

Growing Together
Implementação de um Sistema de Tickets

Gonçalo Cardoso Ferreira da Costa
(nrº 26024, regime diurno)

Orientação de
Patrícia Isabel Sousa Trindade Silva Leite

LICENCIATURA EM ENGENHARIA EM SISTEMAS INFORMÁTICOS
ESCOLA SUPERIOR DE TECNOLOGIA
INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

Identificação do aluno

Gonçalo Cardoso Ferreira da Costa
Aluno número 26024, regime diurno
Licenciatura em Engenharia em Sistemas Informáticos

Orientação

Patrícia Isabel Sousa Trindade Silva Leite

Informação sobre o Estágio

Infocávado - Informática Lda
Barcelos, Portugal
Bruno Gabriel Silva Mota

Resumo

No início deste projeto, identifica-se a necessidade de um sistema de tickets para a plataforma interna Growing Together da Inforcávado. Antes da sua implementação, a comunicação com os utilizadores enfrenta desafios significativos, dificultando o atendimento e a resolução de dúvidas e solicitações.

Para resolver este problema, desenvolve-se um sistema de tickets que permite a criação, acompanhamento e gestão de pedidos, proporcionando uma comunicação mais eficiente e organizada. O sistema é concebido para responder tanto às necessidades internas como às solicitações de potenciais clientes, oferecendo opções para esclarecimento de dúvidas, adesão a novos produtos ou apoio a negociações.

Com a implementação deste sistema, regista-se uma melhoria na organização dos pedidos e na eficiência do atendimento, reduzindo barreiras na comunicação e aumentando a satisfação dos utilizadores. Esta evolução representa um avanço significativo na otimização dos processos dentro da plataforma Growing Together.

Abstract

At the beginning of this project, the need for a ticketing system for Inforcávado's internal Growing Together platform was identified. Prior to its implementation, communication with users faced significant challenges, making it difficult to respond to and resolve queries and requests.

To address this issue, a ticketing system was developed that allows requests to be created, monitored and managed, providing more efficient and organized communication. The system was designed to respond to both internal needs and requests from potential customers, offering options for clarifying doubts, subscribing to new products or providing support for negotiations.

With the implementation of this system, there was an improvement in the organization of requests and the efficiency of service, reducing communication barriers and increasing user satisfaction. This development represents a significant advance in the optimization of processes within the Growing Together platform.

Agradecimentos

Inicialmente, gostaria de agradecer a todos o corpo de docentes do IPCA com quem tive a possibilidade de conhecer, por todo o apoio demonstrado durante estes 3 anos de licenciatura, em especial à minha orientadora, a professora Patrícia Leite.

Também gostaria de agradecer a todos a toda equipa da Inforcávado, que me recebeu muito bem no seu ambiente. Agradeço em especial ao meu orientador, Bruno Mota, à Cátia Oliveira, Bruno Costa e em especial à Patrícia Santos, por todo o apoio e ajuda no inicio deste projeto.

Um obrigado especial à Marta pelo apoio todos os dias e por nunca me deixar desistir.

Muito obrigado aos meus pais por sempre me apoiarem em tudo e estarem sempre presentes e a torcer por mim. Sem vocês, nada disto seria possível.

Por fim, aproveito para dedicar esta jornada de 3 anos à minha avó, espero que estejas orgulhosa de mim.

Gonçalo,

31 de maio de 2025

Conteúdo

1	Introdução	1
1.1	Contexto e Motivação	1
1.2	Objetivos	1
1.3	Estrutura	2
2	Estado da Arte	3
2.1	Os Sistemas de <i>Tickets</i> já existentes	3
3	Proposta de Sistema	5
3.1	Visão Geral do Projeto	5
3.1.1	Estrutura do Projeto: Shared, Client e Server	5
3.2	Conceitos e Tecnologias	6
3.2.1	Ambiente de Desenvolvimento Integrado	7
3.2.2	Linguagem de Programação C#	7
3.2.3	Linguagem JavaScript	8
3.2.4	O Dapper	8
3.2.5	SQL	9
3.2.6	Interface de Programação de Aplicações (API)	10
3.2.7	<i>Blazor</i>	10
3.3	Componentes Reutilizáveis	11
3.4	Requisitos	12
3.5	Modelação	15
3.5.1	Diagrama ER	15
3.5.2	Diagrama de Estados	16
3.5.3	Casos de Uso	16
3.6	Arquitetura do Sistema	18
3.7	Mockups	19
3.7.1	Página Inicial	19

3.7.2	Escolha do Tipo de <i>Ticket</i>	20
3.7.3	Formulário de Criação de um Ticket	21
3.7.4	Página de Acompanhamento de <i>Tickets</i>	22
3.7.5	Página de Login	23
3.7.6	Página Interna do Cliente	23
3.7.7	Página de Tickets do Cliente	24
3.7.8	Página de <i>Tickets</i> da Empresa	25
3.7.9	Página de Colaboradores da Empresa	27
4	Implementação do Projeto	29
4.1	Criação dos Modelos	29
4.2	Criação da base de dados no SQL	30
4.3	Desenvolvimento da API	31
4.3.1	Conexão com a Base de Dados	31
4.3.2	Criação dos Repositórios, Controladores e Serviços	32
4.3.3	Serviço de Envio de Emails	37
4.3.4	Sistema de Login	39
4.4	Desenvolvimento da Interface com o Cliente	42
4.4.1	Ligaçāo com a API via Blazor WebAssembly	42
4.4.2	Criação de componentes	43
4.5	Desenvolvimento de um ChatBot	46
4.5.1	A criação do InforBot	46
4.5.2	Arquitetura e atual funcionamento	47
5	Conclusão e Trabalho Futuro	49
5.1	A minha perspetiva	49

Lista de Figuras

3.1	Estrutura do projeto	6
3.2	Ambiente de Desenvolvimento <i>Visual Studio</i>	7
3.3	Diagrama entidade relação	15
3.4	Diagrama de estados de um Ticket	16
3.5	Diagrama de Casos de Uso	17
3.6	Arquitetura do sistema	18
3.7	Página Inicial	19
3.8	Página de Apoio ao Cliente	20
3.9	Página de Comercial	20
3.10	Formulário	21
3.11	Pagina de Acompanhamento de Tickets	22
3.12	Página de Login	23
3.13	Página Interna do Cliente	23
3.14	Página MeusTickets em vista de Tabela	24
3.15	Página MeusTickets em vista de Lista	24
3.16	Página MeusTickets em vista de Cartões	24
3.17	Página EmpresaTickets em vista de Tabela	25
3.18	Página EmpresaTickets em vista de Lista	25
3.19	Página EmpresaTickets em vista de Cartões	26
3.20	Página Colaboradores Empresa	27
4.1	Base de dados criada	30
4.2	Exemplo de email enviado após a criação de um ticket	39
4.3	Componente a ser chamado na página	46
4.4	<i>Inforbot</i> na tela do utilizador	46
4.5	<i>Inforbot</i> na tela do utilizador	47

Lista de Tabelas

3.1 Requisitos Funcionais	13
3.2 Requisitos Não Funcionais	14
3.3 Casos de Uso do Sistema	17

Listagens de Código

3.1	Método assíncrono para criação de um novo Ticket com o Dapper.	9
4.1	Modelo Comentario	29
4.2	Script SQL para criação da tabela Comentario	30
4.3	Exceto do ficheiro <i>appsettings.json</i>	31
4.4	Configuração da ligação no <i>Program.cs</i>	31
4.5	Classe <i>ConnectionStrings</i>	31
4.6	Configuração da ligação no <i>Program.cs</i>	32
4.7	Configuração da ligação no <i>Program.cs</i>	34
4.8	Configuração da ligação no <i>Program.cs</i>	35
4.9	Configuração do envio de emails no <i>appsettings.json</i>	37
4.10	Classe responsável pelo envio de emails	37
4.11	Modelo Utilizador	40
4.12	Login com geração de JWT	40
4.13	Exemplo de endpoint de login	41
4.14	Serviço de login no frontend	41
4.15	Classe ChatBotService com ligação à API	42
4.16	Componente AvisoCookies.razor	43
4.17	Estilos do componente AvisoCookies.razor.css	44
4.18	Lógica de correspondência por palavras-chave	47

Glossário

.NET Plataforma de desenvolvimento da Microsoft para construção e execução de aplicações modernas, que suporta múltiplas linguagens de programação, como C#, F# e VB.NET. 11

API Interface de Programação de Aplicações (Application Programming Interface), que define como componentes de software devem interagir entre si. x, 29, 31, 33, 35, 37, 39, 41

Back end Camada lógica de uma aplicação responsável pelo processamento de dados, regras de negócio e comunicação com a base de dados ou serviços externos. Em um modelo cliente-servidor, o back-end é o servidor. 6, 10, 11

Blazor Framework da Microsoft para desenvolvimento de aplicações web interativas usando C#, HTML e CSS, que pode ser executada no cliente (via WebAssembly) ou no servidor. 10, 46, 49

ChatBot Programa que simula uma conversa com utilizadores humanos, geralmente por meio de linguagem natural, sendo utilizado em atendimento automático, suporte e assistentes virtuais. 15, 46

Dapper Micro-ORM (Object-Relational Mapper) para .NET que permite mapear resultados de consultas SQL diretamente para objetos C#, com foco em desempenho e simplicidade. 49

Dashboard Painel de controlo visual que apresenta dados e métricas importantes de forma resumida e gráfica, permitindo monitorizar o desempenho e tomar decisões informadas rapidamente. 10

DELETE Método HTTP utilizado para remover um recurso do servidor. 10

DevExpress Conjunto de bibliotecas e componentes de interface de utilizador para aplicações .NET, utilizado para acelerar o desenvolvimento de interfaces ricas e interativas. 10, 49

Entity Framework ORM (Object-Relational Mapper) completo para a plataforma .NET, desenvolvido pela Microsoft, que permite trabalhar com bases de dados utilizando objetos .NET, abstraindo as operações SQL e facilitando o desenvolvimento com acesso a dados. 8

Front end Camada de apresentação da aplicação que fornece uma interface amigável ao utilizador. Em um modelo cliente-servidor, o front-end é o cliente, pois é o componente que pode ser manipulado pelo utilizador.. 10, 11, 18

GET Método HTTP utilizado para requisitar dados de um recurso específico em uma API, sem causar efeitos colaterais no servidor. 10

IDE Ambiente de Desenvolvimento Integrado (*Integrated Development Environment*), que reúne ferramentas como editor de código, depurador, compilador e gerenciamento de projetos em uma única aplicação. 7

Media Query Recurso do CSS que permite aplicar estilos condicionais com base em características do dispositivo, como largura da tela, resolução ou orientação, possibilitando o design responsivo. 44

Mockup Representação visual estática de uma interface ou produto, utilizada para demonstrar o design, layout e funcionalidades previstas antes da implementação final, facilitando validações e ajustes com utilizadores ou partes interessadas. 19, 20, 22, 23

Moscow Técnica de priorização de requisitos utilizada em gestão de projetos, dividindo itens em quatro categorias: Must have (deve ter), Should have (deveria ter), Could have (poderia ter) e Won't have (não terá por agora). 12

NewsLetter Boletim informativo digital enviado periodicamente por e-mail, com o objetivo de comunicar novidades, atualizações, promoções ou conteúdos relevantes a uma lista de subscritores. 13, 15, 17, 19

POST Método HTTP utilizado para enviar dados ao servidor e criar novos recursos. 10

PUT Método HTTP utilizado para atualizar ou substituir completamente um recurso existente no servidor. 10

Query Consulta feita a uma base de dados para procurar, inserir, atualizar ou eliminar informações, geralmente escrita em SQL. 8, 32

REST Estilo arquitetural para desenvolvimento de APIs baseado em operações HTTP simples como GET, POST, PUT e DELETE. 11

Separation of Concerns Princípio da engenharia de software que promove a separação de um sistema em partes distintas, de modo que cada parte trate de uma única responsabilidade ou preocupação. Facilita a manutenção, reutilização e escalabilidade do código. 5

SQL Structured Query Language, linguagem padrão para gestão e manipulação de bases de dados relacionais. x, 8, 9, 11, 30, 31

Siglas & Acrónimos

API Application Programming Interface (Interface de Programação de Aplicações). 5, 10, 11, 18

ASP.NET Active Server Pages .NET (Tecnologia da Microsoft para desenvolvimento de aplicações web dinâmicas na plataforma .NET). 10

HTTP HyperText Transfer Protocol. 10, 18

ORM Object-Relational Mapper. 8

REST Representational State Transfer. 10

WPF Windows Presentation Foundation (Fundação de Apresentação do Windows). 10

1. Introdução

Este relatório foi desenvolvido para a unidade curricular Estágio, da Licenciatura em Engenharia de Sistemas Informáticos, da Escola Superior de Tecnologia do Instituto Politécnico do Cávado e do Ave, na empresa *Inforcávado*, localizada em Arcozelo, Barcelos, que decorreu durante 10 de fevereiro a 30 de maio de 2025.

Neste relatório apresenta-se o projeto desenvolvido de raiz pela *Inforcávado*, como o objetivo melhorar a organização entre funcionários/equipa, com um sistema de gestão de tarefas, e também apresentar facilidade de utilização, para que todos os clientes consigam enviar os seus problemas ou as suas dúvidas de forma rápida e prática.

1.1 Contexto e Motivação

A *Inforcávado*, com mais de 20 anos de atividade, foca-se no desenvolvimento de *software* de raiz. Com produtos com o nome bastante conhecidos no mercado, como o *PROTextil*, atualmente utilizado por mais de 150 empresas a nível nacional e o *ATOM-Gest*, um software de Gestão Comercial com mais de 500 instalações, considerada pelos próprios "uma solução simples e eficiente para gerir pequenos negócios" Jornal T (2024).

Não existindo nenhum tipo de sistema de *Ticket* na plataforma interna da Inforcávado, para a comunicação dos clientes, foi apresentado este problema, que visa melhorar a comunicação com os clientes da Inforcávado ou até mesmo com interessados em conhecer/adquirir algum dos serviços disponíveis.

Enquanto futuro engenheiro de software, encarei este desafio com grande interesse e motivação, pois permitiu-me aplicar, pela primeira vez, os conhecimentos teóricos e práticos adquiridos ao longo da licenciatura num caso real, com impacto direto numa empresa sólida e estabelecida no setor. Esta experiência revelou-se extremamente enriquecedora, tanto a nível técnico como pessoal.

1.2 Objetivos

O principal objetivo é o desenvolvimento de um sistema de *Ticket* para apoio ao Cliente, que será inserido dentro do *Growing Together*.

A *Growing Together* é a nova plataforma interna da Inforcávado, que foi desenvolvida para melhorar o mecanismo de gestão de tarefas dos funcionários. Pretende-se desenvolver um sistema de *Ticket* para melhorar a comunicação com os clientes.

1.3 Estrutura

Este relatório contém 5 capítulos, incluindo já com este, onde se pretende contextualizar o leitor:

1. Introdução, como o nome indica, pretende contextualizar o leitor sobre o projeto e a empresa em que foi realizado;
2. Estado da Arte, pretende explicar a pesquisa que foi realizada e a justificação das escolhas para o desenvolvimento do projeto, seja de tecnologias e/ou funcionalidades;
3. Proposta e desenvolvimento do sistema, onde se pretende explicar todo o processo ao longo de todo estágio curricular ligado ao desenvolvimento do projeto, desde a formulação dos requisitos, modelação até ao fim do seu desenvolvimento;
4. Implementação, onde se apresenta o desenvolvimento do projeto numa linguagem mais técnica, onde se explica com maior pormenor a técnica utilizada com exemplos;
5. Conclusão e ambições para o futuro apresenta uma pequena reflexão sobre todo o projeto desenvolvido, e como o mesmo poderá ser útil para os clientes da Inforcávado.

2. Estado da Arte

Entre março e maio de 2025, foi realizada uma pesquisa bibliográfica com o objetivo de recolher conhecimentos teóricos e práticos essenciais para a fundamentação do presente projeto.

A investigação centrou-se em conceitos fundamentais relacionados com sistemas de gestão de *Tickets*, analisando soluções já existentes no mercado, as suas funcionalidades e limitações. Foram também exploradas as linguagens e tecnologias mais adequadas ao desenvolvimento destes sistemas, tendo em conta o percurso tecnológico da organização envolvida. Para além disso, foram estudados métodos de modelação que facilitam uma melhor compreensão e usabilidade por parte dos utilizadores finais. Para enriquecer a análise, além das fontes académicas, como o Google Académico, recorreu-se também à experiência adquirida enquanto utilizador de sistemas de gestão de tickets, o que permitiu uma avaliação mais crítica e realista das soluções existentes.

2.1 Os Sistemas de *Tickets* já existentes

Antes do início do desenvolvimento deste sistema, foi fundamental realizar uma análise dos sistemas de *tickets* já existentes no mercado, de modo a compreender as suas funcionalidades, benefícios e limitações. Estes sistemas são amplamente utilizados em ambientes empresariais para gerir solicitações de suporte, incidentes, pedidos de serviço e outras comunicações internas ou externas.

Ao longo da pesquisa, foi possível observar a grande quantidade de sistemas já desenvolvidos, em que cada um vai ser adaptado às necessidades da sua empresa. A Inforcávado pretende seguir as suas raízes, desenvolver algo 100% da empresa, desenvolvido do início e que esteja sempre pronto para as suas necessidades. Todos os sistemas desenvolvidos que foram analisados apresentavam bastantes funcionalidades em comum, desde a forma de criar um *Ticket*, a partir de um formulário com alguns campos obrigatórios e outros não. Essa forma de o criar foi selecionada como a mais adequada como a secção de criação de um ticket, no entanto, não seria completo suficiente para o *Ticket* poder expressar da forma mais clara e simples, uma zona onde podem ser adicionadas imagens e ou capturas de ecrã do seu problema. Assim, o processo de criação e análise por um funcionário será de menor dificuldade. O *JavaScript* oferece disponibilidades para o desenvolvimento desse tipo processo.

Ao longo da procura, também foram encontrados sistemas de *Tickets* onde o cliente é notificado quando ocorre alguma alteração, como por exemplo, no momento da própria criação do *Ticket*, o utilizador recebe a confirmação da criação com um link para poder acompanhar o mesmo. Isto é bastante comum em várias empresas de *Tickets*, pedidos de

assistência ou por Transportadoras, como *CTT, DPD* e *In Post*.

Foi decidido que essa metodologia também seria utilizada para este sistema de *Tickets*, ou seja, o cliente ao criar o seu *Ticket*, irá receber um email com essa confirmação. Nesse email, também irá um link para o cliente poder acompanhar todo o processo de desenvolvimento do *Ticket*, como comentários e/ ou alterações de estado. Cada *Ticket* tem a sua respetiva página de acompanhamento, como poderemos visualizar no decorrer do desenvolvimento deste projeto.

Apesar de uma larga procura, os sistemas de *Tickets* já existentes não variam muito entre si, seguindo todos a mesma lógica, o cliente cria o *Ticket* e deve aguardar uma atualização/resposta por parte do responsável da entidade que recebeu o *Ticket*.

Um desenvolvimento comum na Inforcávado é o desenvolvimento na forma "interna", isto é, já não é uma página aberta ao público, mas sim privada para cada cliente. Com isto, foi desenvolvido a "consola interna" do utilizador, onde o cliente poderá ver o estado dos seus *Tickets*, todos os *Tickets* que foram criados pelos outros clientes da sua empresa e os contactos dos mesmos. Com isto, pretende-se facilitar o acesso ao cliente, onde ele poderá ver o estado de todos os seus *Tickets* e de todos os seus colegas de trabalho.

Concluindo, embora tenha sido realizada uma pesquisa abrangente sobre sistemas de *Tickets* existentes, muitas das soluções e abordagens analisadas não foram diretamente implementadas no presente projeto, devido às diretrizes e práticas já estabelecidas pela empresa. A Inforcávado valoriza o desenvolvimento interno e personalizado, alinhado com as suas necessidades específicas, o que levou à criação de um sistema próprio, concebido do zero.

No entanto, a análise das soluções existentes foi fundamental para compreender as melhores práticas do mercado, identificar funcionalidades essenciais e inspirar melhorias na experiência do utilizador. Assim, mesmo que não tenha havido uma aplicação direta de muitas das soluções estudadas, a pesquisa bibliográfica contribuiu significativamente para fundamentar decisões técnicas e validar opções adotadas ao longo do desenvolvimento, assegurando um sistema robusto, funcional e alinhado com os objetivos da organização.

3. Proposta de Sistema

Uma proposta de sistema é um documento que descreve detalhadamente o projeto de um sistema de software, abrangendo seus objetivos, funcionalidades, recursos necessários e etapas de desenvolvimento. Além disso, tem como finalidade apresentar a ideia, justificar sua viabilidade e servir como base para a aprovação e execução do projeto.

De acordo com Unhelkar (2020), a proposta de sistema funciona como um guia essencial para alinhar as expectativas entre todos os envolvidos, detalhando os requisitos e definindo as diretrizes para a construção do sistema.

3.1 Visão Geral do Projeto

Este projeto visa o desenvolvimento de uma aplicação web destinada à gestão de *Tickets* de suporte técnico, permitindo aos utilizadores submeter pedidos de assistência, acompanhar o seu estado e facilitar a comunicação com a equipa de apoio.

A aplicação será utilizada em contexto organizacional, promovendo uma maior organização, rastreabilidade e eficiência no tratamento de solicitações internas.

3.1.1 Estrutura do Projeto: Shared, Client e Server

A solução desenvolvida está organizada em três projetos distintos: **Shared**, **Client** e **Server**, como se pode visualizar na figura 3.1. Esta separação segue o princípio da *Separation of Concerns*, que visa atribuir responsabilidades bem definidas a cada componente do sistema, promovendo uma arquitetura mais limpa, modular e de fácil manutenção.

O projeto **Shared** contém as estruturas de dados partilhadas entre o Cliente e o servidor, como classes de modelos, enumerações e validações. Esta abordagem evita a duplicação de código e garante consistência nos dados utilizados em ambas as camadas.

O projeto **Client** é responsável pela interface com o utilizador. Neste módulo são implementadas as páginas e componentes visuais utilizando *Blazor*, bem como a lógica de navegação e as chamadas à API.

Já o projeto **Server** implementa a camada de *backend* da aplicação. Inclui os controladores da API, os serviços de negócio e os repositórios de acesso à base de dados. Esta separação permite uma maior escalabilidade e segurança, já que a lógica de negócio e o acesso aos dados estão isolados do Cliente.

Esta organização modular é recomendada pela Microsoft para aplicações desenvolvidas com *Blazor WebAssembly* hospedadas com ASP.NET Core, conforme descrito na sua

documentação oficial Microsoft (2023).

A aplicação permite que os utilizadores submetam *Tickets* com informações detalhadas, anexando ficheiros ou colando diretamente conteúdos no formulário, os quais são enviados para o Back end.

Com este projeto, espera-se melhorar a comunicação entre os departamentos, assegurar o registo e acompanhamento adequado de pedidos de suporte, e disponibilizar uma ferramenta moderna, eficiente e fácil de utilizar para os colaboradores da organização.

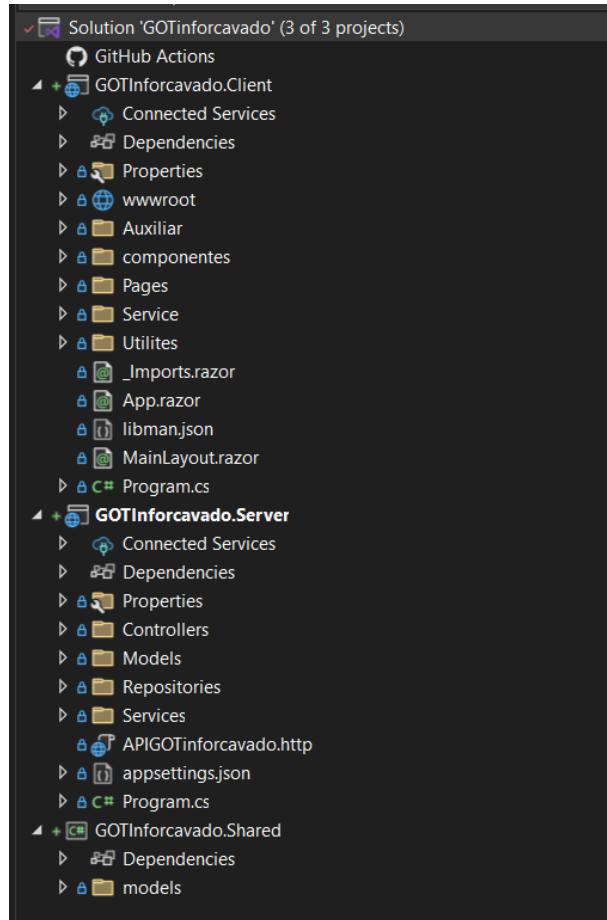


Figura 3.1: Estrutura do projeto

3.2 Conceitos e Tecnologias

Nesta secção abordam-se os principais conceitos que sustentam o desenvolvimento de sistemas de *Tickets*, bem como as tecnologias frequentemente utilizadas na sua implementação. A compreensão destes aspectos é fundamental para a conceção de uma solução eficiente, escalável e alinhada com as necessidades atuais do mercado e dos seus utilizadores.

3.2.1 Ambiente de Desenvolvimento Integrado

O *Visual Studio* é um ambiente de desenvolvimento integrado (IDE) completo desenvolvido pela Microsoft, lançado inicialmente em 1997 McKee (2024). Diferente do *Visual Studio Code*, que é um editor de código mais leve e multiplataforma, o *Visual Studio* é uma plataforma robusta voltada especialmente para o desenvolvimento em ambientes Microsoft, como a plataforma .NET e sistemas Windows McKee (2024); Albahari & Albahari (2022). Ele oferece suporte a diversas linguagens de programação, incluindo C#, C++, .NET e F# Albahari & Albahari (2022).

Esta ferramenta foi escolhida para o desenvolvimento do projeto, não apenas pelas funcionalidades oferecidas, mas também por ser amplamente utilizada na empresa onde o projeto foi realizado. Entre as suas funcionalidades integradas, destacam-se o depurador, designers visuais para interfaces, suporte a testes automatizados e integração com sistemas de controlo de versões e integração contínua McKee (2024); Esposito (2019). Conforme observado por McKee (2024), um IDE eficaz deve reunir editor de código, ferramentas de depuração, compilador e recursos de gestão de projetos — características que o *Visual Studio* satisfaz exemplarmente.

Além disso, o uso de um IDE completo como o *Visual Studio* pode aumentar significativamente a produtividade e a qualidade do código, facilitando o desenvolvimento, testes e manutenção dos sistemas McKee (2024); Esposito (2019).

A figura 3.2 apresenta uma captura de ecrã do ambiente *Visual Studio* utilizado no projeto.

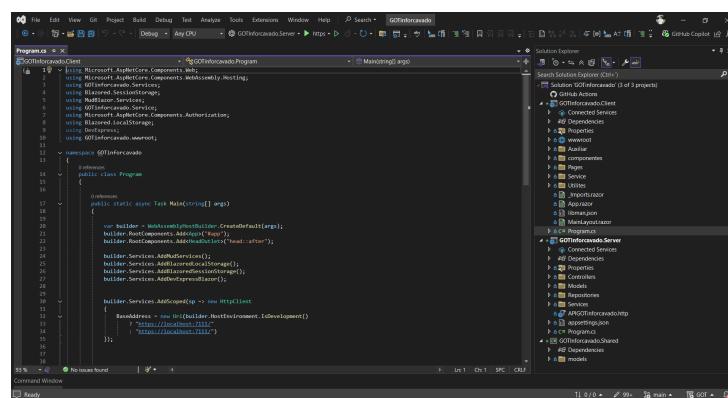


Figura 3.2: Ambiente de Desenvolvimento *Visual Studio*

3.2.2 Linguagem de Programação C#

O C# é uma linguagem de programação moderna e orientada a objetos, desenvolvida pela Microsoft no âmbito da plataforma .NET. É amplamente utilizada para o desenvolvimento de aplicações empresariais, web, desktop e móveis, destacando-se pela sua versatilidade, robustez e desempenho elevado. Uma das principais vantagens do C# reside na sua sintaxe clara e expressiva, que facilita a manutenção e evolução do código. Para além disso, beneficia de um vasto ecossistema de bibliotecas e ferramentas que aceleram o desenvolvimento de soluções complexas. Albahari & Albahari (2022).

No contexto deste projeto, o C# foi selecionado devido à sua integração nativa com a plataforma .NET e às facilidades que proporciona na construção de sistemas escaláveis e

seguros, como é o caso do sistema de *Ticket* em desenvolvimento.

3.2.3 Linguagem JavaScript

O JavaScript é uma linguagem de programação interpretada, baseada em protótipos, amplamente utilizada para tornar páginas da web interativas. Ela roda no navegador e permite manipular o conteúdo da página, responder a ações do utilizador, validar formulários, comunicar-se com servidores entre outras funcionalidades. É uma das três tecnologias fundamentais da Web, junto com HTML e CSS mdn (2024).

3.2.4 O Dapper

Dapper é um micro-Object-Relational Mapper (ORM) para a plataforma .NET, concebido para oferecer uma forma rápida e eficiente de mapear os resultados de consultas SQL diretamente para objetos em C# Stack Exchange (2025). Ao contrário dos ORMs tradicionais, como o *Entity Framework*, que abstraem grande parte da interação com a base de dados, o Dapper adota uma abordagem leve e próxima do SQL, permitindo ao programador manter total controlo sobre as Query executadas McKee (2024).

Um ORM (Object-Relational Mapper) é uma ferramenta que permite a conversão entre dados armazenados em bases relacionais e objetos no código, facilitando o desenvolvimento Albahari & Albahari (2022). Já um micro-ORM, como o Dapper, foca-se em realizar essa tarefa com o mínimo overhead, mantendo a simplicidade e o desempenho.

Esta característica torna o Dapper particularmente indicado para aplicações que exigem alto desempenho e onde a otimização das consultas é crucial. A sua simplicidade e baixo overhead justificam a sua utilização em projetos que valorizam a eficiência sem comprometer a produtividade na manipulação dos dados.

Como se pode visualizar e comprovar a sua simplicidade na listagem de código 3.1, que apresenta o método assíncrono para a criação de um novo *Ticket*.

No desenvolvimento do sistema, para a persistência e manipulação dos dados, foi adotado o micro-ORM Dapper. Em oposição ao *Entity Framework*, que é um ORM completo e mais pesado, o Dapper apresenta uma abordagem mais leve e direta, atuando essencialmente como um mapeador de objetos para consultas SQL escritas manualmente.

Esta opção proporciona maior controlo e flexibilidade na construção das Query, permitindo otimizações específicas e melhor desempenho em operações críticas — aspectos fundamentais para sistemas que requerem respostas rápidas e uma gestão eficiente dos recursos. Adicionalmente, a simplicidade do Dapper facilita a manutenção do código, reduzindo a complexidade da camada de acesso a dados.

A organização utiliza habitualmente o micro-ORM Dapper para acesso a dados nos seus projetos. Esta preferência mantém-se constante ao longo do tempo, devido à leveza, desempenho e facilidade de manutenção do Dapper, evitando a complexidade inerente a frameworks mais robustos como o *Entity Framework*. Assim, o presente projeto segue esta abordagem tecnológica já consolidada na empresa, assegurando coerência com as práticas internas e facilitando a integração com os sistemas existentes.

```

public async Task<Ticket> CreateAsync(Ticket Ticket)
{
    try
    {
        var sql = @"
INSERT INTO Tickets (Codigo, Data, Nome, Empresa, Email,
Assunto, Mensagem, Departamento, TipoTicket,
EstadodoTicket, UtilizadorId, Telefone)
VALUES (@Codigo, @Data, @Nome, @Empresa, @Email, @Assunto,
@Mensagem, @Departamento, @TipoTicket, @EstadodoTicket,
@UtilizadorId, @Telefone);
SELECT CAST(SCOPE_IDENTITY() as int);";

        using (var connection = new SqlConnection(
            _connectionString))
        {
            connection.Open();
            var id = await connection.ExecuteScalarAsync<int>(
                sql, Ticket);
            Ticket.Id = id;
            return Ticket;
        }
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao criar Ticket.", ex);
    }
}

```

Listagem 3.1: Método assíncrono para criação de um novo Ticket com o Dapper.

3.2.5 SQL

O Structured Query Language (SQL) é a linguagem padrão para a gestão e manipulação de bases de dados relacionais. Permite criar, alterar e gerir estruturas de dados, como tabelas, índices e vistas, bem como realizar operações essenciais de inserção, atualização, eliminação e consulta de registo Oppel (2020); Date (2021).

O SQL suporta ainda funcionalidades avançadas, incluindo a definição de restrições de integridade, chaves primárias e estrangeiras, e a gestão de permissões de utilizadores, assegurando a integridade e a consistência dos dados Date (2021). A sua utilização é indispensável em sistemas que dependem do armazenamento estruturado e eficiente de grandes volumes de informação, como acontece nos sistemas de *Tickets*, financeiros e de gestão empresarial.

Além disso, embora o SQL seja um padrão internacional (ANSI/ISO), diferentes sistemas de gestão de bases de dados (SGBDs) implementam variações e extensões próprias, adaptando a linguagem às suas especificidades e funcionalidades Oppel (2020).

3.2.6 Interface de Programação de Aplicações (API)

Uma API (Interface de Programação de Aplicações) corresponde a um conjunto de definições e protocolos que possibilitam a comunicação entre diferentes sistemas ou componentes de software. Neste projeto, a API tem como principal função mediar a comunicação entre o *frontend*, desenvolvido em *Blazor*, e a lógica de negócios implementada no *backend* em ASP.NET.

A API foi concebida segundo o estilo Representational State Transfer (REST), utilizando métodos HyperText Transfer Protocol (HTTP) como GET, POST, PUT e DELETE para realizar operações sobre os dados. Esse padrão promove uma arquitetura simples, escalável e de fácil manutenção, facilitando a integração com outros sistemas e serviços.

3.2.7 *Blazor*

O *Blazor* é uma *framework* para o desenvolvimento de aplicações web, baseado em HTML, CSS e C#. Esta tecnologia permite criar interfaces interativas através de componentes reutilizáveis escritos em C#, promovendo uma maior produtividade e coerência entre o Front end e o Back end.

Uma das principais vantagens do *Blazor* reside na sua flexibilidade, permitindo que a aplicação seja executada tanto no lado do *Tickets* como no servidor. Esta abordagem possibilita a entrega de experiências ricas e responsivas aos utilizadores, mantendo simultaneamente uma forte integração com o ecossistema .NET Freeman (2023); Pialorsi (2020). No contexto deste projeto, o *Blazor* foi adotado como tecnologia principal para o desenvolvimento da interface de utilizador, usando a sua capacidade de forma eficiente e alinhada com a lógica tecnológica já utilizada pela organização.

DevExpress

O *DevExpress* é uma biblioteca de componentes e ferramentas para desenvolvimento de interfaces ricas e interativas em aplicações .NET. Fornece bibliotecas poderosas para a construção de *Dashboard*, tabelas de dados, gráficos, formulários e outros elementos visuais avançados. A sua integração com o *Visual Studio* e a compatibilidade com tecnologias como WinForms, WPF, ASP.NET e *Blazor* tornam o *DevExpress* uma escolha robusta para aplicações empresariais que exigem uma interface de utilizador sofisticada e altamente responsiva.

A Inforcávado utiliza regularmente o *DevExpress* nos seus projetos, especialmente para a criação de interfaces administrativas e painéis internos com elevado grau de interatividade e usabilidade. A adoção desta ferramenta neste projeto contribuiu para acelerar o desenvolvimento da interface, assegurando uma experiência de utilizador consistente e de qualidade, em linha com os padrões já aplicados pela organização. *DevExpress* (2025)

3.3 Componentes Reutilizáveis

A criação de componentes reutilizáveis constitui uma prática essencial no desenvolvimento de aplicações modernas, especialmente em *frameworks* como o *Blazor*, que promove uma abordagem baseada em componentes. Um componente é uma unidade funcional independente da interface, que encapsula estrutura, estilo e comportamento. Pode representar elementos simples, como botões ou campos de formulário, ou estruturas mais complexas, como caixas de texto, formulários de criação de *Tickets* ou tabelas de dados.

A principal vantagem dos componentes reutilizáveis reside na redução de duplicação de código, promovendo uma base de código mais limpa, coesa e fácil de manter. Ao isolar funcionalidades específicas em componentes, é possível reutilizá-los em várias partes da aplicação, garantindo consistência visual e funcional e facilitando futuras alterações ou melhorias.

No contexto deste projeto, a criação de componentes foi essencial para melhorar a produtividade e a escalabilidade do desenvolvimento. Por exemplo, foram criados componentes reutilizáveis para o cabeçalho da aplicação, notificações, caixas de mensagens, formulários de submissão de *Tickets*, e para o carregamento de anexos como imagens ou capturas de ecrã. Estes elementos são utilizados em diferentes partes da aplicação, mantendo um comportamento uniforme e acelerando o desenvolvimento de novas funcionalidades.

Além disso, a utilização de componentes melhora significativamente os testes e a manutenção, uma vez que cada componente pode ser desenvolvido, testado e depurado de forma isolada. Esta abordagem modular também permite que equipas de desenvolvimento trabalhem de forma mais eficiente, distribuindo tarefas específicas relacionadas com componentes distintos.

Assim, a adoção de componentes reutilizáveis alinha-se com as boas práticas de desenvolvimento de software moderno e contribui para a robustez, monitorização e extensibilidade do sistema de *Tickets* desenvolvido.

Interação entre Tecnologias

A interação entre todas as tecnologias mencionadas desempenha um papel crucial no sucesso do sistema de *Tickets* desenvolvido. A escolha do *Blazor* como tecnologia de Front end, aliada à utilização da linguagem C# e da plataforma .NET, permitiu um alinhamento natural com o Back end, garantindo maior consistência entre as camadas da aplicação.

O uso do Dapper na camada de acesso a dados, por sua vez, proporcionou uma abordagem direta e de alto desempenho na comunicação com a base de dados SQL, permitindo que a lógica de negócio se mantenha ágil e orientada ao desempenho, algo essencial em ambientes onde a velocidade de resposta é um fator crítico.

Complementarmente, a API baseada em REST atua como elo central entre o frontend e o Back end, promovendo uma comunicação desacoplada, segura e escalável. Esta arquitetura modular e baseada em boas práticas permite que o sistema evolua facilmente, seja através da introdução de novas funcionalidades, seja pela integração com outros serviços internos ou externos no futuro.

3.4 Requisitos

Os requisitos são descrições detalhadas que estabelecem as necessidades e condições que um sistema, produto ou projeto deve satisfazer. Têm um papel essencial no processo de desenvolvimento, uma vez que orientam todas as fases — desde a conceção até à implementação e validação — assegurando que o sistema cumpre as funções pretendidas e responde às necessidades dos seus utilizadores.

A metodologia Moscow, apresentada de seguida, é amplamente utilizada na gestão de projetos para atribuir prioridades aos requisitos. O termo é um acrónimo que representa quatro categorias de prioridade: Must-haves (Obrigatórios), Should-haves (Recomendados), Could-haves (Opcionais) e Won't-haves (Não incluídos).

Os requisitos podem ser classificados em dois grandes grupos.

Os requisitos funcionais, definem as funcionalidades e os serviços que o sistema deve disponibilizar. Estes descrevem as ações realizadas pelo sistema, bem como as interações possíveis com o utilizador, refletindo as capacidades necessárias para satisfazer os objetivos do sistema.

Abaixo, na tabela 3.1 pode-se observar a lista de requisitos funcionais definidos para este sistema.

Nº Requisito	Descrição	Escala MoSCoW	Utilizador
RF01	O sistema deve permitir ao utilizador submeter um novo Ticket com todas as informações necessárias.	Must	Cliente
RF02	O sistema deve permitir a adição de ficheiros (imagens, documentos) diretamente no formulário.	Should	Cliente
RF03	O sistema deve associar cada Ticket a um departamento e a um tipo de Ticket.	Must	Sistema
RF04	O sistema deve permitir a listagem e pesquisa dos seus Tickets submetidos.	Could	Cliente
RF05	O sistema deve permitir ao utilizador fazer login.	Must	Sistema
RF06	O sistema deve permitir ao utilizador aceder à sua consola interna após realizar o login com sucesso.	Must	Sistema
RF07	O sistema deve enviar o email confirmatório de criação de Ticket.	Must	Sistema
RF08	O utilizador deve conseguir ver todos os seus colegas de trabalho na aba "Colegas".	Must	Sistema
RF09	O utilizador deve conseguir ver todos os Tickets da sua empresa de trabalho na aba "Tickets de Empresa".	Must	Sistema
RF10	Se o utilizador já estiver com o login efetuado com sucesso, ao criar um novo Ticket, no formulário os campos Empresa e Email devem estar automaticamente preenchidos com os dados corretos do Cliente logado.	Could	Sistema
RF11	O utilizador deve conseguir acompanhar o estado de todos os Tickets disponíveis nas diferentes abas (Meus Tickets e Tickets da Empresa).	Should	Sistema
RF12	O utilizador deve conseguir enviar o seu email para se subscrever à <i>Newsletter</i> .	Could	Sistema

Tabela 3.1: Requisitos Funcionais

Já os requisitos não funcionais, apresentados abaixo na tabela 3.2, referem-se a qualidades, restrições e critérios de desempenho que o sistema deve observar. Estes requisitos indicam como o sistema deve operar em determinadas situações, englobando aspectos como a segurança, a usabilidade, o desempenho e a fiabilidade.

Titulo	Descrição
Segurança de Acesso	Implementar um sistema rigoroso de controlo de acesso, garantindo que apenas utilizadores autorizados tenham permissão para visualizar ou manipular informações sensíveis.
Desempenho e Performance - Tempo de resposta	O sistema deve ser capaz de lidar com um número específico de transações simultâneas, e operações comuns, garantindo um tempo de resposta aceitável (menos de 5 segundos) mesmo durante períodos de pico de uso.
Segurança de Dados	O sistema deve estar em conformidade com regulamentações locais e nacionais de privacidade e segurança de dados, como a Lei Geral de Proteção de Dados (LGPD).
Auditabilidade	O sistema deve manter registos detalhados de atividades, permitindo auditorias para garantir a conformidade e rastrear alterações críticas nos dados.
Manutenção	O código do sistema deve ser bem documentado, facilitando a manutenção por parte da equipa de desenvolvimento. Atualizações e manutenções do sistema devem ser realizadas com o mínimo impacto nas operações cotidianas.
Escalabilidade	O sistema deve ser capaz de se adaptar conforme o crescimento da instituição, suportando um aumento no número de utilizadores e volumes de dados sem comprometer o desempenho.
Confiabilidade	Deve ser implementado um sistema de backup regular para garantir a recuperação de dados em caso de falhas no sistema ou perda de dados.
Usabilidade	A interface do sistema deve ser intuitiva e acessível, facilitando o uso por utilizadores com diferentes níveis de experiência e reduzindo a necessidade de treinamento intensivo.
Compatibilidade	O sistema deve ser compatível com diferentes navegadores e dispositivos (desktop, tablet e mobile), garantindo uma experiência consistente para todos os utilizadores.
Disponibilidade	O sistema deve estar disponível para uso 24/7 com um tempo de indisponibilidade planejada mínimo, garantindo alta disponibilidade para os utilizadores.
Portabilidade	O sistema deve ser facilmente migrável para outras plataformas ou ambientes de hospedagem, minimizando custos e esforço em futuras mudanças tecnológicas.

Tabela 3.2: Requisitos Não Funcionais

Após a definição detalhada dos requisitos funcionais e não funcionais, estabelece-se uma base sólida para o desenvolvimento do sistema. Estes requisitos guiarão as etapas subsequentes do projeto, assegurando que as funcionalidades essenciais sejam implementadas conforme as necessidades dos utilizadores, enquanto aspectos cruciais como segurança, desempenho, escalabilidade e usabilidade são devidamente considerados. Com essa estrutura clara, o desenvolvimento poderá ser conduzido de forma organizada, reduzindo riscos e aumentando as chances de sucesso na entrega de uma solução eficaz e alinhada com os objetivos da instituição.

3.5 Modelação

É importante a existência de uma modelação numa proposta de sistema para ajudar na compreensão do sistema por parte dos *stakeholders*.

Abaixo serão apresentados os diagramas que compõem essa modelação.

3.5.1 Diagrama ER

O Diagrama Entidade Relação (ER) é uma representação gráfica usada para representar a estrutura de uma base de dados, que apresenta as entidades (objetos ou elementos de interesse), os atributos das entidades, e os relacionamentos entre elas.

No Diagrama Entidade Relação apresentado abaixo na figura 3.1 mostra como as entidades deste Sistema de *Tickets* se relacionam. É possível observar entidades para armazenar os *Tickets*, os seus Ficheiros (entidade *UploadFiles*), informações sobre os clientes, sobre a NewsLetter e o ChatBot .

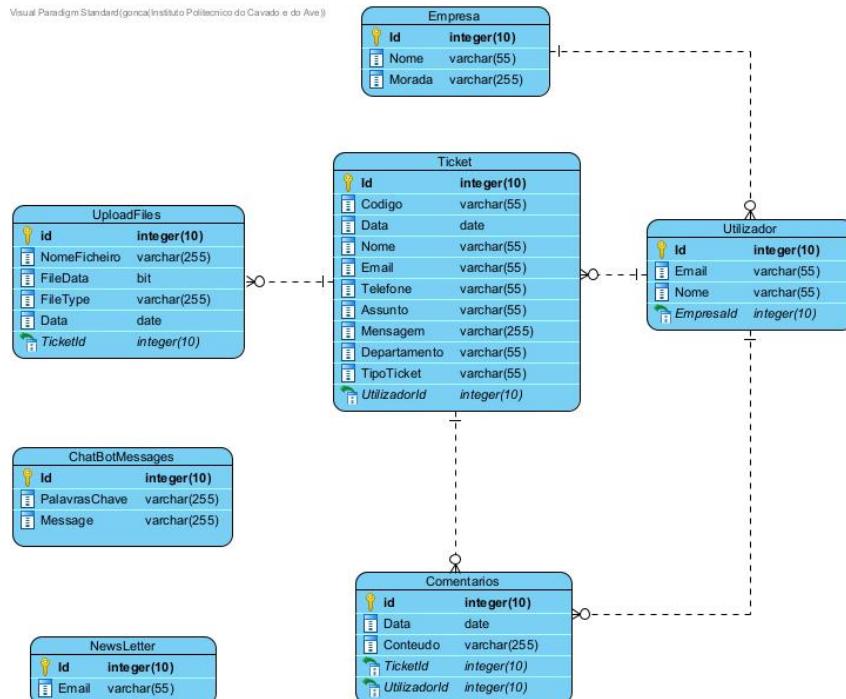


Figura 3.3: Diagrama entidade relação

Esta figura é importante para que o Cliente se aperceba das relações entre as entidades presentes neste sistema.

3.5.2 Diagrama de Estados

Um Diagrama de Estados é uma representação do estado ou situação em que um objeto se pode encontrar no decorrer da execução de processos de um sistema. O diagrama de estados é importante porque permite representar o ciclo de vida de um objeto, apresentando as diferentes etapas pelas quais ele passa, além das transições que ocorrem como resposta a eventos ou condições, o que auxilia na modelagem de comportamentos complexos.

Abaixo, na Figura 3.4, é apresentado o diagrama de estados de um *Ticket*, no qual se podem observar os diferentes estados que compõem o seu ciclo de vida. O processo tem início no estado "Por Iniciar", a partir do qual o sistema avalia se o *Ticket* se encontra em fase de resolução. Consoante essa condição, o *Ticket* poderá transitar para o estado "Pendente" ou "Em Análise". Uma vez concluída a análise, o *Ticket* progride para o estado "Em Curso", onde é objeto de intervenção até ser considerado "Resolvido". Finalmente, após validação da resolução, o *Ticket* altera para o estado "Fechado", concludo assim o seu percurso. Este diagrama proporciona uma representação clara das decisões envolvidas e das possíveis transições entre estados, sendo um elemento fundamental para a modelação de comportamentos complexos no sistema.

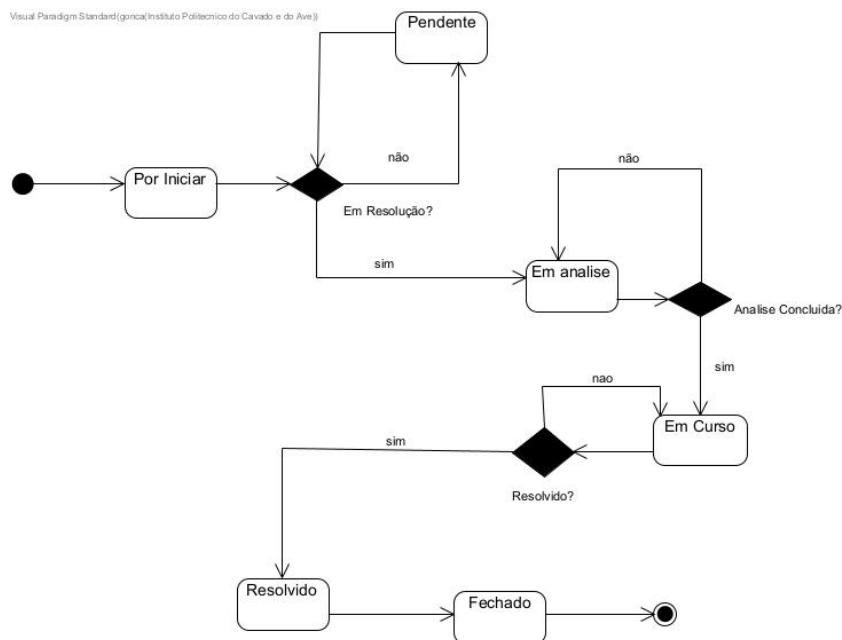


Figura 3.4: Diagrama de estados de um Ticket

3.5.3 Casos de Uso

O diagrama de caso de uso resume as ações dos utilizadores do seu sistema (também conhecidos como atores) e as interações deles com o sistema. Este diagrama representa as ações de todos os atores ativos neste sistema, nomeadamente os clientes, pessoas que fazem os pedidos de assistência, que podem ter ou não uma conta, os Condados não tem conta e os Clientes sim. Assim, essa condição afeta nas ações que os mesmos podem realizar.

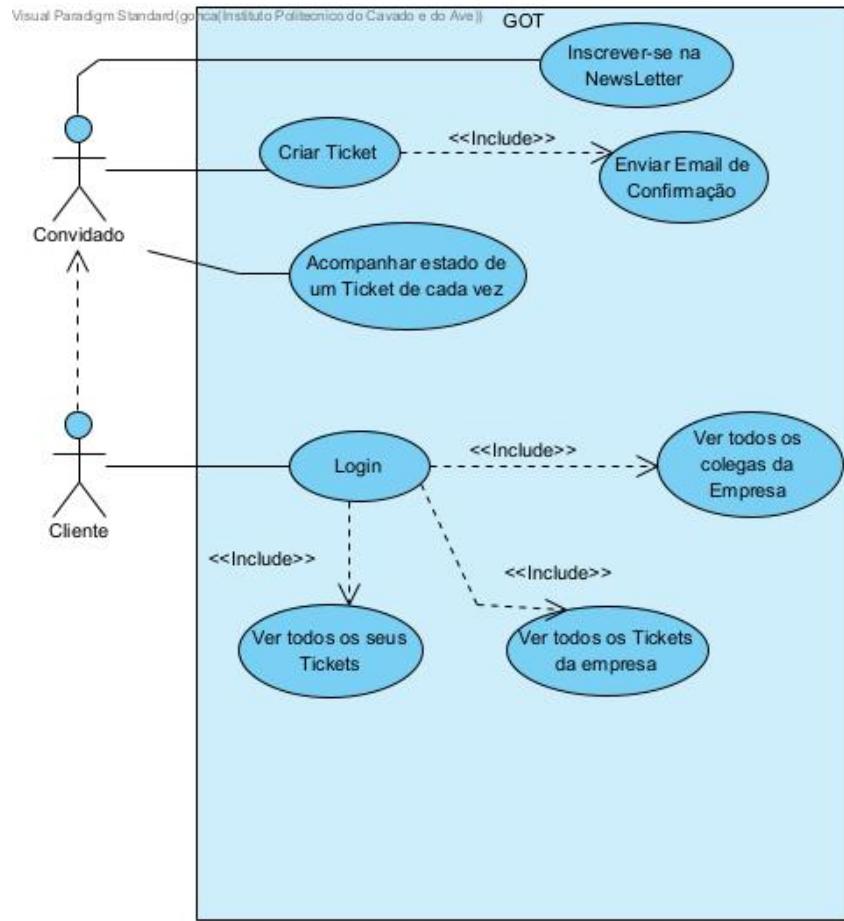


Figura 3.5: Diagrama de Casos de Uso

Numero do caso de Uso	Descrição
UC01	O Convidado pode criar um Ticket no sistema. Este caso de uso inclui o envio de um email de confirmação.
UC02	O Convidado pode acompanhar o estado de um Ticket de cada vez, verificando o progresso ou atualizações do mesmo.
UC03	O Cliente pode fazer login no sistema para aceder a funcionalidades adicionais, como visualizar os seus Tickets e os da empresa.
UC04	Após o login, o Cliente pode ver todos os seus Tickets individuais, além de todos os Tickets da empresa. Também pode visualizar todos os colegas da empresa.
UC05	O Convidado pode inscrever-se na NewsLetter, o que inclui o envio de um email de confirmação.

Tabela 3.3: Casos de Uso do Sistema

3.6 Arquitetura do Sistema

A arquitetura do sistema define a estrutura fundamental do software, especificando os seus principais componentes, as suas responsabilidades e a forma como interagem entre si. Esta organização serve como base para garantir que o sistema seja escalável, seguro, modular e de fácil manutenção.

No contexto deste projeto, a arquitetura adotada segue o padrão **Cliente-Servidor**, utilizando a *framework Blazor* para o Front end e **.NET** para a API *backend*, conectada a uma **base de dados SQL Server**. Esta separação promove a independência entre as partes, facilitando testes, manutenção e evolução do sistema.

Além disso, a comunicação entre os componentes é feita de forma assíncrona via **HTTP (REST API)**, garantindo desempenho e escalabilidade.

Abaixo, na imagem 3.6, podemos observar a representação da arquitetura do sistema deste projeto.

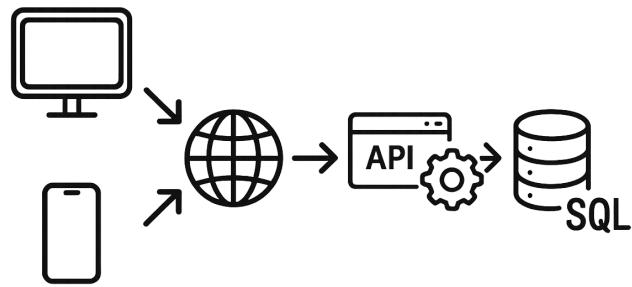


Figura 3.6: Arquitetura do sistema

3.7 Mockups

Um *Mockup* é uma representação visual ou protótipo de um design, usada para apresentar ao Cliente o *layout* e os elementos de um produto, aplicação ou página *web*. Ele fornece uma visão realista da aparência do produto final.

Para o desenvolvimento dos mockups desta aplicação, foi utilizado o *Adobe XD*, uma plataforma dedicada à criação de protótipos visuais. Todos os *mockups* foram elaborados por Cátia Oliveira, *designer* da Inforcávado.

Os *Mockup* foram organizados em duas categorias: com login e sem login, ou seja, páginas acessíveis apenas a utilizadores autenticados e páginas abertas ao público em geral.

Além disso, os *Mockup* foram desenvolvidos com foco na responsividade, garantindo que a aplicação se adapte adequadamente a diferentes tamanhos de ecrã e dispositivos, proporcionando uma experiência de utilizador consistente tanto em computadores como em dispositivos móveis.

3.7.1 Página Inicial

Ao entrar na página de apoio ao Cliente, este encontrará a "página inicial" deste sistema, onde terá diferentes opções tendo em conta o seu objetivo. Se o mesmo tem o objetivo de fazer/criar um novo *Ticket*, terá duas diferentes opções, nomeadamente, "Apoio ao Ticket" e "Comercial". Caso o Cliente queira acompanhar o estado do seu *Ticket*, o mesmo poderá sempre acompanhá-lo. O Cliente poderá ainda inscrever-se na *NewsLetter* da Inforcávado.

Abaixo, na figura 3.7 podemos ver o *Mockup* da página inicial deste sistema de *Tickets*:



Figura 3.7: Página Inicial

3.7.2 Escolha do Tipo de *Ticket*

Após a escolha inicial do Ticket, o mesmo deverá selecionar o Tipo de *Ticket* que pretende criar. Essas opções variam dependendo da escolha anterior, nomeadamente, se foi Apoio ao Cliente ou Comercial. Abaixo, podemos ver o *Mockup* da página de Apoio ao Cliente:

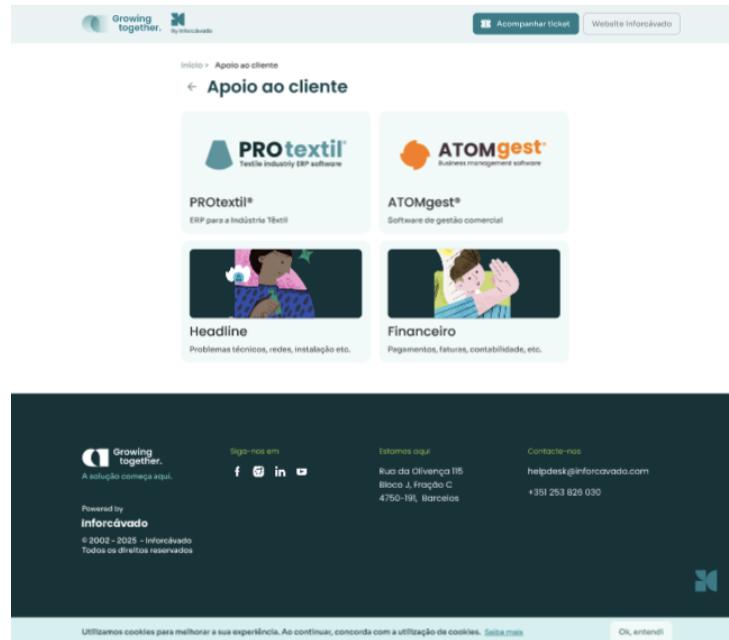


Figura 3.8: Página de Apoio ao Cliente

A outra opção é a de Comercial, que apresenta algumas opções diferentes. Como podemos observar na figura abaixo, segue o *Mockup* da página de Comercial:

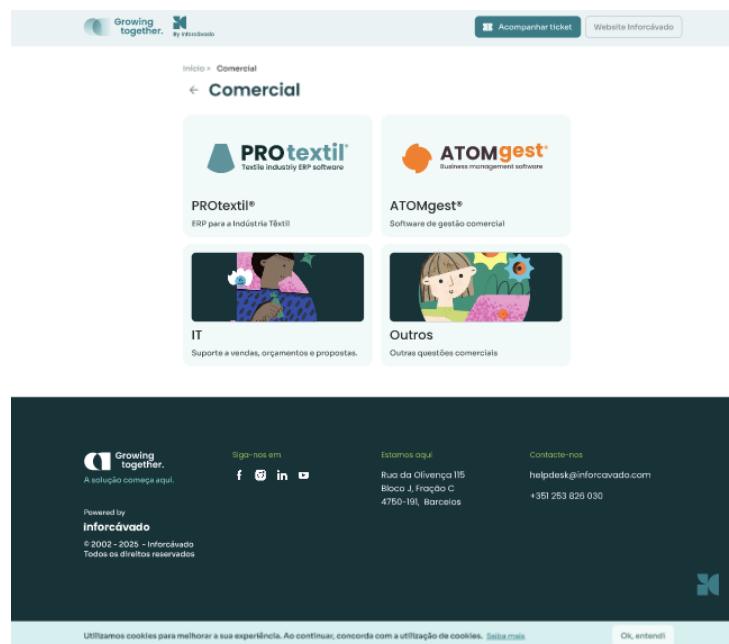


Figura 3.9: Página de Comercial

3.7.3 Formulário de Criação de um Ticket

Após as escolhas serem feitas, a página que se segue é o formulário para criar os *Tickets*. Lá vão ser preenchidos os campos necessários para a criação de um *Ticket*.

Abaixo, a página de formulário para criar *Tickets*:

The screenshot shows a web-based ticket creation form. At the top, there's a header with the PROtextil logo and navigation links for 'Acessar ticket' and 'Webchat Inforcavado'. Below the header, the URL 'ticket > Apoio ao cliente > PROtextil' and a back-link '← PROtextil' are visible.

The main form area contains several input fields:

- Name ***: Input field containing 'Márcia textbus test...'.
- Empresa ***: Input field containing 'Márcia textbus test...'.
- Email ***: Input field containing 'Márcia textbus test...'.
- Telefone**: Input field containing 'Márcia textbus test...'.
- Assunto ***: Input field containing 'Márcia textbus test...'.
- Mensagem ***: A large text area with placeholder text 'Márcia textbus test...' and a red error icon.

Below the message area, there's a section for attachments:

- A placeholder text 'Selecione seu(s) arquivo(s) aqui ou escolha o arquivo para upload'.
- A 'Carregar' (Upload) button.
- Two file attachments listed:
 - 'nome-do_arquivo.txt' (Kind: Text)
 - 'Cenapdo' (Kind: Text)
- Checkboxes for terms of service:
 - Aceito os termos e condições. Meu documento é confidencial.
 - Utilizo este formulário para enviar meu documento.

At the bottom right of the form area, there's a 'Ajuda' (Help) link and a 'Criar ticket' (Create ticket) button.

The footer of the page contains the Growing together. logo, social media links (Facebook, Twitter, LinkedIn), address information ('Gostei de aqui! Rua da Oliveira 115 Bloco J, Freguesia C 4750-161, Barcelos'), contact info ('Contacte-nos helpdesk@inforcavado.com +351 253 826 030'), and a copyright notice ('Powered by Inforcavado © 2002 - 2005 - Inforcavado Todos os direitos reservados').

Figura 3.10: Formulário

3.7.4 Página de Acompanhamento de *Tickets*

Um Cliente também poderá acompanhar o estado do seu *Tickets* pela página de Acompanhamento de *Tickets*. Lá terá todas as informações acerca do seu *Tickets*, além dos dados iniciais, também terá comentários e alterações de estado associados ao mesmo. Abaixo, é possível ver o *Mockup* da pagina de acompanhamento de *Tickets*.

 Growing together.

[Novo ticket](#)

Acompanhamento de tickets

O número do seu ticket permite-nos fornecer-lhe informações atualizadas sobre o estado do pedido que efetuou.

#000 Assunto do ticket

Uma descrição sobre o ticket submetido. Algum tipo que pode ter vários tickets. Pode ser um texto composto por descrever e falar em termos gerais.

00000000

Nome Nunes Soberane	Email exemplo@email.pt	Empresa Nome da empresa, IDA	Telefone (+351) 900 000 000
------------------------	---------------------------	---------------------------------	--------------------------------

CC
exemplo@email.pt ; exemplo@email.pt ;

Arquivo

«Estado atualizado de **Liberado** para **Em análise**»
dia: 24 Oct 2014

» **gomesmotaesrme** criou 2 arquivos


nome do ficheiro que j..._jpg
00000000 - 0144 - 0000


nome do ficheiro que j..._docs
00000000 - 0144 - 0000


nome do ficheiro que j..._pdf
00000000 - 0144 - 0000

dia: 16 Out 2014

» **Nunes Soberane** fixou ticket

«Este ticket está agora para o mesmo objectivo de contacto: escrito que os ficheiros de extensão jpg, .pdf e .docs foram criados para seguir de um livro chamado "Os Hindus Brancos de Daman", escrito pelo autor português de origem Calesa.»

» **gomesmotaesrme** adicionou ticket

Nome Soberane fixou ticket

«Este ticket está agora para o mesmo objectivo de contacto: escrito que os ficheiros de extensão jpg, .pdf e .docs foram criados para seguir de um livro chamado "Os Hindus Brancos de Daman", escrito pelo autor português de origem Calesa.»

» **gomesmotaesrme** criou ticket

Mais tickets na lista

 Growing together.
A solução certeira para a sua.

[Siga-nos em](#)



[Ligue-nos aqui](#)

Rua da Oliveira 165
Bairro J. Fregão C
4750-191, Barcelos

[Contacte-nos](#)

helpdesk@infocavado.com
+351 253 826 000

Powered by
infocavado

© 2002 - 2025 - Infocavado
Todos os direitos reservados

Figura 3.11: Pagina de Acompanhamento de Tickets

3.7.5 Página de Login

Como referido ao longo deste relatório, a Inforcávado pretendia fazer algo diferente dos sistemas de *Tickets*, por isso a criação da página interna para Clientes. Ao introduzir os dados e o login for efetuado com sucesso, o utilizador será reencaminhado para a Página Inicial, onde além de poder realizar todas as funcionalidades que um utilizador sem login pode realizar, além de ter alguns benefícios, como por exemplo, ao preencher o formulário de criação de *Ticket*, se já estiver com o Login efetuado, os campos *Email* e *Empresa* serão completos automaticamente com os dados do utilizador autenticado/logado. Além disso, poderá navegar dentro da página interna do utilizador. Importa salientar que não é possível criar uma conta de forma autónoma, é a Inforcávado que atribui o acesso aos seus Clientes.

Abaixo na figura 3.12, podemos observar o *Mockup* da pagina de Login.

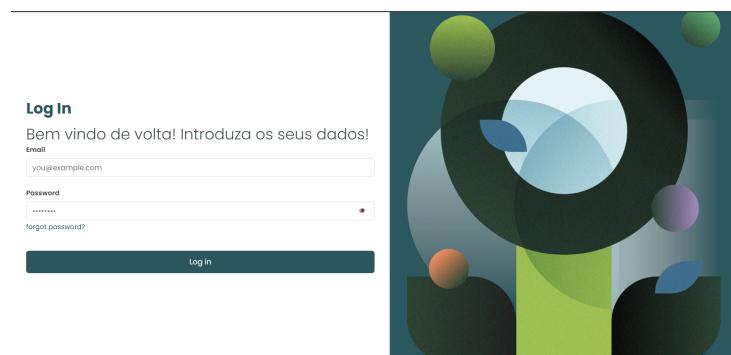


Figura 3.12: Página de Login

3.7.6 Página Interna do Cliente

Após um login bem-sucedido, o Cliente é automaticamente redirecionado para a sua página inicial personalizada. Esta página funciona como ponto central de navegação para todas as funcionalidades disponíveis exclusivamente ao utilizador autenticado.

Cada Cliente possui uma instância individual desta página, assegurando uma experiência personalizada e segura. Através desta interface, o utilizador pode aceder facilmente aos serviços disponibilizados pelo sistema.

Abaixo na figura 3.13, podemos observar o *Mockup* da página interna de Cliente.

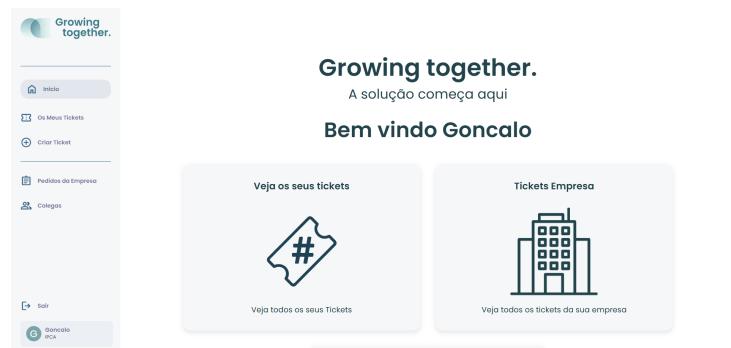


Figura 3.13: Página Interna do Cliente

3.7.7 Página de Tickets do Cliente

Ao selecionar a opção de o Cliente ver os seus *Tickets*, seja na *Navbar* ou no menu principal, o Cliente será reencaminhado para a página onde o Cliente pode ver todos os *Tickets* que foram criados por si listados. O Cliente pode ter várias vistas para ver os seus *Tickets*, a vista em Tabela, figura 3.14, a vista em Lista, figura 3.15 e a vista em Cartões, que esta representado na figura 3.16.

ID	Assunto	Data de Criação	Departamento	Tópico	Estado	Ações
6effb1431ea0d8769	gffffggf	14/05/2025	PROtextil	Apoio	PorIniciar	Email
4fd2f51df12ea273	gfgggfh	14/05/2025	ATOMgest	Comercial	PorIniciar	Email
47f5083ea3e5c835	fghgfghf	14/05/2025	PROtextil	Apolo	PorIniciar	Email
81736834fc7c7464	fhfhhr	15/05/2025	PROtextil	Apoio	PorIniciar	Email
fb235fc85ld70cb5	fhfhhr	15/05/2025	PROtextil	Apoio	PorIniciar	Email

Figura 3.14: Página MeusTickets em vista de Tabela

ID	Assunto	Data de Criação	Departamento	Tópico	Estado	Ações
#6effb1431ea0d8769	gffffggf	14/05/2025	PROtextil	Apoio	PorIniciar	Email
#4fd2f51df12ea273	gfgggfh	14/05/2025	ATOMgest	Comercial	PorIniciar	Email
#47f5083ea3e5c835	fghgfghf	14/05/2025	PROtextil	Apolo	PorIniciar	Email
#81736834fc7c7464	fhfhhr	15/05/2025	PROtextil	Apoio	PorIniciar	Email
#fb235fc85ld70cb5	fhfhhr	15/05/2025	PROtextil	Apoio	PorIniciar	Email

Figura 3.15: Página MeusTickets em vista de Lista

Departamento	Tópico	Ações
PROtextil Apoio	gffffggf 14/05/2025	PorIniciar Email
PROtextil Apoio	4fd2f51df12ea273 fhfhhr 15/05/2025	PorIniciar Email
PROtextil Apoio	47f5083ea3e5c835 fghgfghf 14/05/2025	PorIniciar Email
PROtextil Apoio	81736834fc7c7464 fhfhhr 15/05/2025	PorIniciar Email
PROtextil Apoio	fb235fc85ld70cb5 fhfhhr 15/05/2025	PorIniciar Email
PROtextil Apoio	3a889ddc690005b fdhfhfhg 15/05/2025	PorIniciar Email
PROtextil Apoio	752497c2f27bb8a0 fdhfhfhg 15/05/2025	PorIniciar Email
PROtextil Apoio	7f78e65b696ec429 fdhfhfhg 15/05/2025	PorIniciar Email

Figura 3.16: Página MeusTickets em vista de Cartões

Nas páginas acima representadas, é importante referir que o Cliente não pode apenas ver esta "listagem" dos *Tickets*, mas também pode analisá-lo com maior detalhe, ao carregar no ícone de mensagem, o Cliente será reencaminhado para a página de acompanhamento do seu respetivo *Ticket*.

Além disso, existe uma vista onde o Ticket pode ver em estilo de gráfico de barras a representação dos seus *Tickets*, algo bastante intuitivo para os Tickets verem a quantidade de *Tickets* pelo Departamento do mesmo, por exemplo, PROTextil, ATOMGest, entre outros. Ele apenas serve como uma base de perceber a divisão de *Tickets* até a data da sua consulta.

3.7.8 Página de *Tickets* da Empresa

Ao selecionar a opção de ver os *Tickets* da Empresa, seja na *Navbar* ou no menu principal, o Cliente será reencaminhado para a página onde pode consultar todos os *Tickets* abertos pela sua empresa, ou seja, tanto os *Tickets* criados por si como também os criados pelos seus colegas.

Tal como na página de *Tickets* do Cliente, também estão disponíveis diferentes vistas para facilitar a análise: a vista em Tabela, representada na figura 3.17, a vista em Lista, figura 3.18, e a vista em Cartões, apresentada na figura 3.19.

ID	Criado por	Assunto	Data de Criação	Departamento	Tópico	Estado	Ações
7e61416282f97dd0	bj@gmail.com	2332332	14/05/2025	PROtextil	Apoio	Particular	
6effb1431ce08769	goncalocosta522@gmail.com	gffffgf	14/05/2025	PROtextil	Apoio	Particular	
4fd215df12ea273	goncalocosta522@gmail.com	gtggfh	14/05/2025	ATOMgest	Comercial	Particular	
47f5083ea3e5c835	goncalocosta522@gmail.com	fgfgf	14/05/2025	PROtextil	Apoio	Particular	
81736834fc7c7464	goncalocosta522@gmail.com	fhfhhr	15/05/2025	PROtextil	Apoio	Particular	

Figura 3.17: Página EmpresaTickets em vista de Tabela

Figura 3.18: Página EmpresaTickets em vista de Lista

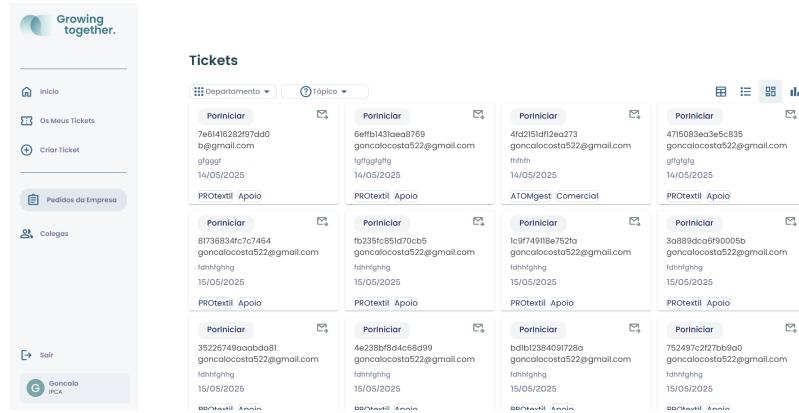


Figura 3.19: Página EmpresaTickets em vista de Cartões

Além das informações básicas de cada *Ticket*, como assunto, estado e data de criação, esta página também exibe o **email do colaborador que criou o *Ticket***. Essa funcionalidade permite um melhor acompanhamento interno, facilitando a identificação de quem submeteu cada pedido.

Tal como na vista individual do *Ticket*, ao carregar no ícone de mensagem presente em cada item da listagem, o utilizador será redirecionado para a página de acompanhamento do *Ticket* selecionado, onde poderá ver mais detalhes e o histórico do atendimento.

Esta página também inclui uma visualização gráfica em formato de gráfico de barras, onde se pode observar de forma clara a distribuição dos *Tickets* por departamento, como por exemplo PROTextil, ATOMGest, entre outros. Esta funcionalidade ajuda o utilizador a perceber rapidamente o volume de *Tickets* em cada área da empresa, até à data da sua consulta.

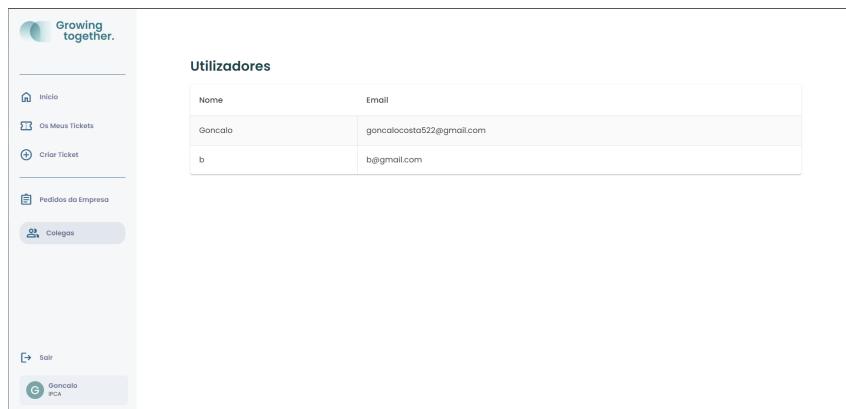
3.7.9 Página de Colaboradores da Empresa

Ao selecionar a opção de visualizar os colaboradores da empresa, o utilizador é encaminhado para uma página dedicada, onde são apresentados de forma clara e organizada todos os colaboradores pertencentes à mesma organização que o Cliente atualmente autenticado.

Nesta página, é possível consultar tanto o **nome** como o **email** de cada colaborador de trabalho, permitindo ao utilizador identificar facilmente os demais membros da sua equipa ou departamento. Esta funcionalidade reforça a transparência e facilita a comunicação interna, promovendo um ambiente mais colaborativo dentro da plataforma.

A apresentação dos dados é feita de forma estruturada, com foco na simplicidade e na legibilidade, de modo a garantir uma navegação intuitiva e uma consulta rápida das informações. Esta página serve, assim, como um ponto de referência para o Cliente ter uma visão global dos utilizadores associados à sua empresa.

O Mockup desta página está apresentado na figura 4.1.



O mockup mostra a interface de usuário da página de colaboradores. No topo, há uma barra com o logo "Growing together." e uma barra lateral esquerda com links para "Início", "Os Meus Tickets", "Criar Ticket", "Pedidos da Empresa", "Colégios" (destacado em azul) e "Sair". O nome do utilizador "Gonçalo" aparece no topo da barra lateral. A seção principal, intitulada "Utilizadores", exibe uma tabela com duas linhas:

Nome	Email
Gonçalo	goncalocosta522@gmail.com
b	b@gmail.com

Figura 3.20: Página Colaboradores Empresa

4. Implementação do Projeto

Neste quarto capítulo, será apresentada em detalhe a implementação deste projeto, desde a criação dos modelos e da API, até ao desenvolvimento da interface do cliente e à interação com o utilizador, bem como a ligação entre essas partes

4.1 Criação dos Modelos

No desenvolvimento de software, os *modelos* constituem uma parte fundamental da estrutura de uma aplicação, sendo responsáveis por representar as entidades e os dados que compõem o domínio da solução. Segundo Martin Fowler, “os modelos representam os dados e as regras que regem o acesso e atualização desses dados. São responsáveis pela lógica de negócio da aplicação” Fowler (2002).

Estes modelos são geralmente definidos como classes que contêm propriedades que refletem os atributos das entidades que existem na base de dados ou no domínio da aplicação. Por exemplo, no caso dos comentários, foi definido um modelo chamado *Comentario*, que armazena informações como a mensagem de resposta e as palavras-chave associadas.

Além disso, os modelos são utilizados em diferentes camadas da aplicação, desde a interface do utilizador, passando pelos serviços, até à camada de persistência de dados, facilitando a comunicação e garantindo a consistência dos dados ao longo do sistema.

Abaixo, na listagem 4.1, podemos ver como exemplo o código que representa o modelo de *Comentario*:

```
public class Comentario
{
    public int Id { get; set; }
    public string Conteudo { get; set; }
    public DateTime Data { get; set; }
    public int UtilizadorId { get; set; }
    public Utilizador? Utilizador { get; set; }
    public int TicketId { get; set; }
    public virtual Ticket? Ticket { get; set; }
}
```

Listagem 4.1: Modelo Comentario

Na imagem 3.3, pode-se ver todos os modelos que foram desenvolvidos para este projeto. Os modelos encontram-se todos na mesma camada deste projeto, denominada *Shared*, que está ligada às restantes camadas da aplicação.

4.2 Criação da base de dados no SQL

Após a criação dos modelos, o passo seguinte foi a criação da base de dados no SQL, uma etapa fundamental para o bom funcionamento deste sistema de *Tickets*. Inicialmente, a base de dados foi criada no servidor local da máquina de desenvolvimento:



Figura 4.1: Base de dados criada

A partir de *scripts*, foram criadas todas as tabelas necessárias com base nos modelos previamente desenvolvidos. Na listagem 4.2, é apresentado o *script* de criação da tabela *Comentario*:

```
CREATE TABLE Comentario (
    Id INT PRIMARY KEY IDENTITY(1,1) ,
    Conteudo NVARCHAR(MAX) NOT NULL,
    Data DATETIME NOT NULL,
    UtilizadorId INT NOT NULL,
    TicketId INT NOT NULL,
    FOREIGN KEY (UtilizadorId) REFERENCES Utilizador(Id),
    FOREIGN KEY (TicketId) REFERENCES Ticket(Id)
);
```

Listagem 4.2: Script SQL para criação da tabela Comentario

Este processo foi repetido para os restantes modelos. Após essa etapa, a base de dados estava pronta.

Este processo é fundamental para o bom funcionamento do projeto.

4.3 Desenvolvimento da API

Nos projetos de software, a camada da API é essencial para o seu funcionamento, pois atua como intermediária entre a base de dados e as aplicações cliente.

4.3.1 Conexão com a Base de Dados

Após a criação da base de dados, conforme demonstrado na etapa anterior, o primeiro passo foi configurar a ligação entre a API e a base de dados. Para isso, foi adicionada ao ficheiro *appsettings.json* do projeto a cadeia de conexão correspondente, conforme apresentado na Listagem 4.3:

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=GONCALO;Database=GrowingT;
      Trusted_Connection=True; TrustServerCertificate=True;"
  }
}
```

Listagem 4.3: Excerto do ficheiro *appsettings.json*

Em seguida, no ficheiro *Program.cs*, foi adicionada a referência à configuração de conexão anteriormente definida:

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.Configure<ConnectionStrings>(
  builder.Configuration.GetSection("ConnectionStrings")
);
```

Listagem 4.4: Configuração da ligação no *Program.cs*

Além disso, foi criada a classe *ConnectionStrings*, responsável por armazenar a cadeia de conexão, a qual será utilizada nas operações com o Dapper:

```
public class ConnectionStrings
{
  public string DefaultConnection { get; set; } = string.Empty
}
```

Listagem 4.5: Classe *ConnectionStrings*

Após esta configuração, a API está pronta para conectar com a Base de Dados no SQL.

4.3.2 Criação dos Repositórios, Controladores e Serviços

Com a base de dados criada e a ligação configurada, foi então desenvolvida a camada da lógica da aplicação, dividida em três partes principais: os repositórios, os serviços e os controladores. Esta separação segue os princípios da arquitetura em camadas, facilitando a manutenção, reutilização e escalabilidade do sistema.

Repositório

O repositório é responsável por interagir diretamente com a base de dados utilizando o Dapper, um micro-ORM que permite executar comandos SQL de forma eficiente, com controlo total sobre as [Query](#). No caso da entidade *NewsLetter*, foi criada a classe *NewsLetterRepository*, que contém métodos para criar, buscar e apagar comentários.

```
public class NewsLetterRepository
{
    private readonly string _connectionString;
    public NewsLetterRepository(IOptions<ConnectionStrings>
        options)
    {
        _connectionString = options.Value.DefaultConnection;
    }
    public async Task<NewsLetter> CreateAsync(NewsLetter
        newsLetter)
    {
        try
        {
            var sql = @"INSERT INTO NewsLetter (Email)
                VALUES (@Email);
                SELECT SCOPE_IDENTITY();";

            using (var connection = new SqlConnection(
                _connectionString))
            {
                connection.Open();
                var id = await connection.ExecuteScalarAsync<int>(sql, new { newsLetter.Email });
                newsLetter.Id = id;
                return newsLetter;
            }
        }
        catch (Exception ex)
        {
            throw new Exception("Erro ao adicionar email à
                newsletter.", ex);
        }
    }
    public async Task<List<NewsLetter>> GetAllAsync()
    {
        try
        {
```

```

        var sql = "SELECT * FROM NewsLetter";

        using (var connection = new SqlConnection(
            _connectionString))
        {
            connection.Open();
            return (await connection.QueryAsync<NewsLetter>(
                sql)).ToList();
        }
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao procurar emails da
            newsletter.", ex);
    }
}
public async Task<bool> DeleteAsync(string email)
{
    try
    {
        var sql = "DELETE FROM NewsLetter WHERE Email =
            @Email";

        using (var connection = new SqlConnection(
            _connectionString))
        {
            connection.Open();
            var affectedRows = await connection.ExecuteAsync
                (sql, new { Email = email });
            return affectedRows > 0;
        }
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao remover email da
            newsletter.", ex);
    }
}
}

```

Listagem 4.6: Configuração da ligação no *Program.cs*

Serviço

A camada de serviço tem como função implementar a lógica de negócios da aplicação, utilizando o repositório como fonte de dados. No caso da ChatBot, a classe ChatBotMessagesService valida os dados recebidos, trata exceções e gera as operações.

```
public class ChatBotMessagesService
{
    private readonly ChatBotMessagesRepository _repository;

    public ChatBotMessagesService(ChatBotMessagesRepository
        repository)
    {
        _repository = repository;
    }

    public async Task<List<ChatBotMessages>>
        ObterTodasMensagensAsync()
    {
        try
        {
            return await _repository.GetAllAsync();
        }
        catch (Exception ex)
        {
            throw new InvalidOperationException("Erro ao obter
                todas as mensagens do chatbot.", ex);
        }
    }

    public async Task<ChatBotMessages> CriarMensagemAsync(
        ChatBotMessages mensagem)
    {
        if (mensagem == null)
            throw new ArgumentNullException(nameof(mensagem));

        try
        {
            return await _repository.CreateAsync(mensagem);
        }
        catch (Exception ex)
        {
            throw new InvalidOperationException("Erro ao criar
                mensagem do chatbot.", ex);
        }
    }

    public async Task<string?>
        ObterMensagemPorPalavrasChaveAsync(string inputUsuario)
    {
        var mensagens = await _repository.GetAllAsync();
```

```

        foreach (var msg in mensagens)
    {
        var palavras = msg.PalavrasChave.Split(' ', ,
            StringSplitOptions.RemoveEmptyEntries | 
            StringSplitOptions.TrimEntries);

        foreach (var palavra in palavras)
        {
            if (inputUsuario.Contains(palavra,
                StringComparison.OrdinalIgnoreCase))
            {
                return msg.Message;
            }
        }
    }

    return null;
}

public async Task<bool> ApagarMensagemAsync(int id)
{
    if (id <= 0)
        throw new ArgumentException("ID inválido.", nameof(
            id));

    try
    {
        return await _repository.DeleteAsync(id);
    }
    catch (Exception ex)
    {
        throw new InvalidOperationException($"Erro ao apagar
            mensagem com ID {id}.", ex);
    }
}
}

```

Listagem 4.7: Configuração da ligação no *Program.cs*

Controlador

O controlador é a camada responsável por expor os endpoints da API RESTful. Recebe as requisições HTTP, valida os dados e encaminha para o serviço apropriado. Abaixo está um exemplo do método que lida com a criação de comentários:

```

using Microsoft.AspNetCore.Mvc;
using APIGOTinforcavado.Services;
using Shared.models;

namespace APIGOTinforcavado.Controllers

```

```

{
    [Route("api/[controller]")]
    [ApiController]
    public class ComentarioController : ControllerBase
    {
        private readonly ComentarioService _comentarioService;

        public ComentarioController(ComentarioService comentarioService)
        {
            _comentarioService = comentarioService;
        }

        [HttpPost]
        public async Task<IActionResult> CreateComentario([FromBody] Comentario comentario)
        {
            if (comentario == null)
                return BadRequest("Dados do comentário inválidos .");

            var createdComentario = await _comentarioService.
                CreateComentarioAsync(comentario);
            return CreatedAtAction(nameof(GetComentarioById),
                new { id = createdComentario.id },
                createdComentario);
        }

        [HttpGet("{id}")]
        public async Task<IActionResult> GetComentarioById(int id)
        {
            var comentario = await _comentarioService.
                GetComentarioByIdAsync(id);
            if (comentario == null)
                return NotFound();

            return Ok(comentario);
        }

        [HttpGet]
        public async Task<IActionResult> GetAllComentarios()
        {
            var comentarios = await _comentarioService.
                GetAllComentariosAsync();
            return Ok(comentarios);
        }

        [HttpGet("por-ticket/{ticketId}")]
        public async Task<IActionResult>
            GetComentariosByTicketId(int ticketId)
        {
            var comentarios = await _comentarioService.
                GetComentariosByTicketIdAsync(ticketId);
        }
}

```

```

        return Ok(comentarios);
    }
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteComentario(int id)
    {
        var success = await _comentarioService.
            DeleteComentarioAsync(id);
        if (!success)
            return NotFound("Comentário não encontrado.");
        return NoContent();
    }
}

```

Listagem 4.8: Configuração da ligação no *Program.cs*

4.3.3 Serviço de Envio de Emails

Uma funcionalidade adicional implementada neste projeto foi o envio automático de emails aquando da criação de um novo *ticket*. Esta funcionalidade tem como objetivo notificar os utilizadores ou administradores sobre novos registo no sistema, promovendo uma resposta mais rápida e eficiente aos pedidos submetidos.

Para tal, foi desenvolvido um serviço denominado *EmailSender*, responsável por enviar mensagens através do protocolo SMTP. O envio é realizado de forma assíncrona utilizando a classe *SmtpClient*, fornecida pelo *framework* .NET. O serviço é configurado para enviar emails através do servidor do Gmail, utilizando as credenciais de uma conta criada especificamente para o efeito.

O serviço foi parametrizado no ficheiro *appsettings.json*, onde estão definidos dados como o host SMTP, a porta, a conta de envio e a respetiva palavra-passe. A listagem 4.9 apresenta esta configuração:

```

"EmailSettings": {
    "FromAddress": "noreply@localhost",
    "SmtpHost": "smtp.gmail.com",
    "SmtpPort": 587,
    "EnableSsl": "true",
    "Username": "goncalocosta522@gmail.com",
    "Password": "jsdy alqs pnjz flyp"
}

```

Listagem 4.9: Configuração do envio de emails no *appsettings.json*

O código do serviço *EmailSender* é apresentado na listagem 4.10. Este serviço lê as configurações a partir do *IConfiguration*, compõe o email com o destinatário, assunto e corpo da mensagem, e realiza o envio de forma segura utilizando SSL.

```

public class EmailSender
{

```

```

private readonly IConfiguration _configuration;

public EmailSender(IConfiguration configuration)
{
    _configuration = configuration;
}

public async Task SendEmailAsync(string toEmail, string
    subject, string body)
{
    var smtpHost = _configuration["EmailSettings:SmtpHost"];
    var smtpPort = int.Parse(_configuration["EmailSettings:
        SmtpPort"]);
    var fromAddress = _configuration["EmailSettings:
        FromAddress"];
    var enableSsl = bool.Parse(_configuration["EmailSettings:
        :EnableSsl"]);
    var username = _configuration["EmailSettings:Username"];
    var password = _configuration["EmailSettings>Password"];

    using (var smtpClient = new SmtpClient(smtpHost,
        smtpPort))
    {
        smtpClient.Credentials = new NetworkCredential(
            username, password);
        smtpClient.EnableSsl = enableSsl;

        var mailMessage = new MailMessage
        {
            From = new MailAddress(fromAddress),
            Subject = subject,
            Body = body,
            IsBodyHtml = true
        };

        mailMessage.To.Add(toEmail);

        await smtpClient.SendMailAsync(mailMessage);
    }
}
}

```

Listagem 4.10: Classe responsável pelo envio de emails

Este serviço é utilizado na camada de negócio responsável pela criação dos *tickets*, sendo invocado logo após a criação de um novo registo na base de dados. A mensagem enviada inclui dados relevantes do *ticket* criado, como o identificador, a descrição e o utilizador associado, conforme ilustrado na figura 4.2.

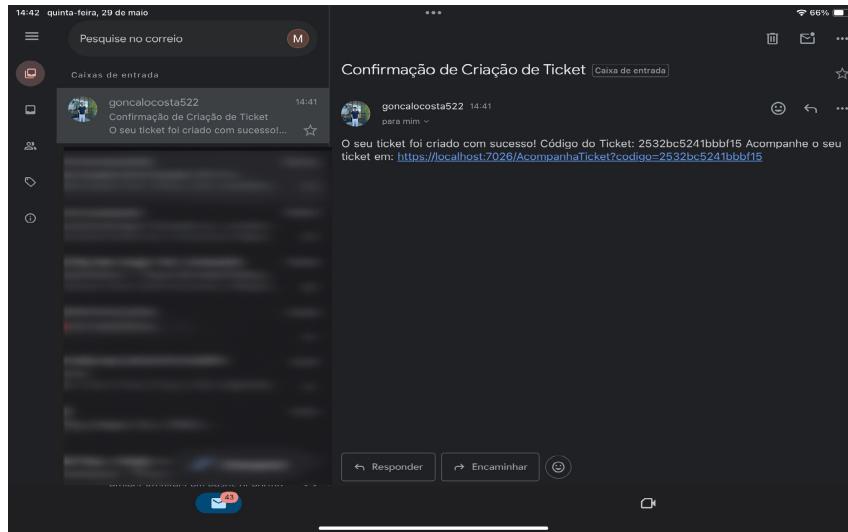


Figura 4.2: Exemplo de email enviado após a criação de um ticket

Com esta funcionalidade, pretende-se melhorar a comunicação entre o sistema e os seus utilizadores, automatizando tarefas rotineiras e reforçando a resposta imediata a eventos relevantes no ciclo de vida dos tickets.

4.3.4 Sistema de Login

Como foi decidido no início deste projeto, este sistema de *Tickets* será integrado como um módulo no projeto principal, o *Growing Together*, que já dispõe de um sistema de login robusto, descrito pelos seus programadores como "uma ferramenta super poderosa".

Apesar disso, foi permitido o desenvolvimento de um sistema de login específico para a consola interna dos utilizadores, com o objetivo de garantir um ambiente isolado, controlado e independente do sistema central. Para tal, foi implementada uma solução baseada em *JWT* (JSON Web Token), que permite autenticação segura e eficiente sem a necessidade de manter sessõesativas no servidor.

Modelo Utilizador

A entidade **Utilizador** contém as informações principais do usuário, incluindo o seu nome, email, password e a ligação com a entidade **Empresa**. Foi também considerado o uso de anotações de validação para garantir integridade de dados:

```
public class Utilizador {
    public int Id { get; set; }
    [Required, EmailAddress]
    public string Email { get; set; }
    [Required]
    public string Password { get; set; }
    [Required]
    public string Nome { get; set; }
    public int EmpresaId { get; set; }
    public Empresa Empresa { get; set; }
    public List<Ticket> Tickets { get; set; } = new List<Ticket>()
}
```

Listagem 4.11: Modelo Utilizador

Lógica de Autenticação

No *backend*, a classe **UtilizadorService** implementa o método de login, que valida as credenciais e, em caso de sucesso, gera um token JWT que é devolvido ao cliente:

```
public async Task<LoginResponse> LoginAsync(LoginRequest request)
{
    var utilizador = await _utilizadorRepository.GetByEmailAsync(
        request.Email);
    if (utilizador == null || request.Password != utilizador.
        Password)
        throw new UnauthorizedAccessException();

    var token = _jwtGenerator.GenerateJwtToken(utilizador);

    return new LoginResponse {
        Token = token,
        Expiration = DateTime.UtcNow.AddHours(3)
    };
}
```

Listagem 4.12: Login com geração de JWT

Este token contém informações essenciais (como ID e email) e pode ser usado em futuras chamadas autenticadas à API. O tempo de expiração de 3 horas foi definido por questões de segurança.

Controller da API

A API disponibiliza vários endpoints para interagir com os dados dos utilizadores, incluindo login, consulta por email, consulta por empresa e atualização de dados:

```
[HttpPost("login")]
public async Task<IActionResult> Login ([FromBody] LoginRequest
request) {
    if (!ModelState.IsValid)
        return BadRequest(ModelState);

    try {
        var response = await _utilizadorService.LoginAsync(
            request);
        return Ok(response);
    } catch (UnauthorizedAccessException ex) {
        return Unauthorized(new { message = ex.Message });
    }
}
```

Listagem 4.13: Exemplo de endpoint de login

Serviço no Frontend

No cliente *Blazor WebAssembly*, foi criado um serviço que realiza chamadas HTTP à API. No caso do login, o token JWT é lido da resposta e pode ser utilizado posteriormente:

```
public async Task<string> AutenticarAsync( string email , string
password ) {
    var loginRequest = new { Email = email , Password = password
};

    var response = await _httpClient.PostAsJsonAsync(LoginUrl ,
        loginRequest);

    if ( !response.IsSuccessStatusCode ) return null;

    var tokenResponse = await response.Content.ReadFromJsonAsync
        <TokenResponse>();
    return tokenResponse?.Token;
}
```

Listagem 4.14: Serviço de login no frontend

Assim, conclui-se que o sistema de login permite que a consola de administração interna funcione de forma autónoma, com autenticação robusta, segura e escalável. A separação clara entre frontend, backend e modelo de dados facilita futuras manutenções, testes e possíveis integrações com outros sistemas.

4.4 Desenvolvimento da Interface com o Cliente

A interface com o cliente foi desenvolvida utilizando a tecnologia Blazor WebAssembly, que permite construir aplicações web interativas em C# e .NET, executadas diretamente no navegador do utilizador. Esta abordagem proporciona uma experiência de utilizador fluida e moderna, sem depender de JavaScript para a maior parte da lógica de interface.

Durante o desenvolvimento, foram criados diversos componentes reutilizáveis para garantir a consistência visual e facilitar a manutenção do código. Componentes como formulários de submissão de tickets, visualização de detalhes, e listagens com filtros foram estruturados de forma modular.

A comunicação com a lógica de negócio é feita através de chamadas assíncronas a APIs REST, garantindo que os dados exibidos estejam sempre atualizados e que a aplicação responda de forma eficiente mesmo em redes mais lentas.

4.4.1 Ligação com a API via Blazor WebAssembly

Para permitir a comunicação entre a interface cliente e o backend, foi implementado um serviço em Blazor WebAssembly que utiliza o `HttpClient`. Este serviço é responsável por enviar e receber dados da API REST desenvolvida, garantindo a separação de responsabilidades e facilitando a manutenção da aplicação.

Um dos serviços criados, o `ChatBotService`, ilustra bem esta integração. Ele contém métodos para enviar uma mensagem ao chatbot e obter uma resposta gerada automaticamente com base nas palavras-chave recebidas. A seguir, na listagem 4.15 apresenta-se um excerto do código da classe:

```
public class ChatBotService
{
    private readonly HttpClient _httpClient;
    public ChatBotService(HttpClient httpClient)
    {
        _httpClient = httpClient;
    }
    public async Task<string> ObterRespostaAsync(string
        mensagemUsuario)
    {
        var resposta = await _httpClient.PostAsJsonAsync("api/
            ChatBotMessages/responder", mensagemUsuario);
        if (resposta.IsSuccessStatusCode)
        {
            var respostaContent = await resposta.Content.
                ReadFromJsonAsync<ResponseMessage>();
            return respostaContent?.Resposta ?? "Desculpe, não_
                consegui entender sua pergunta.";
        }
        return "Erro ao obter resposta.";
    }
}
```

Listagem 4.15: Classe ChatBotService com ligação à API

4.4.2 Criação de componentes

Nesta fase do desenvolvimento, foram criados diversos componentes reutilizáveis com o objetivo de modularizar a interface do utilizador, melhorar a organização do código e facilitar a manutenção da aplicação.

Cada componente foi pensado para cumprir uma função específica, permitindo a sua reutilização em diferentes partes do sistema e promovendo uma maior coesão entre as funcionalidades implementadas. Esta abordagem segue os princípios da programação orientada a componentes, sendo particularmente eficaz no contexto do Blazor WebAssembly, onde a separação de responsabilidades e a reutilização de elementos visuais são fundamentais para o bom desempenho e a escalabilidade da aplicação.

Cada componente em Blazor é normalmente desenvolvido no seu próprio ficheiro .razor, onde se encontra a lógica e o markup associado. Adicionalmente, é possível separar os estilos de cada componente utilizando o ficheiro .razor.css, o que promove uma melhor organização e encapsulamento dos estilos, mantendo o princípio da modularidade. Assim, o Blazor aplica os estilos CSS de forma scoped, garantindo que afetam apenas o componente correspondente.

Na listagem, podemos ver a criação do componente AvisoCookies , que estara presente em bastantes páginas deste sistema de *Tickets*.

```
@using Blazored.SessionStorage
@inject ILocalStorageService SessionStorage
@if (CookieVisivel)
{
    <div class="cookie">
        <div class="texto-container">
            <span class="texto">
                Utilizamos cookies para melhorar a sua experiência. Ao continuar, concorda com a utilização de cookies.
                <a href="https://inforcavado.com" class="sublinhado" target="_blank">
                    Saiba mais
                </a>
            </span>
        </div>
        <button class="fechar-avisocookies" @onclick="Desaparece">Ok, entendi</button>
    </div>
}
@code {
    private bool CookieVisivel = true;
    protected override async Task OnInitializedAsync()
    {
        var accepted = await SessionStorage.GetItemAsync<string>("cookiesAccepted");
        if (accepted == "true")
        {
            CookieVisivel = false;
        }
    }
}
```

```

        }
    }
    private async Task Desaparece()
    {
        CookieVisivel = false;
        await SessionStorage.SetItemAsync("cookiesAccepted", "true");
    }
}

```

Listagem 4.16: Componente `AvisoCookies.razor`

Cada componente Blazor pode possuir um ficheiro de estilos dedicado com o mesmo nome do componente e a extensão `.razor.css` (por exemplo, `AvisoCookies.razor.css`), que permite isolar os estilos aplicados apenas a esse componente. Esta abordagem promove a modularização do código e facilita a manutenção do projeto.

Além disso, é comum o uso de Media Query neste ficheiro de estilos, que são regras CSS utilizadas para criar layouts responsivos. As Media Query permitem adaptar a apresentação do componente a diferentes tamanhos de ecrã, como dispositivos móveis, tablets e ecrãs maiores. Isto garante que o conteúdo seja exibido corretamente e de forma acessível em qualquer dispositivo. No exemplo apresentado, as Media Query ajustam o layout do banner de cookies consoante a largura da janela do navegador, alterando espaçamentos, tamanhos e disposição dos elementos para melhorar a usabilidade e a experiência do utilizador.

```

.cookie {
    height: 56px;
    background: #D1F0EE;
    display: flex;
    align-items: center;
    justify-content: center;
    padding: 0 140px;
}

.texto-container {
    display: flex;
    gap: 5px;
    align-items: center;
}

.texto {
    font-family: "Sora";
    font-size: 14px;
    font-weight: 400;
    color: #0D3439;
}

.sublinhado {
    color: #1B7C87;
    text-decoration: underline #1B7C87;
}

```

```

.fechar-avisocookies {
    margin-left: 78px;
    width: 116px;
    height: 40px;
    background-color: #F0FAF9;
    border: none;
    border-radius: 8px;
    font-family: "Sora";
    font-size: 14px;
    color: #0D3439;
    cursor: pointer;
}

@media (max-width: 640px) {
    .cookie {
        flex-direction: column;
        height: 144px;
        padding: 0 40px;
    }

    .texto {
        width: 308px;
    }

    .fechar-avisocookies {
        margin-left: 0px;
        transform: translateY(-100%);
    }
}

```

Listagem 4.17: Estilos do componente `AvisoCookies.razor.css`

Com a criação de componentes, o código torna-se mais simples, organizado e reutilizável. Uma vez criado o componente, basta invocá-lo nas páginas onde é necessário. Por exemplo, o componente `AvisoCookies` foi desenvolvido para apresentar uma barra de consentimento de cookies e está presente em várias páginas da aplicação. Como se trata de um componente, não é necessário repetir o mesmo bloco de código em cada página. Em vez disso, pode-se simplesmente adicionar a diretiva `<AvisoCookies />` onde for necessário.

Esta abordagem reduz a duplicação de código, facilita futuras alterações (pois qualquer modificação feita no componente refletir-se-á automaticamente em todas as páginas que o utilizam) e contribui para uma maior supervisão do projeto. A separação de responsabilidades entre o conteúdo da página e os seus componentes torna o desenvolvimento mais ágil e o código mais limpo e escalável.

Abaixo, na imagem 4.3 , podemos observar o componente a ser chamado e como o seu processo é simples.

```

82      </div>
83
84      <Rodape />
85
86      <Cookies />
87
88

```

Figura 4.3: Componente a ser chamado na página

4.5 Desenvolvimento de um ChatBot

É importante referir que, além dos requisitos atribuídos inicialmente, foi incentivado, por parte das entidades que supervisionam o programadores deste projeto, que os desenvolvedores tivessem liberdade para adicionar funcionalidades que pudessem ser úteis e funcionais para os utilizadores.

4.5.1 A criação do InforBot

Nesse contexto, foi decidido criar um ChatBot, que simula uma conversa com os utilizadores, geralmente por meio de linguagem natural, sendo utilizado em atendimento automático, suporte e assistentes virtuais Naziri & Das (2020).

O *Inforbot*, como foi nomeado, foi desenvolvido como um componente reutilizável em Blazor, o que facilita a sua integração em diferentes partes da aplicação. Este componente foi incorporado em todas as páginas associadas ao Sistema de *Tickets*, com possibilidade futura de extensão para as restantes páginas das diversas aplicações web da empresa.

A sua presença visa proporcionar um canal de apoio rápido e acessível, respondendo a dúvidas frequentes dos utilizadores e melhorando a experiência de uso da plataforma.

Abaixo, na figura 4.4 podemos observar com maior detalhe como o *Inforbot* aparece disponível no ecrã do utilizador.



Figura 4.4: *Inforbot* na tela do utilizador

Assim de forma rápida e direta, o utilizador poderá colocar as suas perguntas sobre a Inforcávado, sem ser necessário entrar em contacto diretamente com empresa.

4.5.2 Arquitetura e atual funcionamento

O Inforbot foi desenvolvido com base em um sistema de identificação de palavras-chave nas mensagens enviadas pelos utilizadores. Sempre que o utilizador escreve uma mensagem, o sistema realiza a leitura do conteúdo textual e, em seguida, procura por palavras-chave previamente definidas.

Caso sejam encontradas correspondências, é efetuada uma consulta a uma base de dados relacional que contém um conjunto de respostas associadas a essas palavras-chave. O resultado dessa consulta é então apresentado ao utilizador de forma dinâmica e intuitiva, simulando um diálogo natural. Todas as respostas disponíveis no *Inforbot* são previamente definidas e armazenadas por administradores do sistema, garantindo controlo sobre a informação disponibilizada e a consistência nas interações.

O componente foi implementado em *Blazor WebAssembly*, utilizando o padrão de desenvolvimento modular, o que permite a sua fácil manutenção e evolução. Para a comunicação com o backend, foi utilizado o padrão REST, garantindo segurança na troca de informações e compatibilidade com os serviços existentes da empresa.



Figura 4.5: *Inforbot* na tela do utilizador

A seguir, na listagem 4.18 apresenta-se um a função que apresenta como o sistema realiza a correspondência entre a mensagem do utilizador e as respostas armazenadas na base de dados, com base em palavras-chave.

```
public async Task<string?> ObterMensagemPorPalavrasChaveAsync(
    string inputUser)
{
    var mensagens = await _repository.GetAllAsync();
    foreach (var msg in mensagens)
    {
        var palavras = msg.PalavrasChave
            .Split(',', StringSplitOptions.RemoveEmptyEntries |
            StringSplitOptions.TrimEntries);
        foreach (var palavra in palavras)
        {
            if (inputUser.Contains(palavra, StringComparison.OrdinalIgnoreCase))
            {

```

```
        return msg.Message;
    }
}
return null;
}
```

Listagem 4.18: Lógica de correspondência por palavras-chave

Como se pode observar, o método percorre todas as mensagens configuradas na base de dados. Para cada uma delas, verifica-se alguma das suas palavras-chave está contida na entrada do utilizador. Caso seja encontrada uma correspondência, a resposta correspondente é devolvida.

5. Conclusão e Trabalho Futuro

O desenvolvimento deste projeto exigiu uma fase inicial de pesquisa e aquisição de conhecimento sobre as práticas comuns no desenvolvimento de software, especialmente no contexto de aplicações web modernas.

Durante esse processo, foi necessário aprender e aplicar tecnologias específicas, como o DevExpress, o Blazor e o Dapper, compreendendo suas funcionalidades e como utilizá-las de forma eficiente no desenvolvimento da aplicação. Essas ferramentas demonstraram-se essenciais para a criação de interfaces interativas, comunicação com a base de dados e otimização de desempenho.

Além da implementação técnica, este projeto proporcionou um aprofundamento na arquitetura de aplicações web, no uso de APIs REST e nos padrões de autenticação e autorização, com especial atenção à segurança e à escalabilidade.

Este projeto foi concebido como um módulo independente, projetado para futura integração ao projeto principal, o *Growing Together*. Desde a sua conceção, foi adotada uma abordagem modular que permite que esta aplicação seja facilmente incorporada à arquitetura maior do sistema, contribuindo com funcionalidades específicas e complementares.

A integração ao *Growing Together* permitirá ampliar a utilidade do projeto, promovendo uma sinergia entre os diferentes componentes do sistema e reforçando a visão de um ecossistema colaborativo e em constante evolução. Este módulo será responsável por gerir e facilitar a comunicação com os seus clientes e melhorar a forma de responder aos seus pedidos, acrescentando valor à solução global oferecida pelo projeto *mãe*.

Como ambição para o futuro, pretende-se expandir a aplicação com novas funcionalidades, como integração com serviços externos, melhorias na interface com foco na experiência do utilizador e eventualmente a implementação de testes automatizados para garantir maior robustez.

Este projeto serviu como uma base sólida para futuras iniciativas mais complexas, estabelecendo fundamentos técnicos e práticos que poderão ser reutilizados e aprimorados em projetos futuros.

5.1 A minha perspetiva

Após terminado este projeto, sinto-me na oportunidade de poder fazer uma conclusão numa fora do tom formal e técnico que foi apresentado ao longo de todo o relatório.

Este projeto marcou o meu primeiro estágio na área de desenvolvimento de software, o que, por si só, representou um marco importante na minha formação. Ter a oportunidade

de aplicar conhecimentos teóricos em um ambiente real de trabalho foi extremamente enriquecedor. Mais do que aprender novas tecnologias, aprendi a trabalhar em equipa, a comunicar ideias de forma clara e a ouvir sugestões e críticas construtivas.

A dinâmica das reuniões semanais foi essencial para o bom andamento do projeto. Estes encontros permitiram não só alinhar objetivos e prioridades, mas também criar um espaço de partilha onde me senti sempre apoiado. O acompanhamento constante da equipa deu-me confiança para evoluir e assumir responsabilidades progressivamente mais complexas.

Quanto ao projeto em si, considero que os objetivos principais foram plenamente cumpridos. Conseguí desenvolver um módulo funcional, com integração pronta para o sistema principal, respeitando os requisitos definidos inicialmente. Para além disso, tive ainda a oportunidade de ir além do que era esperado: implementei funcionalidades extra, propus melhorias e otimizei partes do sistema para garantir melhor desempenho e manutenibilidade.

No fim, este estágio não só consolidou os conhecimentos que já possuía, como também me abriu portas para novas aprendizagens e, sobretudo, deu-me a certeza de que estou no caminho certo. Saio desta experiência com orgulho no que construí e motivação redobrada para os próximos desafios.

Bibliografia

2024. Javascript | mdn. Acessado em: abril 2025. <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.
- Albahari, Joseph & Ben Albahari. 2022. *C# 11 in a nutshell: The definitive reference*. O'Reilly Media.
- Date, C. J. 2021. *An introduction to database systems*. Pearson Education.
- DevExpress. 2025. Devexpress documentation – .net ui controls. Acedido em: maio 2025. <https://docs.devexpress.com/>.
- Espósito, Dino. 2019. *Programming asp.net core: Building modern web apps with .net*. Microsoft Press.
- Fowler, Martin. 2002. *Patterns of enterprise application architecture*. Addison-Wesley.
- Freeman, Adam. 2023. *Pro asp.net core 7: Develop cloud-ready web applications using mvc, blazor, and razor pages*. Apress.
- Jornal T. 2024. Growing together – um novo projeto de capacitação profissional. Acedido em: maio 2025. <https://jornal-t.pt/servicoespecializado/growing-together/>.
- McKee, John. 2024. *Mastering the visual studio ide*. Apress.
- Microsoft. 2023. Blazor project structure. Microsoft Learn. <https://learn.microsoft.com/en-us/aspnet/core/blazor/host-and-deploy/webassembly?view=aspnetcore-8.0>.
- Naziri, Rashid Khan & Anik Das. 2020. *Designing bots: Creating conversational experiences*. O'Reilly Media.
- Oppel, Andy. 2020. *Sql: A beginner's guide*. McGraw-Hill 5th edn.
- Pialorsi, Paolo. 2020. *Programming blazor: Building web applications with webassembly and c#*. Microsoft Press.
- Stack Exchange. 2025. Dapper tutorial. Acessado em: junho 2025. <https://dapper-tutorial.net/>.
- Unhelkar, Bhuvan. 2020. *Software engineering with uml*. CRC Press.