**DOCUMENTACIÓN TÉCNICA APLICACIÓN CHAT DISTRIBUIDO**

1. **Introducción**

Este documento presenta la documentación técnica del proyecto final de la asignatura **Programación III** del programa de Ingeniería de Sistemas y Computación de la Universidad del Quindío. El proyecto consiste en el desarrollo de una **aplicación de chat distribuido** implementada en **Elixir**, cuyo propósito es permitir la **comunicación en tiempo real entre múltiples usuarios**, utilizando una arquitectura cliente-servidor basada en procesos concurrentes.

El sistema fue concebido para atender las necesidades de una empresa simulada llamada *ComNet Solutions*, dedicada al desarrollo de soluciones de comunicación seguras para entornos colaborativos. Con base en ese contexto, se propuso el diseño de una plataforma propia que garantice **mensajería instantánea**, **gestión de salas**, **historial persistente de mensajes**, y **tolerancia a fallos** mediante supervisores OTP.

A nivel técnico, el proyecto aprovecha las capacidades del lenguaje Elixir y su integración con la **máquina virtual de Erlang**, permitiendo una gestión eficiente de procesos concurrentes, alta escalabilidad, y la posibilidad de distribución horizontal entre nodos. Asimismo, la aplicación cuenta con una interfaz de línea de comandos para usuarios, comandos personalizados para la interacción y mecanismos básicos de autenticación y persistencia.

Esta documentación describe la **arquitectura del sistema**, los **módulos funcionales** que lo componen, el **flujo de operación**, los **comandos disponibles**, así como los aspectos relacionados con **seguridad, supervisión, escalabilidad y persistencia de datos**. Además, se incluyen consideraciones sobre el despliegue del sistema, pruebas realizadas y posibles líneas de mejora.

1. **Arquitectura del Sistema:**

### **2.1 Modelo Cliente-Servidor**

* **Servidor**:  
   El servidor central gestiona la lógica principal de la aplicación. Está encargado de:  
  + Crear y mantener las salas de chat.
  + Controlar la autenticación de usuarios.
  + Reenviar mensajes a los usuarios conectados.
  + Almacenar el historial de conversaciones.
  + Ejecutar comandos administrativos del sistema.
* **Cliente**:  
   Es una aplicación ejecutada desde la línea de comandos. Cada cliente se conecta a un nodo servidor, se autentica, y permite al usuario:  
  + Unirse a salas de chat.
  + Enviar y recibir mensajes en tiempo real.
  + Consultar el historial.
  + Ejecutar comandos como /list, /create, /join, /exit

### **2.2 Concurrencia y Procesos**

* **Sesión por usuario**:  
   Cada usuario conectado es representado por un **proceso independiente** que mantiene su estado de sesión, incluyendo alias, sala actual y conexión activa.
* **Manejo de salas**:  
   Las salas de chat pueden gestionarse como procesos individuales o estructuras compartidas, permitiendo que múltiples usuarios envíen y reciban mensajes dentro del mismo contexto de sala. Este diseño evita bloqueos y facilita la concurrencia.
* **Supervisión OTP**:  
   Se implementa un **árbol de supervisión OTP** que monitorea los procesos críticos:  
  + Si un proceso de sesión falla, se reinicia automáticamente.
  + Si ocurre una falla en el servidor principal, se aplica un reinicio controlado.
  + La estructura OTP asegura la **tolerancia a fallos** del sistema.

### **2.3 Distribución entre nodos**

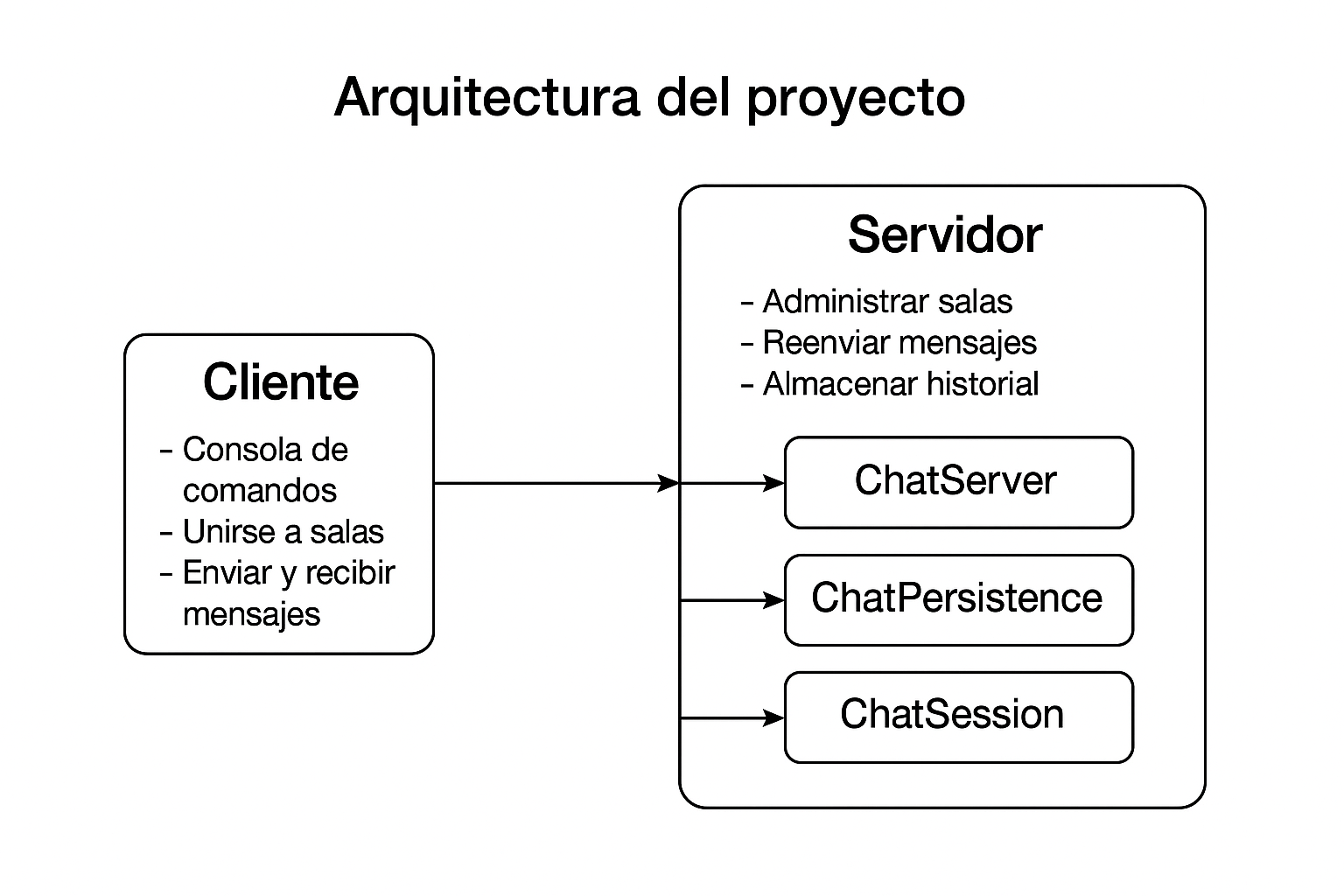
La aplicación está diseñada para ser **distribuida horizontalmente**:

* Utiliza Node.connect/1 para establecer comunicación entre nodos en red.
* Permite que múltiples instancias del servidor cooperen en la gestión de usuarios y salas.
* Esta capacidad de distribución mejora la escalabilidad y permite el balanceo de carga en escenarios con múltiples clientes simultáneos.

### **2.4 Persistencia de datos**

Para garantizar la recuperación de información entre ejecuciones:

* Los mensajes enviados y la información de las salas son **almacenados en archivos .dat.**
* Al reiniciar el servidor, estos archivos son cargados para reconstruir el historial.
* El módulo ChatPersistence se encarga de abstraer la lógica de lectura y escritura de estos datos.

Arquitectura: diagrama  


1. **Estructura del Proyecto:**

La aplicación está compuesta por una serie de módulos organizados funcionalmente, cada uno encargado de una responsabilidad específica dentro del sistema. A continuación, se describen los archivos y módulos principales del proyecto, con sus respectivos propósitos:

| **Archivo** | **Descripción** |
| --- | --- |
| chat\_app.ex / application.ex | Inicia el árbol de supervisión OTP. Controla el ciclo de vida de la aplicación (inicio/parada). Define cómo se inicia el sistema distribuido. |
| chat\_server.ex | Módulo principal del servidor. Gestiona la lógica central: manejo de salas, reenvío de mensajes, integración con usuarios. Utiliza GenServer para controlar su estado y eventos. |
| chat\_user.ex | Controla la gestión de usuarios conectados: alias, validaciones, listado de usuarios activos. Asegura unicidad de nombres y registro adecuado. |
| chat\_session.ex | Representa la sesión individual de cada cliente conectado. Maneja la entrada y salida de mensajes, validación de comandos y control de conexión. |
| chat\_persistence.ex | Proporciona funciones para guardar y recuperar mensajes y salas desde archivos .dat, garantizando persistencia entre ejecuciones. |
| chat\_utils.ex | Contiene funciones auxiliares como validación de comandos, formateo de mensajes, y respuestas del sistema. Mejora la experiencia de usuario en la terminal. |
| iniciar\_servidor.sh | Script para lanzar el servidor. Configura el entorno y ejecuta el nodo principal de Elixir. |
| iniciar\_cliente.sh | Script para lanzar un cliente. Solicita el alias del usuario y establece la conexión con el servidor. |

**3.2 Organización por responsabilidades:**

* **Conexión y sesión:**

chat\_session.ex y chat\_user.ex trabajan en conjunto para gestionar el ciclo de vida de cada conexión. Cada usuario se representa mediante un proceso que controla su actividad, alias y sala.

* **Mensajería y lógica del servidor:**

chat\_server.ex centraliza la lógica de enrutamiento de mensajes, control de salas, y delegación de tareas a otros módulos. Es el núcleo de la interacción.

* **Persistencia de datos:**

chat\_persistence.ex se encarga de almacenar el historial de conversaciones y salas en archivos locales. Su diseño permite cargar estos datos al reiniciar el sistema.

* **Supervisión y tolerancia a fallos:**

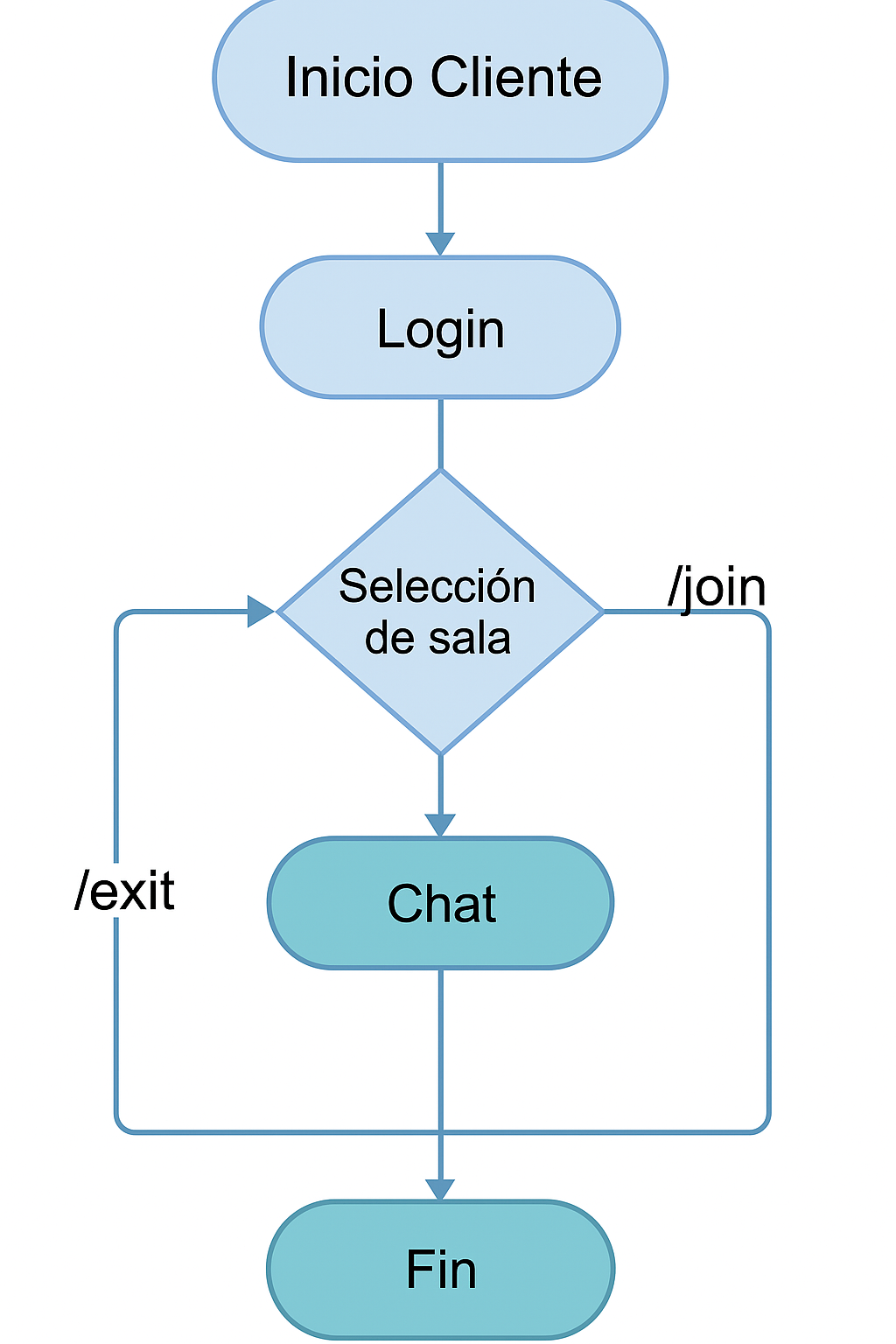
chat\_app.ex define el árbol de supervisión, que garantiza que los procesos críticos se reinicien automáticamente en caso de fallo, conforme al modelo OTP de Elixir.

* **Automatización de ejecución:**

Los scripts .sh simplifican la ejecución del sistema desde la terminal. Se recomienda darles permisos con chmod +x para habilitar su uso directo.

1. **Flujo de la aplicación**

* **Inicio del servidor**Al ejecutar iniciar\_servidor.sh, se despliegan los procesos supervisados (servidor de salas, sesiones, persistencia).
* **Inicio del cliente**  
  Cada cliente corre iniciar\_cliente.sh, se conecta al servidor, y accede a un menú de comandos.
* **Comunicación**  
  Los mensajes son enviados a través del servidor. Este los reenvía a todos los usuarios en la misma sala.
* **Persistencia**  
  Cada mensaje y sala se guarda automáticamente en los archivos .dat, asegurando que no se pierda información al cerrar la app.
* **Finalización**  
  El usuario puede cerrar su sesión con /exit o cambiar de sala con /join <sala>.



1. **Comandos e Interacción**

| **Comando** | **Función** |
| --- | --- |
| /list | Muestra un listado actualizado de todos los usuarios conectados al sistema, incluyendo su alias y la sala en la que se encuentran. |
| /create sala | Crea una nueva sala de chat con el nombre especificado. Si ya existe una sala con ese nombre, se muestra un mensaje de advertencia. |
| /join sala | Permite al usuario unirse a una sala existente. Si la sala no existe, se sugiere crearla primero con /create. El usuario abandona automáticamente su sala anterior. |
| /history | Muestra los últimos mensajes registrados en la sala actual, recuperados desde los archivos persistentes correspondientes. |
| /exit | Finaliza la sesión del usuario, cierra la conexión con el servidor y termina el proceso cliente. El servidor actualiza la lista de usuarios activos. |

* Todos los comandos deben ingresarse con el prefijo /, y los nombres de salas deben evitar caracteres especiales o espacios.
* Si se ingresan comandos inválidos, el sistema muestra un mensaje de ayuda o error indicando el formato correcto.
* El módulo ChatUtils es el encargado de interpretar y enrutar los comandos desde el cliente hacia los procesos del servidor correspondientes.

1. **Supervisión y tolerancia a fallos**

Uno de los pilares fundamentales del sistema desarrollado es su capacidad de recuperación automática ante fallos, garantizada por el uso del modelo de supervisión de Elixir/OTP. Gracias a este enfoque, el sistema puede continuar funcionando de manera estable incluso si algunos de sus procesos individuales fallan durante la ejecución.

**6.1 Árbol de supervisión:**

El proyecto implementa un árbol de supervisión jerárquico, definido en el módulo ChatApp, con las siguientes responsabilidades:

* Supervisar procesos de sesión de usuario: Cada conexión de cliente activa es gestionada como un proceso supervisado. Si un proceso falla inesperadamente, el supervisor reinicia solo ese proceso, sin afectar a los demás usuarios.
* Supervisar el proceso del servidor principal (ChatServer): El servidor de salas y mensajes también está supervisado. En caso de fallo, puede ser reiniciado en un estado limpio o reconstruido a partir de los datos persistidos en archivos .dat.
* Supervisar el módulo de persistencia (ChatPersistence): Este proceso se encarga de guardar y recuperar el historial de mensajes. Si el módulo falla, se reinicia automáticamente para evitar pérdida de datos.

**6.2 Estrategia de reinicio:**

Se utilizan estrategias de reinicio proporcionadas por OTP, como :one\_for\_one, donde solo el proceso fallido se reinicia, sin afectar al resto de la jerarquía. Esto permite:

* Aislar fallos a nivel de usuario (por ejemplo, una conexión interrumpida).
* Garantizar disponibilidad continua del servidor.
* Reducir el impacto de errores imprevistos.

**6.3 Registro y monitoreo de eventos:**

El sistema puede ampliarse fácilmente para incluir:

* Logs de eventos con Logger para registrar errores, intentos de conexión y cambios de sala.
* Alertas o métricas si se desea integrar herramientas externas de monitoreo (como Telemetry o Prometheus en Elixir).

**6.4 Mecanismo de recuperación del estado:**

* La información crítica (como historial de mensajes y salas creadas) se almacena en archivos .dat mediante el módulo ChatPersistence.
* Al reiniciar el sistema, esta información es cargada automáticamente, lo que permite reanudar la operación sin pérdida de datos.

1. **Seguridad y Escalabilidad**

* **Seguridad:** Básica (identificación).
* **Escalabilidad:** Puede distribuirse usando nodos conectados (Node.connect/1), y balancear sesiones entre ellos.