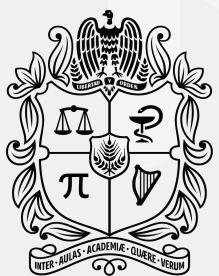


BIT CHRONICLES



UNIVERSIDAD
NACIONAL
DE COLOMBIA

BIT CHRONICLES

Manual de Usuario - Bit Chronicles

1. introducción
2. Propósito
3. Alcance del producto
 - 3.1. Análisis de requisitos
 - 3.2. Diseño y arquitectura
 - 3.3. Programación
 - 3.4. Pruebas Unitarias
 - 3.5. Documentación
 - 3.6. Mantenimiento
4. Entorno operativo
5. Requerimientos funcionales
 - 5.1. Creación y edición de personajes y campañas
 - 5.2. Conexión con IA para generación de narrativa e imágenes
 - 5.3. Registro de eventos de cada partida
 - 5.4. Visualización de mapas y entornos
 - 5.5. Navegación entre secciones
 - 5.6. Comunicación bidireccional con una base de datos
6. Estructura de Paquetes y Componentes Clave
7. Requerimientos no funcionales
 - 6.1. Eficiencia
 - 6.2. Seguridad y lógica de datos
 - 6.3. Hardware
8. Manual de Usuario

RESUMEN

Bit Chronicles es una aplicación móvil desarrollada en Android Studio que transforma la experiencia clásica de Dungeons & Dragons mediante el poder de la inteligencia artificial. Diseñada tanto para jugadores novatos como experimentados, la app permite a los usuarios crear campañas y personajes personalizados, y sumergirse en aventuras dinámicas guiadas por una IA que actúa como Dungeon Master (DM).

Gracias a la integración de IA avanzada, los usuarios pueden generar historias narradas automáticamente, adaptadas a los perfiles de sus personajes, y vivir una experiencia interactiva hablando directamente con el Dungeon Master virtual. A través de simples interacciones, la narrativa evoluciona según las decisiones del jugador, generando una jugabilidad rica, única y completamente personalizada.

Entre las funcionalidades actuales se incluyen:

- Creación de personajes mediante IA con atributos definidos por el usuario.
- Generación automática de historias épicas según la ambientación y personajes seleccionados.
- Interacción fluida con el Dungeon Master por medio de texto o voz, para tomar decisiones, explorar escenarios y avanzar en la historia.

A futuro, se integrarán características como:

- Navegación más intuitiva.
- Creación de personajes a partir de imágenes proporcionadas por el usuario.
- Nuevos módulos de personalización de historia y combate.

Bit Chronicles busca ser más que una herramienta de rol: es una plataforma inmersiva donde la creatividad y la inteligencia artificial se combinan para brindar historias que nunca se repiten.

1. Introducción

Bit Chronicles es una aplicación móvil diseñada para facilitar la creación y desarrollo de campañas de rol (RPG), incluyendo funcionalidades que asisten tanto a jugadores como al "Game Master". Mediante una interfaz amigable, visual e interactiva, la app permite llevar un registro narrativo, gestionar personajes, y explorar entornos visuales generados por una IA que actúa como Dungeon Master.

2. Propósito

El propósito de esta aplicación es ofrecer una herramienta integral para la gestión de partidas de rol narrativas, adaptable a cualquier universo (fantasía, cyberpunk, ciencia ficción, etc.), incluyendo una IA narrativa y visual que potencia la inmersión del usuario sin necesidad de material físico.

3. Alcance del producto

3.1. Análisis de requisitos

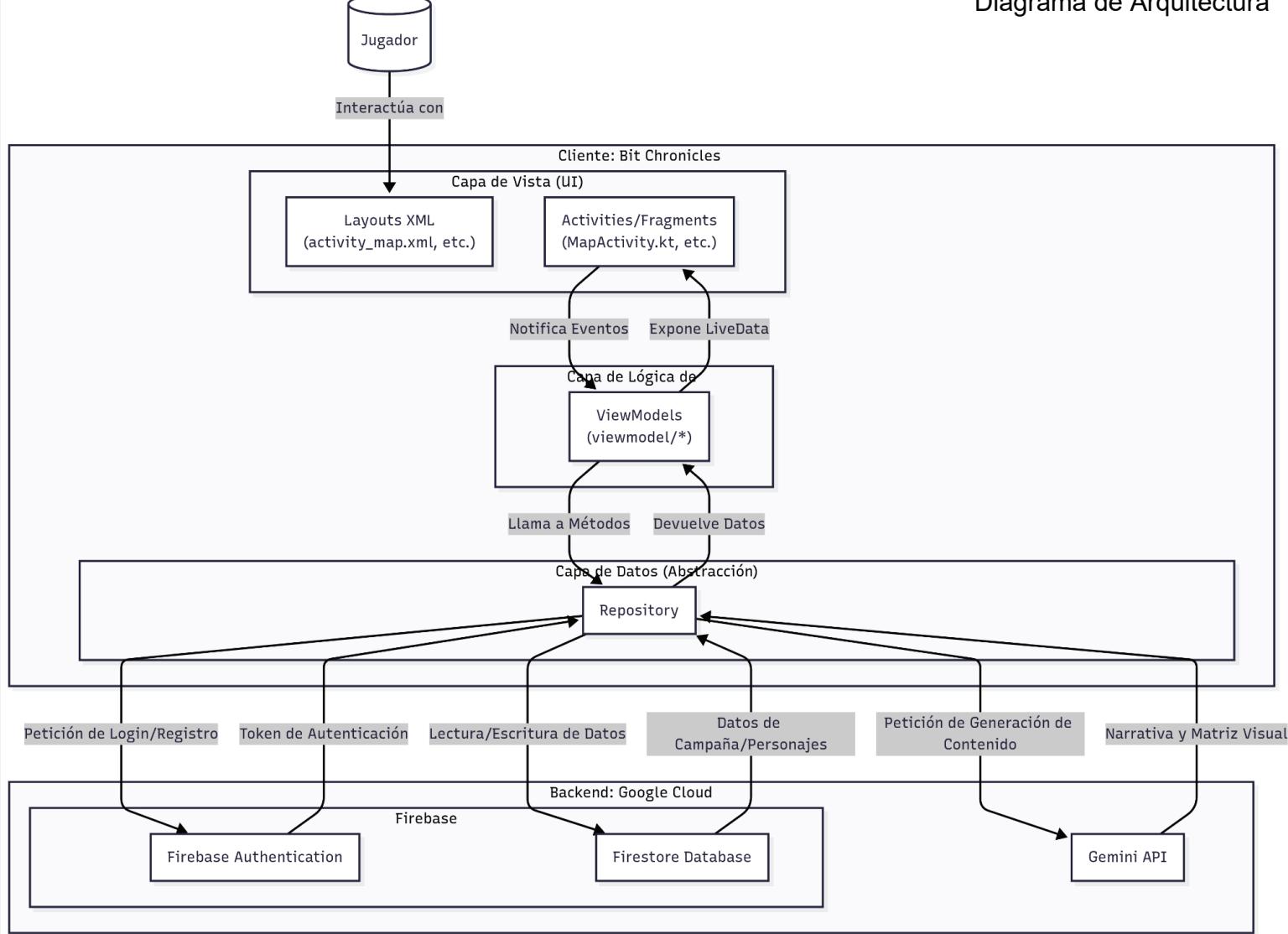
- El usuario podrá crear campañas, personajes y mundos personalizados con asistencia de una IA.
- Se podrá interactuar con una IA para narración y para la visualización de escenas mediante una matriz.
- Visualización de entornos de juego interactivos generados por la IA.

3.2. Diseño y arquitectura

- Arquitectura **MVVM** (Modelo-Vista-ViewModel).
- Comunicación mediante API con **Gemini**.
- Backend y base de datos en la nube con **Firebase**.

- Interfaz modular basada en **Activitys** y **Android Navigation Component**.

Diagrama de Arquitectura



3.3. Programación

- Desarrollado con Android Studio utilizando tanto **Kotlin** como **Java**, aprovechando la total interoperabilidad entre ambos lenguajes.
- Comunicación entre la Vista y el ViewModel.
- Integración con **Firebase SDK** para la gestión de datos y autenticación.
- Comunicación con la API de Gemini mediante su SDK oficial.

3.4. Pruebas Unitarias

- Pruebas de flujo de navegación.
- Validación de respuestas de la IA.

3.5. Documentación

- Repositorio con README técnico y sprint backlog.
- Manual de usuario.

3.6. Mantenimiento

- Sistema modular fácilmente escalable.
 - Sustitución del modelo de IA sin afectar la estructura base.
-

4. Entorno operativo

- **Sistema operativo:** Android 8.0 o superior.
 - **Hardware:** Dispositivo móvil con al menos 2 GB de RAM.
 - **Requisitos:** Requiere conexión a internet para la comunicación con Firebase y la IA.
 - **Arquitectura:** Basada en **Kotlin y Java**.
-

5. Requerimientos funcionales y Funcionamiento Interno

A continuación, se describen las funciones clave de la aplicación y el flujo de interacción entre las clases de la arquitectura **MVVM**, basándonos en la estructura de paquetes del proyecto.

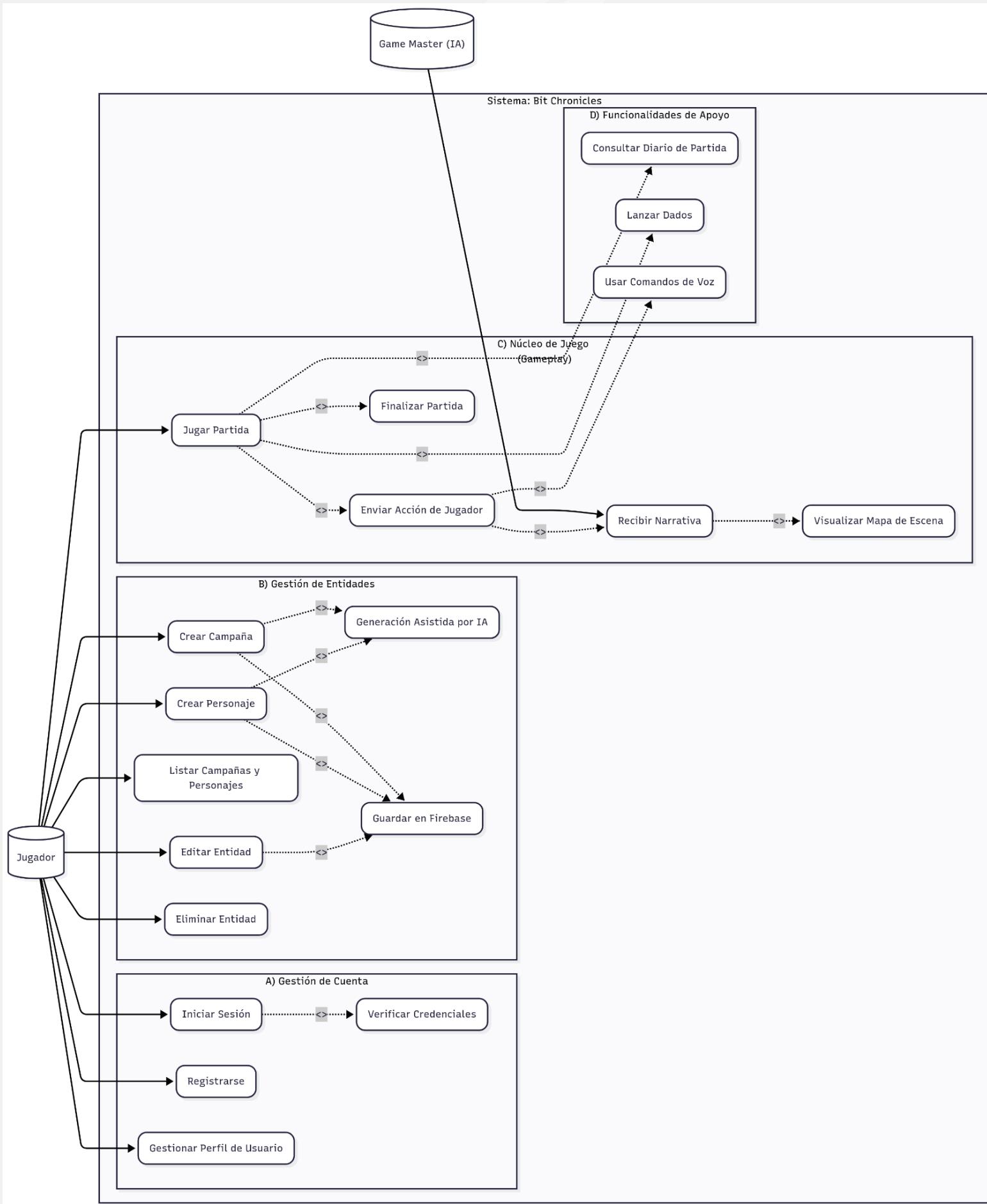


Diagrama de Clases de uso

Descripción de la Función:

La aplicación permite al usuario introducir datos iniciales para una campaña o personaje. Estos datos se envían a la IA Gemini para que genere una historia y detalles completos. El resultado final se almacena en Firebase para su posterior gestión.

Funcionamiento entre Clases (MVVM):

1. **Vista (Layouts: `activity_creat_camp.xml`, `activity_creat_person.xml`):** El usuario interactúa con estas Vistas, controladas por sus **Activities** (`CreateCampActivity.kt`, `CreatePersonActivity.kt`). Al guardar, la Vista notifica al ViewModel correspondiente.
2. **ViewModel (Paquetes: `viewmodel/campaign`, `viewmodel/character`):** Un **ViewModel** como `CampaignCreationViewModel.kt` recibe los datos iniciales. Llama al Repositorio para que gestione la lógica de creación completa:
`repository.createCampaignWithAI(initialData)`.
3. **Modelo (Paquete: `model`):** a. El **Repositorio** recibe la llamada del ViewModel. b. Utiliza una clase del paquete `model/prompts` para construir el *prompt* inicial para Gemini. c. Llama al servicio de IA (definido en `model/api`) para obtener los datos generados. d. Una vez recibe la respuesta, utiliza una clase del paquete `model/firebase` para guardar el objeto (`Campaign` o `Character`) completo en Firestore.

5.2. Conexión con IA para generación de narrativa y visualización

Descripción de la Función:

Gemini actúa como Dungeon Master generando la historia. De forma paralela, el texto narrativo se usa para generar una matriz visual que representa la escena, ofreciendo una experiencia dual.

Funcionamiento entre Clases (MVVM):

1. **Vista (Layout: `activity_map.xml`):** Es la pantalla principal de juego, controlada por `MapActivity.kt`. El usuario escribe una

acción y la envía. La Vista observa los `LiveData` del ViewModel para actualizar el chat y el mapa visual.

2. **ViewModel (Paquete: `viewmodel/map`)**: La clase `MapViewModel.kt` gestiona el estado de la partida. Llama al Repositorio para procesar la acción del jugador: `repository.processPlayerAction(actionText)`.
3. **Modelo (Paquete: `model`)**: a. El **Repositorio** es el orquestador principal. Primero, usa su servicio en `model/firebase` para cargar el contexto de la partida. b. Llama al servicio en `model/api` para enviar el *prompt* narrativo a Gemini. La conversación se estructura usando la clase `model/ChatMessage.kt`. c. Al recibir la respuesta narrativa, el Repositorio invoca a la clase `model/MapGenerator.kt`. Esta clase especializada se encarga de tomar el texto y generar la matriz visual de 0s y 1s. d. Finalmente, el Repositorio devuelve tanto el `ChatMessage` narrativo como la matriz generada al `MapViewModel`.

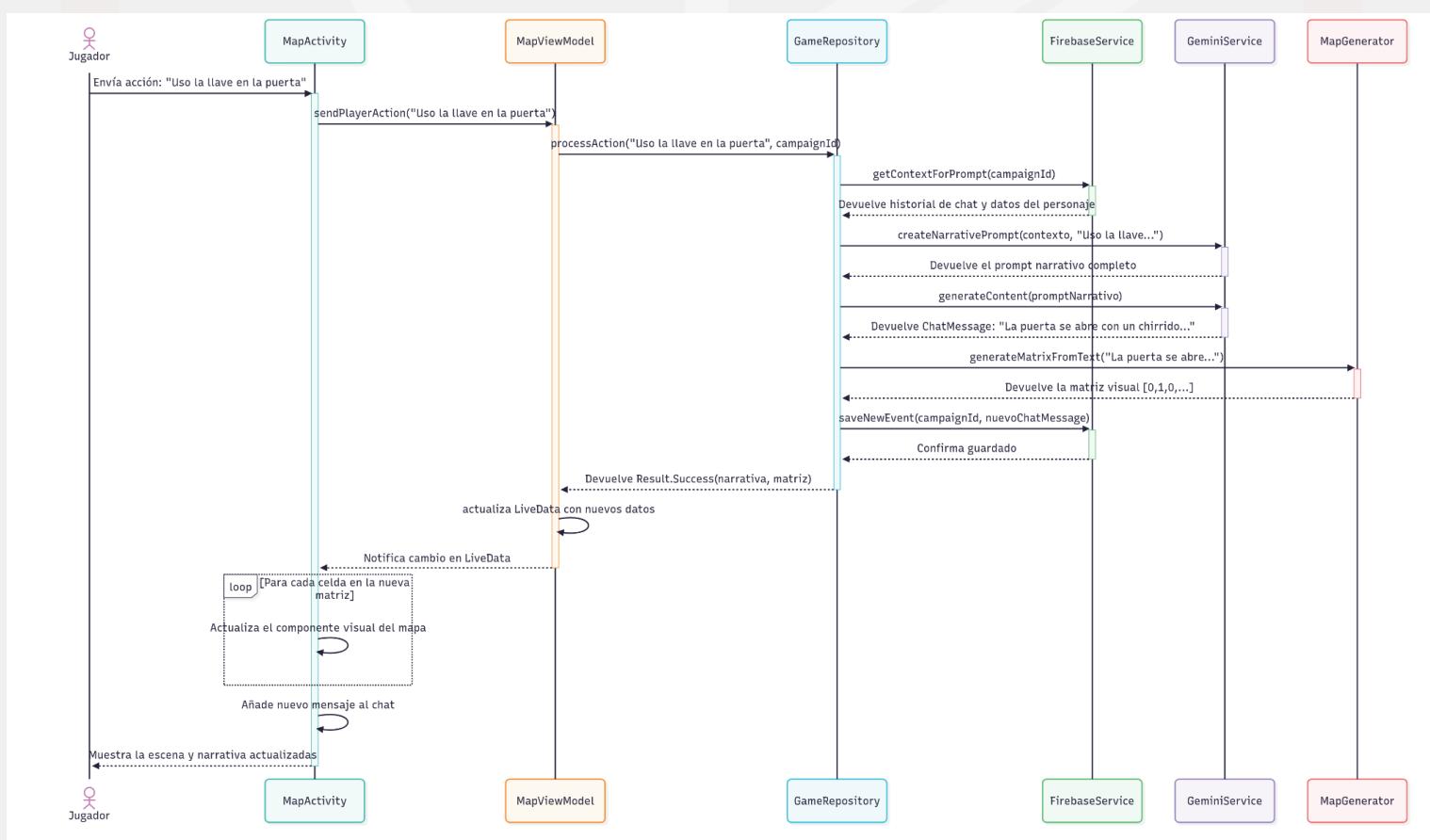


Diagrama de Secuencia

5.3. Registro de eventos de cada partida

Descripción de la Función:

Todas las interacciones se guardan automáticamente en Firebase, creando un registro persistente y consultable de la partida.

Funcionamiento entre Clases (MVVM):

- Escritura de Eventos:** El **Repositorio** (en `model`), tras recibir la respuesta de Gemini, usa su servicio de `model/firebase` para escribir el objeto `ChatMessage` en la colección de eventos de Firestore.
- Lectura de Eventos:** Una **Vista** (ej. un `LogActivity`) solicita el historial a su **ViewModel** (`LogViewModel.kt`), que a su vez lo pide al **Repositorio**, el cual lo obtiene del servicio de `model/firebase`.

5.4. Visualización de mapas y entornos

Descripción de la Función:

La aplicación renderiza la matriz de números generada por la IA en una cuadrícula visual, sincronizada en tiempo real con la narrativa.

Funcionamiento entre Clases (MVVM):

- Vista (Layout: `activity_map.xml`):** La clase `MapActivity.kt` contiene un componente visual (ej. un `RecyclerView`). Este componente está vinculado a un `LiveData` en el `MapViewModel`.
- ViewModel (Paquete: `viewmodel/map`):** El `MapViewModel.kt` expone el estado de la matriz a través de un `LiveData`, posiblemente usando una clase `UIState` (`LiveData<UIState<IntArray>>`).
- Renderizado:** Cuando el `LiveData` se actualiza, la `MapActivity.kt` es notificada y un método en ella se encarga de recorrer la nueva matriz y redibujar la cuadrícula en pantalla.

5.5. Navegación entre secciones

Descripción de la Función:

El usuario se mueve fluidamente entre las diferentes pantallas de la aplicación.

Funcionamiento entre Clases (MVVM):

- Es gestionada por la **capa de Vista** usando el **Android Navigation Component**.
- **Vista (ej. `HomeActivity.kt` o `CampaignListActivity.kt`)**: La lógica de navegación reside aquí. Un `ClickListener` en `home.xml` ejecuta el `NavController` para moverse a otra **Activity**, por ejemplo, a `CampaignListActivity.kt`.

5.6. Comunicación bidireccional con la base de datos (Firebase)

Descripción de la Función:

Firebase actúa como el backend único, guardando y sincronizando todos los datos de la aplicación en la nube.

Funcionamiento entre Clases (MVVM):

- El **Repositorio** en `model` abstrae toda la lógica de la base de datos, delegando las operaciones específicas a clases dentro del paquete `model/firebase`.
- **Flujo de Escritura**: `ViewModel` -> `Repository` -> Servicio de `model/firebase` -> SDK de Firebase.
- **Flujo de Lectura**: Servicio de `model/firebase` -> `Repository` -> `ViewModel` -> `Vista`.

6. Estructura de Paquetes y Componentes Clave

Además de los flujos principales descritos anteriormente, el proyecto está organizado en una serie de paquetes y clases de soporte que garantizan su robustez y mantenibilidad. A continuación, se detalla el propósito de estos componentes.

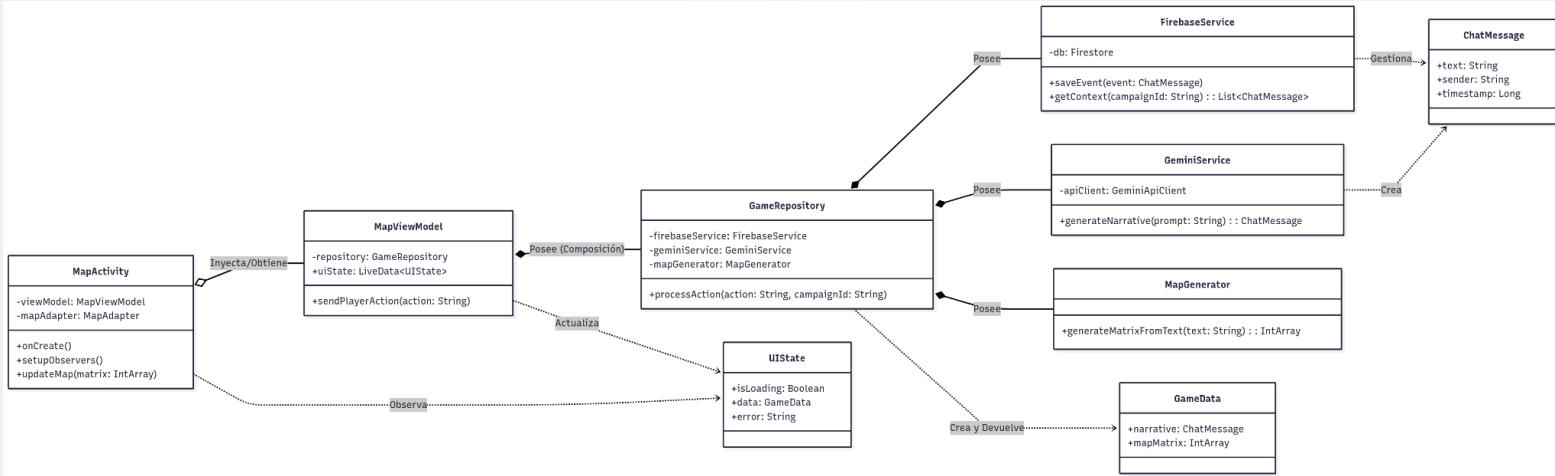


Diagrama de clases

6.1. Capa de Vista y Componentes Reutilizables (UI)

Contiene los elementos visuales de la aplicación, separando las pantallas completas de los componentes reutilizables.

- Archivos: `card_campaign.xml`, `card_character.xml`, `navbar.xml`, `titlename.xml`, `image_item.xml`.
- Función: No son pantallas completas, sino plantillas que se incluyen (`<include>`) dentro de layouts más grandes. Por ejemplo, `activity_campaign_list.xml` usa un `RecyclerView` que muestra una lista de `card_campaign.xml`. Esto asegura una interfaz de usuario consistente y reduce la duplicación de código.

6.2. Clases de Datos (Modelos)

Definen la estructura de los datos que la aplicación maneja.

Residen principalmente en el paquete `model/data`.

- Archivos: `User.kt`, `ChatMessage.kt`, y otros modelos como `Campaign.kt` o `Character.kt`.
- Función: Estas clases actúan como contratos de datos para toda la aplicación. `FirebaseService`, por ejemplo, no trabaja con datos sueltos, sino con objetos tipados (como `ChatMessage`), lo que previene errores y facilita la gestión de la información.

6.3. Clases de Soporte y Utilidades

Son clases auxiliares que realizan tareas esenciales para que la arquitectura funcione correctamente.

- **Archivos:** `MainViewModelFactory.kt`, `UIState.kt`, y clases dentro de `model/prompts`.
- **Función:**
 - `MainViewModelFactory.kt`: Se encarga de la creación de `ViewModels` y de la inyección de dependencias (como el `Repository`), una práctica fundamental de la arquitectura limpia.
 - `UIState.kt`: Es una clase contenedora (wrapper) que permite al `LiveData` comunicar estados claros a la Vista (ej. `CARGANDO`, `ÉXITO`, `ERROR`), mejorando la robustez de la UI.
 - Clases en `model/prompts`: Su responsabilidad es construir las cadenas de texto (prompts) que se envían a Gemini, manteniendo esta lógica fuera del `Repository` y logrando un código más limpio.

6.4. ViewModels Específicos por Funcionalidad

El paquete `viewmodel` está organizado por características, demostrando una aplicación granular del patrón MVVM.

- **Archivos:** Paquetes `viewmodel/auth`, `viewmodel/campaign`, `viewmodel/character`, `viewmodel/home`, etc., y la clase `MainViewModel.kt`.
- **Función:**
 - ViewModels por paquete: Cada pantalla o funcionalidad principal tiene su propio `ViewModel` (ej. `AuthViewModel` para `activity_login.xml`), aislando su lógica y estado.
 - `MainViewModel.kt`: Generalmente, se utiliza para manejar estados que deben ser compartidos entre múltiples pantallas o que pertenecen a la `Activity` principal que las contiene.

6.5. Funcionalidades Adicionales

Son características autocontenidoas que complementan la experiencia principal del usuario.

- Archivos: `DiceRollActivity.kt`, `activity_dice_roll.xml`, y la clase `VoiceCommandPrompt.kt`.
- Función: Estas herramientas, como el lanzador de dados o los comandos por voz, tienen su propio ciclo de vida simple y enriquecen la aplicación sin interferir con el flujo de juego principal.

7. Requerimientos no funcionales

7.1. Eficiencia

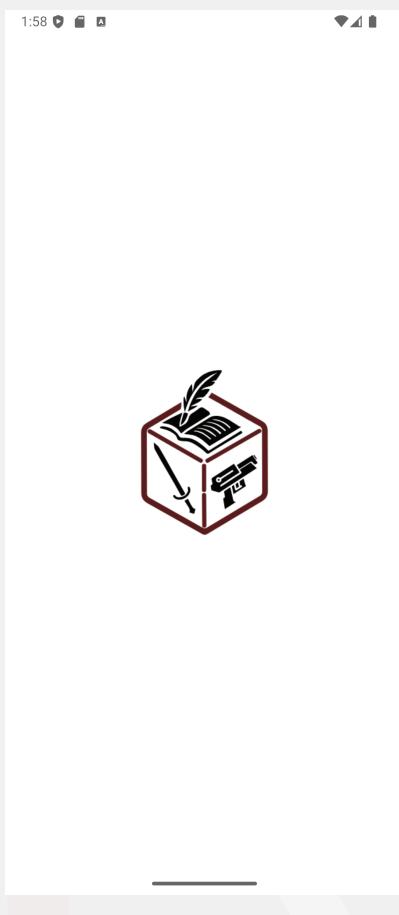
- Respuesta de la IA en menos de 3-5 segundos.
- Animaciones fluidas en la navegación entre vistas.
- Uso optimizado de lecturas/escrituras en Firebase.

7.2. Seguridad

- Autenticación de usuarios mediante **Firebase Authentication**.
- Reglas de seguridad en **Firestore** para garantizar que un usuario solo pueda acceder y modificar sus propios datos.

8. Manual de Usuario (Guía Visual)

NAVEGACIÓN



SplashScreen

Descripción

La Splash Screen es la pantalla de bienvenida que aparece automáticamente al iniciar la aplicación. Su función principal es mostrar el logo de la aplicación y brindar una breve introducción visual mientras se cargan los recursos iniciales del sistema.

Funcionalidad

Se muestra durante unos segundos mientras la app se prepara para abrir la siguiente pantalla.

Ayuda a reforzar la identidad visual de la app mediante el logotipo y colores característicos.

No requiere interacción del usuario.

Log-in

Descripción

La pantalla de Login permite a los usuarios ingresar a la aplicación mediante credenciales. Es un punto de acceso seguro al contenido personalizado y funciones basadas en la IA, como campañas e historias.

Funcionalidad

Campos de entrada:

Correo electrónico o nombre de usuario

Contraseña.

Boton de ingresar



Home Pantalla principal

Descripción

La pantalla principal tiene como objetivo mostrar al usuario las acciones principales en las que se basa la aplicación, tanto el botón campañas como el botón personajes nos llevan a otra pantalla

Funcionalidad

Botones activos:

Boton CAMPañAS

Boton PERSONAJES



Campañas

Descripción

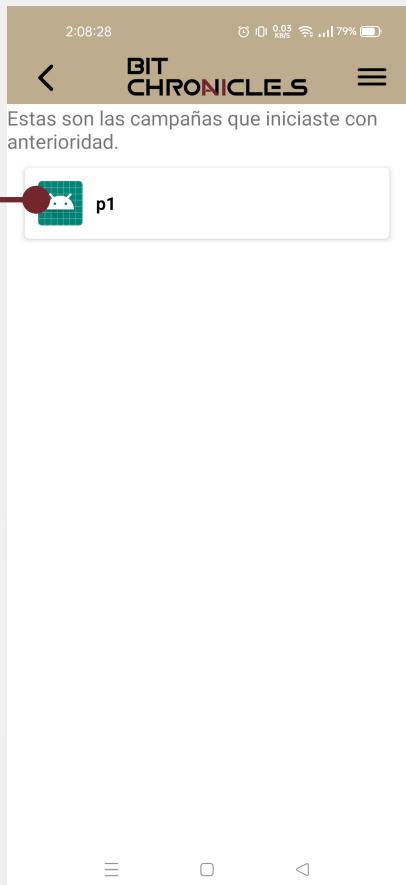
La pantalla de CAMPAÑAS se muestran dos opciones activas la de VER MIS CAMPAÑAS en donde se mostrara un listado de las campañas almacenadas en la DB y la de CREAR CAMPAÑA que corresponde a generar una nueva campaña

Funcionalidad

Botones activos:

Boton VER MIS CAMPAÑAS

Boton CREAR CAMPAÑA



Ver mis campañas

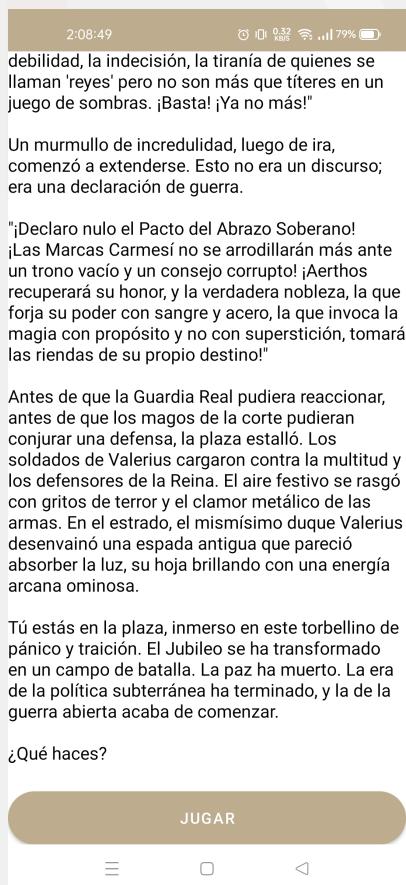
Descripción

La pantalla de CAMPAÑAS se muestran dos opciones activas la de VER MIS CAMPAÑAS en donde se mostrara un listado de las campañas almacenadas en la DB y la de CREAR CAMPAÑA que corresponde a generar una nueva campaña

Funcionalidad

Botones activos:

lista de campañas



Campañas

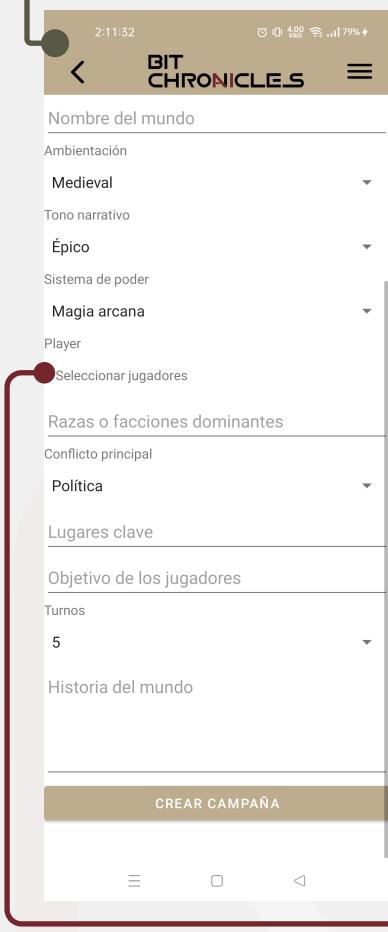
Descripción

La pantalla de las campañas iniciadas anteriormente recopila a modo de lista los datos almacenados en la DB correspondiente en este caso Firebase , es un resumen del contexto de la campaña generada por la IA, con datos como el nombre de la historia.

Funcionalidad

Botones activos:

Boton JUGAR



Crear campaña

Descripción

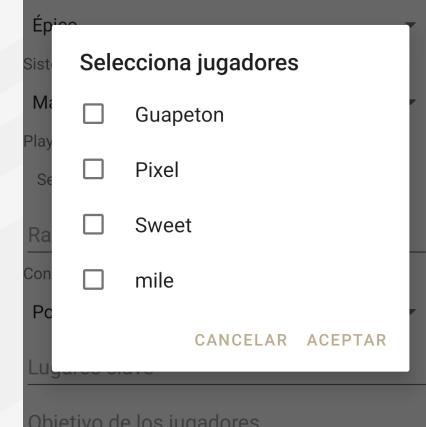
En la pantalla de crear campaña se puede observar varios tipos de campos en donde enviar datos para ser procesados, en este caso por la IA para generar una historia, de dos tipos, menú desplegable y entrada de texto. Entre estos dos se pueden generar historias brindando los datos correspondientes como nombre del mundo, ambientación, historia general, razas, sistema de poder, etc.

Funcionalidad

Botones activos:

Crear campaña

Personajes creados



PERSONAJES

Personajes

Descripción

En la pestaña nuevamente vemos dos opciones, la de ver los personajes ya creados y la de crear un nuevo personaje.

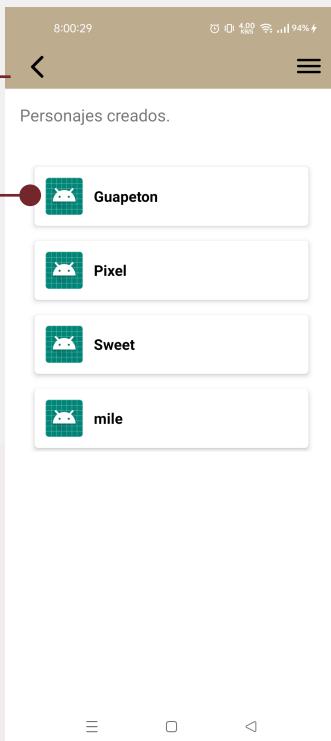
Funcionalidad

Botones activos:

VER MIS PERSONAJES

CREAR PERSONAJE

≡ ○ ◀



Ver mis personajes

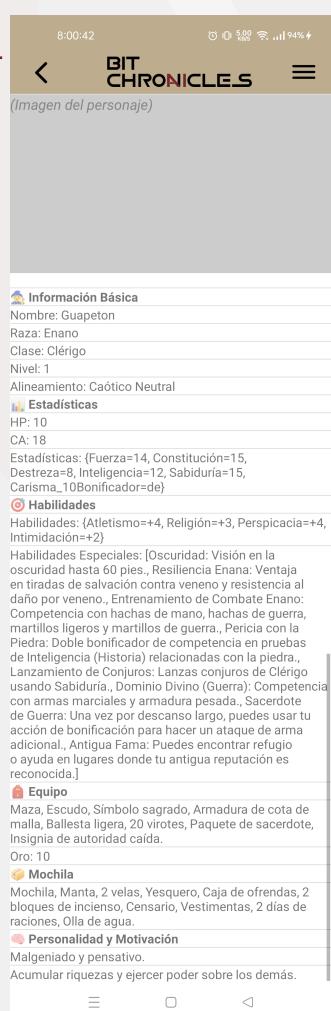
Descripción

en la pantalla de ver mis personajes se ve un listado de los personajes almacenados en la DB, caso parecido al de las campañas , se pueden ver tanto los personajes propios como los de otros jugadores.

Funcionalidad

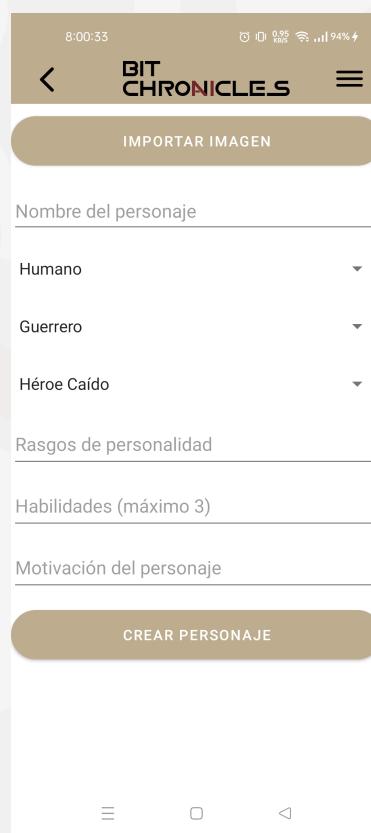
Botones activos:

Personajes creados



Descripción

Al seleccionar un personaje se muestra el resumen creado a partir de los datos ingresados al momento de crearlo, la IA toma estos datos y genera un contexto que se guarda en la DB, esto incluye un resumen de la historia y contexto del personaje, datos como Nombre, raza, clase, nivel, estadísticas y equipamiento.



Crear personaje

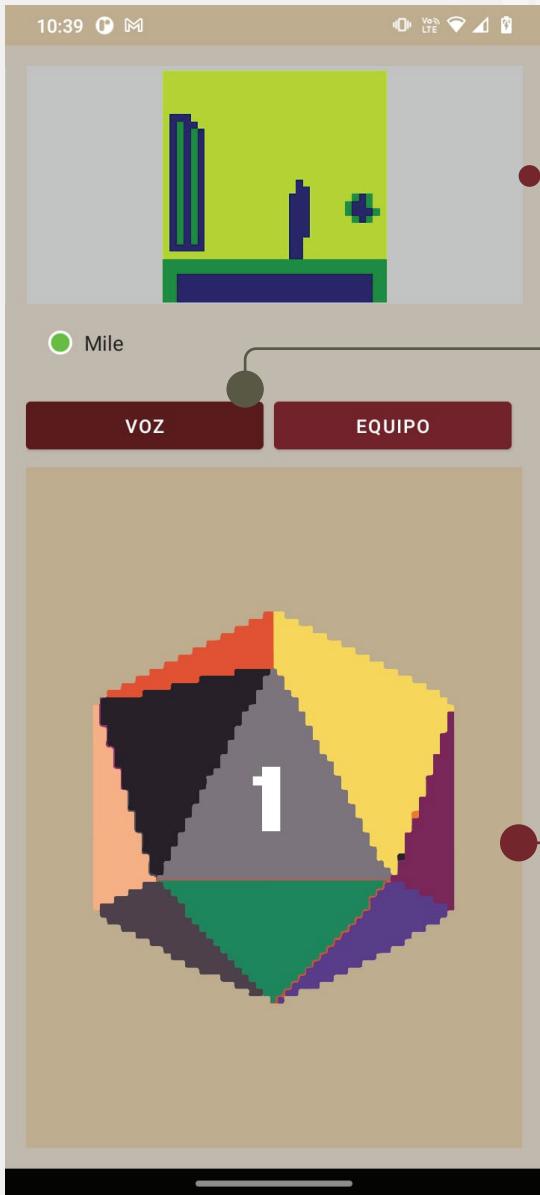
En este caso, la pestaña de creación de personaje se ve conformada por nuevamente varios campos en los que ingresar datos de texto, así como campos en los que hay menú desplegable para limitar un poco y alinear con el contexto de las ambientaciones las elecciones del usuario, después estos datos son procesados por la IA y a continuación se genera la hoja de resumen de personaje para ser almacenada en la base de datos

JUGAR

La pantalla de juego

Descripción

Esta pantalla se divide en tres secciones



Imagen

la primera superior en donde la IA por medio de una matriz de 32 x 32 genera una imagen, en este caso en 3 tonos referente al texto generado, a partir del prompt ingresado por medio del botón voz del usuario

Botones

El botón voz del usuario solicita permiso para acceder al micrófono y por medio del servicio de Google voice to text, escucha y registra la voz del usuario y envía esto como texto a la IA



El dado

Se muestra un dado de 20 caras en donde el resultado de su roll o tirada se almacena en una variable, esta variable se toma como la tirada y dependiendo de esta, se define si la acción que solicito el jugador es exitosa o no.

Créditos

Nombre del desarrollador: Maycol Andrei Figueroa
Diego Alejandro Hurtado
Julian Santiago Barbosa

Nombre del proyecto: Bit Chronicles
Versión actual: 1.0
Repositorio: https://github.com/Mike-arch-code/Bit_Chronicles