**Desafio**

Para atender aos requisitos, irei precisar criar um servidor que forneça uma API para realizar operações CRUD nas entidades mencionadas.

Abaixo, escrevi orientações **sobre** como abordar cada uma dessas operações.

### **Diagrama de Sequência**

O diagrama de sequência ilustra a interação entre objetos ao longo do tempo.

No contexto de um CRUD, vou fornecer um exemplo para a criação de uma unidade condominial.

**Explicação:**

* Cliente: Inicia a solicitação de criação de uma unidade condominial enviando uma requisição HTTP POST para o servidor.
* Controlador (Controller): Recebe a requisição e encaminha-a para o serviço adequado.
* Serviço (Service): Processa a requisição, realiza validações e chama o repositório para persistir a nova unidade condominial.
* Repositório (Repository): Responsável por interagir com o banco de dados para criar a nova unidade condominial.
* Banco de Dados: Armazena a nova unidade condominial.

### **Diagrama de Classes**

O diagrama de classes representa as classes no sistema e os relacionamentos entre elas.

**Explicação:**

* Condominio: Representa a entidade "Condominio" com atributos como Id, Nome e uma lista de UnidadeCondominial.
* UnidadeCondominial: Representa a entidade "UnidadeCondominial" com atributos como Id, Numero, ProprietarioId, e uma referência ao Proprietario.
* Proprietario: Representa a entidade "Proprietario" com atributos como Id e Nome.
* **Estrutura do Projeto**

Não esquecer de organizar o projeto em pastas distintas para models, controllers, rotas, etc.

* **Definir Modelos (Entidades)**

Criar classes para representar as entidades do seu sistema (por exemplo, Condominio, UnidadeCondominial, Proprietario, Inquilino, etc.).

* **Modelagem de Dados**

Criar modelos de dados para as entidades (Condominio, UnidadeCondominial, Proprietario, Inquilino).

* **Configuração do Banco de Dados**

Configure o acesso ao banco de dados utilizando o ORM (Object-Relational Mapping).

**OBS.:**

SQL Server, PostgreSQL, MySQL para dados estruturados;

MongoDB para dados não estruturados.

Utilize o Entity Framework Core para configurar e acessar o banco de dados.

Configure seu **DbContext** e as migrações necessárias.

### **Entidades do Sistema**

* **Condomínio:**

Condominio: Representa um condomínio.

Propriedades: Id, Nome, etc.

* **Unidade Condominial:**

UnidadeCondominial: Representa uma unidade dentro do condomínio.

Propriedades: Id, Número, Proprietários (Relacionamento com a entidade Proprietário).

* **Proprietário:**

Proprietario: Representa o proprietário de uma unidade.

Propriedades: Id, Nome, etc.

* **Inquilino:**

Inquilino: Representa o inquilino de uma unidade.

Propriedades: Id, Nome, etc.

* **Implementação da API:**

Implemente as operações CRUD para cada entidade.

### **Operações CRUD**

#### Condomínio:

* Criar (POST): Criar um novo condomínio.
* Consultar (GET): Obter informações sobre um condomínio específico ou todos os condomínios.
* Atualizar (PUT): Atualizar as informações de um condomínio existente.
* Excluir (DELETE): Excluir um condomínio.
* **Criação (POST):**
* Rota: **/condominios**
* Body da requisição: Dados do novo condomínio.
* Lógica: Criar um novo condomínio no banco de dados.
* **Consulta por ID (GET):**
* Rota: **/condominios/{id}**
* Lógica: Obter informações de um condomínio pelo ID.
* **Consulta de Todos (GET):**
* Rota: **/condominios**
* Lógica: Obter informações de todos os condomínios.
* **Atualização (PUT):**
* Rota: **/condominios/{id}**
* Body da requisição: Dados atualizados do condomínio.
* Lógica: Atualizar informações de um condomínio pelo ID.
* **Exclusão (DELETE):**
* Rota: **/condominios/{id}**
* Lógica: Excluir um condomínio pelo ID.

#### Unidade Condominial:

* Criar (POST): Adicionar uma nova unidade condominial a um condomínio.
* Consultar (GET): Obter informações sobre uma unidade específica ou todas as unidades em um condomínio.
* Atualizar (PUT): Atualizar informações de uma unidade condominial existente.
* Excluir (DELETE): Remover uma unidade condominial.
* **Criação (POST):**
* Rota: **/unidades-condominiais**
* Body da requisição: Dados da nova unidade condominial.
* Lógica: Criar uma nova unidade condominial no banco de dados.
* **Consulta por ID (GET):**
* Rota: **/unidades-condominiais/{id}**
* Lógica: Obter informações de uma unidade condominial pelo ID.
* **Consulta de Todos (GET):**
* Rota: **/unidades-condominiais**
* Lógica: Obter informações de todas as unidades condominiais.
* **Atualização (PUT):**
* Rota: **/unidades-condominiais/{id}**
* Body da requisição: Dados atualizados da unidade condominial.
* Lógica: Atualizar informações de uma unidade condominial pelo ID.
* **Exclusão (DELETE):**
* Rota: **/unidades-condominiais/{id}**
* Lógica: Excluir uma unidade condominial pelo ID.

#### Proprietário e Inquilino (Mesmas operações para ambos):

* Proprietário e Inquilino:
* Criar (POST): Adicionar um novo proprietário ou inquilino.
* Consultar (GET): Obter informações sobre um proprietário ou inquilino específico.
* Atualizar (PUT): Atualizar informações de um proprietário ou inquilino existente.
* Excluir (DELETE): Remover um proprietário ou inquilino.
* **Criação (POST):**
* Rota: **/proprietarios** ou **/inquilinos**
* Body da requisição: Dados do novo proprietário ou inquilino.
* Lógica: Criar um novo proprietário ou inquilino no banco de dados.
* **Consulta por ID (GET):**
* Rota: **/proprietarios/{id}** ou **/inquilinos/{id}**
* Lógica: Obter informações de um proprietário ou inquilino pelo ID.
* **Consulta de Todos (GET):**
* Rota: **/proprietarios** ou **/inquilinos**
* Lógica: Obter informações de todos os proprietários ou inquilinos.
* **Atualização (PUT):**
* Rota: **/proprietarios/{id}** ou **/inquilinos/{id}**
* Body da requisição: Dados atualizados do proprietário ou inquilino.
* Lógica: Atualizar informações de um proprietário ou inquilino pelo ID.
* **Exclusão (DELETE):**
* Rota: **/proprietarios/{id}** ou **/inquilinos/{id}**
* Lógica: Excluir um proprietário ou inquilino pelo ID.
* **Relacionamentos**

Um Condominio pode ter várias UnidadeCondominial.

Um Proprietario pode ser proprietário de várias UnidadeCondominial.

Um Inquilino pode alugar várias UnidadeCondominial.

* **Teste e Documentação**

Utilize ferramentas como Postman ou Swagger para testar suas rotas e verificar se as operações CRUD estão funcionando corretamente.

Documente a API para que os usuários compreendam como interagir com ela, isto é fornecer instruções claras no README sobre como configurar e executar o servidor.

* **Segurança**

Considerando medidas de segurança, como autenticação e autorização.

* **Código:**

### **Diagrama de Classes:**

// Entidade Condomínio

public class Condominio

{

public int Id { get; set; }

public string Nome { get; set; }

// Outras propriedades

// Relacionamento com UnidadeCondominial

public List<UnidadeCondominial> UnidadesCondominiais { get; set; }

}

// Entidade Unidade Condominial

public class UnidadeCondominial

{

public int Id { get; set; }

public int Numero { get; set; }

// Outras propriedades

// Relacionamentos

public int ProprietarioId { get; set; }

public Proprietario Proprietario { get; set; }

}

// Entidade Proprietário

public class Proprietario

{

public int Id { get; set; }

public string Nome { get; set; }

// Outras propriedades

// Relacionamento com UnidadeCondominial

public List<UnidadeCondominial> UnidadesCondominiais { get; set; }

}

// Entidade Inquilino

public class Inquilino

{

public int Id { get; set; }

public string Nome { get; set; }

// Outras propriedades

// Relacionamento com UnidadeCondominial

public List<UnidadeCondominial> UnidadesCondominiais { get; set; }

}

**Implementar Controladores:**

Criando controladores para cada entidade com métodos para realizar operações CRUD.

[ApiController]

[Route("api/[controller]")]

public class CondominioController : ControllerBase

{

private readonly ICondominioService \_condominioService;

public CondominioController(ICondominioService condominioService)

{

\_condominioService = condominioService;

}

[HttpGet]

public IActionResult GetAll()

{

var condominios = \_condominioService.GetAll();

return Ok(condominios);

}

[HttpGet("{id}")]

public IActionResult GetById(int id)

{

var condominio = \_condominioService.GetById(id);

if (condominio == null)

{

return NotFound();

}

return Ok(condominio);

}

[HttpPost]

public IActionResult Create([FromBody] Condominio condominio)

{

\_condominioService.Add(condominio);

return CreatedAtAction(nameof(GetById), new { id = condominio.Id }, condominio);

}

[HttpPut("{id}")]

public IActionResult Update(int id, [FromBody] Condominio condominio)

{

var existingCondominio = \_condominioService.GetById(id);

if (existingCondominio == null)

{

return NotFound();

}

\_condominioService.Update(condominio);

return NoContent();

}

[HttpDelete("{id}")]

public IActionResult Delete(int id)

{

var condominio = \_condominioService.GetById(id);

if (condominio == null)

{

return NotFound();

}

\_condominioService.Delete(condominio);

return NoContent();

}

}

### **Estrutura de Dados:**

**Código de Serviço:**

public class CondominioService

{

private readonly DbContext \_context;

public CondominioService(DbContext context)

{

\_context = context;

}

// Operações CRUD

public Condominio CriarCondominio(Condominio condominio)

{

\_context.Condominios.Add(condominio);

\_context.SaveChanges();

return condominio;

}

public Condominio ObterCondominioPorId(int id)

{

return \_context.Condominios.Find(id);

}

public List<Condominio> ObterTodosCondominios()

{

return \_context.Condominios.ToList();

}

public Condominio AtualizarCondominio(Condominio condominio)

{

\_context.Condominios.Update(condominio);

\_context.SaveChanges();

return condominio;

}

public void ExcluirCondominio(int id)

{

var condominio = \_context.Condominios.Find(id);

if (condominio != null)

{

\_context.Condominios.Remove(condominio);

\_context.SaveChanges();

}

}

}

### **Serviços e Repositórios:**

// Repositório Genérico

public interface IRepository<T>

{

T GetById(int id);

IEnumerable<T> GetAll();

void Add(T entity);

void Update(T entity);

void Delete(T entity);

}

// Repositório Condominio

public interface ICondominioRepository : IRepository<Condominio>

{

// Métodos específicos para Condominio, se necessário

}

public class CondominioRepository : ICondominioRepository

{

// Implementação do Repositório

}

// Serviço Condominio

public interface ICondominioService

{

Condominio GetById(int id);

IEnumerable<Condominio> GetAll();

void Add(Condominio condominio);

void Update(Condominio condominio);

void Delete(Condominio condominio);

}

public class CondominioService : ICondominioService

{

private readonly ICondominioRepository \_repository;

public CondominioService(ICondominioRepository repository)

{

\_repository = repository;

}

// Implementação dos métodos do serviço

}