SangES - Entrega Final

Programação Orientada a Objetos II

Grupo 3: Érico Pedrosa e Pedro Kerkoff.

Professora: Gabriela Martins de Jesus.

Objetivo do Projeto

O projeto SangES foi desenvolvido com o intuito de solucionar problemas recorrentes enfrentados pelos Hemocentros e pelos doadores de sangue. Entre as principais dificuldades estão a longa espera devido a processos burocráticos, como papeladas, formulários e protocolos necessários para a doação.

Funcionalidades Principais

- **Unificação de Informações**: Centralização de todas as informações relevantes tanto para os doadores quanto para os Hemocentros, facilitando o acesso e a gestão dos dados.
- **Agendamento de Doações**: Permite que os doadores agendem suas doações online, reduzindo o tempo de espera e otimizando o fluxo de atendimento nos Hemocentros.
- **Formulários Digitais**: Digitalização dos formulários necessários, que podem ser preenchidos previamente pelos doadores, agilizando o processo na hora da doação.
- **Histórico de Doações**: Acesso fácil ao histórico de doações do usuário, incluindo datas, tipos de sangue doados, e locais de doação.
- **Notificações e Lembretes**: Sistema de notificações para lembrar os doadores das próximas datas de doação e de seus agendamentos.
- **Educação e Conscientização**: Fornecimento de informações educativas sobre a importância da doação de sangue, critérios de elegibilidade, e benefícios para os doadores.

Benefícios

- **Para os Doadores**:
    - Redução do tempo de espera nos Hemocentros.
    - Experiência mais fluida e menos burocrática.
    - Facilidade de acesso a informações pessoais e histórico de doações.
    - Maior engajamento e frequência nas doações através de notificações e lembretes.
- **Para os Hemocentros**:
    - Melhor gestão e organização dos processos internos.
    - Redução da carga de trabalho administrativa com a digitalização de formulários.
    - Capacidade de planejar e gerenciar melhor a demanda de doadores.

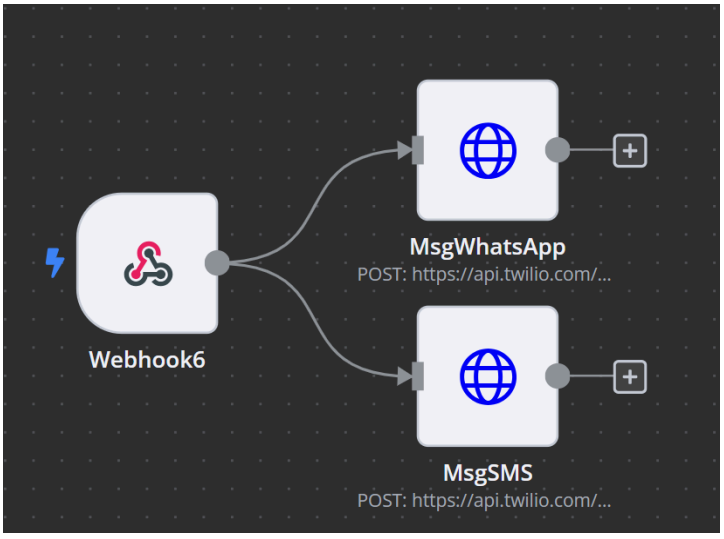
- Acesso a dados e estatísticas importantes para campanhas de doação e planejamento estratégico.

Tecnologias Utilizadas

- **Back-end**: Desenvolvemos o back-end do projeto utilizando C# e Entity Framework garantindo uma estrutura sólida e eficiente para o gerenciamento de dados e lógica de negócios.
- **Front-end**: O front-end foi construído com Bootstrap, CSS e HTML, proporcionando uma interface de usuário responsiva e amigável, com uma experiência visual moderna e consistente.
- **Integrações e Automações**: Para o envio de SMS e mensagens no WhatsApp, utilizamos o n8n em conjunto com a API do Twilio. Essa combinação permite automações poderosas e eficientes, simplificando o processo de comunicação com os doadores.
- **Banco de Dados**: O banco de dados foi implementado utilizando o Microsoft SQL Server, garantindo robustez, segurança e alta performance no armazenamento e gerenciamento das informações.

Implementação Técnica

- O envio de mensagens por SMS e WhatsApp foi implementado utilizando a API do Twilio. Para isso, desenvolvemos uma solução em C# que envia as informações necessárias para um Webhook, criado no n8n. Fizemos duas solicitações HTTP POST na API do Twilio. Optamos por usar o n8n, uma ferramenta poderosa para integrações e automações, em vez de implementar essa funcionalidade diretamente no nosso serviço em C# ou outra linguagem. A escolha pelo n8n se deu pela sua capacidade de facilitar a interpretação do processo, tornando futuras manutenções mais simples e eficientes.



```
1 reference | Niccharff, 1 day ago | 1 author, 1 change
public async void APIWeebHookMSGWPP(AgendamentoDTO model)
{
    var tipoSanguineo = await BuscarTipoSanguineoDoUsuario(model.AuthenticationTypeUser);
    string url = "https://webhook.app.hubhero.com.br/webhook/48ec13d5-039b-4df0-9c9e-b20c40c76806";

    var data = new
    {
        nome = BuscarNomeUsuarioPorId(model.AuthenticationTypeUser),
        nomeHemocentro = _context.Hemocentros.Where(x => x.Id == model.IdHemocentro).Select(s => s.NomeHemocentro).FirstOrDefault(),
        dia = model.data,
        hora = model.hora,
        tipoSanguineo = tipoSanguineo.TipoSanguineo + (tipoSanguineo.IsPositivo ? "+" : "-"),
        telefone = await BuscarTelefoneUsuarioPorId(model.AuthenticationTypeUser)

    };

    var json = Newtonsoft.Json.JsonConvert.SerializeObject(data);
    var content = new StringContent(json, Encoding.UTF8, "application/json");

    //Criar uma instância de HttpClient
    using (HttpClient client = new HttpClient())
    {
        try
        {
            //Enviar a requisição POST
            HttpResponseMessage response = await client.PostAsync(url, content);

            //Ler a resposta da requisição
            string responseBody = await response.Content.ReadAsStringAsync();

            //Verificar se a requisição foi bem-sucedida
            if (response.IsSuccessStatusCode)
            {
                Console.WriteLine("Requisição POST enviada com sucesso!");
                Console.WriteLine("Resposta do servidor: " + responseBody);
            }
            else
            {
                Console.WriteLine("Erro ao enviar a requisição POST.");
                Console.WriteLine("Status Code: " + response.StatusCode);
                Console.WriteLine("Resposta do servidor: " + responseBody);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("Ocorreu um erro ao enviar a requisição POST:");
            Console.WriteLine(ex.Message);
        }

    }
```

As linhas estão comentadas para facilitar o entendimento, mas a requisição é bem simples, basicamente em data, passamos as informações que vão ser consumidas pela API, sendo o nome(email) da pessoa, o nome do hemocentro que ela escolheu, o dia, a hora, o tipo sanguíneo da pessoa, e seu respectivo celular, e depois fazemos uma requisição e uma validação com um log, para podermos acompanhar se acontecer algo.

Link n8n: [n8n.io - a powerful workflow automation tool]

Link da documentação da API do Twilio: [Messaging | Twilio]

- **Conexão Banco com EF (Entity Framework)**
    - No nosso projeto fizemos o BD com CodeFirst, onde consiste na engenharia reversa com o EF, informando PK e FK com a Data Annotations do C# para ele, ele consegue fazer todo o resto, a conexão com o bd é simples, criamos a migration com comando no Packet Manager add-migration {nome desejado da migration}, logo em seguida update-database, vai criar um snapshot do banco, com o script SQL, e já é mandando para a string de conexão com o banco, na program.cs faz a injeção da string de conexão, que está salva no appsettings.json, que chamamos de DefaultConnection:

      "ConnectionStrings": {

      "DefaultConnection": "Data Source={endereço de conexão com o banco};Initial Catalog={nome que vai ser Criado no banco};Integrated Security=True;Pooling=False;Encrypt=True;Trust Server Certificate=True" }

```
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection")
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(connectionString));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
```

```
builder.Services.AddDefaultIdentity<User>(options => options.SignIn.RequireConfirmedAccount = false)
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

- **MVC**
    - Por praticidade usamos o mvc simples no nosso projeto, onde temos a view, e controller para pegar informações dos usuários e manipularmos como quisermos, abaixo vou deixar um exemplos de como fazemos esse processo, e como exibimos as informações do backend para view.

```
<h3 class="card-title text-danger text-center">Informações do Hemocentro</h3>
@foreach (var iten in Model)
{
    <div class="card mt-4 bg-white position-relative" style="height: 300px;">
        <h5 class="card-title text-danger text-center">@iten.Hemocentro.NomeHemocentro</h5>
        <div class="card-body d-flex">
            <img style="width: 25%;" src="https://s2.glbimg.com/h-Y17kR85vnUQLnNX9CtqGRK_-M=/0x0:1280x592/984x0/smart/filters:strip_icc()/i

            <div class="ml-4 flex-grow-1 align-items-start" style="padding-left: 15px;">
                <div class="row">
                    <div class="col-6">
                        <p class="card-text text-muted">Referência: @iten.Hemocentro.Referencia</p>
                        <p class="card-text text-muted">CEP: @iten.Hemocentro.CEP</p>
                        <p class="card-text text-muted">Telefone: @iten.Hemocentro.Telefone</p>
                        <p class="card-text text-muted">Email: @iten.Hemocentro.Email</p>
                        <p class="card-text text-muted">Endereço: @iten.Hemocentro.Endereco</p>
                    </div>
                    <div class="col-6">
                        <p class="card-text text-muted" style="margin-bottom: 0;">Sangues em falta:</p>
                        <ul class="list-group list-group-horizontal">
                            @foreach (var sangueEmFalta in iten.TipoSanguineoFaltando)
                            {
                                <li class="list-group-item" style="font-size: 15px;"><small>@sangueEmFalta</small></li>
                            }
                        </ul>
                    </div>
                </div>
            </div>
        </div>
        <div class="d-flex justify-content-end position-absolute" style="bottom: 10px; right: 10px;">
            @if(User.Identity.IsAuthenticated)
            {
                <a href="@Url.Action("Agendamento","Hemocentro",new {Id = iten.Hemocentro.Id})">
                    <button class="btn btn1 text-white" style="background-color: #C5141A;">Agendar Doação</button>
                </a>
            }
            else{
                <a asp-area="Identity" asp-page="/Account/Login">
                    <button class="btn btn1 text-white" style="background-color: #C5141A;">Logar para Agendar a doação</button>
                </a>
            }
            <a href="@Url.Action("BancoSangue","Hemocentro",new {Id = iten.Hemocentro.Id})" style="margin-left: 20px">
                <button class="btn btn2 text-white" style="background-color: #979797;">Ver Estoque de Sangue</button>
            </a>
```

A Model definimos no começo da página e podemos chamar ela como Model, então para cada item na model, no contexto seria cada informação dos hemocentros, então podemos chamar as informações dos nossos hemocentros passando a variável definida no foreach, em seguida chamamos como exemplo o nome do hemocentro usando: @iten.Hemocentro.NomeHemocentro

```
@using DoacaoSangueMVC.Entities;
@model IList<HemocentroDTO>
```

```
6 references | Nicchorff, 2 days ago | 1 author, 1 change
public class HemocentroDTO
{
    9 references | Nicchorff, 2 days ago | 1 author, 1 change
    public Hemocentro Hemocentro { get; set; }
    2 references | Nicchorff, 2 days ago | 1 author, 1 change
    public IList<string> TipoSanguineoFaltando { get; set; }
}
```

Usamos o ‘HemocentroDTO’ para guardar as informações da view para a controller, sendo assim uma classe Hemocentro, e uma lista com os Tipos de Sangues que estão faltando, elas passam pela controller.

```
0 references | Nicchorff, 2 days ago | 1 author, 3 changes
public async Task<IActionResult> Index()
{
    var viewModel = _workService.MapeamentoParaHemocentroDTOS();
    return View(await viewModel);
}
```

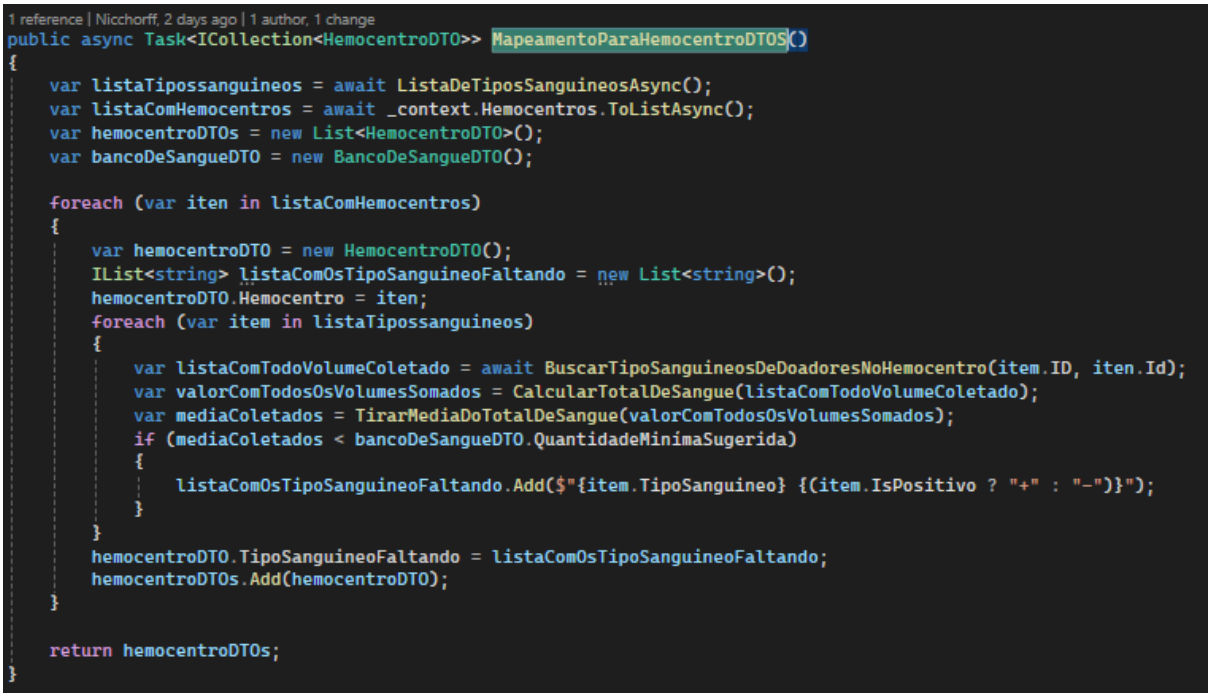
O WorkService seria nossa Model, onde contém todos os nossos métodos necessários, fazemos uma injeção de dependência na controller:

```
1 reference | Nicchorff, 1 day ago | 1 author, 12 changes
public class HemocentroController : Controller
{
    private readonly ApplicationDbContext _context;
    private readonly HemocentroWorkService _workService;

    0 references | Nicchorff, 2 days ago | 1 author, 1 change
    public HemocentroController(ApplicationDbContext context, HemocentroWorkService workService)
    {
        _context = context;
        _workService = workService;
    }
```

Na program a injeção foi feita desta maneira:

```
builder.Services.AddControllersWithViews();
builder.Services.AddTransient<HemocentroWorkService>();
```



```
1 reference | Nicchorff, 2 days ago | 1 author, 1 change
public async Task<ICollection<HemocentroDTO>> MapeamentoParaHemocentroDTOS()
{
    var listaTipossanguineos = await ListaDeTiposSanguineosAsync();
    var listaComHemocentros = await _context.Hemocentros.ToListAsync();
    var hemocentroDTOs = new List<HemocentroDTO>();
    var bancoDeSangueDTO = new BancoDeSangueDTO();

    foreach (var iten in listaComHemocentros)
    {
        var hemocentroDTO = new HemocentroDTO();
        IList<string> listaComOsTipoSanguineoFaltando = new List<string>();
        hemocentroDTO.Hemocentro = iten;
        foreach (var item in listaTipossanguineos)
        {
            var listaComTodoVolumeColetado = await BuscarTipoSanguineosDeDoadoresNoHemocentro(item.ID, iten.Id);
            var valorComTodosOsVolumesSomados = CalcularTotalDeSangue(listaComTodoVolumeColetado);
            var mediaColetados = TirarMediaDoTotalDeSangue(valorComTodosOsVolumesSomados);
            if (mediaColetados < bancoDeSangueDTO.QuantidadeMinimaSugerida)
            {
                listaComOsTipoSanguineoFaltando.Add($"{item.TipoSanguineo} {(item.IsPositivo ? "+" : "-")}");
            }
        }
        hemocentroDTO.TipoSanguineoFaltando = listaComOsTipoSanguineoFaltando;
        hemocentroDTOs.Add(hemocentroDTO);
    }

    return hemocentroDTOs;
}
```

então esse método é chamado na controller, onde ele tem uma lista dos tipos sanguíneos, lista com os hemocentros, então instanciamos o DTO para podermos adicionar as informações que vão ser passada para a view, temos uma série de métodos para fazermos a manipulação dos dados(que para não ficar tão grande, está lá no github, mas por mim eu explicaria tudo) mas é assim que passamos as informações.

Link do repositório do projeto no github: [EricoAP/SangES: Projeto desenvolvido para a matéria de Programação Orientada a Objetos II, com finalidade de combater as dificuldades para a doação de sangue. (github.com)]