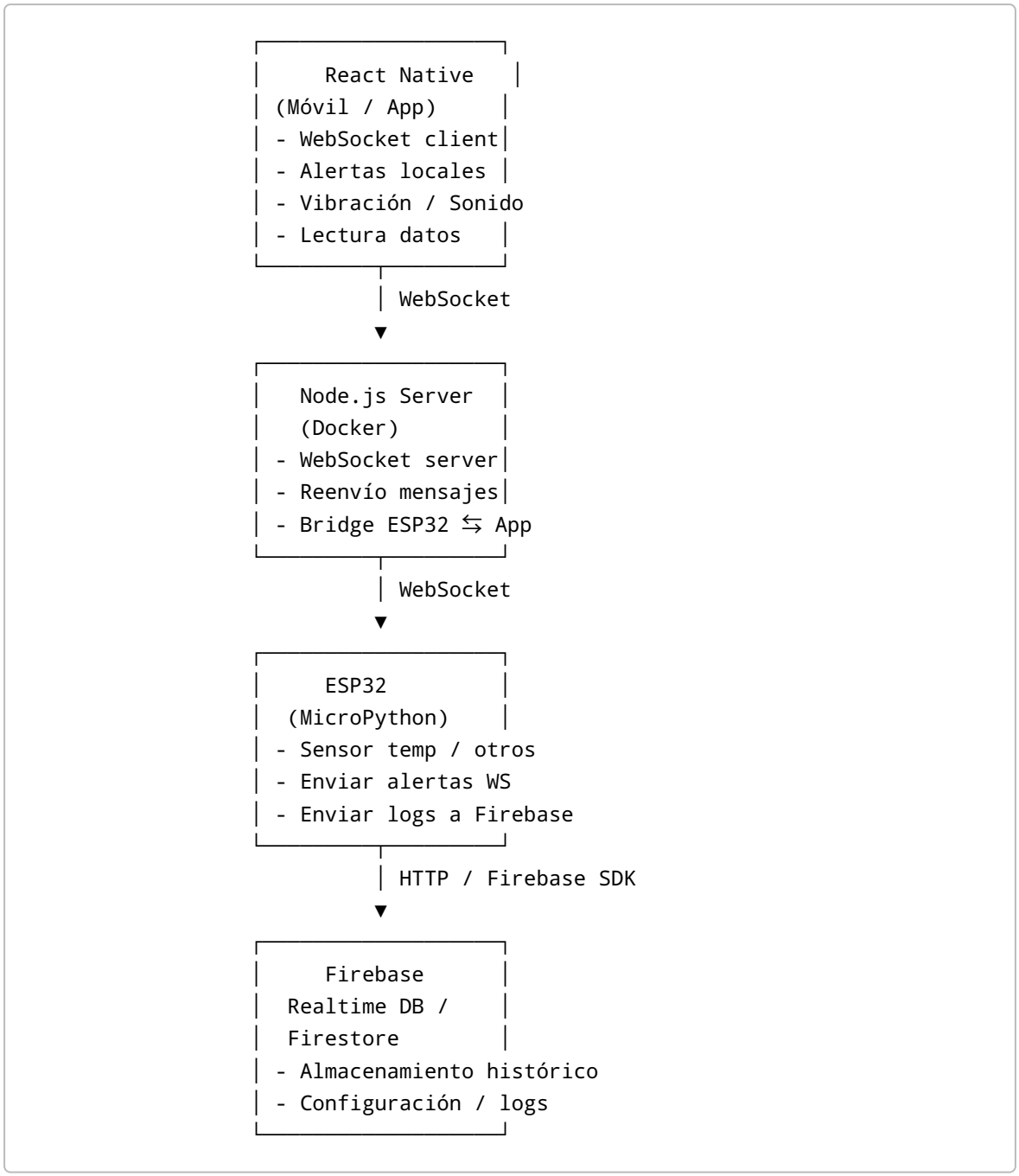
Proyecto IoT Completo: ESP32 + Node.js + React Native + Firebase

1. Arquitectura General



---

2. Flujo de Comunicación

- React Native ↔ Node.js ↔ ESP32: **control en tiempo real (WebSocket)**
- ESP32 → Firebase: **persistencia de datos / logs**
- React Native: **alertas locales** (vibración, sonido, notificación)

---

3. Node.js (Docker) - Código mínimo WebSocket

```
const WebSocket = require("ws");

const wss = new WebSocket.Server({ port: 8080 });

wss.on("connection", (ws) => {
  console.log("Cliente conectado");

  ws.on("message", (msg) => {
    console.log("Mensaje recibido:", msg.toString());

    // Reenvío a todos los demás clientes conectados
    wss.clients.forEach((client) => {
      if (client !== ws && client.readyState === WebSocket.OPEN) {
        client.send(msg.toString());
      }
    });
  });

  ws.on("close", () => {
    console.log("Cliente desconectado");
  });
});

console.log("Servidor WebSocket corriendo en ws://localhost:8080");
```

4. Dockerfile para Node.js

```
FROM node:18
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 8080
CMD ["node", "server.js"]
```

---

5. ESP32 (MicroPython) - Cliente WebSocket

```
import uasyncio as asyncio
import uwebsockets.client

async def main():
    uri = "ws://TU_SERVIDOR:8080"
    async with uwebsockets.client.connect(uri) as websocket:
        print("Conectado al servidor WebSocket")
```

```
        # Enviar un mensaje
        await websocket.send("Hola desde ESP32")

        # Escuchar mensajes
        while True:
            msg = await websocket.recv()
            print("Mensaje recibido:", msg)

asyncio.run(main())
```

---

6. React Native - Notificaciones locales y vibración

- Librerías recomendadas: `expo-notifications` o `react-native-push-notification`
- API de vibración: `Vibration` (built-in React Native)

Flujo de ejemplo: - Recibir mensaje WebSocket → Verificar temperatura → Si ≥ 80°C → Notificación + Vibración + Sonido

---

7. Firebase

- Solo para **persistencia de datos** (historial, configuración).
- ESP32 puede escribir directamente.
- React Native puede leer/escribir datos de configuración.
- Reglas de seguridad deben limitar acceso solo a tu app.

---

8. Recomendaciones para nivel avanzado destacado

1. **Seguridad**: Autenticación WebSocket (JWT), HTTPS/WSS, reglas Firebase estrictas.
2. **Escalabilidad**: Soporte multi-dispositivo, Docker Compose/K8s, logging y monitoreo.
3. **Optimización de recursos**: Filtrado/agrupación de datos antes de Firebase, manejo eficiente de ESP32.
4. **Funcionalidades extra**: Push notifications con Firebase Cloud Messaging, dashboard web con gráficos históricos.
5. **Experiencia de usuario y profesionalización**: UI/UX profesional, testeo unitario e integración, documentación clara.
6. **Ampliar ecosistema IoT**: Integrar más sensores, automatización de reglas, conexión con smart home o asistentes de voz.

---

9. Flujo recomendado de desarrollo

1. Configurar ESP32 con sensor y enviar datos por WebSocket.
2. Montar Node.js en Docker como servidor WebSocket.
3. Crear app React Native que reciba datos y active alertas locales.
4. Configurar Firebase para almacenar historial de datos.
5. Agregar seguridad mínima (credenciales .env, reglas Firebase).
6. Mejoras opcionales: push notifications, dashboard web, multi-dispositivo.

10. Conclusión

- Esta arquitectura separa responsabilidades: **control en tiempo real (WebSocket)** y **persistencia de datos (Firebase)**.
- Modular, escalable y eficiente: Node.js solo gestiona WebSocket, ESP32 se enfoca en hardware y React Native en UX.
- Nivel de programación alcanzado: **intermedio-avanzado a avanzado destacado**, combinando IoT, cloud, móvil y contenedores.

Listo para copiar/pegar o exportar como PDF para tener todo el proyecto documentado.