1. Wczytywanie danych z pliku konfiguracyjnego

Plik konfiguracyjny config.txt jest wczytywany, aby pobrać adres serwera OPC UA (pierwsza linia) oraz listę urządzeń i ich connection stringi (kolejne linie). Adres serwera OPC UA jest zapisywany w zmiennej, a dane o urządzeniach są dodawane do słownika deviceConnectionStrings.

```
string filePath = "config.txt";

string[] lines = File.ReadAllLines(filePath);


// Wczytanie adresu OPC UA serwera (pierwsza linia)

string opcServerAddress = lines[0];


// Dodanie połączeń do urządzeń z kolejnych linii

foreach (string line in lines.Skip(1))

{

  var parts = line.Split(',');

  if (parts.Length == 2)

  {

    string deviceId = parts[0].Trim();

    string connectionString = parts[1].Trim();

    deviceConnectionStrings.Add(deviceId, connectionString);

  }

}
```

2. Inicjalizacja połączenia z urządzeniami

Każde urządzenie (device) nawiązuje połączenie z Azure IoT za pomocą connection stringu. Dla każdego urządzenia ustawia się obsługę bezpośrednich metod (EmergencyStop, ResetErrorStatus), a także monitoruje zmiany w Device Twin za pomocą metody MonitorDeviceTwinChangesAsync.

```
foreach (var deviceId in deviceConnectionStrings.Keys)
```

```
{
   var deviceClient =
DeviceClient.CreateFromConnectionString(deviceConnectionStrings[deviceId],
TransportType.Mqtt);
   deviceClients.Add(deviceId, deviceClient);


   await deviceClient.SetMethodHandlerAsync("EmergencyStop",
EmergencyStopMethodHandler, deviceId);
   await deviceClient.SetMethodHandlerAsync("ResetErrorStatus",
ResetErrorStatusMethodHandler, deviceId);


   await MonitorDeviceTwinChangesAsync(deviceId);
}
```

3. Monitorowanie i odczytywanie danych z OPC UA

Dla każdego urządzenia odczytywane są dane telemetryczne z serwera OPC UA za pomocą komend OpcReadNode. Wyniki są zapisywane jako telemetryData, które następnie są wysyłane do Azure IoT Hub. Odczytywane są kluczowe dane, takie jak: status produkcji, temperatura, ilość dobrych i złych produktów oraz ewentualne błędy urządzenia.

```
foreach (var deviceId in deviceClients.Keys)
{
   OpcReadNode[] commands = new OpcReadNode[] {
      new OpcReadNode($"ns=2;s={deviceId}/ProductionStatus"),
      new OpcReadNode($"ns=2;s={deviceId}/ProductionRate"),
      new OpcReadNode($"ns=2;s={deviceId}/Temperature"),
      new OpcReadNode($"ns=2;s={deviceId}/GoodCount"),
      new OpcReadNode($"ns=2;s={deviceId}/BadCount"),
      new OpcReadNode($"ns=2;s={deviceId}/DeviceError"),
   };
```

```
    IEnumerable<OpcValue> values = opcClient.ReadNodes(commands);

    var telemetryData = new
    {
        DeviceId = deviceId,
        ProductionStatus = values.ElementAt(0).Value,
        Temperature = values.ElementAt(2).Value,
        GoodCount = values.ElementAt(3).Value,
        BadCount = values.ElementAt(4).Value
    };

    await SendMessageToIoTHub(deviceId, telemetryData);
}
```

4. Obsługa zmiany parametru Device Twin (Production Rate)

Funkcja reaguje na zmiany w desired properties Device Twin, w szczególności na parametr ProductionRate. Gdy pojawi się nowa wartość, wywoływana jest funkcja ChangeProductionRate, która zmienia szybkość produkcji urządzenia, a następnie aktualizowany jest raportowany status tej wartości w Azure IoT.

```
private static async Task OnDesiredPropertyChanged(TwinCollection desiredProperties,
object userContext)
{
    string deviceId = (string)userContext;

    if (desiredProperties.Contains("ProductionRate"))
    {
        int desiredProductionRate = desiredProperties["ProductionRate"];
        ChangeProductionRate(deviceId, desiredProductionRate);
        await UpdateReportedProductionRateAsync(deviceId, desiredProductionRate);
```

```
    }
}
```

5. Wysłanie wiadomości telemetrycznej do Azure IoT Hub

Funkcja SendMessageToIoTHub serializuje dane telemetryczne urządzenia do formatu JSON i tworzy wiadomość do wysłania na Azure IoT Hub. Do wiadomości dodawany jest typ (MessageType: Telemetry), a następnie wiadomość jest wysyłana za pomocą metody SendEventAsync.

```
private static async Task SendMessageToIoTHub(string deviceId, object telemetryData)
{
    var messageString = JsonConvert.SerializeObject(telemetryData);
    var message = new Message(Encoding.ASCII.GetBytes(messageString));
    message.Properties.Add("MessageType", "Telemetry");
    await deviceClients[deviceId].SendEventAsync(message);
}
```

6. Zarządzanie błędami urządzeń

Jeśli wykryty błąd urządzenia (deviceErrorCode) różni się od aktualnie raportowanego w Device Twin, kod wysyła nową wiadomość o błędzie do IoT Hub i aktualizuje właściwości Device Twin. Wiadomość zawiera kod błędu, wiadomość o błędzie oraz czas wystąpienia.

```
if (deviceErrorCode != reportedDeviceError && deviceErrorCode != 0)
{
    var deviceError = new
    {
        DeviceId = deviceId,
        ErrorCode = deviceErrorCode,
        ErrorMessage = errorMessage,
        Timestamp = DateTime.UtcNow
    };
```

```
    await SendDeviceErrorEventToIoTHub(deviceId, deviceError);
    await UpdateReportedDeviceErrorAsync(deviceId, deviceErrorCode, errorMessage);
}
```

7. Obsługa Direct Methods (Emergency Stop i Reset Error Status)

Dwie funkcje obsługujące **Direct Methods**: EmergencyStop i ResetErrorStatus. Każda z tych metod przyjmuje zapytanie typu MethodRequest oraz kontekst urządzenia (userContext), który identyfikuje urządzenie, dla którego metoda została wywołana.

- **Emergency Stop** zatrzymuje urządzenie poprzez wywołanie logiki PerformEmergencyStop.
- **Reset Error Status** resetuje status błędów urządzenia za pomocą funkcji ResetDeviceErrorStatus.

Obie metody zwracają odpowiedź w formacie JSON z kodem 200, jeśli operacja została pomyślnie wykonana.

```
private static Task<MethodResponse> EmergencyStopMethodHandler(MethodRequest methodRequest, object userContext)
{
    string deviceId = (string)userContext;
    Console.WriteLine($"Emergency Stop triggered for device {deviceId}.");

    // Logika zatrzymania urządzenia
    PerformEmergencyStop(deviceId);

    string result = "{\"result\":\"Emergency Stop executed successfully\"}";
    return Task.FromResult(new MethodResponse(Encoding.UTF8.GetBytes(result), 200));
}

private static Task<MethodResponse> ResetErrorStatusMethodHandler(MethodRequest methodRequest, object userContext)
{
    string deviceId = (string)userContext;
```

```
    Console.WriteLine($"Reset Error Status triggered for device {deviceId}.");

    // Logika resetowania błędów urządzenia
    ResetDeviceErrorStatus(deviceId);

    string result = "{\"result\":\"Error status reset successfully\"}";
    return Task.FromResult(new MethodResponse(Encoding.UTF8.GetBytes(result), 200));
}
```