INSTITUTO POLITÉCNICO
DO CÁVADO E DO AVE
ESCOLA SUPERIOR
DE TECNOLOGIA

INTEGRAÇÃO DE SISTEMAS DE INFORMAÇÃO (ISI)

Trabalho Prático II

Docente: Luís Ferreira

Aluno: António Ferreira Nº9657

Data: 28/12/2025

Conteúdo

Lista de Abreviaturas

- **API** — *Application Programming Interface* (Interface de Programação de Aplicações)
- **REST** — *Representational State Transfer*
- **SOAP** — *Simple Object Access Protocol*
- **WSDL** — *Web Services Description Language*
- **HTTP** — *Hypertext Transfer Protocol*
- **JSON** — *JavaScript Object Notation*
- **XML** — *eXtensible Markup Language*
- **JWT** — *JSON Web Token*
- **CRUD** — *Create, Read, Update, Delete*
- **SQL** — *Structured Query Language*
- **SSMS** — *SQL Server Management Studio*
- **ADO.NET** — *ActiveX Data Objects .NET*
- **IoT** — *Internet of Things* (Internet das Coisas)
- **UI** — *User Interface* (Interface do Utilizador)
- **WinForms** — *Windows Forms*
- **PaaS** — *Platform as a Service*
- **CI/CD** — *Continuous Integration / Continuous Delivery* (ou Deployment)
- **RBAC** — *Role-Based Access Control* (Controlo de Acesso por Perfis/Funções)
- **ETL** — *Extract, Transform, Load* (se não usaste, podes remover)
- **URL** — *Uniform Resource Locator*

1. Introdução

1.1 Contexto e motivação

Os sistemas de informação atuais exigem frequentemente **interoperabilidade** entre serviços e aplicações distintas (web, mobile, desktop), sustentada por **serviços web** bem definidos. Em ambientes inteligentes (*smart environments*), este requisito torna-se ainda mais relevante, pois é comum integrar informação de gestão com **fontes externas** (ex.: meteorologia) e com eventos provenientes de **sensores/IoT** (reais ou simulados).

O projeto SmartCondo foi desenvolvido como um caso prático para consolidar competências de integração: desenho de API, segurança, persistência, documentação e demonstração por aplicação cliente.

1.2 Problema a resolver

A gestão de condomínios envolve a necessidade de centralizar informação (condomínios, sensores, alertas), disponibilizar operações de manutenção e consulta, e apresentar indicadores agregados para suporte à decisão (dashboards). Além disso, é útil enriquecer o contexto operacional com informação externa — por exemplo, meteorologia — que pode influenciar situações e alertas no condomínio.

O problema central do projeto consiste em criar uma solução **modular, segura e reutilizável**, que disponibilize estas capacidades através de uma **API de interoperabilidade**, consumível por aplicações terceiras.

1.3 Objetivos

Os objetivos do trabalho foram:

- Desenvolver uma **API REST** para expor funcionalidades do domínio e permitir consumo por clientes externos;
- Garantir persistência em **SQL Server**, usando **ADO.NET** e queries parametrizadas;
- Implementar **autenticação** e **autorização** com **JWT** e perfis (*roles*), protegendo operações sensíveis;

- Integrar um serviço externo de **meteorologia** e registar histórico de consultas (WeatherLogs);
- Implementar **logging estruturado** (JSON) para auditoria e suporte ao diagnóstico;
- Construir um **cliente WinForms** para demonstrar, de forma end-to-end, o consumo da API.

1.4 Organização do relatório

O relatório apresenta: requisitos e alinhamento com o enunciado (Capítulo 2), arquitetura e desenho (Capítulo 3), implementação (Capítulo 4), testes e validação (Capítulo 5), publicação e execução (Capítulo 6) e conclusões/trabalho futuro (Capítulo 7). No final incluem-se anexos com evidências (prints) dos testes e do cliente.

2. Requisitos e alinhamento com o enunciado

2.1 Requisitos funcionais

RF1 — Autenticação (login) e emissão de token

O sistema deve autenticar utilizadores por credenciais e emitir um token JWT para acesso autenticado à API.

RF2 — Autorização por perfis (roles)

O acesso a endpoints deve ser condicionado por perfis (Admin), garantindo o princípio do menor privilégio.

RF3 — Gestão de sensores (CRUD)

Criar, listar, editar e remover sensores associados a condomínios, incluindo estado (ativo/inativo).

RF4 — Gestão de condomínios (CRUD/gestão base)

Listar e gerir condomínios (inserir/editar/remover conforme permissões).

RF5 — Dashboards e agregações

Disponibilizar indicadores agregados por cidade/condomínio (totais, alertas por tipo, alertas últimas 24h).

RF6 — Integração meteorológica (serviço externo)

Consultar meteorologia por cidade e devolver temperatura, humidade, descrição e data/hora.

RF7 — WeatherLogs (histórico de consultas)

Registar automaticamente cada consulta de meteorologia para histórico e auditoria.

RF8 — Aplicação cliente demonstradora (WinForms)

Disponibilizar um cliente desktop que demonstre autenticação, CRUD, dashboards e meteorologia.

2.2 Requisitos não funcionais

RNF1 — Segurança

Tokens JWT válidos, proteção de endpoints e regras de acesso por role.

RNF2 — Consistência REST e códigos HTTP

Respostas coerentes (200/400/401/403/404/500) e mensagens previsíveis para consumo por terceiros.

RNF3 — Observabilidade

Logging estruturado (JSON) de requests e erros para auditoria e diagnóstico.

RNF4 — Manutenibilidade

Separação em camadas (API/Dados/Domínio/Cliente) para reduzir acoplamento e facilitar evolução.

RNF5 — Robustez no acesso a dados

Queries parametrizadas e configuração externa evitando credenciais hardcoded.

2.3 Matriz Requisito

Requisito	Evidência
Autenticação JWT	Swagger login com token (Figura 4) + Postman login (Figura 7)
Proteção de endpoints	Endpoint sem token devolve 401 (Figura 6)
Integração serviço externo (Weather)	Swagger Weather 200 + JSON (Figura 8) + WinForms Weather (Figura 15)
Persistência / histórico (WeatherLogs)	SSMS SELECT * FROM WeatherLogs (Figura 9)
Logging estruturado	Ficheiro requests-YYYYMMDD.log (Figura 10)
Cliente demonstrador	WinForms: Login/CRUD/Dashboard/Weather (Figuras 11–115)

Tabela 1 - Evidências

3. Arquitetura e desenho da solução

3.1 Visão geral

A solução foi desenhada para separar responsabilidades, garantindo clareza e facilidade de manutenção. O fluxo típico é: o cliente WinForms autentica, obtém token, e consome endpoints da API; a API valida permissões, executa operações na base de dados via camada de dados, e integra meteorologia via serviço externo quando necessário.

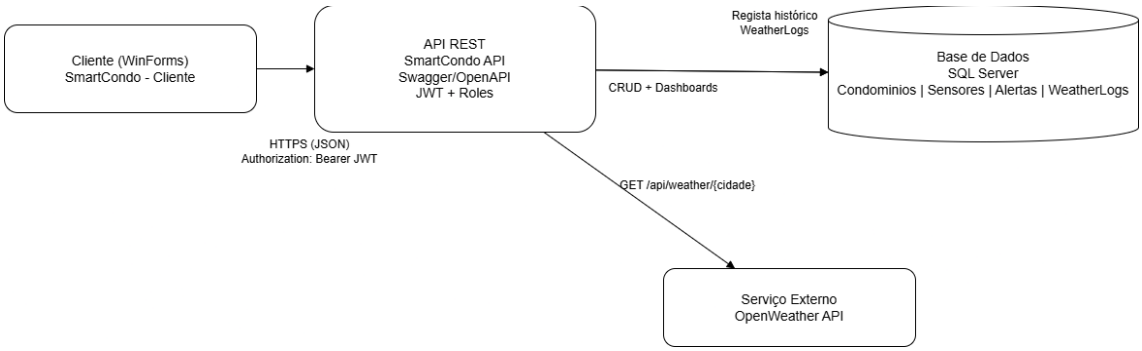


Figura 1 - Diagrama de contexto

3.2 Componentes

- **Cliente WinForms (SmartCondo – Cliente)**: interface para autenticação, gestão e dashboards.
- **API REST (SmartCondo API)**: expõe endpoints, valida permissões e devolve JSON.
- **Camada de Dados (ADO.NET)**: acesso ao SQL Server com queries parametrizadas.
- **Base de Dados (SQL Server)**: persistência das entidades e histórico meteorológico.
- **Serviço Externo (Meteorologia)**: consulta por cidade e registo histórico em WeatherLogs.
- **Logging**: registo estruturado de requests em ficheiros JSON.

3.3 Decisões de desenho relevantes

- **Separação por responsabilidades**: o cliente não acede diretamente à BD; consome sempre a API.
- **Segurança**: endpoints protegidos por JWT e regras de role.
- **Observabilidade**: logs de pedidos para auditoria e apoio ao debugging.
- **Evolução**: a divisão em camadas facilita futura separação por microserviços (ex.: Autenticação separado, Serviço de Dashboards separado).

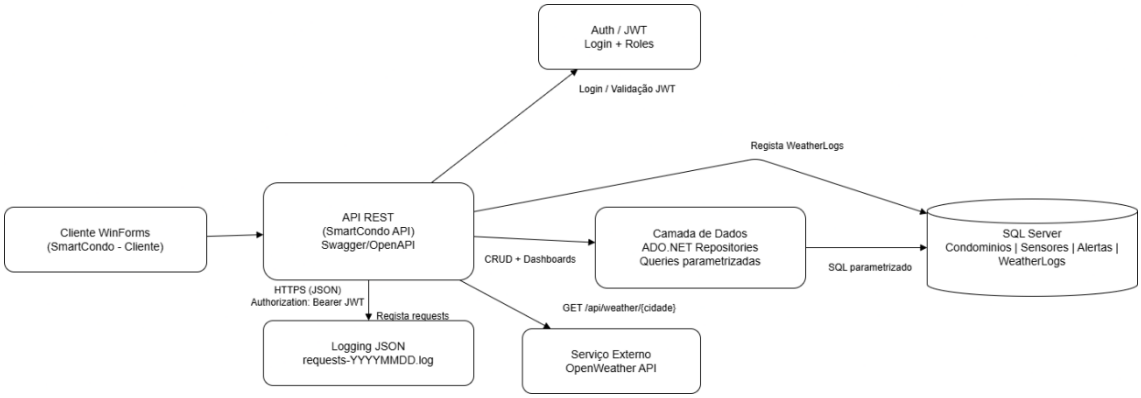


Figura 2 - WinForms - API - Data - SQL – OpenWeather

3.4 Identificação de fronteiras funcionais

Embora a solução tenha sido implementada como um conjunto integrado de projetos (camadas), foi feita a identificação de fronteiras funcionais que permitem evoluir para uma arquitetura baseada em **microservices**. Esta análise é importante para escalabilidade, manutenção e desenvolvimento independente de módulos.

3.4.1 Proposta de divisão em serviços

A solução pode ser separada nos seguintes serviços independentes:

1. **Autenticacao Service**

- **Responsabilidade:** autenticação (login), emissão/validação de tokens, gestão de utilizadores e roles.
- **Endpoints típicos:** POST /api/auth/login, GET /api/auth/me

- **Justificação:** segurança e identidade são transversais e beneficiam de isolamento.

2. **Condomínios Service**

- **Responsabilidade:** CRUD e regras do domínio para condomínios (localização, estado ativo/inativo).
- **Endpoints típicos:** GET/POST/PUT/DELETE /api/condominios
- **Justificação:** módulo administrativo central, pode evoluir independente de sensores/alertas.

3. **Sensores & Alertas Service**

- **Responsabilidade:** gestão de sensores e registo/consulta de alertas.
- **Endpoints típicos:** GET/POST/PUT/DELETE /api/sensores, GET /api/alertas
- **Justificação:** componente "IoT/eventos" é naturalmente separável e escalável.

4. **Weather Service (Integração Externa)**

- **Responsabilidade:** integração com OpenWeather (ou equivalente) e normalização dos dados; registo do histórico.
- **Endpoints típicos:** GET /api/weather/{cidade}, GET /api/weatherlogs
- **Justificação:** integrações externas têm dependências próprias (chaves, rate limits, cache) e merecem isolamento.

5. **Dashboard/Analytics Service**

- **Responsabilidade:** agregações, métricas e dashboards (por cidade/condomínio).
- **Endpoints típicos:** GET /api/dashboard/{cidade}, GET /api/dashboard/condominio/{id}
- **Justificação:** dashboards são compute-heavy e podem crescer muito; separar evita impacto nos serviços CRUD.

3.4.2 Comunicação entre serviços

Numa evolução para microservices, a comunicação pode ser:

- **Síncrona (REST)** para operações simples (ex.: Auth validar token; Dashboard pedir dados agregados);
- **Assíncrona (eventos/mensageria)** como melhoria futura (ex.: quando surge um alerta novo, publicar evento para o Dashboard atualizar métricas, sem acoplamento).

3.4.3 Dados e ownership

Para evitar acoplamento, cada microserviço deve ser "dono" dos seus dados (ou, no mínimo, das suas tabelas). Exemplo de ownership:

- Autenticacao → Users/Roles
- Condomínios → Condominios
- Sensores/Alertas → Sensores/Alertas
- Weather → WeatherLogs (+ cache)
- Dashboard → tabelas agregadas (opcional) ou queries de leitura sobre vistas

3.4.4 Benefícios esperados

- **Evolução independente** (ex.: alterar Weather sem mexer nos CRUD);
- **Escalabilidade** por serviço (dashboard e alertas podem escalar mais);
- **Manutenção e testes** mais simples por módulo;

4. Implementação

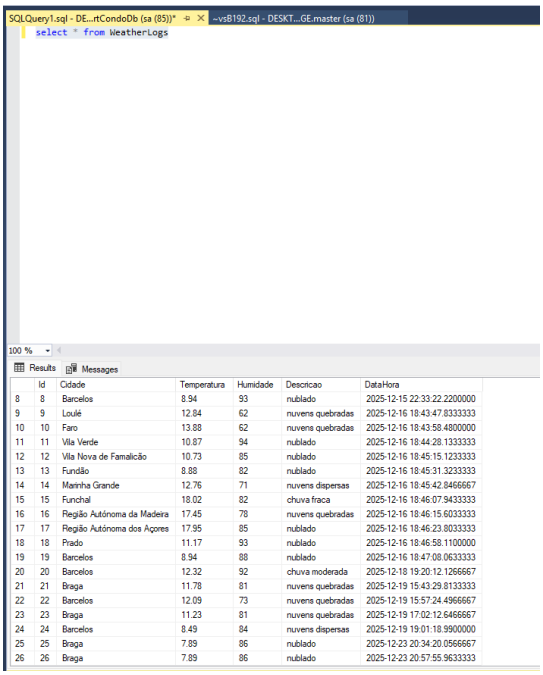
4.1 Tecnologias e ferramentas

- **C# / .NET** (API e WinForms)
- **Swagger/OpenAPI** para documentação e testes dos endpoints
- **SQL Server** para persistência
- **ADO.NET** para acesso a dados (queries parametrizadas)
- **JWT** para autenticação e **roles** para autorização
- **Postman** para validação de endpoints
- **Logging JSON** para auditoria (requests)

4.2 Modelo de dados e persistência

A persistência é assegurada em SQL Server. Para a componente meteorológica existe uma tabela de histórico **WeatherLogs**, que regista cidade, temperatura, humidade, descrição e data/hora da consulta.

A Figura 3 demonstra que as consultas meteorológicas são efetivamente persistidas, permitindo auditoria e utilização futura em análises e dashboards.



	Id	Cidade	Temperatura	Humidade	Descricao	DataHora
8	8	Barcelos	8.94	93	nublado	2025-12-15 22:33:22.2200000
9	9	Loulé	12.84	62	nuvens quebradas	2025-12-16 18:43:47.8333333
10	10	Faro	13.88	62	nuvens quebradas	2025-12-16 18:43:58.4800000
11	11	Vila Verde	10.87	94	nublado	2025-12-16 18:44:28.1333333
12	12	Vila Nova de Famalicão	10.73	85	nublado	2025-12-16 18:45:15.1233333
13	13	Fundão	8.88	82	nublado	2025-12-16 18:45:31.3233333
14	14	Marinha Grande	12.76	71	nuvens dispersas	2025-12-16 18:45:42.8466667
15	15	Funchal	18.02	82	chuva fraca	2025-12-16 18:46:07.9433333
16	16	Região Autónoma da Madeira	17.45	78	nuvens quebradas	2025-12-16 18:46:15.6033333
17	17	Região Autónoma dos Açores	17.95	85	nublado	2025-12-16 18:46:23.8033333
18	18	Prado	11.17	93	nublado	2025-12-16 18:46:58.1100000
19	19	Barcelos	8.94	88	nublado	2025-12-16 18:47:08.0633333
20	20	Barcelos	12.32	92	chuva moderada	2025-12-18 19:20:12.1266667
21	21	Braga	11.78	81	nuvens quebradas	2025-12-19 15:43:29.8133333
22	22	Barcelos	12.09	73	nuvens quebradas	2025-12-19 15:57:24.4966667
23	23	Braga	11.23	81	nuvens quebradas	2025-12-19 17:02:12.6466667
24	24	Barcelos	8.49	84	nuvens dispersas	2025-12-19 19:01:18.9900000
25	25	Braga	7.89	86	nublado	2025-12-23 20:34:20.0566667
26	26	Braga	7.89	86	nublado	2025-12-23 20:57:55.9633333

Figura 3 - SQL Server Inserção

4.3 API REST e documentação Swagger

A API disponibiliza endpoints REST para autenticação, operações de gestão e consultas agregadas. A documentação e teste são realizados via Swagger.

Endpoints principais (exemplos)

- POST /api/auth/login — autenticação e emissão de JWT
- GET /api/sensores — listagem de sensores (protegido)
- GET /api/weather/{cidade} — meteorologia por cidade
- GET /api/dashboard/{cidade} — dashboard por cidade (indicadores + distribuição por tipo)
- GET /api/condominios — listagem de condomínios

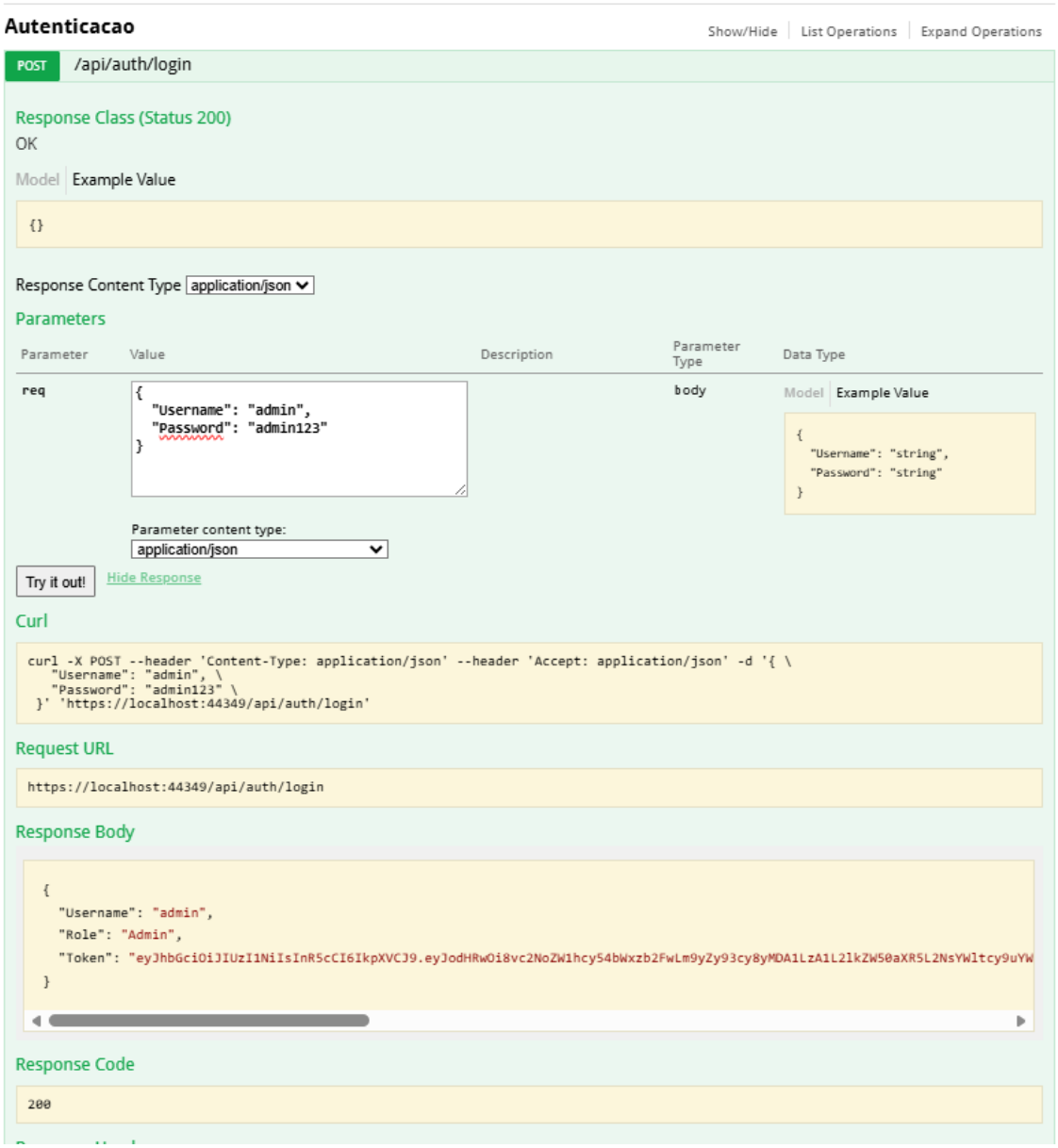


Figura 4 - Swagger: login com credenciais e retorno do token JWT

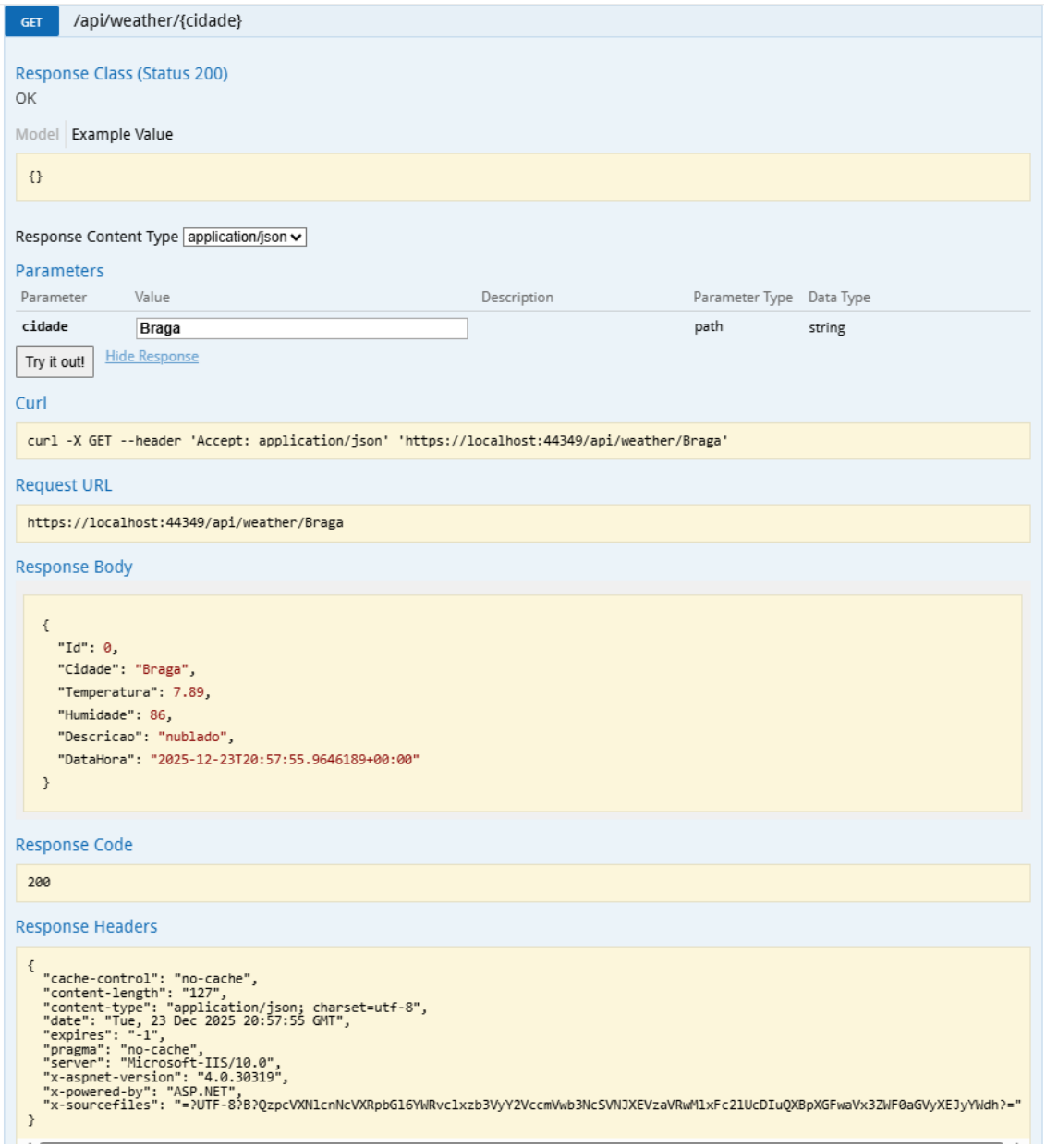


Figura 5 - Swagger: Weather com response 200 e JSON.

4.4 Segurança: JWT e autorização por roles

A segurança é suportada por um fluxo simples e robusto:

1. O utilizador executa login (/api/auth/login) e recebe um **token JWT**;
2. O token é enviado nas chamadas seguintes via header Authorization: Bearer <token>;
3. A API valida token e aplica regras por role.

A evidência da proteção de endpoints é demonstrada na Figura 7: ao chamar /api/sensores sem credenciais, a API responde com **401 (Unauthorized)**, confirmando controlo de acesso.

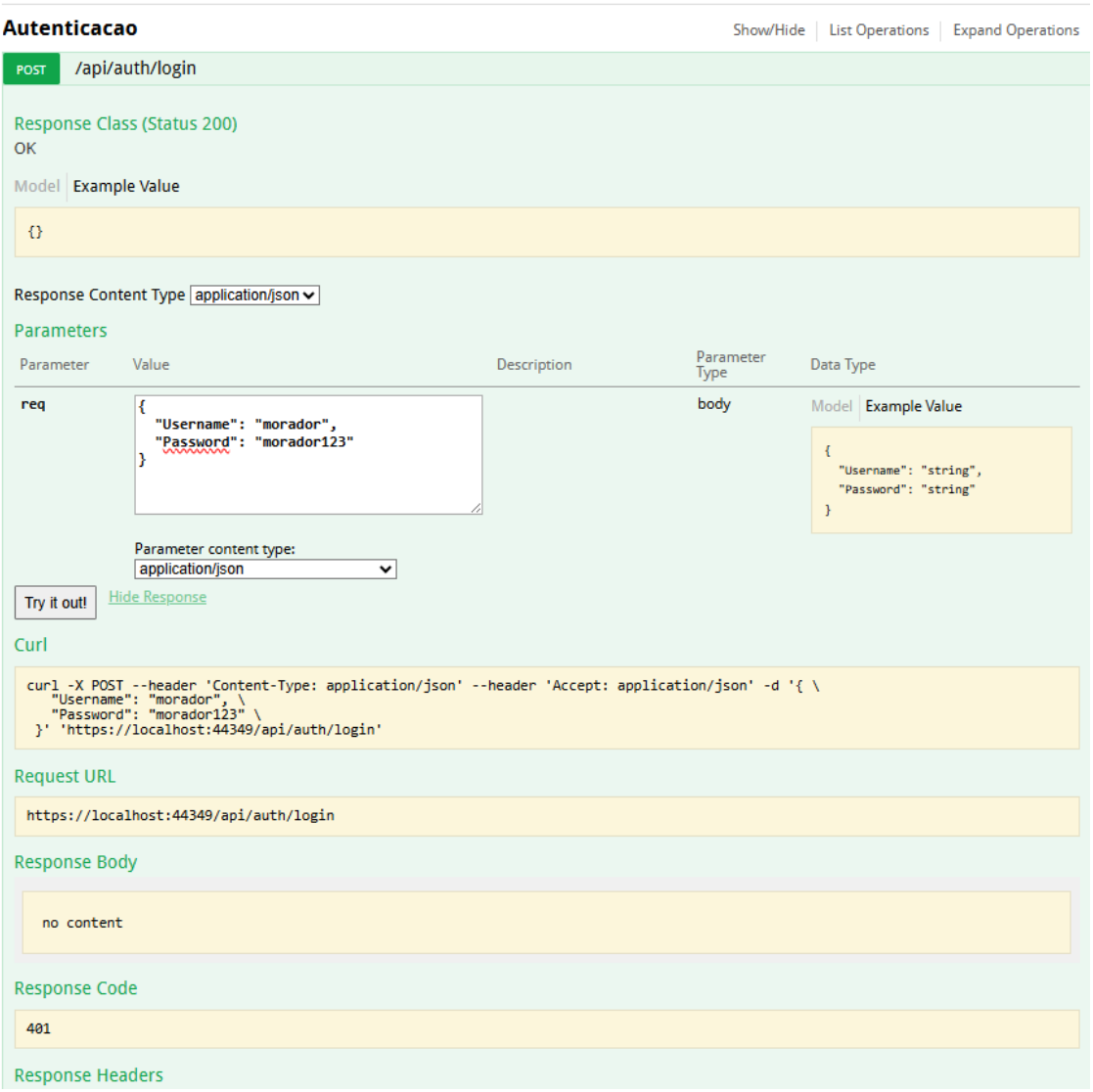


Figura 6 - Swagger: endpoint protegido sem token devolve 401

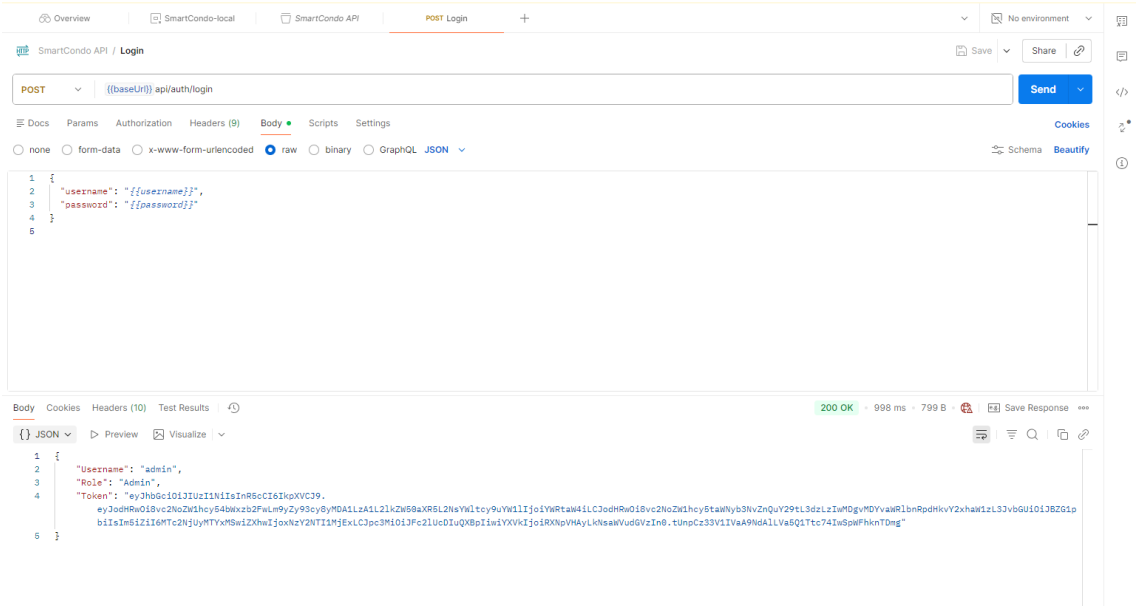


Figura 7 - Postman: login e retorno do token

4.5 Integração externa: meteorologia e registo em WeatherLogs

A integração meteorológica é realizada por cidade e devolve informação atual (temperatura, humidade, descrição e timestamp). Sempre que é feita uma consulta, é criado um registo em **WeatherLogs**. Isto permite:

- comprovar a integração externa;
- manter histórico de consultas;
- suportar dashboards e auditoria.

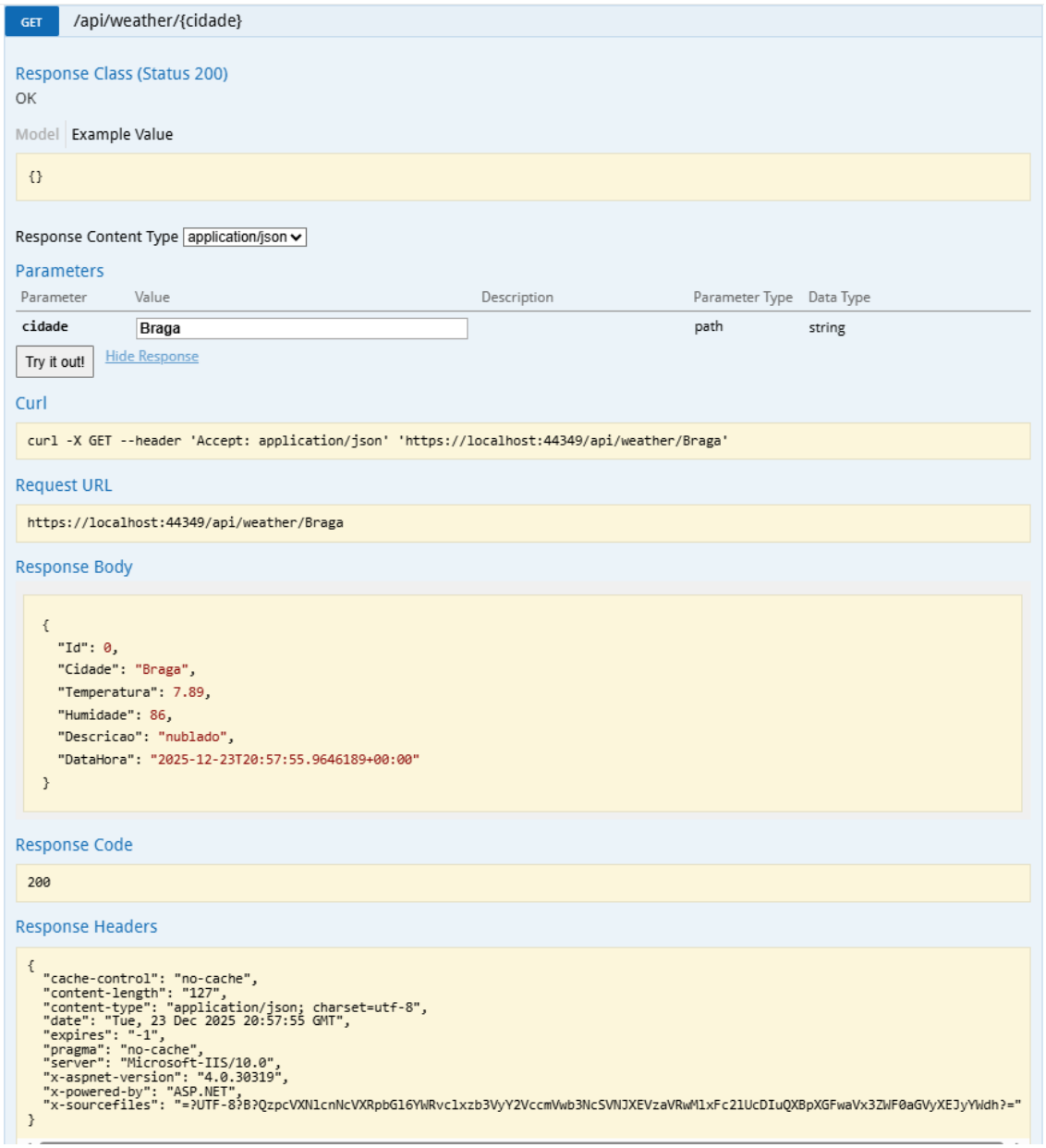


Figura 8 -Swagger: Weather com response 200 (Braga)

SQLQuery1.sql - DE...rtCondoDb (sa (85))* ~vsB192.sql - DESKT...GE.master (sa (81))

```
select * from WeatherLogs
```

100 %

Results Messages

	Id	Cidade	Temperatura	Humidade	Descricao	DataHora
8	8	Barcelos	8.94	93	nublado	2025-12-15 22:33:22.2200000
9	9	Loulé	12.84	62	nuvens quebradas	2025-12-16 18:43:47.8333333
10	10	Faro	13.88	62	nuvens quebradas	2025-12-16 18:43:58.4800000
11	11	Vila Verde	10.87	94	nublado	2025-12-16 18:44:28.1333333
12	12	Vila Nova de Famalicão	10.73	85	nublado	2025-12-16 18:45:15.1233333
13	13	Fundão	8.88	82	nublado	2025-12-16 18:45:31.3233333
14	14	Marinha Grande	12.76	71	nuvens dispersas	2025-12-16 18:45:42.8466667
15	15	Funchal	18.02	82	chuva fraca	2025-12-16 18:46:07.9433333
16	16	Região Autónoma da Madeira	17.45	78	nuvens quebradas	2025-12-16 18:46:15.6033333
17	17	Região Autónoma dos Açores	17.95	85	nublado	2025-12-16 18:46:23.8033333
18	18	Prado	11.17	93	nublado	2025-12-16 18:46:58.1100000
19	19	Barcelos	8.94	88	nublado	2025-12-16 18:47:08.0633333
20	20	Barcelos	12.32	92	chuva moderada	2025-12-18 19:20:12.1266667
21	21	Braga	11.78	81	nuvens quebradas	2025-12-19 15:43:29.8133333
22	22	Barcelos	12.09	73	nuvens quebradas	2025-12-19 15:57:24.4966667
23	23	Braga	11.23	81	nuvens quebradas	2025-12-19 17:02:12.6466667
24	24	Barcelos	8.49	84	nuvens dispersas	2025-12-19 19:01:18.9900000
25	25	Braga	7.89	86	nublado	2025-12-23 20:34:20.0566667
26	26	Braga	7.89	86	nublado	2025-12-23 20:57:55.9633333

Figura 9 - SQL Server: WeatherLogs preenchido com várias cidades

4.6 Logging e observabilidade

Para garantir auditabilidade e facilitar diagnóstico, o sistema mantém logs estruturados em JSON com informação dos pedidos efetuados (método, URL, headers relevantes e timestamps). A Figura 10 mostra exemplos de entradas no ficheiro requests-20251223.log, incluindo chamadas de login, sensores, condomínios, dashboard e weather.

Figura 10 - Log JSON: registo de requests em ficheiro

4.7 Aplicação cliente WinForms (prova de consumo)

Foi desenvolvido um cliente **Windows Forms** para demonstrar consumo real da API e validar o sistema end-to-end.

Funcionalidades comprovadas no cliente

- **Login** (obtenção de token e identificação de role)
- **Gestão de sensores** (CRUD: novo/guardar/eliminar + listagem)
- **Gestão de condomínios** (CRUD/listagem)
- **Dashboard** com indicadores agregados e gráfico “Alertas por tipo”
- **Meteorologia** por cidade com apresentação de temperatura/humidade/descrição e data/hora

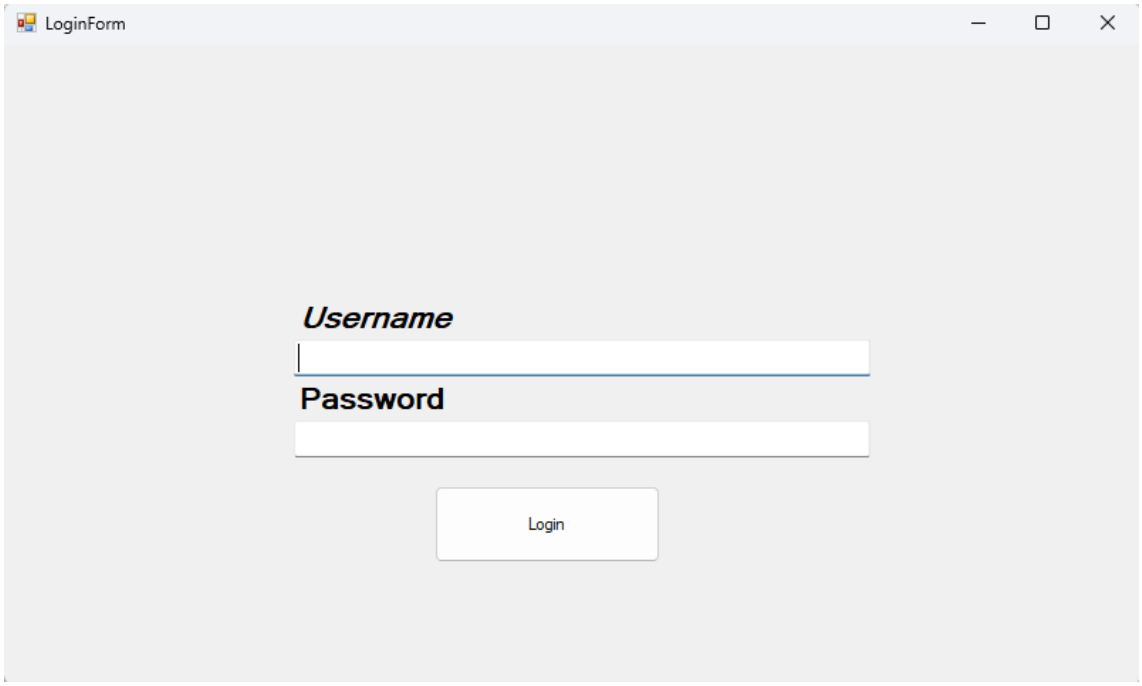


Figura 11 - WinForms - Login

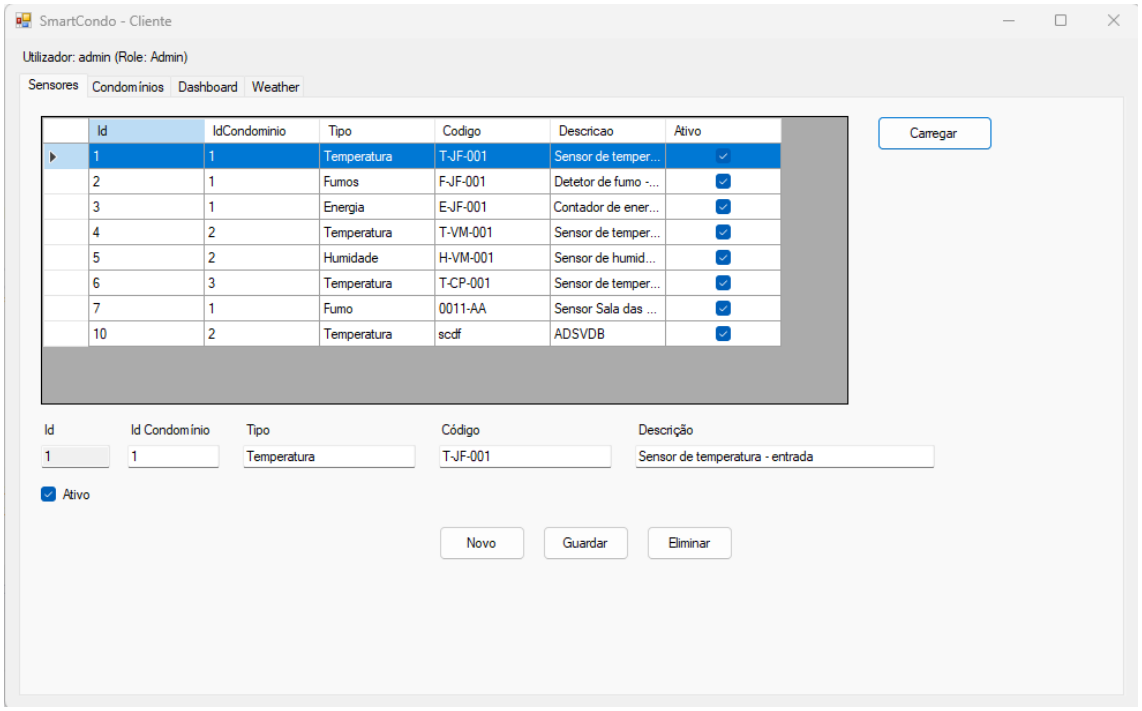


Figura 12 - WinForms Sensores CRUD

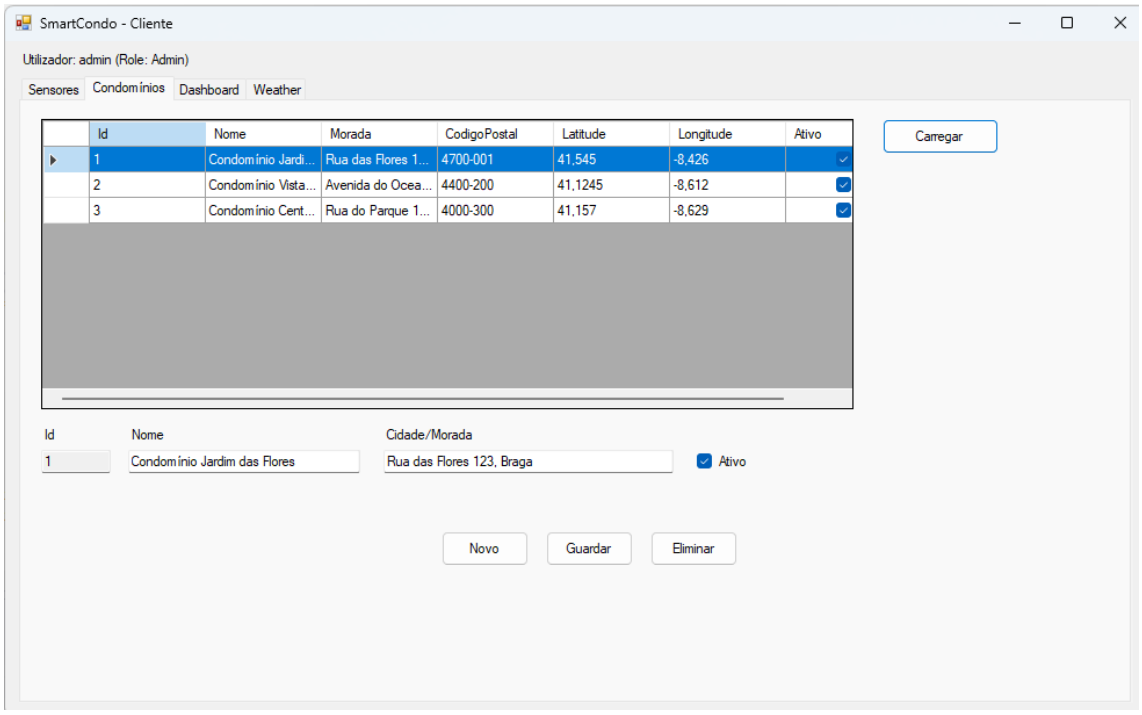


Figura 13 - WinForms Condomínios CRUD

Figura 14 - WinForms Dashboard

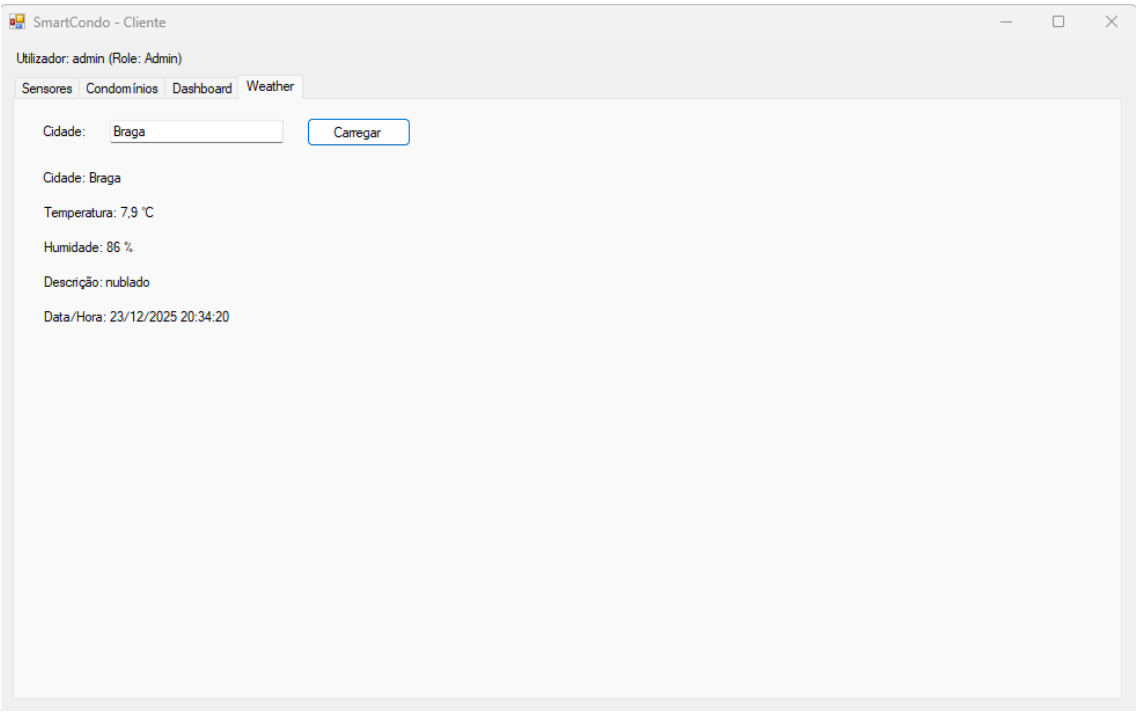


Figura 15 - WinForms Weather

4.8 Serviço SOAP

Para além dos serviços RESTful, foi disponibilizado um **serviço SOAP** com o objetivo de reforçar a componente de interoperabilidade e compatibilidade com integrações "legadas", seguindo uma abordagem típica SOA. Este serviço SOAP atua como um **Data Layer Service**, expondo operações específicas do domínio (neste caso, operações relacionadas com a componente meteorológica e/ou acesso controlado a dados) e permitindo que outros consumidores (não-REST) possam integrar a solução através de contratos WSDL.

4.8.1 Motivação e papel na arquitetura

A adoção de SOAP neste projeto tem três objetivos principais:

- **Interoperabilidade por contrato**: disponibilizar um contrato formal (WSDL) com operações e tipos definidos, facilitando integração com sistemas que exigem SOAP.
- **Separação de responsabilidades**: concentrar operações específicas no serviço SOAP, mantendo a API REST como camada de exposição moderna e orientada a clientes.

- **Reutilização/Extensão**: permitir que outras aplicações consumam o serviço SOAP diretamente ou através da API REST, sem duplicação de lógica.

A Figura 2 ilustra o fluxo geral, onde a API REST pode atuar como “facade”, consumindo internamente o serviço SOAP quando aplicável.

4.8.2 Contrato WSDL e operações expostas

O serviço SOAP disponibiliza um contrato WSDL acessível através do endpoint:

http://localhost:64810/Service1.svc?wsdl

O WSDL descreve formalmente o serviço, os tipos de dados, as mensagens, as operações e o respetivo binding/port, permitindo que qualquer cliente SOAP gere automaticamente proxies de consumo com base no contrato publicado. A **Figura 16** apresenta o WSDL completo do serviço SOAP.

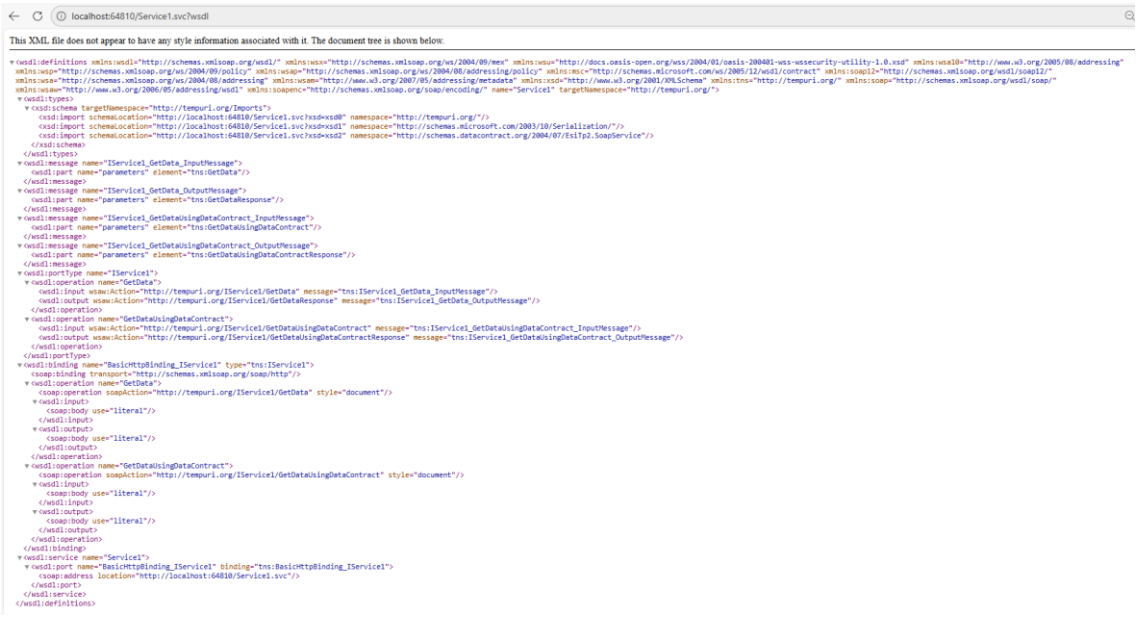


Figura 16 - Serviço SOAP WSDL

Operações expostas (wsdl:operation)

De acordo com a secção <wsdl:portType name="IService1">, o serviço expõe as seguintes operações:

- **GetData** — operação de teste/validação do serviço (útil para confirmar disponibilidade e funcionamento do endpoint);

- **GetDataUsingDataContract** — operação baseada num DataContract, permitindo testar troca de objetos estruturados (serialização).

A **Figura 17** apresenta um detalhe da zona do WSDL onde se encontram as operações (<wsdl:operation>) e a respetiva configuração de binding.

Figura 17 - Serviço SOAP detalhe das operações

Binding e Port (binding/port)

O serviço utiliza um binding do tipo **BasicHttpBinding**, indicado no WSDL como:

<wsdl:binding name="BasicHttpBinding_IService1" type="tns:IService1">

Este binding define a utilização de SOAP sobre HTTP e especifica o soapAction associado a cada operação (ex.: http://tempuri.org/IService1/GetData). Adicionalmente, o WSDL define o serviço e o respetivo endpoint através do elemento <soap:address>, apontando para:

http://localhost:64810/Service1.svc

Esta configuração confirma que o serviço SOAP está corretamente publicado e acessível, podendo ser consumido por ferramentas como SoapUI, Postman (SOAP) ou WCF Test Client, servindo como evidência objetiva do cumprimento do requisito de suporte a SOAP.

4.8.3 Integração REST - SOAP

Para manter a experiência moderna de consumo, a solução disponibiliza **API REST** (consumida pelo cliente WinForms) e, em paralelo, um **serviço SOAP** baseado em contrato (WSDL), que pode ser consumido por clientes SOAP (ex.: Postman/SoapUI/WCF Test Client).
Nesta fase, o serviço SOAP é usado como **evidência de interoperabilidade** por contrato e compatibilidade com integrações legadas, coexistindo com a camada REST.

O fluxo típico de utilização é:

1. O cliente (WinForms/Swagger/Postman) consome a **API REST** para as funcionalidades principais (ex.: GET /api/weather/{cidade}, CRUD e dashboards);
2. Em paralelo, consumidores SOAP podem invocar diretamente o **endpoint SOAP** (Service1.svc) utilizando as operações definidas no WSDL (ex.: **GetData** e **GetDataUsingDataContract**);
3. O serviço SOAP processa o pedido e devolve a resposta em **XML (SOAP Envelope)**, conforme o contrato publicado.

A **Figura 18** demonstra a execução de uma chamada SOAP realizada no Postman, evidenciando o envio do envelope SOAP para a operação **GetData** e a respetiva resposta **GetDataResponse/GetDataResult** com **200 OK**, confirmando o correto funcionamento do serviço SOAP.

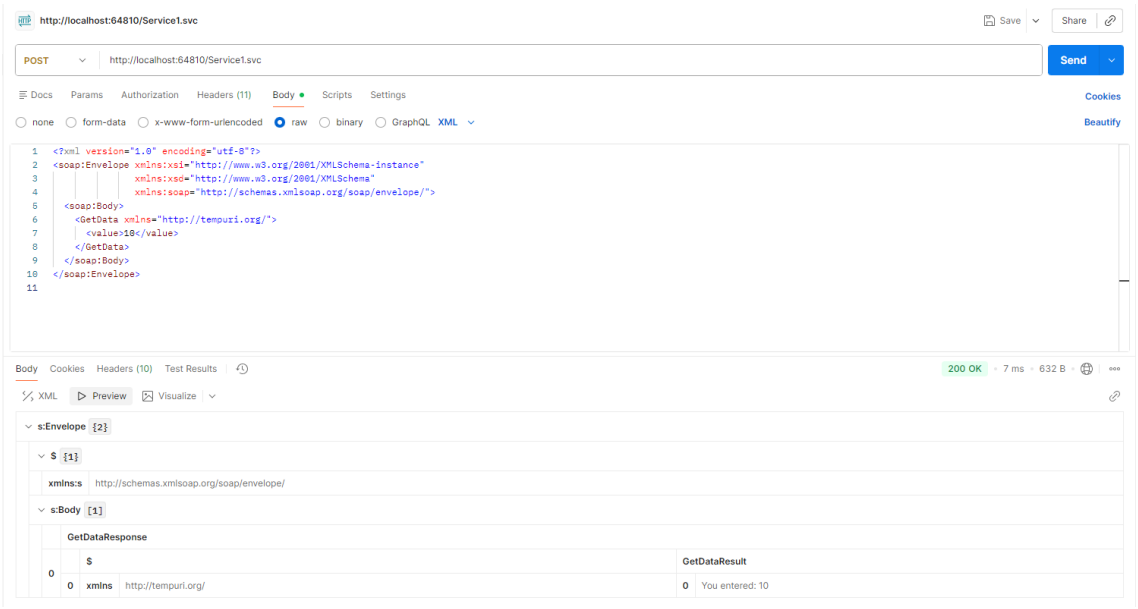


Figura 18 -Execução do serviço SOAP Postman - pedido POST ao endpoint Service1.svc

4.8.4 Benefícios e limitações

Benefícios:

- Contrato formal (WSDL) útil para integrações empresariais;
- Separação da lógica em serviços;
- Demonstra competências SOA além de REST.

Limitações/considerações:

- SOAP é mais verboso (XML) e tende a ser menos prático para clientes modernos;
- Requer configuração adequada (bindings, endpoints, segurança) em ambiente de publicação.

5. Testes e validação

5.1 Estratégia de validação

A validação foi realizada combinando:

- **Swagger** para testar contratos, inputs/outputs e códigos HTTP;
- **Postman** para reforçar testes (ex.: login e uso de token);
- **Cliente WinForms** para validar o fluxo completo end-to-end;
- **SQL Server** para confirmar persistência (ex.: WeatherLogs);
- **Logs JSON** para confirmar rastreio e auditoria dos pedidos.

5.2 Casos de teste executados

- **CT1**: Login válido devolve token (Swagger/Postman) → Figuras 4 e 7
- **CT2**: Endpoint protegido sem token devolve 401 → Figura 6
- **CT3**: Weather devolve dados e regista em WeatherLogs → Figuras 5 e 9
- **CT4**: CRUD Sensores e Condomínios via cliente → Figuras 12 e 13
- **CT5**: Dashboard por cidade mostra indicadores e gráfico → Figura 14
- **CT6**: Logs registam todas as chamadas efetuadas → Figura 10
- **CT7** (novo): WSDL publicado e operações identificáveis → Figuras 16 e 17

6. Publicação e execução

6.1 Execução local

A solução pode ser executada localmente nos seguintes passos:

1. Iniciar a base de dados SQL Server e garantir que a connection string está correta;
2. Executar a API (ex.: https://localhost:44349/);
3. Testar via Swagger (/swagger) e/ou Postman;
4. Abrir o Cliente WinForms e efetuar login;
5. Validar funcionalidades (sensores, condomínios, dashboard, weather).

6.2 Publicação em cloud

Como evolução natural e alinhamento com o enunciado, a solução está preparada para ser publicada numa plataforma cloud (PaaS), por exemplo:

- API em **Azure App Service** (ou equivalente);
- BD em **Azure SQL Database**;
- Configuração via variáveis de ambiente (connection string, chave JWT, chave do serviço externo).
  Após publicação, a validação pode ser repetida com Swagger/Postman contra o URL público.

7. Conclusão

O SmartCondo cumpre os objetivos definidos para uma solução orientada à interoperabilidade, disponibilizando uma **API REST** documentada em **Swagger/OpenAPI**, com **segurança baseada em JWT e roles**, persistência em **SQL Server** através de **ADO.NET**, e integração com um **serviço externo de meteorologia**. A implementação do histórico em **WeatherLogs** comprova a persistência e permite auditoria e utilização futura em análises e dashboards. A aplicação cliente **WinForms** demonstra de forma prática o consumo da API e valida o funcionamento end-to-end (autenticação, operações CRUD, dashboards e meteorologia). Adicionalmente, o **logging estruturado** em ficheiro acrescenta observabilidade e suporte ao diagnóstico. Como complemento aos serviços REST, foi também disponibilizado um **serviço SOAP com contrato WSDL**, validando interoperabilidade baseada em contrato e compatibilidade com integrações legadas.

Durante o desenvolvimento surgiram algumas dificuldades relevantes. A principal esteve relacionada com a **integração e consistência entre camadas** (cliente - API - dados - SQL), exigindo validação cuidadosa das queries, dos modelos e da forma como os dados eram devolvidos ao cliente. Também houve desafios na **segurança**, sobretudo na correta aplicação do token JWT nos pedidos e na validação de permissões, garantindo que endpoints protegidos devolviam os códigos HTTP adequados (ex.: 401). A integração com o serviço externo de meteorologia implicou ainda atenção à **gestão de erros e respostas**, bem como ao registo consistente das consultas no histórico (WeatherLogs). Por fim, a componente SOAP exigiu validação do contrato (WSDL) e da execução de pedidos/respostas em formato XML, para garantir conformidade e interoperabilidade.

Como continuação natural do projeto, pretende-se evoluir a solução com:

I. **publicação em cloud (PaaS)**, incluindo API e base de dados com configuração por variáveis de ambiente e validação contra um endpoint público;

II. reforço de **testes automatizados** e pipeline de integração/entrega (CI/CD);
III. expansão dos **dashboards** com métricas temporais e indicadores por condomínio;
IV. evolução do cliente para **web/mobile**, mantendo a mesma API;
V. melhoria do serviço SOAP para expor operações diretamente úteis ao domínio (por exemplo, operações relacionadas com meteorologia/alertas), mantendo um contrato estável para consumidores externos. Em síntese, o SmartCondo apresenta uma base sólida, modular e extensível, alinhada com o objetivo de disponibilizar serviços reutilizáveis num contexto de smart environment.