Guia Completo: Desenvolvimento do Seu Site de Vendas de Comidas com .NET MVC e MySQL

Olá! Seja bem-vindo a este guia completo que preparei para te ajudar a desenvolver seu site de vendas de comidas. Como você é iniciante, detalhei cada passo da forma mais clara possível, explicando os conceitos e fornecendo exemplos de código. O objetivo é que você consiga construir o site desde a configuração do ambiente até a implementação das funcionalidades principais, com um design inspirado no Figma que você compartilhou.

Visão Geral do Projeto

Vamos construir um site utilizando: * **Backend:** C# com ASP.NET Core MVC * **Frontend:** HTML, CSS e JavaScript (com auxílio de Bootstrap para estrutura e Material Icons para iconografia, buscando similaridade com o Material Design do Figma) * **Banco de Dados:** MySQL (inicialmente local, com preparação para nuvem) * **Autenticação:** Login com Google * **Hospedagem Futura:** Azure

Funcionalidades Principais: 1. Página Inicial com descrição da loja. 2. Cardápio com descrição dos alimentos, preços e imagens. 3. Sistema de Login com Google. 4. Página para Agendamento de Pedidos (requer login). 5. Página para visualização de "Meus Agendamentos" (requer login). 6. Menu lateral navegável (acionado por um botão de "três listras").

Cronograma Estimado (para Iniciantes)

Este é um cronograma estimado, considerando que você dedicará algumas horas por dia ao projeto. O tempo pode variar conforme sua familiaridade com as tecnologias e a complexidade que decidir adicionar.

- **Fase 1: Preparação do Ambiente e Criação do Projeto (1-2 dias)**
    - Detalhes: Instalação do .NET SDK, MySQL Server, MySQL Workbench (ou outro cliente), Visual Studio Code (ou Visual Studio Community). Criação do projeto ASP.NET Core MVC e entendimento inicial da estrutura de pastas.

- **Fase 2: Modelagem do Banco de Dados e Configuração Inicial (1-2 dias)**
    - Detalhes: Definição das classes de modelo (Usuario, Produto, Agendamento, ItemPedido). Configuração do ApplicationDbContext, string de conexão com o MySQL e execução das primeiras migrations para criar as tabelas.
- **Fase 3: Implementação da Autenticação com Google (2-3 dias)**
    - Detalhes: Registro do aplicativo no Google Cloud Console, obtenção de ClientID/ClientSecret, instalação de pacotes NuGet, configuração da autenticação no `Program.cs`, criação do `AccountController` e das views de Login/Logout. Integração com o modelo `Usuario` para salvar/recuperar dados do usuário.
- **Fase 4: Desenvolvimento do Layout Base e Página Inicial (2-3 dias)**
    - Detalhes: Adaptação do `_Layout.cshtml` para incluir o cabeçalho (Top App Bar) e a estrutura do menu lateral (Nav Drawer) conforme o Figma. Implementação do HTML, CSS e JavaScript para o menu navegável. Criação da `HomeController` e da view da Página Inicial com conteúdo descritivo.
- **Fase 5: Desenvolvimento da Página de Cardápio (com dados do banco) (2-3 dias)**
    - Detalhes: Criação do `CardapioController` para buscar os produtos do banco de dados (tabela `Produtos`). Desenvolvimento da view para listar os produtos em formato de cards, exibindo nome, descrição, preço e imagem. Estilização para aproximar do design do Figma.
- **Fase 6: Desenvolvimento da Funcionalidade de Agendamento (3-5 dias)**
    - Detalhes: Criação do `AgendamentoController` com actions para `Criar` (GET e POST) e `MeusAgendamentos`. Definição de um `AgendamentoViewModel`. Desenvolvimento da view `Criar.cshtml` com o formulário de agendamento, seleção de produtos do cardápio e JavaScript para resumo dinâmico do pedido. Desenvolvimento da view `MeusAgendamentos.cshtml`. Implementação da lógica para salvar os agendamentos e seus itens no banco de dados.
- **Fase 7: Testes, Validação de Layout e Refinamentos (2-4 dias)**
    - Detalhes: Execução de testes funcionais em todas as partes do site. Comparação detalhada do layout com o Figma e realização de ajustes no CSS. Testes de responsividade em diferentes tamanhos de tela. Correção de bugs e polimento geral da aplicação.
- **Fase 8: Preparação para Deploy no Azure (1 dia)**
    - Detalhes: Pesquisa e estudo da documentação oficial da Microsoft sobre como publicar um aplicativo ASP.NET Core no Azure App Service e como configurar o Azure Database for MySQL.

Tempo Total Estimado: 3 a 5 semanas.

Estrutura do Guia

Este guia está organizado nas seguintes seções principais, que correspondem às fases do desenvolvimento:

1. **Configuração do Ambiente de Desenvolvimento**
2. **Criação e Estrutura do Projeto .NET MVC**
3. **Definição dos Modelos de Dados e Configuração do Banco (MySQL com EF Core)** (Ref: `modelos_de_dados.md`)
4. **Implementando Autenticação com Google** (Ref: `autenticacao_google_dotnet.md`)
5. **Criando o Layout Base e as Páginas Principais (HTML, CSS, JS)** (Ref: `criacao_paginas_principais.md`)
6. **Adicionando Funcionalidades Dinâmicas ao Cardápio e Agendamento** (Ref: `funcionalidades_cardapio_agendamento.md`)
7. **Validando o Funcionamento, Ajustando o Layout e Refinamentos Finais** (Ref: `validacao_e_ajustes_finais.md`)
8. **Próximos Passos: Deploy no Azure**

Os arquivos Markdown referenciados (`*.md`) contêm os detalhes técnicos e exemplos de código para cada etapa específica e serão fornecidos junto com este guia.

Seção 1: Configuração do Ambiente de Desenvolvimento

Antes de começar a codificar, você precisa preparar seu computador com as ferramentas necessárias.

- **1.1. Instalar o .NET SDK (Software Development Kit):**
    - O .NET SDK inclui tudo que você precisa para construir e rodar aplicações .NET.
    - Vá para o site oficial da Microsoft: dotnet.microsoft.com/download
    - Baixe e instale a versão mais recente do .NET SDK (recomendado .NET 6 ou superior, pois os exemplos foram baseados nele).
    - Após a instalação, abra um terminal (Prompt de Comando, PowerShell, ou Terminal no Linux/macOS) e digite `dotnet --version` para verificar se foi instalado corretamente. Você deverá ver a versão do SDK.

- **1.2. Instalar o MySQL Server e uma Ferramenta de Gerenciamento:**
    - **MySQL Server:** É o sistema de banco de dados que usaremos.
        - Vá para dev.mysql.com/downloads/mysql/
        - Baixe o "MySQL Community Server" para o seu sistema operacional.
        - Durante a instalação, você definirá uma senha para o usuário `root`. Anote essa senha, pois você precisará dela.
    - **MySQL Workbench (Recomendado):** É uma ferramenta visual para gerenciar seu banco de dados MySQL.
        - Vá para dev.mysql.com/downloads/workbench/
        - Baixe e instale.
        - Alternativas: DBeaver, HeidiSQL, ou via linha de comando.
- **1.3. Instalar um Editor de Código/IDE:**
    - **Visual Studio Code (VS Code):** Um editor de código leve, gratuito e muito popular, com excelente suporte para C# e .NET através de extensões.
        - Baixe em code.visualstudio.com
        - Instale as seguintes extensões no VS Code: "C#" (da Microsoft).
    - **Visual Studio Community:** Uma IDE (Ambiente de Desenvolvimento Integrado) completa e gratuita da Microsoft, com muitas ferramentas integradas para desenvolvimento .NET.
        - Baixe em visualstudio.microsoft.com/vs/community/
        - Durante a instalação, selecione a carga de trabalho "Desenvolvimento ASP.NET e Web".

Seção 2: Criação e Estrutura do Projeto .NET MVC

- **2.1. Criar um Novo Projeto ASP.NET Core MVC:**
    - **Usando o Terminal (CLI do .NET):**
        1. Abra o terminal.
        2. Navegue até a pasta onde você quer criar seu projeto (ex: `cd MeusProjetos`).
        3. Execute o comando: `dotnet new mvc -n NomeDaSuaLoja --framework net6.0` (substitua `NomeDaSuaLoja` pelo nome que desejar e `net6.0` pela versão do .NET que instalou, se for diferente).
        4. Isso criará uma nova pasta `NomeDaSuaLoja` com a estrutura do projeto.
        5. Entre na pasta do projeto: `cd NomeDaSuaLoja`.

- **Usando o Visual Studio Community:**
    1. Abra o Visual Studio.
    2. Clique em "Criar um novo projeto".
    3. Procure por "ASP.NET Core Web App (Model-View-Controller)". Selecione e clique em "Próximo".
    4. Dê um nome ao projeto (ex: `NomeDaSuaLoja`), escolha o local e clique em "Próximo".
    5. Selecione o Framework (ex: .NET 6.0). Deixe as outras opções padrão por enquanto (Tipo de autenticação: Nenhum, pois configuraremos o Google manualmente). Clique em "Criar".

- **2.2. Entender a Estrutura de Pastas Principal:**
    - `Controllers`: Contém as classes C# que manipulam as requisições do usuário, interagem com os modelos e selecionam as Views a serem exibidas.
    - `Models`: Contém as classes C# que representam os dados da sua aplicação (ex: Produto, Usuario) e, às vezes, a lógica de negócios relacionada a esses dados. Também pode conter ViewModels.
    - `Views`: Contém os arquivos `.cshtml` (Razor) que definem a interface do usuário (o HTML que será enviado ao navegador).
        - `Views/Shared`: Views que são compartilhadas entre diferentes controllers (ex: `_Layout.cshtml`, `_LoginPartial.cshtml`).
        - `Views/[ControllerName]`: Pastas específicas para as views de cada controller (ex: `Views/Home/Index.cshtml`).
    - `wwwroot`: Contém os arquivos estáticos do site, como CSS, JavaScript, imagens e bibliotecas frontend.
    - `Program.cs` (ou `Startup.cs` em versões mais antigas): Ponto de entrada da aplicação, onde os serviços são configurados e o pipeline de requisições HTTP é definido.
    - `appsettings.json`: Arquivo de configuração para strings de conexão, chaves de API, etc.

Seção 3: Definição dos Modelos de Dados e Configuração do Banco

Nesta fase, você definirá as classes C# que representam suas tabelas no banco de dados e configurará o Entity Framework Core para se comunicar com o MySQL.

Consulte o arquivo `modelos_de_dados.md` para o passo a passo detalhado e os códigos de exemplo para: * Criação das classes `Usuario.cs`, `Produto.cs`,

`Agendamento.cs`, `ItemPedido.cs` na pasta `Models`. * Criação e configuração do `ApplicationDbContext.cs` na pasta `Data`. * Adição da string de conexão do MySQL no `appsettings.json`. * Registro do `DbContext` no `Program.cs`. * Criação das migrations (`dotnet ef migrations add InitialCreate`) e atualização do banco de dados (`dotnet ef database update`).

Seção 4: Implementando Autenticação com Google

Aqui você permitirá que os usuários façam login no seu site usando suas contas do Google.

Consulte o arquivo `autenticacao_google_dotnet.md` para o passo a passo detalhado e os códigos de exemplo para: * Registrar seu aplicativo no Google Cloud Console e obter `ClientID` e `ClientSecret`. * Instalar o pacote NuGet `Microsoft.AspNetCore.Authentication.Google`. * Configurar o serviço de autenticação do Google no `Program.cs` (ou `Startup.cs`), usando o `ClientID` e `ClientSecret` (armazenados de forma segura). * Criar o `AccountController.cs` com actions para `Login`, `LoginWithGoogle`, `GoogleCallback` e `Logout`. * Implementar a lógica no `GoogleCallback` para buscar ou criar um `Usuario` no seu banco de dados com as informações do Google. * Criar a View `Login.cshtml` e atualizar o `_Layout.cshtml` (ou `_LoginPartial.cshtml`) para exibir os links de Login/Logout e informações do usuário. * Proteger controllers/actions com o atributo `[Authorize]`.

Seção 5: Criando o Layout Base e as Páginas Principais

Nesta etapa, você construirá a estrutura visual principal do site e as páginas estáticas ou com listagem simples de dados.

Consulte o arquivo `criacao_paginas_principais.md` para o passo a passo detalhado e os códigos de exemplo para: * Analisar o Figma e adaptar o `Views/Shared/_Layout.cshtml` para ter o cabeçalho (Top App Bar) e o menu lateral (Nav Drawer). * Implementar o HTML, CSS (`wwwroot/css/site.css`) e JavaScript (`wwwroot/js/site.js`) para o funcionamento do menu lateral. * Desenvolver a Página Inicial (`HomeController` e `Views/Home/Index.cshtml`) com a descrição da loja. * Desenvolver a Página de Cardápio (`CardapioController` e `Views/Cardapio/Index.cshtml`) para listar os produtos (inicialmente com dados mockados, depois do banco). * Estilização básica dos componentes para se assemelhar ao Material Design.

Seção 6: Adicionando Funcionalidades Dinâmicas ao Cardápio e Agendamento

Agora, você conectará as páginas de Cardápio e Agendamento ao banco de dados para que elas se tornem dinâmicas.

Consulte o arquivo `funcionalidades_cardapio_agendamento.md` para o passo a passo detalhado e os códigos de exemplo para: * Modificar o `CardapioController` para buscar e exibir produtos da tabela `Produtos`. * Implementar o `AgendamentoController`: * Action `Criar` (GET): Carregar produtos disponíveis para seleção e exibir o formulário de agendamento. * Action `Criar` (POST): Validar os dados do formulário, pegar o ID do usuário logado, criar os registros `Agendamento` e `ItemPedido` no banco de dados, e calcular o valor total. * Action `MeusAgendamentos`: Buscar e exibir os agendamentos feitos pelo usuário logado, incluindo os detalhes dos itens de cada pedido. * Criar/Ajustar as Views `Views/Agendamento/Criar.cshtml` e `Views/Agendamento/MeusAgendamentos.cshtml`. * Utilizar o `AgendamentoViewModel.cs` para facilitar a transferência de dados entre a View e o Controller na criação do agendamento.

Seção 7: Validando o Funcionamento, Ajustando o Layout e Refinamentos Finais

Esta é a fase de testes intensivos e polimento.

Consulte o arquivo `validacao_e_ajustes_finais.md` para um checklist detalhado de: * Testes Funcionais: Percorrer todas as funcionalidades do site (autenticação, navegação, cardápio, agendamento) para garantir que tudo opera como esperado. * Validação do Layout: Comparar cada página e componente com o design do Figma, ajustando HTML e CSS para cores, tipografia, espaçamentos, ícones, etc. * Testes de Responsividade: Usar as ferramentas de desenvolvedor do navegador para verificar como o site se comporta em diferentes tamanhos de tela (mobile, tablet, desktop) e fazer os ajustes necessários com media queries. * Dicas para Debugging: Como usar o inspetor de elementos, console do navegador e debugger para encontrar e corrigir problemas de CSS e JavaScript. * Refinamento Geral: Revisão de textos, usabilidade, performance básica e links quebrados.

Seção 8: Próximos Passos: Deploy no Azure

Após desenvolver e testar seu site localmente, o próximo passo é publicá-lo na internet para que outras pessoas possam acessá-lo. Você mencionou o Azure como sua preferência de hospedagem.

- **8.1. Crie uma Conta no Azure:**
  - Se você ainda não tem, crie uma conta gratuita no Azure em azure.microsoft.com. Contas gratuitas geralmente vêm com créditos para você experimentar os serviços.
- **8.2. Principais Serviços do Azure que Você Usará:**
  - **Azure App Service:** Para hospedar seu aplicativo web ASP.NET Core. Ele gerencia a infraestrutura para você.
  - **Azure Database for MySQL:** Um serviço de banco de dados MySQL gerenciado na nuvem. Isso permitirá que você migre seu banco de dados local para a nuvem.
- **8.3. Passos Gerais para o Deploy (Consulte a Documentação Oficial do Azure para detalhes):**
  1. **Crie um Azure App Service:**
     - No portal do Azure, crie um novo recurso "App Service".
     - Configure o nome do aplicativo (que fará parte da URL, ex: `nomedasuloja.azurewebsites.net`), a pilha de runtime (.NET 6 ou a versão que você usou), e o sistema operacional (Windows ou Linux).
     - Escolha um plano de serviço (há opções gratuitas ou pagas com mais recursos).
  2. **Crie um Azure Database for MySQL:**
     - No portal do Azure, crie um novo recurso "Azure Database for MySQL server".
     - Configure o nome do servidor, região, versão do MySQL, e as credenciais de administrador (usuário e senha).
     - Configure as regras de firewall para permitir o acesso do seu App Service e do seu IP local (para gerenciamento).
  3. **Migre seu Banco de Dados Local para o Azure Database for MySQL:**
     - Você pode usar ferramentas como o MySQL Workbench para exportar seu esquema e dados do banco local e importá-los para o banco de dados do Azure.

4. **Atualize a String de Conexão no seu Aplicativo:**
    - No portal do Azure, vá para as configurações do seu App Service.
    - Encontre a seção "Configuração" e adicione uma nova "String de Conexão".
    - O nome da string de conexão deve ser o mesmo que você usou no `appsettings.json` (ex: `DefaultConnection`).
    - O valor será a string de conexão para o seu Azure Database for MySQL (você pode obtê-la no portal do Azure, na página do seu servidor MySQL).
    - **Importante:** Remova ou comente a string de conexão local do seu `appsettings.json` ou use arquivos de configuração específicos do ambiente (ex: `appsettings.Production.json`) para o deploy.
5. **Publique seu Aplicativo no Azure App Service:**
    - **Do Visual Studio:** Clique com o botão direito no projeto > "Publicar". Escolha Azure, selecione seu App Service e siga as instruções.
    - **Do VS Code ou CLI:** Você pode configurar o deploy via Git, GitHub Actions, Azure DevOps, ou usando a CLI do Azure.
6. **Configure o `ClientID` e `ClientSecret` do Google no Azure:**
    - Assim como a string de conexão, adicione as configurações `Authentication:Google:ClientId` e `Authentication:Google:ClientSecret` nas configurações do aplicativo do seu App Service no portal do Azure. Não os deixe no `appsettings.json` em produção.
    - **Atualize os URIs de Redirecionamento Autorizados no Google Cloud Console:** Adicione o URI de redirecionamento do seu site no Azure (ex: `https://nomedasu loja.azurewebsites.net/signin-google`) às credenciais OAuth 2.0 do seu projeto no Google Cloud Console.

- **8.4. Teste o Site em Produção:**
    - Após o deploy, acesse a URL do seu site no Azure e teste todas as funcionalidades novamente.

Lembre-se que a documentação oficial da Microsoft Azure é sua melhor amiga para o processo de deploy. Ela é detalhada e sempre atualizada.

Este guia, juntamente com os arquivos Markdown detalhados para cada seção, deve fornecer uma base sólida para você construir seu site. O desenvolvimento de software é um processo de aprendizado contínuo. Não hesite em pesquisar, experimentar e, o mais importante, se divertir construindo seu projeto!

Boa sorte!