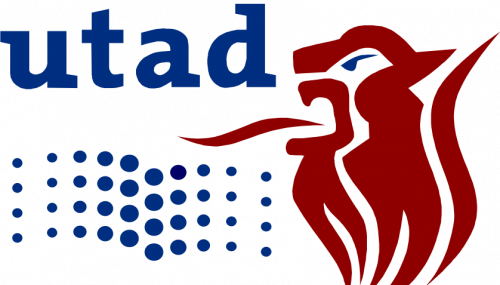
**Relatório Técnico:**

**Sistema de Monitorização em Tempo Real e API REST para Integração de Dados**



**Realizado por:**

**Al 70831 José Eduardo Martins Catarino**

**Al 75893 Alexandre Miranda Madureira Freitas**

**1. Introdução**

**Este relatório descreve a implementação de dois subsistemas principais:**

1. **Painel de Monitorização em Tempo Real (GUI\_Stream) – Interface gráfica para visualização de métricas de produção.**
2. **Sistema de Monitorização de Falhas (MonitorizaçãoFalha) – Aplicação dedicada ao rastreamento de peças defeituosas.**
3. **API REST para Integração de Dados – Disponibiliza endpoints para gestão de produtos e testes.**

**2. Painel de Monitorização em Tempo Real (GUI\_Stream)**

**2.1. Objetivo**

**Desenvolver uma interface gráfica que exibe métricas críticas de produção em tempo real, consumindo dados de um *stream* RabbitMQ.**

**2.2 Decisão de Design**

* **Segurança entre Threads: Uso de Invoke para atualização segura da GUI.**
* **Persistência do Stream: Configuração de MaxLengthBytes para armazenamento prolongado.**
* **Implementamos o RabbitMQ Streams para viabilizar o processamento assíncrono e não destrutivo das informações, assegurando que múltiplos consumidores possam consultar o histórico de dados sem interferência.**
* **Codificamos os registos de produção em JSON.**
* **A divisão em módulos distintos permite que vários utilizadores com diferentes necessidades acedam aos dados que são adequados às suas funções.**

**2.3. Implementação**

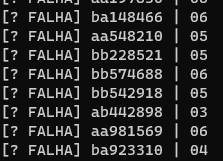
**2.3.1. Sender**

Publica continuamente mensagens simuladas no stream. Cada mensagem contêm toda a informação detalhada da Peça.



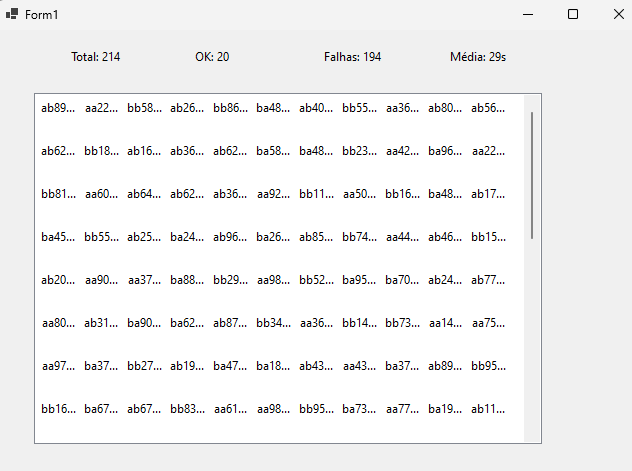
**2.3.2. Receiver (Consumer CLI)**

**Este lê as mensagens desde o início e acumula estatísticas em tempo real.**



**2.3.3. Consumer GUI (Forms)**

**Este mostra uma interface gráfica com labels e uma ListView que mostra todo o histórico (codigo\_peca) à medida que chegam os dados).**



**3. RabbitMQ Topics**

**3.1. Objetivo**

**Monitorar peças com falhas em tempo real, consumindo mensagens do tópico "peca.falha" no RabbitMQ.**

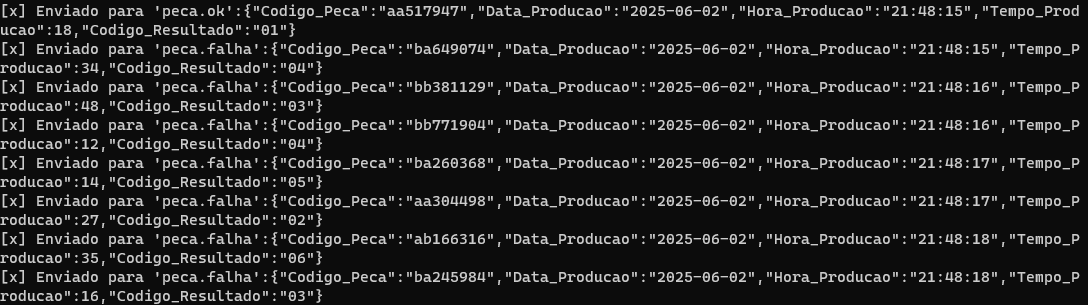
**3.2. Decisão de Design**

* **Foi implementado um modelo Publisher/Subscriber com tópicos.**
* **Dividimos o sistema em 3 Aplicações Independentes (Sender, Receiver e uma GUI)**

**3.3. Implementação**

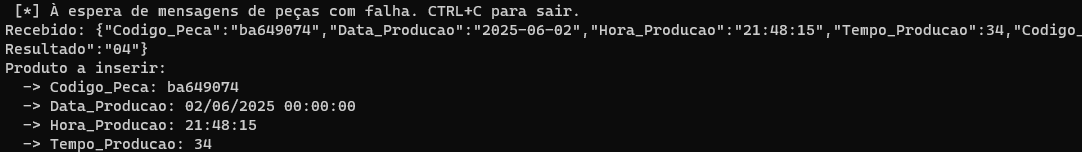
**3.3.1. Sender**

**O Sender vai enviar mensagens de produção com os vários dados das peças codificados em JSON.**



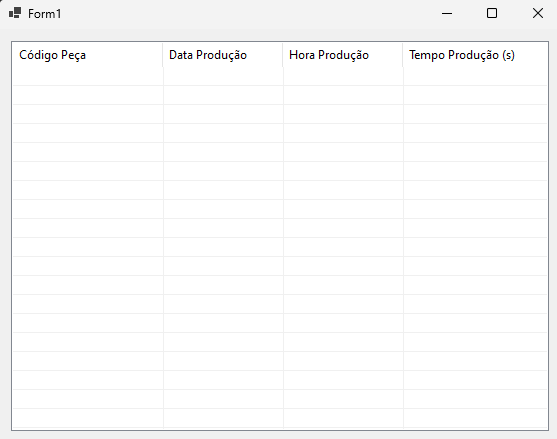
**3.3.2. Receiver**

**O Receiver vai receber as mensagens de peças com falha, vai processar esses dados e enviá-los para a plataforma Web.**

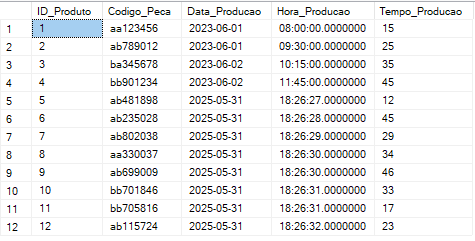


**3.3.3. GUI (Windows Forms)**

**Resolvemos utilizar uma GUI (Forms) de forma a mostrar em tempo real numa ListView, as peças que falharam o teste.**



Os dados também são atualizados na Base de Dados:



**No rabbitMQ Management podemos confirmar:**



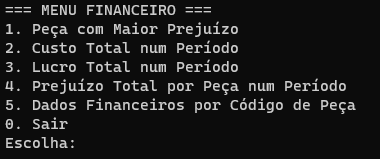
**4. Integração SOAP**

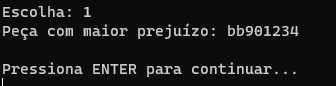
**4.1. Decisão de Design**

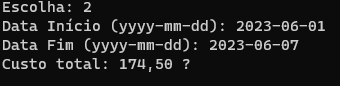
* Utilizamos o modelo Web Services .asmx em ASP.NET para garantir compatibilidade com sistemas legados.

**4.2. Implementação**

**SOAP (WebServiceClient)**

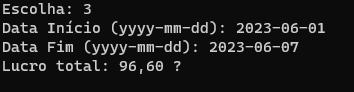
**Começamos com um Pequeno Menu:**  


Escolhendo a Opção 1:  


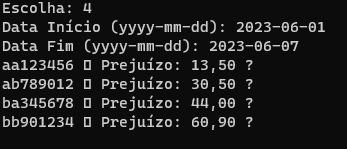
Devolve a Peça com maior prejuízo existente na Base de Dados.  
  
Escolhendo a Opção 2:  


É nos pedido um período de Datas para ser calculado o valor total de Custo nesse período.

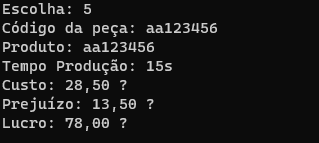
Escolhendo a Opção 3:



É nos pedido um período de Datas para ser calculado o valor total de Lucro nesse período.

Escolhendo a Opção 4:  


É nos pedido um período de Datas para ser calculado o valor de Prejuízo por peça nesse período.

Escolhendo a Opção 5:  


Por fim nesta opção é nos pedido o código da Peça em questão e devolve-nos todos os dados sobre a Peça.

Todas as opções podem ser confirmadas a partir das tabelas criadas nas Bases de Dados. E a partir do Browser aberto ao dar “Start” na aplicação “WebSOAP”.

**5. Conclusão**

**Os sistemas desenvolvidos permitem:**

* **Monitorização em tempo real da produção com métricas críticas.**
* **Identificação rápida de falhas através de filtros específicos.**
* **Integração segura via API REST com validações robustas.**

**No desenvolvimento da aplicação tivemos vários obstáculos, incluindo alguns que não nos foi possível resolver incluindo a “Não Atualização” da GUI do ReceiverCom\_Topicos, e também não conseguimos implementar corretamente para que nesta aplicação o Receiver ao Funcionar com a API REST enviasse tanto as Peças novas geradas como os testes das mesmas Peças, neste caso os Testes não eram enviados para a Base de Dados, mas as Peças sim.**