Universidad Veracruzana

Facultad de Ingeniería Eléctrica y Electrónica

Proyecto Final

Nombre:

Areli Arisai Meza Rendón

Matricula:

zS21002401

Experiencia Educativa:

Programación de redes

Lunes 15 de diciembre de 2025

Índice

Introducción

Las aplicaciones y las redes sociales hoy juegan un papel muy importante en la comunicación entre diferentes dispositivos conectados por medio de una red, redes que como tal no logramos percibir visualmente pero que existen. En si este proyecto tiene como objetivo poder desarrollar una aplicación cliente – servidor en donde se implementen diferentes mecanismos de comunicación en una red, en este caso es mas que nada utilizando TCP y UDP, además de incluir servicios RESful y RMI, esto con el fin de poder permitir la interacción concurrente de varios clientes con un servidor central.

En si el sistema permite a los usuarios poder conectarse de una manera segura, ingresando con su nombre y contraseña, además de poder intercambiar mensajes en tiempo real, recibir notificaciones y gestionar información utilizando una pequeña base de datos. Podemos decir que se logran cumplir todos los requerimientos mínimos mencionados en el documento principal, a pesar de que sea un proyecto muy pequeño y sencillo.

Objetivos

Objetivo general

Desarrollar una aplicación cliente – servidor utilizando TCP y UDP para la comunicación, RMI y RESful para poder invocar métodos remotos y una pequeña base de datos con la información del usuario, permitiendo una conexión concurrente con múltiples dispositivos.

Objetivos específicos

- Implementar una comunicación confiable entre cliente y servidor usando TCP.
- Enviar notificaciones (mensajes) rápidos usando UDP.
- Utilizar RMI y RESful para permitir el acceso remoto a los servidores.
- Implementar programación concurrente con varios clientes conectados a la vez.
- Aplicar mecanismo de seguridad usando una autenticación y hashing.

Alcance del proyecto

El proyecto en si logra abarcar e implementar un sistema que es capaz de:

- Gestionar múltiples clientes conectados al mismo tiempo.
- Permitir el intercambio de mensajes en tiempo real.
- Registrar y autenticar usuarios con datos de forma segura
- Enviar mensajes a través de UDP.

Requisitos del sistema

Requisitos funcionales

- El sistema permite la conexión de al menos 2 dispositivos diferentes.
- El cliente proporciona una interfaz para poder interactuar con el usuario.
- El servidor procesa las solicitudes y responder.
- El sistema implementa
- TCP
- UDP
- RMI
- RESful (get y post)
- El sistema debe permitir enviar mensajes y notificaciones en tiempo real.
- Los usuarios deben autenticarse para acceder al chat.

Requisitos no funciones

- La aplicación maneja errores sin fallos críticos.
- El código debe ser claro.
- Las contraseñas deben estar cifradas.

Diseño del sistema

Arquitectura Cliente – Servidor

El sistema tiene una arquitectura cliente – servidor en donde el servidor actúa como núcleo de todo el sistema, gestionando la lógica de lo demás, es decir, la comunicación en la red y la persistencia de los datos.

Los clientes acceden al servidor mediante una interfaz web con una conexión especifica.

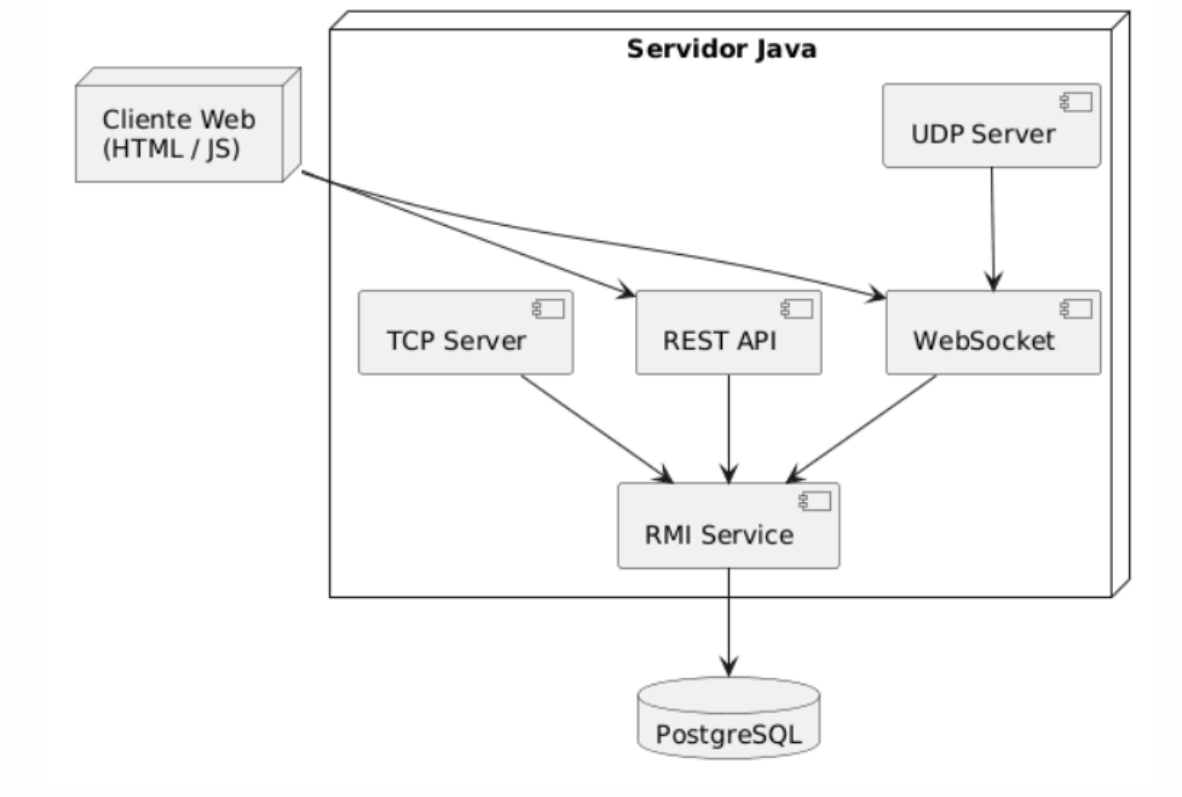


Diagrama de casos de Uso

El diagrama de casos de uso logra mostrar las principales funciones del sistema desde una perspectiva del cliente y del administrador, esto incluye la autenticación, el envió de mensajes y las notificaciones.

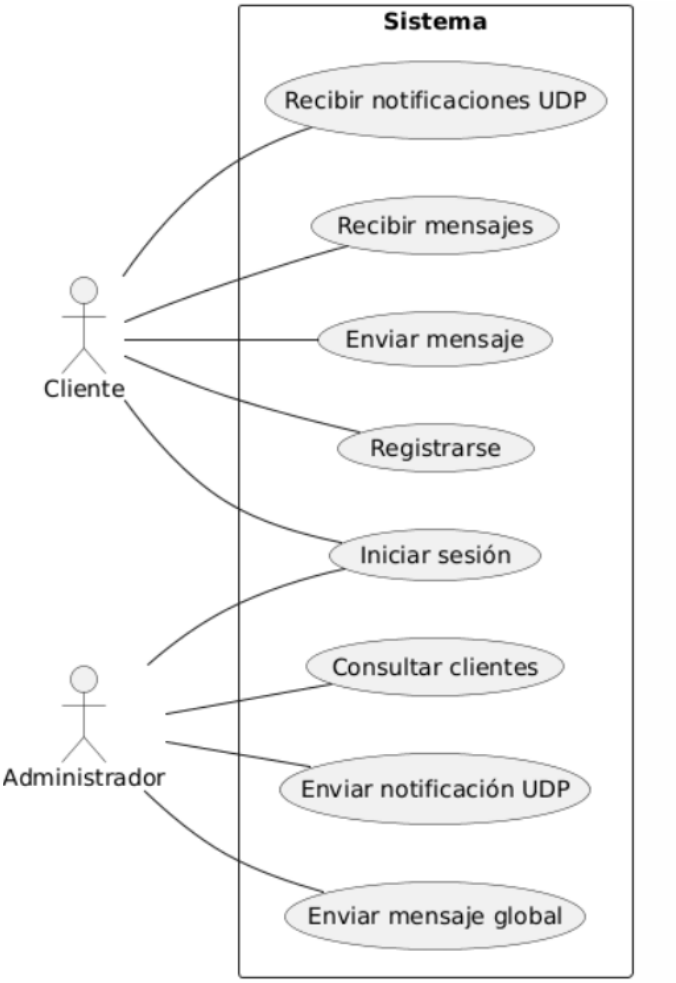


Diagrama de clases

El diagrama de clases representa como tal la estructura del sistema y sus relaciones entre los componentes principales, como son el servidor, controladores REST, los servicios de RMI y la conexión de la base de datos.

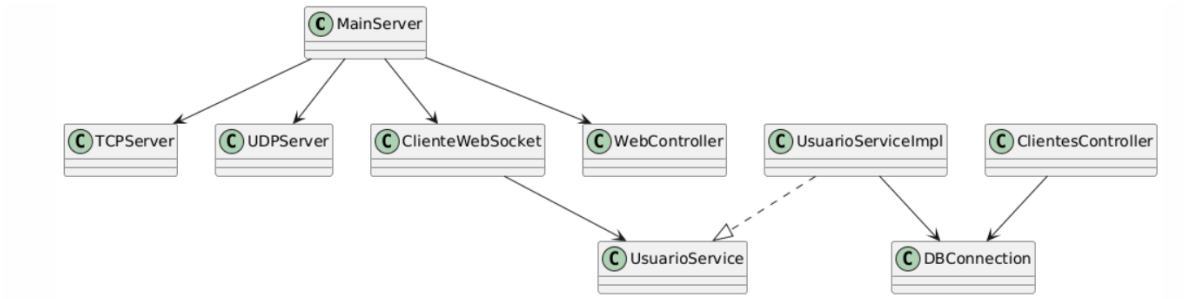
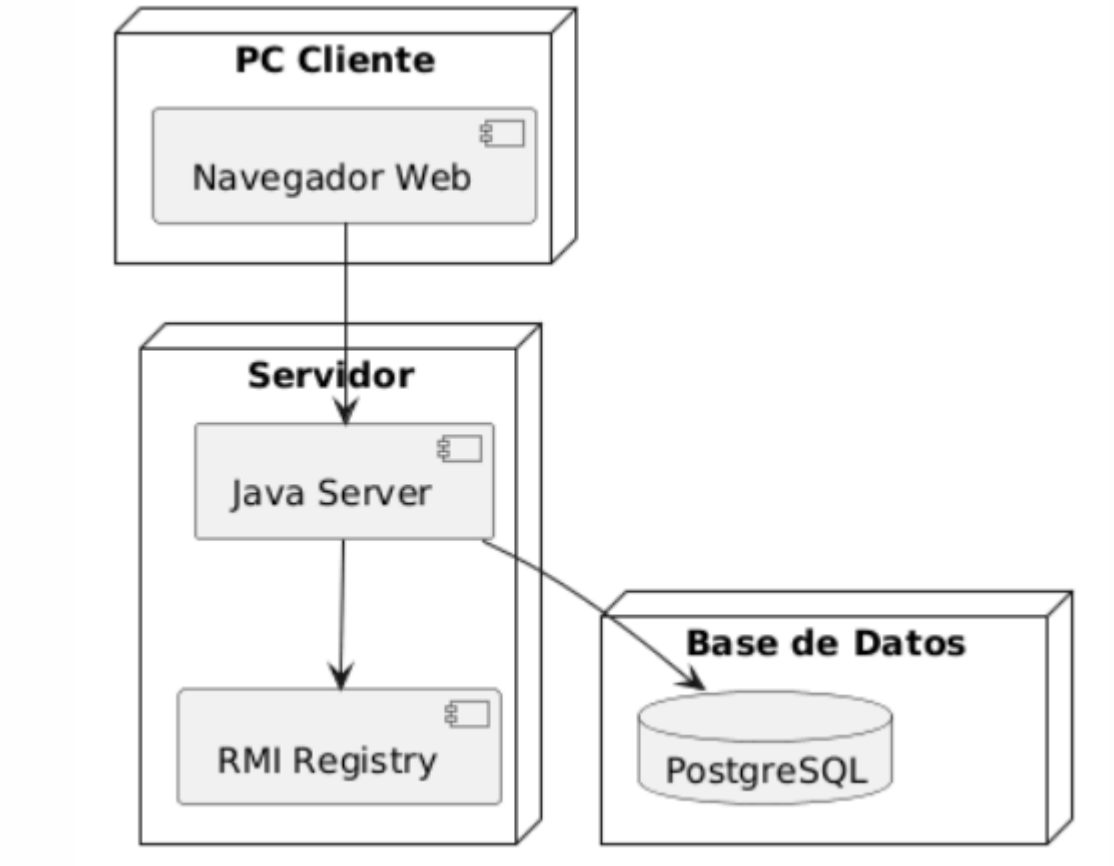


Diagrama de despliegue

Este diagrama muestra la distribución física de los componentes de este proyecto, entre ellos incluye el cliente web, el servidor Java y la base de datos



Implementación

Sockets TCP

El TCP se utiliza para poder establecer una comunicación mas confiable y orientada a la conexión entre un cliente y el servidor, cada cliente es atendido con un hilo independiente y permite que la conexión sea concurrente con diversos clientes.

```java
public class TCPServer {

    private int puerto;

    public TCPServer(int puerto) {
        this.puerto = puerto;
    }

    public void start() {
        try (ServerSocket serverSocket = new ServerSocket(puerto)) {

            while (true) {
                Socket cliente = serverSocket.accept();
                System.out.println("Cliente conectado: " + cliente.getInetAddress());

                //aqui es donde se va manejando el multi hilo
                ClientHandler handler = new ClientHandler(cliente);
                new Thread(handler).start();
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```java
public class TCPClient {

    private String host;
    private int puerto;

    public TCPClient(String host, int puerto) {
        this.host = host;
        this.puerto = puerto;
    }

    public void start() {
        try (
            Socket socket = new Socket(host, puerto);
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream
            PrintWriter out = new PrintWriter(socket.getOutputStream(), autoFlush: true);
            Scanner scanner = new Scanner(System.in);
        ) {

            // Mensajes del servidor
            System.out.println(in.readLine()); // Bienvenida
            System.out.print(in.readLine() + " ");
            out.println(scanner.nextLine());

            System.out.print(in.readLine() + " ");
            out.println(scanner.nextLine());

            // Respuesta final
            System.out.println(in.readLine());

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
Servicio RMI activo en puerto 1099
Servidor TCP escuchando en puerto 5000
```

Sockets UDP

En el UDP se emplea para poder enviar mensajes rápido (aunque no te asegura que llegue completo), esto se implementa como “notificación” del administrador, aunque bueno solo es cosa de que el admi mande un mensaje como UDP, así se prioriza mas la confiabilidad y confiabilidad.

```java
public UDPServer(int puerto) {
    try {
        UDPServer.puerto = puerto;
        socket = new DatagramSocket();
        address = InetAddress.getByName(host: "localhost");
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public static void enviarMensaje(String mensaje) {

    if (socket == null) {
        System.out.println(x: "[UDP ERROR] Socket no inicializado");
        return;
    }

    try {
        byte[] buffer = mensaje.getBytes();
        DatagramPacket packet =
            new DatagramPacket(buffer, buffer.length, address, puerto);

        socket.send(packet);

        //con esto todo se dentro de la web
        org.example.ws.ClienteWebSocket.notificarUDP(mensaje);

        System.out.println("[UDP ENVIADO] " + mensaje);

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

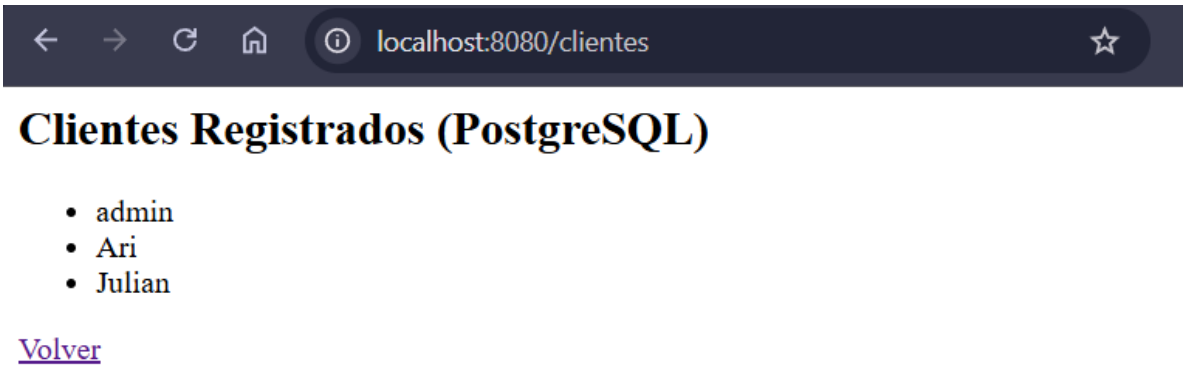
```java
public class UDPClient {

    public void start() {
        try (DatagramSocket socket = new DatagramSocket(port: 6000)) {

            byte[] buffer = new byte[256];

            System.out.println(x: "Cliente UDP escuchando...");

            while (true) {
                DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
                socket.receive(packet);

                String mensaje = new String(packet.getData(), offset: 0, packet.getLength());
                System.out.println("UDP recibido: " + mensaje);
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

ONLINE Julian

NOTI UDP: ADMIN: Esto es un UDP

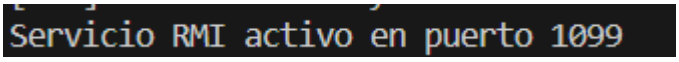
Servicios RESTful

Se implementan usando framework Spark, empleando método HTTP GET y POST para la autenticación, la consola del cliente es registrados y este dispara la acción como en envío de mensajes UDP.



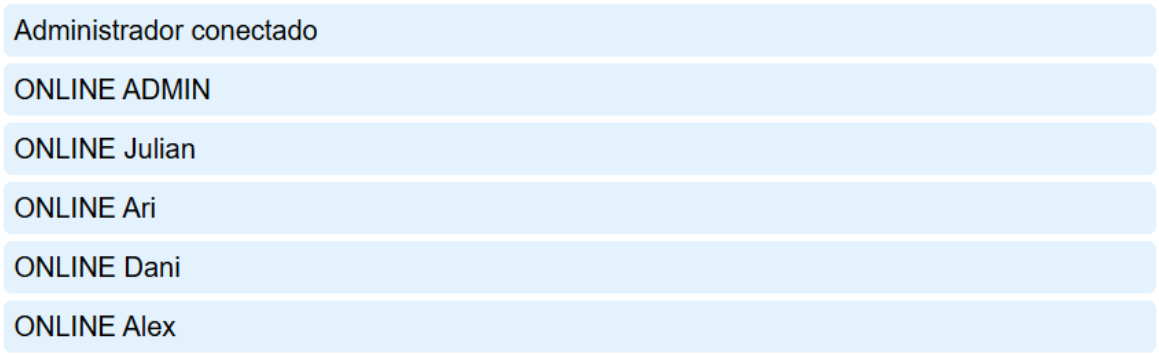
RMI

Este es para poder permitir la ejecución remota de operaciones relacionadas con la autenticación y la gestión de los usuarios.



Multithreading

Como tal se usa una programación concurrente para poder manejar múltiples clientes de forma simultánea, con esto nos aseguramos que cada conexión se atienda de forma independiente sin que esta se bloquee.



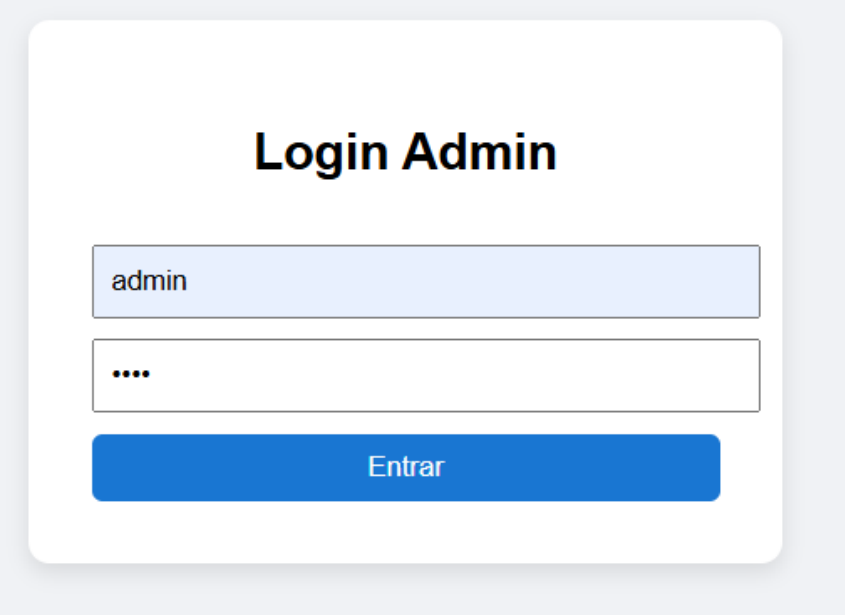
Seguridad

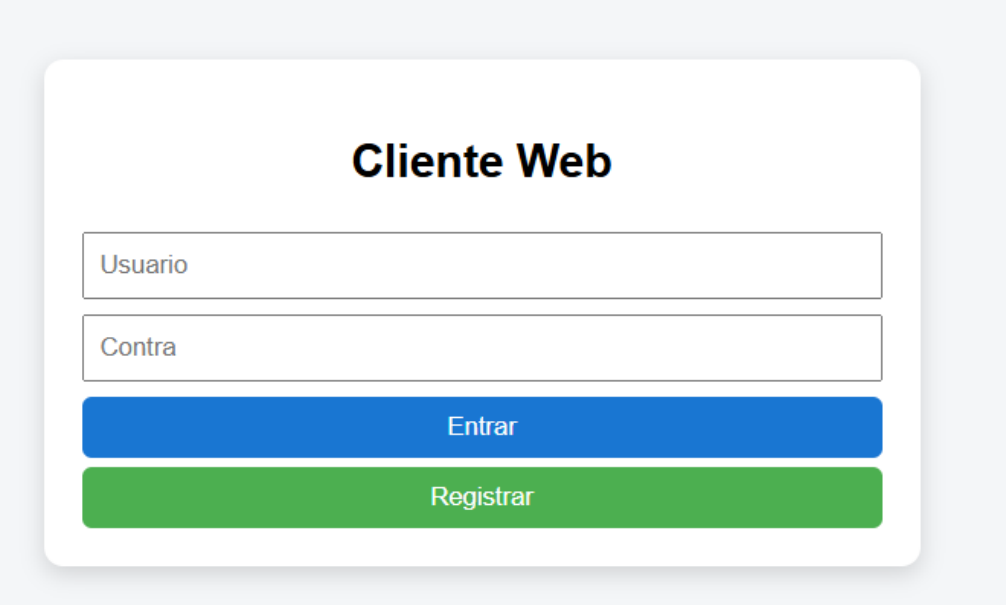
El programa tiene un mecanismo de seguridad muy básico:

Autentica a los usuarios (admi y cliente).

Almacena las contraseñas con hashing.

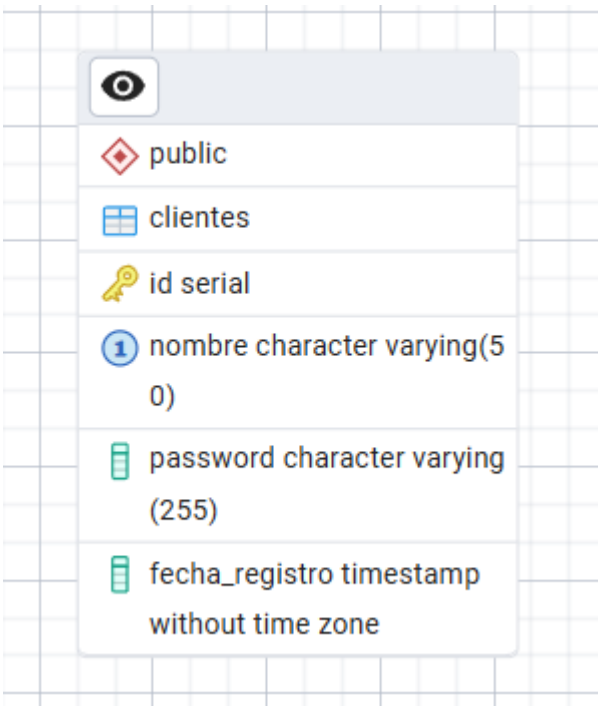
Evita que los nombres estén duplicados





Base de datos

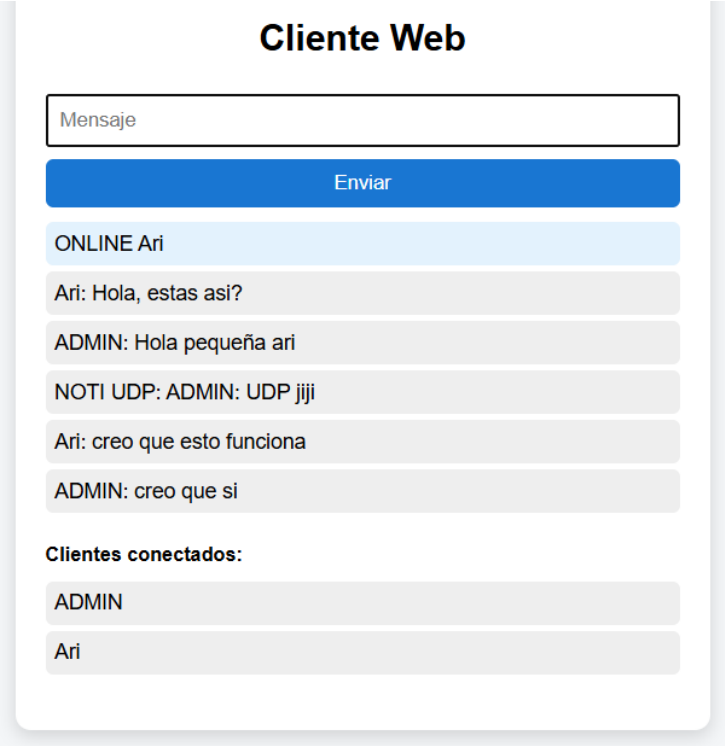
El programa usa PostgreSQL como base de datos para así poder almacenar la información de los usuarios registrados. Como tal es una sola tabla y se emplea para evitar vulnerabilidad al momento de ingresar.

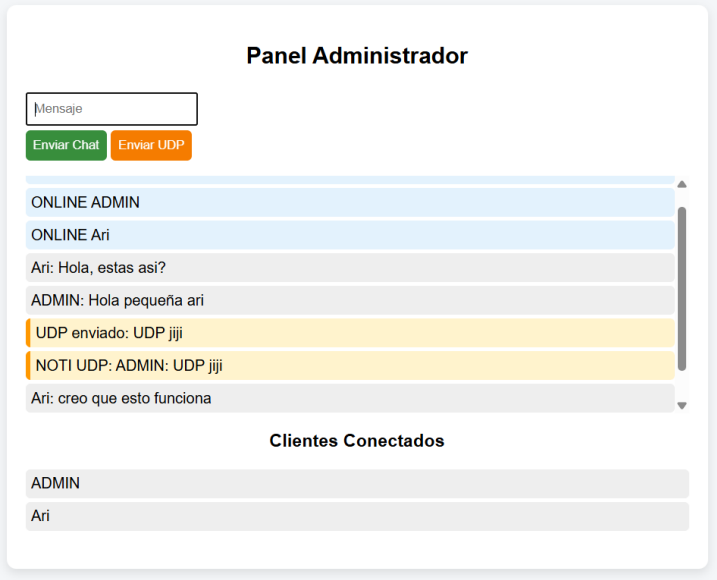


	id [PK] integer	nombre character varying (50)	password character varying (255)	fecha_registro timestamp without time zone
1	3	Ari	5797a75e3754b4fe63a52a135919240eff8830d717444a7d1d1935da8111e0...	2025-12-15 04:06:50.325891
2	4	admin	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4	2025-12-15 04:33:45.632594
3	5	Julian	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4	2025-12-15 04:36:12.185361
4	6	Dani	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4	2025-12-15 14:43:51.464359
5	7	Alex	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4	2025-12-15 14:44:07.731193

Resultados

El sistema como tal permite una conexión concurrente con varios usuarios, el envío y recibimiento de mensajes en tiempo real, mensaje (notificaciones) UDP por parte del admi y una administración de clientes desde un panel web.



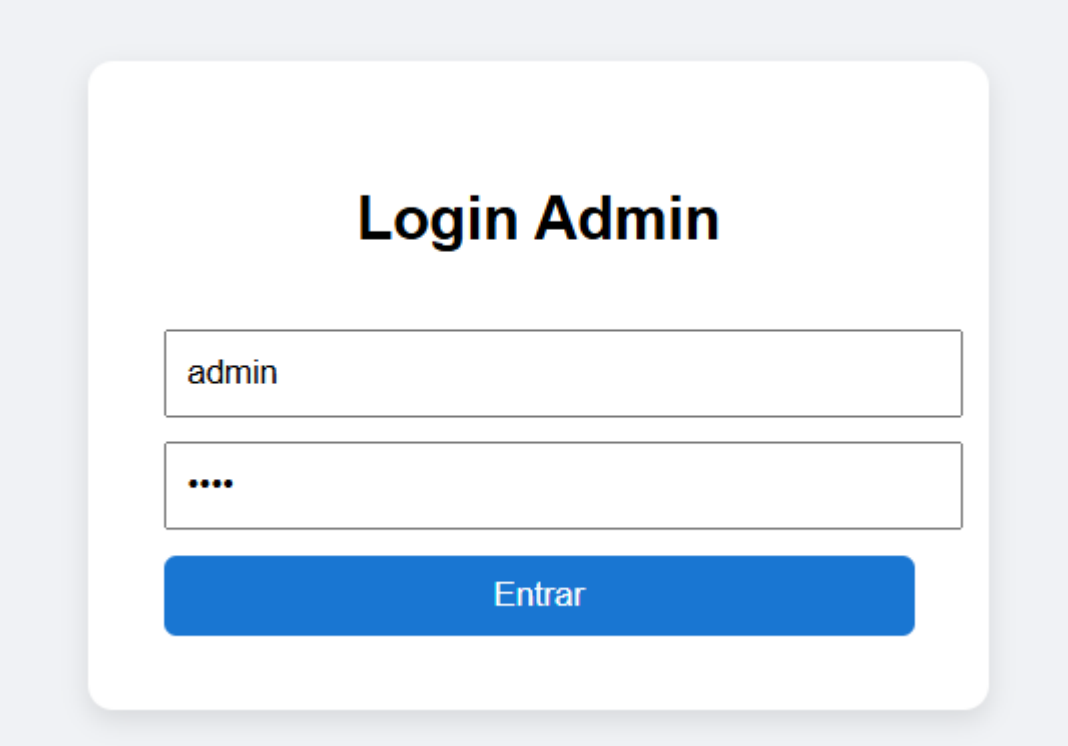


Manual de usuario

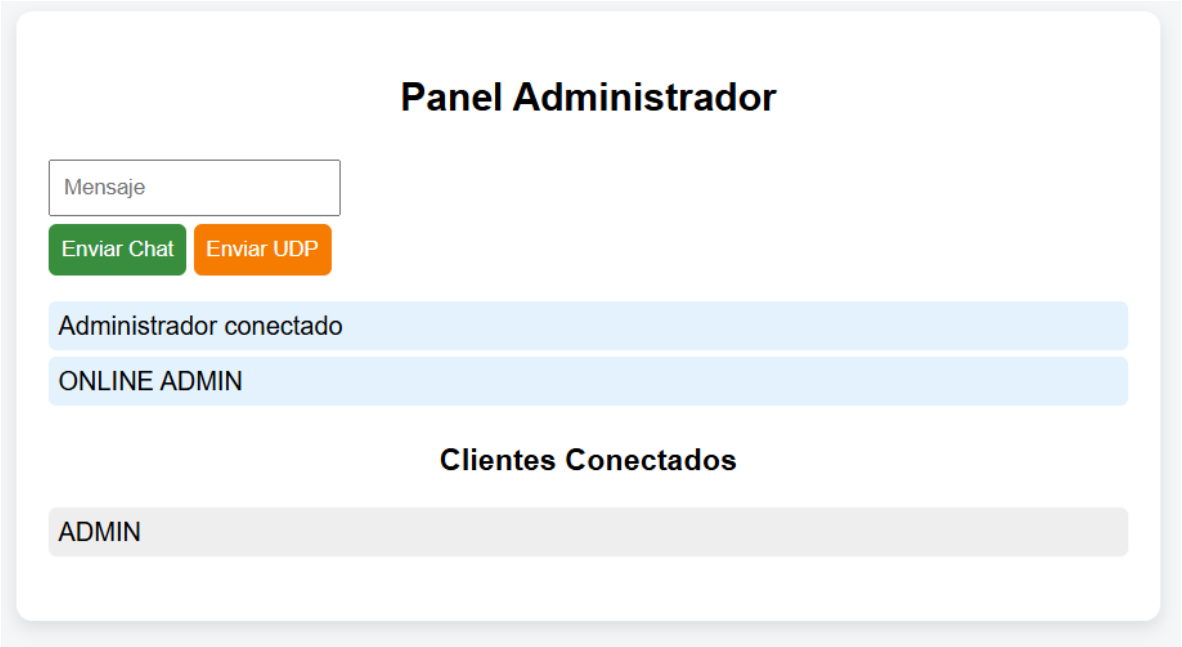
Administrador

Primero ingresamos al url (Web Admin disponible en http://localhost:8080) para que nos direccione a la página del administrador.

Una vez aquí estarás visualizando el login del administrador en donde tendrás que ingresar en el apartado de Usuario: admin y en la Contraseña:1234.

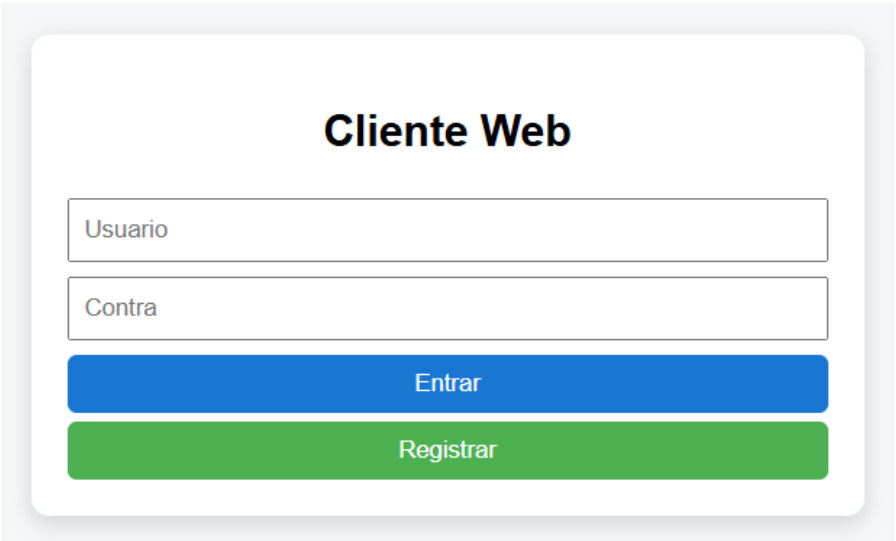


Una vez estando ahí y darle a entrar puedes ver la vista en donde podrás enviar mensajes y notificaciones UDP, así como también podrás ver a los clientes conectados.

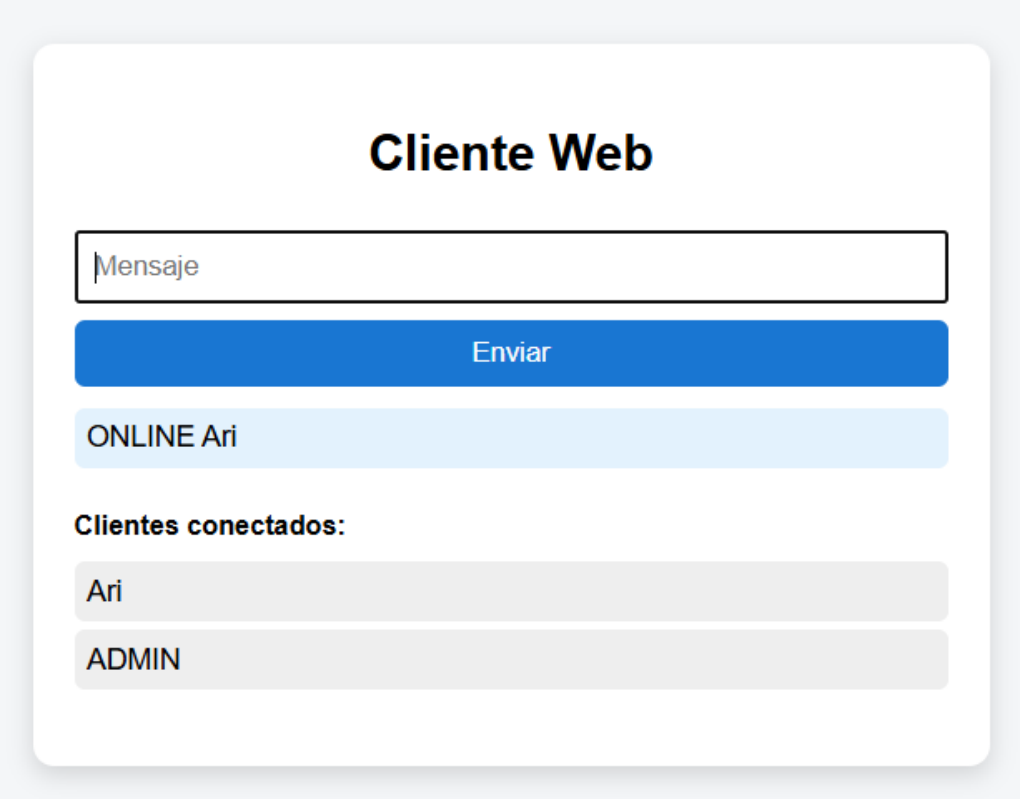


Cliente

En el caso del cliente debes acceder a un url diferente (Cliente Web disponible en http://localhost:8080/cliente.html) en donde te aparecerá una vista como la de la imagen. Si ya estabas registrado puedes poner tu nombre y contraseña, en caso de que seas nuevo puedes registrarte.



En cualquiera de los 2 casos al dar en entrar o en registrar, ambos te van a redireccionar a otra vista en donde puedes enviar mensajes al servidor, así como también recibir mensajes y notificaciones UDP, al igual que el admi puedes ver quienes están conectados.



Conclusiones

En si el proyecto como tal no estoy del todo orgullosa, pero pude aplicar de forma practica diversos fundamentos vistos en clase, además de que integre los protocoles que considero mas sencillos en una sola aplicación, aunque he de admitir aun me sigue causando confusión dado que mi idea era implementarlo diferente y ver la diferencia una de otra.

Pero como tal se lograron cumplir todos los requerimientos mínimos establecidos logrando así un pequeño sistema de chat aplicando la concurrencia y seguridad.

Claro de forma más a futuro buscaría mejorar esta implementación e incluso poder implementar el HTTPS, que en si intente en otros prototipos, pero me causaba muchos problemas, así como también gustaría poder implementar otros tipos de seguridad.

El trabajo en si me ayudo a ver que todo lo relacionado a redes no se me da en absoluto, pero es importante conocerlos ya que como tal la tecnología solo avanza a pasos gigantescos y el no estar conociendo estos puntos importantes de comunicación e incluso seguridad es como que no tendría mucho sentido mas que como tal esta relacionado a mi carrera, por lo que me agrada saber que si logro conocer aunque sea lo básico, tal vez no llegue a tener algo mas grande siguiendo las redes, pero en caso de que se necesite yo sabre como apoyar.