Diseño de software

Historia interactiva generada por IA SDD

UACM

Universidad Autónoma de la Ciudad de México

NADA HUMANO ME ES AJENO

Alejandro Viveros Hernandez

Profesor: Máximo Eduardo Sánchez Gutiérrez

Índice

Versión

Historial de Versiones - Documento de Arquitectura

Versión	Viewpoint Actualizado	Autor	Fecha	Descripción del Cambio
1.0	Documento inicial	Alejandro Viveros Hernández	26/02/2025	Creación del documento base con estructura de viewpoints
1.1	Context viewpoint (5.2)	Alejandro Viveros Hernández	07/03/2025	Definición inicial del contexto del sistema y actores principales, añadidos diagramas de contexto
1.2	Composition viewpoint (5.3)	Alejandro Viveros Hernández	14/03/2025	Establecimiento de componentes principales y sus relaciones estructurales
1.3	Logical viewpoint (5.4)	Alejandro Viveros Hernández	23/03/2025	Desarrollo de la vista lógica incluyendo el modelo de dominio y principales abstracciones

1.4	Patterns use viewpoint (5.7)	Alejandro Viveros Hernández	11/04/2025	Documentación de patrones arquitectónicos aplicados y justificación de uso
1.5	Interface viewpoint (5.8)	Alejandro Viveros Hernández	25/04/2025	Especificación detallada de interfaces entre componentes y sistemas externos
1.6	Structure & Interaction viewpoints (5.9, 5.10)	Alejandro Viveros Hernández	02/05/2025	Actualización conjunta de la estructura del sistema y flujos de interacción principales
1.7	State Dynamics viewpoint (5.11)	Alejandro Viveros Hernández	09/05/2025	Definición de los estados del sistema y transiciones entre estados
1.8	Algorithm viewpoint (5.12)	Alejandro Viveros Hernández	16/05/2025	Implementación de algoritmos clave y lógica de procesamiento central

1. Introducción

1.1 Propósito

El presente documento tiene como objetivo describir los requerimientos, diseño y arquitectura del sistema para el desarrollo de una historia interactiva generativa con Inteligencia Artificial (IA). Este sistema busca proporcionar una experiencia narrativa única y personalizada, en la que cada usuario pueda tomar decisiones que afecten el desarrollo de la historia.

La IA desempeñará un papel clave en la generación de contenido dinámico, asegurando coherencia en la trama y adaptándose en tiempo real a las elecciones del usuario. Además, el sistema incluirá un mecanismo de dificultad adaptativa que ajustará la complejidad y los desafíos dentro de la historia en función del desempeño y las decisiones del usuario.

El presente documento servirá como una guía integral para el desarrollo e implementación del sistema, proporcionando una estructura clara que facilitará la colaboración entre desarrolladores, diseñadores, testers y cualquier otro equipo involucrado en la creación del producto. Se detallarán los aspectos técnicos esenciales, incluyendo la arquitectura del software, los módulos principales, los requerimientos funcionales y no funcionales, así como los mecanismos de interacción entre los componentes del sistema.

En términos generales, este proyecto busca innovar en la forma en que las historias interactivas se presentan, utilizando el poder de la IA para proporcionar experiencias personalizadas y desafiantes.

1.2 Alcance

El sistema será una aplicación web de una sola página (Single Page Application - SPA) desarrollada con React para el frontend y Node.js para el backend. La plataforma permitirá a los usuarios interactuar con una historia generada dinámicamente por IA, donde sus elecciones influirán en la progresión de la narrativa.

Principales funcionalidades del sistema:

- **Generación de contenido dinámico:**
  La IA creará historias interactivas en tiempo real basándose en las preferencias y decisiones del usuario. El contenido generado garantizará coherencia narrativa y variabilidad en cada sesión de juego.
- **Dificultad adaptativa:**
  La ia evaluará el comportamiento del usuario en tiempo real y ajustará la dificultad de los desafíos en la historia. Esto se basará en parámetros como tiempo de respuesta, decisiones tomadas complejidad de la historia, y el punto de la historia.
- **Interfaz interactiva y envolvente:**
  La aplicación contará con una UI atractiva y fácil de usar, con una navegación fluida para mejorar la inmersión del usuario en la historia.
- **Gestión del progreso:**
  Se guardara el progreso temporalmente en diferentes puntos de la historia, permitiéndoles continuar la narrativa sin perder coherencia
- **Seguimiento del usuario:**
  El sistema almacenará datos sobre las elecciones del usuario para generar historias

coherentes y adaptar la trama a sus preferencias. Esto permitirá una experiencia más personalizada.

Este sistema será desarrollado como una aplicación web y no contemplará versiones para dispositivos móviles o escritorio en su fase inicial.

1.3 Visión General

1.3.1 Descripción del Proyecto

El proyecto consiste en el desarrollo de una **historia interactiva generativa con Inteligencia Artificial (IA)**, la cual permitirá a los usuarios sumergirse en una narrativa en constante evolución. A través de un sistema de **toma de decisiones**, los jugadores influirán en el desarrollo de la trama, haciendo que cada partida sea única.

El núcleo del sistema es un motor de IA capaz de generar contenido dinámico basado en las elecciones del usuario y en un mecanismo de dificultad adaptativa. Este último ajustará la complejidad de los eventos dentro de la historia en función del desempeño del jugador, asegurando un equilibrio entre desafío y entretenimiento.

El sistema se desarrollará como una **aplicación web de una sola página (SPA)** utilizando **React** para el frontend y **Node.js** para el backend.

1.3.2 Objetivos del Proyecto

El proyecto tiene como objetivos principales:

- **Crear una experiencia de historia interactiva personalizada:**
    - Permitir a los usuarios influir activamente en la narrativa a través de decisiones clave.
    - Generar historias dinámicas y coherentes en cada sesión.
- **Incorporar un sistema de dificultad adaptativa:**
    - Ajustar los desafíos en función del comportamiento y rendimiento del usuario.
    - Hacer que la historia evolucione en complejidad según el contexto y las elecciones del jugador.

- **Aprovechar la IA para la generación de contenido:**
    - Implementar algoritmos capaces de construir eventos narrativos, diálogos y descripciones de manera automática.
    - Hacer uso de PLN para mejorar la interacción y la coherencia de la historia.
- **Desarrollar una plataforma accesible y fácil de usar:**
    - Diseñar una interfaz intuitiva y envolvente utilizando React.
    - Garantizar una experiencia fluida en distintos dispositivos mediante tecnologías web modernas.

1.3.3 Características Clave del Sistema

El sistema ofrecerá varias funcionalidades diseñadas para maximizar la inmersión y personalización de la experiencia del usuario.

1. **Historia Generativa Dinámica:**
    - La IA generará eventos, descripciones y diálogos basados en las decisiones del usuario.
    - Cada historia será única, ofreciendo rejugabilidad y experiencias variadas.
2. **Toma de Decisiones con Impacto:**
    - Los jugadores enfrentarán múltiples opciones en cada punto clave de la historia.
    - Las elecciones influirán en el rumbo de la narrativa, cambiando personajes, entornos y conflictos.
3. **Sistema de Dificultad Adaptativa:**
    - El juego analizará el desempeño del usuario para modificar la complejidad de los desafíos.
    - Los eventos se ajustarán dinámicamente para mantener el balance entre reto y accesibilidad.
4. **Seguimiento del Usuario y Contexto Narrativo:**
    - Se registrarán las decisiones y el progreso para generar una historia coherente.

- La IA tendrá en cuenta las elecciones pasadas para dar continuidad lógica a la narrativa.

5. **Gestión del Progreso y Continuidad:**
    - Los jugadores podrán guardar su avance y continuar la historia en otra sesión.
    - La IA retomará el punto de la historia considerando el contexto previo.
6. **Interfaz Web Moderna e Intuitiva:**
    - Diseñada con React para una experiencia de usuario fluida.
    - Animaciones y efectos visuales para mejorar la inmersión narrativa.
7. **Filtro y Validación de Contenido**

    El sistema deberá incluir un módulo que analice el output de la IA antes de mostrarlo al usuario. Este módulo verificará que el contenido generado no incluya términos, frases o temas prohibidos según la lista de restricciones definidas.

    El prompt enviado a la IA deberá incluir instrucciones explícitas sobre los límites éticos y de contenido. Por ejemplo: “No generar contenido violento, discriminatorio o sexualmente explícito.”

    En caso de detectarse contenido no permitido, el sistema deberá:

    Reintentar la generación del contenido con un prompt modificado.

    Registrar el incidente en un log de seguridad.

    Notificar de forma discreta al usuario si fuera necesario (por ejemplo, mostrando un mensaje genérico).

1.3.4 Beneficios del Proyecto

El desarrollo de este sistema presenta múltiples beneficios tanto para los jugadores como para la industria del entretenimiento interactivo:

- **Personalización Total:** Se pretende que con el transcurso del aprendizaje de la ia el usuario vivirá una historia distinta, aumentando la inmersión y el compromiso con la narrativa.

- **Desafío Equilibrado:** La dificultad se adapta en tiempo real, asegurando una experiencia entretenida sin frustraciones.
- **Narrativas Únicas:** La IA genera contenido variado, permitiendo múltiples líneas argumentales y desenlaces diferentes.
- **Rejugabilidad Mejorada:** Gracias a la aleatoriedad y a las decisiones, cada partida ofrece una historia distinta.
- **Innovación en Juegos Narrativos:** La combinación de IA, PLN y dificultad adaptativa lleva la narrativa interactiva a un nuevo nivel.

1.3.5 Público Objetivo

El sistema está diseñado para una amplia gama de usuarios, incluyendo:

- **Aficionados a los juegos de rol y narrativas interactivas:** Jugadores que disfrutan de experiencias donde sus elecciones afectan el desarrollo de la historia.
- **Amantes de la literatura y la narración:** Usuarios interesados en explorar historias únicas generadas en tiempo real.
- **Entusiastas de la Inteligencia Artificial:** Personas curiosas sobre cómo la IA puede mejorar la creación de contenido narrativo.
- **Desarrolladores y diseñadores de juegos:** Profesionales que buscan inspiración en sistemas de generación procedural y adaptabilidad.

1.3.6 Alcance y Limitaciones

Si bien el sistema ofrecerá una experiencia narrativa avanzada, existen ciertas limitaciones que deben considerarse en su fase inicial.

Alcance:

Generación de historias dinámicas basada en decisiones.
Implementación de dificultad adaptativa según el desempeño del usuario.
Interfaz web optimizada para PC y dispositivos móviles.
Sistema de seguimiento del progreso y toma de decisiones.

Limitaciones:

No incluirá soporte para multijugador en la primera versión.
La IA requerirá ajustes para evitar errores en la coherencia narrativa.

El sistema no permitirá edición avanzada de historias en esta fase inicial. Puede haber restricciones en la cantidad de eventos generados en una sesión.

1.4 Material de Referencia

Este documento se basa en los siguientes estándares y referencias:

- **IEEE 1016-2016** – Estándar para Documentación de Diseño de Software (SDD).
- **UML 2.5** – Unified Modeling Language para diagramas de análisis y arquitectura.
- **Metodologías de desarrollo ágil** – Para la implementación iterativa del sistema.
- **Documentación de frameworks y bibliotecas utilizadas** (React, Node.js, IA Generativa).

1.5 Definiciones y Acrónimos

Término	Definición
SPA	Aplicación de una sola página (Single Page Application).
IA	Inteligencia Artificial, utilizada para la generación de contenido.
UML	Unified Modeling Language, utilizado para diagramas del sistema.
Dificultad Adaptativa	Sistema que ajusta la dificultad de la historia según el desempeño del usuario.
API	Interfaz de Programación de Aplicaciones, utilizada para la comunicación entre módulos.

Tabla 1.0 acrónimos y definiciones del proyecto

2. SYSTEM OVERVIEW

2.1 DESIGN VIEWPOINTS

2.2 introducción

Esta sección describe los diferentes puntos de vista del diseño del sistema, proporcionando una perspectiva clara de su estructura, interacciones y componentes. Cada punto de vista enfatiza un aspecto particular del diseño para garantizar la coherencia y la alineación con los requisitos del proyecto.

- Punto de vista del contexto
- Logical Viewpoint
- Information Viewpoint
- Interface Viewpoint •

Interaction Viewpoint

- State Dynamic Viewpoint

2.3 Punto de vista del contexto

Este punto de vista define los límites del sistema y sus interacciones con entidades externas. Incluye:

- Usuarios finales del sistema.
- Sistemas externos con los que interactúa.
- Flujos de datos entre el sistema y su entorno.

id	nombre	descripcion	Acciones
Cu1	Autenticar	El usuario se registra e inicia sesión en la plataforma para acceder a sus historias, preferencias y progreso.	Registro de nuevo usuario. Inicio de sesión con credenciales válidas. Recuperación y gestión de sesión activa (por ejemplo, token de autenticación). (Posible integración de verificación de email u otros métodos de autenticación).
Cu2	Personalizar la Experiencia Narrativa	El usuario establece sus preferencias (género, tono, complejidad, etc.) para que la IA pueda generar una historia coherente y adaptada a sus gustos.	Selección de opciones de personalización antes de iniciar una historia. Actualización de preferencias en cualquier momento durante la sesión.
Cu3	Iniciar o Reanudar	El usuario inicia una nueva historia o reanuda una historia previamente guardada, asegurándose de gue con el contexto y progreso guardado.	Selección de “nueva historia” o “reanudar historia” en el menú principal. Carga de contexto, resumen y decisiones previas desde la base de datos.
Cu4	Generación de Contenido por IA	El sistema solicita al motor de IA (por ejemplo, ChatGPT) la generación de contenido narrativo tomando en cuenta el contexto actual, las	Envío de prompts enriquecidos que incluyan resúmenes, decisiones anteriores y parámetros éticos.

Tabla 2. Tabla de casos de uso

		decisiones y las preferencias definidas.	Recepción y validación del contenido generado.
Cu5	Validación Ética y Filtro de Contenido	El sistema analiza y filtra el contenido generado por la IA para asegurar que no se incluyan temas, lenguaje o contenidos prohibidos según el marco ético establecido.	Análisis del output mediante algoritmos de NLP o reglas heurísticas. Reintento automático de generación en caso de contenido inadecuado, con registro del incidente. Notificación discreta o logueo para auditorías internas.
Cu6	Toma de Decisiones y Evolución de la Historia	Durante el desarrollo de la historia, el usuario elige entre diversas opciones que influyen en la narrativa y en el desarrollo de la dificultad.	Presentar al usuario una serie de opciones en cada página o escena. Registrar la opción elegida y actualizar el estado de la historia. Confirmar y actualizar el contexto narrativo que se enviará en futuras interacciones con la IA.
Cu7	Adaptación Dinámica de la Dificultad	El sistema evalúa el rendimiento y las decisiones del usuario para ajustar la dificultad de la narrativa en tiempo real (por ejemplo, complicando las opciones o modificando la complejidad del texto generado).	Análisis automático del desempeño mediante métricas o heurísticas definidas. Modificación de parámetros de generación (e.g., dificultad, número y complejidad de opciones) en el motor de IA

Cu8	Gestión del Historial y Seguimiento del Usuario	El sistema guarda y presenta un resumen del progreso del usuario, permitiendo revisar las decisiones tomadas, el estado actual de la historia y el rendimiento global.	Almacenamiento persistente de cada página, decisión y resumen del contexto en la base de datos. Visualización por parte del usuario de un historial o cronología de su historia (incluyendo gráficos o estadísticas simples si se requiere). Opciones para exportar o compartir el historial.

Diagrama de casos de uso general

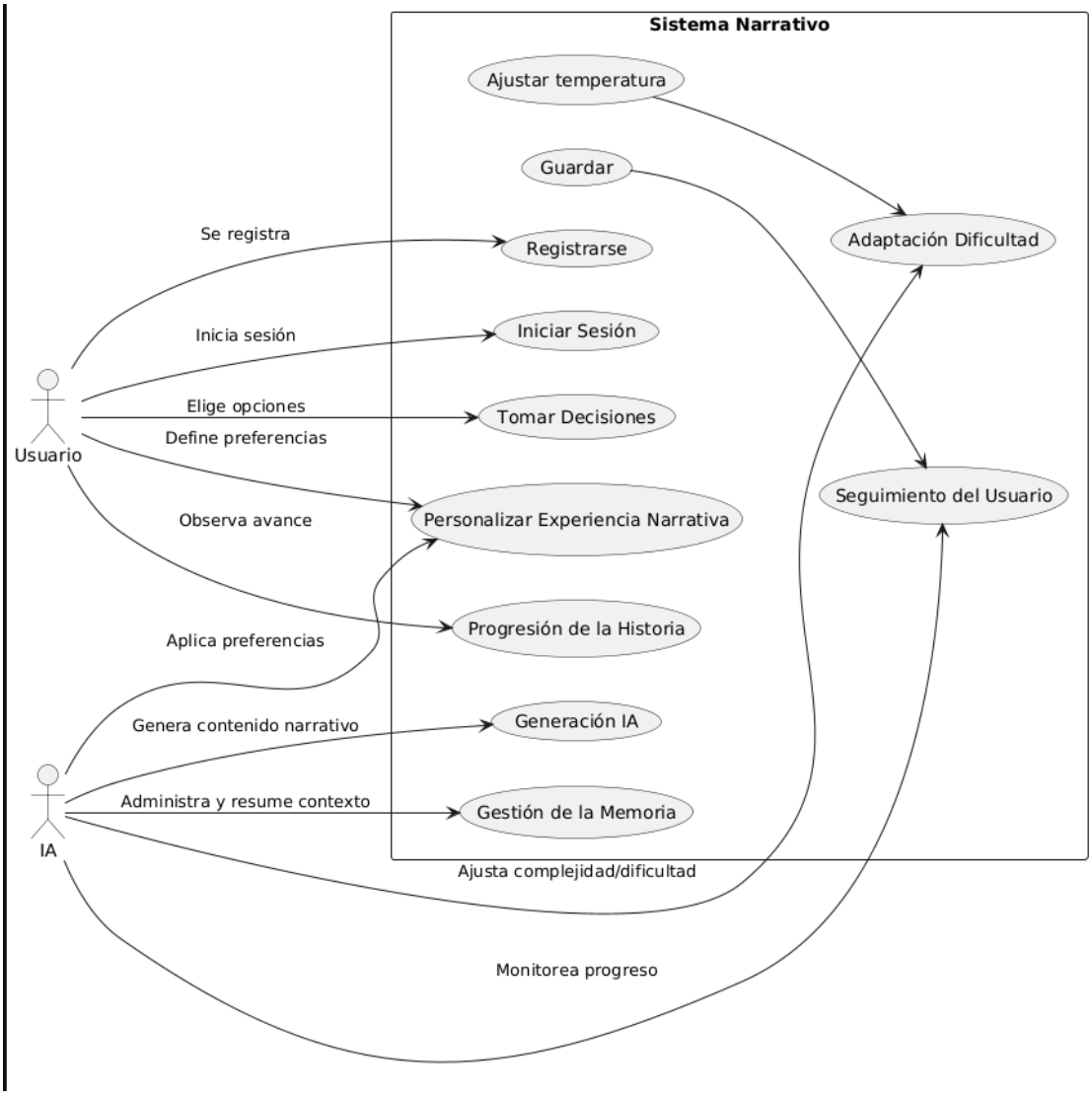


Figura 1 Diagrama de casos de uso

2.3.1 Iniciar Sesión

Descripción

El usuario debe poder acceder a la plataforma proporcionando sus credenciales de acceso (correo electrónico y contraseña). El sistema verificará la autenticidad de los datos y, en caso de éxito, permitirá el ingreso del usuario a la aplicación. El actor en este caso de uso es el usuario, pues este interactuara con la funcionalidad Iniciar sesión, para poder hacer uso de esta funcionalidad al usuario ya debe de estar registrado en la plataforma. Una vez hecho esto el flujo de esta funcionalidad se define de la siguiente manera, el usuario ya registrado presiona el botón, de iniciar sesión, esto desplegara un formulario en el cual ingresara los datos de email y contraseña ya registrados previamente, se validarán los datos del usuario en la base de datos y si coinciden el programa la dará acceso a la pagina en caso de ser incorrectos no podrá interactuar con la pagina de la manera que se espera.

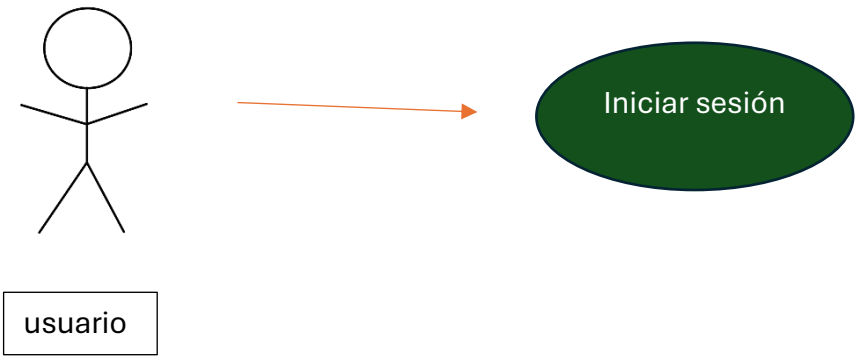


Figura 2. visualización del caso de uso iniciar

2.3.2 Tomar Decisiones

EL usuario interactuara con el programa tomando decisiones las cuales afectaran el progreso de la historia. El actor principal en esta funcionalidad es el usuario pues este interactuará directamente en este caso de uso una vez iniciada la historia al final de cada página habrá tres opciones de las cuales el usuario podrá decidir cual decisión tomar. EL flujo de esta funcionalidad se define de la siguiente manera. Se generarán tres opciones las cuales

interferirán en el desarrollo de la historia. De esta manera estas opciones estarán dispuestas al final de la página, el usuario debe decidir cual de estas opciones tomar, una vez el usuario elige una opción se marcará le decisión tomada y se mostrara en el lateral de la pagina para que el usuario sepa que decisión tomo y apartir de esa decisión tomada la historia debe continuar.

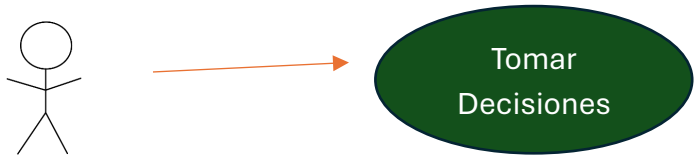


Figura 3. visualización del caso de uso Tomar decisiones

2.3.3 Generación I.A

La IA genera usuario narrativo dinámico basado en las decisiones del usuario y las condiciones predefinidas del sistema. EL Actor en esta funcionalidad es la IA pues esta generara el contenido dinámico coherente de acuerdo con la historia y a las decisiones del usuario. EL flujo de esta funcionalidad se define de la siguiente manera: La I.A generara una historia de acuerdo a las preferencias del usuario las cuales serán limitadas para poder llevar la coherencia de la historia y el dinamismo con el usuario, La IA generara un tema y un titulo del "cuento" De acuerdo a los estados de la historia, de los cuales la IA dispondrá para saber en que momento de la historia generara el texto, por ejemplo si la IA sabe que la historia esta en el inicio aperturara la historia si sabe que esta en el desarrollo en los etsados donde esto sucede generara la trama de acuerdo adonde se encuentre y así sucesivamente también dispondrá de las decisiones del usuario y de un contexto clave para poder generar la siguiente página.

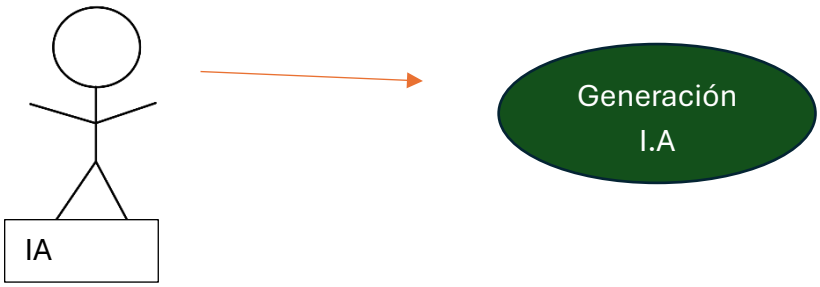


Figura 3. visualización del caso de uso Generación IA

2.3.4 Seguimiento usuario.

La IA debe de dar seguimiento al usuario para poder ir generando la historia de manera coherente. El actor en esta funcionalidad es la IA pues debe de darle seguimiento a las preferencias del usuario, a las decisiones que ha tomado, y a su historia para poder manejar y generar la historia de una manera coherente. EL flujo de esta funcionalidad se define de la siguiente manera, Se guardarán temporalmente las decisiones, preferencias usuario y contextos claves de la historia para que esta pueda darle un seguimiento a la historia y al usuario, esto ayudara a que la historia sea coherente y dinámica y se relacione mas con las preferencias del usuario.

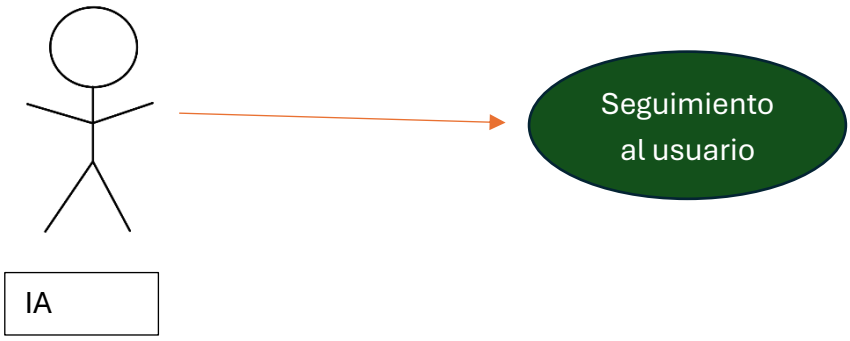


Figura 4. visualización del caso de uso Tomar decisiones

2.3.5 Adaptación de dificultad

La IA ajustara la dificultad de acuerdo al parámetro de temperatura y al momento de la historia en el que se encuentre el usuario. EL actor en esta funcionalidad es el IA ya que de acuerdo al momento de la historia, las decisiones que ha tomado el usuario y la temperatura, esta generara decisiones mas largas, y la historia puede ser mas ambigua sin decisiones tan concretas, EL flujo de esta funcionalidad seria el siguiente Una vez la IA hace el seguimiento al usuario, y se ajusta el estado de la historia y el parámetro de temperatura la IA hará mas o menos rebuscada la historia y las decisiones de acuerdo a todos los parámetro antes mencionado.

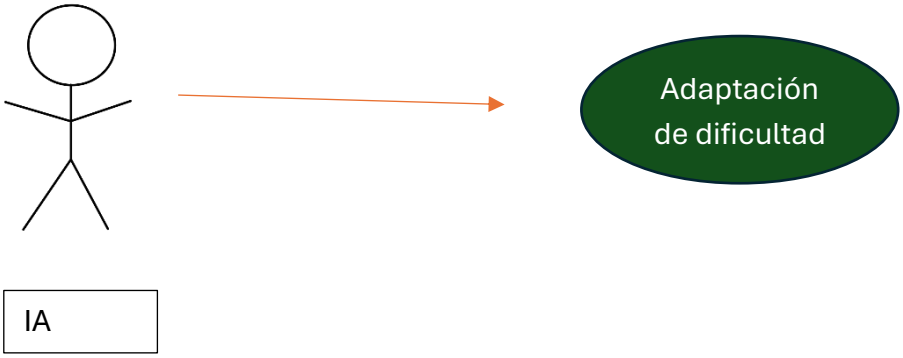


Figura 5. visualización del caso de uso adaptación de dificultad

2.3.6 Registrarse

El usuario debe registrase en la pagina para poder hacer uso de esta. El actor en esta funcionalidad es el usuario pues este proporcionara las credenciales solicitadas las cuales se almacenarán en la base de datos para poder hacer uso de la página. Además de que la IA podrá hacer un seguimiento del usuario para poder hacer los ajustes necesarios en la historia.

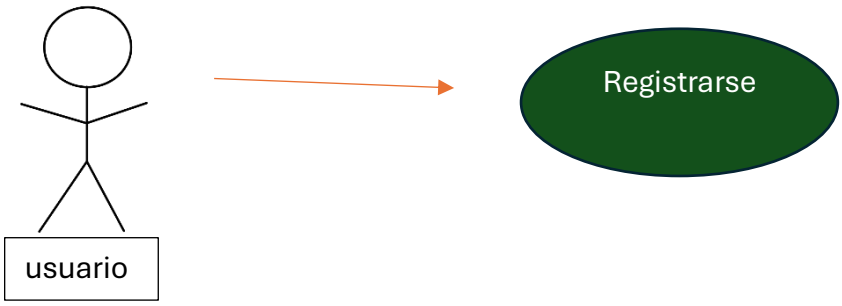


Figura 6. visualización del caso de uso iniciar

2.3.7 Personalizar historia

El usuario interactuara con esta funcionalidad pues se desplegara un pequeño de menú donde podrá elegir ciertas preferencias para su historia y la IA accederá a estas preferencias para poder hacer una historia mas personalizada y coherente. Los actores en esta funcionalidad son el usuario y la IA el usuario define las preferencias que quieres para la historia y la ia interactuara con estas preferencias para poder realizar una historia mucho más concisa.

Figura 7. visualización del caso de uso Personalizar

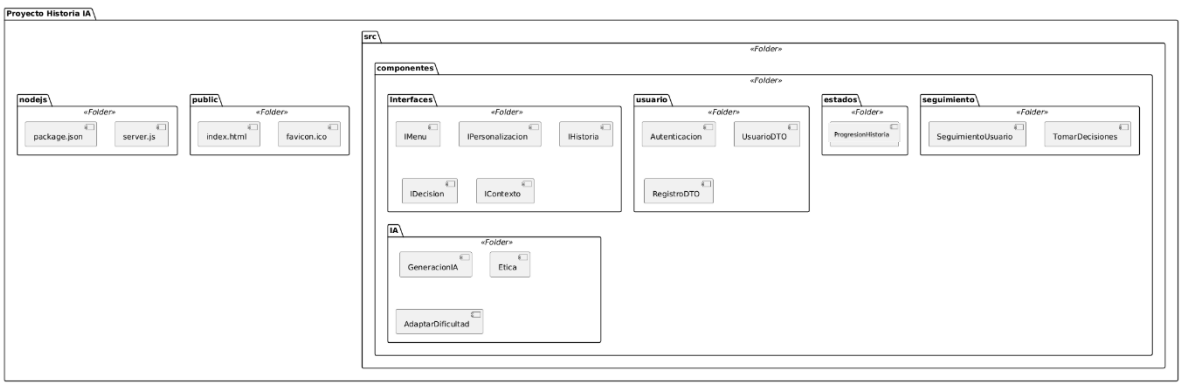
2.4 Composition viewpoint

Descripción del Diagrama de Paquetes

El diagrama de paquetes organiza el proyecto a nivel de carpetas, ilustrando la estructura de un repositorio React con un backend de Node.js simulado. En la raíz (“Proyecto Historia IA”) se ubican tres grandes carpetas: nodejs, que contiene package.json y server.js para el servidor local; public, con index.html y activos estáticos como el favicon; y src, donde reside todo el código React. Dentro de src/componentes cada subcarpeta agrupa clases y módulos por responsabilidad: Interfaces para los DTOs, usuario para autenticación y modelos de usuario, IA para la lógica de generación de historias (incluyendo ética, dificultad y progresión), dificultad y estados para sus componentes especializados, condiciones_eticas con la clase de validación ética y seguimiento que alberga el panel de historial y la gestión de decisiones. Esta disposición en paquetes facilita la navegación del equipo, refuerza la modularidad y hace evidente dónde debe vivir cada pieza de la aplicación.

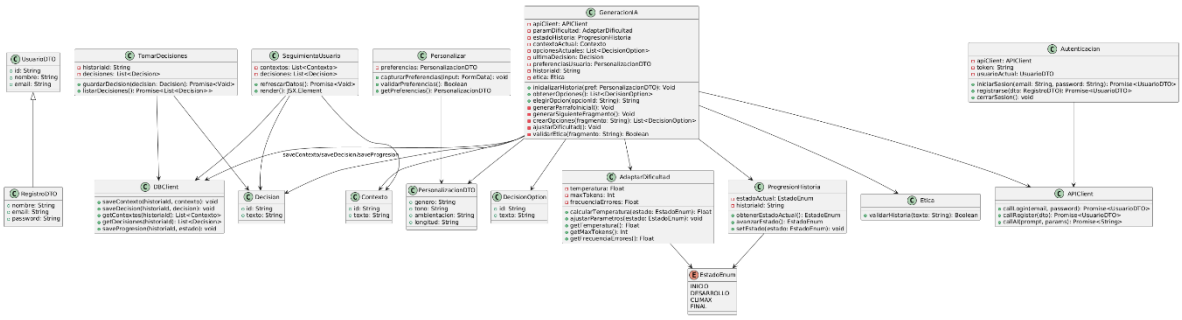
Estructura del Diagrama de Paquetes para este Proyecto

Figura 5.5 - Diagrama de paquetes



2.5 Logical viewpoint

2.5.1 Diagrama de clases



Descripción del Diagrama de Clases

El diagrama de clases modela la arquitectura interna de nuestro sistema de generación de historias con IA en React/Node.js, mostrando claramente la separación entre las entidades de dominio, la lógica de negocio y la capa de infraestructura. En la sección de “Clases de Dominio” se representan los DTOs (UsuarioDTO, RegistroDTO, PersonalizacionDTO, Contexto, Decision, DecisionOption) y el catálogo EstadoEnum, que definen la estructura de datos fundamental. A continuación, la sección “Infraestructura/Camadas” incorpora APIClient y DBClient, los responsables de la comunicación HTTP con el backend simulado y de la persistencia de contextos, decisiones y progresión de la historia.” Figuran los componentes principales: Autenticacion para el ciclo de usuario, Personalizar para capturar preferencias, AdaptarDificultad para ajustar parámetros de IA, ProgresionHistoria para aplicar el patrón State, TomarDecisiones para guardar y recuperar elecciones, Etica para validar contenidos y GeneracionIA como núcleo coordinador de generación, contexto, decisiones, dificultad y ética. Finalmente, SeguimientoUsuario cierra el modelo mostrando la vista de histórico contextual y de decisiones. Las flechas y dependencias entre ellos reflejan flujos claros de llamadas y relaciones de composición, asegurando un diseño modular y altamente cohesionado.

2.5.2 Diagrama de objetos

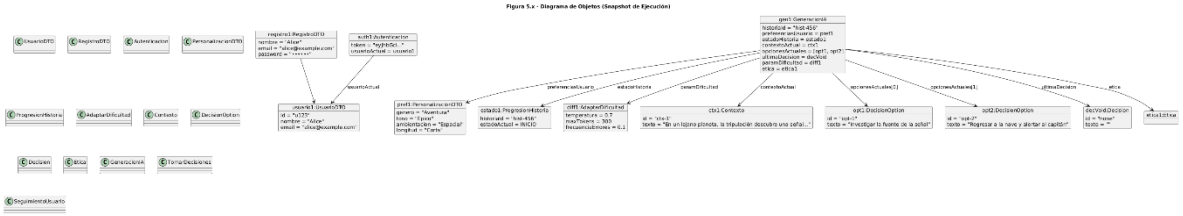
Descripción detallada

En esta instantánea, **UsuarioDTO** usuario1 (Alice) y **RegistroDTO** registro1 reflejan que la usuaria se creó en el sistema. El objeto **Autenticacion** auth1 retiene el JWT/token devuelto y apunta a usuario1 en su atributo usuarioActual.

A continuación, la usuaria definió una **PersonalizacionDTO** pref1 con género “Aventura”, tono “Épico”, ambientación “Espacial” y longitud “Corta”. Con esas preferencias, el sistema inicializa la **ProgresionHistoria** estado1 en INICIO, y crea un **AdaptarDificultad** diff1 con parámetros iniciales (temperatura 0.7, etc.).

La **GeneracionIA** gen1 combina todos esos objetos: lee pref1, su propio estado (estado1), y su parámetro de dificultad (diff1) para llamar al motor de texto. El resultado primera invocación es un **Contexto** ctx1 con el párrafo “En un lejano planeta…”. Inmediatamente tras el texto, la IA genera dos **DecisionOption** (opt1 y opt2) que se almacenan en la lista opcionesActuales. Al no haberse tomado todavía decisión, el atributo ultimaDecision apunta a un objeto vacío decVoid.

Por último, antes de mostrar cualquier contenido al usuario, **GeneracionIA** invoca a la clase **Etica** etica1 mediante su método validarHistoria(), asegurando que el fragmento ctx1.texto sea aceptable. Una vez validado, la interfaz de React podrá presentar el párrafo y las dos opciones para que Alice elija el siguiente paso.



2.5.3 Dependency viewpoint

2.6.1 Descripción del Diagrama de Paquetes

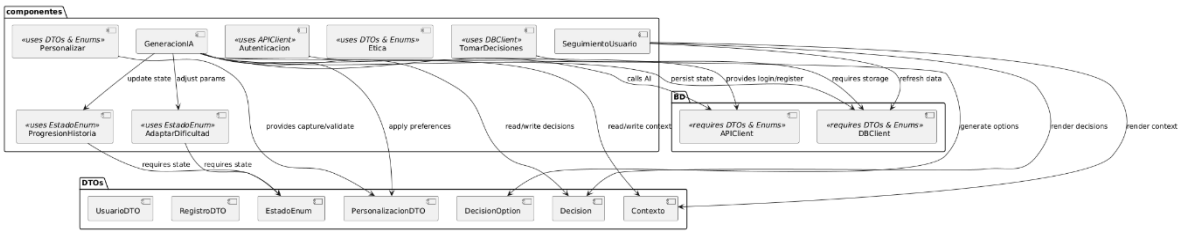
El Diagrama de Dependencias presenta tres grandes paquetes o subsistemas —**DTOs** y **componentes**— y muestra cómo cada componente dentro de estos grupos se relaciona, ordena y consume servicios de los demás.

Dentro de **DTOs** están los objetos de transferencia (UsuarioDTO, RegistroDTO, PersonalizacionDTO, Contexto, Decision, DecisionOption) y la enumeración EstadoEnum. Este paquete no "usa" activamente a nadie; más bien, "provee" estructuras de datos y constantes que el resto del sistema necesita para intercambiar información y gobernar el flujo narrativo.

El paquete **BD** contiene los componentes APIClient y DBClient, ambos estereotipados como "requires DTOs & Enums": dependen de las definiciones de DTO y de la enumeración de estados para serializar llamadas HTTP y para leer o persistir registros. APIClient ofrece los servicios de autenticación y de invocación a la IA, mientras que DBClient ofrece métodos de almacenamiento y consulta para contexto, decisiones y progreso narrativo.

En **Business Logic**, cada componente "usa" uno o más elementos de Infrastructure o de DTOs & Enums:

- **Autenticacion** consume APIClient para realizar login, registro y gestión de tokens.
- **Personalizar** y **Etica** dependen directamente de los DTOs para capturar y validar las preferencias y contenidos.
- **AdaptarDificultad** y **ProgresionHistoria** "requieren" el enum EstadoEnum para calcular parámetros de IA y avanzar el estado interno de la historia.
- **TomarDecisiones** invoca a DBClient para almacenar o recuperar las elecciones del usuario.
- **GeneracionIA**, núcleo del sistema, orquesta llamadas a APIClient y DBClient, consulta y actualiza el estado de la historia (ProgresionHistoria), lee y escribe contextos y decisiones, aplica preferencias (PersonalizacionDTO) y refina parámetros de generación dinámicamente usando AdaptarDificultad.
- Finalmente, **SeguimientoUsuario** vuelve a DBClient, Contexto y Decision para refrescar y renderizar la interfaz de seguimiento de la sesión.



2.5.4 Diagrama de componentes

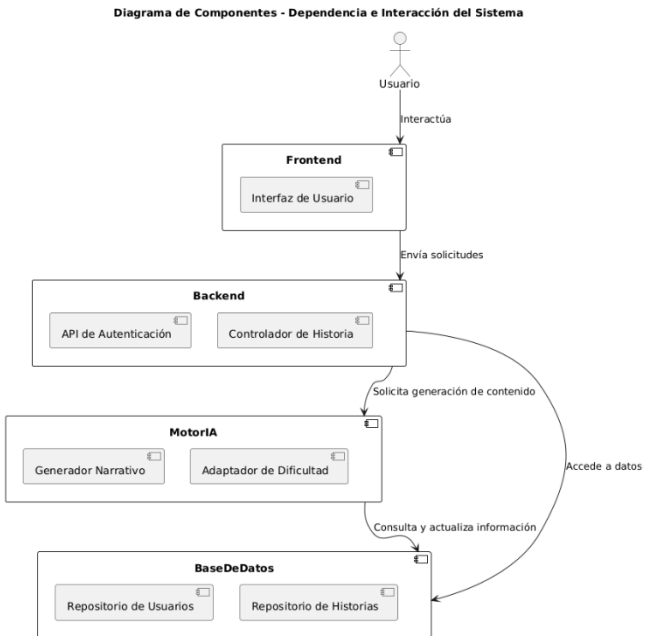
Descripción del Diagrama de Componentes

El diagrama de componentes muestra de manera detallada cómo se comunican y dependen los distintos módulos de software a nivel de componentes, ofreciendo una visión integral del flujo de información en el sistema.

En este diagrama, se identifica a un **actor Usuario** que interactúa directamente con el componente Frontend, el cual representa la interfaz de usuario. El Frontend se conecta con el componente Backend, responsable de la lógica central del sistema, que incluye servicios como la API de autenticación y el controlador encargado de manejar la historia interactiva. A su vez, el Backend se comunica con el MotorIA, que es el encargado de generar contenido narrativo dinámico y de ajustar la dificultad de la historia según el rendimiento del usuario.

La BaseDeDatos se integra en este esquema para almacenar la información tanto de los usuarios como del progreso y estado de las historias. Las relaciones establecidas, mediante flechas, indican que el Usuario inicia la interacción a través del Frontend, que a su vez envía solicitudes al Backend para que este coordine las operaciones necesarias con el MotorIA y la BaseDeDatos.

Este diagrama permite visualizar claramente la estructura de comunicación entre componentes, lo que resulta esencial para la planificación de la integración y el mantenimiento del sistema. Además, ayuda a identificar posibles cuellos de botella y dependencias críticas, facilitando la creación de casos de prueba de integración y la detección temprana de problemas. La modularidad mostrada en el diagrama de componentes favorece la escalabilidad y la reutilización de componentes, asegurando que cada parte del sistema se pueda actualizar o reemplazar de manera independiente sin afectar el funcionamiento global.



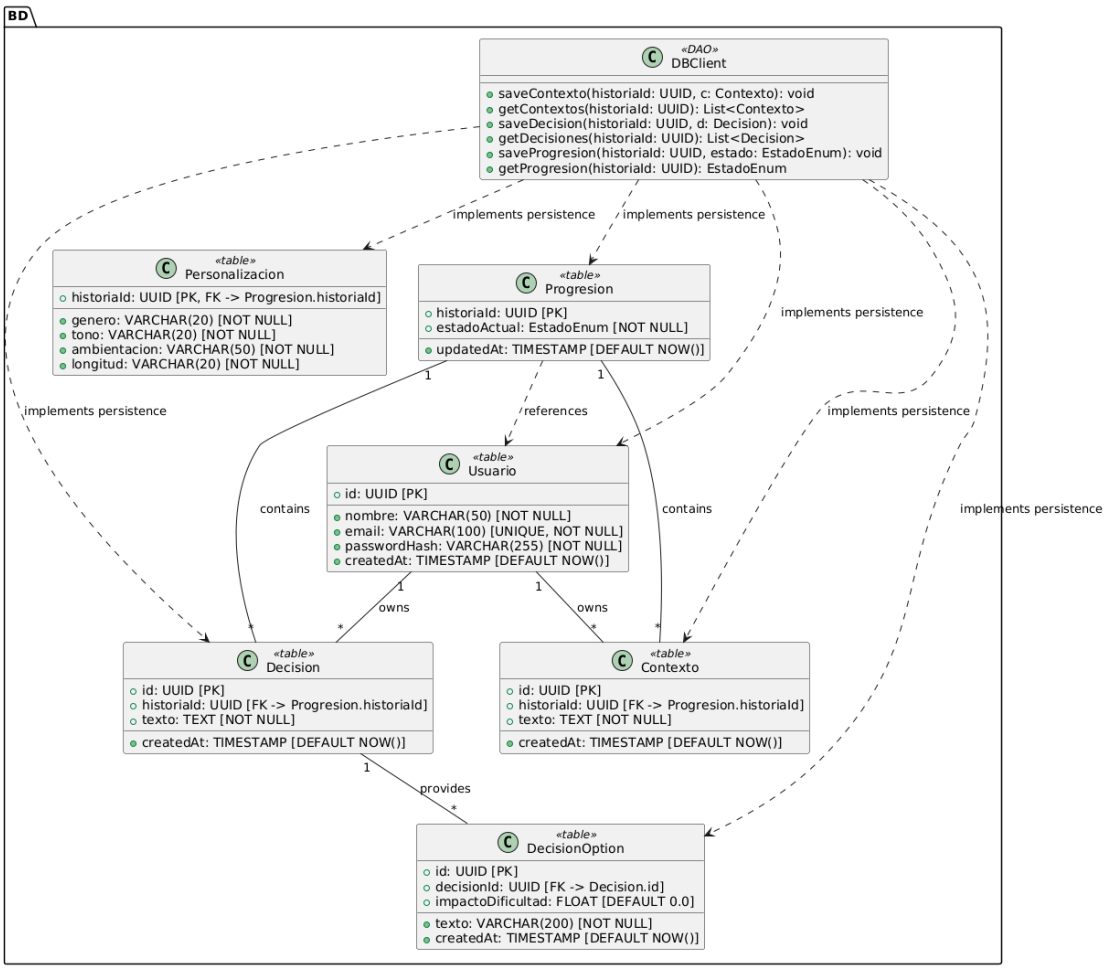
2.5.5 information viewpoint

El **Information Viewpoint** se emplea para documentar y analizar de forma exhaustiva toda la información permanente que el sistema debe manejar: su estructura, relaciones, restricciones y mecanismos de acceso. En esta vista se definen entidades de datos (tablas o clases persistentes), sus atributos con tipos y valores por defecto, las reglas de integridad

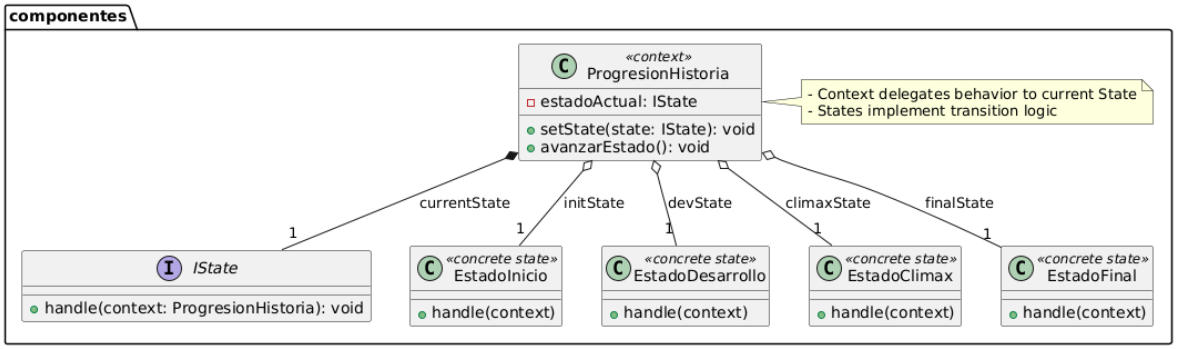
(claves primarias, foráneas, unicidad, no nulos), y la forma en que se leerán y escribirán esos datos (DAO, repositorios o clientes de base de datos). Al centrar el diseño en la gestión de datos, esta perspectiva ayuda a asegurar que las decisiones relativas a almacenamiento, rendimiento, consistencia y calidad queden explícitamente reflejadas desde etapas tempranas, facilitando la planificación de la base de datos, la generación de scripts de esquema y el desarrollo de pruebas de validación de datos.

En el diagrama generado, cada entidad persistente está representada como una tabla con sus columnas y metadatos:

- **Usuario** almacena identificación, nombre, correo, contraseña cifrada y fecha de creación.
- **Personalizacion** vincula una historia con las preferencias de género, tono, ambientación y longitud.
- **Contexto** y **Decision** guardan fragmentos de texto y elecciones pasadas, referenciando la historia a la que pertenecen y con sellos temporales.
- **DecisionOption** enumera las alternativas generadas para cada elección, indica su impacto en la dificultad y registra cuándo fueron creadas.
- **Progresion** mantiene el estado actual de la historia (INICIO, DESARROLLO, CLIMAX, FINAL) y la fecha de actualización.


BD
«DAO»
DBClient
+ saveContexto(historiaId: UUID, c: Contexto): void
+ getContextos(historiaId: UUID): List<Contexto>
+ saveDecision(historiaId: UUID, d: Decision): void
+ getDecisiones(historiaId: UUID): List<Decision>
+ saveProgresion(historiaId: UUID, estado: EstadoEnum): void
+ getProgresion(historiaId: UUID): EstadoEnum
implements persistence
implements persistence
implements persistence
implements persistence
implements persistence
implements persistence
«table»
Personalizacion
○ historiaId: UUID [PK, FK -> Progresion.historiaId]
+ genero: VARCHAR(20) [NOT NULL]
+ tono: VARCHAR(20) [NOT NULL]
+ ambientacion: VARCHAR(50) [NOT NULL]
+ longitud: VARCHAR(20) [NOT NULL]
«table»
Progresion
○ historiaId: UUID [PK]
○ estadoActual: EstadoEnum [NOT NULL]
+ updatedAt: TIMESTAMP [DEFAULT NOW()]
1
1
references
contains
contains
«table»
Usuario
○ id: UUID [PK]
+ nombre: VARCHAR(50) [NOT NULL]
+ email: VARCHAR(100) [UNIQUE, NOT NULL]
+ passwordHash: VARCHAR(255) [NOT NULL]
+ createdAt: TIMESTAMP [DEFAULT NOW()]
1
1
owns
owns
*
*
*
*
«table»
Decision
○ id: UUID [PK]
○ historiaId: UUID [FK -> Progresion.historiaId]
○ texto: TEXT [NOT NULL]
+ createdAt: TIMESTAMP [DEFAULT NOW()]
«table»
Contexto
○ id: UUID [PK]
○ historiaId: UUID [FK -> Progresion.historiaId]
○ texto: TEXT [NOT NULL]
+ createdAt: TIMESTAMP [DEFAULT NOW()]
1
provides
*
«table»
DecisionOption
○ id: UUID [PK]
○ decisionId: UUID [FK -> Decision.id]
○ impactoDificultad: FLOAT [DEFAULT 0.0]
+ texto: VARCHAR(200) [NOT NULL]
+ createdAt: TIMESTAMP [DEFAULT NOW()]

2.5.6 Patterns viewpoint



El Patterns Use Viewpoint se emplea para capturar y documentar las “ideas de diseño” —es decir, los patrones de colaboración entre roles y conectores que surgen en la arquitectura— sin perder de vista las clases y componentes concretos. En esta perspectiva se identifica el patrón, se abstraen sus roles (por ejemplo, Context y State), y se muestran las relaciones de colaboración y los conectores que permiten reutilizar soluciones probadas a problemas comunes.

En el diagrama aplicado al manejo de estados de la narración, se ha desplegado el State Pattern de la siguiente manera:

Contexto (ProgresionHistoria) actúa como el objeto principal que mantiene una referencia compuesta (currentState) a una interfaz abstracta IState.

Interfaz IState define el contrato handle(context) que encapsula la acción a ejecutar cuando se solicita avanzar de estado.

Estados concretos (EstadoInicio, EstadoDesarrollo, EstadoClimax, EstadoFinal) implementan IState y contienen la lógica específica de transición y manipulación del contexto narrativo.

A través de agregaciones etiquetadas (initState, devState, climaxState, finalState), ProgresionHistoria posee instancias de cada estado listo para asignarse según convenga.

Al invocar avanzarEstado(), ProgresionHistoria delega inmediatamente ese comportamiento al objeto currentState, permitiendo que cada clase de estado determine su sucesor o acción asociada.

De este modo, la vista de patrones muestra claramente cómo el sistema reutiliza el State Pattern para aislar las variaciones de comportamiento según el estado de la historia, facilita la extensión (añadir nuevos estados sin alterar ProgresionHistoria) y mejora la claridad de la colaboración entre las clases implicadas.

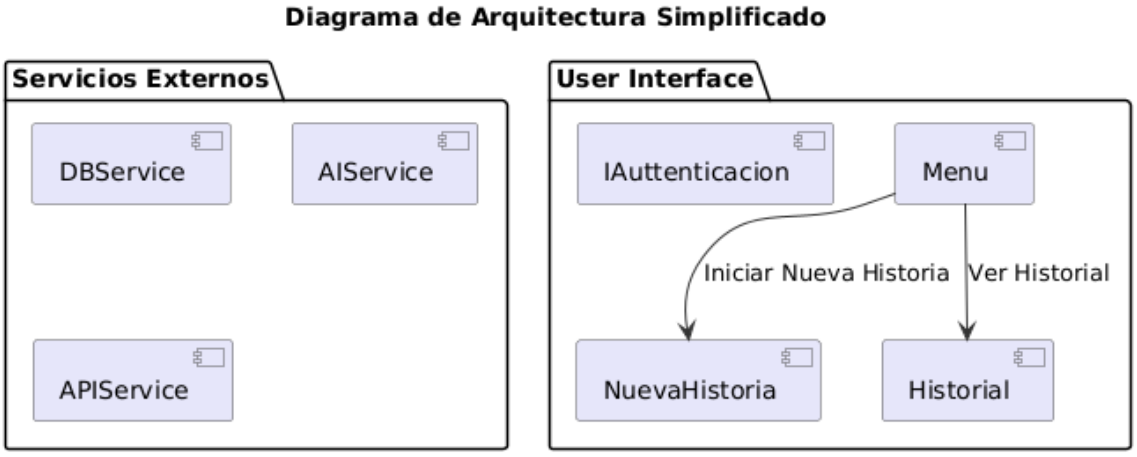
2.5.6 Interfaces

La interfaz de usuario de **HistoriaInteractiva** se articula en cuatro componentes principales, cada uno implementando su propio contrato (interfaz) y conectándose con servicios backend estandarizados. Cuando el usuario accede por primera vez, el **Autenticacion** despliega los campos de nombre de usuario y contraseña y un botón de autenticación que, al pulsarse, invoca el método iniciarSesion de la interfaz IAutenticacion. Tras validar las credenciales, el componente lanza un evento de éxito o error que desencadena la transición al Menú Principal (o la presentación de un mensaje de fallo). Este flujo garantiza que solo usuarios verificados puedan avanzar y establece la sesión con un token que otros componentes pueden reutilizar para llamadas protegidas.

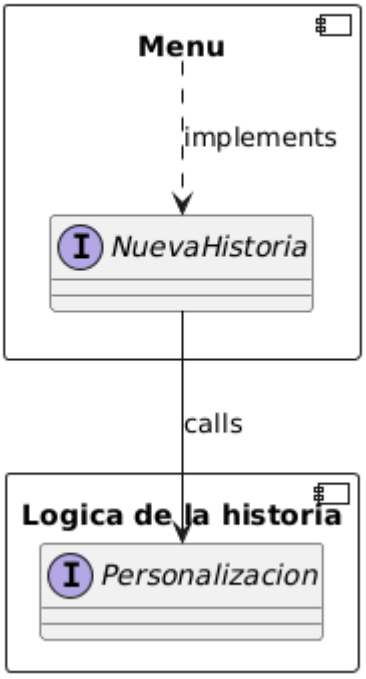
Una vez autenticado, el **MenuC**, responsable de la interfaz IMenu, ofrece dos acciones: “Nueva Historia ” “Historial” “Cerrar Sesión”. La primera envía al usuario a la pantalla de narrativa, mientras que la segunda invoca de nuevo a IAutenticacion para invalidar el token y regresar al login. Este menú actúa como punto de control, permitiendo elegir entre reanudar

o comenzar una historia nueva y asegurando que el cierre de sesión limpie cualquier estado residual.

NuevaHistoria concentra la experiencia central. A la izquierda muestra, mediante , la fase actual de la narrativa (Inicio, Desarrollo, Clímax o Final), obtenida por el componente En el panel central aparece el texto íntegro de la página generada, cargado y actualizado desde IProgresionHistoria y IGeneracionIA. Justo debajo, una lista de botones de decisión permite

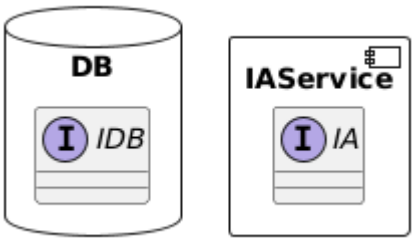


Detalle de Interface INuevaHistoria



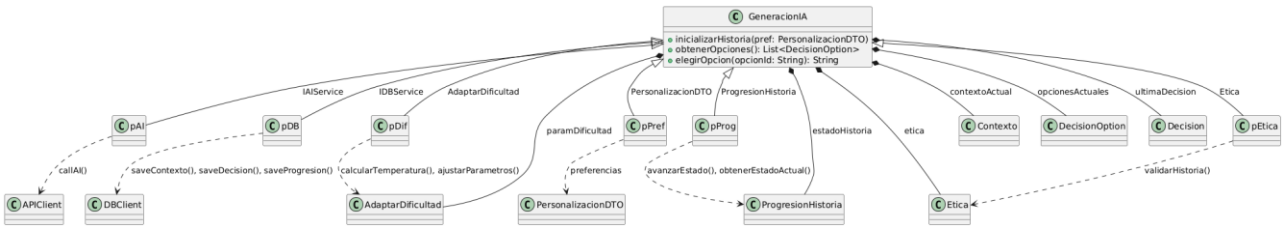
En este diagrama agrupamos todas las interfaces de la UI dentro del paquete src/componentes/Interfaces. Cada interfaz vive en su propia carpeta virtual dentro de “Interfaces”, lo que facilita encontrarla y mantener la coherencia de nombres. Fuera de ese paquete, en src/componentes, tenemos carpetas dedicadas a la lógica: usuario contiene la clase que implementa IAutenticacion, historia agrupa el motor de generación IA, Idecisiones almacena la gestión de elecciones y contexto maneja el historial narrativo solo para mostrarun historial de decisiones del usuario en el transcurso de su historia. De este modo la capa de presentación (Interfacesquedan claramente separadas, promoviendo la modularidad y la mantenibilidad del proyecto.

Tambien se cuenta con interfaces externas como la de la ia la cual se usara mediante un token y la de la bse de datos



2.5.7.2Structure viwpoint

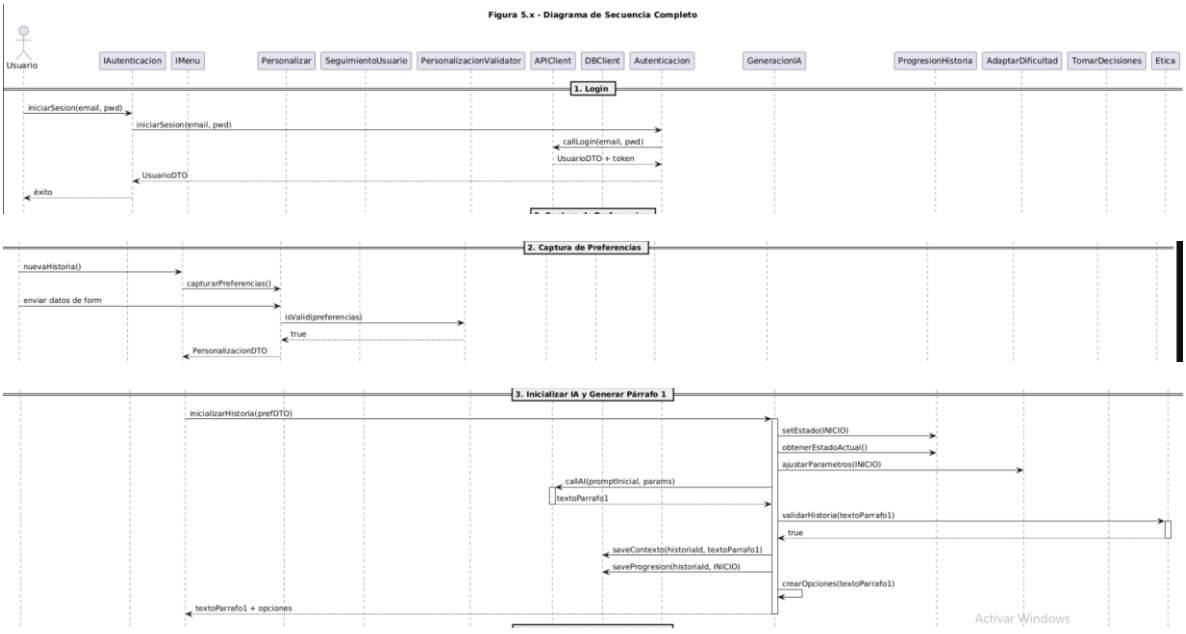
El **Structure Viewpoint** se aplica para descomponer un componente complejo —en este caso GeneracionIA— en sus partes constituyentes, puertos e interfaces internas, de modo que quede claro cómo se organiza y relaciona cada elemento dentro de la clase. En el diagrama Composite Structure, GeneracionIA se presenta como un contenedor con **puertos** (pAI, pDB, pDif, pProg, pEtica, pPref) que representan las interfaces necesarias para la generación de texto (IAIService), la persistencia de datos (IDBService), el cálculo de dificultad, el avance de estado y la validación ética, así como el acceso a las preferencias del usuario. Cada uno de estos puertos se **conecta** a la implementación concreta (por ejemplo, APIClient, DBClient, AdaptarDificultad, ProgresionHistoria, Etica, PersonalizacionDTO) a través de **conectores**, que indican qué métodos se invocan en tiempo de ejecución (como callAI(), saveContexto(), avanzarEstado() o validarHistoria()). Al mismo tiempo, las **partes internas** —atributos compuestos como paramDificultad, estadoHistoria, contextoActual u opcionesActuales— muestran cómo GeneracionIA mantiene su propio estado y lógica de negocio mediante composición. En conjunto, este diagrama deja patente la organización interna de GeneracionIA, la forma en que sus subcomponentes y servicios colaboran, y cómo se gestionan tanto la reutilización como las dependencias para garantizar un diseño modular y cohesionado.

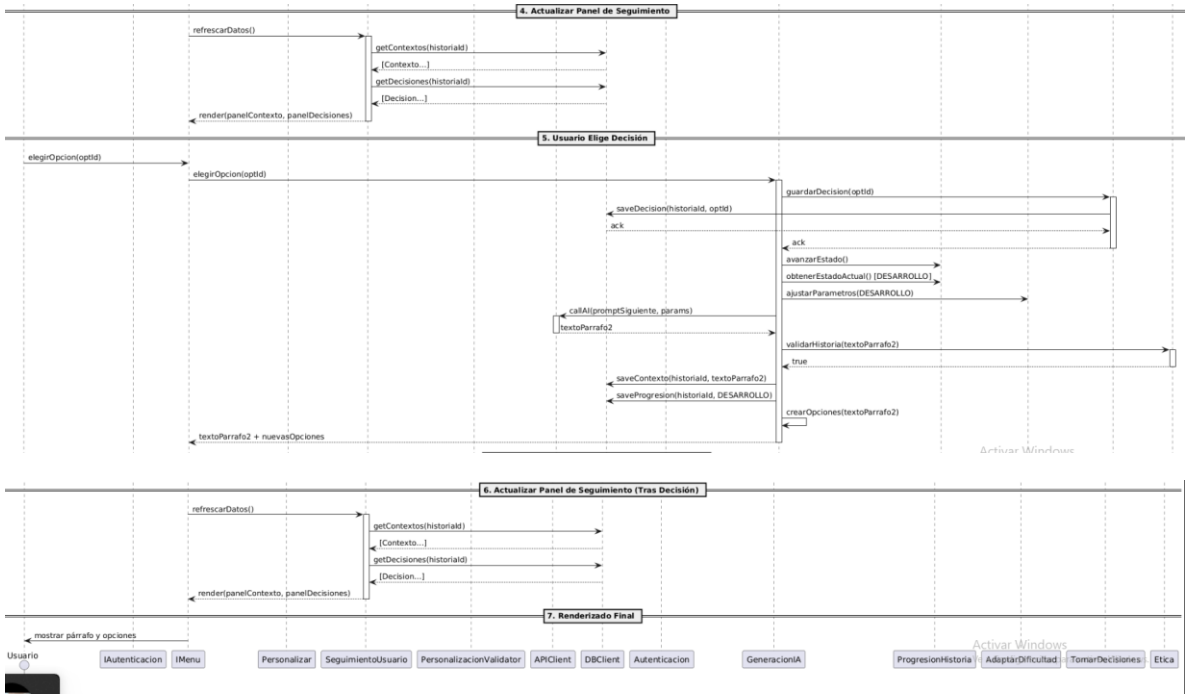


2.5.8 interaction viewpoint

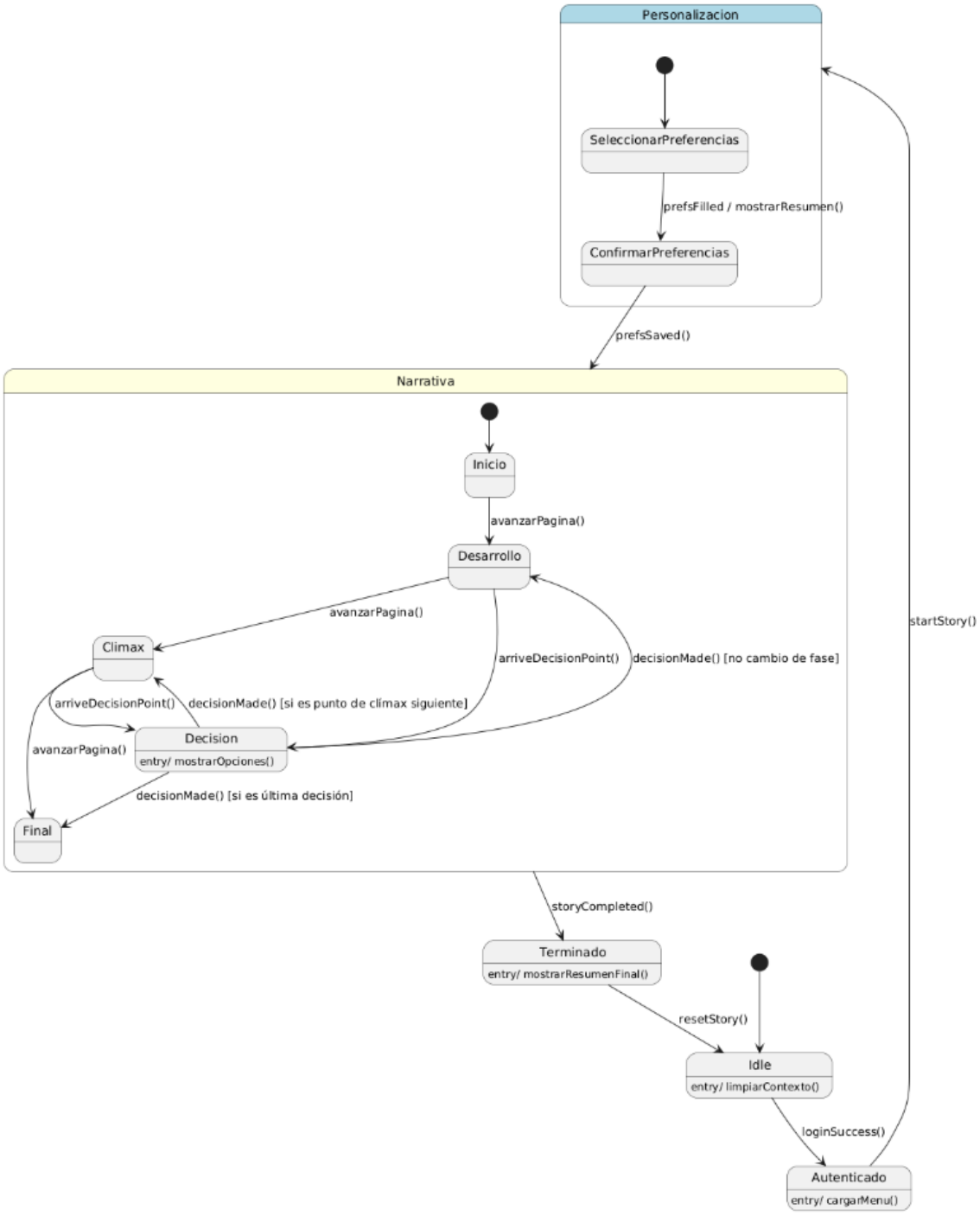
El diagrama de secuencia expone cómo **Usuario** inicia sesión a través de la interfaz empezando por la interfa de autenticación la interfaz de autenticación hace uso de la clase autenticación que la que maneja la lógica ya sea para que el usuario se registre o pueda iniciar sesión, el sistema comprueba las credenciales en la base de datos del usuario con usuarioDTO y una vez el usuario ingrese se mostrara el menú ahí el usuario podrá seleccionar

nueva historia esta opción te llevara a la personalización que se bada en un formulario donde el usuario puede detallar o poner ciertas condiciones para que se genere su historia al enviar datosse valida las preferencias asio como se personalizacioDTO guarda los datos en la base de datos una vez pasamos esta parte se genera una pantalla donde se mostrara la historia que generara la IA mediante la funcionInicializarIA una vez iniciada la historia esta selecciona el estado con set estado y después lo obtiene para generar la historia de acuerdo al estado con el que nos encontramos también interactúa con la clase adaptar dificultad para ajustar que se ajusten los parámetros de la temperatura de la dificultad ya sea aumentando o disminuyendo una vez hecho esto y generado la historia db cleint guarda el contexto y la progresión para que la ia pueda hacer uso de esto en la siguiente iteración una vez se guardan la ia crea opciones las cuales el usuario podrá elegir para interactuar con la historia ahora seguimos con el seguimiento al usuario en esta parte de reciben los saves de contexto y de decisiones del usuario para que este se muestre a lado así como tener un historial de usuario y depsue spueda visualizarlo, Una vez el usuario toma una decisión esta se guarda en DBclient se avanza el estado y la IA lo vuelve a obtener y en la temperatura se ajustan los parámetros de dificultad para que esta los pueda obtener y modificar la complejidad de la historia o las decisiones nuevamente se valida la historia, se genera y así hasta que lelgue al final .




4. Actualizar Panel de Seguimiento
refrescarDatos()
getContextos(historiaId)
[Contexto...]
getDecisiones(historiaId)
[Decision...]
render(panelContexto, panelDecisiones)
5. Usuario Elige Decisión
elegirOpcion(optId)
elegirOpcion(optId)
guardarDecision(optId)
saveDecision(historiaId, optId)
ack
ack
avanzarEstado()
obtenerEstadoActual() [DESARROLLO]
ajustarParametros(DESARROLLO)
callAI(promptSiguiente, params)
textoParrafo2
validarHistoria(textoParrafo2)
true
saveContexto(historiaId, textoParrafo2)
saveProgresion(historiaId, DESARROLLO)
crearOpciones(textoParrafo2)
textoParrafo2 + nuevasOpciones
6. Actualizar Panel de Seguimiento (Tras Decisión)
refrescarDatos()
getContextos(historiaId)
[Contexto...]
getDecisiones(historiaId)
[Decision...]
render(panelContexto, panelDecisiones)
7. Renderizado Final
mostrar párrafo y opciones
Usuario
IAutenticacion
IMenu
Personalizar
SeguimientoUsuario
PersonalizacionValidator
APIClient
DBClient
Autenticacion
GeneracionIA
ProgresionHistoria
AdaptarDificultad
TomarDecisiones
Etica

2.5.7 state dinamics



Aquí se muestra como es lel dinamimos de los estados siendo una iteración una tras otras en la parte de la narrativa en la cuallos estados van cambiando después de las decisiones que toma el usuario desde el inicio pasando por el desarrollo climax y final una vez se llega al final y hay una conclusión la historia se da por terminada se limpia los contexto actuales para no generar información basura y volvemos la pantalla de menu

2.5.8 **algorithm viewpoint**

Cuando el usuario abre la aplicación, se muestra la pantalla de login. Al enviar credenciales válidas se dispara el evento loginSuccess() y la máquina transita a **Autenticado**, cargando el menú principal. Desde allí, al seleccionar "Nueva Historia" se activa IniciarHistoria() y entramos en el estado compuesto **Personalización**, que primero lleva al subestado **SeleccionarPreferencias** y luego, una vez completado el formulario, a **ConfirmarPreferencias**, donde se resumen las elecciones del usuario (técnicamente la acción mostrarResumen()). Cuando el usuario confirma (prefsSaved()), la narrativa efectivamente comienza al entrar en el subestado inicial de la submáquina **Narrativa**.

Dentro de **Narrativa** la historia progresa de forma lineal: primero **Inicio**, luego **Desarrollo**, **Clímax** y finalmente **Final**, avanzando con llamadas internas como . Sin embargo, cada vez que la lectura alcanza un "punto de decisión" se entra en el estado **Decision**, que detiene momentáneamente el flujo para presentar al usuario las opciones disponibles (entry/ mostrarOpciones()). Al elegir una opción (decisionMade()), si no cambia la fase, se regresa a **Desarrollo**; en otros casos podrá transicionar directamente a **Clímax** o **Final** según la encrucijada de la trama. Concluida la última fase y disparado storyCompleted(), la máquina pasa a **Terminado**, donde se invoca mostrarResumenFinal() para que el usuario revise el desenlace, y finalmente, al activar resetStory(), se regresa al estado **Idle** para poder iniciar una nueva experiencia desde cero.

El **Algorithm Viewpoint** para nuestra aplicación de historia interactiva con IA se centra en proporcionar a los desarrolladores una visión detallada de la lógica interna y los algoritmos que sustentan cada entidad de diseño, de modo que puedan implementarse de forma eficiente, fiable y con pruebas unitarias bien fundamentadas. En este enfoque describimos primero las principales preocupaciones de diseño, luego detallamos los elementos de diseño con sus atributos clave y, finalmente, presentamos con detalle el funcionamiento interno de los métodos críticos, incluyendo su control de contingencias y consideraciones de rendimiento.

En cuanto a las preocupaciones de diseño, es fundamental entender que la generación de la historia depende de una secuencia precisa de eventos: primero se recoge la personalización

del usuario, a continuación se construye un prompt concatenando la base textual y el historial de contexto, y finalmente se envía este prompt a un servicio externo de IA. Cada una de estas etapas tiene requisitos de datos específicos —por ejemplo, el historial debe estar siempre inicializado antes de solicitar el primer fragmento— y debe respetar restricciones de tiempo, como el uso de un mecanismo de reintento con un tiempo máximo de espera de treinta segundos para cada llamada a la API. Además, las decisiones del usuario se registran inmediatamente en la base de datos bajo transacciones atómicas, y cualquier error en la inserción desencadena una reversión (rollback) y una notificación apropiada para garantizar la consistencia. La prioridad de ejecución recae en primero obtener el fragmento de historia, luego procesar la decisión del usuario, y finalmente ajustar la dificultad antes de preparar el siguiente prompt.

Los elementos de diseño más relevantes incluyen la clase **GeneracionIA**, que mantiene atributos como promptBase (texto inicial de la historia), historialContexto (lista ordenada de fragmentos y decisiones) y apiKey para autenticación con el servicio de IA. La clase **AdaptarDificultad** almacena su nivel actual y umbrales de referencia para determinar cómo debe variar la complejidad narrativa. Por su parte, **ProgresionHistoria** controla el estado actual de la narración (inicio, desarrollo, clímax o final), **TomarDecisiones** gestiona la validación y el guardado en base de datos de cada elección del usuario, y **SeguimientoUsuario** acumula el contexto y las decisiones para ofrecer coherencia en cada llamada a la IA.

Profundizando en los atributos de procesamiento, el método generarHistoria() de **GeneracionIA** comienza verificando que historialContexto contenga al menos una entrada y, a continuación, concatena promptBase con una serialización de todo el historial. A continuación, envía este prompt al servicio externo y espera la respuesta, utilizando un bucle de reintentos de hasta tres intentos en caso de error. Si tras el tercer intento no se obtiene un resultado válido, lanza una excepción para interrumpir el flujo y notificar la incidencia. Una vez recibido el texto, lo añade al historial con su marca temporal y lo devuelve para su renderizado. La complejidad temporal de este método escala linealmente con el tamaño del historial, ya que cada llamada requiere volver a serializar todo el contexto.

El ajuste de dificultad, implementado en **AdaptarDificultad**, se basa en una combinación del nivel de historia (inicio, medio o final) y el porcentaje de decisiones acertadas por el usuario. Por ejemplo, si el usuario se encuentra en las etapas iniciales y mantiene más de un 80 % de aciertos, la temperatura se reduce a 0.7 para generar opciones más sencillas; si baja de ese umbral, la temperatura aumenta a 0.9. En las etapas intermedias, estas cifras oscilarán entre 0.5 y 0.8, y al llegar al final la temperatura se fija en 0.3 para ofrecer retos épicos. Este proceso se motorizza mediante una tabla de decisión interna que se consulta en cada iteración.

En lo que respecta a **TomarDecisiones**, el método procesarDecision(decision) valida que el identificador y las opciones recibidas sean válidos, abre una transacción en la base de datos para insertar el registro de decisión junto con la información de usuario y de historia, y en caso de fallo realiza un rollback antes de propagar el error. Luego invoca SeguimientoUsuario.registrarContexto() para incluir esta decisión en el historial de contexto, garantizando que la siguiente llamada a la IA disponga de toda la información pertinente.

Para facilitar la implementación y la planificación de pruebas, se han definido ejemplos de pseudo-código y diagramas Warnier. El pseudo-código de generarHistoria() recoge la secuencia completa de inicialización de promt, envío, reintentos y actualización de historial, mientras que el diagrama Warnier para AdaptarDificultad describe de forma esquemática las condiciones de niveles y porcentajes de acierto que determinan la temperatura. Estos artefactos servirán tanto para guiar al programador en la traducción a código real como para estructurar los casos de prueba de cada flujo de decisión y contingencia.

Aquí se mostrara el código de GeneracionIA para ver el funcionamiento de esta

```
class GeneracionIA {
  // Propiedades (atributos) privadas simuladas con "_"
  _apiClient;
  _paramDificultad;
  _estadoHistoria;
  _contextoActual;
  _opcionesActuales;
  _ultimaDecision;
  _preferenciasUsuario;
  _historiaId;
  _etica;
  _historia;

  constructor(apiKey) {
    this._apiClient = apiKey;  // En tu código actual, esto es solo la key string
    this._paramDificultad = null; // Por implementar: AdaptarDificultad
```

```
    this._estadoHistoria = null;  // Por implementar: ProgresionHistoria
    this._contextoActual = null;   // Por implementar: Contexto
    this._opcionesActuales = [];
    this._ultimaDecision = null;   // Por implementar: Decision
    this._preferenciasUsuario = null; // PersonalizacionDTO
    this._historiaId = null;       // String
    this._etica = null;            // Por implementar: Etica
    this._historia = "";
  }

  async inicializarHistoria(preferencias) {
    this._preferenciasUsuario = preferencias;
    this._historiaId = this._generarIdHistoria(); // Ejemplo, método privado para ID
    await this._generarParrafoInicial();
    this._opcionesActuales = await this._crearOpciones(this._historia);
  }

  obtenerOpciones() {
    return this._opcionesActuales;
  }

  async elegirOpcion(opcionId) {
    // Buscamos el texto de la opción seleccionada
    const opcionSeleccionada = this._opcionesActuales.find(op => op.id ===
opcionId);
    if (!opcionSeleccionada) throw new Error("Opción inválida");
```

```
    this._ultimaDecision = opcionSeleccionada;
    await this._generarSiguienteFragmento(opcionSeleccionada.texto);
    this._opcionesActuales = await this._crearOpciones(this._historia);

    return this._historia;
  }

  async _generarParrafoInicial() {
    const prompt = this._crearPromptInicial(this._preferenciasUsuario);
    this._historia = await this._llamarApi(prompt);
  }

  async _generarSiguienteFragmento(textoDecision) {
    const prompt = this._historia + "\n\nContinuar historia considerando: " +
textoDecision;
    const siguienteFragmento = await this._llamarApi(prompt);

    if (this._validarEtica(siguienteFragmento)) {
      this._historia += "\n\n" + siguienteFragmento;
    } else {
      this._historia += "\n\n[Fragmento eliminado por contenido no ético]";
    }
  }

  async _crearOpciones(fragmento) {
```

```
    const promptOpciones = `
Dada la siguiente historia:
"${fragmento}"

Sugiere 3 opciones para continuar la historia. Devuelve un arreglo JSON con objetos que tengan "id" y "texto", ejemplo:

[
  { "id": "1", "texto": "Opción uno" },
  { "id": "2", "texto": "Opción dos" },
  { "id": "3", "texto": "Opción tres" }
]

Solo devuelve el JSON, nada más.
`;

    const respuesta = await this._llamarApi(promptOpciones);

    try {
      const opciones = JSON.parse(respuesta);
      if (!Array.isArray(opciones) || opciones.length === 0) return [];
      return opciones;
    } catch {
      return [];
    }
  }
```

```
  _crearPromptInicial(preferencias) {
    return `Genera una historia de género ${preferencias.genero}, con tono
${preferencias.tono}, ambientación ${preferencias.ambientacion} y longitud
${preferencias.longitud}. Comienza la historia.`;
  }

  _validarEtica(fragmento) {
    // Por implementar: validar contenido ético del texto
    return true;
  }

  _ajustarDificultad() {
    // Por implementar: lógica para adaptar la dificultad
  }

  _generarIdHistoria() {
    // Ejemplo simple de generación de ID único
    return "hist-" + Date.now();
  }

  async _llamarApi(prompt) {
    const response = await fetch("https://api.cohere.ai/v1/generate", {
      method: "POST",
      headers: {
        "Authorization": `Bearer ${this._apiClient}`,
        "Content-Type": "application/json",
```

```
        "Cohere-Version": "2022-12-06"
      },
      body: JSON.stringify({
        model: "command",
        prompt: prompt,
        max_tokens: 300,
        temperature: 0.9
      })
    });

    if (!response.ok) {
      const errorData = await response.json().catch(() => null);
      const errorMsg = errorData?.message || response.statusText;
      throw new Error("Error en API: " + errorMsg);
    }

    const data = await response.json();
    return data.generations[0].text.trim();
  }
}

// Código para la página (igual que antes, solo nombres actualizados)
document.addEventListener("DOMContentLoaded", async () => {
  const historiaDiv = document.getElementById("historia");
  const opcionesDiv = document.getElementById("opciones");
```

```
  const preferenciasJSON = localStorage.getItem("preferenciasUsuario");
  if (!preferenciasJSON) {
    historiaDiv.textContent = "No se encontraron preferencias. Regresa a la página de personalización.";
    return;
  }

  const preferencias = JSON.parse(preferenciasJSON);

  const apiKey = "3mC5a9lysvBHTkYqHTLWLiExLgQFVLK5qShlX9V8";

  const generador = new GeneracionIA(apiKey);

  historiaDiv.textContent = "Generando historia...";
  try {
    await generador.inicializarHistoria(preferencias);
    historiaDiv.textContent = generador._historia;

    const mostrarOpciones = () => {
      opcionesDiv.innerHTML = "";
      const opciones = generador.obtenerOpciones();
      if (opciones.length === 0) {
        opcionesDiv.textContent = "No hay opciones disponibles para continuar la historia.";
        return;
      }
      opciones.forEach(opcion => {
```

```
        const btn = document.createElement("button");
        btn.textContent = opcion.texto;
        btn.onclick = async () => {
          btn.disabled = true;
          historiaDiv.textContent += "\n\n>>> Elegiste: " + opcion.texto +
"\n\nGenerando continuación...";
          try {
            await generador.elegirOpcion(opcion.id);
            historiaDiv.textContent = generador._historia;
            mostrarOpciones();
          } catch (e) {
            alert("Error generando la historia: " + e.message);
          }
        };
        opcionesDiv.appendChild(btn);
      });
    };

    mostrarOpciones();
  } catch (e) {
    historiaDiv.textContent = "Error generando la historia: " + e.message;
  }
});
```

Conclusión

El diseño descrito a lo largo de este documento SDD refleja una arquitectura sólida y modular orientada a ofrecer una experiencia narrativa interactiva, personalizada y coherente para el usuario. A través de una secuencia bien definida de autenticación, personalización de preferencias, generación de contenido narrativo mediante IA, seguimiento de decisiones y adaptación dinámica del contexto, el sistema demuestra ser capaz de ofrecer historias que evolucionan de manera significativa con base en las decisiones del usuario.

El **diagrama de secuencia completo** permite visualizar con precisión el comportamiento y la colaboración entre componentes clave, como la interfaz de usuario, los servicios de lógica, validadores, módulos de dificultad, ética y persistencia de datos. Esta representación no solo proporciona una guía clara para el desarrollo e implementación del sistema, sino que también asegura la trazabilidad del comportamiento esperado y la extensibilidad del modelo.

En resumen, el diseño propuesto satisface los requerimientos funcionales establecidos, facilita el mantenimiento a través de una clara separación de responsabilidades, y proporciona una base firme para futuras mejoras y escalabilidad. Este documento concluye con la validación del diseño como adecuado para guiar la implementación del sistema interactivo basado en inteligencia artificial, que busca no solo entretener, sino también promover decisiones éticas y personalizadas a lo largo de una historia evolutiva.