Logotipo

Descripción generada automáticamente

DOCUMENTO DAS ARQUITECTURA DE SOFTWARE PROYECTO

Documento de Arquitectura de Software IEEE-1471-2000 Control del documento

Integrantes: Fabian Gonzalez, Bastián Venegas

• Nombre Proyecto: HollowMates

Nombre Sub Proyecto: AppSouls

Arquitectura Proyecto Semestral



Contenido

[**1.**](#_heading=h.gjdgxs) **Introducción** 3

[**2.**](#_heading=h.30j0zll) **Propósito** 3

[**3.**](#_heading=h.1fob9te) **Alcance** 3

[**4.**](#_heading=h.3znysh7) **Equipo de Trabajo-Actores del Desarrollo** 4

[**5.**](#_heading=h.2et92p0) **Recomendaciones de conformidad con esta práctica.** 4

[Para asegurar que la arquitectura del sistema cumpla con los requisitos y estándares establecidos, se recomiendan las siguientes prácticas: 4](#_heading=h.tyjcwt)

[1. Implementar revisiones de arquitectura cada xxxx para asegurar que se ajusten a los cambios de requisitos. 4](#_heading=h.3dy6vkm)

[2. Utilizar patrones de diseño desacoplados como MVC para modularidad y escalabilidad o por capas. 4](#_heading=h.1t3h5sf)

[3. Adoptar metodologías ágiles como Scrum para la entrega continua de mejoras y funcionalidades. 4](#_heading=h.4d34og8)

[**6.**](#_heading=h.2s8eyo1) **Referencias y estándares aplicables a este documento:** 5

[**7.**](#_heading=h.17dp8vu) **Definiciones, acrónimos y abreviaciones.** 5

[**8.**](#_heading=h.3rdcrjn) **DESCRIPCIÓN DE ARQUITECTURA:** La arquitectura del sistema se presenta mediante el uso de vistas basadas en el Modelo 4+1 de Kruchten, con el fin de proporcionar una visión completa y estructurada del sistema desde diferentes perspectivas. La arquitectura del sistema se presenta mediante el uso de vistas basadas en el Modelo 4+1 de Kruchten, con el fin de proporcionar una visión completa y estructurada del sistema desde diferentes perspectivas. 6

[8.1 VISTAS: 6](#_heading=h.26in1rg)

[8.2 TIPOS DE VISTAS: 6](#_heading=h.lnxbz9)

[8.3 Framework Conceptual 7](#_heading=h.35nkun2)

[8.3.1 Descripción de la arquitectura en contexto 7](#_heading=h.1ksv4uv)

[8.3.1 Actores Usuarios y sus roles: 8](#_heading=h.44sinio)

[8.3.2 Actividades de arquitectura en el ciclo de vida 9](#_heading=h.2jxsxqh)

[8.3.3 Descripciones prácticas de arquitectura. 9](#_heading=h.z337ya)

[8.4 Documentación de la arquitectura 10](#_heading=h.3j2qqm3)

[8.5 Selección de puntos de vista de la arquitectura 10](#_heading=h.1y810tw)

[8.6 Vistas de la arquitectura 11](#_heading=h.4i7ojhp)

[UML-VISTA DE ESCENARIO 11](#_heading=h.2xcytpi)

[UML-VISTA LÓGICA-DIAGRAMA DE CLASES 12](#_heading=h.1ci93xb)

[UML-VISTA LÓGICA-DIAGRAMA DE BASE DE DATOS 14](#_heading=h.qsh70q)

[UML-VISTA DE PROCESO-DIAGRAMAS DE ACTIVIDAD 16](#_heading=h.1pxezwc)

[UML-VISTA DE DESPLIEGUE 18](#_heading=h.49x2ik5)

[UML-VISTA FÍSICA – DIAGRAMA INFRAESTRUCTURA 19](#_heading=h.2p2csry)

[Usabilidad y Calidad de software- Front-end -Vistas Principales 21](#_heading=h.147n2zr)

[Usabilidad y Calidad de software – Vistas específicas de usuario 22](#_heading=h.3o7alnk)

[8.7 Consistencia en la cantidad de vistas de la arquitectura. 23](#_heading=h.23ckvvd)

[8.7.1 DESCRIPCION DE MODULOS 23](#_heading=h.ihv636)

[8.7.2 DESCRIPCIÓN DE COMPONENTES 23](#_heading=h.32hioqz)

[8.7.3 DESCRIPCIÓN DE CONECTORES 23](#_heading=h.1hmsyys)

[9.1 Acceso a fuentes de información del proyecto 25](#_heading=h.41mghml)

# **Introducción**

El presente informe tiene como objetivo describir en detalle la arquitectura de un sistema unificado diseñado para centralizar y facilitar la interacción de la comunidad de la saga Dark Souls. Este sistema está orientado a automatizar la administración de tareas y procesos comunes dentro de una plataforma dedicada a los fanáticos de la saga, permitiendo a los usuarios compartir contenido, colaborar en proyectos y conectar con otros miembros de la comunidad de forma más eficiente.

El sistema propuesto busca mejorar la experiencia del usuario al integrar todas las interacciones de la comunidad en un solo lugar, resolviendo el problema de dispersión en múltiples plataformas. Será implementado en una aplicación móvil disponible en Google Play, con la posibilidad de expandirse a otras franquicias de videojuegos. El sistema también proporcionará una interfaz intuitiva y accesible para todos los usuarios, asegurando que las gestiones se realicen de manera eficiente y organizada, basada en los estándares de accesibilidad y usabilidad más actuales, potenciando tanto a usuarios casuales como a creadores de contenido y patrocinadores

# **Propósito**

El propósito de este documento es servir como guía para el diseño e implementación de la arquitectura del sistema unificado de la comunidad Dark Souls. En él se detallan las decisiones técnicas, los componentes principales y los puntos de integración que permitirán implementar una solución robusta y escalable, capaz de gestionar la interacción de múltiples usuarios y creadores de contenido bajo un esquema de administración centralizado.

Este informe es esencial para los desarrolladores y stakeholders del proyecto, ya que proporciona una visión detallada de la estructura y comportamiento del sistema, así como de las relaciones entre sus diferentes funcionalidades y componentes. Asimismo, establece las bases para futuras expansiones hacia otras franquicias de videojuegos y mejoras del sistema, teniendo en cuenta factores clave como la disponibilidad, la seguridad y la usabilidad para garantizar una experiencia óptima para todos los usuarios.

# **Alcance**

El sistema gestionará las siguientes áreas:

* Inicio de sesión de 3 tipos de usuarios distintos (Free, Premium, Admin)
* Posibilidad de elegir 1 de los 3 juegos souls para navegar en su feed propio
* Feed acorde a la entrega, con posibilidad de subir posts, comentarlos y filtrarlos por su categoría
* En la fase inicial las publicaciones se dividirán en 2, de ayudas consejos y tutoriales, y de búsqueda de jugadores
* Los usuarios premium podrán crear grupos donde se reunirán los jugadores que sean invitados y comunicarse a través de un chat grupal
* Los usuarios podrán personalizar y cambiar datos de su perfil
* Los usuarios podrán agregar como amigos a otros amigos e interactuar a través de un chat
* Al iniciar como administrador se podrán gestionar los usuarios, las publicaciones y los grupos
* Los os datos de los usuarios y las publicaciones serán guardadas en supabase, el cual usar PostgreSQL

# **Equipo de Trabajo-Actores del Desarrollo**

El desarrollo del sistema requiere la participación de un equipo multidisciplinario con roles claramente definidos para asegurar que el proyecto se lleve a cabo de manera eficiente. A continuación, se presenta una tabla con los principales roles y sus responsabilidades:

| Rol | Responsabilidades |
| --- | --- |
| Cliente-Stakeholder | * Creadores de contenido de dark souls * Twitck * Kick * Jugadores, creadores de mods, cosplaters,etc |
| Product Owner | Christan Lazcano |
| Desarrolladores Backend | Bastián Venegas |
| Desarrolladores Frontend | Bastián Venegas |
| DBA-Especialista en Base de Datos | Bastián Venegas |
| Tester/QA | Fabian Gonzalez |
| Analista de Requisitos | Fabian Gonzalez |
| Administrador Plataformas | Fabian Gonzalez |

# **Recomendaciones de conformidad con esta práctica.**

Para asegurar que la arquitectura del sistema unificado de la comunidad Dark Souls cumpla con los requisitos y estándares establecidos, se recomiendan las siguientes prácticas:

* Implementar revisiones periódicas de la arquitectura cada 1 o 2 meses para garantizar que se ajusten a las nuevas necesidades de los usuarios, la expansión hacia otras franquicias y las demandas del mercado.
* Implementar la capacidad de creación de informes por parte de los administradores. Esto con la intención de tomar mejores decisiones en el futuro
* Agregar un personal de servicio al cliente que tenga comunicación directa con los usuarios, esto con la finalidad de estar atentos de cualquier situación o complicación
* Utilizar patrones de diseño desacoplados, como el modelo MVC (Model-View-Controller) o una arquitectura por capas, para asegurar la modularidad y escalabilidad del sistema, facilitando futuras expansiones y mantenimientos.
* Adoptar metodologías ágiles, como Scrum o Kanban, para asegurar la entrega continua de mejoras y nuevas funcionalidades, manteniendo una comunicación fluida entre los desarrolladores y los stakeholders.
* Garantizar un enfoque en la seguridad y la usabilidad, adoptando las mejores prácticas en términos de autenticación, protección de datos y accesibilidad para brindar una experiencia segura y fluida a los usuarios de la comunidad.
* Monitorear el rendimiento del sistema de manera constante, utilizando herramientas de análisis para identificar posibles cuellos de botella y áreas de mejora, con el objetivo de optimizar la experiencia del usuario.

# **Referencias y estándares aplicables a este documento:**

El documento sigue los siguientes estándares de calidad para asegurar la robustez, escalabilidad y seguridad del sistema unificado de la comunidad Dark Souls:

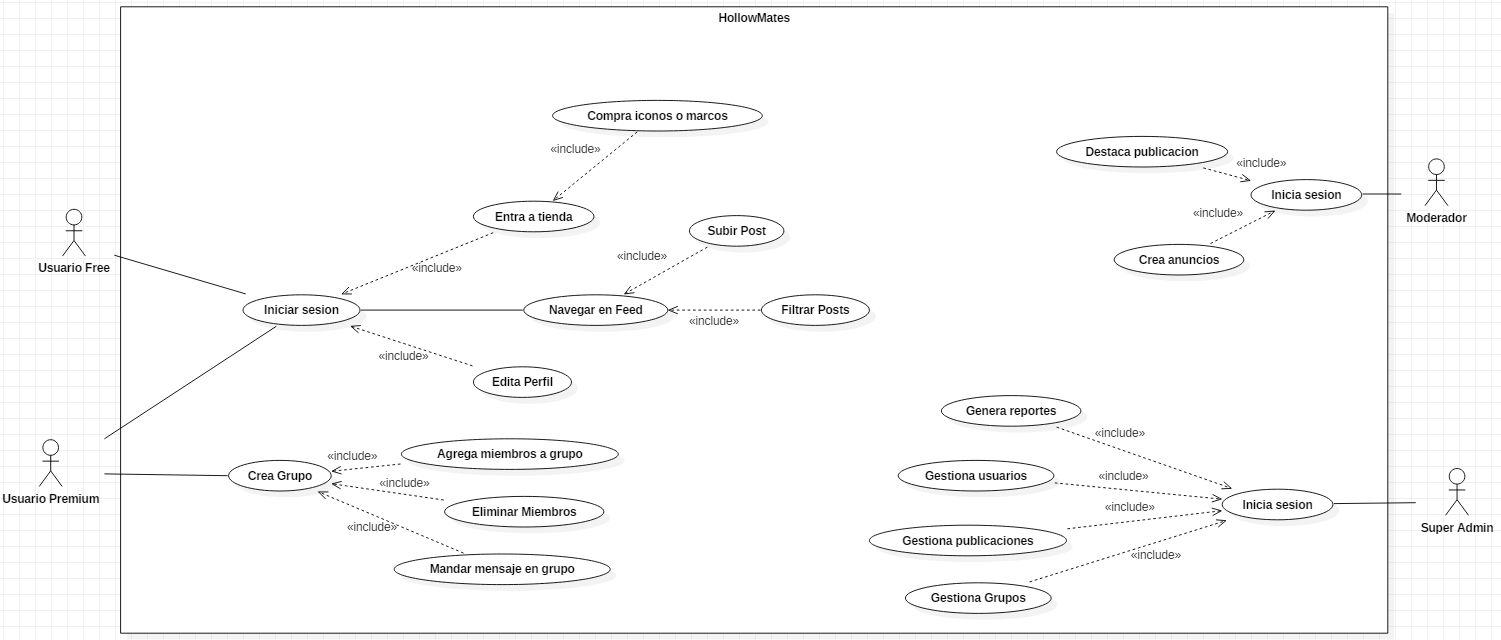
* **IEEE 830-1998 ST:** Estandarización de requisitos de software.
* **ISO 9126-2001:** Métricas de calidad del software.
* **ISO 25010:** Evalúa la calidad del software. Define criterios y métricas para medir aspectos como la funcionalidad, eficiencia, confiabilidad, usabilidad y otros factores importantes en el software.
* **Modelo 4+1 de Kruchten:** Modelo estructural basado en múltiples vistas de escenarios.
* **ATAM (Architecture Tradeoff Analysis Method):** Metodología para evaluar y documentar decisiones arquitectónicas.
* **ISO/IEC 27001:** Estándar internacional para la gestión de la seguridad de la información, asegurando la protección de los datos de los usuarios.
* **CMMI (Capability Maturity Model Integration):** Modelo para la mejora y evaluación de procesos, que ayuda a garantizar la calidad del desarrollo de software y servicios.
* **IEEE 12207-2017:** Estándar para el ciclo de vida del software, cubriendo todas las fases, desde la concepción hasta el mantenimiento, para garantizar una entrega eficiente y con calidad.
* **ISO/IEC 29110:** Un estándar diseñado para mejorar los procesos de desarrollo de software en pequeñas organizaciones, que puede ser útil si tu equipo de desarrollo es reducido.

# **Definiciones, acrónimos y abreviaciones.**

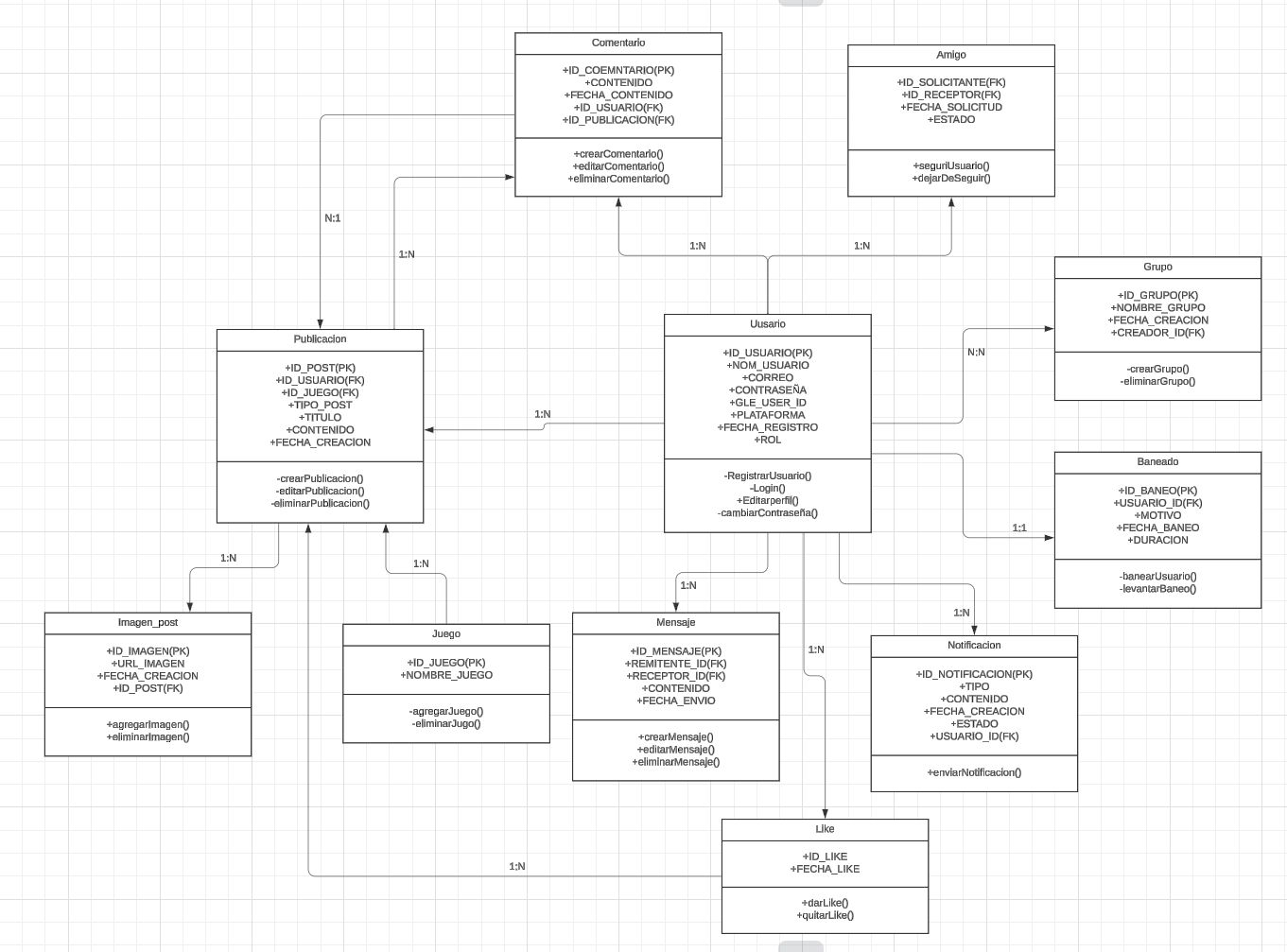
| Término | Definición |
| --- | --- |
| DAS | Documento de Arquitectura de Software |
| MVC | Modelo-Vista-Controlador, patrón de diseño utilizado para separar la lógica de negocio de la interfaz de usuario |
| UML | Unified Modeling Language, lenguaje gráfico para representar la arquitectura del sistema |
| CRUD | Operaciones básicas de Create, Read, Update, Delete para la gestión de datos. |

# **DESCRIPCIÓN DE ARQUITECTURA:** La arquitectura del sistema se presenta mediante el uso de vistas basadas en el Modelo 4+1 de Kruchten, con el fin de proporcionar una visión completa y estructurada del sistema desde diferentes perspectivas. La arquitectura del sistema se presenta mediante el uso de vistas basadas en el Modelo 4+1 de Kruchten, con el fin de proporcionar una visión completa y estructurada del sistema desde diferentes perspectivas.

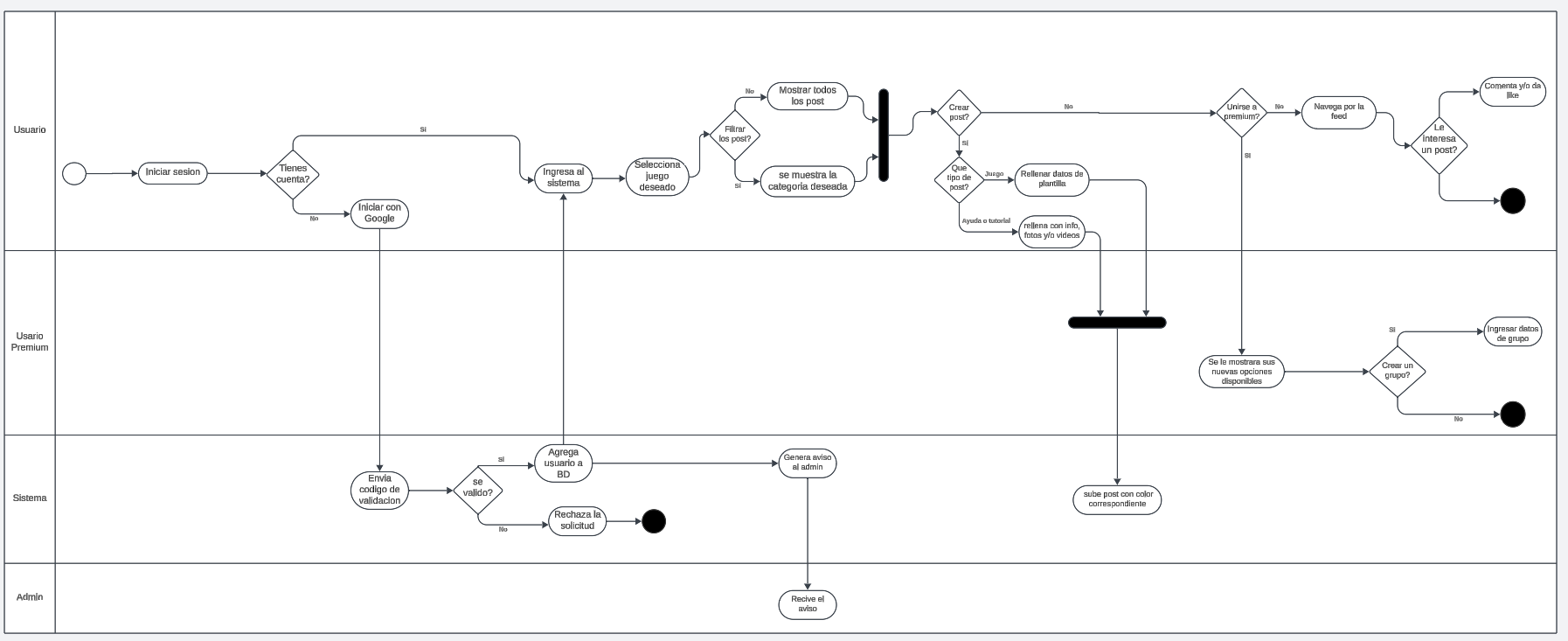
## VISTAS:

**Diagrama de Caso de Uso general**

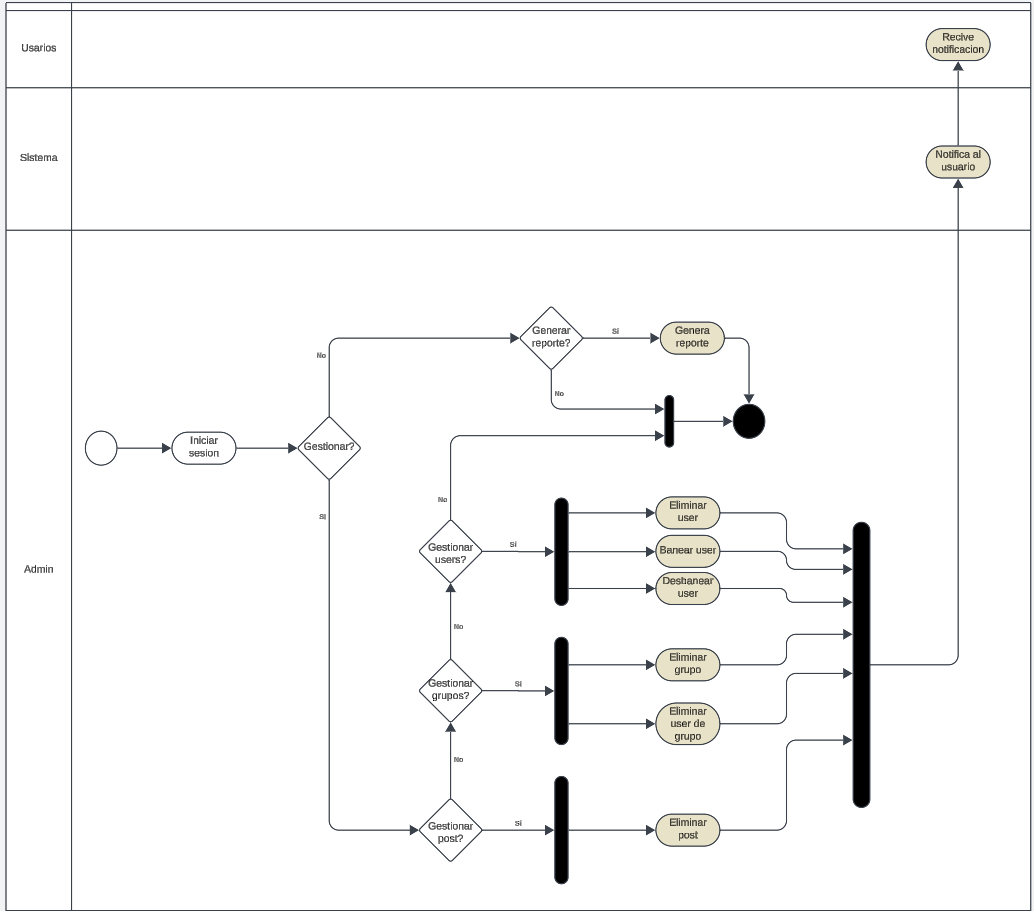
**Diagrama de Clases**



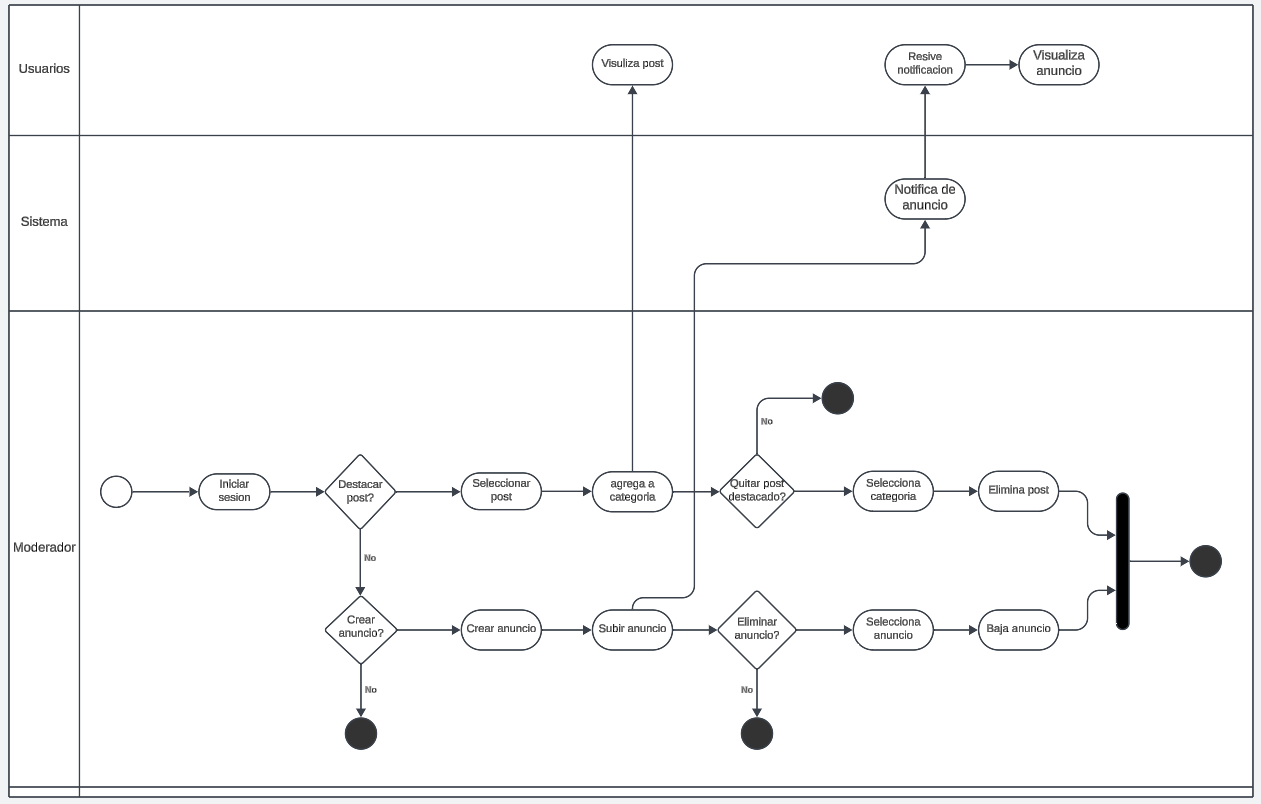
**Diagrama de Actividad**

Procesos de usuarios

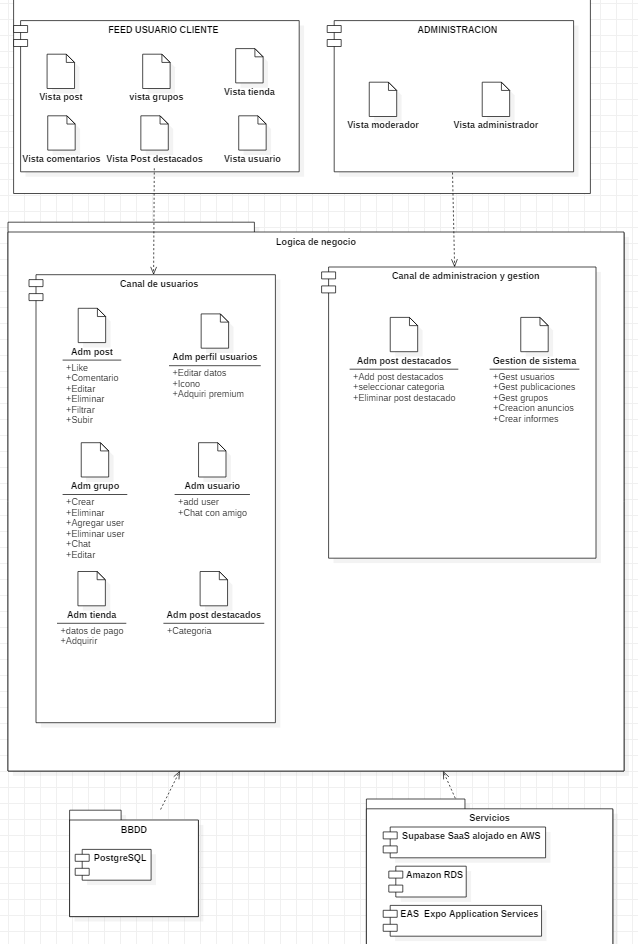
Procesos de administrador



Procesos de moderador

****

**Diagrama de Componentes**

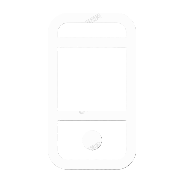
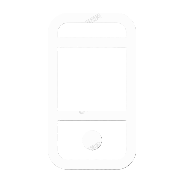
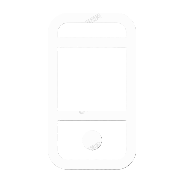
****

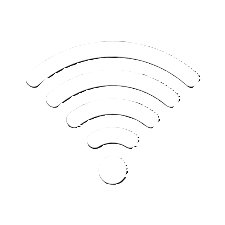
**Diagrama Topológico**

User Premium

User Free

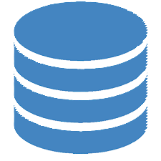
Admin

****

****

Internet

****

****

BBDD

(Supabase)

App

### TIPOS DE VISTAS:

1. **Vista de Escenarios (Diagrama de Caso de Uso)**: Describe las funcionalidades del sistema y cómo interactúan los diferentes actores con él.
2. **Vista Lógica (Diagrama de Clases)**: Detalla la estructura de datos y las relaciones entre las principales entidades del sistema.
3. **Vista de Procesos (Diagrama de Actividad)**: Muestra el flujo de actividades dentro del sistema y cómo se coordinan los diferentes procesos.
4. **Vista de Despliegue (Diagrama de Componentes)**: Representa la distribución de componentes en la infraestructura física.
5. **Vista Física (Diagrama Topológico)**: Representa la infraestructura de red y la comunicación entre los diferentes servicios del sistema

## Framework Conceptual

### Descripción de la arquitectura en contexto

El sistema será diseñado y desarrollado utilizando un enfoque ágil con metodologías como Scrum, lo que permitirá la entrega incremental de funcionalidades, la retroalimentación constante y la capacidad de adaptarse a cambios en los requisitos a lo largo del ciclo de vida del proyecto. Esto asegurará que se puedan abordar necesidades emergentes y maximizar el valor del producto para los usuarios.

El diseño del sistema se basará en la separación de vistas, que ofrecerán diferentes perspectivas de la arquitectura para una comprensión integral. A continuación, se describen las principales vistas:

* **Vista de Escenarios**: Representada mediante diagramas de Casos de Uso, que describen las interacciones del usuario con el sistema y los flujos principales.
* **Vista Lógica**: Incluye el Modelo de Clases, que muestra las entidades y relaciones del sistema, así como el Modelo de Datos que detalla la estructura de la base de datos.
* **Vista de Procesos**: Utiliza Diagramas de Actividad para ilustrar los flujos internos del sistema y las interacciones entre los distintos módulos.
* **Vista de Despliegue**: Se implementará a través de Diagramas de Componentes o Modelos de Capas para reflejar la distribución de la lógica de negocio, la interfaz de usuario y la capa de acceso a datos.
* **Vista Física**: Utiliza Diagramas Topológicos o de Servicios para mostrar la distribución del sistema en el entorno físico, como servidores, redes y otros componentes de infraestructura.

Cada una de estas vistas está interconectada para garantizar la coherencia y la alineación con los requisitos de negocio. No hay ninguna vista separada de una implementación específica, sino que todas contribuyen a una visión holística de la arquitectura del sistema descrita en este documento.

Estas vistas se han creado utilizando el Lenguaje de Modelado Unificado (UML) en su versión 2.0 y han sido desarrolladas empleando herramientas de modelado estándar como **StarUML** y Lucidchart y en front-end **Canva, Miro, Figma u otro** para facilitar la creación de diagramas visuales que respalden la documentación arquitectónica.

El estilo arquitectónico se centra en una combinación de patrones de diseño orientados a servicios independientes y desacoplados (ejemplo: SOA) y arquitectura por capas, en donde cada módulo del sistema se corresponde con un conjunto específico de casos de uso que resuelven los procesos definidos para la operación.

Estos procesos corresponden a:

Ejemplo:

* **Gestión de Proyectos y Colaboraciones:** Administración de proyectos colaborativos entre usuarios (como mods o eventos), con la posibilidad de gestionar recursos compartidos o tareas dentro del proyecto.
* **Gestión de Contenidos de la Comunidad:** Moderación y distribución de contenido generado por los usuarios (como guías, videos, imágenes) y validación del cumplimiento de las normas de la comunidad.
* **Monitoreo y Seguridad del Sistema:** Control de accesos, supervisión de actividades inusuales, y aplicación de sanciones en caso de infracciones, como comportamiento tóxico o contenido inapropiado.

### Actores Usuarios y sus roles:

Este documento representa la identificación de Actores/Usuarios Stakeholders y sus roles a partir de la interpretación de los casos de uso del Negocio asociados.

|  |  |  |
| --- | --- | --- |
| ACTOR | ROL / CARGO | FUNCION |
| Twitch, Kick | Patrocinador | Financiar el mantenimiento a largo plazo de la app |
| Creadores de contenido | Patrocinador | Financiar el levantamiento de app |
| Usuario | Free | Es el cliente final que usara las funciones básicas de la app |
| Usuario | Premium | Similar al usuario free, pero con la capacidad de crear grupo, editarlos e invitar a quien desee |
| Usuario | Administrador | Usuario que podrá gestionar las publicaciones, grupos y los 2 tipos de usuarios |

### 8.3.2 Actividades de arquitectura en el ciclo de vida

El desarrollo de la arquitectura del sistema HolowMates se abordó utilizando un ciclo de vida **iterativo incremental** basado en **Scrum**. Esta metodología fue seleccionada porque permite realizar entregas parciales y frecuentes de software funcional, facilitando la validación temprana de los requisitos por parte de los stakeholders y el productowner con la adaptación a cambios en el entorno de negocio.

El ciclo de vida se organizó en **sprints** de 2 a 3 semanas, con cada iteración centrada en la construcción y mejora de un módulo específico del sistema (por ejemplo, el módulo de gestión de Contenidos de la Comunidad o el módulo monitoreo y Seguridad del Sistema). Al final de cada sprint, se realiza una revisión detallada de los resultados para asegurar la alineación con las expectativas del cliente y se priorizan las tareas para el siguiente sprint.

### 8.3.3 Descripciones prácticas de arquitectura.

Durante el proceso de desarrollo de la plataforma unificada para la comunidad de Dark Souls, la arquitectura del sistema evolucionó desde un caso de uso principal hasta un diseño modular, siguiendo estos pasos:

* **Análisis de Requerimientos:** Se analizaron los requisitos iniciales, identificando casos de uso clave como "Gestionar Proyectos y Colaboraciones", "Moderar Contenido de la Comunidad" y "Monitoreo y Seguridad del Sistema".
* **Definición de la Vista Lógica:** A partir del escenario principal, se identificaron las entidades y relaciones clave utilizando un diagrama de clases UML, que representaba el núcleo de la plataforma, incluyendo usuarios, proyectos colaborativos y moderadores.
* **Modelo de Datos:** Se diseñó un diagrama de base de datos para definir la estructura de la información, asegurando que los datos de los usuarios, proyectos y contenido compartido estuvieran alineados con la vista lógica.
* **Vista de Procesos:** Se modelaron los flujos de actividades esenciales de la plataforma, como la creación y gestión de proyectos colaborativos, la interacción de los usuarios y la moderación del contenido, mediante diagramas de actividad.
* **Despliegue:** Se definió la infraestructura física y lógica para alojar la plataforma en la nube, detallando los servicios necesarios como balanceadores de carga, servidores de aplicaciones y bases de datos distribuidas para garantizar la escalabilidad y disponibilidad del sistema.
* **Definición de Planes de Pruebas y Técnicas de Testing:** Se establecieron planes de pruebas detallados para verificar la funcionalidad de cada módulo, asegurando la calidad y el correcto funcionamiento del sistema antes de su lanzamiento.

## Documentación de la arquitectura

* **ISO/IEC 25010**: Se utilizó para evaluar la calidad del sistema en términos de funcionalidad, seguridad y eficiencia.
* **IEEE 1471-2000**: Proporcionó la guía para la representación de vistas arquitectónicas y la identificación de stakeholders.
* **Modelo 4+1 de Kruchten**: Estructura la arquitectura en vistas lógicas, de desarrollo, de procesos y física, facilitando la visualización desde múltiples perspectivas.
* **Planes de prueba**:
  + **Pruebas Unitarias:** Verificación de que cada componente o módulo del sistema (por ejemplo, la gestión de usuarios, el feed de contenido, etc.) funcione correctamente de manera aislada.
  + **Pruebas de Integración:** Asegurarse de que los diferentes módulos del sistema (login, gestión de grupos, moderación de contenido, etc.) se integren de manera fluida y funcionen correctamente en conjunto.
  + **Pruebas de Carga y Rendimiento:** Evaluación del comportamiento de la plataforma bajo diferentes niveles de carga para garantizar su rendimiento y escalabilidad, especialmente durante picos de tráfico de usuarios.
  + **Pruebas de Seguridad:** Validación de la resistencia del sistema ante amenazas de seguridad, como la protección de datos personales, control de accesos y prevención de vulnerabilidades.
  + **Pruebas de Usabilidad:** Verificación de que la interfaz de usuario sea intuitiva, accesible y fácil de navegar, asegurando una buena experiencia para los usuarios finales.
  + **Pruebas de Regresión:** Asegurar que las actualizaciones y mejoras en el sistema no afecten negativamente a las funcionalidades ya existentes.

## Selección de puntos de vista de la arquitectura

|  |  |  |
| --- | --- | --- |
| VISTAS | UML | Cantidad |
| Escenario Principal | Diagrama de Caso de uso | 1 (2-3 los necesarios) |
| Vista Lógica | Diagrama de Clases  MER | 1  1 |
| Vista de Proceso | Diagrama de Actividad:  Nombre 1.  Nombre 2  Nombre 3. | 1 (2-3, los necesarios) |
| Vista Física | Diagrama Topológico de Servicios | 1 |
| Vista de Despliegue | Diagrama de componentes  Diagrama de Capas (MVC) | 1  1 |

## Vistas de la arquitectura

|  |
| --- |
| UML-VISTA DE ESCENARIO |
| Diagrama de Caso de uso extendido |

**Registrar con Google:** Permite crear una cuenta utilizando la cuenta de Google del usuario

**Selección de juego:** Permitirá seleccionar el título en que se deseas navegar (Dark Souls 1, Dark Souls 2, Dark Souls 3)

**Subir Post:** El usuario podrá subir el post que desee, y seleccionar que tipo de post será (Juego o ayuda y tutoriales)

**Regla de negocio 1:** Los tipos de post serán identificados con un color único, Juego = Verde, Ayuda y tutoriales = Rojo

**Filtrar Posts:** Al estar navegando en la feed, se podrá filtrar el tipo de posts que se desea ver

**Crear grupo:** Si eres usuario premium, podrás crear grupos y gestionar aspectos básicos de ellos

**Comprar iconos:** Se podrá adquirir iconos u otros objetos para personalizar los perfiles de los usuarios

**Crear anuncios:** Como administrador se podrán crear anuncios que verán los usuarios, para anunciar cosas, mini retos, advertencias, etc.

**Generar reportes:** El administrador podrá generar reportes basados en los datos de la app, esto para tomar mejores decisiones.

**Gestionar usuarios:** El administrador podrá gestionar a los usuarios registrados en la app, eliminándolos, baneándolos y desbaneándolos.

**Gestionar publicaciones:** Al iniciar sesión como administrador se podrá gestionar todas las publicaciones, advirtiendo y/o eliminándolas

**Gestionar grupos:** El admin también podrá gestionar los grupos creados, eliminando usuarios, grupos y advirtiendo

**Destacar publicación:** Permite que el moderador destaque aquellas publicaciones que sean relevantes, y agregarlas a una de las tantas categorías de post destacados

|  |
| --- |
| UML-VISTA LÓGICA-DIAGRAMA DE CLASES |
| [Diagrama de Clases] |
|  |

El **Diagrama de Clases** para el sistema de gestión de condominios muestra las clases involucradas, sus atributos, métodos y las relaciones entre ellas. Este diagrama ayuda a modelar la estructura y el comportamiento del sistema de una manera orientada a objetos.

#### **1. Elementos Principales**

* Clases
  + Usuario
    - Atributos: ID\_Usuario, NOM\_USUARIO, CORREO, CONTRASEÑA, GLE\_USER\_ID, PLATAFORMA, FECHA\_REGISTRO, ROL
    - Métodos: registrarUsuario(), login(), editarPerfil(), cambiarContrasena()
  + Publicación
    - Atributos: ID\_POST, ID\_USUARIO, ID\_JUEGO, TIPO\_POST, TITULO, CONTENIDO, FECHA\_CREACION
    - Métodos: crearPublicacion(),editarPublicacion() , eliminarPublicacion()
  + Imagen\_post
    - Atributos: ID\_IMAGEN, URL\_IMAGEN, FECHA\_CREACION, ID\_POST
    - Métodos: agregarImagen(), eliminarImagen()
  + Juego
    - Atributos: ID\_JUEGO, NOMBRE\_JUEGO
    - Métodos: agregarJuego(), eliminarJuego()
  + Mensaje
    - Atributos: ID\_MENSAJE, REMITENTE, RECEPTOR, CONTENIDO, FECHA\_ENVIO
    - Métodos: crearMensaje(), editarMensaje(), eliminarMensaje()
  + Notificación
    - Atributos: ID\_NOTIFICACION, TIPO, CONTENIDO, FECHA\_CREACION, ESTADO, ID\_USUARIO
    - Métodos: enviarNotificacion()
  + Grupo
    - Atributos: ID\_GRUPO, NOMBRE\_GRUPO, FECHA\_CREACION, CREADOR\_ID
    - Métodos: crearGrupo(), eliminarGrupo()
  + Baneado
    - Atributos: ID\_BANEO, USUARIO\_ID, MOTIVO, FECHA\_BANEO, DURACION
    - Métodos: banearUsuario(), levantarBaneo()
  + Amigo
    - Atributos: ID\_SOLICITANTE, ID\_RECEPTOR, FECHA\_SOLICITUD, ESTADO
    - Métodos: seguirUsuario(), dejarDeSeguir()
  + Comentario
    - Atributos: ID\_COMENTARIO, CONTENIDO, FECHA\_CONTENIDO, ID\_USUARIO, ID\_PUBLICACION
    - Métodos:crearComentario(), editarComentario(), eliminarComentario()
  + Like
    - Atributos: ID\_LIKE, FECHA\_LIKE, ID\_USUARIO, ID\_PUBLICACION
    - Métodos: darLike() ,quitarLike()

#### **2. Relaciones:**

* Usuario tiene una relación de asociación con Publicación (1:N), ya que un usuario puede crear múltiples publicaciones.
* Publicación tiene una relación de asociación con Comentario (1:N), ya que una publicación puede tener múltiples comentarios.
* Publicación tiene una relación de asociación con Imagen\_post (1:N), ya que una publicación puede contener varias imágenes.
* Publicación tiene una relación de asociación con Juego (1:N), permitiendo categorizar las publicaciones por el juego relacionado.
* Usuario tiene una relación de asociación con Mensaje (1:N), permitiendo enviar múltiples mensajes.
* Notificación tiene una relación de asociación con Usuario (1:1), enviando múltiples notificaciones a los usuarios.
* Usuario tiene una relación de asociación con Amigo (1:N), permitiendo seguir y ser seguido por otros usuarios.
* Usuario tiene una relación de composición con Baneado (1:1)
* Usuario tiene una relación de composición con Grupo (N:N), ya que muchos usuarios pueden estar en muchos grupos y viceversa
* Like tiene una relación de asociación con Publicación y Usuario (1:N), ya que un usuario puede dar like a varias publicaciones y una publicación puede recibir múltiples likes

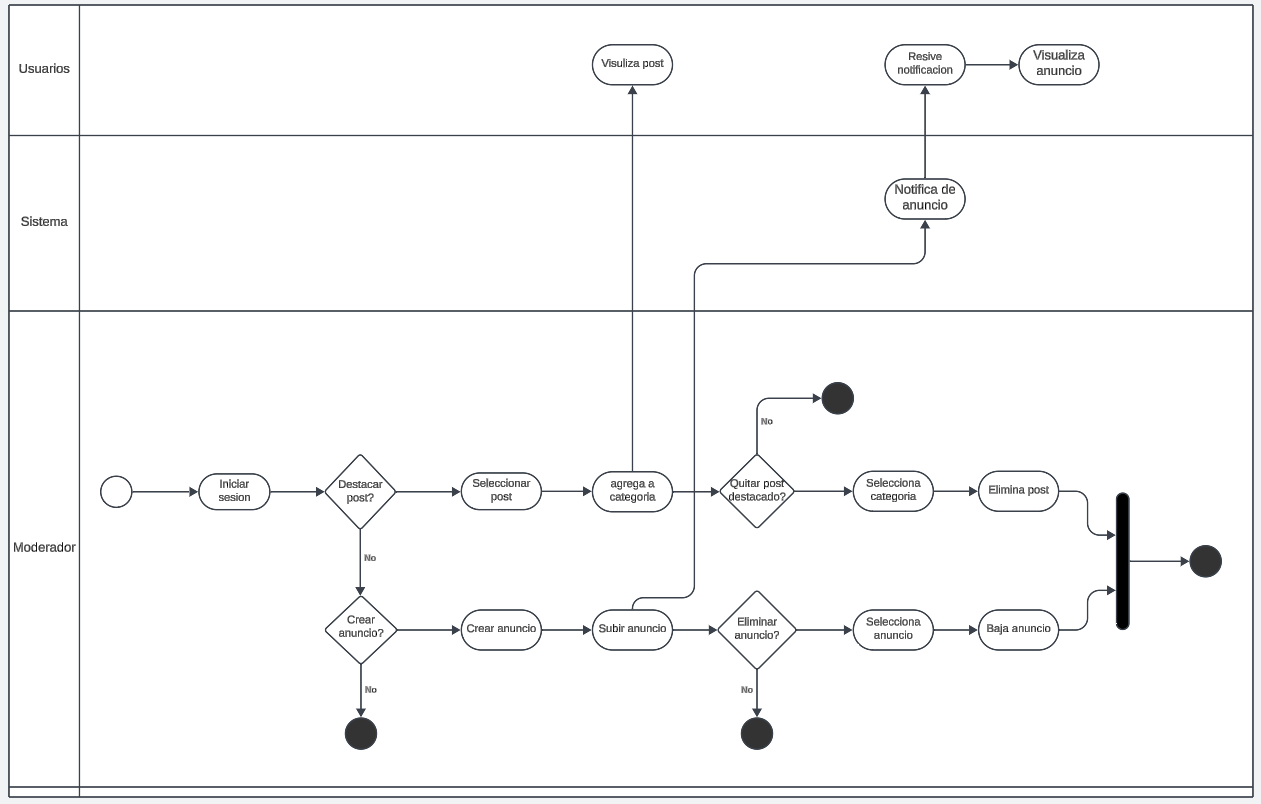
|  |
| --- |
| UML-VISTA LÓGICA-DIAGRAMA DE BASE DE DATOS |
| [Diagrama de BD - MER] |
|  |

El **Modelo Entidad-Relación (MER)** para el sistema de gestión de condominios representa las entidades involucradas, sus atributos y las relaciones que existen entre ellas. Este modelo permite visualizar cómo se organiza la información relacionada con la administración del condominio.

#### **2. Elementos Principales**

* Entidades
  + Usuario
    - Atributos: ID\_Usuario (PK), NOM\_USUARIO, CORREO, CONTRASEÑA, GLE\_USER\_ID (UK), PLATAFORMA, FECHA\_REGISTRO, ROL
  + Publicación
    - Atributos: ID\_POST (PK), ID\_USUARIO (FK), ID\_JUEGO (FK), TIPO\_POST, TITULO, CONTENIDO, FECHA\_CREACION
  + Imagen\_post
    - Atributos: ID\_IMAGEN (PK), URL\_IMAGEN, FECHA\_CREACION, ID\_POST (FK)
  + Juego
    - Atributos: ID\_JUEGO (PK), NOMBRE\_JUEGO
  + Mensaje
    - Atributos: ID\_MENSAJE (PK), REMITENTE\_ID (FK), RECEPTOR\_ID (FK), CONTENIDO, FECHA\_ENVIO
  + Notificación
    - Atributos: ID\_NOTIFICACION (PK), TIPO, CONTENIDO, FECHA\_CREACION, ESTADO, ID\_USUARIO (FK)
  + Grupo
    - Atributos: ID\_GRUPO (PK), NOMBRE\_GRUPO, FECHA\_CREACION, CREADOR\_ID (FK)
  + Baneado
    - Atributos: ID\_BANEO (PK), USUARIO\_ID (FK), MOTIVO, FECHA\_BANEO, DURACION
  + Amigo
    - Atributos: ID\_SOLICITANTE (PF), ID\_RECEPTOR (PF), FECHA\_SOLICITUD, ESTADO
  + Comentario
    - Atributos: ID\_COMENTARIO (PK), CONTENIDO, FECHA\_CONTENIDO, ID\_USUARIO (FK), ID\_PUBLICACION (FK)
  + Like
    - Atributos: ID\_LIKE, FECHA\_LIKE, ID\_USUARIO (PF), ID\_PUBLICACION (PK)
* **Relaciones**:
  + **Usuario** puede estar asociado a **múltiples condominios**.
  + **Condominio** tiene **múltiples propiedades** y **múltiples espacios comunes**.
  + **Propiedad** tiene **múltiples usuarios residentes** y puede tener un **propietario**.
  + **Gasto Común** está asociado a una **propiedad** y es administrado por un **administrador**.
  + **Multa** se asocia a un **gasto común** específico.
  + **Pago** está vinculado a un **gasto común** y realizado por un **usuario**.
  + **Espacio Común** pertenece a un **condominio** y puede tener **múltiples reservas**.
  + **Reserva de Espacio** está asociada a un **espacio común** y a un **usuario**.
  + **Historial de Actividades** está vinculado a un **usuario** y a un **componente del sistema**.

|  |
| --- |
| UML-VISTA DE PROCESO-DIAGRAMAS DE ACTIVIDAD |
| [Diagramas de Actividad] |
|  |



El **diagrama de actividad** representa el flujo de actividades que se llevan a cabo en un proceso específico, en este caso, Los procesos de los usuarios y el proceso de gestión del administrador. Estos diagramas son útiles para visualizar los pasos secuenciales y las decisiones que un residente debe tomar al realizar un pago.

Procesos de usuarios

Permite visualizar los procesos que pueden usar los usuarios free y los usuarios premium. De esta forma nos podemos dar cuenta del orden de los pasos para realizar un proceso en especial, aunque varios de ellos se pueden saltar, ya que no son relevantes o no están vinculados al proceso.

Procesos de administrador

Este diagrama nos muestra visualmente los procesos de gestión que puede llevar a cabo el administrador. También muestra como interactúa con otros aspectos como el sistema y los usuarios

Procesos de moderador

Este diagrama nos muestra visualmente los procesos }e puede llevar a cabo el moderador.

|  |
| --- |
| UML-VISTA DE DESPLIEGUE |
| Sistema HollowMates |
|  |

* **Base de datos (SupaBase)**: Una base de datos en la nube que se usara tanto para almacenar los usuarios registrados, como las publicaciones junto con los comentarios y likes
* **AppService Backend:** Contendrá todas las funciones y características que se podrá llevar a acabo la aplicación, esta se conectará a la base de datos mediante una API o un link que permanecerá en el código
* **AdminBackend:** Es un personal que podrá solucionar errores o crear mejoras, es el único autorizado a manipular el backend del sistema. Usaran archivos js para la creación del Backend
* **AppService Frontend:** Usando React Native se creará las vistas Frontend de la aplicación. Este componente se comunicará a través de HTTP con el backend
* **Aplicación satélite:** Este componente ejemplifica la aplicación que será lanzada a los usuarios, se conectará al componente de internet mediante HTTPS
* **Aplicación Local:** Este componente será el que sea exclusivo del super administrador, ente contendrá todas las funciones que podrá llevar a cabo el admin
* **Super Administrador:** Este componente representa al usuario del administrador del sistema, este usuario podrá gestionar múltiples aspectos del sistema

|  |
| --- |
| UML-VISTA FÍSICA – DIAGRAMA INFRAESTRUCTURA |
| Sistema Gestión De Condominios |
|  |

**Aplicación principal**: Es el corazón del sistema donde se aloja la infraestructura central. Incluye servidores dedicados para la aplicación web, base de datos, notificaciones y backups. Todos estos elementos están protegidos por un firewall para garantizar la seguridad de los datos.

**Red de usuarios y administradores**: Esta red incluye los dispositivos de acceso dentro de la aplicación, como las gestiones del administrador y del usuario.

**Balanceador de carga:** distribuye el tráfico entrante entre los servidores backend. Esto asegura que los usuarios tengan acceso estable y equilibrado, sin sobrecargar un solo servidor.

**Conexiones Externas**: El acceso a Internet está proporcionado por un Proveedor de Servicios de Internet y se distribuye entre el centro de datos y la red de la app.

|  |
| --- |
| Usabilidad y Calidad de software- Front-end -Vistas Principales |
| Vistas principales del sistema – Vista Home Principal de entrada |
|  |

La **Vista Front-End** del sistema de inicio de sesion y selección de juegos (Vista principal) ha sido diseñada para ofrecer una experiencia de usuario intuitiva, moderna y fácil de navegar. La interfaz está dividida en diferentes secciones con menús y paneles dinámicos que se adaptan a cada tipo de usuario, brindando acceso a funcionalidades específicas según su rol (Administrador, Usuario Free, Usuario premium). Se presenta un diseño limpio y ordenado que resalta las opciones más importantes, permitiendo a los usuarios interactuar con el sistema de manera eficiente

|  |
| --- |
| Usabilidad y Calidad de software – Vistas específicas de usuario |
| Vistas principales del sistema – Vista Principal de Usuario del feed de juego, feed de los grupos y del perfil de usuarios, Tambien la vista principal del admin |
|  |

Los 2 tipo de usuarios tiene una vista compartida exceptuando la funcionalidad exclusiva de los usuarios premium, también la vista de admistrador solo muestra las herramientas de gestión que puede usar:

* **Administrador**: Gestión total usuarios, Publicaciones y grupos.
* **Usuario Free**: Visualización de selección de juego, feed del juego, posts especificos al filtrar, creación de post, perfil de usuario, tienda, Feed de grupos
* **Usuario Premium**: Comparte las mismas vistas que el usuario Free pero se le agrega la vista de creación de grupos y del grupo como tal.

## Consistencia en la cantidad de vistas de la arquitectura.

Se garantiza que cada vista esté alineada con los requisitos funcionales y no funcionales del sistema, evitando redundancias y manteniendo la coherencia en toda la arquitectura

### DESCRIPCION DE MODULOS

Ejemplos:

|  |  |  |
| --- | --- | --- |
| Nombre del módulo | Descripción | Componentes incluidos |
| Gestión de Publicaciones | Permite crear, editar y eliminar publicaciones, además de gestionar comentarios e imágenes. | Módulo de publicaciones, Módulo de comentarios, Módulo de imágenes |
| Gestión de Usuarios | Administración de cuentas, perfiles, y configuración de seguridad de usuarios. | Módulo de registro, Módulo de autenticación, Módulo de perfiles |
| Sistema de Notificaciones | Envía notificaciones a los usuarios sobre actividades relevantes dentro de la plataforma. | Módulo de notificaciones, Módulo de configuración de alertas |
| Sistema de Mensajería | Permite la comunicación entre usuarios mediante mensajes privados. | Módulo de envío de mensajes, Módulo de historial de mensajes |
| Gestión de Grupos | Permite la creación, edición y administración de grupos dentro de la comunidad. | Módulo de grupos, Módulo de miembros, Módulo de permisos |
| Moderación y Baneos | Administra las acciones de moderación y baneos de usuarios en la plataforma. | Módulo de moderación, Módulo de baneos, Módulo de reportes |
| Sistema de Reacciones | Permite a los usuarios dar like a publicaciones y comentarios. | Módulo de likes, Módulo de historial de reacciones |
| Gestión de Publicaciones | Permite crear, editar y eliminar publicaciones, además de gestionar comentarios e imágenes. | Módulo de publicaciones, Módulo de comentarios, Módulo de imágenes |

### DESCRIPCIÓN DE COMPONENTES

|  |  |  |
| --- | --- | --- |
| Nombre del Componente | Descripción | Componentes Relaciones |
| Gestor de Publicaciones | Administra la creación, edición y eliminación de publicaciones dentro de la plataforma. | Módulo de Gestión de Publicaciones, Módulo de Reacciones |
| Motor de Comentarios | Permite agregar, editar y eliminar comentarios en publicaciones. | Módulo de Gestión de Publicaciones |
| Gestor de Usuarios | Maneja la creación, edición y eliminación de cuentas de usuario y perfiles. | Módulo de Gestión de Usuarios, Módulo de Seguridad |
| Motor de Notificaciones | Envía notificaciones a los usuarios sobre actividades relevantes y mensajes emergentes. | Módulo de Sistema de Notificaciones, Módulo de Mensajería |
| Sistema de Mensajería | Facilita la comunicación entre usuarios mediante mensajes privados y directos. | Módulo de Mensajería, Módulo de Notificaciones |
| Gestor de Grupos | Administra la creación y gestión de grupos dentro de la comunidad, incluyendo miembros y roles. | Módulo de Gestión de Grupos, Módulo de Usuarios |
| Sistema de Moderación | Permite la administración de sanciones, moderación y gestión de usuarios baneados. | Módulo de Moderación y Baneos, Módulo de Reportes |

### DESCRIPCIÓN DE CONECTORES

|  |  |  |
| --- | --- | --- |
| Nombre del Conector | Tipo | Propósito |
| REST API | Comunicación | Transferencia de datos entre los módulos de la aplicación |
| Socket.io | Comunicación | Proporciona actualizaciones en tiempo real, como notificaciones y mensajes instantáneos |
| WebSocket | Comunicación | Permite la comunicación bidireccional entre cliente y servidor para eventos en vivo |
| OAuth 2.0 | Autenticación | Provee autenticación segura y autorización de usuarios mediante terceros |
| SupaBase Cloud Messaging | Notificación | Envío de notificaciones push a dispositivos móviles de los usuarios |

* + - 1. **Arquitectura lógica**
      2. Performances

El sistema está diseñado para manejar una alta concurrencia, soportando más de aproximadamente 1000 usuarios simultáneamente sin comprometer el rendimiento. Para lograr tiempos de respuesta inferiores a 2 segundos por transacción, se implementarán las siguientes estrategias:

* **Escalabilidad Horizontal**: Se utilizará la base de datos que permite almacenar datos de los usuarios, a los mismos usuarios, los posts, comentarios, y todos los datos de la app, también
* permite agregar recursos dependiendo de la demanda
* **Balanceo de Carga**: Se implementará un balanceador de carga que distribuye las solicitudes de los usuarios entre las diferentes instancias de los microservicios, asegurando una distribución equitativa del tráfico y evitando sobrecargas en un único servicio.
* **Optimización de Consultas**: Se aplicarán técnicas de optimización en las consultas a la base de datos PostgreSQL, incluyendo índices y consultas asíncronas para reducir los tiempos de respuesta.
  + - 1. Criterios de Calidad

El sistema debe cumplir con los siguientes criterios de calidad:

* **Disponibilidad 24/7**: La arquitectura se diseñará para asegurar un tiempo de actividad continuó, implementando redundancia y recuperación ante desastres en la infraestructura de AWS.
* **Respuesta Rápida**: Todas las transacciones deberán completarse en menos de 2 segundos, lo que se logrará mediante el uso de técnicas de caché y un diseño eficiente de la API.
* **Alta Seguridad**: Se implementará cifrado AES-256 para proteger datos sensibles, así como autenticación y autorización robustas para garantizar la integridad de la información y la privacidad del usuario.
  + - 1. Detalles de la implementación

La implementación del sistema se basará en una arquitectura de microservicios, utilizando las siguientes tecnologías:

* **Backend**: Node.js con Express para la creación de APIs RESTful.
* **Frontend**: React junto con Expo para la creación de una interfaz de usuario moderna y responsiva.
* **Base de Datos**: PostgreSQL para la gestión eficiente de datos estructurados.
  + - 1. Lenguajes y plataformas

Las tecnologías y versiones específicas que se utilizarán en el desarrollo del sistema incluyen:

* **Node.js**: v10.x
* **Expo**: v6.3.10
* **React**: v16.x
* **PostgreSQL**: v9.6
* **Supabase**
* **Visual Studio Code**: v1.50

## Acceso a fuentes de información del proyecto

Para acceder a la carpeta de evidencias acumuladas y otros documentos relevantes del proyecto, puede utilizar el siguiente enlace:  
  
Insertar link: https://github.com/Havel-45/2024\_2\_MA\_CAPSTONE\_001D\_GRUPO\_2  
  
Esta carpeta incluye documentación técnica, diagramas, pruebas de rendimiento y cualquier otra evidencia relacionada con el desarrollo y la implementación del sistema