THOMAS GREG
do Brasil

CRUD Clientes

Mais 5 propriedades

Aplicações Principais

- **Aplicação Backend (API REST):**

A aplicação backend é responsável por fornecer os serviços e funcionalidades para os clientes por meio de uma API RESTful. Ela será desenvolvida utilizando C# com ASP.NET Core, e será hospedada em um servidor web, como o IIS (Internet Information Services).

- **Aplicação Frontend (Interface do Usuário):**

A aplicação frontend consiste na interface com a qual os usuários interagem para realizar operações como criar, atualizar, visualizar e remover clientes e logradouros. Ela pode ser desenvolvida utilizando ASP.NET Core MVC para renderização de páginas HTML tradicionais com Razor.

- **Banco de Dados:**

O banco de dados é uma aplicação fundamental que armazena e gerencia os dados da aplicação, como informações de clientes e logradouros. Para este projeto, será utilizado o SQL Server como sistema de gerenciamento de banco de dados relacional. O SQL Server será acessado pela aplicação backend para realizar operações de leitura e gravação de dados.

- **Serviços de Terceiros:**

A solução pode depender de serviços externos, como serviços de armazenamento em nuvem (por exemplo, AWS S3, Azure Blob Storage), para fornecer funcionalidades adicionais e complementares, principalmente para armazenamento dos logotipos.

Além das aplicações principais mencionadas anteriormente, algumas aplicações de apoio podem ser necessárias para facilitar o desenvolvimento, teste e operação da solução. Aqui estão algumas das aplicações de apoio comumente utilizadas:

Aplicações de Apoio

- **Visual Studio 2019/2022**

IDEs (Integrated Development Environments) populares para desenvolvimento de software em plataforma Microsoft. Elas oferecem recursos avançados de edição de código, depuração, controle de versão e integração com outras ferramentas de desenvolvimento.

- **SQL Server Management Studio (SSMS):**

SSMS é uma ferramenta de gerenciamento e administração para o SQL Server. Ela permite realizar tarefas como criar e gerenciar bancos de dados, escrever consultas SQL, configurar segurança e monitorar o desempenho do banco de dados.

- **Postman/Insomnia:**

Ferramentas de colaboração para o desenvolvimento de APIs. Ela permite testar, depurar e documentar APIs de forma eficiente, facilitando a interação com os endpoints da API, envio de solicitações HTTP e visualização das respostas.

- **Swagger / OpenAPI:**

Swagger é uma ferramenta de documentação de API que permite descrever, documentar e testar APIs de forma interativa. Ela gera automaticamente documentação para a API com base em um arquivo de especificação OpenAPI (anteriormente conhecido como Swagger Specification).

- **Docker:**

Docker é uma ferramenta que permite criar, gerenciar e executar contêineres em um ambiente de desenvolvimento local. Ela simplifica o processo de empacotamento de aplicativos e suas dependências em contêineres, garantindo consistência entre ambientes de desenvolvimento, teste e produção.

- **Git / GitHub:**

Git é um sistema de controle de versão distribuído amplamente utilizado para o rastreamento de mudanças no código-fonte durante o desenvolvimento de software. GitHub é uma plataforma de hospedagem de código que oferece recursos de colaboração, gerenciamento de projetos e integração contínua para equipes de desenvolvimento.

- **SonarQube:**

SonarQube é uma plataforma de código aberto para inspeção contínua da qualidade do código-fonte para realizar análises estáticas de código, identificar problemas de código e aplicar boas práticas de desenvolvimento.

- **Azure DevOps CI/CD:**

Essas são plataformas de integração contínua e entrega contínua (CI/CD) que permitem automatizar o processo de construção, teste e implantação de software. Elas oferecem recursos como pipelines de CI/CD, gerenciamento de projetos, controle de versão e colaboração em equipe.

Bibliotecas

Back-end

- NET Core: Framework web usado para construir o backend da aplicação.
- Entity Framework Core: ORM (Object-Relational Mapper) usado para mapear objetos de domínio para o banco de dados.
- EntityFrameworkCore.SqlServer: Provedor de banco de dados SQL Server para o Entity Framework Core.
- Extensions.DependencyInjection: Biblioteca para injeção de dependência no ASP.NET Core.
- Extensions.Http: Biblioteca para gerenciamento de requisições HTTP no ASP.NET Core.
- Extensions.Options: Biblioteca para configurações baseadas em opções no ASP.NET Core.
- VisualStudio.Web.CodeGeneration.Design: Biblioteca de geração de código para ASP.NET Core.

Front-end

- NET Core: Framework web usado para construir o backend da aplicação.
- Entity Framework Core: ORM (Object-Relational Mapper) usado para mapear objetos de domínio para o banco de dados.

- EntityFrameworkCore.SqlServer: Provedor de banco de dados SQL Server para o Entity Framework Core.
- Extensions.DependencyInjection: Biblioteca para injeção de dependência no ASP.NET Core.
- Extensions.Http: Biblioteca para gerenciamento de requisições HTTP no ASP.NET Core.
- Extensions.Options: Biblioteca para configurações baseadas em opções no ASP.NET Core.
- VisualStudio.Web.CodeGeneration.Design: Biblioteca de geração de código para ASP.NET Core.

Banco de Dados

- Banco SQL Server.

Para a modelagem de dados do desafio proposto, vamos criar um esquema simples que represente as entidades necessárias para o cadastro de clientes e logradouros. Vamos utilizar uma abordagem relacional básica, considerando os requisitos fornecidos.

Entidades

Cliente:

ID (Chave Primária)

Nome

Email (Único)

Logotipo (Armazenado como BLOB no banco de dados)

Logradouro:

ID (Chave Primária)

ClienteID (Chave Estrangeira referenciando a entidade Cliente)

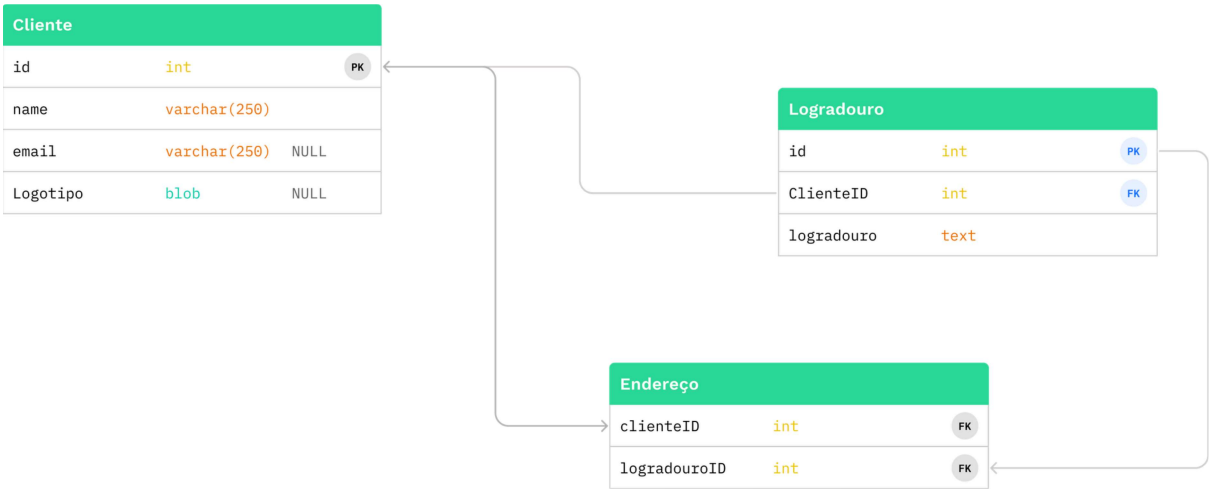
Logradouro (Endereço)

Endereço:

ClienteID (Chave Estrangeira referenciando a entidade Cliente)

LogradouroID (Chave Estrangeira referenciando a entidade Logradouro)

Diagrama de Entidade-Relacionamento (DER)



Descrição das Entidades

- Cliente: Representa os clientes cadastrados na aplicação. Cada cliente possui um ID único, nome, e-mail único e um logotipo em formato de BLOB para representar a imagem da empresa.
- Logradouro: Representa os endereços associados a cada cliente. Cada logradouro possui um ID único, um ID de cliente que faz referência ao cliente proprietário e o endereço em si.
- Endereço: Representa o relacionamento entre Cliente e Logradouro.

Arquitetura de Microsserviços

- **Serviço de Cliente:**

Responsável por lidar com todas as operações relacionadas aos clientes, incluindo cadastro, atualização, visualização e remoção de clientes.

Implementado como um serviço independente, permitindo escalabilidade e evolução independentes.

Comunica-se com o serviço de Logradouro para gerenciar os endereços dos clientes.

- **Serviço de Logradouro:**

Responsável por lidar com todas as operações relacionadas aos logradouros, incluindo cadastro, atualização, visualização e remoção de logradouros.

Implementado como um serviço independente para modularidade e reuso.

- **Serviço de Armazenamento de Arquivos (File Storage Service):**

Responsável por armazenar e recuperar os arquivos, como os logotipos dos clientes.

Implementado separadamente para garantir a escalabilidade e disponibilidade dos arquivos.

Pode ser integrado com serviços de armazenamento em nuvem (por exemplo, AWS S3, Azure Blob Storage).

- **Arquitetura de Cada Serviço**

Cada serviço seguirá uma arquitetura em camadas, consistindo de:

- **Camada de Apresentação:**

  Responsável por lidar com as requisições HTTP, validar os dados de entrada e formatar as respostas da API.

  Implementada utilizando controllers, que recebem as requisições HTTP e chamam os serviços apropriados na camada de aplicação.

- **Camada de Aplicação:**

  Encarregada de coordenar as ações solicitadas pelos clientes, aplicando as regras de negócio da aplicação.

  Implementa a lógica de negócio e faz chamadas para a camada de acesso a dados para realizar operações no banco de dados.

- **Camada de Acesso a Dados (Data Access Layer):**

  Responsável por realizar operações de leitura e escrita no banco de dados.

  Utiliza um ORM (Object-Relational Mapping) para mapear objetos para as tabelas do banco de dados e realizar consultas.

Migrações e Gerenciamento de Esquema:

O Entity Framework Core oferece suporte a migrações de banco de dados, o que permite que você mantenha o esquema do banco de dados em sincronia com o modelo de dados conforme sua aplicação evolui. Isso facilita a implementação de alterações de esquema de forma controlada e automática.

Segurança:

O SQL Server oferece recursos avançados de segurança, incluindo autenticação baseada em conta, criptografia de dados, controle de acesso granular e auditoria. Esses recursos serão utilizados para garantir a segurança dos dados da aplicação e o cumprimento das regulamentações de proteção de dados.

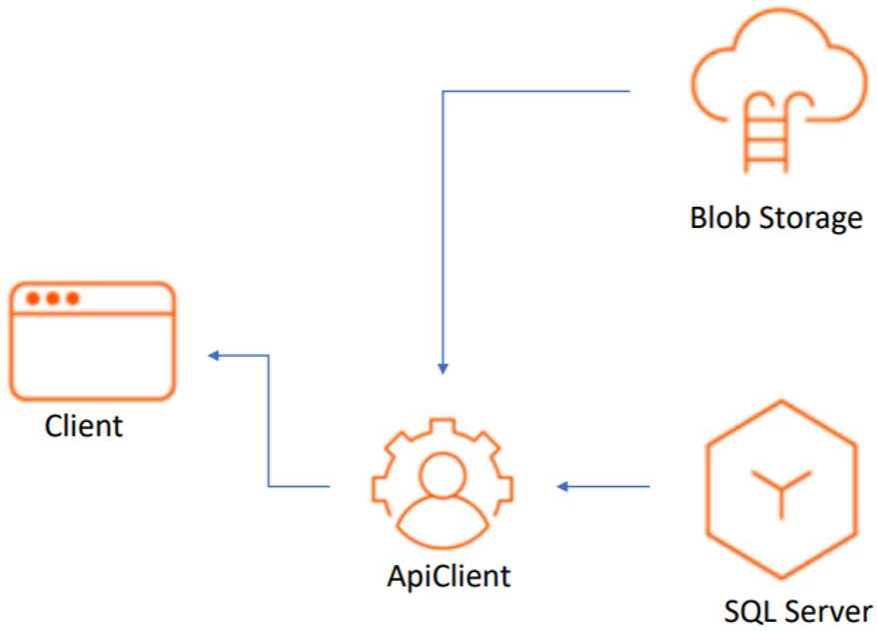
Comunicação entre Serviços

A comunicação entre os serviços será realizada através de chamadas HTTP utilizando o protocolo REST. Cada serviço expõe uma API RESTful que permite que outros serviços e clientes consumam suas funcionalidades.

Tecnologias Utilizadas

- Linguagem de Programação: C# (.NET Core 8.0)
- Framework Web: ASP.NET Core
- Banco de Dados: SQL Server 2019 ou superior
- ORM: Entity Framework Core
- Armazenamento de Arquivos: Azure Blob Storage (ou similar)

Fluxograma da Aplicação



Descrição do Fluxo

Interação do Cliente:

- O cliente (usuário) interage com a interface do usuário para realizar operações relacionadas ao cadastro de clientes.

Autenticação:

- Antes de realizar qualquer operação, o cliente é autenticado pelo serviço de autenticação. Se a autenticação for bem-sucedida, o cliente recebe um token JWT que será incluído em todas as requisições subsequentes.

Chamada ao Client Service:

- O cliente realiza chamadas ao serviço de cliente (Client Service) para realizar operações como criar, atualizar, visualizar ou remover clientes.

Processamento das Operações:

- O Client Service processa as operações solicitadas, incluindo validação de dados, aplicação de regras de negócio e chamadas ao banco de dados para persistência dos dados.

Endpoints da API:

Clientes:

```
GET /api/clientes/Listar: Retorna a lista de todos os clientes cadastrados. GET /api/clientes/ObterbyId/{id}: Retorna os detalhes de um cliente específico. POST /api/clientes/Adicionar: Cria um novo cliente. PUT /api/clientes/Editar/{id}: Atualiza os detalhes de um cliente existente. DELETE /api/clientes/Excluir/{id}: Remove um cliente existente.
```

Logradouros:

```
GET /api/clientes/{clienteId}/logradouros: Retorna a lista de logradouros de um cliente específico. GET /api/clientes/{clienteId}/logradouros/{logradouroId}: Retorna os detalhes de um logradouro específico de um cliente. POST /api/clientes/{clienteId}/logradouros: Adiciona um novo logradouro ao cliente. PUT /api/clientes/{clienteId}/logradouros/{logradouroId}: Atualiza os detalhes de um logradouro específico de um cliente. DELETE /api/clientes/{clienteId}/logradouros/{logradouroId}: Remove um logradouro específico de um cliente.
```

Autenticação e Autorização:

A API deve ter autenticação e autorização para proteger os recursos e garantir que apenas usuários autorizados possam acessá-los. Isso pode ser implementado utilizando JWT (JSON Web Tokens) para autenticação baseada em token e atribuição de roles ou claims para autorização de usuários.

Formato de Resposta:

A API deve retornar respostas no formato JSON, seguindo as práticas comuns de design de APIs RESTful. O uso de códigos de status HTTP apropriados (por exemplo, 200 OK, 201 Created, 404 Not Found, etc.) e estruturas de resposta consistentes ajudará na compreensão e integração dos clientes da API.

Documentação:

É fundamental documentar a API para que os desenvolvedores possam entender facilmente como utilizá-la. O uso de ferramentas como o Swagger pode facilitar a criação de documentação interativa que descreva todos os endpoints, parâmetros de requisição, códigos de status e exemplos de resposta.

Testes:

A API deve ser testada de forma abrangente para garantir que ela funcione corretamente em diferentes cenários. Isso inclui testes unitários, testes de integração e testes de aceitação para validar o comportamento da API em relação aos requisitos do projeto.

POC – Getting Started

Instruções para Executar os Projetos

Backend

Pré-requisitos

Certifique-se de ter o SDK do .NET Core instalado em sua máquina. Você pode baixá-lo em dotnet.microsoft.com.

Executando o Backend

- Navegue até o diretório raiz do projeto de backend.
- Abra um terminal ou prompt de comando.

- Execute o seguinte comando para restaurar as dependências do projeto:

```
dotnet restore
```

Após a restauração das dependências, você deve rodar o Banco de Dados SQL Server, através de um container do Docker:

- Certifique-se de que você tem o Docker instalado na sua máquina. Você pode baixá-lo e instalá-lo a partir do site oficial: https://www.docker.com/get-started.
- Navegue até o diretório onde está localizado o seu Dockerfile usando o terminal ou prompt de comando.
- Execute o comando docker build para construir a imagem Docker a partir do Dockerfile. Este comando cria uma imagem Docker com base nas instruções no Dockerfile.

```
docker build -t nome_da_imagem .
```

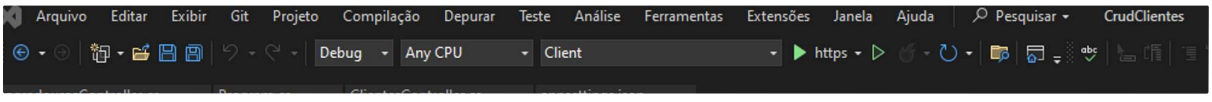
Substitua "nome_da_imagem" pelo nome que você deseja dar à sua imagem Docker.

```
docker run --name nome_do_contêiner -p 1433:1433 nome_da_imagem
```

Substitua "nome_do_contêiner" pelo nome que você deseja dar ao seu contêiner e "nome_da_imagem" pelo nome da imagem que você construiu anteriormente.

O parâmetro -p é usado para mapear uma porta do host para uma porta dentro do contêiner.

Compilando o projeto



Dentro do diretório principal do projeto, você vai abrir a Solution CrudClientes.sln e após rodar, você terá a opção de buildar os projetos "APIClients" e "Client". Após a API estiver rodando, compile também o projeto Client e sua aplicação estará rodando e funcionando para testes.