TicketHelper

Relatório

Tecnologias da internet III

Diogo Regadas 2022103

Introdução

Este relatório tem como objetivo descrever o desenvolvimento e implementação de um sistema de gestão de tickets, focado em aprimorar a eficiência e a organização na resolução de problemas relacionados a hardware e software em uma organização. Este projeto surge da necessidade de otimizar a comunicação entre utilizadores e equipas de suporte técnico, bem como de fornecer uma solução integrada que possibilite o acompanhamento detalhado e sistemático de cada solicitação de serviço.

Índice

Descrição do Problema

O objetivo deste projeto é criar uma plataforma de gestão de tickets para atender solicitações de serviços de hardware (HW) e software (SW). A plataforma visa otimizar o processo de registro, acompanhamento e resolução de tickets, garantindo que todas as solicitações sejam atendidas de forma eficiente e organizada.

Objetivos

Melhorar a eficiência - Automatizar o processo de registo e atendimento de tickets para reduzir o tempo de resposta e aumentar a produtividade dos técnicos de helpdesk.

Organizar solicitações - Manter um registo organizado e acessível de todas as solicitações de hardware e software.

Aumentar a transparência - Fornecer aos utilizadores a capacidade de acompanhar o status dos seus tickets em tempo real.

Gerir acessos - Implementar um sistema de login e gestão de permissões para assegurar que apenas utilizadores autorizados possam aceder determinadas funcionalidades.

Requisitos Funcionais

Gestão de Tickets

- Permitir a criação de tickets de hardware e software.
- Registar automaticamente o identificador, data e hora da criação do ticket.
- Associar o ticket ao utilizador que o criou.
- Permitir a atualização do status do ticket (e.g., “porAtender”, “atendido”).
- Registar automaticamente a data e hora do atendimento do ticket.
- Associar o ticket ao técnico que o atendeu.

Atendimento de Tickets

- Permitir que os técnicos visualizem e atendam tickets.
- Atualizar os tickets com a descrição da intervenção e/ou reparação, peças utilizadas e o estado de atendimento.

Visualização de Tickets

- Permitir que os utilizadores listem tickets e visualizem os detalhes e status de cada ticket.
- Permitir a filtragem de tickets a mostrar pelo tipo de ticket (e.g., “Hardware”, “Software”).

Requisitos Não-Funcionais

Automação de Respostas e Atendimentos Padrão

- Implementar a capacidade de definir respostas automáticas para perguntas frequentes ou problemas comuns, agilizando o processo de atendimento e proporcionando uma experiência consistente ao utilizador.

Notificações Automatizados:

- Configurar notificações automáticas para informar os utilizadores sobre atualizações no status de seus tickets (por exemplo, quando um ticket é atendido ou existe alguma mudança de status). Além disso, oferecer lembretes automáticos para os técnicos sobre tickets que precisam ser atendidos dentro de um prazo específico.

Integração com Sistemas de Monitoramento e Diagnóstico Remoto

- Integrar o sistema de tickets com ferramentas de monitoramento e diagnóstico remoto de hardware e software. Isso permite aos técnicos obter informações detalhadas sobre o estado dos equipamentos ou sistemas afetados pelo ticket.

Metodologia

A metodologia a ser adotada para o desenvolvimento da plataforma TicketHelper será o padrão de arquitetura MVC (Model-View-Controller), implementado em C#.

O padrão MVC separa a aplicação em três componentes principais:

Model: Representa a lógica de negócios e a manipulação de dados. Inclui as classes que representam os dados e as regras de negócios associadas.
View: Representa a interface do usuário. Inclui as páginas e os elementos visuais que exibem os dados para o usuário.
Controller: Controla a comunicação entre o Model e a View. Processa as entradas do usuário, manipula os dados no Model e atualiza a View.

Passos Gerais para o desenvolvimento do projeto:

Definição dos Modelos - Desenvolver as classes que representam os dados e as regras de negócios (e.g., Ticket, HardwareTicket, SoftwareTicket, User).

Criação da Database – Criar as tabelas que são necessárias para suportar o sistema de gestão de tickets, incluindo tabelas para guardar informações de tickets de hardware e software, bem como informações de utilizadores e níveis de acesso.

Desenvolvimento das Views - Criar as páginas e elementos visuais que permitirão aos usuários interagir com a aplicação (e.g., criação e visualização de tickets).

Implementação dos Controllers - Implementar os controladores que irão processar as entradas dos usuários, interagir com os modelos e atualizar as views adequadas.

Integração e Testes - Integrar todos os componentes e realizar testes para assegurar que a aplicação funciona conforme o esperado.

Documentação - Documentar o código e funcionalidades.

Etapas de Desenvolvimento

Criação das Classes

A primeira etapa no desenvolvimento do sistema foi a criação das classes base, que representam os objetos principais utilizados no sistema.

```
20 references
public class Ticket
{
    private Guid _id;

    private DateTime _dataCriacao;

    private DateTime? _dataAlteracao;

    private string _userCriador;

    private string _userAlteracao;

    //Estado dos tickets: porAtender, emAtendimento, atendido.
    private string _status;

    //Estado do atendimento: aberto, resolvido, naoResolvido.
    private string _statusAtendimento;

    private string _tipo;
```

Imagem 1 - Classe Ticket

A Classe `Ticket` é a classe base do projeto que contem os campos comuns aos dois tipos de ticket

```
15 references
public class HardwareTicket : Ticket
{
    private string _equipamento;
    private string _avaria;
    private string _descReparacao;
    private string _pecas;
```

Imagem 2 - Classe HardwareTicket

A classe `HardwareTicket` é uma especialização da classe base `Ticket`. Isso significa que `HardwareTicket` herda todas as propriedades e métodos da classe `Ticket`, mas também adiciona ou modifica funcionalidades específicas para problemas de hardware.

```
15 references
public class SoftwareTicket:Ticket
{
    private string _software;
    private string _necessidade;
    private string _descIntervencao;
```

Imagem 3 - Classe SoftwareTicket

A classe `SoftwareTicket` é uma especialização da classe base `Ticket`. Isso significa que `Softwareicket` herda todas as propriedades e métodos da classe `Ticket`, mas também adiciona ou modifica funcionalidades específicas para problemas de software.

Criação das Tabelas na base de dados

TABELA USERS

```
USE [GestaoTickets]
GO

/** Object:  Table [dbo].[t_Users]    Script Date: 26/06/2024 14:20:29 **/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[t_Users](
	[id] [varchar](50) NOT NULL,
	[username] [varchar](200) NOT NULL,
	[password] [varchar](200) NOT NULL,
	[email] [varchar](200) NOT NULL,
	[nivelAcesso] [tinyint] NOT NULL,
 CONSTRAINT [PK_t_Users] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY],
UNIQUE NONCLUSTERED
(
```

```
	[username] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY],
UNIQUE NONCLUSTERED
(
	[username] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO
```

Tabela Ticket

```
USE [GestaoTickets]
GO

/** Object:  Table [dbo].[t_Ticket]    Script Date: 26/06/2024 14:20:23 **/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO
```

```
CREATE TABLE [dbo].[t_Ticket](
	[id] [varchar](50) NOT NULL CONSTRAINT [DF_t_Ticket_id]  DEFAULT (newid()),
	[dataCriacao] [datetime] NOT NULL CONSTRAINT [DF_Table_1_dtCriacao] DEFAULT (getdate()),
	[dataAlteracao] [datetime] NOT NULL,
	[userCriador] [varchar](100) NOT NULL,
	[userAlteracao] [varchar](100) NULL,
	[status] [varchar](50) NOT NULL,
	[statusAtendimento] [varchar](50) NOT NULL,
	[tipo] [varchar](50) NOT NULL,
 CONSTRAINT [PK_t_Ticket] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO
```

Tabela Hardware Ticket

```
USE [GestaoTickets]
GO

/** Object:  Table [dbo].[t_HardwareTicket]    Script Date: 26/06/2024 14:20:03 **/
SET ANSI_NULLS ON
GO
```

```
SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[t_HardwareTicket](
	[id] [varchar](50) NOT NULL,
	[equipamento] [varchar](200) NOT NULL,
	[avaria] [varchar](200) NOT NULL,
	[descReparacao] [varchar](200) NOT NULL,
	[pecas] [varchar](200) NOT NULL,
 CONSTRAINT [PK_t_HardwareTicket] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO
```

Tabela SoftwareTIcket

```
USE [GestaoTickets]
GO

/** Object:  Table [dbo].[t_SoftwareTicket]    Script Date: 26/06/2024 14:20:16 **/
```

```
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[t_SoftwareTicket](
	[id] [varchar](50) NOT NULL,
	[software] [varchar](200) NOT NULL,
	[necessidade] [varchar](200) NOT NULL,
	[descIntervencao] [varchar](200) NOT NULL,
 CONSTRAINT [PK_t_SoftwareTicket] PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO
```

Implementação do *TicketHelper.cs*

TicketHelper é uma classe de apoio que esta encarregue de realizar as querys á base de dados. Por exemplo "*INSERT*", "*SELECT*" e "*UPDATE*". O TicketHelper é o intermediário entre o Controller e a Base de dados.

```
1 reference
public List<Ticket> list(string tipo)
{
    DataTable tickets = new DataTable();
    List<Ticket> outList = new List<Ticket>();
    SqlDataAdapter telefone = new SqlDataAdapter();
    SqlCommand comando = new SqlCommand();
    SqlConnection conexao = new SqlConnection(base.ConnectionDB);


    if (tipo == "Todos")
    {
        comando.CommandText = "SELECT * FROM t_Ticket ORDER BY [dataCriacao] ASC";
    }
    else
    {
        comando.CommandText = "SELECT * FROM t_Ticket WHERE tipo = @tipo ORDER BY [dataCriacao] ASC";
        comando.Parameters.AddWithValue("@tipo", tipo);
    }

    comando.Connection = conexao;
    telefone.SelectCommand = comando;
    telefone.Fill(tickets);

    conexao.Dispose();
```

Imagem 4 - Função Listar

Primeira Parte da Listagem é criar as respetivas variáveis para realizar a query a base de dados, neste caso primeiro vamos buscar todos os tickets na tabela Ticket.

```
foreach (DataRow linhaticket in tickets.Rows)
{
    string tipoTicket = linhaticket["tipo"].ToString();
    Ticket ticket;

    if (tipoTicket == "Hardware")
    {
        var hardwareTicket = new HardwareTicket();
        PreencherHardwareTicket(hardwareTicket, linhaticket["id"].ToString());
        ticket = hardwareTicket;
    }
    else if (tipoTicket == "Software")
    {
        var softwareTicket = new SoftwareTicket();
        PreencherSoftwareTicket(softwareTicket, linhaticket["id"].ToString());
        ticket = softwareTicket;
    }
    else
    {
        ticket = new Ticket();
    }


    // Preencher propriedades comuns
    ticket.Id = Guid.Parse(linhaticket["id"].ToString());
    ticket.DataCriacao = Convert.ToDateTime(linhaticket["dataCriacao"]);
    ticket.DataAlteracao = Convert.ToDateTime(linhaticket["dataAlteracao"]);
    ticket.UserCriador = linhaticket["userCriador"].ToString();
    ticket.UserAlteracao = linhaticket["userAlteracao"].ToString();
    ticket.Status = linhaticket["status"].ToString();
    ticket.StatusAtendimento = linhaticket["statusAtendimento"].ToString();
    ticket.Tipo = linhaticket["tipo"].ToString();

    outList.Add(ticket);
}

return outList;
```

Imagem 5 - Função Listar

Após ir buscar todos os dados na tabela Ticket, cada registo tem o seu campo “tipo” e com o tipo fazemos a respetiva separação para os objetos `HardwareTicket` ou `SoftwareTicket`. Antes de retornar a lista preenchemos os campos comuns aos dois tipos de ticket.

```
3 references
private void PreencherHardwareTicket(HardwareTicket hardwareTicket, string id)
{
    SqlConnection conexao = new SqlConnection(base.ConnectionDB);
    SqlCommand comando = new SqlCommand();
    DataTable tickets = new DataTable();

    comando.CommandType = CommandType.Text;
    comando.CommandText = "SELECT * FROM t_HardwareTicket WHERE id = @id";
    comando.Parameters.AddWithValue("@id", id.ToString());

    comando.Connection = conexao;
    SqlDataAdapter telefone = new SqlDataAdapter(comando);
    telefone.Fill(tickets);

    if (tickets.Rows.Count > 0)
    {
        DataRow linhaticket = tickets.Rows[0];
        hardwareTicket.Equipamento = linhaticket["equipamento"].ToString();
        hardwareTicket.Avaria = linhaticket["avaria"].ToString();
        hardwareTicket.DescReparacao = linhaticket["descReparacao"].ToString();
        hardwareTicket.Pecas = linhaticket["pecas"].ToString();
    }

    conexao.Close();
    conexao.Dispose();
}

3 references
private void PreencherSoftwareTicket(SoftwareTicket softwareTicket, string id)
{
    SqlConnection conexao = new SqlConnection(base.ConnectionDB);
    SqlCommand comando = new SqlCommand();
    DataTable tickets = new DataTable();

    comando.CommandType = CommandType.Text;
    comando.CommandText = "SELECT * FROM t_SoftwareTicket WHERE id = @id";
    comando.Parameters.AddWithValue("@id", id);

    comando.Connection = conexao;
    SqlDataAdapter telefone = new SqlDataAdapter(comando);
    telefone.Fill(tickets);

    if (tickets.Rows.Count > 0)
    {
        DataRow linhaticket = tickets.Rows[0];
        softwareTicket.Software = linhaticket["software"].ToString();
        softwareTicket.Necessidade = linhaticket["necessidade"].ToString();
        softwareTicket.DescIntervencao = linhaticket["descIntervencao"].ToString();
    }

    conexao.Close();
    conexao.Dispose();
}
```

Imagem 6 - Função Preencher Hardware/Software Ticket

A função PreencherSoftwareTicket / HardwareTicket é utilizada para fazer a pesquisa na base de dados nas tabelas HardwareTicket e SoftwareTicket para obtermos os campos específicos de cada tipo de ticket e guardar os respetivos campos na lista de Tickets no programa.

```
1 reference
public void Save(Ticket ticket)
{
    using (SqlConnection conexao = new SqlConnection(base.ConnectionDB))
    {
        conexao.Open();
        SqlTransaction transaction = conexao.BeginTransaction();

        try
        {
            // Verificar se o ticket já existe na tabela t_Ticket
            bool ticketExists = false;
            SqlCommand comandoVerificacao = new SqlCommand
            {
                Connection = conexao,
                Transaction = transaction,
                CommandType = CommandType.Text,
                CommandText = "SELECT COUNT(*) FROM t_Ticket WHERE id = @id"
            };
            comandoVerificacao.Parameters.AddWithValue("@id", ticket.Id);

            int count = (int)comandoVerificacao.ExecuteScalar();
            if (count > 0)
            {
                ticketExists = true;
            }
```

Imagem 7 - Função Save (Criar ou Editar)

A função é responsável por receber um objeto do tipo Ticket e partir dai ela vai verificar se já existe na base de dados e se não existir cria um novo ticket, caso já exista aquele ticket ela vai apenas atualizar o registo e não fazer a criação de um novo. A função faz a pesquisa do ticket na base de dados com o campo ID.

```
if (!ticketExists)
{
    // Inserir o ticket na tabela t_Ticket
    SqlCommand comandoInsertTicket = new SqlCommand
    {
        Connection = conexao,
        Transaction = transaction,
        CommandType = CommandType.Text,
        CommandText = "INSERT INTO t_Ticket (id, dataCriacao, dataAlteracao, userCriador, userAlteracao, status, statusAtendimento, tipo) " +
                      "VALUES (@id, @dataCriacao, @dataAlteracao, @userCriador, @userAlteracao, @status, @statusAtendimento, @tipo)"
    };

    comandoInsertTicket.Parameters.AddWithValue("@id", ticket.Id);
    comandoInsertTicket.Parameters.AddWithValue("@dataCriacao", ticket.DataCriacao);
    comandoInsertTicket.Parameters.AddWithValue("@dataAlteracao", ticket.DataCriacao);
    comandoInsertTicket.Parameters.AddWithValue("@userCriador", ticket.UserCriador);
    comandoInsertTicket.Parameters.AddWithValue("@userAlteracao", ticket.UserAlteracao);
    comandoInsertTicket.Parameters.AddWithValue("@status", ticket.Status);
    comandoInsertTicket.Parameters.AddWithValue("@statusAtendimento", ticket.StatusAtendimento);
    comandoInsertTicket.Parameters.AddWithValue("@tipo", ticket.Tipo);

    comandoInsertTicket.ExecuteNonQuery();

    // Inserir os dados adicionais conforme o tipo de ticket
    if (ticket is HardwareTicket hardwareTicket)
    {
        SqlCommand comandoInsertHardware = new SqlCommand
        {
            Connection = conexao,
            Transaction = transaction,
            CommandType = CommandType.Text,
            CommandText = "INSERT INTO t_HardwareTicket (id, equipamento, avaria, descReparacao, pecas) " +
                          "VALUES (@id, @equipamento, @avaria, @descReparacao, @pecas)"
        };

        comandoInsertHardware.Parameters.AddWithValue("@id", hardwareTicket.Id);
        comandoInsertHardware.Parameters.AddWithValue("@equipamento", hardwareTicket.Equipamento);
        comandoInsertHardware.Parameters.AddWithValue("@avaria", hardwareTicket.Avaria);
        comandoInsertHardware.Parameters.AddWithValue("@descReparacao", hardwareTicket.DescReparacao);
        comandoInsertHardware.Parameters.AddWithValue("@pecas", hardwareTicket.Pecas);

        comandoInsertHardware.ExecuteNonQuery();
    }
    else if (ticket is SoftwareTicket softwareTicket)
    {
        SqlCommand comandoInsertSoftware = new SqlCommand
        {
            Connection = conexao,
            Transaction = transaction,
            CommandType = CommandType.Text,
            CommandText = "INSERT INTO t_SoftwareTicket (id, software, necessidade, descIntervencao) " +
                          "VALUES (@id, @software, @necessidade, @descIntervencao)"
        };

        comandoInsertSoftware.Parameters.AddWithValue("@id", softwareTicket.Id);
        comandoInsertSoftware.Parameters.AddWithValue("@software", softwareTicket.Software);
        comandoInsertSoftware.Parameters.AddWithValue("@necessidade", softwareTicket.Necessidade);
        comandoInsertSoftware.Parameters.AddWithValue("@descIntervencao", softwareTicket.DescIntervencao);

        comandoInsertSoftware.ExecuteNonQuery();
    }
}
```

Imagem 8 - Função Save (Criar Ticket)

Se o ticket não existir vamos inserir na tabela Ticket os respetivos campos e de seguida vamos verificar qual foi o objeto enviado para a função Save. Se for `HardwareTicket`

então vamos a tabela HardwareTIcket inserir o resto dos campos, se não vamos a tabela SoftwareTicket.

Para atualizar um registo na base de dados o processo é o mesmo que o Criar a única diferença é na Query á base de dados trocamos o “INSERT” por “UPDATE”.

Outra função implementada no TicketHelper é a Função get e esta função tem como objetivo de ir buscar apenas 1 ticket a base de dados.

Implementação dos Controllers

TicketController

```
public IActionResult Criar(Ticket ticket)
{

    if (_conta.NivelAcesso > 0)
    {
        if (ticket == null) return BadRequest();

        // Identificar o tipo de ticket e criar a instância apropriada
        if (Request.Form["tipoticket"] == "Hardware")
        {
            HardwareTicket hardwareTicket = new HardwareTicket
            {
                Id = ticket.Id,
                DataCriacao = ticket.DataCriacao,
                DataAlteracao = ticket.DataAlteracao,
                UserCriador = _conta.Nome,
                UserAlteracao = ticket.UserAlteracao,
                Status = ticket.Status,
                StatusAtendimento = ticket.StatusAtendimento,
                Tipo = Request.Form["tipoticket"],
                // Campos especificos do HardwareTicket
                Equipamento = Request.Form["equipamento"],
                Avaria = Request.Form["avaria"],
            };

            SaveTicket(hardwareTicket);
        }
        else if (Request.Form["tipoticket"] == "Software")
        {
            SoftwareTicket softwareTicket = new SoftwareTicket
            {
                Id = ticket.Id,
                DataCriacao = ticket.DataCriacao,
                DataAlteracao = ticket.DataAlteracao,
                UserCriador = _conta.Nome,
                UserAlteracao = ticket.UserAlteracao,
                Status = ticket.Status,
                StatusAtendimento = ticket.StatusAtendimento,
                Tipo = Request.Form["tipoticket"],
                // Campos especificos do SoftwareTicket
                Software = Request.Form["software"],
                Necessidade = Request.Form["necessidade"],

            };

            SaveTicket(softwareTicket);
        }
        else
        {
            return BadRequest("Tipo de ticket inválido.");
        }
    }

    return RedirectToAction("Listar", "Ticket", new { tipo = "Todos" });
```

Imagem 9 - TicketController ([HttpPOST] Criar)

[HttpPost] Criar recebe informação do formulário na View Criare cria um novo objeto de acordo com o tipo. Após fazer essa criação é chamada a função SaveTicket(). A função SaveTicket cria um novo helper e chama a função Save do Helper.

```
4 references
private void SaveTicket(Ticket ticket)
{
    TicketsHelper th = new TicketsHelper();
    th.Save(ticket);
}



[HttpGet]
0 references
public IActionResult Detalhes(string id)
{
    if (_conta.NivelAcesso > 0)
    {
        TicketsHelper th = new TicketsHelper();
        Ticket? ticket = th.get(id); // Obtém o ticket pelo ID

        if (ticket == null)
        {
            return RedirectToAction("Listar", "Ticket", new { tipo = "Todos" });
        }

        return View(ticket);


    }

    return RedirectToAction("Login", "Conta");
}


[HttpGet]
0 references
public IActionResult Voltar()
{
    if (_conta.NivelAcesso > 0)
    {

        return RedirectToAction("Listar", "Ticket", new { tipo = "Todos" });


    }

    return RedirectToAction("Login", "Conta");
}
```

Imagem 10 - Mais Funções do Controller

Função Detalhes recebe um id e vai enviar esse id para o helper função get(), que depois retorna o ticket completo com todas as informações.

Função Voltar é uma função para voltar sempre para a pagina principal (Listar).

Implementação das Views

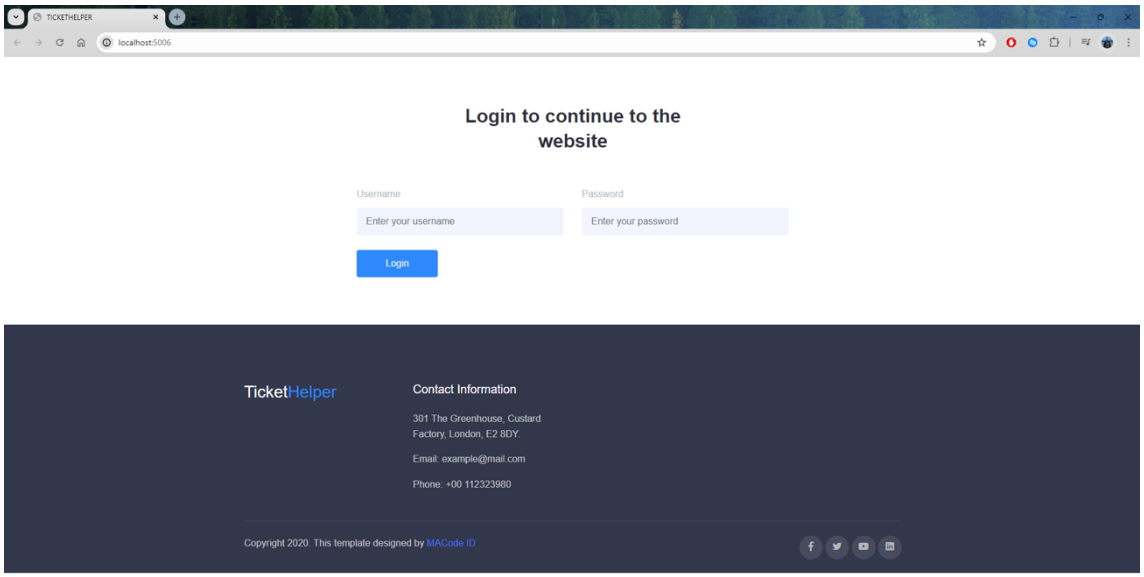


Imagem 11 - Pagina Login

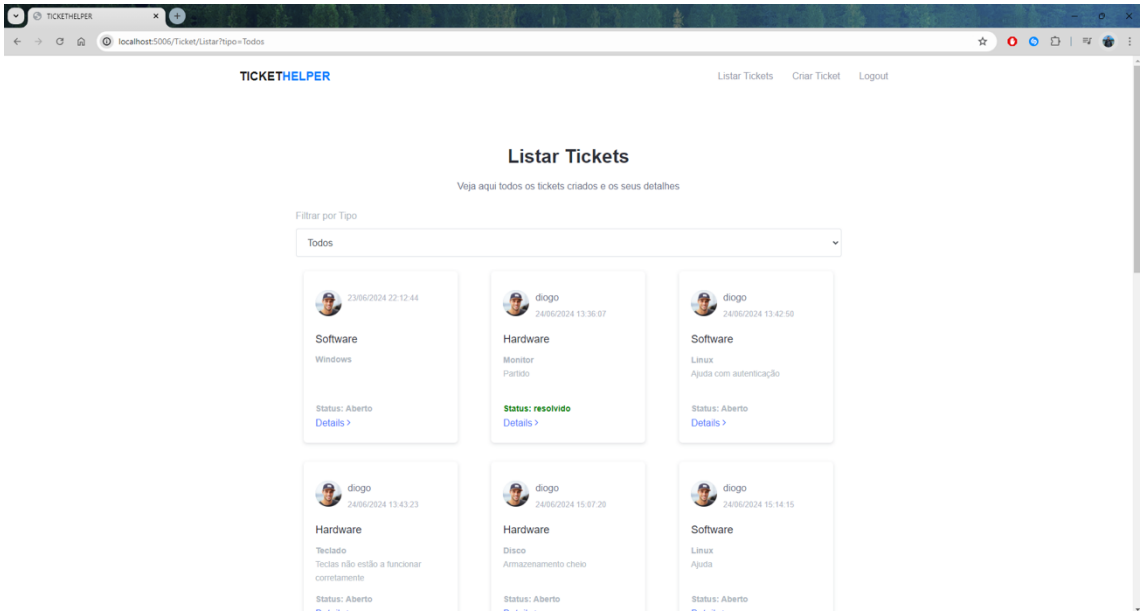


Imagem 12 - Pagina Listar

Imagem 13 - Caixa de 1 ticket (Close up)

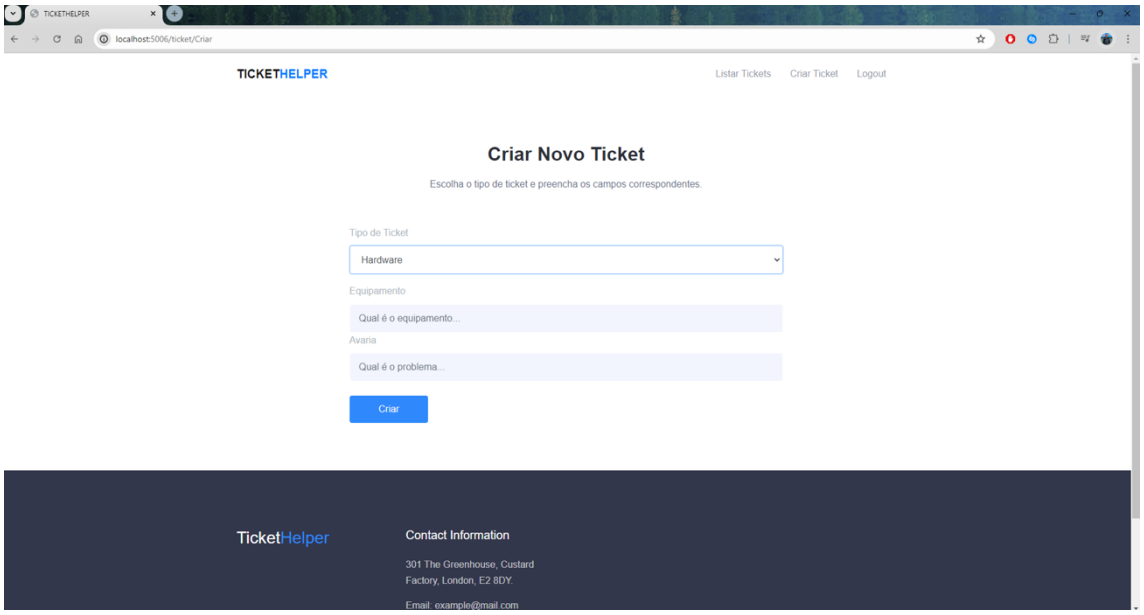


Imagem 14 - Pagina Criar

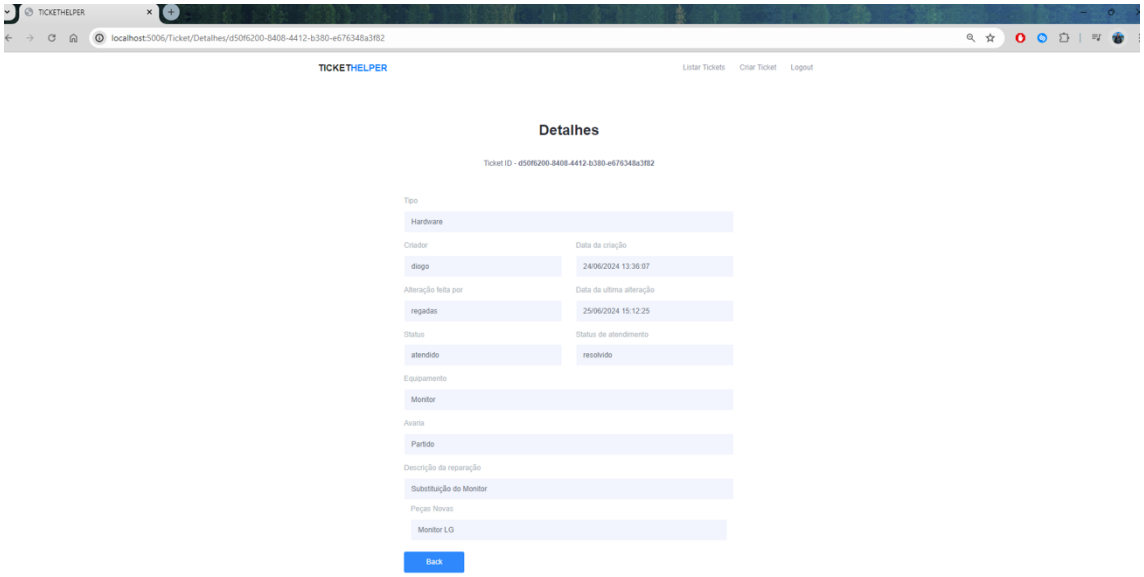


Imagem 15 - Pagina Detalhes

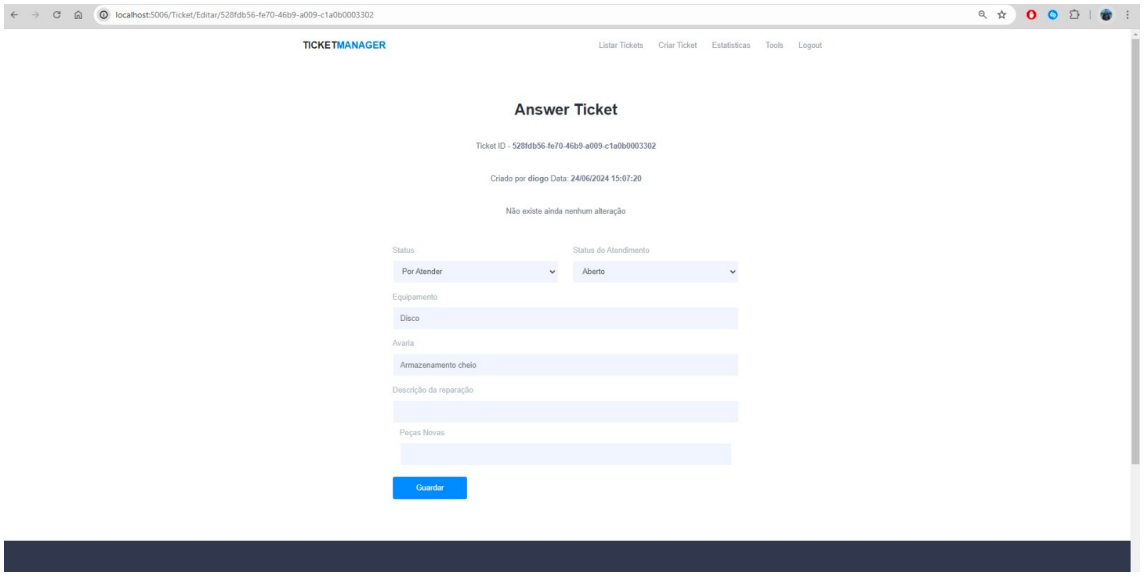
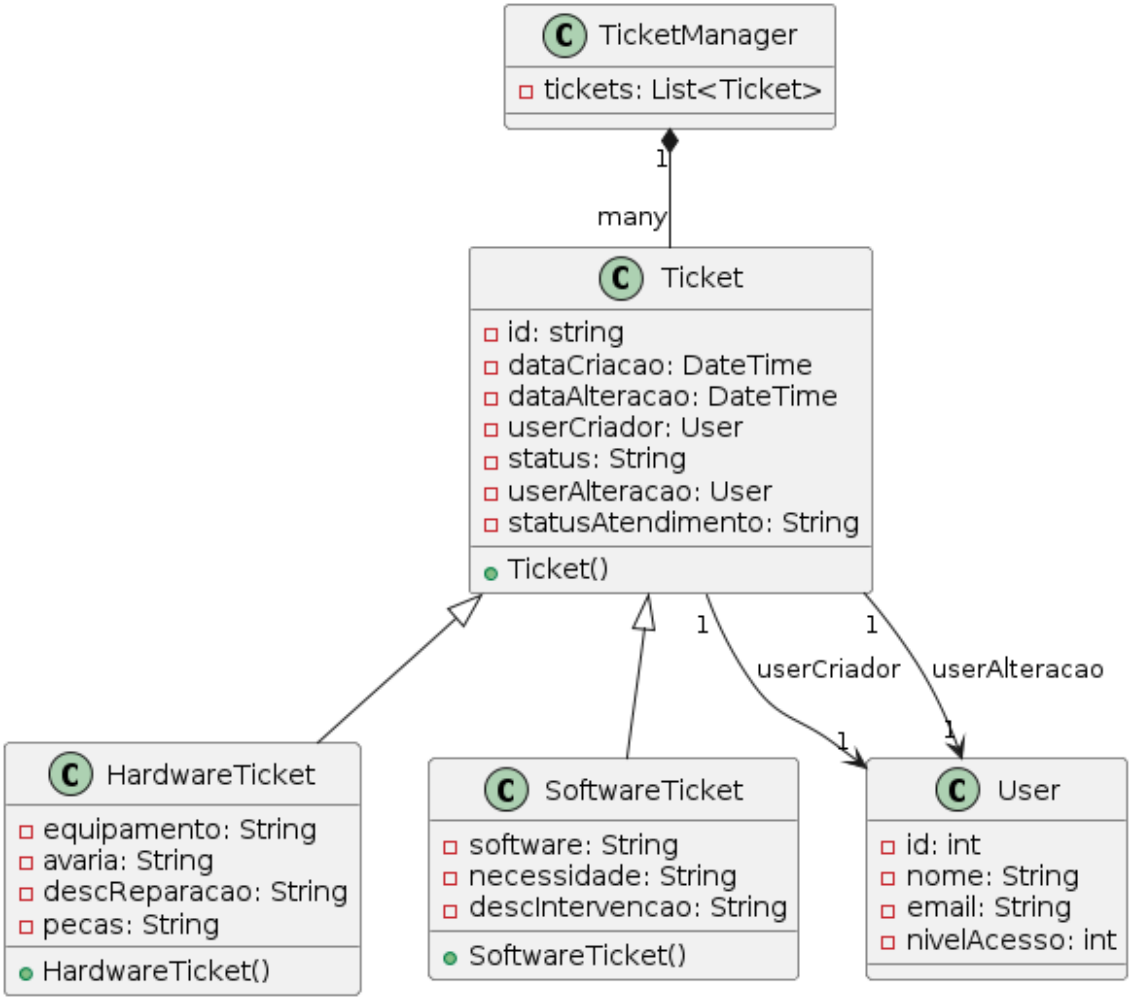
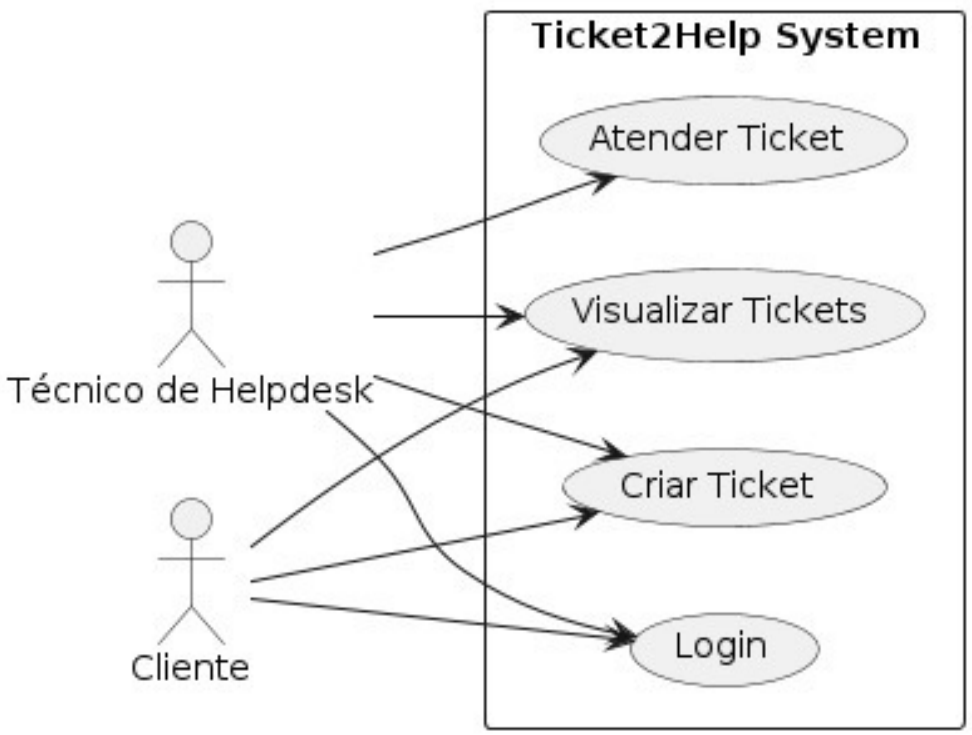


Imagem 16 - Pagina Editar

Diagrama de Classes



Use-Case


Ticket2Help System
Atender Ticket
Visualizar Tickets
Criar Ticket
Login
Técnico de Helpdesk
Cliente

GITHUB

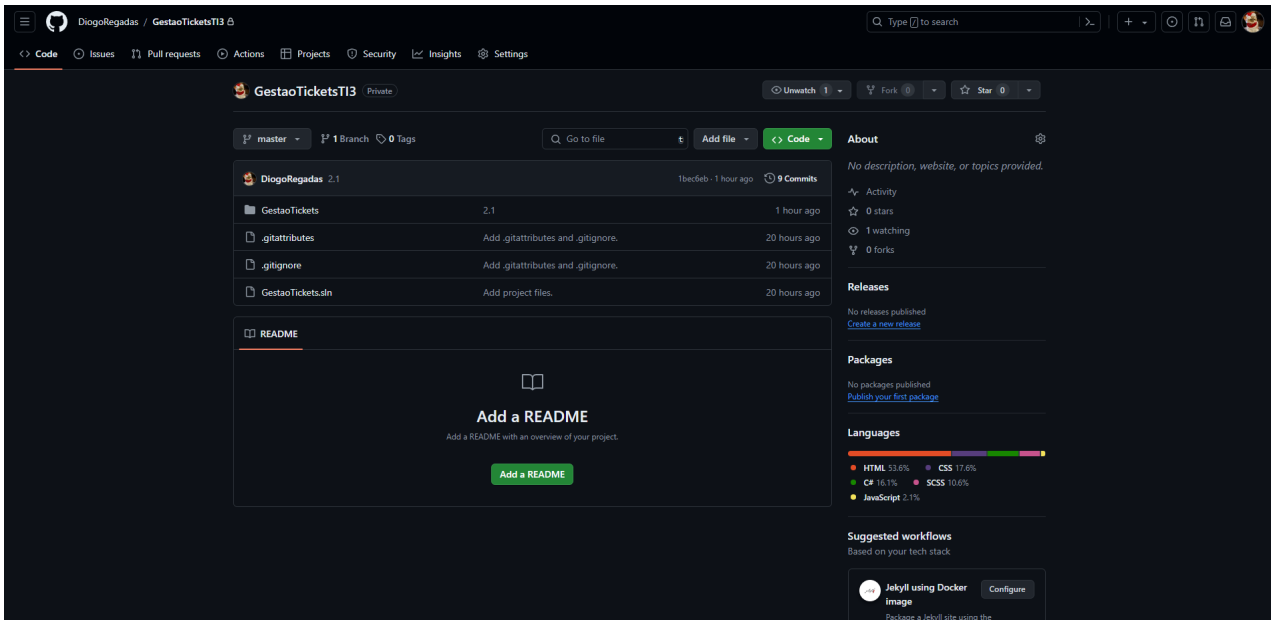
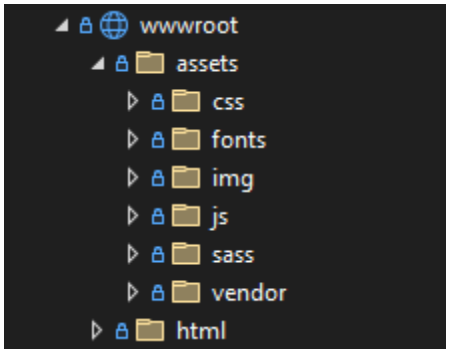
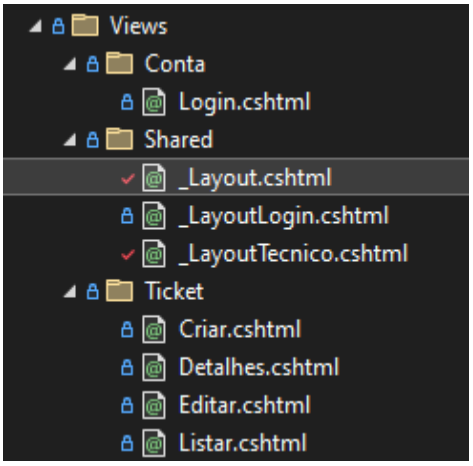
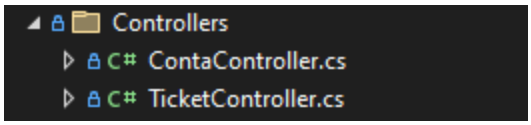


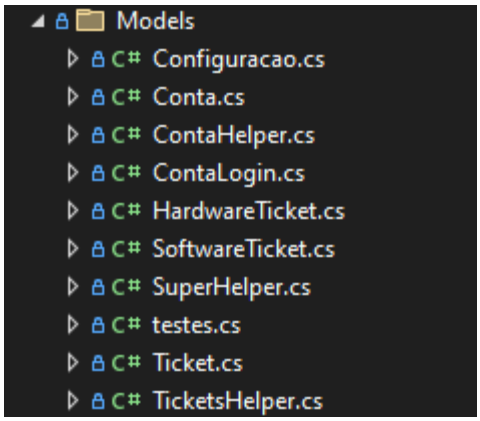
Imagem 17 - GITHUB repository

Vista Geral dos ficheiros do Projeto






Controllers
ContaController.cs
TicketController.cs


Models
Configuracao.cs
Conta.cs
ContaHelper.cs
ContaLogin.cs
HardwareTicket.cs
SoftwareTicket.cs
SuperHelper.cs
testes.cs
Ticket.cs
TicketsHelper.cs