

**Instituto Politécnico do Cávado e do Ave
Escola Superior de Tecnologia**

**Licenciatura em
Engenharia de Sistemas Informáticos**

SafeHome – Descrição

Fábio Rafael Gomes Costa – a22997

Lino Emanuel Oliveira Azevedo – a23015

Barcelos, dezembro de 2025

**Licenciatura em
Engenharia de Sistemas Informáticos**

SafeHome – Descrição

Fábio Rafael Gomes Costa (nº 22997)

Lino Emanuel Oliveira Azevedo (nº a23015)

Unidades Curriculares:

Integração de Sistemas de Informação

Docentes:

Óscar Ribeiro

Barcelos, dezembro de 2025

Ficha de Identificação

Elaborado por Fábio Rafael Gomes Costa – a22997

Lino Emanuel Oliveira Azevedo – a23015

Contato a22997@alunos.ipca.pt

a23015@alunos.ipca.pt

Unidade Curricular Integração de Sistemas de Informação (ISI)

Curso Licenciatura em Engenharia de Sistemas Informáticos

Instituição Escola Superior de Tecnologia do Instituto Politécnico do Cávado e do Ave

Professor Orientador Óscar Ribeiro

Contato oribeiro@ipca.pt

Data de início 4 de dezembro de 2025

Data de conclusão 28 de dezembro de 2025

Índice

Ficha de Identificação.....	I
Índice	II
Índice de Figuras	III
Índice de Tabelas	III
1. Introdução.....	1
1.1. Contextualização	1
1.2. Motivação/Pretensões e Objetivos	1
1.3. Estrutura do Documento	3
2. Tema.....	4
3. Descrição do problema.....	5
4. Documentação do trabalho desenvolvido	6
4.1. Descrição do problema	6
4.1.1. Enquadramento	6
4.1.2. Requisitos principais (funcionais)	6
4.1.3. Requisitos não funcionais (segurança, desempenho, etc.).....	8
4.2. Arquitetura prevista para a solução	9
4.2.1. Visão geral.....	9
4.2.2. Componentes principais e responsabilidades	10
4.2.3. Fluxo geral (front → API → BD → serviços externos).....	11
4.3. Modelo de dados.....	13
4.3.1. Modelo ER (Entidade–Relação).....	13
4.3.2. Entidades principais.....	14
4.3.3. Relações.....	14
4.3.4. Chaves primárias e estrangeiras	14
4.4. Identificação das principais rotas (endpoints)	15
4.5. Arquitetura da solução (implementação).....	19

4.5.1.	Operações implementadas com SOAP (XML / WCF / SoapCore).....	19
4.5.2.	Operações implementadas com RESTful	20
4.5.3.	Principais serviços isolados e comunicação	21
4.5.4.	Sistema de gestão de base de dados utilizado.....	22
4.5.5.	Serviços externos.....	22
4.5.6.	Testes e Validação	24
4.5.7.	Deploy e Publicação na Cloud (Azure)	26
5.	Conclusão	27
6.	Referências bibliográficas	29

Índice de Figuras

Figura 1 - Arquitetura	10
Figura 2 - Diagrama Entidade Relação.....	13

Índice de Tabelas

Tabela 1 - Auth (/api/Auth)	15
Tabela 2 - Buildings (/api/Buildings)	15
Tabela 3 - Sensors (/api/Sensors)	16
Tabela 4 - SensorReadings (/api/SensorReadings).....	16
Tabela 5 - Alerts (/api/Alerts).....	16
Tabela 6 - Incidents (/api/Incidents).....	17
Tabela 7 - Users (/api/Users) — Admin only.....	17
Tabela 8 - Reports (/api/reports).....	18
Tabela 9 - Data portability (/api/data-portability)	18
Tabela 10 - Social integrations (/api/social)	18
Tabela 11 - Testes e Validação.....	25
Tabela 12 - Principais Endpoints.....	26

1. Introdução

1.1. Contextualização

No âmbito da Unidade Curricular (UC) de Integração de Sistemas de Informação (ISI), inserida no 1.º semestre do 3.º ano da Licenciatura em Engenharia de Sistemas Informáticos, foi proposta a realização de um trabalho prático como instrumento de avaliação, sob a orientação do docente Dr. Óscar Ribeiro. Este trabalho tem como foco a exploração e o desenvolvimento de processos de interoperabilidade entre sistemas, suportados por serviços web, promovendo a criação de novos serviços SOAP e RESTful, bem como a reutilização de serviços externos existentes.

Neste contexto, foi desenvolvido o projeto SafeHome, uma plataforma IoT orientada à monitorização e gestão de segurança de habitações/edifícios, agregando informação proveniente de sensores e eventos operacionais, como leituras, alertas e incidentes. A solução disponibiliza uma API REST para consumo por clientes modernos e um serviço SOAP dedicado à gestão de incidentes, reforçando a interoperabilidade em cenários empresariais e legados. Complementarmente, a plataforma integra serviços externos (por exemplo, meteorologia) para enriquecer a informação disponibilizada ao utilizador.

1.2. Motivação/Pretensões e Objetivos

A evolução dos smart environments e a crescente utilização de sensores IoT em contextos residenciais e empresariais tornam essencial a existência de plataformas capazes de centralizar dados, automatizar processos e permitir interoperabilidade com aplicações terceiras. O enunciado do trabalho reforça esta necessidade ao propor a modelação de um problema associado à gestão de ambientes inteligentes, exigindo a disponibilização de uma API de serviços que suporte integrações e consumo por diferentes tipos de clientes.

Assim, o SafeHome pretende responder ao desafio de criar uma solução unificada para monitorização e segurança, com serviços e mecanismos que permitam: recolher e gerir informação operacional, garantir controlo de acesso e disponibilizar interfaces de integração adequadas (REST e SOAP). Como motivação adicional, o tema “SafeHome”

foi escolhido por se enquadrar num cenário realista de segurança em edifícios residenciais situados em zonas de risco (ex.: incêndios ou cheias).

Temos como objetivo desenvolver e disponibilizar, na cloud, uma plataforma de serviços web para gestão e consulta de informação de monitorização e segurança em edifícios, garantindo interoperabilidade, segurança e documentação adequada.

Objetivos específicos

- Implementar uma API RESTful com operações CRUD sobre as entidades do sistema, em conformidade com os requisitos do trabalho.
- Implementar um serviço SOAP para a vertente de incidentes, exposto no endpoint /Service.asmx.
- Proteger o acesso aos serviços com autenticação e autorização baseada em JWT, suportando perfis/roles e validações de segurança.
- Documentar a API através do standard OpenAPI/Swagger, incluindo suporte a autenticação Bearer .
- Integrar serviços externos para enriquecimento e interoperabilidade, nomeadamente:
 - Meteorologia (OpenWeather), consumida via HTTP e parametrizada por latitude/longitude ;
 - Partilha de incidentes para um endpoint externo configurável (no projeto, usando postman-echo.com/post como endpoint de teste).
- Disponibilizar a solução e os seus componentes na cloud, garantindo acessibilidade e facilitando testes/consumo por clientes.
- Desenvolver pelo menos uma aplicação cliente para demonstrar a utilização da API, recorrendo a um front-end web e a um “Swagger UI externo” para consumo do OpenAPI publicado.

1.3. Estrutura do Documento

Este relatório foi organizado de forma a apresentar o trabalho desenvolvido de maneira clara e sequencial. Inicia-se com a capa, subcapa e Ficha de Identificação, seguida do Índice, que inclui tanto os tópicos principais como o Índice de Figuras, e a Lista de Siglas e Acrónimos.

A Introdução apresenta o enquadramento geral do projeto, incluindo a contextualização, os objetivos e a metodologia.

Na secção seguinte tem o Tema e a Descrição do problema, onde é apresentado o domínio do projeto, a solução proposta (SafeHome), detalha o problema a resolver e as necessidades identificadas no contexto de monitorização e segurança em edifício.

A Documentação do trabalho descreve a arquitetura, o modelo de dados, os principais endpoints/serviços (REST e SOAP), mecanismos de segurança, integrações externas e aspetos de implementação e deploy.

O relatório é concluído com as Considerações Finais, onde são apresentadas as reflexões sobre o trabalho realizado, as aprendizagens obtidas e sugestões de melhoria.

Por fim, são apresentadas as Referências Bibliográficas, onde se encontram listados todos os links consultados durante o desenvolvimento do projeto.

2. Tema

Resumo: SafeHome — Plataforma IoT para monitorização e segurança de habitações/edifícios (smart home / smart building).

O tema do projeto consiste no desenvolvimento de uma plataforma de monitorização e gestão de segurança para edifícios, denominada SafeHome, com foco na integração centralizada de dados provenientes de sensores e eventos operacionais. A solução agrupa informação relacionada com leituras de sensores, alertas e incidentes, permitindo a consulta e gestão consistente do estado do edifício e do histórico associado.

A plataforma disponibiliza uma API REST com autenticação baseada em JWT, garantindo controlo de acesso às operações. Em paralelo, é disponibilizado um serviço SOAP dedicado à gestão de incidentes, permitindo interoperabilidade com clientes e integrações que utilizem este tipo de serviço, o que é relevante em cenários empresariais e legados.

Em termos funcionais, a solução contempla a gestão de edifícios, sensores, leituras, alertas, incidentes e utilizadores, incluindo ainda integrações externas, como:

- consulta de meteorologia (ex.: OpenWeather) para enriquecer o contexto operacional;
- partilha de incidentes para endpoints externos configuráveis, suportando integração com sistemas terceiros.

Em termos de disponibilização, tanto a API como a base de dados encontram-se alojadas na cloud Microsoft Azure, permitindo acesso remoto seguro e simplificando o consumo da plataforma por diferentes clientes. A API encontra-se publicada como serviço web e a base de dados é disponibilizada como serviço gerido, garantindo disponibilidade e escalabilidade adequadas ao contexto do projeto.

3. Descrição do problema

Em contextos residenciais e empresariais, a segurança e a monitorização de edifícios dependem cada vez mais de sistemas distribuídos. Múltiplos sensores recolhem leituras contínuas (ambientais ou de estado) que podem originar alertas e, consequentemente, exigir o registo e acompanhamento rigoroso de incidentes (abertura, atualização e resolução). No entanto, quando não existe uma plataforma unificada, a informação tende a permanecer dispersa por diferentes sistemas e formatos, o que dificulta a operação e a tomada de decisão.

A ausência de uma solução centralizada impacta diretamente:

- Centralização de dados: torna-se complexo consultar, de forma consistente, o estado dos edifícios e sensores, bem como o histórico de leituras e ocorrências.
- Gestão de incidentes: dificulta a resposta rápida a alertas e o controlo do ciclo de vida dos incidentes (criação, acompanhamento e fecho).
- Interoperabilidade: limita a integração com serviços externos (por exemplo, meteorologia para contextualização) e o envio de informação para entidades terceiras.
- Segurança: compromete o controlo de acesso, sendo necessário garantir que apenas utilizadores autenticados e autorizados executam operações sensíveis.

Assim, o problema a resolver consiste em criar uma solução capaz de centralizar, organizar e operar sobre os dados de monitorização e segurança, de forma segura e interoperável. O sistema deve disponibilizar interfaces adequadas tanto para consumidores modernos (REST) como para integrações empresariais (SOAP), permitindo a gestão de edifícios e sensores (ex.: fumo, gás, temperatura) e o registo e tratamento de incidentes.

O objetivo central é garantir uma visão unificada do estado do imóvel, suportando a deteção de riscos, o acompanhamento de ocorrências e a integração com serviços externos que contribuam para uma resposta mais informada e eficiente.

4. Documentação do trabalho desenvolvido

4.1. Descrição do problema

O projeto SafeHome pretende resolver a necessidade de centralizar e gerir informação de monitorização e segurança em edifícios, recolhida por sensores (IoT). A solução agrupa dados de edifícios, sensores, leituras, alertas, incidentes e utilizadores, suportando operações de consulta e gestão sobre estes elementos (CRUD), bem como integrações externas para enriquecer o contexto e apoiar decisões.

Ao nível técnico, a plataforma foi desenvolvida com uma API REST protegida por autenticação e autorização e, em paralelo, expõe um endpoint SOAP para interoperabilidade com clientes empresariais/legados, sendo ambos disponibilizados pela mesma aplicação

4.1.1. Enquadramento

Em ambientes residenciais e empresariais, sensores distribuídos (ex.: temperatura, fumo, gás) produzem leituras contínuas que podem originar alertas e levar à abertura de incidentes. Sem uma solução centralizada, a informação tende a ficar dispersa, dificultando a consulta do estado atual do edifício, a rastreabilidade do histórico e a resposta a ocorrências.

Para responder a este cenário, o SafeHome organiza os dados num modelo relacional com entidades como Buildings, Sensors, SensorReadings, Alerts, Incidents e Users, bem como as relações (ex.: sensores pertencem a um edifício; leituras e alertas pertencem a um sensor; incidentes pertencem a um edifício). Além disso, ao obter detalhes de um edifício, a solução enriquece a informação com meteorologia atual via serviço externo.

4.1.2. Requisitos principais (funcionais)

RF1 — Autenticação e gestão de utilizadores

- Registar utilizadores e validar perfis permitidos (“Admin” e “User”)
- Login com devolução de token e operações protegidas como consulta do utilizador atual (Me) e alteração de palavra-passe (ChangePassword)

RF2 — Gestão de edifícios

- Listar, consultar, criar, atualizar e remover edifícios via REST
- Consultar detalhes de edifício incluindo meteorologia atual obtida externamente

RF3 — Gestão de sensores

- CRUD de sensores (listar, consultar por id, criar, atualizar e remover)
- Associar sensores a edifícios (BuildingId) e garantir integridade referencial (ex.: erro quando edifício não existe)

RF4 — Gestão de leituras de sensores

- CRUD de leituras (listar, consultar, criar, atualizar, remover)

RF5 — Gestão de alertas

- CRUD de alertas (listar, consultar, criar, atualizar, remover)

RF6 — Gestão de incidentes

- CRUD de incidentes via REST (listar, consultar, criar, atualizar, remover)
- Disponibilizar serviço SOAP para gestão/integração na vertente de incidentes

RF7 — Relatórios e exportação

- Fornecer “snapshot” do dashboard (métricas agregadas: totais, abertos/resolvidos, etc.) e exportações em CSV de alertas e incidentes

RF8 — Portabilidade de dados (import/export)

- Exportar leituras para CSV e importar leituras em lote (sensor-readings)

RF9 — Interações externas

- Consulta de meteorologia (OpenWeather) para localização do edifício
- Partilha de incidentes para um endpoint externo configurável (ex.: “api/social ... /share”)

4.1.3. Requisitos não funcionais (segurança, desempenho, etc.)

RNF1 — Segurança

- A API REST utiliza JWT Bearer para autenticação e valida emissor/audiência/assinatura
- Endpoints relevantes estão protegidos com [Authorize] (ex.: Buildings e afins)
- Passwords são armazenadas com hash BCrypt (não em texto simples)
- Força redirecionamento HTTPS no pipeline da aplicação

RNF2 — Interoperabilidade

- Disponibilização simultânea de REST (controllers) e SOAP no mesmo backend

RNF3 — Documentação e testabilidade

- API documentada via Swagger com suporte para autenticação Bearer
- Estrutura de BD e relações são reproduzíveis em testes através de criação de esquema (facilita validação)

RNF4 — Desempenho e eficiência

- Operações do backend são assíncronas (async/await) nos controllers, reduzindo bloqueios em I/O
- Relatórios usam queries agregadas para devolver um snapshot único (evita múltiplas chamadas no front)

RNF5 — Manutenibilidade e extensibilidade

- Separação por camadas (Controllers + Services) e injeção de dependências (DI) para serviços internos e externos
- Interações externas usam HttpClient via DI/HttpClientFactory e configuração por options

4.2. Arquitetura prevista para a solução

A arquitetura do SafeHome foi desenhada para suportar um cenário típico de IoT/smart building, em que um cliente (front-end) consome serviços de backend para gerir entidades (edifícios, sensores, leituras, alertas, incidentes) e integrar informação externa (ex.: meteorologia). A solução combina REST (para clientes modernos) e SOAP (para interoperabilidade), mantendo uma separação clara entre apresentação, serviços, dados e integrações.

4.2.1. Visão geral

Arquitetura em camadas (alto nível):

1. Camada de Apresentação (Front-end)
 - Interface web (páginas HTML/CSS/JS)
 - Responsável por autenticação, navegação e visualização (dashboard, listagens, detalhes, criação/edição)
2. Camada de Serviços (Backend / API)
 - Aplicação .NET publicada no Microsoft Azure (Azure App Service), expondo:
 - API REST (JSON sobre HTTP/HTTPS)
 - Serviço SOAP (XML) focado em incidentes
 - Contém a lógica de negócio, validações e orquestração (ex.: obter edifício + meteorologia)
3. Camada de Dados (Persistência)
 - Base de dados relacional em Azure SQL Database (SQL Server gerido na Azure)
 - Responsável por armazenar entidades e relações (edifícios, sensores, leituras, alertas, incidentes, utilizadores)
4. Camada de Integrações Externas
 - Serviço de meteorologia (ex.: OpenWeather)
 - Endpoints externos configuráveis para partilha de incidentes (integração social/terceiros)

Mini-diagrama:

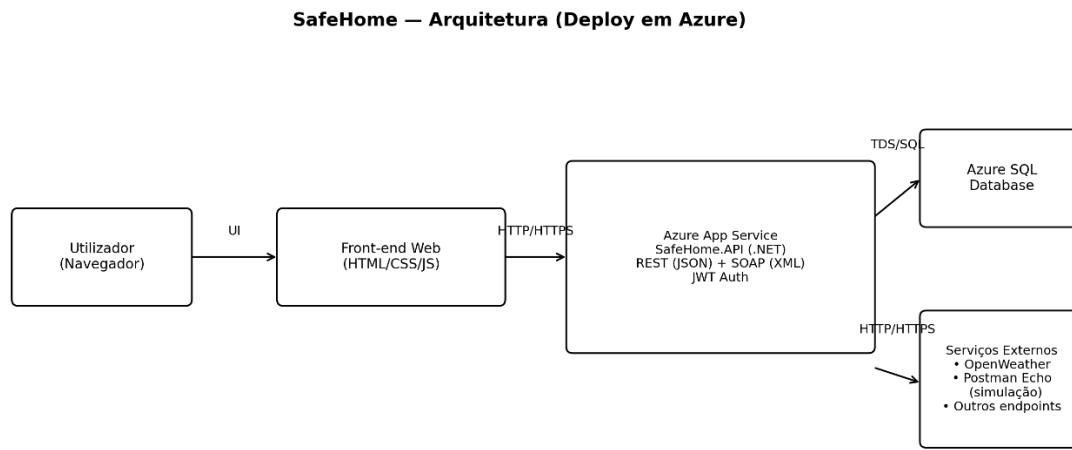


Figura 1 - Arquitetura

4.2.2. Componentes principais e responsabilidades

A) Front-end (Web UI)

Responsabilidades:

- Autenticação (login) e armazenamento do token (JWT)
- Consumo de endpoints REST com Authorization: Bearer <token>
- Gestão do ciclo de uso da aplicação:
 - listar edifícios/sensores/alertas/incidentes
 - apresentar detalhes e formulários
 - ações administrativas (quando aplicável, via role)

B) Backend (.NET)

Responsabilidades:

- Expor endpoints REST para operações CRUD e consultas
- Expor endpoint SOAP para operações de incidentes (interoperabilidade)
- Aplicar regras de segurança:
 - autenticação via JWT
 - autorização por perfil/role

- Orquestrar processos:
 - ex.: “detalhe do edifício” + consulta de meteorologia externa
 - exportações/importações (CSV), relatórios agregados

C) Base de Dados (Azure SQL Database)

Responsabilidades:

- Persistência de dados estruturados e relacionais
 - Persistência de dados estruturados e relacionais num serviço gerido na Azure
 - Integridade referencial entre entidades
 - Suporte a consultas agregadas (dashboard/relatórios)

D) Serviços Externos

- Meteorologia (OpenWeather ou equivalente)
 - Enriquecer o contexto do edifício (ex.: condições meteorológicas atuais)
- Partilha externa / integração social
 - Permitir enviar informação de incidentes para um endpoint externo configurável (ex.: sistemas terceiros)

4.2.3. Fluxo geral (front → API → BD → serviços externos)

Fluxo 1 — Autenticação e acesso ao sistema

- Utilizador abre o front-end e efetua login
- Front-end envia credenciais para a API (REST)
- API valida utilizador, gera JWT
- Front-end guarda o token e passa a incluir Bearer token nos pedidos seguintes

Fluxo 2 — Consulta e gestão de dados (ex.: edifícios/sensores/alertas)

- Front-end pede listagens/detalhes (REST)

- API valida token (autenticação/autorização)
- A API, alojada no Azure App Service, comunica com a base de dados Azure SQL Database através de ligação segura (connection string configurada no ambiente).
- API devolve resposta (JSON) ao front-end para renderização

Fluxo 3 — Detalhes do edifício com enriquecimento externo (meteorologia)

- Front-end pede detalhes de um edifício
- API consulta a BD para obter dados do edifício
- API chama o serviço externo de meteorologia (HTTP/HTTPS)
- API agrupa os resultados (dados do edifício + meteorologia) e devolve ao front-end

Fluxo 4 — Incidentes via SOAP (interoperabilidade)

- Um cliente externo (legado/empresarial) chama o serviço SOAP
- Backend processa a operação (criar/consultar/atualizar incidente)
- Backend acede à BD e devolve resposta em XML/SOAP

Fluxo 5 — Partilha de incidentes para terceiros

- Front-end (ou cliente) solicita “partilhar incidente”
- Backend obtém dados na BD (incidente + edifício)
- Backend envia payload para endpoint externo configurável
- Backend devolve o resultado ao cliente

4.3. Modelo de dados

O SafeHome utiliza um modelo relacional composto por 6 tabelas principais: Buildings, Sensors, SensorReadings, Alerts, Incidents e Users . O modelo foi definido para suportar monitorização (leituras), deteção (alertas) e gestão operacional (incidentes) por edifício.

4.3.1. Modelo ER (Entidade–Relação)

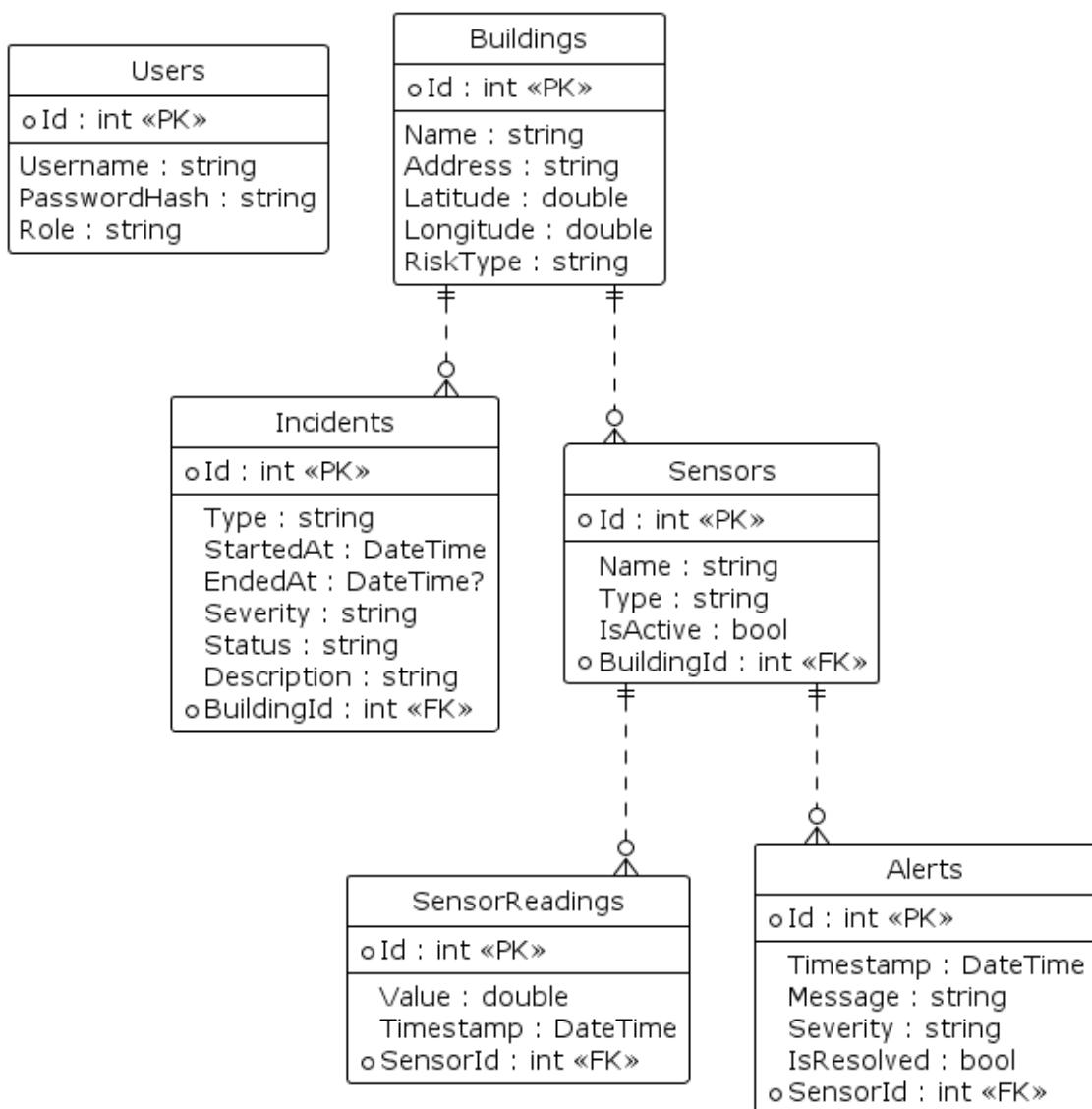


Figura 2 - Diagrama Entidade Relação

4.3.2. Entidades principais

Buildings: representa o edifício/habitação monitorizada (inclui localização e tipo de risco)

Sensors: sensores instalados num edifício (tipo, estado ativo)

SensorReadings: leituras produzidas por um sensor (valor + timestamp)

Alerts: alertas associados a um sensor (severidade, resolvido/não resolvido)

Incidents: incidentes associados a um edifício (tipo, severidade, estado, descrição e datas)

Users: utilizadores do sistema (username, hash da password, role)

4.3.3. Relações

Buildings (1) → Sensors (N): um edifício pode ter vários sensores

Sensors (1) → SensorReadings (N): um sensor pode gerar várias leituras

Sensors (1) → Alerts (N): um sensor pode originar vários alertas

Buildings (1) → Incidents (N): um edifício pode ter vários incidentes

Users: não está ligado por FK às restantes entidades no modelo atual .

4.3.4. Chaves primárias e estrangeiras

PK (Primary Keys)

- Todas as tabelas usam Id como PK com IDENTITY(1,1)

FK (Foreign Keys)

- Sensors.BuildingId → Buildings.Id
- SensorReadings.SensorId → Sensors.Id
- Alerts.SensorId → Sensors.Id
- Incidents.BuildingId → Buildings.Id

4.4. Identificação das principais rotas (endpoints)

Auth (/api/Auth)

Método	URL	Descrição	Autenticação
POST	/api/Auth/Register	Regista um utilizador (roles permitidos: Admin/User)	Não
POST	/api/Auth/Login	Autêntica e devolve token JWT	Não
GET	/api/Auth/Me	Devolve dados do utilizador autenticado	Sim (JWT)
POST	/api/Auth/ChangePassword	Altera password do utilizador autenticado	Sim (JWT)

Tabela 1 - Auth (/api/Auth)
Buildings (/api/Buildings)

Método	URL	Descrição	Autenticação
GET	/api/Buildings	Lista edifícios	Sim (JWT)
GET	/api/Buildings/{id}	Detalhe do edifício + meteorologia atual	Sim (JWT)
POST	/api/Buildings	Cria edifício	Sim (JWT)
PUT	/api/Buildings/{id}	Atualiza edifício	Sim (JWT)
DELETE	/api/Buildings/{id}	Remove edifício	Sim (JWT)

Tabela 2 - Buildings (/api/Buildings)

Sensors (/api/Sensors)

Método	URL	Descrição	Autenticação
GET	/api/Sensors	Lista sensores	Sim (JWT)
GET	/api/Sensors/{id}	Detalhe de sensor	Sim (JWT)
POST	/api/Sensors	Cria sensor	Sim (JWT)
PUT	/api/Sensors/{id}	Atualiza sensor	Sim (JWT)
DELETE	/api/Sensors/{id}	Remove sensor	Sim (JWT)

Tabela 3 - Sensors (/api/Sensors)
SensorReadings (/api/SensorReadings)

Método	URL	Descrição	Autenticação
GET	/api/SensorReadings	Lista leituras	Sim (JWT)
GET	/api/SensorReadings/{id}	Detalhe de leitura	Sim (JWT)
POST	/api/SensorReadings	Cria leitura	Sim (JWT)
PUT	/api/SensorReadings/{id}	Atualiza leitura	Sim (JWT)
DELETE	/api/SensorReadings/{id}	Remove leitura	Sim (JWT)

Tabela 4 - SensorReadings (/api/SensorReadings)
Alerts (/api/Alerts)

Método	URL	Descrição	Autenticação
GET	/api/Alerts	Lista alertas	Sim (JWT)
GET	/api/Alerts/{id}	Detalhe de alerta	Sim (JWT)
POST	/api/Alerts	Cria alerta	Sim (JWT)
PUT	/api/Alerts/{id}	Atualiza alerta	Sim (JWT)
DELETE	/api/Alerts/{id}	Remove alerta	Sim (JWT)

Tabela 5 - Alerts (/api/Alerts)

Incidents (/api/Incidents)

Método	URL	Descrição	Autenticação
GET	/api/Incidents	Lista incidentes	Sim (JWT)
GET	/api/Incidents/{id}	Detalhe de incidente	Sim (JWT)
POST	/api/Incidents	Cria incidente	Sim (JWT)
PUT	/api/Incidents/{id}	Atualiza incidente	Sim (JWT)
DELETE	/api/Incidents/{id}	Remove incidente	Sim (JWT)

Tabela 6 - Incidents (/api/Incidents)
Users (/api/Users) — Admin only

Método	URL	Descrição	Autenticação
GET	/api/Users	Lista utilizadores	Sim (JWT) + Role=Admin
GET	/api/Users/{id}	Detalhe de utilizador	Sim (JWT) + Role=Admin
POST	/api/Users	Cria utilizador	Sim (JWT) + Role=Admin
PUT	/api/Users/{id}	Atualiza role e/ou faz reset de password	Sim (JWT) + Role=Admin
DELETE	/api/Users/{id}	Remove utilizador	Sim (JWT) + Role=Admin

Tabela 7 - Users (/api/Users) — Admin only

Reports (/api/reports)

Método	URL	Descrição	Autenticação
GET	/api/reports/dashboard	Snapshot agregado para dashboard	Sim (JWT)
GET	/api/reports/export/alerts	Exporta alertas (CSV)	Sim (JWT)
GET	/api/reports/export/incidents	Exporta incidentes (CSV)	Sim (JWT)

Tabela 8 - Reports (/api/reports)
Data portability (/api/data-portability)

Método	URL	Descrição	Autenticação
GET	/api/data-portability/sensor-readings/export?sensorId={id?}	Exporta leituras (CSV), opcionalmente filtradas por sensor	Sim (JWT)
POST	/api/data-portability/sensor-readings/import	Importa leituras em lote (JSON)	Sim (JWT)

Tabela 9 - Data portability (/api/data-portability)
Social integrations (/api/social)

Método	URL	Descrição	Autenticação
POST	/api/social/incidents/{id}/share	Partilha um incidente para endpoint externo configurável	Sim (JWT)

Tabela 10 - Social integrations (/api/social)

4.5. Arquitetura da solução (implementação)

4.5.1. Operações implementadas com SOAP (XML / WCF / SoapCore)

Tecnologia/stack

- Serviço SOAP implementado com SoapCore e serialização XmlSerializer.
- Endpoint SOAP: /Service.asmx.

Lista das operações SOAP (IIncidentService)

O contrato SOAP está marcado com [ServiceContract] e cada operação com [OperationContract].

- ReportIncident(type, description, buildingId, severity)

Regista um incidente com StartedAt = UtcNow e Status = "Reported" e devolve uma mensagem com o ID criado.

- GetUnresolvedIncidents()

Devolve incidentes cujo estado não é Resolved (com join ao edifício).

- GetAllIncidents()

Lista todos os incidentes (com join ao edifício).

- GetIncidentById(id)

Obtém um incidente por ID (com join ao edifício).

- CreateIncident(incident)

Cria um incidente (garante StartedAt se vier vazio) e devolve a entidade com o Id atribuído.

- UpdateIncident(id, incident)

Atualiza campos do incidente; devolve true/false consoante existiu linha afetada.

- DeleteIncident(id)

Remove o incidente por ID; devolve true/false consoante existiu linha afetada.

Nota importante (reutilização entre SOAP e REST)

- O mesmo IIncidentService é injetado e usado também no controller REST de incidentes, ou seja, a lógica de incidentes é partilhada entre SOAP e REST.

4.5.2. Operações implementadas com RESTful

Controllers/recursos cobertos (principais)

- Auth (api/Auth/...)
 - POST api/Auth/Register (registro com hash BCrypt)
 - POST api/Auth/Login (devolve token JWT)
 - GET api/Auth/Me (requer autenticação)
 - POST api/Auth/ChangePassword (requer autenticação)
 - Geração do JWT com Jwt:Key/Issuer/Audience e claims (Name/Role).
- Buildings (api/Buildings) – CRUD + detalhe com meteorologia atual via IWeatherService.
- Sensors (api/Sensors) – CRUD.
- SensorReadings (api/SensorReadings) – CRUD.
- Alerts (api/Alerts) – CRUD.
- Incidents (api/Incidents) – CRUD (via IIIncidentService).
- Users (Admin) (api/Users) – CRUD/gestão de utilizadores (apenas role Admin).
- Reports (api/reports/...) – dashboard e exportação CSV de alerts/incidents.
- Data Portability (api/data-portability/...) – export/import de leituras (CSV + import JSON).
- Social Integrations (api/social/...) – partilha de incidente para endpoint externo configurável.

Autenticação nos endpoints REST

- A maioria dos controllers está marcada com [Authorize] (ex.: Buildings/Sensors/Alerts /Incidents/Reports/Data Portability/Social).
- O controller Users exige [Authorize(Roles="Admin")].

4.5.3. Principais serviços isolados e comunicação

Serviços internos (camada de lógica). Registados por DI no Program.cs:

- Incidentes (SOAP/REST): IIncidentService → IncidentService
- Sensores: ISensorService → SensorService
- Edifícios: IBuildingService → BuildingService
- Alertas: IAlertService → AlertService
- Leituras: ISensorReadingService → SensorReadingService
- Utilizadores: IUserService → UserService
- Reporting: IReportingService → ReportingService
- Portabilidade: IDataPortabilityService → DataPortabilityService
- Interações sociais: ISocialIntegrationService → SocialIntegrationService

Como comunicam entre si

- Controllers → Services por injeção de dependências (DI) (ex.: BuildingsController chama IBuildingService e IWeatherService).
- Services → BD através de IDbConnectionFactory (abrem SqlConnection e executam SQL).
- O serviço de incidentes é usado tanto por REST como por SOAP (mesma interface IIncidentService).

Como comunicam com o exterior

- Meteorologia: IWeatherService → OpenWeatherService via HttpClient.
- Partilha/Interações: SocialIntegrationService usa IHttpConnectionFactory (cliente "social-sharing") e faz POST para ApiUrl com payload JSON e (opcionalmente) Authorization: Bearer <ApiKey>.

4.5.4. Sistema de gestão de base de dados utilizado

Tecnologia

- Azure SQL Database (SQL Server), i.e., SQL Server como serviço gerido na cloud Azure.

Justificação breve

- O projeto exige um modelo relacional com integridade referencial (FKs), queries com joins (ex.: edifícios → sensores → leituras/alertas) e consistência transacional, o que é bem suportado por SQL Server.
- A utilização de Azure SQL Database reduz overhead de administração (backups/alta disponibilidade como serviço) e facilita o acesso remoto pela API alojada na mesma cloud.

Estrutura base (tabelas principais)

- Buildings, Sensors, SensorReadings, Alerts, Incidents, Users.

Nota de configuração/deploy

- A connection string usada pela API é configurada no ambiente de execução (Azure App Service settings), podendo diferir do valor usado em desenvolvimento local (ex.: localhost).

4.5.5. Serviços externos

Meteorologia (OpenWeather)

- Serve para enriquecer a consulta de um edifício com o contexto meteorológico (no endpoint de detalhe do edifício).
- Integração via HTTP GET para a API do OpenWeather com lat/lon, units=metric e lang=pt, usando OpenWeather:ApiKey em configuração.

Partilha/integrações externas (SocialNetworks) — com Postman Echo

O sistema inclui uma funcionalidade de partilha de incidentes para um endpoint externo configurável (via /api/social/incidents/{id}/share).

Como não foi utilizada uma API real de rede social (por limitações típicas de credenciais e políticas), foi usado o Postman Echo como endpoint de teste.

O que é o Postman Echo e porquê foi usado?

O Postman Echo é um serviço público que “devolve” (echo) no response os dados que recebeu no request (headers, body, parâmetros).

No projeto, ele é usado para simular a integração com uma rede social/API externa, permitindo validar:

- o payload enviado (conteúdo do incidente, mensagem, etc.);
- a presença de headers (ex.: Authorization: Bearer ..., quando configurado);
- o comportamento do serviço de integração sem depender de um provider real (ex.: Twitter).

Como é feita a integração?

A API constrói um payload JSON com os dados do incidente e envia um HTTP POST para o ApiUrl configurado em SocialNetworks, apontando para o endpoint do Postman Echo (simulação).

Isto permite demonstrar e testar a integração “end-to-end” (API → serviço externo) de forma controlada.

4.5.6. Testes e Validação

A validação do projeto foi suportada por um conjunto de testes automatizados desenvolvidos no projeto SafeHome.Tests, recorrendo ao framework xUnit. Estes testes permitem verificar, de forma sistemática, o comportamento dos serviços e a consistência das regras principais do sistema, nomeadamente nas áreas de CRUD, autenticação/autorização, relatórios, portabilidade de dados e integração externa.

Para garantir reproduzibilidade, os testes utilizam uma base de dados efémera em SQL Server LocalDB, criada dinamicamente a cada execução e removida no final. O esquema é gerado automaticamente (tabelas e relações) através de um utilitário interno (TestDatabase), permitindo validar a lógica com dados realistas sem depender de um ambiente externo permanente. Esta abordagem aproxima os testes de um cenário de integração “leve”, pois valida o comportamento dos serviços em conjunto com a persistência.

A execução dos testes pode ser realizada a partir da solução através do comando:
 “dotnet test”

No projeto existem vários testes distribuídos por diferentes ficheiros. Para simplificar a leitura do relatório, a tabela seguinte apresenta 8 exemplos representativos (um por componente principal), bem como o ficheiro de suporte utilizado no setup/teardown da base de dados de teste.

ID	Ficheiro	O que valida	Exemplo (método)
01	ServiceCrudTests.cs	Operações base (CRUD) e regras essenciais (ex.: hashing)	UserService_HashesPasswords() (linha 59)
02	AdditionalServiceTests.cs	Serviços adicionais (snapshot/portabilidade/rede social/roles)	ReportingService_ReturnsAggregatedSnapshot() (linha 19)
03	AuthorizationAttributeTests.cs	Proteção de endpoints com [Authorize]	ReportsController_HasAuthorizeAttribute() (linha 11)
04	CascadeDeletionAndAuthTests.cs	Eliminações em cascata + fluxos de autenticação + exports	BuildingService_DeleteBuilding_RemovesIncidentsSens

			orsReadingsAndAlerts() (linha 27)
05	ReportingAndPortability EdgeTests.cs	Casos limite em relatórios/CSV /import-export	ReportingService_ExportInc identsCsv_EscapesQuotesIn Description() (linha 69)
06	RobustnessEdgeCaseTest s.cs	Robustez (operações sobre entidades inexistentes, etc.)	BuildingService_DeleteBu ilding>ReturnsFalse_WhenBu ildingDoesNotExist() (linha 26)
07	SocialIntegrationDefault MessageTests.cs	Integração externa (headers/ comportamento quando falta ApiKey)	SocialIntegrationService_Do esNotSetAuthorizationHead er_WhenApiKeyMissing() (linha 42)
08	TestDatabase.cs (<i>suporte</i>)	Infraestrutura de testes (BD efêmera + helpers de seed)	CreateAsync() (linha 25)

Tabela 11 - Testes e Validação

Além destes testes funcionais, foram ainda realizados testes de verificação de segurança ao nível da API, confirmando que endpoints relevantes se encontram protegidos com [Authorize], assegurando que o acesso a funcionalidades sensíveis (relatórios, portabilidade e integrações) exige autenticação válida.

4.5.7. Deploy e Publicação na Cloud (Azure)

A solução SafeHome foi publicada na cloud através da plataforma Microsoft Azure, garantindo disponibilidade e acesso remoto aos serviços implementados. O backend (API) encontra-se alojado em Azure App Service, na região Spain Central, estando acessível publicamente através do seguinte endereço:

- Base URL (API):

<https://safehomeapi-22997-23015-h8dufye2amezh5f4.spaincentral-01.azurewebsites.net>

Para facilitar a validação e o consumo dos serviços, ficam disponíveis os principais endpoints:

Componente	URL
Swagger UI	https://safehomeapi-22997-23015-h8dufye2amezh5f4.spaincentral-01.azurewebsites.net/swagger
OpenAPI (JSON)	https://safehomeapi-22997-23015-h8dufye2amezh5f4.spaincentral-01.azurewebsites.net/swagger/v1/swagger.json
Serviço SOAP	https://safehomeapi-22997-23015-h8dufye2amezh5f4.spaincentral-01.azurewebsites.net/Service.asmx

Tabela 12 - Principais Endpoints

A persistência de dados é assegurada por uma base de dados em Azure SQL, sendo o servidor de base de dados:

- SQL Server (Azure SQL): safehome-server-eu.database.windows.net

A configuração de acesso à base de dados (connection string) e outros parâmetros sensíveis (ex.: chaves externas) são definidos através das App Settings/Configuration do Azure App Service, evitando a inclusão de credenciais no repositório e garantindo uma separação clara entre configuração de desenvolvimento e de produção.

5. Conclusão

A realização deste trabalho prático permitiu aplicar, de forma consistente, os conceitos da Unidade Curricular de Integração de Sistemas de Informação através do desenvolvimento do SafeHome, uma solução orientada à integração e interoperabilidade por serviços. O objetivo central consistiu em disponibilizar uma plataforma de serviços web para suporte à monitorização e gestão de segurança foi concretizado com a implementação de uma API REST e de um serviço SOAP, complementados por documentação e mecanismos de segurança adequados.

Ao longo do desenvolvimento, foi possível consolidar a criação de uma arquitetura funcional e organizada, separando o projeto em backend (API), frontend e um Swagger UI externo, permitindo testar e demonstrar a solução de forma prática. A API foi devidamente documentada com Swagger/OpenAPI e preparada para utilização com autenticação JWT Bearer, assegurando controlo de acesso aos endpoints (ex.: relatórios, portabilidade de dados e integrações sociais). Em paralelo, foi disponibilizado um endpoint SOAP dedicado (via /Service.asmx), reforçando a interoperabilidade com cenários mais tradicionais.

Um aspeto relevante foi a integração com serviços externos, permitindo enriquecer o sistema e aproximar-lo de um cenário real: a componente de meteorologia consome a API da OpenWeather com parâmetros de localização e configuração por chave de API, e a funcionalidade de partilha de incidentes expõe um endpoint dedicado (/api/social/incidents/{id}/share) que comunica com um endpoint externo configurável. Para apoiar a análise e extração de informação, foram ainda incluídas funcionalidades como dashboard e exportação de dados em CSV, bem como mecanismos de importação/exportação de leituras. A robustez do trabalho foi reforçada com testes que validam serviços adicionais (ex.: relatórios, portabilidade e integração externa).

Apesar dos objetivos terem sido atingidos, é importante reconhecer algumas limitações práticas. Por exemplo, em ambientes onde CORS não esteja configurado, um Swagger externo pode falhar; no nosso projeto foi aplicada uma policy CORS permissiva para suportar clientes externos. Além disso, a integração de “partilha” foi desenhada para ser configurável e testável, mas não representa necessariamente uma integração completa com APIs reais de redes sociais, dependendo sempre de credenciais, políticas e requisitos de cada fornecedor.

Como trabalhos futuros, sugere-se:

- (i) reforçar a componente de observabilidade (logs estruturados e auditoria de ações);
- (ii) evoluir a integração social para fornecedores reais, mantendo a estratégia configurável;
- (iii) melhorar o cenário multi-cliente (ex.: proxy para contornar CORS no Swagger externo, ou um gateway dedicado);
- (iv) aprofundar o dashboard com métricas e filtros mais completos, aproveitando os endpoints já existentes.

Em suma, o projeto demonstra com sucesso como uma solução baseada em serviços combinando REST, SOAP, autenticação JWT, documentação OpenAPI e integrações externas consegue transformar um domínio “SafeHome” numa plataforma funcional, extensível e pronta a ser consumida por diferentes tipos de clientes.

6. Referências bibliográficas

API Documentation & Design Tools for Teams | Swagger. (n.d.). Retrieved December 26, 2025, from <https://swagger.io/>

API Management | Microsoft Azure. (n.d.). Retrieved December 26, 2025, from <https://azure.microsoft.com/en-us/products/api-management>

Azure REST API reference documentation | Microsoft Learn. (n.d.). Retrieved December 26, 2025, from <https://learn.microsoft.com/en-us/rest/api/azure/>

Azure SQL Database | Microsoft Azure. (n.d.). Retrieved December 26, 2025, from <https://azure.microsoft.com/en-us/products/azure-sql/database>

C# | Linguagem de programação de código aberto moderno para o .NET. (n.d.). Retrieved December 26, 2025, from <https://dotnet.microsoft.com/pt-br/languages/csharp>

Current weather and forecast - OpenWeatherMap. (n.d.). Retrieved December 26, 2025, from <https://openweathermap.org/>

Download REST & SOAP Automated API Testing Tool | Open Source | SoapUI. (n.d.). Retrieved December 26, 2025, from <https://www.soapui.org/downloads/soapui/>

JSON Web Tokens - jwt.io. (n.d.). Retrieved December 26, 2025, from <https://www.jwt.io/>

Message from Welcome to Solo.io. (n.d.). Retrieved December 26, 2025, from <https://www.solo.io/topics/api-management/azure-api-management>

Microsoft SQL Server. (n.d.). Retrieved December 26, 2025, from <https://www.microsoft.com/en-us/sql-server>

.NET – Crie aplicativos modernos e serviços de nuvem avançados. (n.d.). Retrieved December 26, 2025, from <https://dotnet.microsoft.com/pt-br/>

NuGet Gallery | SoapCore 1.2.1.12. (n.d.). Retrieved December 26, 2025, from <https://www.nuget.org/packages/SOAPCore>

Postman Echo | Documentation | Postman API Network. (n.d.). Retrieved December 26, 2025, from <https://www.postman.com/postman/published-postman-templates/documentation/ae2ja6x/postman-echo>

REST e SOAP: entenda as diferenças. (n.d.). Retrieved December 26, 2025, from <https://www.redhat.com/pt-br/topics/integration/whats-the-difference-between-soap-rest>

What Is a SOAP API and How Does It Work? | Postman Blog. (n.d.). Retrieved December 26, 2025, from <https://blog.postman.com/soap-api-definition/>

What Is JWT? | Postman Blog. (n.d.). Retrieved December 26, 2025, from <https://blog.postman.com/what-is-jwt/>