# Implementando Autenticação com Google em .NET MVC

Olá! Nesta etapa, vamos configurar o sistema de login do seu site para que os usuários possam se autenticar usando suas contas do Google. Este é um processo comum e seguro, que facilita o acesso dos usuários sem que eles precisem criar e lembrar de uma nova senha para o seu site.

Vamos dividir o processo nas seguintes etapas:

1. **Registrar seu aplicativo no Google Cloud Console.**
2. **Instalar o pacote de autenticação do Google no seu projeto .NET.**
3. **Configurar o serviço de autenticação do Google no seu aplicativo.**
4. **Criar as ações no Controller para lidar com o login e o retorno do Google.**
5. **Atualizar a interface do usuário para incluir o botão de login e exibir informações do usuário.**

## 1. Registrar seu Aplicativo no Google Cloud Console

Antes de poder usar o login do Google, você precisa informar ao Google sobre o seu aplicativo. Isso é feito no Google Cloud Console.

- **1.1. Acesse o Google Cloud Console:** Vá para console.cloud.google.com e faça login com sua conta do Google.
- **1.2. Crie um Novo Projeto (ou selecione um existente):**
  - No topo da página, clique no seletor de projetos (geralmente ao lado do logo "Google Cloud Platform").
  - Clique em "NOVO PROJETO".
  - Dê um nome ao seu projeto (ex: "MeuSiteDeComidas") e clique em "CRIAR".
- **1.3. Habilite as APIs necessárias:**
  - Com o projeto selecionado, vá para o menu de navegação (ícone de hambúrguer no canto superior esquerdo) > "APIs e Serviços" > "Biblioteca".
  - Procure por "Google People API" e habilite-a. Esta API permite buscar informações do perfil do usuário.
  - Você também pode precisar da "Identity Toolkit API" ou garantir que as APIs relacionadas à identidade estejam ativas.

- **1.4. Configure a Tela de Consentimento OAuth:**
  - No menu de navegação, vá para "APIs e Serviços" > "Tela de consentimento OAuth".
  - Escolha o tipo de usuário:
    - **Externo:** Para que qualquer usuário com uma conta Google possa usar. Selecione esta opção.
    - **Interno:** Apenas para usuários na sua organização G Suite (não é o caso aqui).
  - Clique em "CRIAR".
  - Preencha as informações do aplicativo:
    - **Nome do app:** O nome que os usuários verão (ex: "Meu Site de Vendas de Comida").
    - **E-mail para suporte do usuário:** Seu e-mail.
    - **Logotipo do app (opcional):** Pode adicionar depois.
    - **Domínios autorizados:** Adicione o domínio principal do seu site quando ele estiver online (ex: `meusitedecomidas.com.br`). Para desenvolvimento local, você não precisa preencher isso agora, mas precisará adicionar os URIs de redirecionamento corretos na próxima etapa.
    - **Informações de contato do desenvolvedor:** Seu e-mail.
  - Clique em "SALVAR E CONTINUAR".
  - **Escopos:** Clique em "ADICIONAR OU REMOVER ESCOPOS". Selecione os escopos mínimos necessários. Para login, você geralmente precisa de:
    - `.../auth/userinfo.email` (para ver o endereço de e-mail do usuário)
    - `.../auth/userinfo.profile` (para ver informações básicas do perfil como nome e foto)
    - `openid`
    - Clique em "ATUALIZAR" e depois "SALVAR E CONTINUAR".
  - **Usuários de teste (Opcional durante o desenvolvimento):** Se seu app estiver em modo de teste, você pode adicionar contas Google específicas que poderão usá-lo. Clique em "ADICIONAR USUÁRIOS" e insira os e-mails. Depois de publicar o app, ele estará disponível para todos.
  - Clique em "SALVAR E CONTINUAR" e revise o resumo. Volte ao painel.
- **1.5. Crie as Credenciais OAuth 2.0:**
  - No menu de navegação, vá para "APIs e Serviços" > "Credenciais".
  - Clique em "+ CRIAR CREDENCIAIS" e selecione "ID do cliente OAuth".
  - **Tipo de aplicativo:** Selecione "Aplicativo da Web".
  - **Nome:** Um nome para esta credencial (ex: "Credencial Web MeuSiteDeComidas").

    - **URIs de redirecionamento autorizados:** Esta é uma etapa crucial. O Google redirecionará o usuário de volta para o seu aplicativo após a autenticação. Você precisa adicionar o URI correto.
        - Para desenvolvimento local com ASP.NET Core, o URI padrão é geralmente `https://localhost:<PORTA>/signin-google` ou `http://localhost:<PORTA>/signin-google`. Verifique a porta que seu projeto usa quando você o executa. Por exemplo, se for `https://localhost:7001`, adicione `https://localhost:7001/signin-google`.
        - **Importante:** `/signin-google` é o caminho padrão que o middleware de autenticação do Google no ASP.NET Core escuta. Você deve adicionar este URI.
    - Clique em "CRIAR".
- **1.6. Anote seu `Client ID` e `Client Secret`:** Após a criação, uma janela pop-up mostrará seu **ID do cliente** e sua **Chave secreta do cliente**. Copie esses valores e guarde-os em um local seguro. Você precisará deles no seu código. **Nunca compartilhe sua Chave Secreta publicamente (ex: em repositórios de código públicos).**

## 2. Instalar o Pacote de Autenticação do Google

No seu projeto .NET MVC, você precisará adicionar um pacote NuGet que facilita a integração com o Google.

- Abra seu projeto no Visual Studio ou Visual Studio Code.
- Abra o Console do Gerenciador de Pacotes (no Visual Studio: Ferramentas > Gerenciador de Pacotes NuGet > Console do Gerenciador de Pacotes) ou um terminal na raiz do projeto.
- Execute o seguinte comando: `powershell Install-Package Microsoft.AspNetCore.Authentication.Google` Ou usando a CLI do .NET: `bash dotnet add package Microsoft.AspNetCore.Authentication.Google`

## 3. Configurar o Serviço de Autenticação do Google

Agora, você precisa configurar seu aplicativo para usar a autenticação do Google. Isso é feito no arquivo `Program.cs` (para .NET 6 e mais recentes) ou `Startup.cs` (para versões mais antigas do .NET Core).

**Para .NET 6+ ( `Program.cs` ):**

```
// Program.cs
var builder = WebApplication.CreateBuilder(args);
```

```
// Adicionar serviços ao contêiner.
builder.Services.AddControllersWithViews();

// Configuração da Autenticação
builder.Services.AddAuthentication(options =>
{
    options.DefaultScheme =
Microsoft.AspNetCore.Authentication.Cookies.CookieAuthenticationDefaults.AuthenticationSch
    options.DefaultChallengeScheme =
Microsoft.AspNetCore.Authentication.Google.GoogleDefaults.AuthenticationScheme;
})
.AddCookie(options =>
{
    options.LoginPath = "/Account/Login"; // Página para redirecionar se não estiver
logado e tentar acessar recurso protegido
})
.AddGoogle(Microsoft.AspNetCore.Authentication.Google.GoogleDefaults.AuthenticationSchem
options =>
{
    // Obter ClientID e ClientSecret do appsettings.json ou User Secrets
    options.ClientId = builder.Configuration["Authentication:Google:ClientId"];
    options.ClientSecret =
builder.Configuration["Authentication:Google:ClientSecret"];
    // options.CallbackPath = "/signin-google"; // O padrão já é este, mas pode ser
explicitado
});

// Adicionar DbContext (exemplo, você já terá isso configurado para o MySQL)
// builder.Services.AddDbContext<ApplicationDbContext>(options =>
//    options.UseMySql(builder.Configuration.GetConnectionString("DefaultConnection"),
new MySqlServerVersion(new Version(8, 0, 21))));

var app = builder.Build();

// Configurar o pipeline de requisição HTTP.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication(); // << IMPORTANTE: Adicionar ANTES de UseAuthorization
app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
```

```
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

**Armazenando ClientID e ClientSecret de forma segura:**

É uma má prática colocar o `ClientId` e `ClientSecret` diretamente no código. Use o sistema de configuração do .NET:

1. **`appsettings.json` (para desenvolvimento, mas não ideal para produção para segredos):** `json // appsettings.Development.json ou appsettings.json { "Logging": { "LogLevel": { "Default": "Information", "Microsoft.AspNetCore": "Warning" } }, "AllowedHosts": "*", "Authentication": { "Google": { "ClientId": "SEU_CLIENT_ID_AQUI", "ClientSecret": "SEU_CLIENT_SECRET_AQUI" } }, "ConnectionStrings": { // Sua string de conexão com o MySQL } }`
2. **User Secrets (Recomendado para desenvolvimento):**
    - Clique com o botão direito no projeto no Visual Studio > "Gerenciar Segredos do Usuário".
    - Isso abrirá um arquivo `secrets.json`. Adicione suas credenciais lá: `json { "Authentication:Google:ClientId": "SEU_CLIENT_ID_AQUI", "Authentication:Google:ClientSecret": "SEU_CLIENT_SECRET_AQUI" }`
    - Para produção (ex: Azure App Service), você usará as configurações do aplicativo do serviço de hospedagem.

**Explicação da Configuração:** * `AddAuthentication()`: Registra os serviços de autenticação. * `DefaultScheme = CookieAuthenticationDefaults.AuthenticationScheme`: Define que a autenticação baseada em cookies é o esquema padrão para logar o usuário no seu site após a autenticação externa. * `DefaultChallengeScheme = GoogleDefaults.AuthenticationScheme`: Define que, se um usuário não autenticado tentar acessar um recurso protegido, ele deve ser desafiado (redirecionado) para o login do Google. * `AddCookie()`: Configura a autenticação por cookie. `LoginPath` é para onde o usuário é redirecionado se o cookie de autenticação não for válido. * `AddGoogle()`: Configura o provedor de autenticação do Google. * `options.ClientId` e `options.ClientSecret`: São as credenciais que você obteve do Google Cloud Console. * `app.UseAuthentication();` e `app.UseAuthorization();`: Estes middlewares devem ser adicionados ao pipeline de requisições, e `UseAuthentication` deve vir antes de `UseAuthorization`.

## 4. Criar o `AccountController`

Este controller lidará com as solicitações de login e o retorno do Google.

- Crie um novo controller chamado `AccountController.cs` na pasta `Controllers`.

```
// Controllers/AccountController.cs
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Authentication.Google;
using Microsoft.AspNetCore.Mvc;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
// Adicione o using para seus modelos e DbContext, ex:
// using MeuSiteDeComidas.Data;
// using MeuSiteDeComidas.Models;

public class AccountController : Controller
{
    // Se você estiver usando Entity Framework Core, injete seu DbContext aqui
    // private readonly ApplicationDbContext _context;
    // public AccountController(ApplicationDbContext context) { _context = context; }

    [HttpGet]
    public IActionResult Login(string returnUrl = "/")
    {
        // Se o usuário já estiver logado, redirecione para a returnUrl ou página inicial
        if (User.Identity.IsAuthenticated)
        {
            return LocalRedirect(returnUrl);
        }
        ViewData["ReturnUrl"] = returnUrl;
        return View(); // Uma view simples com o botão "Login com Google"
    }

    [HttpGet]
    public IActionResult LoginWithGoogle(string returnUrl = "/")
    {
        var properties = new AuthenticationProperties
        {
            RedirectUri = Url.Action(nameof(GoogleCallback)), // Ação que o Google
chamará após o login
            Items = { { "ReturnUrl", returnUrl } } // Para guardar a URL de retorno
        };
        return Challenge(properties, GoogleDefaults.AuthenticationScheme);
    }

    public async Task<IActionResult> GoogleCallback()
    {
```

```
        var authenticateResult = await
HttpContext.AuthenticateAsync(GoogleDefaults.AuthenticationScheme);

        if (!authenticateResult.Succeeded)
        {
            // Tratar falha na autenticação do Google (ex: usuário negou acesso)
            TempData["ErrorMessage"] = "Falha ao autenticar com o Google.";
            return RedirectToAction(nameof(Login));
        }

        // Informações do usuário vindas do Google
        var googleUser = authenticateResult.Principal;
        var userId = googleUser.FindFirstValue(ClaimTypes.NameIdentifier); // ID único
do Google
        var email = googleUser.FindFirstValue(ClaimTypes.Email);
        var name = googleUser.FindFirstValue(ClaimTypes.Name);
        var givenName = googleUser.FindFirstValue(ClaimTypes.GivenName);
        var surname = googleUser.FindFirstValue(ClaimTypes.Surname);
        var profilePic = googleUser.FindFirstValue("urn:google:picture") ??
googleUser.FindFirstValue("picture"); // Google pode usar nomes diferentes para o
claim da foto

        // TODO: Aqui você deve:
        // 1. Verificar se o usuário (pelo userId ou email do Google) já existe no seu banco
de dados (tabela Usuario).
        //    var usuarioExistente = await _context.Usuarios.FirstOrDefaultAsync(u => u.Id ==
userId || u.Email == email);
        // 2. Se não existir, crie um novo registro Usuario com os dados do Google.
        //    if (usuarioExistente == null)
        //    {
        //        var novoUsuario = new Usuario
        //        {
        //            Id = userId, // Usar o ID do Google como chave primária é uma boa
prática
        //            Nome = name,
        //            Email = email,
        //            FotoUrl = profilePic
        //        };
        //        _context.Usuarios.Add(novoUsuario);
        //        await _context.SaveChangesAsync();
        //        usuarioExistente = novoUsuario;
        //    }
        //    else if (string.IsNullOrEmpty(usuarioExistente.Id) && !
string.IsNullOrEmpty(userId))
        //    {
        //        // Caso o usuário exista pelo email mas ainda não tem o Google ID
associado
        //        usuarioExistente.Id = userId;
        //        usuarioExistente.FotoUrl = profilePic; // Atualizar foto
        //        _context.Usuarios.Update(usuarioExistente);
        //        await _context.SaveChangesAsync();
        //    }
```

```
        // Criar os claims para o cookie de autenticação local
        var claims = new List<Claim>
        {
            new Claim(ClaimTypes.NameIdentifier, userId), // Ou o ID do seu banco de
dados: usuarioExistente.Id.ToString()
            new Claim(ClaimTypes.Name, name),
            new Claim(ClaimTypes.Email, email),
            // Adicione outros claims que seu aplicativo possa precisar
            // new Claim("urn:google:picture", profilePic)
        };

        var claimsIdentity = new ClaimsIdentity(claims,
CookieAuthenticationDefaults.AuthenticationScheme);
        var authProperties = new AuthenticationProperties
        {
            IsPersistent = true, // Manter o usuário logado mesmo após fechar o
navegador (opcional)
            // ExpiresUtc = DateTimeOffset.UtcNow.AddMinutes(60) // Tempo de expiração do
cookie (opcional)
        };

        // Logar o usuário no seu sistema (cria o cookie de autenticação)
        await HttpContext.SignInAsync(
            CookieAuthenticationDefaults.AuthenticationScheme,
            new ClaimsPrincipal(claimsIdentity),
            authProperties);

        // Redirecionar para a URL de retorno original ou para a página inicial
        string returnUrl = authenticateResult.Properties.Items[".redirect"]; // O
middleware do Google pode usar ".redirect"
        if (string.IsNullOrEmpty(returnUrl) &&
authenticateResult.Properties.Items.TryGetValue("ReturnUrl", out var
originalReturnUrl))
        {
            returnUrl = originalReturnUrl;
        }

        return LocalRedirect(returnUrl ?? "/");
    }

    [HttpPost]
    [ValidateAntiForgeryToken] // Proteção contra CSRF
    public async Task<IActionResult> Logout()
    {
        await
HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
        // Opcional: também fazer logout do Google (geralmente não é necessário para o
logout do seu site)
        // await HttpContext.SignOutAsync(GoogleDefaults.AuthenticationScheme);
        return RedirectToAction("Index", "Home");
```

```
    }
}
```

**Explicação do `AccountController`:** * `Login(string returnUrl)`: Esta ação exibe a página de login. Se o usuário já estiver autenticado, ele é redirecionado. `returnUrl` é usado para redirecionar o usuário de volta para a página que ele estava tentando acessar antes de ser solicitado o login. * `LoginWithGoogle(string returnUrl)`: Esta ação é chamada quando o usuário clica no botão "Login com Google". Ela cria `AuthenticationProperties` para configurar o redirecionamento de volta para `GoogleCallback` e para armazenar a `returnUrl`. Em seguida, ela chama `Challenge` com o esquema do Google, o que inicia o processo de autenticação com o Google. * `GoogleCallback()`: Esta é a ação que o Google chama após o usuário se autenticar (ou negar) no site do Google. O `HttpContext.AuthenticateAsync(GoogleDefaults.AuthenticationScheme)` tenta obter o resultado da autenticação do Google. * Se a autenticação falhar, o usuário é redirecionado de volta para a página de login com uma mensagem de erro. * Se for bem-sucedido, as informações do usuário (claims) são extraídas do `authenticateResult.Principal`. * **TODO:** A seção comentada é onde você integrará com seu banco de dados: buscar ou criar o `Usuario`. * Novos `Claims` são criados para o cookie de autenticação local do seu site. * `HttpContext.SignInAsync()` é chamado para logar o usuário no seu aplicativo, criando o cookie de autenticação. * Finalmente, o usuário é redirecionado para a `returnUrl` original ou para a página inicial. * `Logout()`: Esta ação faz o logout do usuário do seu site, removendo o cookie de autenticação.

## 5. Atualizar a Interface do Usuário (Views)

### 5.1. Criar a View de Login (`Views/Account/Login.cshtml`):

```
 @* Views/Account/Login.cshtml *@
@{
    ViewData["Title"] = "Login";
    var returnUrl = ViewData["ReturnUrl"] as string ?? "/";
}

<h2>@ViewData["Title"]</h2>

@if (TempData["ErrorMessage"] != null)
{
    <div class="alert alert-danger" role="alert">
        @TempData["ErrorMessage"]
    </div>
}

<div>
```

```
    <h4>Use sua conta do Google para entrar.</h4>
    <hr />
    <form asp-action="LoginWithGoogle" asp-route-returnUrl="@returnUrl"
method="get">
        <button type="submit" class="btn btn-primary">Login com Google</button>
    </form>
</div>
```

**5.2. Adicionar Links de Login/Logout no Layout ( `Views/Shared/_Layout.cshtml` ):**

Você vai querer mostrar um link de "Login" se o usuário não estiver autenticado, e informações do usuário e um link de "Logout" se ele estiver.

```
 @* Exemplo de como adicionar no _Layout.cshtml, geralmente no header ou menu
*
@using System.Security.Claims

<!DOCTYPE html>
<html>
<head>
    <title>@ViewData["Title"] - Meu Site de Comidas</title>
    @* Seus links de CSS e outros meta tags aqui *@
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-
white border-bottom box-shadow mb-3">
            <div class="container">
                <a class="navbar-brand" asp-area="" asp-controller="Home" asp-
action="Index">Meu Site de Comidas</a>
                <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target=".navbar-collapse" aria-controls="navbarSupportedContent"
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex justify-content-
between">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home"
asp-action="Index">Início</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Cardapio"
asp-action="Index">Cardápio</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-
controller="Agendamento" asp-action="Criar">Agendar Pedido</a>
                        </li>
```

```
                </ul>
                <ul class="navbar-nav">
                    @if (User.Identity.IsAuthenticated)
                    {
                        <li class="nav-item">
                            <span class="navbar-text">Olá, @User.Identity.Name!</span>
                            @{
                                // Para pegar a foto, você precisaria ter adicionado o claim
"urn:google:picture" ou similar
                                // var profilePictureUrl =
User.FindFirstValue("urn:google:picture");
                                // if (!string.IsNullOrEmpty(profilePictureUrl))
                                // { <img src="@profilePictureUrl" alt="Foto"
style="width:30px; height:30px; border-radius:50%; margin-left:10px;" /> }
                            }
                        </li>
                        <li class="nav-item">
                            <form asp-controller="Account" asp-action="Logout"
method="post" id="logoutForm" class="form-inline">
                                <button type="submit" class="nav-link btn btn-link text-
dark">Logout</button>
                            </form>
                        </li>
                    }
                    else
                    {
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-controller="Account" asp-
action="Login">Login</a>
                        </li>
                    }
                </ul>
            </div>
        </div>
    </nav>
</header>

<div class="container">
    <main role="main" class="pb-3">
        @RenderBody()
    </main>
</div>

@* Seus scripts JS aqui *@
@await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

**5.3. Proteger Controllers ou Actions:**

Para exigir que um usuário esteja logado para acessar certas partes do seu site (como a página de agendamento), use o atributo `[Authorize]`.

```
// Exemplo no AgendamentoController.cs
using Microsoft.AspNetCore.Authorization; // Não esqueça deste using
using Microsoft.AspNetCore.Mvc;

[Authorize] // Aplica a todas as actions neste controller
public class AgendamentoController : Controller
{
    // ... suas actions ...

    // Se quiser autorizar apenas uma action específica:
    // [Authorize]
    public IActionResult Criar()
    {
        return View();
    }
}
```

## Testando

1. Execute seu aplicativo.
2. Tente acessar uma página protegida pelo `[Authorize]` (ex: `/Agendamento/Criar`). Você deve ser redirecionado para `/Account/Login`.
3. Na página de login, clique no botão "Login com Google".
4. Você será redirecionado para a página de login do Google. Faça login com uma conta Google (uma das contas de teste, se você configurou assim e seu app ainda não está publicado).
5. Conceda as permissões solicitadas.
6. Você deve ser redirecionado de volta para o seu aplicativo, para a ação `GoogleCallback`, e então para a página original que tentou acessar ou para a página inicial, já logado.
7. Verifique se o nome do usuário aparece no layout e se o link de "Logout" está visível.
8. Clique em "Logout" para testar o processo de logout.

Lembre-se de implementar a lógica de banco de dados no `GoogleCallback` para salvar ou carregar as informações do usuário. Este é um guia completo, mas cada projeto pode ter suas particularidades. Vá testando passo a passo!