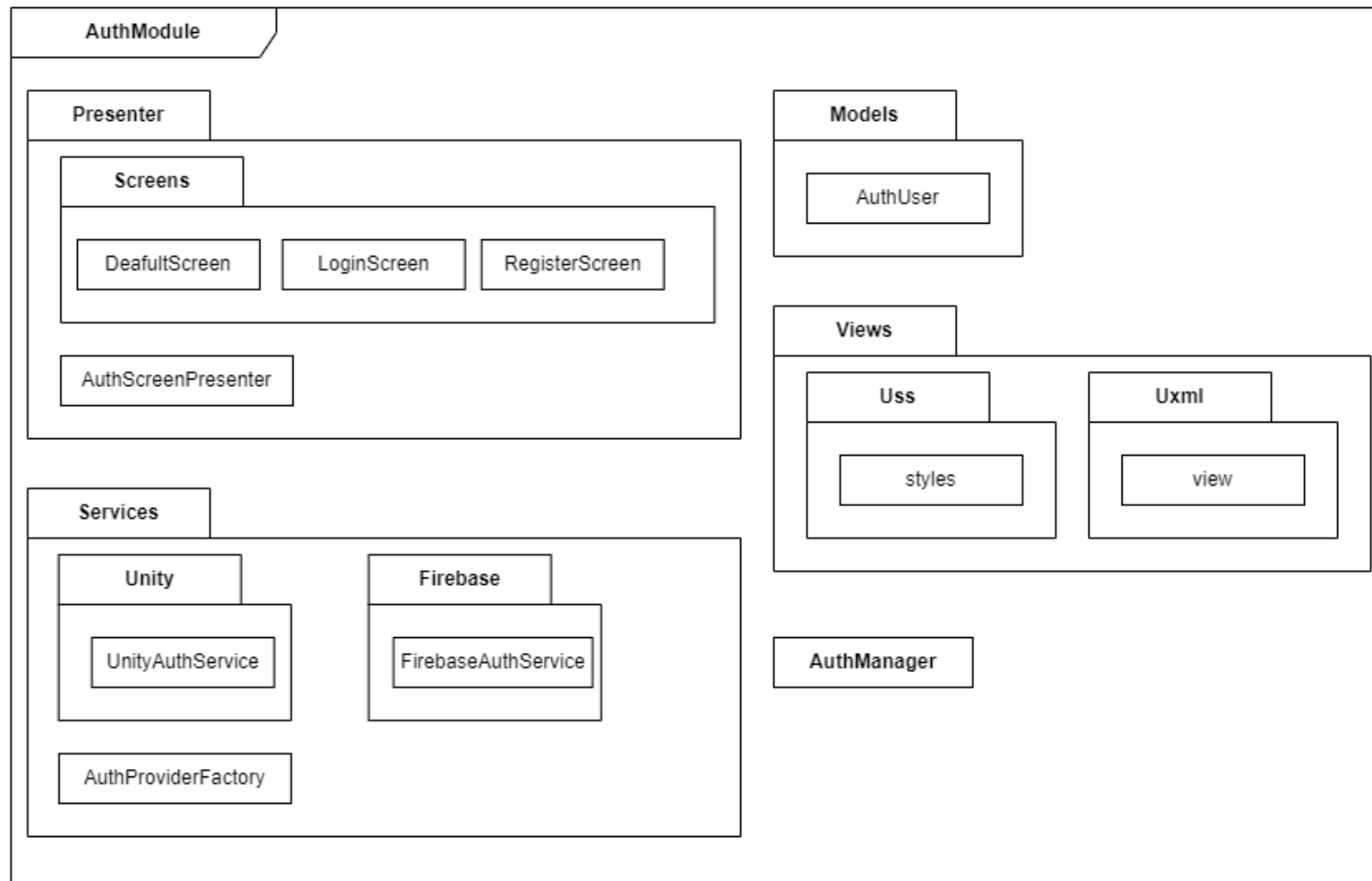
Patrón MVP
Patrón Strategy
Patrón Factory

NOTACIÓN:

UML

ARQUITECTO:

Carlos Andrés Castaño



PROYECTO: Infinity Tower

VISTA: Operación - Login

VERSIÓN:

1.0

FECHA:

2023-08-30

ESTILO:

Cliente - Servidor

TÁCTICAS O PATRONES:

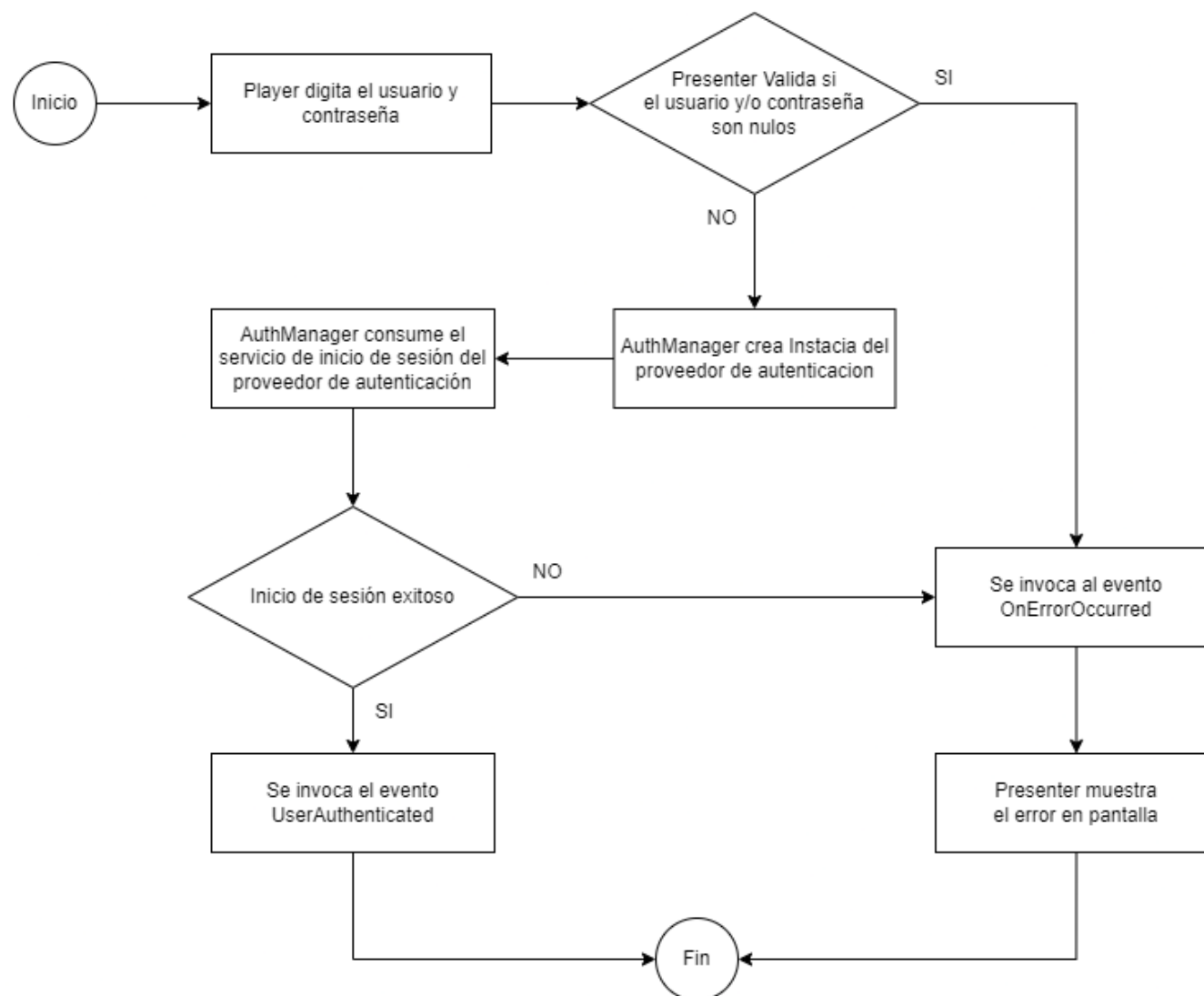
Patrón MVP
Patrón Strategy
Patrón Factory

NOTACIÓN:

Diagrama de flujo

ARQUITECTO:

Carlos Andrés Castaño



PROYECTO: Infinity Tower

VISTA: Operación - Register

VERSIÓN:

1.0

FECHA:

2023-08-30

ESTILO:

Cliente - Servidor

TÁCTICAS O PATRONES:

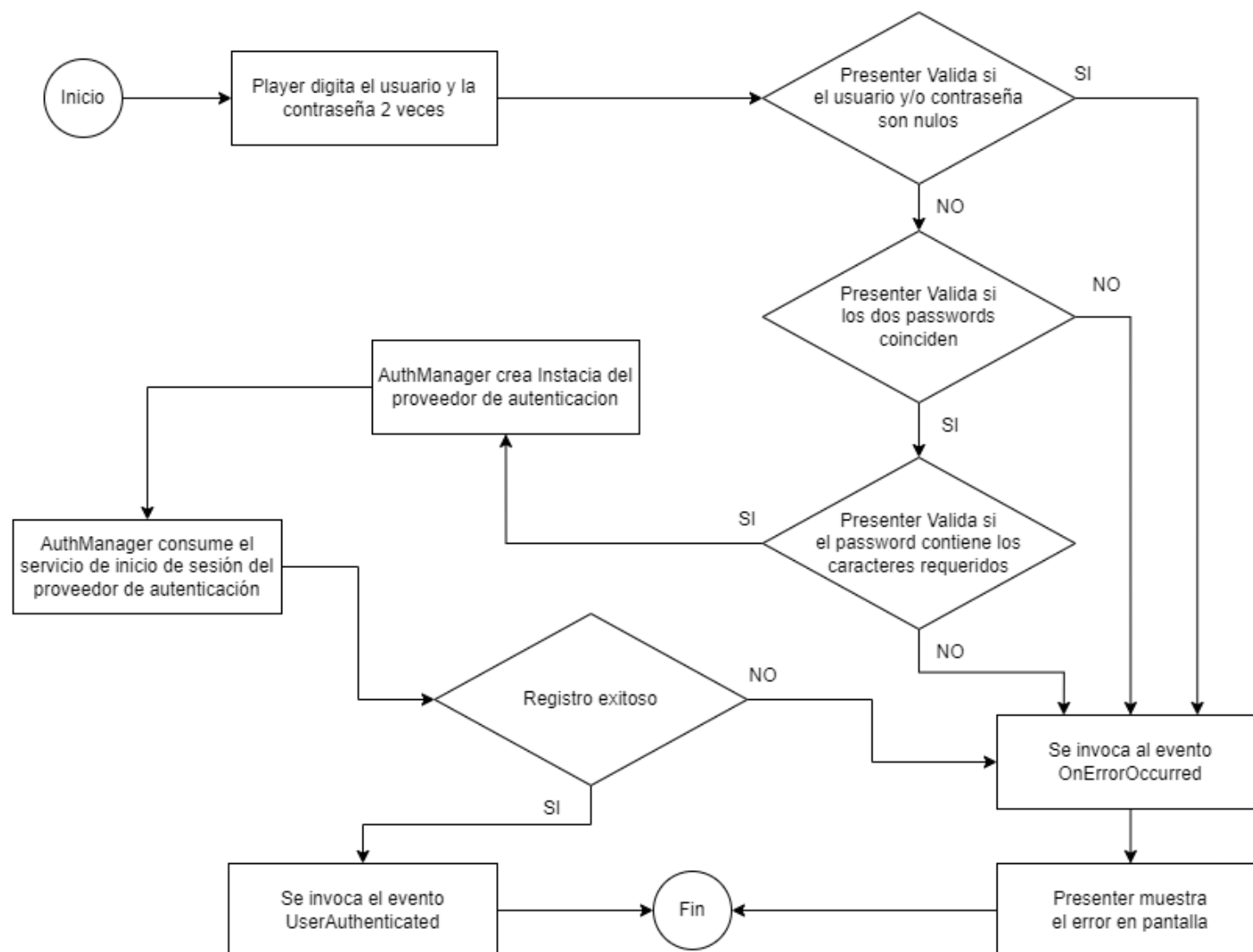
Patrón MVP
Patrón Strategy
Patrón Factory

NOTACIÓN:

Diagrama de flujo

ARQUITECTO:

Carlos Andrés Castaño



7 Validación y análisis de resultados

	DESCRIPCIÓN
RESULTADOS	<p>El diseño de la arquitectura ha producido un buen resultado al implementar patrones que han transformado el código en algo comprensible, fácilmente mantenible y adaptable para la inclusión de nuevos servicios de autenticación.</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>
ACCIONES A SEGUIR	<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