



Universidad
Europea MADRID

PROYECTO INTEGRADOR

1DAM/DAW

ADRAKODE



[Adrián ARCONES GÓMEZ]

[Ainhoa BLANCA GÁLVEZ]

[Daniel CORREA VILLA]

[Rubén PEÑA GONZALEZ]

Índice

Resumen	3
1. Introducción	4
2. Objetivos	5
3. Tecnologías utilizadas	9
4. Desarrollo e implementación	11
5. Metodología	21
6. Resultados y conclusiones	21
7. Trabajos futuros	29
Anexos	31
Anexo I – Listado de requisitos de la aplicación	31
Anexo II – Guía de uso de la aplicación	34
Anexo III	34

Resumen

El proyecto "AdraKode" surge como una iniciativa dentro del ámbito académico para desarrollar una aplicación destinada a gestionar los personajes y partidas de un club de rol de la UEM. Con el objetivo de facilitar la administración y consulta de información, este proyecto integra conocimientos de Programación, Bases de Datos y Entornos de Desarrollo.

En su núcleo, la aplicación se enfoca en ofrecer a los miembros del club una plataforma intuitiva y funcional para manejar diversas actividades relacionadas con el mundo del rol. Desde la gestión de personajes hasta el control de las sesiones de juego, "AdraKode" busca optimizar los procesos administrativos y mejorar la experiencia de los usuarios.

La arquitectura de la aplicación sigue el patrón Modelo-Vista-Controlador (MVC), que separa la lógica de negocio de la interfaz de usuario y la gestión de datos. En este sentido, Java es el lenguaje utilizado para la implementación de la lógica de la aplicación, mientras que MySQL se emplea como sistema de gestión de base de datos para almacenar y recuperar la información necesaria.

Para el desarrollo y gestión del código, se utilizan herramientas como Eclipse y Git, que permiten una colaboración eficiente entre los miembros del equipo y un control de versiones adecuado durante todo el proceso de desarrollo.

En cuanto a la metodología de trabajo, se opta por Scrum, una metodología ágil que favorece la flexibilidad y la adaptación a los cambios a lo largo del proyecto. Esto permite un desarrollo iterativo y colaborativo, donde se priorizan las funcionalidades más importantes y se entregan incrementos de software de forma regular.

En resumen, "AdraKode" es un proyecto ambicioso que combina conocimientos técnicos con la pasión por el rol, con el objetivo de ofrecer una herramienta útil y eficiente para la comunidad de jugadores de la UEM. Con una estructura bien definida, una arquitectura robusta y una metodología ágil de desarrollo, este proyecto promete transformar la manera en que se gestionan y disfrutan las partidas de rol.

Palabras clave: ADRAKODE; aplicación; usuario; personajes; partidas.

1. Introducción

El proyecto "AdraKode" se desarrolla a partir de las asignaturas de Programación, Bases de Datos y Entornos de Desarrollo. Este proyecto tiene como objetivo principal la creación de una aplicación para gestionar personajes y partidas de un club de rol de la UEM. La aplicación busca facilitar la administración y consulta de información por parte de los miembros del club, ofreciendo funcionalidades como el alta, baja y modificación de partidas y personajes, así como la consulta y listado de miembros, partidas y personajes.

La información a manejar en la aplicación se estructura de la siguiente manera:

- Miembros del club:
 - Identificación: Cada miembro se identifica con un número único.
 - Número de expediente: Un identificador adicional para los registros académicos.
 - Nombre y apellidos: Información personal de los miembros.
 - Estudios: Detalles sobre lo que están estudiando.
- Personajes de las partidas:
 - Identificador único: Cada personaje tiene un identificador único.
 - Nombre del personaje: El nombre del personaje en el juego.
 - Asociación a un jugador: Los personajes están asociados a un jugador mediante su ID.
 - Raza: Cada personaje pertenece a una raza específica.
 - Clase: Cada personaje pertenece a una clase específica.

- Nivel de experiencia: El nivel de experiencia de un personaje que cambia según los eventos de cada partida.
- Características: Cada personaje tiene las siguientes características, cada una con un valor menor de 100:
 - Fuerza (str)
 - Destreza (dex)
 - Constitución (con)
 - Inteligencia (int)
 - Sabiduría (wis)
 - Carisma (cha)
- Información sobre las Partidas:
 - Identificador único: Cada partida tiene un identificador único.
 - Nombre de la partida: El título o nombre de la partida.
 - Ambientación: Un texto que describe la ambientación de la partida.
 - Personajes participantes: Los personajes que participan en la partida.
 - Horario: Día y hora de la semana en que se juega la partida.
 - Duración: La duración de cada sesión.
 - Número de sesión: El número de sesión en la que están.
 - Estado: Indicador booleano para saber si la partida está en curso o terminada.

La aplicación sigue una arquitectura MVC (Modelo - Vista - Controlador), utilizando Java para la lógica de la aplicación, MySQL para la base de datos y herramientas Eclipse y Git para el desarrollo y la gestión del código. La metodología Scrum se emplea para la gestión ágil del proyecto, permitiendo un desarrollo iterativo y colaborativo.

2. Objetivos.

Objetivos Generales

El proyecto "AdraKode" se desarrolló con un enfoque en la colaboración, la comunicación efectiva y la resolución de problemas. Desde el inicio, se establecieron roles claros y definidos para cada miembro del equipo, asegurando una distribución equitativa de las responsabilidades. Las

reuniones regulares permitieron discutir las cargas de trabajo, compartir ideas y tomar decisiones de forma conjunta.

A pesar de los inevitables desafíos, como diferentes opiniones, retos inesperados y la presión por cumplir los plazos, el equipo supo mantener un ambiente de trabajo positivo y respetuoso. La comunicación abierta y la búsqueda de soluciones en conjunto permitieron superar estas dificultades y mantener el espíritu de innovación y actualización del trabajo hasta el final del proyecto.

Objetivos Específicos

Bases de datos

- Creación de bases de datos sólidas: Se definió la estructura y las características de los elementos de la base de datos siguiendo los principios del modelo relacional, asegurando una base sólida para el almacenamiento de información.
- Diseño de modelos lógicos eficientes: Se interpretaron diagramas entidad-relación (E/R) para diseñar modelos lógicos de bases de datos que fueran eficientes, precisos y libres de redundancias.
- Implementación de bases lógicas eficientes: Se utilizaron herramientas gráficas, asistentes y el lenguaje de datos (DDL) para traducir el modelo lógico en una implementación física eficiente en el sistema de gestión de bases de datos.
- Manipulación de datos con fluidez: Se dominó el lenguaje de manipulación de datos (DML) para consultar, insertar, actualizar y eliminar información almacenada en las bases de datos.

Programación

- Diseño de clases necesarias para seguir el patrón MVC: Se implementó el patrón MVC (Modelo-Vista-Controlador) en el proyecto, una metodología de desarrollo de software que ayudó a organizar las clases y separar las responsabilidades.
- Implementación de cada clase para lograr el objetivo:
 - Vista: Se crearon interfaces gráficas atractivas e intuitivas, asegurando una interacción fluida y agradable para el usuario.

- Modelo: Se representaron los datos de la aplicación mediante clases de modelo, garantizando la integridad y consistencia de la información.
- Control: Se implementó la lógica de presentación y el almacenamiento de datos.
- Desarrollo de una aplicación completa conectada a una base de datos:
 - Conexión a la base de datos: Se establecieron conexiones seguras y eficientes con la base de datos relacional utilizando los mecanismos adecuados para acceder y manipular datos.
 - Gestión de la información: Se desarrolló una aplicación funcional que gestiona información almacenada en la base de datos, utilizando las clases de modelo, vista y control para crear la interacción entre el usuario, la aplicación y la base de datos.

Entornos de Desarrollo

- Análisis y diseño de aplicaciones utilizando técnicas UML: Se dominó el lenguaje de modelo unificado (UML), una herramienta indispensable para analizar y diseñar aplicaciones de software, creando diagramas que representan la estructura y el comportamiento del sistema.
- Documentación de aplicaciones de forma clara y completa: Se elaboró documentación completa y comprensible para las aplicaciones desarrolladas, incluyendo especificaciones funcionales, descripciones técnicas y manuales de usuario, facilitando la comprensión y el mantenimiento del software.
- Gestión de versiones de software y trabajo colaborativo: Se utilizaron herramientas y metodologías para gestionar las diferentes versiones de un software, controlando los cambios y asegurando la colaboración efectiva entre los miembros del equipo de desarrollo.
- Realización de pruebas de software exhaustivas: Se implementaron estrategias de pruebas de software para identificar y corregir errores, garantizando la calidad y el buen funcionamiento de las aplicaciones.
- Aplicación de metodologías ágiles para el desarrollo y planificación de software: Se adoptó la metodología ágil Scrum para planificar, desarrollar y entregar software de manera incremental y adaptativa, respondiendo a las necesidades cambiantes del proyecto.

3. Tecnologías utilizadas

Durante el desarrollo del sistema pedido por el Club de Rol de la UEM, se utilizarán varias herramientas que permitirán avanzar en el proyecto sin limitaciones y/o problemas.

- Java

El lenguaje de programación que se usará para codificar la aplicación será Java. Este es un lenguaje escalable, capaz de soportar grandes sistemas y aplicaciones con un alto número de usuarios. Su arquitectura basada en máquinas virtuales (JVM) permite ejecutar código Java en diferentes plataformas sin necesidad de recopilación, lo que facilita la implementación y distribución del sistema. Por eso se elige este lenguaje orientado a objetos el cual permitirá implementar un sistema CRUD para poder realizar consultas SQL desde la propia aplicación de Java como crear, leer, actualizar o eliminar. Esto será útil ya que se podrán realizar validaciones en las consultas si ocurre algún error en estas. Además, se utilizará un modelo de arquitectura MVC (Modelos, Vistas y Controladores) para separar las responsabilidades de los modelos lógicos, mejorando la depuración y la organización del proyecto.

- Eclipse

Eclipse brinda herramientas para diseñar interfaces gráficas de usuario (GUI) de manera intuitiva. Se pueden crear formularios, menús, barras de herramientas y otros elementos gráficos con facilidad. Además, permite organizar los elementos de la interfaz gráfica en una estructura jerárquica clara, facilitando la comprensión y el mantenimiento del código. En definitiva, se utilizará Eclipse para implementar las clases encargadas de la interfaz gráfica, así como las clases de la lógica de la aplicación. Además, servirá para conectar la aplicación con la base de datos, en este caso MySQL, mediante el sistema CRUD hecho en Java. Se tendrá en cuenta la jerarquía en el diseño, de tal forma que sean fácilmente identificables los elementos principales de la misma, así como el orden de ejecución. Respecto a la interfaz gráfica, se implementará una disposición clara de los contenidos, evitando interfaces sobrecargadas (una disposición de elementos limpia).

- Git

Git permite realizar un seguimiento detallado de los cambios realizados en el código del proyecto. Cada vez que un desarrollador modifica el código, puede crear un “commit” que registra el estado del código en ese momento. Esto permite revisar el historial de cambios, volver a versiones anteriores si es necesario e identificar quién realizó cada cambio. Además, permite crear ramas del código principal, lo que permite trabajar en diferentes funcionalidades o correcciones de errores sin afectar el código principal. Esto facilita el desarrollo paralelo y de esta manera no se produce ningún conflicto. Se realizará el análisis y el diseño de la aplicación empleando técnicas UML. Se creará un repositorio de GitHub donde se controlarán las versiones de desarrollo del proyecto. Se gestionarán las diferentes versiones del software y el trabajo colaborativo, además se realizarán pruebas de testeo sobre los programas.

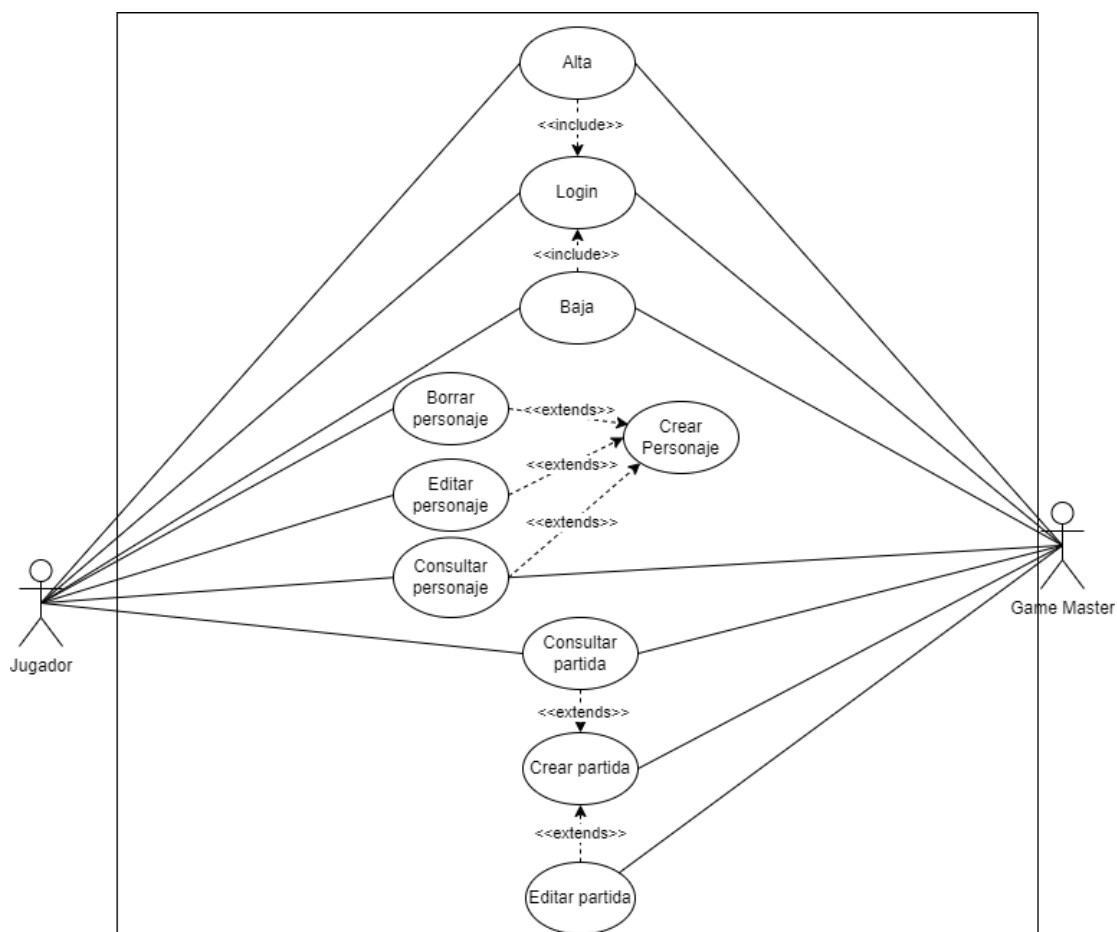
- MySQL

Se definirá la estructura de la base de datos, estableciendo las tablas que la componen, las relaciones entre ellas y los atributos de cada tabla. Esto se realizará siguiendo el modelo relacional, que organiza la información en tablas relacionadas entre sí. Se diseñarán modelos lógicos normalizados para evitar redundancia de datos y garantizar la integridad de la información. La normalización se realiza interpretando diagramas entidad-relación (E-R), que representan las entidades del sistema y sus relaciones. Se traducirá el modelo lógico en un diseño físico de la base de datos, utilizando herramientas gráficas, asistentes y el lenguaje SQL. Por lo tanto, se utilizará MySQL para crear la base de datos, definiendo así su estructura y las características de sus elementos según el modelo relacional. Se diseñarán modelos lógicos normalizados interpretando diagramas entidad/relación, se realizará el diseño físico de bases de datos utilizando asistentes, herramientas gráficas y el lenguaje de definición de datos (SQL), y por último se consultará y modificará la información almacenada. Además, se almacenará la información del Club de Rol UEM y a posteriori se utilizarán consultas SQL para recuperar información específica de la base de datos.

4. Desarrollo e implementación

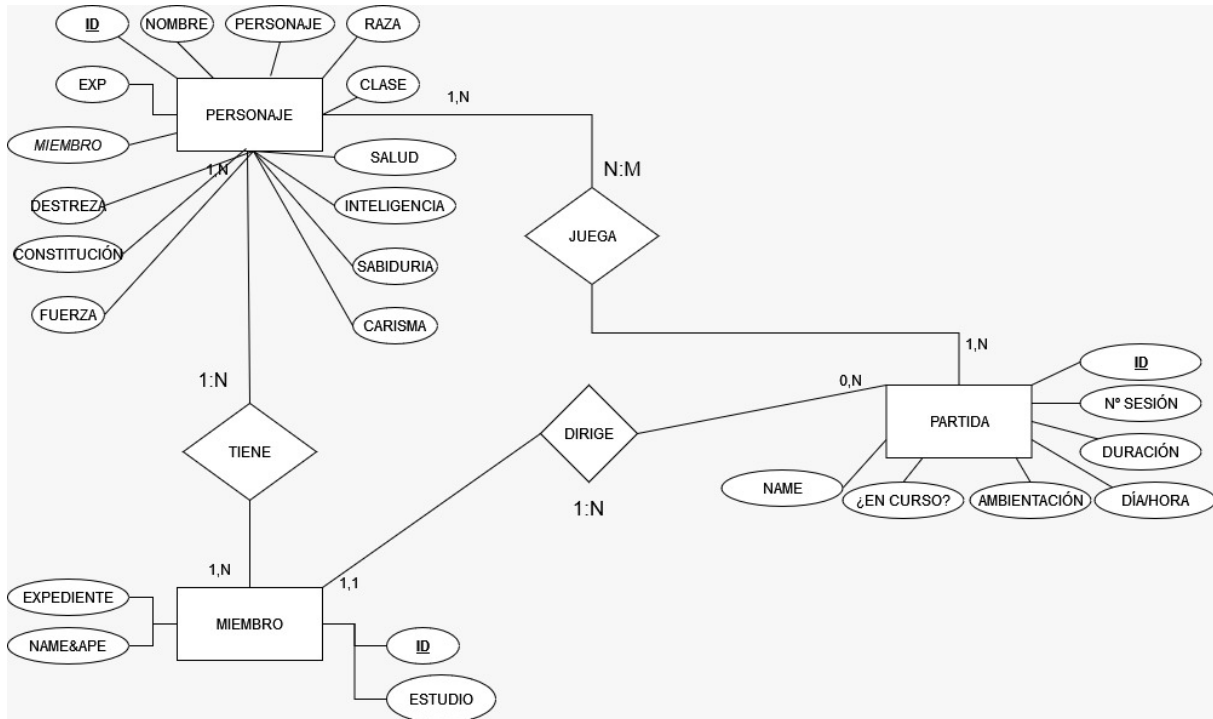
DIAGRAMA DE CASOS DE USO.

Para el diseño del diagrama de casos de uso, se enfoca en cómo sería la interfaz. De esta forma, se añaden para el login inclusiones que se consideran necesarias para entrar al juego y, por otro lado, también se añaden varias extensiones para crear personaje y crear partida. También se puede apreciar que hay dos actores: jugador y game master, con sus respectivas operaciones.



GENERACIÓN DEL MODELO RELACIONAL. NORMALIZACIÓN.

Se ha pasado el modelo entidad relación al modelo relacional, estableciendo las tablas correspondientes a las entidades y las relaciones que lo necesiten. Además, se ha comprobado que la base está normalizada porque no tiene atributos multievaluados.



MIEMBRO				PERSONAJE											
ID	EXPEDIENTE	NAME&APE	ESTUDIO	ID	NOMBRE	PERSONAJES	RAZA	EXP	CLASE	DESTREZA	SALUD	CONSTITUCIÓN	SABIDURIA	FUERZA	CARISMA

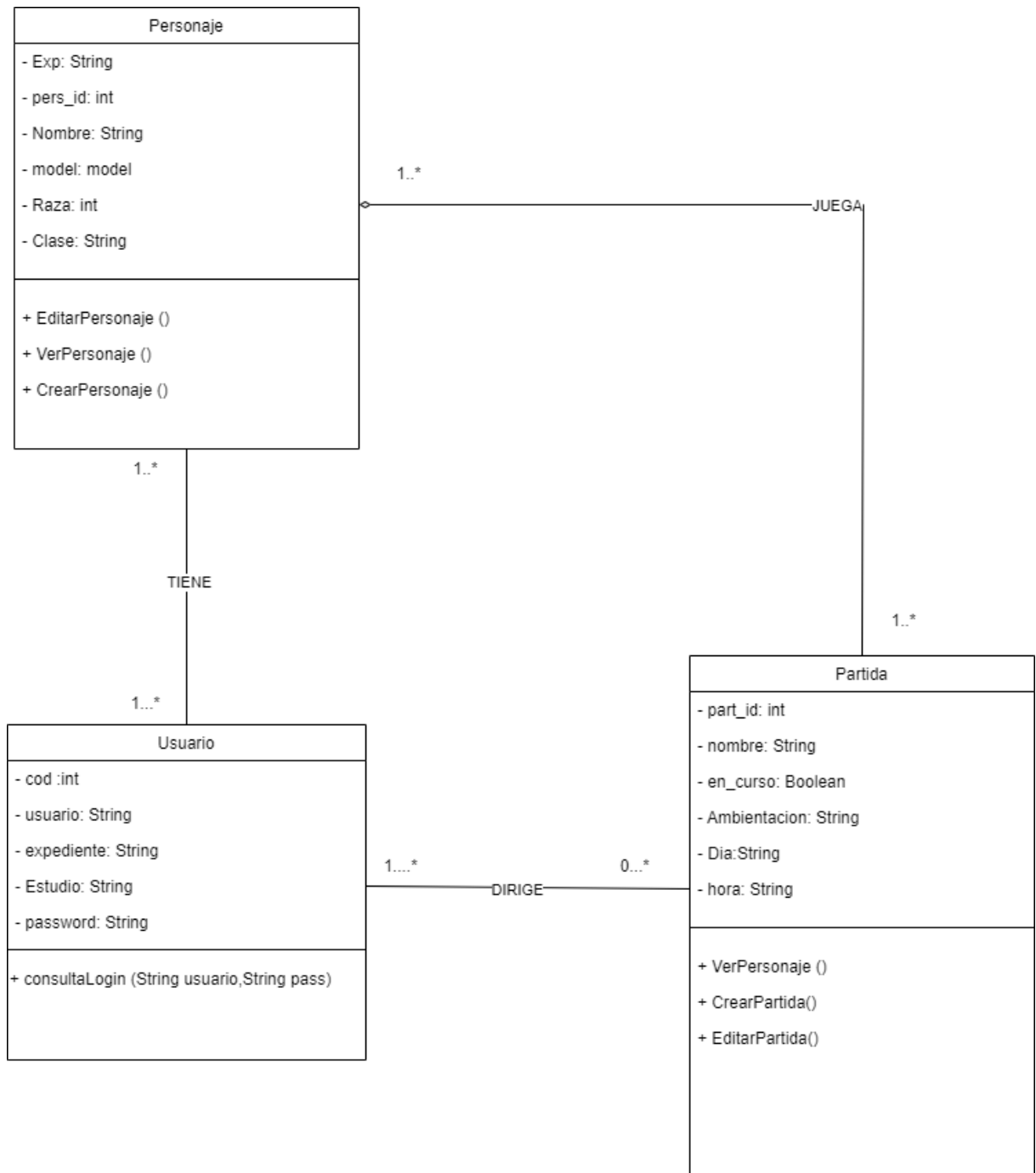
PARTIDA								JUEGA		TIENE	
ID	NAME	¿EN CURSO?	AMBIENTACIÓN	DÍA/HORA	DURACIÓN	N° SESIÓN	ID MIEMBRO	ID PERSONAJE	ID PARTIDA	ID MIEMBRO	ID PERSONAJE

GENERACIÓN DEL DIAGRAMA DE CLASES.

A partir del diagrama de clases:

- Creación de las clases pertenecientes al Modelo MODEL:
 - Delete: Elimina cualquier dato de la base de datos.
 - Insert: Inserta cualquier dato a la base de datos.

- Model: Conexión a la base de datos MYSQL, la cual tiene atributos que almacenan la información de la conexión y el método `Model_mysql_connect()` para realizar la conexión con la base de datos.
- Query: Clase que sirve para hacer consultas a la base de datos.
- Select: Representa consultas a la base de datos. Tiene un atributo `database_name` que almacena el nombre y dos métodos `create_sql()` y `exec_sql()` que devolverán datos.
- Update: Se utilizará para posibles actualizaciones en la base de datos.
- LISTENERS:
 - EditarPersonajeListener: Botón para editar el personaje.
 - ListenerBottonLogin: Botón para iniciar sesión en el login.
 - LoginListener: Clase del listener para comprobar si es correcto. Si lo es, cerrará la ventana de login y mostrará la ventana principal (menú).
 - PMenuListener: Clase en la cual se introducen las acciones de las demás clases para cambiar la ventana según la opción que se clique.
 - VerPersonajesListener: Botón de editar personaje. Redirige a la ventana dependiendo del botón.
- Creación de las clases pertenecientes al Control encargadas del acceso a BBDD y del manejo de la aplicación.



PLANIFICACIÓN GENERAL DEL PROYECTO

En el proyecto se busca desarrollar un sistema informático que permita gestionar sus actividades y eventos de manera eficiente. En este apartado, se presentará una introducción al diseño de la interfaz de usuario y las clases que conformarán la estructura del sistema. La aplicación podrá ser usada por cualquier miembro, pero estos tendrán

diferentes permisos si son jugadores o Game Master. Estos se diferenciarán al entrar en el juego.

Para diseñar la interfaz gráfica, se crearán tres clases, las cuales se llamarán Personajes, Partidas y Miembros del club, para asignarlas a cada pestaña de la aplicación se podrán acceder a ellas a través del menú.

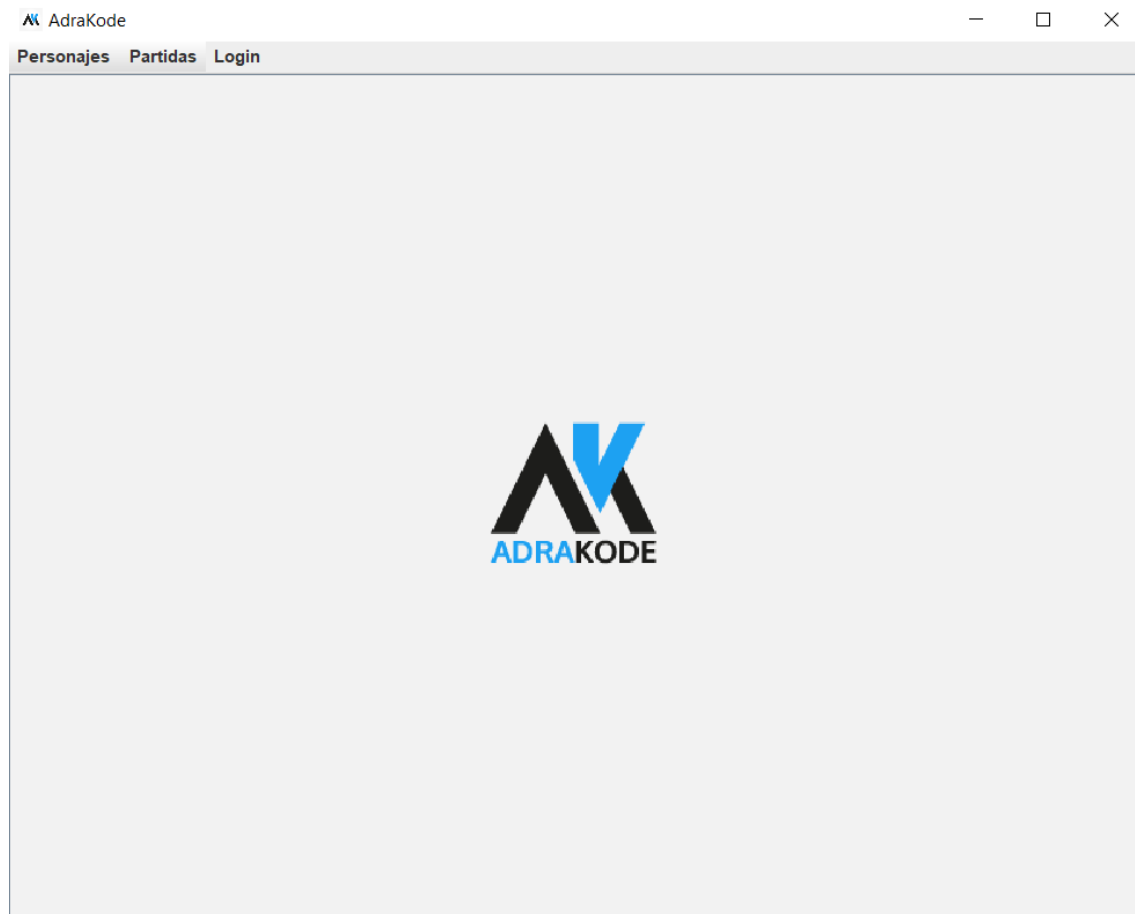
Dentro de la clase personajes habrá cuatro métodos que serán los siguientes:

- Nuevo personaje: Servirá para añadir los datos de los personajes que vayan a participar.
- Ver personajes: Para poder ver los personajes que han sido añadidos.
- Modificar personaje: Si algún personaje está mal introducido, este se podrá modificar.
- Borrar personaje: En el caso de que se quiera eliminar un personaje, este método lo permitirá.

Dentro de la clase partida se crearán los siguientes métodos:

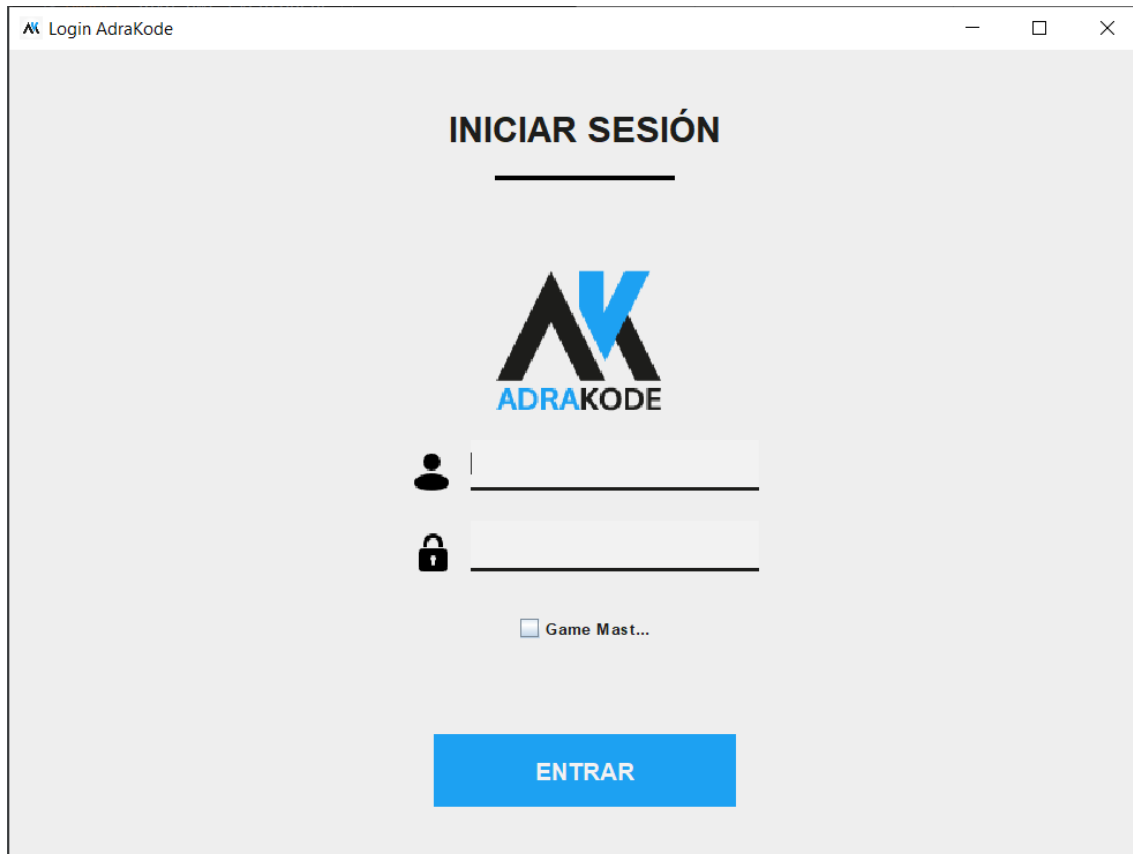
- Ver partidas: En este apartado se podrán observar las partidas en las que está el usuario.

Cada uno de los personajes que se introduzcan tendrá un identificador único.



Los requisitos mínimos para la interfaz serán los siguientes:

- Se creará una pantalla de login con la que los miembros podrán acceder con usuario y contraseña.



- El jugador podrá consultar cualquier partida, pero no modificarla.
- El jugador podrá editar y borrar sus propios personajes, crear personajes nuevos y consultar cualquier personaje en el sistema.
- El Game Master podrá consultar cualquier partida y editar aquellas que esté dirigiendo. También podrá crear partidas nuevas (que se crearán con 0 sesiones por defecto y marcadas como “en curso”).

CREACIÓN DE LAS CLASES PERTENECIENTES A LA VISTA.

La creación de clases es una etapa fundamental para organizar la funcionalidad y el comportamiento del programa. Por lo tanto, a continuación, se presenta el diseño de las clases.

- **CrearPartida:** Dentro de esta clase se han aplicado varios atributos necesarios que contendrán la información de la partida y se han creado tres métodos los cuales son `CrearPartida`, `initialize_components` y `setListener` donde `CrearPartida` sirve para inicializar los componentes, `initialize_components` para añadir los atributos a la interfaz con sus características y `setListener` el cual manejará eventos de los componentes ya mencionados anteriormente.

- **CrearPersonaje:** Al igual que en la anterior clase se han aplicado atributos necesarios para crear el personaje y también se han aplicado tres métodos necesarios, dos de ellos tendrán la misma estructura que **CrearPartida** pero el tercero tendrá la función de mostrar la ventana. Además, se implementan getters para controlar el acceso y la modificación de los atributos de la clase y los setters que permiten modificar los valores de los atributos privados de una clase.
- **EditarPartida:** Esta clase tiene las mismas funciones que **CrearPartida** pero aquí se añade una tabla para poder observar más detalladamente la información de las partidas y así de esta forma le parecerá más intuitivo al usuario. Además, se añade el botón de guardar.
- **EditarPersonaje:** Al igual que se ha mencionado que **EditarPartida** tiene las mismas funcionalidades que **CrearPartida**, pasa lo mismo para **EditarPersonaje**.
- **Home:** En esta parte de la interfaz se enseñará una foto del logo del proyecto.
- **Menu:** Estando dentro de cualquier ventana, se podrá desplazar a otras ventanas gracias al menú ya que da la opción de hacerlo gracias a sus 11 métodos que son los siguientes:
 - **Menu:** Constructor de la clase que inicializa la ventana con el título especificado y un booleano que indicará si el usuario es Game Master o no.
 - **Initialize_components:** Inicializa los componentes de la ventana, como el tamaño, la posición y el panel principal.
 - **CrearMenu:** Crea la barra de menú con los elementos principales (Personaje y Partidas) y sus submenús.
 - **SetListener:** Asigna un listener a cada botón del menú para que se ejecuten las acciones correspondientes cuando se haga clic en ellos.
 - **Make_visible:** Hace visible la ventana principal.
 - **Cargar_panel:** Cambia el panel que se muestra en la ventana principal por el panel especificado.
 - **SetIcon:** Cambia el icono de la ventana principal por la imagen del logo.
 - **MostrarMensajeConfirm:** Muestra un mensaje de confirmación al usuario para preguntar si desea salir de la aplicación.

- `MostrarMensajeConfirmBorrado`: Muestra un mensaje de confirmación al usuario para preguntar si desea borrar un elemento.
 - `Getter y setter`: Devuelve el valor del atributo `gameMaster` que indica si el usuario es Game Master.
- `VentanaPrincipalLogin`: Ventana principal que contendrá información en la base de datos sobre los usuarios para cuando quieran acceder a la aplicación e iniciar sesión. Esta clase contiene los siguientes métodos:
 - `Constructor` que inicializa la ventana con el título “Login Adrakode” y establece el color de fondo.
 - `Initialize_components`: Configura el diseño de la ventana, el tamaño, la posición y varios componentes de la interfaz de usuario como etiquetas, campos de texto, botones y una casilla de verificación.
 - `SetListener`: Asigna un objeto `LoginListener` al botón “iniciar sesión” y a la casilla de verificación “Game Master”. Este listener manejará eventos de interacción del usuario como hacer clic en estos elementos.
 - `MostrarMensajeConfirm`: Muestra un diálogo de confirmación preguntando al usuario si desea salir de la aplicación.
 - `MostrarMensajeErrorLogin`: Muestra un mensaje de diálogo indicando un nombre de usuario o contraseña incorrectos durante el inicio de sesión.
- `VerPartidas`: Aquí se mostrará una lista de partidas, información sobre la partida seleccionada y un botón para unirse a la partida. Los métodos que contiene son los siguientes:
 - `VerPartidas`: Inicializa el panel con un color de fondo y asigna un objeto.
 - `Initialize_components`: Configura el diseño del panel, el tamaño y diversos componentes de la interfaz de usuario.
 - `SetListener`: Asigna un `VerPartidaListener` al botón “jugar” para manejar los clics.
- `VerPartidasMaster`: Este muestra una lista de partidas creadas por el Master, información sobre la partida seleccionada y botones para editar, borrar y unirse a la partida.

- **VerPersonaje:** Esta clase se encarga de mostrar una interfaz gráfica para que un usuario pueda ver sus personajes. La clase hereda de JPanel lo que significa que representa un panel dentro de una ventana más grande.

DISEÑO DEL LOGO.

El logo del Club de Rol UEM es un elemento fundamental de su identidad visual, que representa los valores y la esencia del club. En este apartado, se analizará en detalle el proceso de diseño del logo, destacando las decisiones tomadas y los resultados obtenidos. La elección de la paleta de colores azules se basa en su asociación con la fantasía, la imaginación y la creatividad, valores fundamentales del rol. Los tonos azules transmiten confianza, seguridad y profesionalismo, aspectos importantes para la imagen del club. La combinación de los colores azules con blanco y negro aporta equilibrio y contraste al logo. El blanco simboliza pureza, claridad y nuevos comienzos, mientras que el negro aporta elegancia, fuerza y seriedad.



5. Metodología

Sprint #1: 18 de marzo - 31

Se realizaron presentaciones del módulo para PR1, DB1 Y ED1 con el objetivo de dar a conocer el alcance del proyecto y sus objetivos.

Durante el desarrollo del sistema pedido por el Club de Rol de la UEM, utilizamos varias herramientas que nos permitieron avanzar en el proyecto sin limitaciones y/o problemas teniendo en cuenta los requisitos de la aplicación, incluyendo funcionalidades, interfaces de usuario y flujos de datos, por lo que llegamos a un acuerdo en utilizar Java, Eclipse, Git y MySQL todo esto en relación con el software.

Para el hardware dividimos los requisitos entre mínimos y recomendados. Estos requisitos están dirigidos a los desarrolladores de la aplicación, pero también se aplican para los usuarios de esta.

Se analizaron las especificaciones técnicas del proyecto, incluyendo la arquitectura del sistema y las tecnologías a utilizar. También se elaboró un diagrama de entidad-relación (E/R) para modelar la estructura de la base de datos.

Se creó un repositorio de proyecto en GitHub para la gestión del código fuente y se estableció un tablero de Trello para la planificación y seguimiento de las tareas.

name	start	end	members
TERMINADO SPRINT 1			
Análisis y diseño de los requisitos hardware y software	2024-03-22 13:29:55	2024-03-30 11:25:55	Daniel Correa Villa, Ainhoa Blanca, Adrián Arcones
Proyecto en GitHub. Planificación con Trello.	2024-03-22 17:25:56	2024-03-30 20:25:56	Rubén Peña
Análisis de las especificaciones del proyecto. Diagrama E/R.	2024-03-22 18:07:25	2024-03-30 13:07:25	Rubén Peña

Análisis de las especificaciones del proyecto.	2024-03-22 12:36:22	2024-03-30 15:36:22	Ainhoa Blanca, Adrián Arcones
--	------------------------	------------------------	-------------------------------

Sprint #2: 1 de abril - 14 abril

Generación del modelo relacional: Se diseñó un modelo relacional que representa la estructura de la base de datos, asegurando la normalización de las tablas para evitar redundancia y garantizar la integridad de los datos.

Creación de la BBDD: Se materializó el modelo relacional en una base de datos real en el servidor, incluyendo la creación de las tablas, la definición de índices para optimizar las consultas y la implementación de triggers para automatizar tareas específicas.

Inserción de datos iniciales: Se poblaron las tablas de la base de datos con los datos iniciales necesarios para el funcionamiento básico de la aplicación, permitiendo realizar pruebas y demostraciones del sistema.

Diseño de interfaces:

- Explorando la experiencia del usuario: Se utilizaron herramientas de diseño gráfico como Figma o Adobe XD para plasmar en bocetos digitales la interfaz de usuario de las ventanas principales de la aplicación. Cada elemento fue cuidadosamente diseñado, teniendo en cuenta la usabilidad, la estética y la coherencia con la identidad de marca.
- Priorizando la claridad y la intuición: Se prestó especial atención a la organización de la información, la distribución de los elementos y la selección de colores y tipografías. El objetivo principal fue crear una interfaz que fuera fácil de usar e intuitiva, guiando al usuario de manera natural a través de las funcionalidades de la aplicación.

Desarrollo de las clases Vista:

- Traduciendo el diseño en código: Con base en los diseños creados, se implementaron las clases pertenecientes a la capa Vista en el código fuente de la aplicación. Se siguieron los principios de programación orientada a objetos, encapsulando la lógica y el comportamiento de cada elemento visual en clases independientes.
- Estableciendo la comunicación entre interfaces y lógica: Estas clases Vista se encargan de la representación visual de la información y la

interacción con el usuario. Actúan como un puente entre la lógica del negocio (Modelo) y la experiencia del usuario, traduciendo los datos y las acciones en elementos gráficos y viceversa.

Diseño del logo: Se creó un logo que represente la identidad de la aplicación, siguiendo las directrices de marca establecidas. El logo debe ser atractivo, memorable y transmitir el mensaje correcto a los usuarios.

TERMINADO SPRINT 2			
Logo y paleta color	2024-04-02 11:26:33	2024-04-13 12:26:33	Adrián Arcones
Crear base de datos	2024-04-02 08:17:33	2024-04-13 13:17:33	Rubén Peña, Daniel Correa Villa
Añadir trello y github a documentación	2024-04-02 21:28:40	2024-04-13 12:28:40	Ainhoa Blanca, Adrián Arcones
Salud a los participantes (Documentación)	2024-04-02 19:51:28	2024-04-13 22:51:28	Rubén Peña, Daniel Correa Villa, Ainhoa Blanca
Casos de uso	2024-04-02 09:07:23	2024-04-13 14:07:23	Ainhoa Blanca
Diseño de interfaz Java	2024-04-02 15:52:25	2024-04-13 22:52:25	Rubén Peña, Daniel Correa Villa, Adrián Arcones
Creación de las clases Java	2024-04-02 21:46:27	2024-04-13 23:46:27	Rubén Peña, Daniel Correa Villa, Ainhoa Blanca

Sprint #3: 15 de abril - 28 abril

Con base en el diagrama de clases, se procedió a la creación de las clases pertenecientes a dos capas esenciales de la aplicación:

- Clases Modelo:

- Estas clases representan la información central del sistema, encapsulando los datos y la lógica que rige su comportamiento. Se diseñaron cuidadosamente para reflejar los objetos del mundo real que

la aplicación gestiona, asegurando una representación precisa y consistente de la información.

- Clases Control:

- Estas clases actúan como intermediarias entre la interfaz de usuario (Vista) y el Modelo, encargándose de la gestión de la interacción con la base de datos (BBDD) y del control del flujo de la aplicación. Se implementaron siguiendo los principios de diseño orientado a objetos, promoviendo la reusabilidad y la separación de responsabilidades.

TERMINADO SPRINT 3			
Generación diagrama de clases	2024-04-13 18:00:10	2024-04-28 08:00:10	Ainhoa Blanca
Corrección vistas java sprint 2	2024-04-13 18:20:10	2024-04-28 23:20:10	Adrián Arcones
Corrección BBDD sprint 2	2024-04-13 09:25:38	2024-04-28 15:25:38	Rubén Peña
Menú funcionando Java	2024-04-13 12:02:30	2024-04-28 13:02:30	Rubén Peña
Creación de las clases pertenecientes al Control encargadas del acceso a BBDD y del manejo de la aplicación	2024-04-13 12:27:24	2024-04-28 13:27:24	Rubén Peña, Daniel Correa Villa
Corrección menu java para diferenciar master o no	2024-04-13 16:40:45	2024-04-28 22:40:45	Rubén Peña

Sprint #4: 29 de abril - 12 mayo

En el Sprint #4, nos hemos centrado en el refinamiento del programa, la realización de pruebas, la documentación y la integración de todos los elementos. Se han llevado a cabo las siguientes actividades:

Refinamiento del programa:

Se ha revisado y mejorado el código fuente de la aplicación, corrigiendo errores, optimizando el rendimiento y aplicando las mejores prácticas de programación. Se ha puesto especial atención a la legibilidad, la mantenibilidad y la robustez del código.

Realización de pruebas JUnit. Documentación de la aplicación JavaDoc:

Se han realizado pruebas unitarias utilizando JUnit para verificar el correcto funcionamiento de las diferentes partes del código. Además, se ha generado documentación detallada de la aplicación utilizando JavaDoc, proporcionando información sobre las clases, métodos y atributos del código.

Integración de todos los elementos y solución de las incidencias de integración que se produzcan:

Se han integrado todos los componentes de la aplicación, incluyendo la interfaz de usuario, la lógica del negocio y la base de datos. Se han solucionado las incidencias de integración que se han producido durante el proceso.

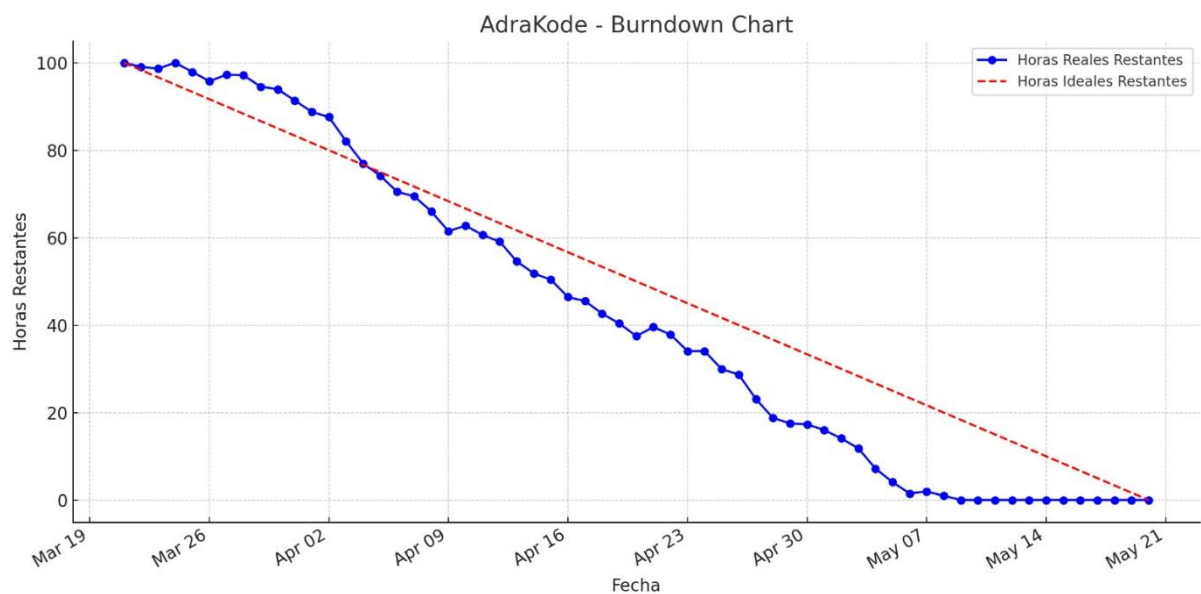
Realización del manual de usuario en la wiki de GitHub:

Se ha elaborado un manual de usuario completo en la wiki de GitHub, explicando en detalle la instalación, configuración y uso de la aplicación. El manual incluye capturas de pantalla, ejemplos y una sección de preguntas frecuentes.

Coordinación del equipo:

- Se siguieron las mismas prácticas de coordinación que en los sprints anteriores.
- Se realizaron pruebas de integración para asegurar que las nuevas funcionalidades funcionaban correctamente con el resto del sistema.

TERMINADO SPRINT 4				
Implementar base de datos en Java	2024-04-29 13:06:40	2024-05-11 13:06:40	Rubén Peña, Daniel Correa Villa	
Creación de botones funcionales	2024-04-29 01:16:50	2024-05-11 18:16:50	Rubén Peña, Daniel Correa Villa, Ainhoa Blanca, Adrián Arcones	
Creación del Javadoc	2024-04-29 20:16:50	2024-05-11 21:16:50	Adrián Arcones, Rubén Peña	
Creación del manual de usuario en GitHub	2024-04-29 17:22:43	2024-05-11 10:22:43	Daniel Correa Villa	
Terminar consultas de las vistas	2024-04-29 13:32:36	2024-05-11 16:32:36	Rubén Peña, Daniel Correa Villa	
Pruebas JUnit	2024-04-29 16:46:57	2024-05-11 11:46:57	Rubén Peña, Daniel Correa Villa, Adrián Arcones, Ainhoa Blanca	
Documentación	2024-04-29 23:22:44	2024-05-11 10:22:44	Ainhoa Blanca	
Creación vistas faltantes	2024-04-29 23:33:00	2024-05-11 17:12:05	Adrián Arcones	



Enlace a Git: <https://github.com/ruben170305/AdraCode.git>

6. Resultados y conclusiones

Resultados

El proyecto "AdraKode" ha alcanzado los objetivos propuestos al inicio, proporcionando una herramienta eficiente para la gestión de personajes y partidas en el club de rol de la UEM. A continuación, se resumen los principales logros obtenidos:

- **Implementación Completa del Sistema:** Se desarrolló una aplicación robusta utilizando Java para la lógica de negocio y MySQL para la gestión de bases de datos. La aplicación sigue el patrón Modelo-Vista-Controlador (MVC), lo que ha permitido una clara separación de responsabilidades y una mejor organización del código.
- **Gestión de Personajes y Partidas:** La aplicación permite a los usuarios crear, modificar y eliminar personajes y partidas. Los Game Masters pueden gestionar las partidas, incluyendo la creación y edición de sesiones, mientras que los jugadores pueden manejar sus propios personajes, asignarles habilidades y ver las partidas en las que participan.
- **Interfaz de Usuario Intuitiva:** Se diseñaron interfaces gráficas atractivas e intuitivas utilizando Eclipse. Esto facilitó la interacción de los usuarios con la aplicación, mejorando la experiencia de uso.
- **Trabajo Colaborativo y Control de Versiones:** El uso de Git y GitHub permitió una gestión eficiente del código fuente, facilitando la colaboración entre los miembros del equipo y el control de versiones del software.
- **Pruebas y Documentación:** Se llevaron a cabo pruebas exhaustivas utilizando JUnit para asegurar el correcto funcionamiento del software. Además, se generó documentación detallada con JavaDoc y un manual de usuario en la wiki de GitHub para facilitar la instalación, configuración y uso de la aplicación.

Conclusiones

- El desarrollo del proyecto "AdraKode" ha sido una experiencia enriquecedora que ha permitido al equipo aplicar conocimientos teóricos en un entorno práctico, enfrentando y superando diversos desafíos. Entre las principales conclusiones destacan:
- **Eficiencia y Organización:** La aplicación del patrón MVC y el uso de metodologías ágiles como Scrum han demostrado ser estrategias efectivas para el desarrollo de software, permitiendo una organización clara del trabajo y una entrega incremental de funcionalidades.
- **Colaboración y Comunicación:** El trabajo colaborativo y la comunicación efectiva fueron fundamentales para el éxito del proyecto. Las reuniones regulares y la distribución equitativa de responsabilidades facilitaron la coordinación del equipo y la resolución conjunta de problemas.
- **Adaptación y Flexibilidad:** La metodología ágil permitió al equipo adaptarse a cambios y ajustar prioridades según las necesidades del proyecto, lo que resultó en un desarrollo más dinámico y eficiente.
- **Aprendizaje Continuo:** Cada miembro del equipo ha adquirido habilidades valiosas en áreas como la programación, diseño de bases de datos, pruebas de software y gestión de proyectos. Esta experiencia ha fortalecido sus competencias y preparado mejor para futuros desafíos en el ámbito profesional.
- En resumen, "AdraKode" no solo ha cumplido con sus objetivos técnicos, sino que también ha proporcionado una plataforma para el desarrollo profesional y personal de sus integrantes, demostrando la importancia de la colaboración y la innovación en el desarrollo de software.

7. Trabajos futuros

A medida que la aplicación va evolucionando, es importante tener mejoras que puedan incrementar su funcionalidad y atractivo. Los proyectos futuros no solo mejorarán las eficiencias del sistema, sino que también enriquecerán la experiencia de los usuarios, fomentando una mayor participación y de esta manera contribuirá al crecimiento de nuestro club.

Por lo tanto, los proyectos que tenemos en mente son los siguientes:

- Inserción de fotos a los personajes: Los jugadores podrán dar vida a sus personajes mediante imágenes, aumentando la inmersión y la conexión emocional con ellos.

Las fotos facilitarán la identificación rápida y visual de los personajes, tanto para los jugadores como para los Game Masters.

El añadir imágenes hará que la interfaz sea más atractiva y moderna, mejorando la experiencia del usuario.

Los usuarios podrán subir imágenes desde sus dispositivos para asociarlas a sus personajes, además se implementará una galería donde los usuarios podrán seleccionar y gestionar las fotos de sus personajes. Por otro lado, antes de confirmar la subida, los usuarios podrán ver una vista previa de la imagen para asegurarse de que es la correcta.

Cabe destacar que el sistema soportará múltiples formatos de imagen (JPG, PNG, GIF).

Como es una función que puede llegar a dar problemas, se tendrá que poner atención a los siguientes puntos:

- Asegurar suficiente espacio en el servidor para almacenar las imágenes de todos los personajes.
- Implementar medidas de seguridad para evitar la subida de contenido inapropiado o malicioso.
- Optimizar el tamaño y la carga de las imágenes para mantener un rendimiento óptimo de la aplicación.

Anexos

Anexo I - Listado de requisitos de la aplicación

Incluir el listado de requisitos de programación y BD necesarios para garantizar el correcto funcionamiento de la aplicación.

Este documento establece los requisitos de programación y base de datos (BD) necesarios para el correcto funcionamiento de la aplicación. Se han añadido nuevos requisitos específicos para la interfaz de usuario (Vista), tomando en cuenta las diferentes funcionalidades y roles de los usuarios.

1. Requisitos de programación

1.1 Interfaz de usuario (Vista)

- La aplicación deberá implementar una pantalla de login donde los usuarios puedan ingresar su nombre de usuario y contraseña.
- Solo se permitirá el acceso a usuarios registrados.
- Se implementarán mecanismos de recuperación de contraseña en caso de olvido.
- Al iniciar sesión, el usuario deberá seleccionar si desea acceder como jugador o como Game Master (GM).
- La interfaz mostrará opciones y funcionalidades diferentes según el rol seleccionado.
- Se implementará un menú principal con las opciones de consultas, alta, baja y modificación.
- Las opciones disponibles en el menú dependerán del rol del usuario (jugador vs. Game Master).

Funcionalidades por rol:

Jugador:

- El jugador podrá editar y eliminar sus propios personajes.
- El jugador podrá crear nuevos personajes.
- El jugador podrá consultar cualquier personaje en el sistema.
- El jugador podrá consultar cualquier partida, pero no editarla.

Game Master (GM):

- El GM podrá consultar cualquier partida.
- El GM podrá editar las partidas que esté dirigiendo (para cambiar sus datos).
- El GM podrá crear nuevas partidas.
- Las nuevas partidas se crearán con 0 sesiones por defecto y marcadas como "en curso".

1.2 Lógica del negocio (Modelo)

- Se implementará la lógica necesaria para gestionar los diferentes roles de usuario (jugador y GM).
- Se implementará la lógica para el control de acceso a las diferentes funcionalidades de la aplicación, según el rol del usuario.
- Se implementará la lógica para la gestión de personajes (alta, baja, modificación y consulta).
- Se implementará la lógica para la gestión de partidas (alta, baja, modificación y consulta).

1.3 Control de acceso y seguridad

- Se implementarán mecanismos de autenticación y autorización para controlar el acceso a la aplicación.
- Los datos de los usuarios se almacenarán de forma segura y protegida.
- Se implementarán medidas de seguridad para evitar ataques a la aplicación.

1.4 Integración con la base de datos

- La aplicación se conectará a la base de datos de manera eficiente y segura.
- Se implementarán consultas SQL optimizadas para recuperar y manipular datos de la base de datos.
- Las transacciones de base de datos se gestionarán de manera confiable.

1.5 Manejo de errores

- Se implementarán mecanismos para detectar y manejar errores de manera adecuada.
- Se proporcionará información útil al usuario en caso de error.
- Los errores se registrarán en un archivo de registro para facilitar su análisis y resolución.

2. Requisitos de base de datos (BD)

2.1 Modelo de datos

- Se diseñará un modelo de datos normalizado que refleje la estructura de la información de la aplicación.
- Se definirán las relaciones entre las entidades del modelo de datos.
- Se asegurará la integridad y consistencia de los datos.

2.2 Almacenamiento de datos

- Los datos se almacenarán de forma eficiente y segura en la base de datos.
- Se implementarán mecanismos de copia de seguridad y recuperación de datos.
- Se optimizará el espacio de almacenamiento en la base de datos.

2.3 Rendimiento de la base de datos

- Se optimizarán las consultas SQL para mejorar el rendimiento de la base de datos.
- Se implementarán mecanismos de caché para reducir el tiempo de respuesta de las consultas.
- La base de datos se escalará para manejar el volumen de datos y tráfico esperado.

2.4 Seguridad de la base de datos

- La base de datos se protegerá contra accesos no autorizados.
- Los datos sensibles almacenados en la base de datos se encriptarán.
- Se implementarán mecanismos de auditoría para registrar el acceso a la base de datos.

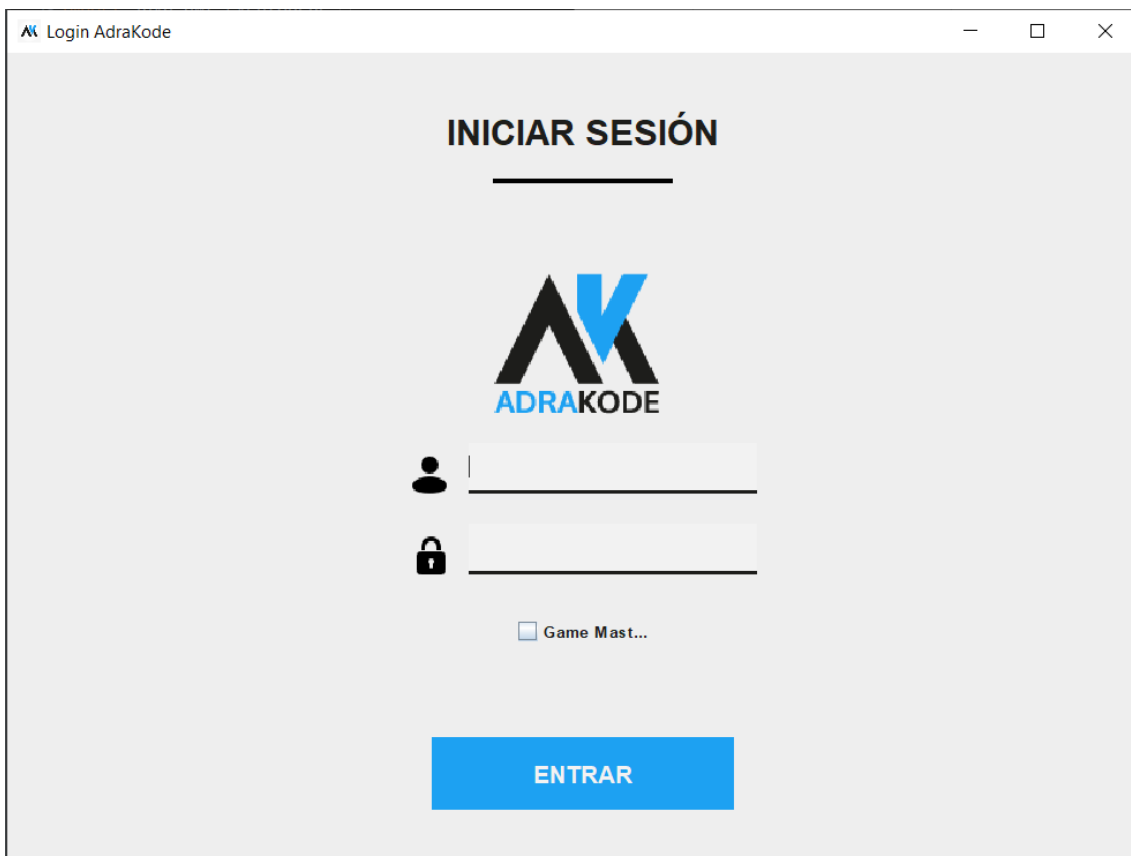
Anexo II – Guía de uso de la aplicación

Este documento sirve como guía de usuario para la aplicación, detallando los pasos necesarios para navegar por sus funcionalidades y realizar las acciones deseadas. Se incluyen capturas de pantalla para ilustrar el proceso y facilitar su comprensión.

1. Acceso a la aplicación

1.1. Inicio de sesión

Paso 1: Acceder a la página de inicio de sesión de la aplicación.



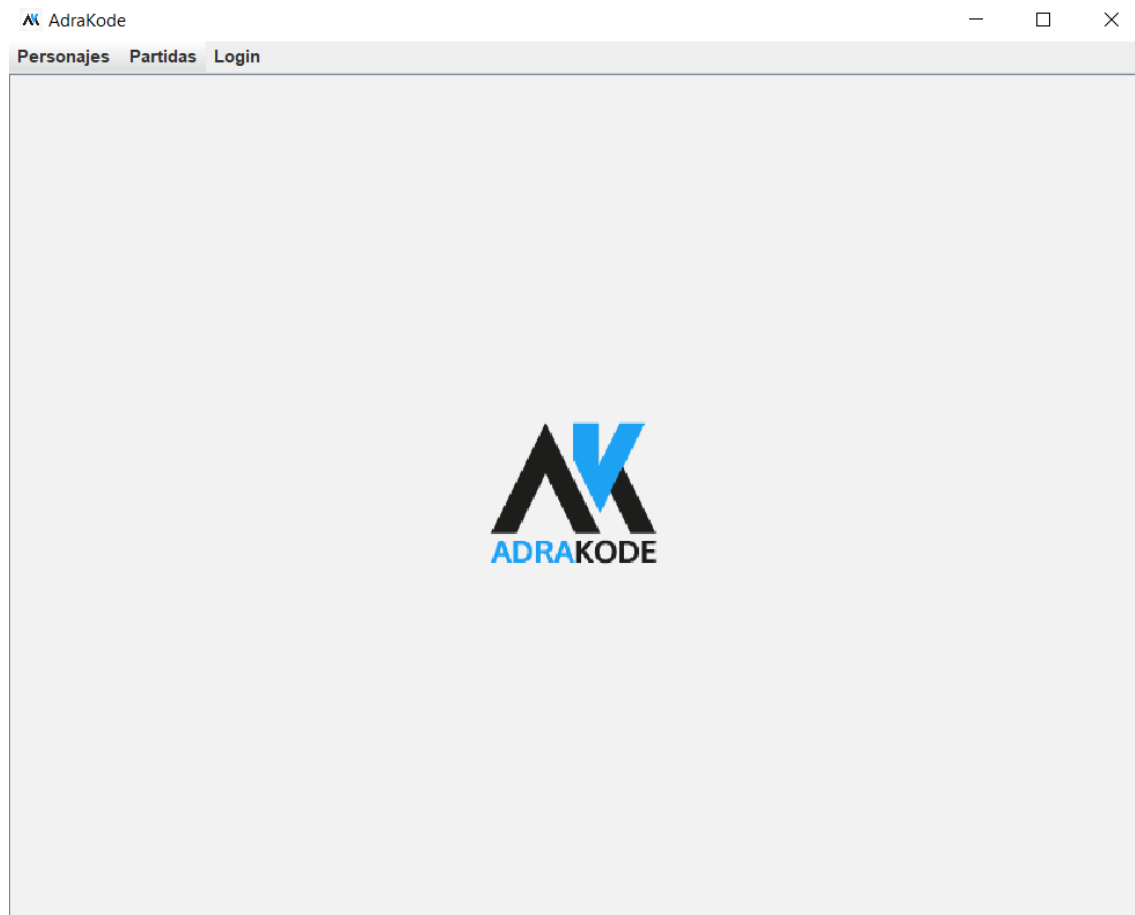
Paso 2: Ingresar el nombre de usuario y contraseña en los campos correspondientes. En este caso el nombre de usuario es Bob y la contraseña pass.

Paso 3: Si quiere iniciar sesión como Game Master clique en el cuadrado donde pone Game Master. Si quiere iniciar sesión como usuario normal simplemente dele a entrar.

Paso 4: Hacer clic en el botón "ENTRAR" para acceder a la aplicación.

2. Funcionalidades para jugadores

Una vez iniciado sesión nos aparecerá el home de la aplicación y en la parte superior derecha el menú donde podrás elegir entre las diferentes opciones.



2.1. Creación de personajes


Si quieres crear un personaje, en el menú clicka a personajes y luego a nuevos personajes. Una vez clickado te llevará automáticamente a la ventana de crear personaje y ahí podrás poner el nombre a tu personaje, la raza y la clase además de que podrás asignarle las habilidades que creas convenientes para tu personaje.

Si ves que la habilidad experiencia no te deja modificarla es porque esta opción solo está habilitada para el Game Master.

Adrakode








Personajes Partidas Login


CREAR PERSONAJE



Raza

Clase

	Experiencia	<input type="text" value="0"/>
	Fuerza	<input type="text" value="0"/>
	Destreza	<input type="text" value="0"/>
	Constitución	<input type="text" value="0"/>
	Inteligencia	<input type="text" value="0"/>
	Sabiduría	<input type="text" value="0"/>
	Carisma	<input type="text" value="0"/>

CREAR 


2.2. Ver Personaje

Si quieres ver los personajes que has creado, ve a la barra del menú y en personajes clicka Ver Personaje.

Ahí podrás ver los personajes que has creado y además te aparecen dos opciones para editar el personaje y borrarlo en caso de que ya no lo quieras utilizar más.


VER PERSONAJES


PERSONAJE Jugador 1




Raza
Clase


Experiencia 0 %

Fuerza 0 %

Destreza 0 %


Constitución 0 %

Inteligencia 0 %

Sabiduría 0 %

Carisma 0 %





SELECCIONAR

2.3. Modificar Personaje

Aunque en ver personaje tienes la opción de editado y borrado, metiéndote de nuevo en personajes en la barra del menú y luego modificar personajes, tendrás las mismas opciones.

EDITAR PERSONAJE

PERSONAJE 1



Experiencia	0
Fuerza	0
Destreza	0
Constitución	0
Inteligencia	0
Sabiduría	0
Carisma	0

Raza _____

Clase _____

GUARDAR



2.4. Ver Partidas

Si quieres ver las partidas en las que estas metido y además saber si siguen en curso o han finalizado, deberás irte a partidas en la barra de menú, clickarle y automáticamente te llevará a la ventana.

VER PARTIDAS

Partida 1



 Anfitrión 1

 4

 30'

 13-04 16:00 pm

 En curso

ID	Anfitrión	Jugadores	Duración	Fecha	Estado
Partida 1	Usuario 1	4	30'	13-04 16:00 pm	En curso
Partida 2	Usuario 2	3	15'	16-04 15:00 pm	En espera
Partida 3	Usuario 5	2	25'	12-04 11:00 am	Finalizada
Partida 2	Usuario 2	3	15'	16-04 15:00 pm	En espera
Partida 3	Usuario 5	2	25'	12-04 11:00 am	Finalizada
Partida 2	Usuario 2	3	15'	16-04 15:00 pm	En espera

JUGAR

3. Login

Si quieres volver al login para iniciar sesión, deberás ir a la barra del menú y clickar a login.

4. Game Master


Entrando a la aplicación como Game Master podrás realizar las mismas operaciones que el usuario y además podrás hacer las siguientes:


4.1. Crear partidas


Podrás crear tus propias partidas yendo a la opción partidas en el menú y luego accede a crear partida. A continuación, te saldrá la siguiente ventana en la que deberás rellenar los campos y se guardarán automáticamente presionando el botón crear.


CREAR PARTIDA


Partida: _____




 Anfitrión _____


 Jugadores _____

 Duración _____

 Fecha _____

 Estado _____

CREAR



4.2. Ver partidas

En esta opción podrás ver las partidas y al igual que en ver personajes podías editar y borrar, también se han integrado estas dos opciones en ver partidas, por lo tanto, para llegar a poder hacer esto, deberás irte al menú y meterte en ver partidas, ahí podrás hacer lo mencionado anteriormente.

VER PARTIDAS

Partida 1



 Anfitrión 1

 4

 30'

 13-04 16:00 pm

 En curso

ID	Anfitrión	Jugadores	Duración	Fecha	Estado	
Partida 1	Usuario 1	4	30'	13-04 16:00 pm	En curso	▲
Partida 2	Usuario 2	3	15'	16-04 15:00 pm	En espera	
Partida 3	Usuario 5	2	25'	12-04 11:00 am	Finalizada	
Partida 2	Usuario 2	3	15'	16-04 15:00 pm	En espera	
Partida 3	Usuario 5	2	25'	12-04 11:00 am	Finalizada	
Partida 2	Usuario 2	3	15'	16-04 15:00 pm	En espera	▼





JUGAR

4.3. Editar partida

El Game Master puede editar las partidas, por lo tanto, para llegar a esta opción se tendrá que ir a la opción partidas y allí clickar a editar partidas.

Una vez editadas las partidas solo tendrás que dar al botón guardar y se guardarán automáticamente.

EDITAR PARTIDA

Partida: _____



 Anfitrión _____

 Jugadores _____

 Duración _____

 Fecha _____

 Estado _____

Jugador	Raza	Clase	
Jugador 1	Raza 1	Clase 1	^
Jugador 2	Raza 2	Clase 2	
Jugador 3	Raza 3	Clase 3	
Jugador 4	Raza 4	Clase 4	
Jugador 5	Raza 5	Clase 5	
Jugador 6	Raza 6	Clase 6	v

GUARDAR



Por último, para cerrar la aplicación tendrás que dar a la “X” y te saldrá un mensaje de verificación para saber si realmente quieres cerrar la aplicación.



Anexo III

Creación de la inserción de datos de la partida a la base de datos a través de la aplicación.

```
public void crearPartidaBBDD() {
    String sql = "INSERT INTO partida (nombre, duracion, dificultad, fecha, numero_jugadores, ambientacion)"
        + " VALUES (?, ?, ?, ?, ?, ?)";

    Model mysql = new Model();
    Connection conn = mysql.get_connection();
    PreparedStatement ps = null;

    try {
        ps = conn.prepareStatement(sql);
        ps.setString(1, cp.getTxtNombrePartida().getText());
        ps.setInt(2, Integer.parseInt(cp.getTxtDuracion().getText()));
        ps.setInt(3, Integer.parseInt(cp.getTxtDificultad().getText()));
        ps.setString(4, cp.getTxtFecha().getText());
        ps.setInt(5, Integer.parseInt(cp.getTxtJugadores().getText()));
        ps.setString(6, cp.getTxtAnfitrión().getText());

        ps.executeUpdate();
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (Exception e) {
        // TODO: handle exception
        menu.mostrarMensajeRellenaCampos();
    } finally {
        try {
            ps.close();
            conn.close();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
        }
    }
}
```

Creación de la inserción de datos de los personajes a la base de datos a través de la aplicación.

```
// Recupera los valores de los campos
String insert = "INSERT INTO personaje (nombre, personaje, raza, clase, expe, fuerza, destreza, constitucion, inteligencia, sabiduria, carisma, cod_miembro)"
    + " VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
try (Connection conn = mysql.getConnection(); PreparedStatement pstmt = conn.prepareStatement(insert);) {

    // Recupera los valores de los campos
    String nombre = cPersonaje.getLblSeleccionarPersonaje().getText();
    String personaje = cPersonaje.getLblSeleccionarPersonaje().getText();
    String raza = cPersonaje.getTxtRaza().getText();
    String clase = cPersonaje.getTxtClase().getText();
    Integer expe = (Integer) cPersonaje.getSpinnerExperiencia().getValue();
    Integer fuerza = (Integer) cPersonaje.getSpinnerFuerza().getValue();
    Integer destreza = (Integer) cPersonaje.getSpinnerDestreza().getValue();
    Integer constitucion = (Integer) cPersonaje.getSpinnerConstitucion().getValue();
    Integer inteligencia = (Integer) cPersonaje.getSpinnerInteligencia().getValue();
    Integer sabiduria = (Integer) cPersonaje.getSpinnerSabiduria().getValue();
    Integer carisma = (Integer) cPersonaje.getSpinnerCarisma().getValue();
    Integer cod_miembro = user.getUser_id();

    // Verifica que los campos obligatorios no estén vacíos o nulos
    if (nombre == null || nombre.isEmpty() ||
        personaje == null || personaje.isEmpty() ||
        raza == null || raza.isEmpty() ||
        clase == null || clase.isEmpty() ||
        expe == null ||
        fuerza == null ||
        destreza == null ||
        constitucion == null ||
        inteligencia == null ||
        sabiduria == null ||
        carisma == null ||
        cod_miembro == null) {

        throw new IllegalArgumentException("Todos los campos obligatorios deben ser completados.");
    }

    // Ejecuta la inserción
    pstmt.executeUpdate();
} catch (SQLException sqle) {

} catch (IllegalArgumentException e) {
    // TODO: handle exception
    menu.mostrarMensajeRellenaCampos();
}
```

Captura de la consulta para realizar actualizaciones a los personajes desde

```
public void db_update() {

    // Inicializamos la conexión con MySQL
    Model mysql = new Model();
    String sql = "UPDATE personaje "
        + "SET clase=?, raza=?, expe=?, fuerza=?, destreza=?, constitucion=?, inteligencia=?, sabiduria=?, carisma=?, cod_miembro=? "
        + "WHERE cod=?";
    Connection conn = mysql.get_connection();

    try {
        // Consulta para actualizar la tabla
        PreparedStatement pstmt = conn.prepareStatement( sql );
        pstmt.setString( 1, ep.getTxtClase().getText() );
        pstmt.setString( 2, ep.getTxtRaza().getText() );
        pstmt.setInt(3, (int) ep.getSpinnerExperiencia().getValue());
        pstmt.setInt(4, (int) ep.getSpinnerFuerza().getValue());
        pstmt.setInt(5, (int) ep.getSpinnerDestreza().getValue());
        pstmt.setInt(6, (int) ep.getSpinnerConstitucion().getValue());
        pstmt.setInt(7, (int) ep.getSpinnerInteligencia().getValue());
        pstmt.setInt(8, (int) ep.getSpinnerSabiduria().getValue());
        pstmt.setInt(9, (int) ep.getSpinnerCarisma().getValue());
        pstmt.setInt(10, (int) user.getUser_id());
        pstmt.setInt(11, Integer.parseInt(ep.getLblId().getText()));

        pstmt.executeUpdate();

    } catch ( SQLException sqle ) {
        sqle.printStackTrace();
    }
}

/**
 * Metodo que permite subir imagenes
 */
public void subirImagen() {
    menu.mostrarMensajeConstruccion();
}
}
```

EditarPersonaje.

Parte de código el cual nos permite entrar a la aplicación.

```
public void actionPerformed( ActionEvent e ) {
    Object source = e.getSource();

    if ( source instanceof JCheckBox ) {

        // Capturamos el valor del Checkbox
        JCheckBox checkBox = ( JCheckBox ) source;
        if ( "gamemaster".equals( checkBox.getName() ) )
            esMaster = checkBox.isSelected();

        // Si se ordena iniciar sesión, comparamos valores
    } else if ( source instanceof JButton && e.getActionCommand().equals( "ENTRAR" ) ) {

        try {

            // Capturamos los valores de los inputs
            user = login.getUser();
            String user_name = login.getUsuario().getText();
            String password = login.getContraseña().getText();

            // Si el usuario y la contraseña son correctos, continuamos
            if ( user.login( user_name, password, user ) ) {

                login.dispose();

                // Llamamos al metodo arranque para que cree la ventana menu y nos devuelva esa ventana
                // Posteriormente, mostramos la ventana
                this.menu = MenuMain.boot( esMaster );
                menu.make_visible();
            } else {
                login.mostrarMensajeErrorLogin();
            }

        } catch ( SQLException sqle ) {
            sqle.printStackTrace();
        }
    }
}
```


Clase menú que nos permitirá desplazarnos entre las diferentes opciones.

```
public class PMenuListener implements ActionListener {

    private Menu ventana;
    private Home home;
    private views.CrearPersonaje cPersonaje;
    private views.EditarPersonaje ePersonaje;
    private views.VentanaPrincipalLogin login;
    private VerPersonajes vPersonajes;
    private CrearPartida cPartida;
    private EditarPartida ePartida;
    private VerPartidas vPartidas;
    private VerPartidasMaster vPartidasMaster;

    public PMenuListener(
        Menu ventana
        , Home home
        , CrearPersonaje cPersonaje
        , EditarPersonaje ePersonaje
        , VerPersonajes vPersonajes
        , CrearPartida cPartida
        , EditarPartida ePartida
        , VerPartidas vPartidas
        , VerPartidasMaster vPartidasMaster
        , VentanaPrincipalLogin login
    ) {
        this.ventana          = ventana;
        this.home              = home;
        this.cPersonaje        = cPersonaje;
        this.ePersonaje        = ePersonaje;
        this.vPersonajes       = vPersonajes;
        this.cPartida          = cPartida;
        this.ePartida          = ePartida;
        this.vPartidas         = vPartidas;
        this.vPartidasMaster   = vPartidasMaster;
        this.login             = login;
    }
}
```

```
public void actionPerformed((ActionEvent ae) {  
    if ( ae.getSource() instanceof JMenuItem ) {  
  
        // Dependiendo de la instrucción, mostramos una vista u otra  
        switch ( ae.getActionCommand() ) {  
            case "Nuevo personaje":  
                ventana.cargarPanel( cPersonaje );  
                break;  
  
            case "Modificar personaje":  
                ventana.cargarPanel( ePersonaje );  
                break;  
  
            case "Ver personajes":  
                vPersonajes.cargarDatosEnComboBox();  
                ventana.cargarPanel( vPersonajes );  
                break;  
  
            case "Crear partida":  
                ventana.cargarPanel( cPartida );  
                break;  
  
            case "Ver partidas":  
                ventana.cargarPanel( vPartidas );  
                break;  
  
            case "Ver partidas Master":  
                ventana.cargarPanel( vPartidasMaster );  
                break;  
  
            case "Login":  
                ventana.dispose();  
                ventana.cargarPanel( home );  
                login.make_visible();  
                break;  
  
            case "Salir":  
                ventana.mostrarMensajeConfirm();  
                break;  
  
            default:  
                break;  
        }  
    }  
}
```

Consulta que permitirá actualizar a los personajes desde VerPersonaje.

```
public void update_data() {
    Model mysql = new Model();
    mysql.get_connection();

    String update = "UPDATE personaje SET nombre = ?, personaje = ?, raza = ?, clase = ?, expe = ? WHERE cod_miembro = ? AND id_personaje = ?";
    try (Connection conn = mysql.get_connection(); PreparedStatement pstmt = conn.prepareStatement(update)) {
        pstmt.setString(1, cPersonaje.getLblSeleccionarPersonaje().getText());
        pstmt.setString(2, cPersonaje.getLblSeleccionarPersonaje().getText());
        pstmt.setString(3, cPersonaje.getTxtRaza().getText());
        pstmt.setString(4, cPersonaje.getTxtClase().getText());
        pstmt.setInt(5, 0); // Si 'expe' es un entero, debes definir cómo se obtiene el valor
        pstmt.setInt(6, user.getUser_id());
        pstmt.setInt(7, personaje.getCod());

    } catch (SQLException sqle) {
        sqle.printStackTrace();
    }
}

public void delete_data() {
    Model mysql = new Model();
    mysql.get_connection();

    try {
        String delete = "DELETE FROM personaje WHERE cod_miembro = ? AND cod = ?";
        Connection conn = mysql.get_connection();
        PreparedStatement pstmt = conn.prepareStatement(delete);
        pstmt.setInt(1, user.getUser_id());
        pstmt.setInt(2, Integer.parseInt(vPersonajes.getIdLbl().getText()));

        pstmt.executeUpdate();
    } catch (SQLException sqle) {
        sqle.printStackTrace();
    }
}
```

51