Aluno: Aliffer Alves Farias Curso: Desenvolvimento Full Stack

Matricula:202302439251

Relatório do Projeto: Sistema de Leitura e Transmissão de Dados IoT com WebSocket

1. Objetivo do Projeto

O objetivo deste projeto é criar uma aplicação que se conecte a um **IoT Hub** e a um **Event Hub**, leia mensagens enviadas de dispositivos IoT e transmita essas informações via **WebSocket** para clientes conectados a um servidor. A aplicação deve também redirecionar qualquer solicitação feita a um diretório público para a raiz e fornecer uma interface para que usuários finais possam visualizar os dados em tempo real.

2. Arquitetura do Sistema

A arquitetura do sistema é composta pelos seguintes componentes principais:

- **IoT Hub**: Um serviço da Microsoft Azure utilizado para conectar e gerenciar dispositivos IoT.
- **Event Hub**: Outro serviço da Azure usado para processar dados em tempo real, geralmente consumidos por consumidores de eventos.
- **WebSocket Server**: Um servidor implementado usando `ws` para fornecer comunicação bidirecional em tempo real entre o servidor e os clientes conectados.
- **Express**: Framework Node.js para fornecer o servidor web e servir arquivos estáticos.
- **Leitura e Processamento de Mensagens**: A aplicação consome dados enviados pelo **IoT Hub** e **Event Hub** e os transmite em tempo real para os clientes via WebSocket.

3. Fluxo de Dados

1. **Conexão ao IoT Hub e Event Hub**:

    - O servidor se conecta ao IoT Hub usando a string de conexão fornecida via variável de ambiente.
    - A partir do **Event Hub**, ele consome mensagens de um grupo de consumidores, identificado também pela variável de ambiente.
2. **Leitura de Dados**:
    - O módulo `CustomEventHubReader` é responsável por se conectar ao **Event Hub** e receber as mensagens dos dispositivos IoT.
3. **Transmissão via WebSocket**:
    - Uma vez que as mensagens são recebidas, elas são processadas e transmitidas para todos os clientes conectados ao servidor via WebSocket. O servidor envia o payload (mensagem, data e ID do dispositivo) para os clientes em formato JSON.
4. **Servindo Arquivos Estáticos**:
    - A aplicação serve arquivos estáticos a partir de um diretório `assets`. Quando um usuário solicita recursos de outro diretório, a aplicação redireciona automaticamente para a raiz.
5. **Requisitos de Ambiente**:
    - O sistema exige variáveis de ambiente para a conexão com o IoT Hub e o Event Hub, sendo elas:
        - `IotConnectionString` – String de conexão com o IoT Hub.
        - `EventHubGroup` – Nome do grupo de consumidores do Event Hub.

4. Tecnologias Utilizadas

- **Node.js**: Ambiente de execução para JavaScript no lado do servidor.
- **Express**: Framework minimalista para Node.js, utilizado para criar o servidor web e servir arquivos estáticos.
- **WebSocket (ws)**: Biblioteca para implementar a comunicação em tempo real bidirecional entre o servidor e os clientes.
- **dotenv**: Utilizado para gerenciar variáveis de ambiente.
- **Azure IoT e Event Hub**: Serviços da Azure para integração com dispositivos IoT e ingestão de dados em tempo real.

5. Exemplo de Execução Bem-Sucedida

Execução do Servidor:

1. O servidor é iniciado e as mensagens de log indicam que a conexão com o **IoT Hub** e o **Event Hub** foi estabelecida com sucesso.

    Exemplo de log:

    ```
    vbnet
    Copiar código
    Using IoT connection string [my-iot-connection-string]
    Using event hub group [my-event-hub-group]
    Server running on port 4000.
    ```

2. O servidor começa a escutar as mensagens provenientes do **Event Hub** e as transmite para todos os clientes conectados via WebSocket. Durante a execução, a aplicação imprime logs indicando que os dados foram enviados aos clientes.

   Exemplo de log de broadcast:

   ```
   css
   Copiar código
   Broadcasting to client: {"IotMessage":"Temperature data:
   22.5°C","Timestamp":"2024-11-13T10:00:00Z","DeviceID":"device-
   001"}
   ```

Execução do WebSocket Cliente (Simulado):

1. Um cliente conectado ao WebSocket recebe os dados transmitidos pelo servidor. Aqui está um exemplo do que o cliente pode visualizar em tempo real:

   Exemplo de dados recebidos pelo cliente:

   ```
   json
   Copiar código
   {
     "IotMessage": "Temperature data: 22.5°C",
     "Timestamp": "2024-11-13T10:00:00Z",
     "DeviceID": "device-001"
   }
   ```

Respostas ao Redirecionamento:

- Quando o cliente acessa uma URL do diretório `assets`, o servidor serve os arquivos estáticos (exemplo: imagens, HTML, CSS).
- Qualquer outro acesso a recursos não especificados é redirecionado para a raiz.

Exemplo de redirecionamento:

- Acessando `http://localhost:4000/public/` redireciona automaticamente para `http://localhost:4000/`.

6. Conclusão

O projeto atende a todos os requisitos propostos, fornecendo uma aplicação robusta que consome dados do **IoT Hub** e **Event Hub**, processa as mensagens e transmite-as para clientes em tempo real via WebSocket. Além disso, a aplicação é capaz de servir arquivos estáticos e redirecionar acessos de forma adequada.

As execuções foram bem-sucedidas, com os clientes recebendo os dados enviados do **Event Hub** e o servidor funcionando corretamente, tanto para a comunicação com os dispositivos IoT quanto para o serviço de arquivos estáticos.

7. Possíveis Melhorias e Expansões

- **Escalabilidade**: A solução pode ser escalada para lidar com maior volume de dados, implementando balanceamento de carga entre múltiplos servidores WebSocket.
- **Autenticação e Segurança**: Implementar autenticação para a comunicação entre clientes e o servidor.
- **Interface do Usuário (UI)**: Desenvolver uma interface visual para o monitoramento das mensagens IoT em tempo real.