Trabalho realizado por:   
Mafalda Leitão al74728  
Nuno Ferreira al76889

Trabalho prático 2

Mestrado em Engenharia Informática

# Introdução

O projeto realizado consistiu na construção de uma API SOAP, responsável por disponibilizar um conjunto de operações financeiras críticas, nomeadamente o apuramento de resultados financeiros, cálculo de lucros, prejuízos e custos associados a períodos e a operações específicas. Este serviço foi desenvolvido com recurso à framework .NET Framework e à tecnologia WCF, tirando partido da sua capacidade de definição rigorosa de contratos de serviço e integração eficiente com bases de dados relacionais SQL Server, através da invocação de stored procedures previamente existentes.

A escolha da tecnologia WCF foi justificada pela sua adequação para a construção de serviços empresariais baseados em SOAP, assegurando a compatibilidade com o paradigma tradicional de integração de sistemas e a interoperabilidade com outras aplicações através de contratos WSDL. Este modelo de integração permite garantir a formalização das interfaces e a definição precisa dos métodos disponibilizados, promovendo a robustez e a consistência da comunicação entre sistemas heterogéneos.

Paralelamente ao desenvolvimento da API SOAP, procedeu-se à criação de uma aplicação cliente WPF (Windows Presentation Foundation), cuja função principal consiste em consumir os serviços disponibilizados pela API. Esta aplicação gráfica permite ao utilizador consultar dados financeiros e gerar relatórios de forma intuitiva e interativa, com recurso à funcionalidade Connected Services do Visual Studio para a ligação direta ao serviço SOAP implementado. A aplicação foi concebida para apresentar os resultados de forma clara e acessível, promovendo a usabilidade e facilitando a interpretação dos dados financeiros processados pela API.

# Decisões de Design e Implementação da API SOAP

A presente secção tem como finalidade descrever as opções de conceção e implementação associadas ao desenvolvimento da API SOAP, que constituiu a componente central do sistema de serviços financeiros desenvolvido no âmbito deste projeto. A integração baseada em SOAP visou proporcionar um conjunto de operações financeiras essenciais — nomeadamente o apuramento de lucros, prejuízos e custos associados a diferentes períodos e operações — através de uma interface formalmente definida, interoperável e centralizada. Esta decisão surgiu no contexto da necessidade de modernização dos sistemas de informação financeiros e da consequente centralização da lógica de negócio num serviço único e acessível a partir de diferentes aplicações e plataformas.

Para a concretização deste objetivo, optou-se pela utilização da tecnologia Windows Communication Foundation (WCF), sobre a framework .NET Framework 4.8.1, justificando-se esta escolha pelas características distintivas do WCF, nomeadamente o suporte nativo ao protocolo SOAP, a possibilidade de definição formal de contratos de serviço através de WSDL (Web Services Description Language), a facilidade de configuração de bindings e endpoints, bem como a integração eficiente com bases de dados SQL Server. Adicionalmente, a maturidade da tecnologia e a sua compatibilidade com ambientes empresariais tradicionais, frequentemente suportados por serviços SOAP, constituíram fatores determinantes na decisão.

A implementação da API iniciou-se pela definição dos DataContracts e ServiceContracts, elementos estruturantes de um serviço WCF. Os DataContracts foram concebidos para representar as diferentes entidades financeiras manipuladas pelo serviço, como relatórios de resultados, dados de lucros e prejuízos, e listas de operações. Estes contratos asseguraram a correta serialização e interoperabilidade dos objetos transportados entre o serviço e os respetivos consumidores, garantindo simultaneamente a integridade dos dados no processo de comunicação.

Posteriormente, proceder-se-ia à definição dos ServiceContracts, responsáveis pela descrição das operações disponibilizadas pelo serviço. Entre os métodos implementados destacam-se CalcularLucroPorPeriodo, ObterCustoTotal, ObterOperacoesPorTipo e ObterPeçaComMaiorPrejuizo. Cada uma destas operações foi devidamente anotada com o atributo [OperationContract], assegurando a sua inclusão no contrato formal do serviço e respetiva disponibilização no WSDL. Esta abordagem garantiu a padronização da interface, promovendo a interoperabilidade com eventuais aplicações externas ou sistemas de terceiros.

No que se refere ao acesso e manipulação de dados, optou-se por manter a lógica financeira crítica concentrada no servidor de base de dados, recorrendo a stored procedures previamente implementadas na base de dados SQL Server. Esta decisão teve por base a necessidade de preservar a consistência e a integridade dos processos financeiros existentes, assegurando simultaneamente a eficiência no processamento de dados e a compatibilidade com as estruturas de dados já estabelecidas. Para concretizar esta comunicação, foi desenvolvida uma camada de acesso a dados (Data Access Layer — DAL), cuja função consistiu em encapsular todas as operações de interação com a base de dados, promovendo o princípio de separação de responsabilidades e garantindo uma maior modularidade e manutenibilidade da solução.

A execução das stored procedures foi concretizada através da biblioteca System.Data.SqlClient, com recurso a comandos parametrizados que asseguraram a integridade e segurança na passagem de parâmetros. Esta abordagem permitiu obter os resultados financeiros de forma eficiente e estruturada, devolvendo-os posteriormente à camada de serviço, que, por sua vez, procedeu à sua exposição aos consumidores da API.

Relativamente à configuração da infraestrutura de comunicação, esta foi realizada integralmente no ficheiro App.config, onde se definiram os parâmetros de comunicação necessários à disponibilização do serviço. A configuração incluiu a definição de um endpoint SOAP com binding basicHttpBinding, selecionado pela sua simplicidade e ampla compatibilidade com consumidores SOAP externos, bem como pela adequação aos requisitos funcionais e de interoperabilidade do projeto. Definiram-se ainda os tempos de timeout, a política de segurança e o comportamento de metadados (ServiceMetadataBehavior), permitindo a publicação automática do WSDL para validação e consumo pelos respetivos clientes.

Concluída a implementação da API SOAP, e no seguimento de um requisito adicional definido no decurso do projeto, procedeu-se ao desenvolvimento de uma aplicação cliente WPF (Windows Presentation Foundation), destinada a consumir os serviços disponibilizados pela API. Para tal, recorreu-se à funcionalidade Connected Services do Visual Studio, que possibilitou a ligação automática ao serviço SOAP e a geração das classes proxy necessárias para a comunicação com a API. A aplicação gráfica foi concebida de forma a garantir uma utilização intuitiva e funcional, disponibilizando menus e formulários para a consulta de relatórios financeiros, verificação de custos e apuramento de prejuízos, bem como outras operações disponibilizadas pela API.

A articulação entre a aplicação cliente WPF e a API SOAP permitiu validar a operacionalidade dos métodos implementados e assegurar a consistência dos resultados apresentados face aos dados registados na base de dados. Este processo de integração evidenciou a eficácia do modelo de comunicação adotado, demonstrando a viabilidade da solução proposta para ambientes empresariais que exijam a centralização e interoperabilidade de serviços financeiros críticos.

A abordagem adotada contribuiu para a aplicação prática de princípios de arquitetura orientada a serviços (SOA), evidenciando boas práticas no que respeita à formalização de contratos de serviço, separação da camada de dados, configuração parametrizada de endpoints e desenvolvimento de soluções interoperáveis. Adicionalmente, possibilitou a consolidação de conhecimentos no domínio da integração de sistemas baseados em serviços SOAP, com enfoque na centralização da lógica de negócio e na uniformização de interfaces de acesso a dados críticos.

# Decisões de Design e Implementação Do Publisher e Consumer de Dados

No âmbito da modernização do sistema de informação e substituição de soluções legadas, surgiu a necessidade de conceber e implementar uma nova aplicação responsável pela publicação, em tempo real, dos dados de produção para um sistema de mensageria, de forma a permitir o desacoplamento entre a geração e o processamento das informações. Esta decisão procurou assegurar a escalabilidade e a flexibilidade da arquitetura do sistema, permitindo a sua evolução futura sem comprometer os componentes já existentes.

Com este propósito, decidiu-se pela utilização do RabbitMQ, atuando como message broker, estabelecendo-se um modelo de comunicação assíncrona baseado no padrão publish/subscribe. A escolha do RabbitMQ justificou-se pela sua leveza, maturidade, elevada aceitação no meio empresarial e suporte nativo para múltiplos padrões de troca de mensagens, nomeadamente fanout, direct, topic e headers exchanges. Acresce referir que esta solução foi a que se revelou mais acessível e ajustada ao contexto pedagógico do projeto, tendo igualmente sido abordada em profundidade no decurso das aulas práticas.

A aplicação produtora foi desenvolvida sob a forma de aplicação de consola (Console Application), concebida para possibilitar a publicação de mensagens de forma manual ou automática. Para tal, foi criado um menu interativo, permitindo ao utilizador optar entre três modos de operação distintos: geração de uma única peça, geração de um conjunto de cinco peças ou execução em modo autónomo, simulando a produção contínua com intervalos de tempo controlados por lógica interna. Cada peça produzida é publicada como uma mensagem JSON, contendo atributos relevantes como o identificador da peça, o estado (bom ou defeituoso) e o tempo de produção. Esta abordagem garantiu a uniformização do formato das mensagens e facilitou a respetiva análise e tratamento posterior.

Relativamente ao modelo de distribuição de mensagens, optou-se pela criação de um exchange do tipo fanout, permitindo disseminar todas as mensagens publicadas para os diferentes consumidores subscritos, independentemente da lógica de filtragem ou número de subscritores existentes. Esta decisão foi motivada pela simplicidade de configuração e gestão associada a este tipo de exchange, bem como pela sua adequação a cenários onde se pretende a distribuição simultânea de dados para múltiplos sistemas, como é o caso da monitorização, armazenamento e análise em tempo real.

Com vista a assegurar a receção e tratamento eficiente dos dados de produção, procedeu-se ao desenvolvimento de uma aplicação consumidora, com interface gráfica, recorrendo à framework WPF (Windows Presentation Foundation). Esta aplicação foi projetada para consumir, de forma contínua, as mensagens disponibilizadas pelo RabbitMQ, apresentando os dados em tempo real ao utilizador. Além da leitura e visualização dos dados, a aplicação implementa funcionalidades adicionais, nomeadamente:

* Notificação imediata da chegada de cada nova peça.
* Cálculo de métricas relevantes, como o número total de peças produzidas, peças em bom estado, peças com falha e tempo médio de produção.
* Visualização gráfica e estatística do desempenho produtivo.
* Envio de cada mensagem processada para uma API REST, desenvolvida especificamente para esse efeito, permitindo a persistência dos dados em base de dados relacional para posterior análise.

Para potenciar a capacidade de processamento em tempo real e garantir a retenção durável das mensagens, explorou-se a integração com o plugin RabbitMQ Stream, ferramenta que possibilita o armazenamento e consumo contínuo de fluxos de dados. Embora tenha sido considerada a utilização do Apache Kafka como alternativa, a decisão recaiu sobre a manutenção do RabbitMQ e respetivo módulo Stream, visando a coerência tecnológica com o ecossistema já implementado e a redução da complexidade associada à instalação e gestão de clusters Kafka.

A aplicação consumidora foi concebida para detetar automaticamente novas mensagens no stream configurado, processando-as e atualizando a interface de forma imediata. Adicionalmente, cada mensagem consumida é enviada, via HTTP, para a API REST, garantindo a persistência dos dados em base de dados e permitindo a sua reutilização para relatórios ou análises futuras. A integração desta funcionalidade contribuiu para a criação de um pipeline completo de recolha, disseminação, processamento e armazenamento de dados de produção.

Esta abordagem modular e baseada em sistemas de mensageria assegurou um elevado nível de desacoplamento entre os diferentes componentes do sistema, promovendo a flexibilidade, escalabilidade e interoperabilidade da solução desenvolvida. As opções de design adotadas permitiram ainda a aplicação prática de conceitos de integração de sistemas distribuídos, nomeadamente no que respeita à comunicação assíncrona, gestão de mensagens, monitorização em tempo real e integração de serviços REST.

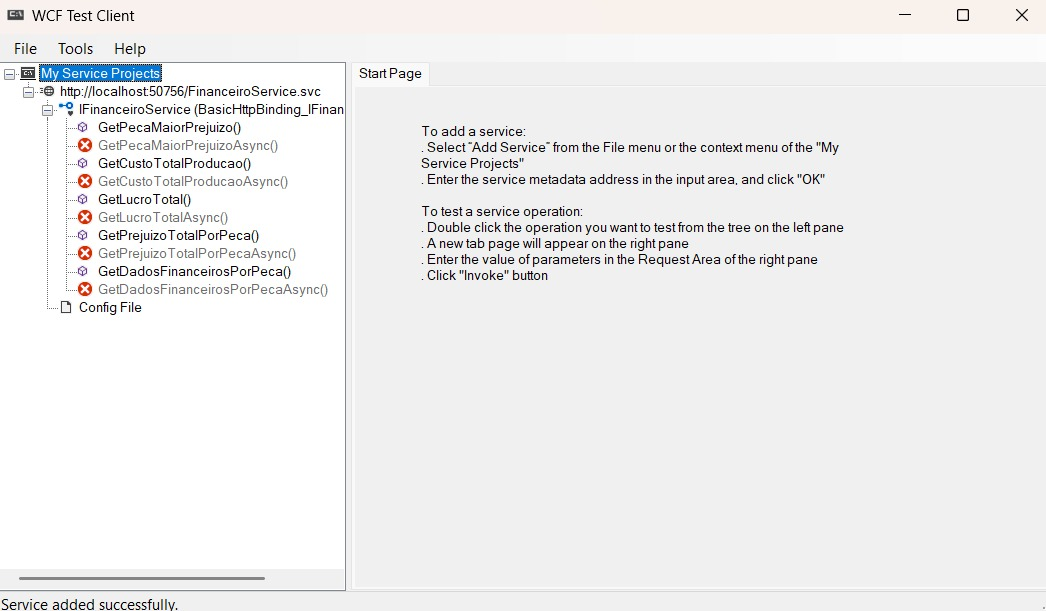
# Testes e Validação das Soluções Desenvolvidas

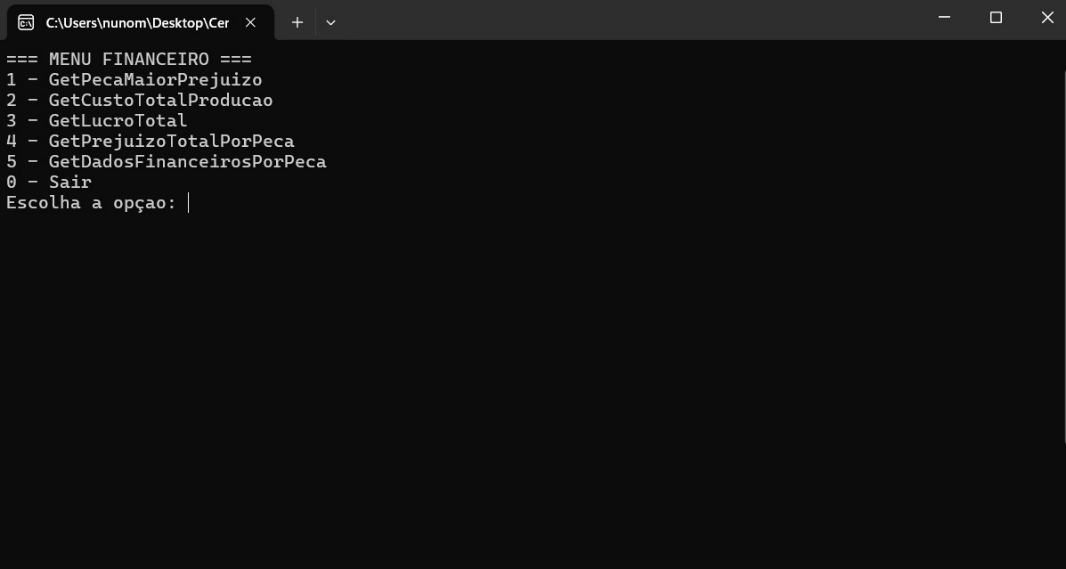
De forma a garantir a fiabilidade, robustez e correto funcionamento das integrações desenvolvidas no âmbito deste projeto, foram realizados diversos testes funcionais e operacionais sobre os diferentes componentes implementados. Esta fase teve como principal objetivo validar a conformidade da solução face aos requisitos previamente definidos, assegurando a correta interoperabilidade entre os sistemas e a integridade dos dados produzidos, processados e armazenados. Os testes realizados subdividiram-se em três categorias distintas, correspondendo à API SOAP, à aplicação produtora (Publisher) e à aplicação consumidora (Consumer).

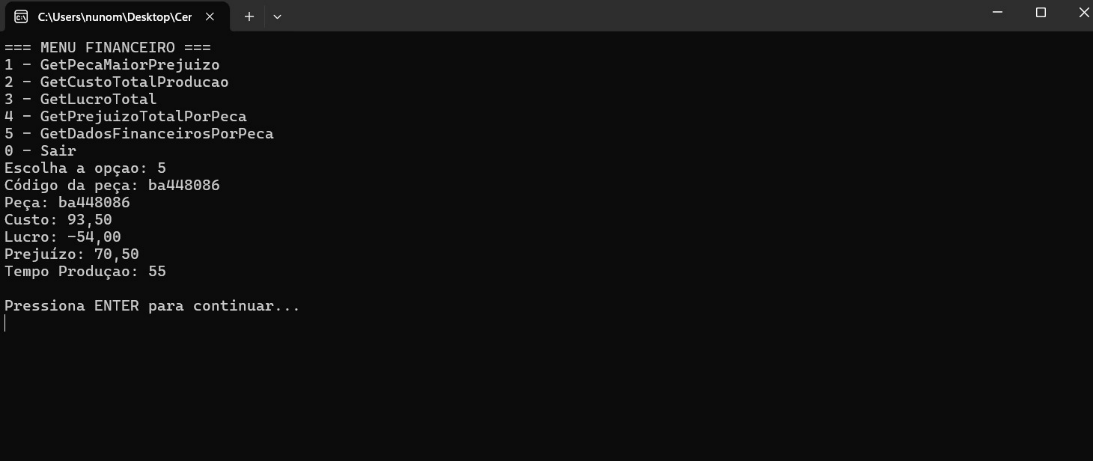
## Testes à API SOAP e Aplicação Cliente

Após a implementação da API SOAP baseada em WCF, procedeu-se à execução de testes funcionais sobre os métodos disponibilizados pelo serviço. A validação foi realizada através de dois mecanismos distintos: invocação direta dos métodos utilizando o WCF Test Client e consumo dos serviços pela aplicação cliente WPF desenvolvida para o efeito.

No primeiro caso, efetuou-se a invocação de cada um dos métodos expostos pela API, verificando-se o correto retorno dos resultados e a conformidade dos dados com os valores registados na base de dados SQL Server. No segundo caso, através da aplicação WPF, foi possível testar a seleção de intervalos de datas, consulta de relatórios financeiros e operações específicas, avaliando-se a consistência dos dados apresentados e a estabilidade da comunicação com o serviço.



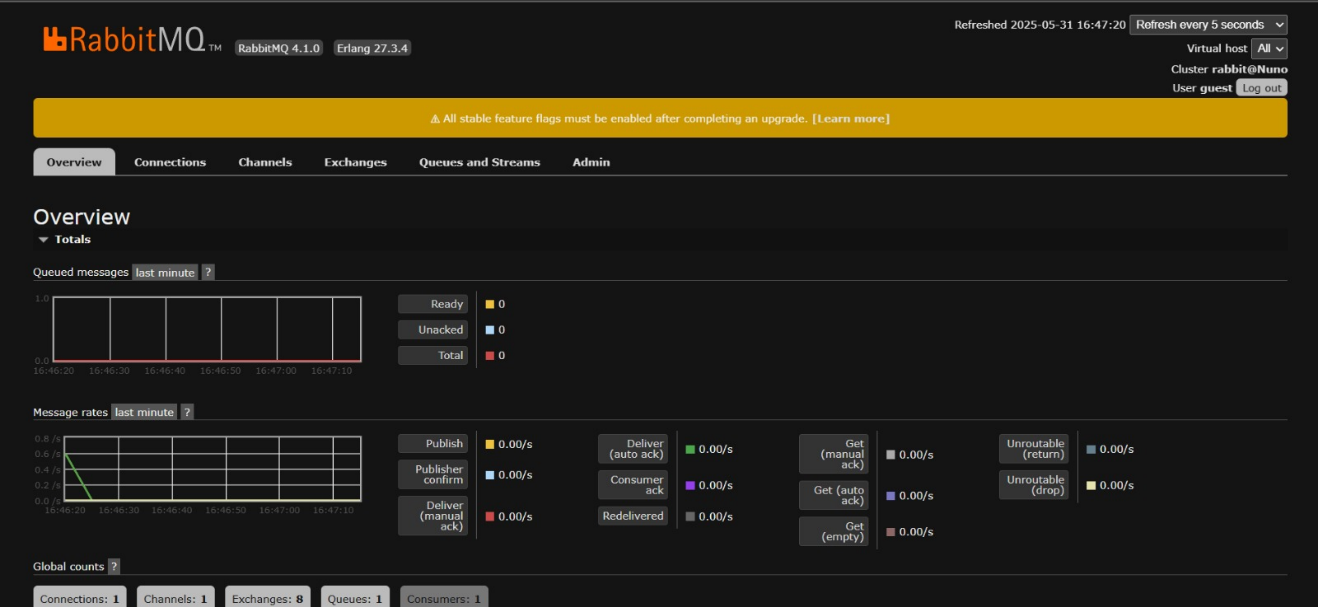


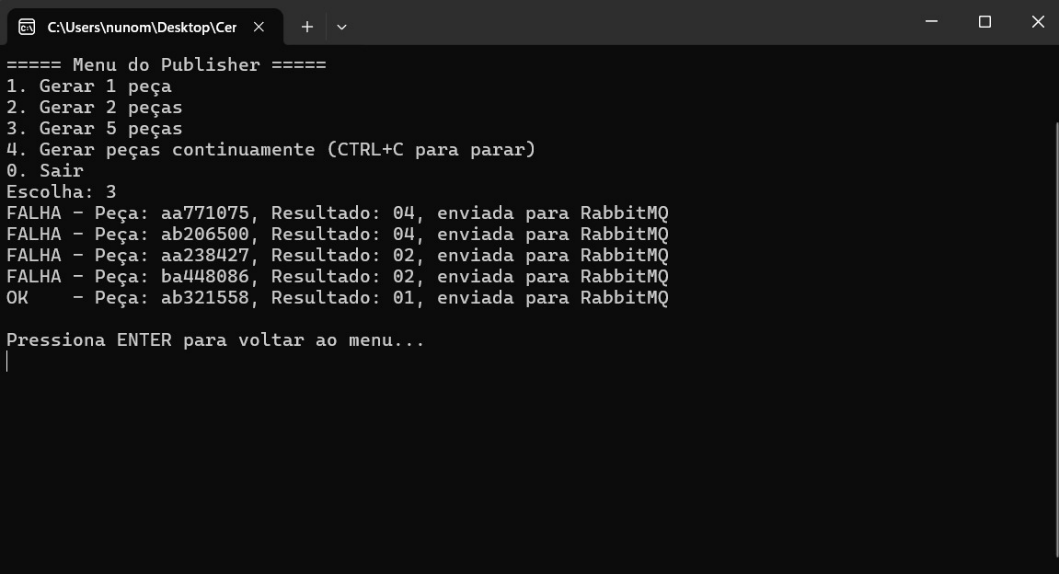


## Testes à Aplicação Produtora (Publisher)

A aplicação produtora, desenvolvida sob a forma de Console Application, foi sujeita a testes de execução em todos os modos de funcionamento implementados: geração de uma peça, de cinco peças consecutivas e modo autónomo. Para cada um destes modos, procedeu-se à verificação da correta publicação das mensagens no RabbitMQ, utilizando para o efeito a interface de gestão web disponibilizada pelo broker (RabbitMQ Management UI).

Confirmou-se, em todos os casos, que as mensagens eram publicadas corretamente no exchange configurado, disseminadas para a respetiva fila e posteriormente disponibilizadas aos consumidores subscritos. Adicionalmente, foi verificado o conteúdo das mensagens no formato JSON, assegurando a presença dos atributos definidos (identificador da peça, estado e tempo de produção) e a conformidade estrutural dos dados.

Foram igualmente recolhidas evidências gráficas, nomeadamente capturas de ecrã da consola da aplicação produtora, do RabbitMQ Management UI e dos dados a circular no exchange, confirmando o correto funcionamento do sistema de publicação de mensagens em todos os cenários testados.



## Testes à Aplicação Consumidora (Consumer)

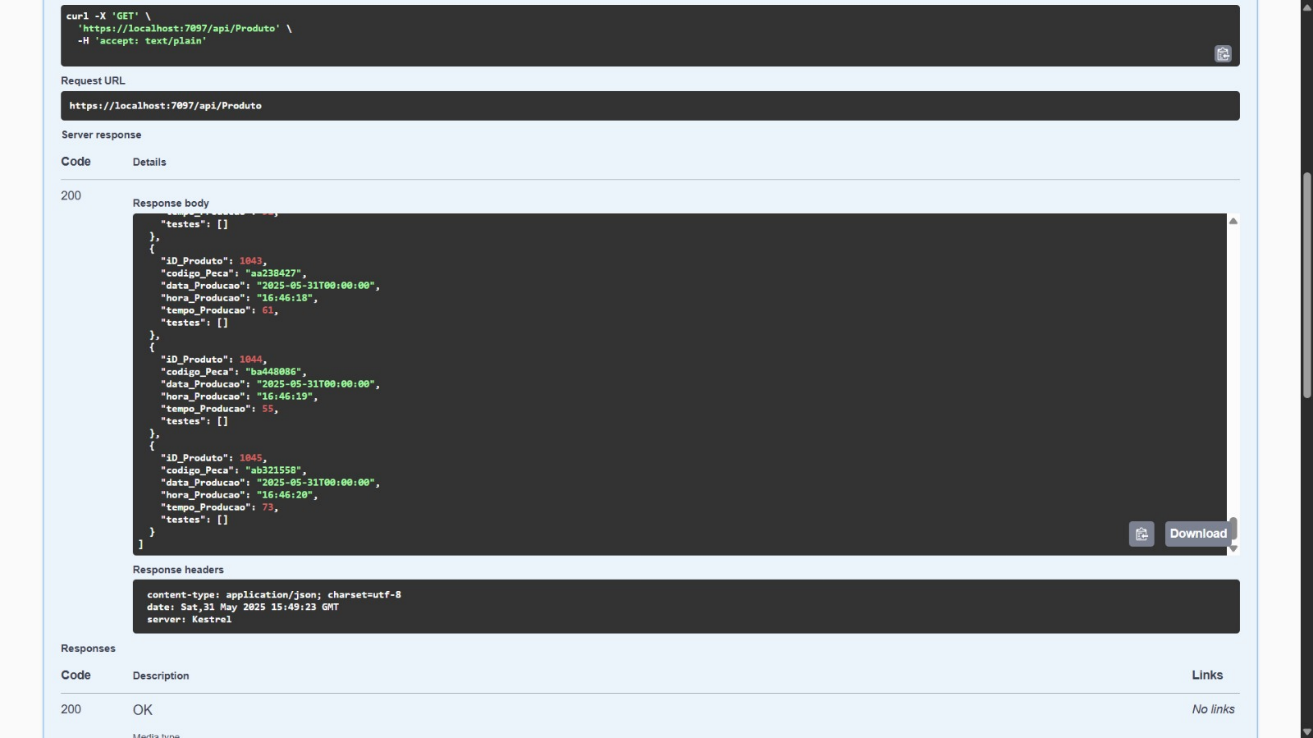
A aplicação consumidora, desenvolvida em WPF, foi sujeita a um conjunto de testes orientados à validação da comunicação com o RabbitMQ Stream e ao processamento dos dados recebidos. Os testes iniciaram-se com a inicialização da aplicação e verificação da ligação bem-sucedida ao stream previamente configurado, assegurando a disponibilidade para consumo de mensagens em tempo real.

Seguidamente, foram publicadas mensagens de produção através da aplicação produtora, confirmando-se que os dados eram corretamente processados e apresentados na interface gráfica da aplicação consumidora. Foram verificadas as notificações automáticas da chegada de cada peça, a atualização dos totais de peças produzidas, a contagem de peças com e sem falha e o cálculo do tempo médio de produção.

Adicionalmente, testou-se a funcionalidade de envio de cada mensagem recebida para a API REST, validando-se a persistência dos dados em base de dados e o correto registo dos mesmos para posterior análise. Esta operação foi confirmada através da consulta direta à base de dados e da verificação dos dados armazenados.

Por fim, procedeu-se à análise do comportamento da aplicação em situações de interrupção da ligação ao broker, simulando a indisponibilidade temporária do RabbitMQ. Após a reposição do serviço, confirmou-se que as mensagens em retenção foram processadas corretamente, sem perda de dados, assegurando a fiabilidade e resiliência da solução implementada.

Estas validações permitiram confirmar a conformidade funcional de todos os componentes desenvolvidos, garantindo a robustez, estabilidade e integridade da solução integrada.



# Conclusão

O desenvolvimento deste projeto permitiu consolidar conhecimentos na área da integração de sistemas, com especial foco na construção de serviços SOAP robustos e compatíveis com ambientes empresariais. Através da implementação de uma API SOAP utilizando WCF e da criação de uma aplicação cliente em WPF, foi possível demonstrar a interoperabilidade entre componentes, o encapsulamento da lógica de negócio financeira e a separação clara entre camadas de serviço e apresentação.

Durante o projeto, foram aplicadas boas práticas de desenvolvimento orientado a serviços, com destaque na utilização de contratos formais, comunicação baseada em mensagens e integração com base de dados através de stored procedures. Os testes realizados permitiram validar o correto funcionamento das funcionalidades expostas, bem como a fiabilidade do sistema em cenários de utilização normal e em situações de erro controlado.

A solução desenvolvida apresenta-se como uma base sólida e extensível, à qual poderão ser futuramente integradas novas funcionalidades, como uma camada REST, mecanismos de mensageria assíncrona ou persistência adicional em sistemas NoSQL. Em resumo, este trabalho revelou-se uma oportunidade prática para aplicar conceitos fundamentais de interoperabilidade, modularidade e robustez em contextos reais de integração empresarial.