Criando as Páginas Principais com Layout Baseado no Figma (.NET MVC)

Olá! Agora que já temos os modelos de dados e a autenticação com Google planejada, vamos focar na criação das páginas principais do seu site de vendas de comidas. Usaremos o padrão MVC (.NET) e tentaremos replicar o visual que você viu no Figma, utilizando HTML, CSS e JavaScript.

As páginas principais são: 1. **Página Inicial:** Com a descrição da sua loja. 2. **Cardápio:** Para listar os alimentos, preços e descrições. 3. **Agendamento de Pedidos:** Onde os usuários logados poderão agendar seus pedidos.

Também vamos abordar a criação do layout base, incluindo o menu lateral navegável (acionado pelo botão de "três listras").

1. Estrutura do Layout Base (`_Layout.cshtml`)

O arquivo `_Layout.cshtml` (localizado em `Views/Shared`) é o template principal do seu site. Ele define a estrutura comum a todas as páginas, como o cabeçalho, menu de navegação e rodapé.

Objetivo: Replicar a estrutura visual do Figma, especialmente o cabeçalho com o nome do site/logo e o botão que aciona o menu lateral.

Passos e Explicações:

- **1.1. Analise o Figma:** Identifique os elementos principais do layout no Figma: o `top app bar` (cabeçalho), o `nav drawer` (menu lateral) e como eles se comportam.
- **1.2. Modifique `_Layout.cshtml` :**

  ``` html <!DOCTYPE html>

  

  NomeDaSuaLoja

  ```

```
        <div class="collapse navbar-collapse">
          <ul class="navbar-nav ms-auto mb-2 mb-lg-0">
            @await Html.PartialAsync("_LoginPartial") <!-- Parte de login/logout
que criamos antes -->
```

```
            </ul>
          </div>
        </div>
    </nav>
</header>

<!-- Sidebar (Menu Lateral) -->
<div class="d-flex" id="wrapper">
    <div class="bg-light border-end" id="sidebar-wrapper">
        <div class="sidebar-heading border-bottom bg-light">Menu Principal</
div>
        <div class="list-group list-group-flush">
            <a class="list-group-item list-group-item-action list-group-item-light
p-3" asp-controller="Home" asp-action="Index">Início</a>
            <a class="list-group-item list-group-item-action list-group-item-light
p-3" asp-controller="Cardapio" asp-action="Index">Cardápio</a>
            <a class="list-group-item list-group-item-action list-group-item-light
p-3" asp-controller="Agendamento" asp-action="Criar">Agendar Pedido</a>
            @if (User.Identity.IsAuthenticated)
            {
                <a class="list-group-item list-group-item-action list-group-item-light
p-3" asp-controller="Agendamento" asp-action="MeusAgendamentos">Meus
Agendamentos</a>
            }
        </div>
    </div>

    <!-- Conteúdo da Página -->
    <div id="page-content-wrapper">
        <div class="container-fluid px-4">
            <main role="main" class="pb-3 pt-3">
                @RenderBody() <!-- Aqui o conteúdo específico de cada página será
renderizado -->
            </main>
        </div>
    </div>
</div>

<footer class="border-top footer text-muted mt-auto py-3 bg-light">
    <div class="container">
        &copy; @DateTime.Now.Year - NomeDaSuaLoja - Todos os direitos
reservados
    </div>
</footer>

<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
<!-- Ou sua lib Material Design JS -->
<script src="~/js/site.js" asp-append-version="true"></script>
@await RenderSectionAsync("Scripts", required: false)
```

* **1.3. CSS para o Layout e Sidebar (`wwwroot/css/site.css`):** Você precisará de CSS para estilizar o sidebar e o toggle. css / wwwroot/css/site.css / body { overflow-x: hidden; / Previne scroll horizontal causado pelo sidebar / }

wrapper {

```
 display: flex;
padding-top: 56px; /* Altura da navbar fixa */
```

}

sidebar-wrapper {

```
 min-height: calc(100vh - 56px);
width: 250px;
margin-left: -250px; /* Esconde o sidebar por padrão */
transition: margin 0.25s ease-out;
background-color: #f8f9fa; /* Cor de fundo do sidebar */
border-right: 1px solid #dee2e6;
```

}

sidebar-wrapper .sidebar-heading {

```
 padding: 0.875rem 1.25rem;
font-size: 1.2rem;
```

}

sidebar-wrapper .list-group {

```
 width: 100%;
```

}

page-content-wrapper {

```
 flex: 1;
min-width: 0; /* Permite que o conteúdo encolha */
padding-top: 1rem; /* Espaçamento do topo do conteúdo */
```

}

wrapper.toggled #sidebar-wrapper {

```
 margin-left: 0;
```

}

/ Estilos para telas menores onde o sidebar cobre tudo / @media (max-width: 991.98px) { #sidebar-wrapper { margin-left: -250px; } #wrapper.toggled #sidebar-wrapper { margin-left: 0; position: fixed; / Ou absolute, dependendo do efeito desejado / z-index: 1020; / Abaixo da navbar, mas acima do conteúdo / height: 100vh; overflow-y: auto; } #wrapper.toggled #page-content-wrapper { / Adicionar um overlay ou escurecer o conteúdo quando o sidebar estiver aberto / } } * **1.4. JavaScript para o Toggle do Sidebar (`wwwroot/js/site.js`):** javascript // wwwroot/js/site.js $(document).ready(function () { $("#sidebarToggle").on("click", function (e) { e.preventDefault(); $("#wrapper").toggleClass("toggled"); });

```
 // Opcional: Fechar o sidebar ao clicar fora dele em telas menores
$(document).on('click', function(event) {
    if ($("#wrapper").hasClass("toggled") && $(event.target).closest("#sidebar-wrapper").length === 0 && $(event.target).closest("#sidebarToggle").length === 0) {
        if ($(window).width() < 992) { // Apenas para telas menores
            $("#wrapper").removeClass("toggled");
        }
    }
});
```

}); ``` **Observação:** O exemplo acima usa Bootstrap 5 para a estrutura e ícones Material Icons. Se você optar por uma biblioteca Material Design como Materialize CSS ou componentes Material para Bootstrap, os nomes das classes e a estrutura podem variar. A ideia é ter um cabeçalho fixo e um menu lateral que desliza para dentro e para fora.

2. Página Inicial (`HomeController` e `Views/Home/Index.cshtml`)

Esta página geralmente apresenta sua loja.

- **2.1. `HomeController.cs` (em `Controllers`):** Normalmente, já vem criado com o projeto. A action `Index` é a padrão. ```csharp // Controllers/HomeController.cs using Microsoft.AspNetCore.Mvc; using System.Diagnostics; // using NomeDaSuaLoja.Models; // Se precisar de modelos específicos para a Home

  public class HomeController : Controller { public IActionResult Index() { // Você pode passar dados para a View se necessário // ViewBag.MensagemBoasVindas = "Bem-vindo à nossa loja de comidas!"; return View(); }

```
 public IActionResult Privacy()
{
    return View();
}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
public IActionResult Error()
{
    return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
}
```

} * **2.2. `Index.cshtml` (em `Views/Home`):** Aqui você colocará o conteúdo HTML da sua página inicial. html @ Views/Home/Index.cshtml @ @{ ViewData["Title"] = "Página Inicial"; }

Bem-vindo à NomeDaSuaLoja!

Aqui você encontra as melhores comidas caseiras, feitas com carinho e ingredientes frescos.

Explore nosso cardápio e faça seu agendamento de forma rápida e fácil.

Ver Cardápio Agendar Pedido

Sobre Nós

Somos apaixonados por culinária e nosso objetivo é levar até você pratos deliciosos que aquecem o coração. Nossa história começou em [ano] com o sonho de compartilhar nossas receitas de família...

Nossa Loja
`` **Dica:** Crie uma pasta images dentro de wwwroot` para guardar as imagens do seu site.

3. Página de Cardápio (`CardapioController` e `Views/Cardapio/Index.cshtml`)

Esta página listará seus produtos (comidas).

- **3.1. Crie `CardapioController.cs` (em `Controllers`):** ```csharp // Controllers/CardapioController.cs using Microsoft.AspNetCore.Mvc; // using NomeDaSuaLoja.Data; // Seu DbContext // using NomeDaSuaLoja.Models; // Seus modelos, incluindo Produto // using Microsoft.EntityFrameworkCore; // Para ToListAsync() using System.Collections.Generic; // Para List using System.Threading.Tasks; // Para Task

  public class CardapioController : Controller { // private readonly ApplicationDbContext _context; // Injete seu DbContext

```
 // public CardapioController(ApplicationDbContext context)
// {
//     _context = context;
// }

public async Task<IActionResult> Index()
{
    // List<Produto> produtos = await _context.Produtos.ToListAsync(); // Busca produtos do banco
    // return View(produtos);

    // ----- DADOS MOCKADOS PARA TESTE (substitua pela busca no banco) -----
    var produtosMockados = new List<Produto>
    {
        new Produto { Id = 1, Nome = "Feijoada Completa", Descricao = "Deliciosa feijoada com todas as carnes nobres, acompanha arroz, couve, farofa e laranja.", Preco = 35.90m, UrlImagem = "/images/feijoada.jpg", Categoria = "Pratos Principais" },
        new Produto { Id = 2, Nome = "Lasanha à Bolonhesa", Descricao = "Lasanha caseira com molho à bolonhesa artesanal e queijo mussarela
```

```
gratinado.", Preco = 28.50m, UrlImagem = "/images/lasanha.jpg", Categoria =
"Massas" },
        new Produto { Id = 3, Nome = "Suco de Laranja Natural", Descricao =
"Suco feito com laranjas frescas, espremidas na hora.", Preco = 8.00m,
UrlImagem = "/images/suco_laranja.jpg", Categoria = "Bebidas" },
        new Produto { Id = 4, Nome = "Pudim de Leite Condensado", Descricao =
"Clássico pudim de leite condensado com calda de caramelo.", Preco =
12.00m, UrlImagem = "/images/pudim.jpg", Categoria = "Sobremesas" }
    };
    // Crie a pasta wwwroot/images e adicione as imagens feijoada.jpg,
lasanha.jpg, etc.
    return View(produtosMockados); // Passe os produtos para a View
    // ----- FIM DOS DADOS MOCKADOS -----
}

// Opcional: Action para ver detalhes de um produto
// public async Task<IActionResult> Detalhes(int? id)
// {
//     if (id == null)
//     {
//         return NotFound();
//     }
//     var produto = await _context.Produtos.FirstOrDefaultAsync(m => m.Id ==
id);
//     if (produto == null)
//     {
//         return NotFound();
//     }
//     return View(produto);
// }
```

} `` **Importante:** No código acima, comentei a parte que busca do banco de dados (_context.Produtos.ToListAsync() `) e adicionei dados "mockados" (falsos) para você poder testar a view sem ter o banco de dados e o Entity Framework totalmente configurados ainda. Lembre-se de descomentar e ajustar quando o banco estiver pronto.

- **3.2. Crie a View `Index.cshtml` (em `Views/Cardapio`):** ```html @ Views/Cardapio/Index.cshtml @ @model IEnumerable @ Define que o modelo desta View é uma lista de Produtos @

  @{ ViewData["Title"] = "Nosso Cardápio"; }

@ViewData["Title"]

Confira nossas delícias! Para agendar um pedido, clique aqui.

@foreach (var item in Model) {
@if (!string.IsNullOrEmpty(item.UrlImagem)) { @item.Nome } else { Imagem não disponível }

@Html.DisplayFor(modelItem => item.Nome)

@Html.DisplayFor(modelItem => item.Descricao)

Categoria: @Html.DisplayFor(modelItem => item.Categoria)

R$ @item.Preco.ToString("N2")

@ Botão para adicionar ao carrinho/pedido (funcionalidade futura) ou ver detalhes @ @ Ver Detalhes @ Adicionar ao Pedido (Em breve)
}

@section Scripts { } `` **Estilização:** O HTML acima usa classes do Bootstrap 5 para criar um layout de cards responsivo. Você pode ajustar o CSS em site.css` para que os cards e a página se pareçam mais com o design do Figma (ex: sombras, bordas, fontes, cores).

4. Página de Agendamento de Pedidos (`AgendamentoController`)

Esta funcionalidade terá algumas partes: * Formulário para criar um novo agendamento (requer login). * Lista de agendamentos feitos pelo usuário (requer login).

- **4.1. Crie `AgendamentoController.cs` (em `Controllers`):** ```csharp // Controllers/AgendamentoController.cs using Microsoft.AspNetCore.Mvc; using Microsoft.AspNetCore.Authorization; // Para proteger o controller using System.Security.Claims; // Para pegar o ID do usuário logado // using NomeDaSuaLoja.Data; // using NomeDaSuaLoja.Models; // using Microsoft.EntityFrameworkCore; using System.Threading.Tasks; using System.Linq; using System.Collections.Generic; // Para List using System;

  [Authorize] // Exige que o usuário esteja logado para acessar qualquer action aqui public class AgendamentoController : Controller { // private readonly ApplicationDbContext _context;

  ```
  // public AgendamentoController(ApplicationDbContext context)
  // {
  //     _context = context;
  // }
  ```

```
// GET: Agendamento/Criar
public async Task<IActionResult> Criar()
{
    // Você pode querer carregar a lista de produtos para o usuário selecionar
    // ViewBag.ProdutosDisponiveis = await _context.Produtos.ToListAsync();

    // ----- DADOS MOCKADOS PARA PRODUTOS (substitua pela busca no banco) -----
    var produtosMockados = new List<Produto>
    {
        new Produto { Id = 1, Nome = "Feijoada Completa", Preco = 35.90m },
        new Produto { Id = 2, Nome = "Lasanha à Bolonhesa", Preco = 28.50m },
        new Produto { Id = 3, Nome = "Suco de Laranja Natural", Preco = 8.00m }
    };
    ViewBag.ProdutosDisponiveis = produtosMockados;
    // ----- FIM DOS DADOS MOCKADOS -----

    var model = new AgendamentoViewModel(); // Um ViewModel pode ser
útil aqui
    model.DataHoraAgendamento = DateTime.Now.AddDays(1); // Sugestão
inicial
    return View(model);
}

// POST: Agendamento/Criar
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Criar(AgendamentoViewModel viewModel)
{
    // string userId = User.FindFirstValue(ClaimTypes.NameIdentifier); // Pega
o ID do usuário logado

    // if (ModelState.IsValid)
    // {
    //     Agendamento novoAgendamento = new Agendamento
    //     {
    //         UsuarioId = userId,
    //         DataHoraAgendamento = viewModel.DataHoraAgendamento,
    //         Observacoes = viewModel.Observacoes,
    //         Status = "Pendente", // Status inicial
    //         DataHoraCriacao = DateTime.UtcNow,
    //         ValorTotal = 0 // Calcular com base nos itens selecionados
    //     };

    //     // Lógica para adicionar ItensPedido ao novoAgendamento
    //     // foreach (var itemSelecionado in viewModel.ItensSelecionados)
    //     // {
    //     //     var produto = await
_context.Produtos.FindAsync(itemSelecionado.ProdutoId);
    //     //     if (produto != null)
    //     //     {
    //     //         var itemPedido = new ItemPedido
    //     //         {
```

```
    //     //             ProdutoId = produto.Id,
    //     //             Quantidade = itemSelecionado.Quantidade,
    //     //             PrecoUnitario = produto.Preco // Preço no momento da
compra
    //     //         };
    //     //         novoAgendamento.ItensPedido.Add(itemPedido);
    //     //         novoAgendamento.ValorTotal += (produto.Preco *
itemSelecionado.Quantidade);
    //     //     }
    //     // }

    //     // _context.Agendamentos.Add(novoAgendamento);
    //     // await _context.SaveChangesAsync();
    //     TempData["Sucesso"] = "Agendamento criado com sucesso!";
    //     return RedirectToAction(nameof(MeusAgendamentos));
    // }

    // Se chegou aqui, algo falhou, recarregue o formulário
    // ViewBag.ProdutosDisponiveis = await _context.Produtos.ToListAsync(); //
Recarregar produtos
    // ----- DADOS MOCKADOS PARA PRODUTOS (substitua pela busca no banco) -----
    var produtosMockados = new List<Produto>
    {
        new Produto { Id = 1, Nome = "Feijoada Completa", Preco = 35.90m },
        new Produto { Id = 2, Nome = "Lasanha à Bolonhesa", Preco = 28.50m },
        new Produto { Id = 3, Nome = "Suco de Laranja Natural", Preco = 8.00m }
    };
    ViewBag.ProdutosDisponiveis = produtosMockados;
    // ----- FIM DOS DADOS MOCKADOS -----
    TempData["Erro"] = "Falha ao criar agendamento. Verifique os dados.";
    return View(viewModel);
}

// GET: Agendamento/MeusAgendamentos
public async Task<IActionResult> MeusAgendamentos()
{
    // string userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
    // var agendamentosDoUsuario = await _context.Agendamentos
    //                         .Where(a => a.UsuarioId == userId)
    //                         .Include(a => a.ItensPedido) // Inclui os itens do
pedido
    //                         .ThenInclude(ip => ip.Produto) // Inclui os detalhes
do produto em cada item
    //                         .OrderByDescending(a => a.DataHoraCriacao)
    //                         .ToListAsync();
    // return View(agendamentosDoUsuario);

    // ----- DADOS MOCKADOS PARA TESTE (substitua pela busca no banco) -----
    var agendamentosMockados = new List<Agendamento>
    {
        new Agendamento { Id = 1, UsuarioId = "mockUserId",
DataHoraAgendamento = DateTime.Now.AddDays(2), Status = "Pendente",
```

```
ValorTotal = 43.90m, Observacoes = "Caprichar na feijoada!", ItensPedido =
new List<ItemPedido> { new ItemPedido { Produto = new Produto { Nome =
"Feijoada Completa"}, Quantidade = 1, PrecoUnitario = 35.90m }, new
ItemPedido { Produto = new Produto { Nome = "Suco de Laranja"},
Quantidade = 1, PrecoUnitario = 8.00m } } },
        new Agendamento { Id = 2, UsuarioId = "mockUserId",
DataHoraAgendamento = DateTime.Now.AddDays(3), Status = "Confirmado",
ValorTotal = 28.50m, ItensPedido = new List<ItemPedido> { new ItemPedido
{ Produto = new Produto { Nome = "Lasanha à Bolonhesa"}, Quantidade = 1,
PrecoUnitario = 28.50m } } }
    };
    return View(agendamentosMockados);
    // ----- FIM DOS DADOS MOCKADOS -----
}
```

} **ViewModel (`AgendamentoViewModel.cs` em `Models` ou uma nova pasta `ViewModels`):** Para o formulário de criação, um ViewModel pode ser útil para agrupar os dados que a view precisa e para a validação. csharp // Models/AgendamentoViewModel.cs (ou ViewModels/AgendamentoViewModel.cs) using System; using System.Collections.Generic; using System.ComponentModel.DataAnnotations;

public class ItemSelecionadoViewModel { public int ProdutoId { get; set; } public int Quantidade { get; set; } public string NomeProduto { get; set; } // Para exibir no resumo public decimal PrecoProduto { get; set; } // Para exibir no resumo }

public class AgendamentoViewModel { [Required(ErrorMessage = "A data e hora do agendamento são obrigatórias.")] [Display(Name = "Data e Hora para Agendamento")] // Adicionar validação para não permitir datas passadas ou muito futuras public DateTime DataHoraAgendamento { get; set; }

```
 [Display(Name = "Observações Adicionais")]
[StringLength(500)]
public string Observacoes { get; set; }

// Lista de itens que o usuário selecionou. O JavaScript no formulário
preencherá isso.
public List<ItemSelecionadoViewModel> ItensSelecionados { get; set; }

public AgendamentoViewModel()
{
    ItensSelecionados = new List<ItemSelecionadoViewModel>();
}
```

} ```

- **4.2. Crie a View `Criar.cshtml` (em `Views/Agendamento`):** Este será o formulário para o usuário fazer o agendamento. ```html @ Views/Agendamento/Criar.cshtml @ @model AgendamentoViewModel

  @{ ViewData["Title"] = "Agendar Pedido"; var produtosDisponiveis = ViewBag.ProdutosDisponiveis as List ?? new List(); }

  # @ViewData["Title"]

  Complete os dados abaixo para fazer seu agendamento.

  @if (TempData["Erro"] != null) {

  @TempData["Erro"]
  }

  @Html.AntiForgeryToken()

```
<div class="row">
  <div class="col-md-7">
    <h4>Selecione os Itens do Cardápio:</h4>
    <div class="list-group mb-3" id="listaProdutosCardapio">
      @foreach (var produto in produtosDisponiveis)
      {
        <div class="list-group-item d-flex justify-content-between align-items-center">
          <div>
            <h6 class="my-0">@produto.Nome</h6>
            <small class="text-muted">@produto.Descricao</small>
          </div>
          <div class="text-end">
            <span class="text-muted">R$ @produto.Preco.ToString("N2")</span>
            <input type="number" min="0" value="0" class="form-control form-control-sm d-inline-block ms-2" style="width: 70px;" data-produto-id="@produto.Id" data-produto-nome="@produto.Nome" data-produto-preco="@produto.Preco.ToString("F2")" onchange="atualizarResumoPedido()">
          </div>
        </div>
      }
    </div>

    <div class="form-group mb-3">
      <label asp-for="DataHoraAgendamento" class="control-label"></label>
      <input asp-for="DataHoraAgendamento" type="datetime-local"
```

```
class="form-control" />
            <span asp-validation-for="DataHoraAgendamento" class="text-
danger"></span>
        </div>

        <div class="form-group mb-3">
            <label asp-for="Observacoes" class="control-label"></label>
            <textarea asp-for="Observacoes" class="form-control" rows="3"></
textarea>
            <span asp-validation-for="Observacoes" class="text-danger"></span>
        </div>
    </div>

    <div class="col-md-5">
        <h4>Resumo do Pedido:</h4>
        <ul class="list-group mb-3" id="resumoPedidoItens">
            <!-- Itens selecionados aparecerão aqui via JavaScript -->
            <li class="list-group-item d-flex justify-content-between">
                <span>Total (R$)</span>
                <strong id="valorTotalPedido">R$ 0,00</strong>
            </li>
        </ul>
        <!-- Inputs hidden para enviar os itens selecionados -->
        <div id="itensSelecionadosHidden"></div>
    </div>
</div>

<div class="form-group mt-3">
    <input type="submit" value="Finalizar Agendamento" class="btn btn-
primary" />
    <a asp-controller="Home" asp-action="Index" class="btn btn-
secondary">Cancelar</a>
</div>
```

@section Scripts { @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}