2DO PARCIAL - Tecnologías Web II

Estudiante: Marvin Mollo Ramirez 2do parcial Tecno. Web II

Implementación de un Microservicio con WebSocket para Notificaciones en Tiempo Real

1. Estructura del Proyecto y Configuración Inicial (15 pts)
   Identificación del Endpoint (5 pts):
   - - Endpoint WebSocket: ws://localhost:3000
   - - **Método**: Conexión persistente bidireccional (WebSocket)
   - - Eventos:
     - - REGISTER: Para registrar conexiones de administradores (id_cuenta: 2).
     - - NUEVO_REGISTRO: Notificación enviada cuando se registra un nuevo usuario.

- Justificación (5 pts):
  - **WebSocket vs REST**: Se eligió WebSocket para notificaciones en tiempo real sin necesidad de polling.
  - **Escalabilidad**: Ideal para flujos de eventos continuos (como registros en tiempo real).

Descripción Técnica (5 pts):

- Estructura de mensajes:

```
// Registro de admin (cliente → servidor)
{
    "type": "REGISTER",
    "id_cuenta": 2
  }

  // Notificación de nuevo registro (servidor → cliente)
  {
    "type": "NUEVO_REGISTRO",
    "data": {
      "nombres": "Ana",
      "apellido": "Gómez",
      "correo": "ana@ucb.edu.bo",
      "timestamp": "2024-05-05T12:00:00.000Z"
    }
  }
```

2. Diseño del Microservicio (15 pts)

Objetivo (5 pts):

Notificar en tiempo real al administrador (id_cuenta: 2) cuando un nuevo usuario se registra en el sistema.

Tecnología (5 pts): WebSocket: Protocolo full-duplex sobre TCP (ideal para notificaciones push).

Librería: ws para Node.js (ligera y compatible con estándares).

Diagrama de Flujo (5 pts):

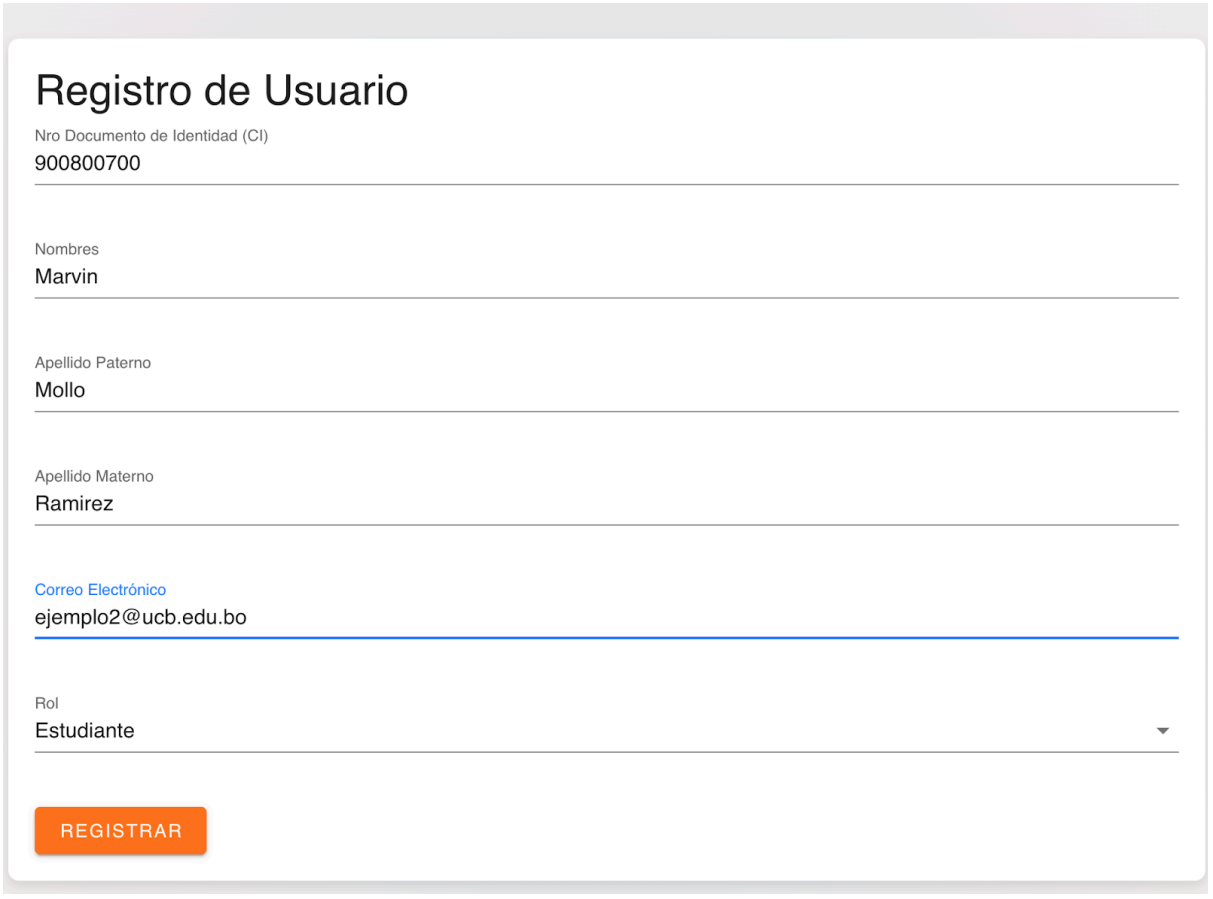
3. Implementación Técnica (40 pts)

Desarrollo del microservicio funcional (15 pts)

```
import { WebSocketServer } from 'ws';
const wss = new WebSocketServer({ port: 8083 });
```

```
const activeConnections = new Map();
wss.on('connection', (ws, req) => {
    console.log('Nuevo cliente WebSocket conectado');
    ws.on('message', (message) => {
        const data = JSON.parse(message);
        if (data.type === 'REGISTER_ADMIN' && data.id_cuenta === 1) { // ID 1 para
admin
            activeConnections.set(data.id_cuenta, ws);
            console.log(`Admin conectado: ${data.id_cuenta}`);
        }
    });
    ws.on('close', () => {
        activeConnections.forEach((value, key) => {
            if (value === ws) {
                activeConnections.delete(key);
                console.log(`Usuario desconectado: ${key}`);
            }
        });
    });
});
export function notifyNewUserRegistration(userData) {
    const adminConnection = activeConnections.get(2); // ID 1 para admin
    if (adminConnection && adminConnection.readyState === WebSocket.OPEN) {
        adminConnection.send(JSON.stringify({
            type: 'NUEVO_REGISTRO',
            data: {
                nombres: userData.nombres,
                apellido: userData.apellidoPat,
                correo: userData.correo,
                usuario: userData.usuario,
                fecha: new Date().toISOString()
            }
        }));
    } else {
        console.log('Admin no está conectado al WebSocket');
    }
}
```

Consumo correcto del endpoint externo (10 pts)



Procesamiento y transformación de los datos (10 pts)

```
export const Registrar = async (req, res) => {

   try {

        const { nombres, apellidoPat, apellidoMat, correo, ci, rol } = req.body;

        const user = await cuentasModel.buscarPorcorreo(correo);

        const dominioUcb = /^[a-zA-Z0-9._%+-]+@ucb\.edu\.bo$/;

        const letras = /^[a-zA-ZáéíóúÁÉÍÓÚñÑ\s]+$/;

        if (user) {

            return res.status(409).json({ ok: false, msg: "Este correo ya fue
registrado" });

        } else if (!letras.test(nombres) || !letras.test(apellidoPat) ||
!letras.test(apellidoMat)) {

            return res.status(409).json({ ok: false, msg: "Los nombres y apellidos
solo deben contener letras" });

        } else if (!dominioUcb.test(correo)) {

            return res.status(409).json({ ok: false, msg: "El correo debe pertenecer
al dominio @ucb.edu.bo" });

        } else if (ci.length < 4) {

            return res.status(409).json({ ok: false, msg: "El Carnet de identidad
ingresado es incorrecto" });

        } else {

            try {

                const newPerson = await cuentasModel.crearCuenta({ nombres,
apellidoPat, apellidoMat, correo, ci });

                const id_persona = newPerson[0].id_persona;

                const salt = await bcryptjs.genSalt(10);

                const contrasenia = await bcryptjs.hash(genPassword(ci,
apellidoPat), salt);

                const usuario = genUser(ci, apellidoPat);

                const newAccount = await cuentasModel.crearUsuario({

                    correo,

                    usuario,

                    contrasenia,
```

```
                    rol,
                    id_persona,
                    hab: 2
                });
                // Enviar notificación WebSocket al admin
                notifyNewUserRegistration({
                    nombres,
                    apellidoPat,
                    correo,
                    usuario,
                    id_cuenta: newAccount[0].id_cuenta // Asegúrate de que tu modelo devuelva esto
                });
                // Enviar correo
                await sendEmail({
                    to: correo,
                    subject: 'Cuenta registrada en UCB',
                    text: `Hola ${nombres} ${apellidoPat},\n\nTu cuenta ha sido registrada exitosamente.\n\nUsuario: ${usuario}\nContraseña: ${genPassword(ci, apellidoPat)}\n\nGuarda esta información.`
                });
                return res.status(201).json({ ok: true, msg: "Usuario registrado exitosamente" });
            } catch (error) {
                return res.status(409).json({ ok: false, msg: "Error al registrar: " + error });
            }
        }
    } catch (error) {
        return res.status(500).json({
            ok: false,
            msg: 'Error en el servidor: ' + error}}};
```

Exposición de endpoint propio/documentado (5 pts)

```
router.post('/Registrar-Usuario', cuentasController.Registrar);
```

4. Pruebas y Documentación (15 pts)

Evidencias Funcionales (5 pts):

```
Restarting 'src/index.js'
🚀 Servidor HTTP listo en http://localhost:3000
⚡ WebSocket activo en ws://localhost:3000
Nueva conexión WebSocket establecida
Mensaje WebSocket recibido: { type: 'REGISTER', id_cuenta: 2 }
Usuario registrado en WebSocket: 2
Nueva conexión WebSocket establecida
Mensaje WebSocket recibido: { type: 'REGISTER', id_cuenta: 2 }
Usuario registrado en WebSocket: 2
Admin no está conectado al WebSocket
Correo enviado correctamente
Admin no está conectado al WebSocket
Correo enviado correctamente
Admin no está conectado al WebSocket
Correo enviado correctamente
```

5. Presentación Final (15 pts)

Explicación Técnica (10 pts):

Este microservicio de WebSocket notifica en tiempo real al administrador (ID 2) cuando un nuevo usuario se registra. Al iniciar sesión, el panel admin establece una conexión WebSocket persistente con el backend y se registra usando {"type":"REGISTER","id_cuenta":2}. Cuando un usuario completa el formulario de registro, el backend envía automáticamente una notificación tipo NUEVO_REGISTRO con los datos del usuario al admin conectado. El frontend admin recibe estos datos y muestra alertas o actualiza su interfaz sin necesidad de refrescar. El servicio incluye reconexión automática si falla la red y filtra mensajes por tipo para seguridad. Usa el protocolo WebSocket (sobre TCP) para baja latencia y comunicación bidireccional.

Demostración Funcional (5 pts):


```
Restarting 'src/index.js'
🚀 Servidor HTTP listo en http://localhost:3000
⚡ WebSocket activo en ws://localhost:3000
Nueva conexión WebSocket establecida
Mensaje WebSocket recibido: { type: 'REGISTER', id_cuenta: 2 }
Usuario registrado en WebSocket: 2
Nueva conexión WebSocket establecida
Mensaje WebSocket recibido: { type: 'REGISTER', id_cuenta: 2 }
Usuario registrado en WebSocket: 2
Admin no está conectado al WebSocket
Correo enviado correctamente
Admin no está conectado al WebSocket
Correo enviado correctamente
Admin no está conectado al WebSocket
Correo enviado correctamente
```