Manejo de Datos en el Código

1. Módulos Importados

- **http**: Proporciona la funcionalidad para crear un servidor HTTP que puede recibir solicitudes y enviar respuestas.
- **fs**: Permite la interacción con el sistema de archivos, lo que es crucial para leer el archivo HTML que se sirve a los clientes.
- **WebSocket**: Permite establecer conexiones WebSocket para la comunicación en tiempo real entre el servidor y los clientes.
- **net**: Proporciona la funcionalidad para crear un servidor TCP, permitiendo la comunicación a través de sockets.
- **path**: Ayuda a manejar las rutas de archivos de manera que se eviten problemas de compatibilidad entre sistemas operativos.

2. Definición de Constantes

- **hostname**: Define la dirección IP en la que el servidor estará escuchando.
- **httpPort**: Establece el puerto para el servidor HTTP (3000).
- **tcpPort**: Establece el puerto para el servidor TCP (3001).

3. Creación del Servidor HTTP

- **http.createServer(callback)**: Crea un servidor HTTP que maneja las solicitudes de los clientes.
    - **req**: Representa la solicitud del cliente. Contiene información como la URL solicitada, los encabezados, etc.
    - **res**: Representa la respuesta que se enviará al cliente. Se utiliza para enviar datos de vuelta al cliente.

Manejo de Solicitudes:

- **Condicional if (req.url === '/')**: Verifica si la URL solicitada es la raíz (**/**).
    - **fs.readFile(path.join(__dirname, 'index.html'), callback)**: Intenta leer el archivo **index.html** desde el directorio actual.
        - **Callback**: Se ejecuta una vez que se intenta leer el archivo.
            - **err**: Si hay un error (por ejemplo, si el archivo no existe), se establece un código de estado 500 (error interno del servidor).
            - **data**: Si la lectura es exitosa, se establece un código de estado 200 (OK) y se envía el contenido del archivo como respuesta.

- **Else**: Si la URL no es la raíz, se responde con un código 404 (no encontrado) y un mensaje de error.

4. Creación del Servidor WebSocket

- **new WebSocket.Server({ server })**: Crea un servidor WebSocket que se asocia con el servidor HTTP. Esto permite que los clientes se conecten a través de WebSocket.
- **wss.on('connection', callback)**: Escucha eventos de conexión. Cada vez que un cliente se conecta, se ejecuta el callback.
    - **ws.on('message', callback)**: Escucha mensajes enviados desde el cliente WebSocket.
        - **message**: Cada vez que se recibe un mensaje, se itera sobre el array **tcpClients** y se envía el mensaje a cada cliente TCP conectado.

5. Creación del Servidor TCP

- **net.createServer(callback)**: Crea un servidor TCP que puede aceptar conexiones de clientes.
- **Manejo de Conexiones TCP**:
    - **socket**: Cada vez que un cliente se conecta, se recibe un objeto **socket**, que representa la conexión con ese cliente.
    - **tcpClients.push(socket)**: Se agrega el socket del cliente al array **tcpClients**, que mantiene un registro de todos los clientes conectados.
    - **socket.on('data', callback)**: Escucha eventos de datos. Cuando se reciben datos del cliente, se ejecuta el callback.
        - **data.toString().trim()**: Convierte los datos recibidos a una cadena y elimina espacios en blanco. Esto es importante para procesar correctamente los comandos.
        - **socket.write(...)**: Dependiendo del comando recibido, se envía una respuesta al cliente TCP:
            - Si el comando es **on**, se envía "LED encendido".
            - Si el comando es **off**, se envía "LED apagado".
            - Si el comando no es reconocido, se envía "Comando no reconocido".

6. Escuchar en Puertos

- **server.listen(httpPort, hostname, callback)** - Inicia el servidor HTTP en el puerto definido (**httpPort**) y en la dirección IP especificada (**hostname**).

- **Callback**: Se ejecuta una vez que el servidor comienza a escuchar, imprimiendo un mensaje en la consola que indica la URL donde el servidor está disponible.

Resumen del Manejo de Datos

- **HTTP**: Se utiliza para manejar solicitudes de archivos y responder a los clientes con contenido estático. Cada solicitud es independiente y se maneja de manera asíncrona.
- **WebSocket**: Permite la comunicación en tiempo real, donde los mensajes enviados desde un cliente WebSocket se distribuyen a todos los clientes TCP conectados, facilitando la interacción en tiempo real.
- **TCP**: Se utiliza para establecer conexiones persistentes con los clientes, permitiendo el envío y recepción de comandos en tiempo real. Cada comando recibido se procesa y se responde adecuadamente.

Este enfoque permite que la aplicación maneje tanto la entrega de contenido web como la comunicación en tiempo real, combinando las fortalezas de HTTP y TCP para crear una experiencia interactiva y dinámica.