

UNIVERSIDADE PAULISTA - UNIP EaD
Projeto Integrado Multidisciplinar
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Deivysson Gomes da Silva – 2307785

MECANISMO DE ACESSO A UM TRECHO DO BANCO DE DADOS DE UM
SISTEMA EM C# COM PROTÓTIPOS EM ASP.NET E ANDROID

Brasília/DF

2023

Deivysson Gomes da Silva - 2307785

MECANISMO DE ACESSO A UM TRECHO DO BANCO DE DADOS DE UM
SISTEMA EM C# COM PROTÓTIPOS EM ASP.NET E ANDROID

Aluno: Deivysson Gomes da Silva

RA: 2307785

Curso: Análise e Desenvolvimento de Sistemas

Semestre: 4º

Brasilia/DF

2023

RESUMO

Neste projeto acadêmico desenvolveremos um sistema para acessar um banco de dados criado no MySQL utilizando a linguagem de programação C#. Além disso, criaremos dois protótipos de interface com o usuário: um em ASP.Net e outro em Android. Essas interfaces permitirão que o usuário interaja com os dados armazenados no banco de dados. Abordaremos também o conceito de Classes e Objetos, seguindo a arquitetura MVC (Model View Controller), em que cada camada terá suas responsabilidades específicas dentro do projeto.

Palavras-chave: Mecanismo de acesso. C#. MySQL. ASP.Net. Android. MVC.

ABSTRACT

In this academic project we will develop a system to access a database created in MySQL using the C# programming language. In addition, we will create two user interface prototypes: one in ASP.Net and the other in Android. These interfaces will allow the user to interact with the data stored in the database. We will also address the concept of Classes and Objects, following the MVC (Model View Controller) architecture, in which each layer will have its specific responsibilities within the project.

Keywords: Access mechanism. C#. MySQL. ASP.Net. Android. MVC.

SUMÁRIO

INTRODUÇÃO	5
1. ESPECIFICAÇÕES DO PROJETO.....	6
1.1. Arquitetura MVC.....	6
1.2. Linguagem C#	6
1.3. Camada Controller.....	6
1.4. Classe PessoaController	7
1.5. Camada DAL.....	9
1.6. Camada Model.....	13
1.7. Objeto Pessoa	14
1.8. Camadas View	15
2. BANCO DE DADOS	16
3. PROTÓTIPO EM ASP.NET.....	17
4. PROTÓTIPO EM ANDROID.....	19
CONCLUSÃO.....	20
REFERÊNCIAS	21

INTRODUÇÃO

Nesta iniciativa, estaremos criando um mecanismo de acesso a um banco de dados de um sistema implementado em C#. Essa porção do banco de dados terá a finalidade de manter o registro das informações das pessoas no sistema. O mencionado mecanismo desempenhará as funções básicas de um banco de dados, conhecidas como CRUD.

Existem variadas abordagens para a implementação dessa funcionalidade, porém, neste projeto, optaremos por desenvolver a aplicação utilizando o conceito de Orientação a Objetos, em conformidade com a arquitetura MVC. Com tal estrutura, cada camada terá suas responsabilidades bem definidas, garantindo que a camada de acesso ao banco de dados tenha a tarefa exclusiva de realizar as operações de acesso.

Adicionalmente, serão criados protótipos de interfaces gráficas para o usuário, utilizando ASP.Net e Android. Essas interfaces possibilitarão ao usuário a realização das operações (CRUD) na mencionada seção do banco de dados.

1. ESPECIFICAÇÕES DO PROJETO

O projeto será implementado utilizando C# e contará com um sistema capaz de se conectar a um banco de dados MySQL. Esse sistema terá a funcionalidade de executar as operações CRUD (Create, Read, Update and Delete) e será compatível com as interfaces ASP.Net e Android, as quais terão acesso e interação com essa parte específica do banco de dados.

1.1. Arquitetura MVC

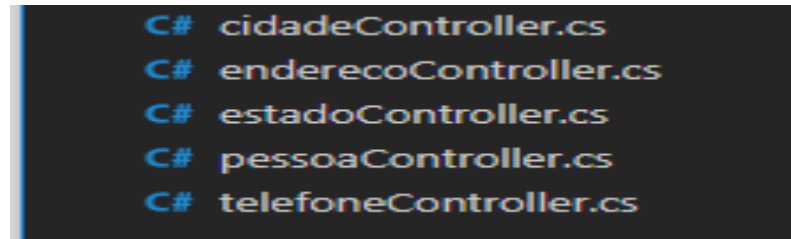
O sistema adotará a arquitetura MVC (Model View Controller). Essa arquitetura possibilita o desenvolvimento em camadas distintas, cada uma com suas próprias responsabilidades. Na ilustração 01, apresentamos a estrutura MVC do sistema.

1.2. Linguagem C#

O sistema foi implementado com o uso da linguagem de programação C#, que é uma linguagem fortemente tipada e multiparadigma. Essa linguagem viabiliza o desenvolvimento do sistema por meio do conceito de programação orientada a objetos, além de facilitar a interação entre as camadas do sistema

1.3. Camada Controller

De acordo com a arquitetura MVC, é essencial criar uma camada chamada Controller. Cada objeto presente na classe Model deve ter uma classe correspondente nessa camada, seguindo o padrão de nomenclatura: o nome da classe, seguido pela palavra "Controller", como ilustrado na Figura 01.



```
C# cidadeController.cs
C# enderecoController.cs
C# estadoController.cs
C# pessoaController.cs
C# telefoneController.cs
```

Figura 01. Estrutura da Camada Controller.

Fonte: Autor.

Cada classe na camada Controller desempenha um papel fundamental ao receber um objeto, executar as validações necessárias e estabelecer a comunicação com a camada de acesso ao banco de dados. Essa camada atua como uma ponte entre as interfaces de usuário e a lógica de negócio, garantindo que as informações sejam devidamente validadas e persistidas no banco de dados.

A presença de propriedades em uma camada Controller pode variar, no entanto, é certo que ela sempre conterá métodos. Esses métodos podem ser utilizados para validar as propriedades do objeto em si, bem como para validar as regras de negócio. Além disso, os métodos da camada Controller podem envolver a troca de informações com o banco de dados, seja para enviar ou receber dados. Por fim, é comum que uma camada Controller possua um método construtor.

1.4. Classe PessoaController

A classe PessoaController desempenha um papel importante como uma das classes controladoras no sistema. Na Figura 02, é apresentada a estrutura do método Gravar. Esse método é responsável por receber um objeto Pessoa como parâmetro, iniciar um bloco Try para capturar possíveis erros durante a execução e preparar uma instrução SQL que será enviada para a camada de acesso ao banco de dados.

Caso não ocorra nenhum erro, seja de manipulação ou execução no banco de dados, o método retornará um valor booleano verdadeiro, indicando à camada solicitante que a execução foi concluída com êxito.


```

1  using System;
2  using System.Data.SqlClient;
3
4  namespace Controller
5  {
6      public class PessoaController
7      {
8          // ...
9
10         public bool Gravar(Pessoa pessoa)
11         {
12             try
13             {
14                 // Preparar instrução SQL para gravar os dados no banco de dados
15                 string sql = "INSERT INTO TabelaPessoa (Nome, CPF) VALUES (@Nome, @CPF)";
16
17                 using (SqlConnection connection = new SqlConnection("ConnectionString"))
18                 {
19                     connection.Open();
20
21                     using (SqlCommand command = new SqlCommand(sql, connection))
22                     {
23                         // Adicionar os parâmetros da instrução SQL
24                         command.Parameters.AddWithValue("@Nome", pessoa.Nome);
25                         command.Parameters.AddWithValue("@CPF", pessoa.CPF);
26
27                         // Executar a instrução SQL
28                         command.ExecuteNonQuery();
29
30                         return true; // Operação realizada com sucesso
31                     }
32                 }
33             }
34             catch (Exception ex)
35             {
36                 // Tratar exceção conforme necessário
37                 Console.WriteLine("Ocorreu um erro ao gravar os dados: " + ex.Message);
38                 return false; // Ocorreu um erro durante a operação
39             }
40         }
41
42         // ...
43     }
44 }
45

```

Figura 02. Classe PessoaController.

Fonte: Autor.

Na Figura 03, observa-se que, para instanciar a classe PessoaController, é necessário que o objeto Pessoa seja previamente instanciado e passado como parâmetro para o método construtor da classe. Essa abordagem permite que a classe PessoaController tenha acesso aos dados e comportamentos do objeto Pessoa, estabelecendo a conexão necessária para sua funcionalidade no sistema.

```

1  using System;
2
3  namespace gravarPessoa
4  {
5      public class Pessoa
6      {
7          public string Nome { get; set; }
8          public string CPF { get; set; }
9
10         // Outras propriedades e métodos da classe Pessoa...
11     }
12
13     public class PessoaController
14     {
15         private Pessoa pessoa;
16
17         // Construtor da classe PessoaController que recebe um objeto Pessoa como parâmetro
18         public PessoaController(Pessoa pessoa)
19         {
20             this.pessoa = pessoa;
21         }
22
23         // Outros métodos da classe PessoaController...
24         public void ExibirInformacoes()
25         {
26             Console.WriteLine($"Nome: {pessoa.Nome}");
27             Console.WriteLine($"CPF: {pessoa.CPF}");
28         }
29     }
30
31     public class Program
32     {
33         public static void Main()
34         {
35             // Instancia um objeto Pessoa
36             Pessoa pessoa = new Pessoa();
37             pessoa.Nome = "João";
38             pessoa.CPF = "123456789";
39
40             // Instancia um objeto PessoaController, passando o objeto Pessoa como parâmetro no construtor
41             PessoaController controller = new PessoaController(pessoa);
42
43             // Chama um método do PessoaController para exibir as informações da pessoa
44             controller.ExibirInformacoes();
45
46             // Outras operações com o objeto Pessoa ou PessoaController...
47         }
48     }
49 }
50

```

Figura 03. Instanciação da classe PessoaController.

Fonte: Autor.

1.5. Camada DAL

A camada Data Access Layer tem a responsabilidade de lidar com a comunicação entre o sistema e o banco de dados. Somente essa camada possui os mecanismos necessários para realizar essa interação.

No contexto deste sistema, utilizaremos uma única classe chamada Connection na camada DAL. Essa classe terá objetos estáticos relacionados à conexão com o banco de dados, como MySqlConnection, MySqlCommand, MySqlDataAdapter, entre outros. Além disso, a classe Connection também contará com métodos importantes, como Connect, Execute e Disconnect, conforme apresentado na Figura 04. Esses métodos desempenham um papel essencial na execução das operações de conexão, execução de comandos e desconexão com o banco de dados.

```

1  using MySql.Data.MySqlClient;
2
3  namespace ExemploASPNET
4  {
5      public class Connection
6      {
7          private static MySqlConnection connection;
8          private static MySqlCommand command;
9          private static MySqlDataAdapter adapter;
10
11         // Método para conectar ao banco de dados
12         public static void Connect(string connectionString)
13         {
14             connection = new MySqlConnection(connectionString);
15             connection.Open();
16
17             command = new MySqlCommand();
18             command.Connection = connection;
19
20             adapter = new MySqlDataAdapter();
21         }
22
23         // Método para executar uma consulta ou comando no banco de dados
24         public static void Execute(string query)
25         {
26             command.CommandText = query;
27             command.ExecuteNonQuery();
28         }
29
30         // Método para executar uma consulta no banco de dados e retornar um MySqlDataReader
31         public static MySqlDataReader ExecuteReader(string query)
32         {
33             command.CommandText = query;
34             return command.ExecuteReader();
35         }
36
37         // Método para executar uma consulta no banco de dados e retornar um DataTable
38         public static DataTable ExecuteQuery(string query)
39         {
40             command.CommandText = query;
41
42             DataTable dataTable = new DataTable();
43             adapter.SelectCommand = command;
44             adapter.Fill(dataTable);
45
46             return dataTable;
47         }
48
49         // Método para desconectar do banco de dados
50         public static void Disconnect()
51         {
52             if (connection != null && connection.State == ConnectionState.Open)
53             {
54                 connection.Close();
55                 connection.Dispose();
56                 connection = null;
57             }
58         }
59     }

```

Figura 04. Camada DAL.

Fonte: Autor.

Na Figura 05, encontramos o método Execute. Esse método desempenha a função de receber uma ou várias instruções SQL, estabelecer uma nova conexão com o banco de dados, preparar os objetos de transação, preencher os comandos e executá-los. Se ocorrer algum erro durante a execução dessa sequência, devido à utilização do objeto Transaction, será realizado um RollBack, desfazendo as alterações feitas na sessão atual. No entanto, se tudo ocorrer sem problemas, será executado um Commit, efetivando assim as alterações no banco de dados.

```

1  using System;
2  using System.Configuration;
3  using System.Data;
4  using System.Data.SqlClient;
5
6  namespace ExemploASPNET
7  {
8      public class DataHelper
9      {
10         private string connectionString;
11         private SqlConnection connection;
12         private SqlTransaction transaction;
13
14         public void Connect()
15         {
16             connectionString = ConfigurationManager.ConnectionStrings["ConnectionStringName"].ConnectionString;
17             connection = new SqlConnection(connectionString);
18             connection.Open();
19             transaction = connection.BeginTransaction();
20         }
21
22         public void Disconnect()
23         {
24             if (transaction != null)
25             {
26                 transaction.Rollback();
27                 transaction.Dispose();
28                 transaction = null;
29             }
30             if (connection != null && connection.State == ConnectionState.Open)
31             {
32                 connection.Close();
33                 connection.Dispose();
34                 connection = null;
35             }
36         }
37         public void ExecuteRollback()
38         {
39             try
40             {
41                 Connect();
42
43                 throw new Exception("Ocorreu um erro durante a transação.");
44             }
45             catch (Exception ex)
46             {
47                 Console.WriteLine("Ocorreu um erro: " + ex.Message);
48                 transaction.Rollback();
49             }
50             finally
51             {
52                 Disconnect();
53             }
54         }
55     }
56 }

```

Figura 05. Método Execute.

Fonte: Autor.

Na Figura 06, encontramos o método Disconnect, que tem a finalidade de encerrar a conexão com o banco de dados e destruir todos os objetos criados para liberar memória e evitar que a sessão permaneça aberta no banco de dados.

```

1  using System;
2  using System.Configuration;
3  using System.Data;
4  using System.Data.SqlClient;
5
6  namespace ExemploASPNET
7  {
8      public class DataHelper
9      {
10         private string connectionString;
11         private SqlConnection connection;
12         public void Connect()
13         {
14             connectionString = ConfigurationManager.ConnectionStrings["ConnectionStringName"].ConnectionString;
15             connection = new SqlConnection(connectionString);
16             connection.Open();
17         }
18         public void Disconnect()
19         {
20             if (connection != null && connection.State == ConnectionState.Open)
21             {
22                 connection.Close();
23                 connection.Dispose();
24                 connection = null;
25             }
26         }
27         public DataTable GetDataTable(string tableName)
28         {
29             DataTable dataTable = new DataTable();
30             try
31             {
32                 Connect();
33                 string query = $"SELECT * FROM {tableName}";
34                 SqlCommand command = new SqlCommand(query, connection);
35
36                 SqlDataAdapter adapter = new SqlDataAdapter(command);
37
38                 adapter.Fill(dataTable);
39             }
40             catch (Exception ex)
41             {
42                 Console.WriteLine("Ocorreu um erro: " + ex.Message);
43             }
44             finally
45             {
46                 Disconnect();
47             }
48             return dataTable;
49         }
50     }
51 }
52
53

```

Figura 06. Método Disconnect.

Fonte: Autor.

Na Figura 07, encontramos o método denominado GetDataTable. Essa função é responsável por preencher um objeto do tipo DataTable. Ao ser invocado, recebendo como parâmetro uma instrução SQL, o método retornará uma tabela contendo os registros solicitados ao banco de dados. Essa abordagem permite obter os dados necessários de forma organizada e estruturada para posterior manipulação ou exibição.

```

1  using System;
2  using System.Configuration;
3  using System.Data;
4  using System.Data.SqlClient;
5
6  namespace ExemploASPNET
7  {
8      public class DataHelper
9      {
10         public DataTable GetDataTable(string tableName)
11         {
12             string connectionString = ConfigurationManager.ConnectionStrings["ConnectionStringName"].ConnectionString;
13
14             using (SqlConnection connection = new SqlConnection(connectionString))
15             {
16                 DataTable dataTable = new DataTable();
17
18                 try
19                 {
20                     connection.Open();
21
22                     string query = $"SELECT * FROM {tableName}";
23                     SqlCommand command = new SqlCommand(query, connection);
24                     SqlDataAdapter adapter = new SqlDataAdapter(command);
25                     adapter.Fill(dataTable);
26                 }
27                 catch (Exception ex)
28                 {
29                     Console.WriteLine("Ocorreu um erro: " + ex.Message);
30                 }
31                 return dataTable;
32             }
33         }
34     }
35 }

```

Figura 07. Método GetDataTable.

Fonte: Autor.

1.6. Camada Model

A camada Model (ou modelo) tem como responsabilidade principal representar os objetos do mundo real. Essa camada deve incluir todos os objetos presentes no projeto, cada um deles com suas respectivas propriedades. A Figura 08 ilustra a estrutura da camada Model, exibindo a organização dos objetos e suas inter-relações.

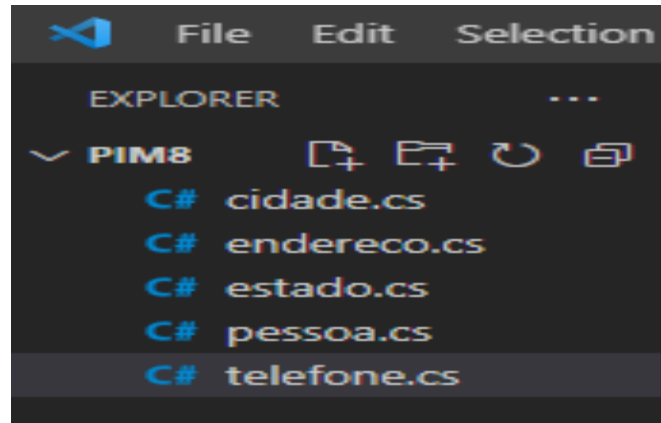


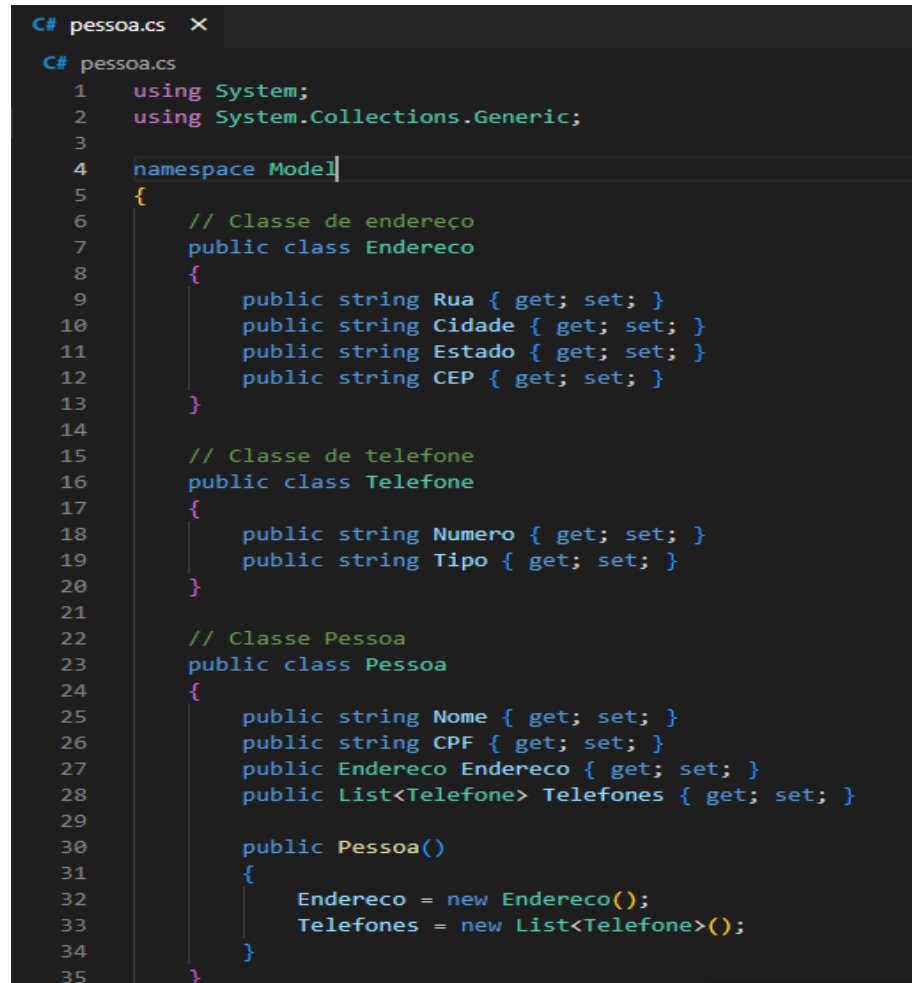
Figura 08. Estrutura da Camada Model.

Fonte: Autor.

1.7. Objeto Pessoa

Cada classe representa a abstração de um objeto do mundo real e tem a responsabilidade de encapsular seus comportamentos e propriedades. Com base nessas classes, os objetos são criados em tempo de execução.

No contexto do nosso projeto, a classe Pessoa é a principal. Conforme ilustrado na Figura 09, ela possui suas próprias propriedades, como Nome e CPF, e também propriedades relacionadas a outras classes, como Endereço e uma lista de Telefones. Essa estrutura de propriedades permite que a classe Pessoa armazene e manipule as informações relevantes ao objeto, fornecendo uma representação completa dos dados associados a uma pessoa no sistema.



```

C# pessoa.cs x
C# pessoa.cs
1  using System;
2  using System.Collections.Generic;
3
4  namespace Model
5  {
6      // Classe de endereço
7      public class Endereco
8      {
9          public string Rua { get; set; }
10         public string Cidade { get; set; }
11         public string Estado { get; set; }
12         public string CEP { get; set; }
13     }
14
15     // Classe de telefone
16     public class Telefone
17     {
18         public string Numero { get; set; }
19         public string Tipo { get; set; }
20     }
21
22     // Classe Pessoa
23     public class Pessoa
24     {
25         public string Nome { get; set; }
26         public string CPF { get; set; }
27         public Endereco Endereco { get; set; }
28         public List<Telefone> Telefones { get; set; }
29
30         public Pessoa()
31         {
32             Endereco = new Endereco();
33             Telefones = new List<Telefone>();
34         }
35     }

```

Figura 09. Objeto Pessoa

Fonte: Própria.

1.8. Camadas View

A camada View é responsável por permitir a interação do usuário com a aplicação. Conforme definição, essa camada não deve conter validações ou regras de negócio; todas essas responsabilidades são atribuídas às camadas internas, como Model ou Controller, dependendo do padrão adotado.

De acordo com esse conceito, um projeto MVC pode ter uma ou mais camadas View, sendo a diferença determinada pela linguagem e plataforma em que a camada será executada. Devido à estrutura lógica bem separada nas demais camadas, não é necessário realizar implementações adicionais para validação ou gravação desses objetos no banco de dados.

2. BANCO DE DADOS

O componente de acesso, implementado em C#, será responsável por fornecer acesso a um segmento específico do banco de dados para o restante do sistema. Sempre que uma parte do sistema precisar interagir com esse segmento do banco de dados, deverá fazê-lo por meio desse componente previamente desenvolvido.

Para o desenvolvimento do banco de dados, utilizamos o programa MySQL Workbench 8.0.33.

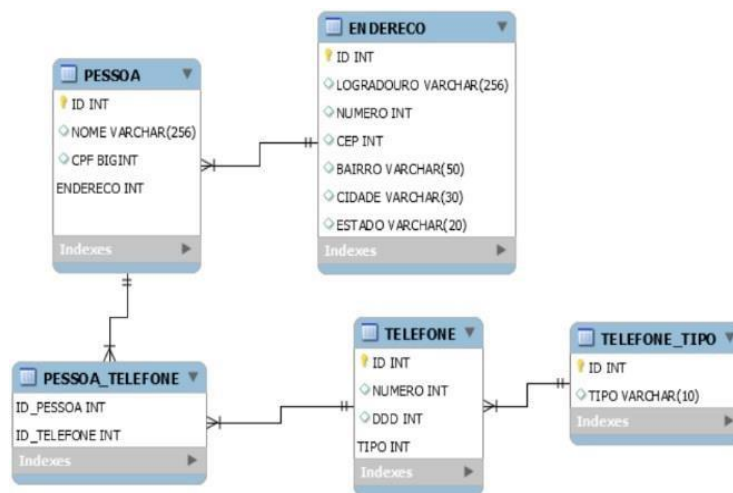


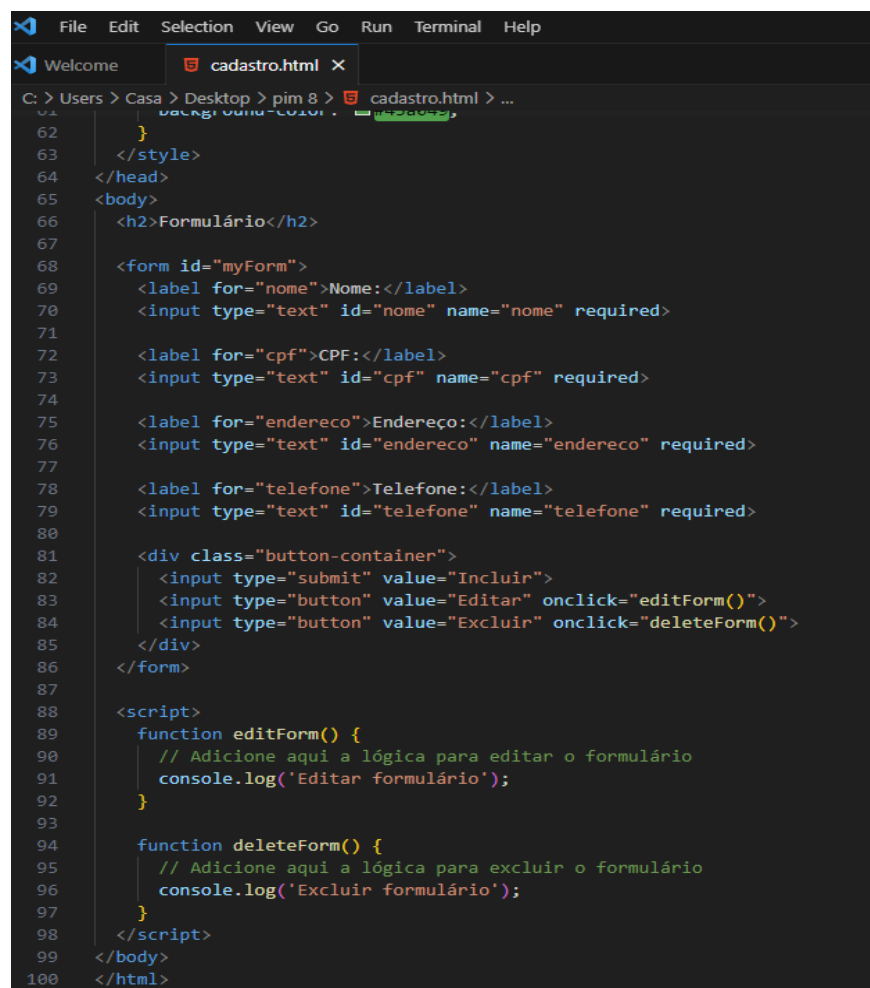
Figura 10. Diagrama de Cadastro.

Fonte: Autor.

3. PROTÓTIPO EM ASP.NET

Uma das camadas View do projeto foi implementada utilizando ASP.Net, dentro desse projeto, criamos um protótipo de interface gráfica que oferece ao usuário as funcionalidades CRUD em ASP.Net. Essa interface permite que o usuário interaja com os dados modelados por essa porção do banco de dados.

Na Figura 11, apresentamos uma parte do script HTML da página de cadastro. Esse trecho de código representa a estrutura da página e os elementos utilizados para realizar o cadastro de dados

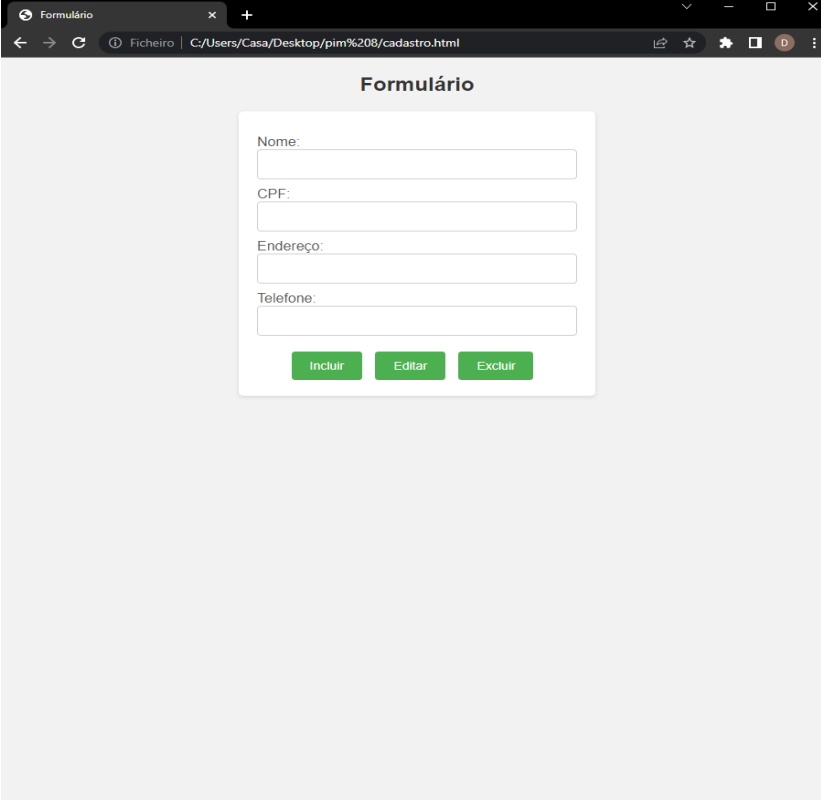


```
61     background-color: #f2f2f2;
62 }
63 </style>
64 </head>
65 <body>
66     <h2>Formulário</h2>
67
68     <form id="myForm">
69         <label for="nome">Nome:</label>
70         <input type="text" id="nome" name="nome" required>
71
72         <label for="cpf">CPF:</label>
73         <input type="text" id="cpf" name="cpf" required>
74
75         <label for="endereco">Endereço:</label>
76         <input type="text" id="endereco" name="endereco" required>
77
78         <label for="telefone">Telefone:</label>
79         <input type="text" id="telefone" name="telefone" required>
80
81         <div class="button-container">
82             <input type="submit" value="Incluir">
83             <input type="button" value="Editar" onclick="editForm()">
84             <input type="button" value="Excluir" onclick="deleteForm()">
85         </div>
86     </form>
87
88     <script>
89         function editForm() {
90             // Adicione aqui a lógica para editar o formulário
91             console.log('Editar formulário');
92         }
93
94         function deleteForm() {
95             // Adicione aqui a lógica para excluir o formulário
96             console.log('Excluir formulário');
97         }
98     </script>
99 </body>
100 </html>
```

Figura 11. Parte do Script HTML da página de cadastro.

Fonte: Autor.

Após a compilação, o projeto será executado por meio do servidor LocalHost, exibindo os campos que foram definidos no HTML, conforme ilustrado na Figura 12. Para organizar o estilo da página, todo o código CSS foi inserido diretamente no elemento Head, utilizando a tag Style. Dentro dessa seção, foram adicionadas as instruções CSS necessárias para estilizar adequadamente a página.



The image shows a web browser window with a single tab titled 'Formulário'. The address bar displays the file path 'C:/Users/Casa/Desktop/pim%208/cadastro.html'. The page content features a central white box with the title 'Formulário' at the top. Below the title are four text input fields, each preceded by a label: 'Nome:', 'CPF:', 'Endereço:', and 'Telefone:'. At the bottom of this box are three green buttons labeled 'Incluir', 'Editar', and 'Excluir'.

Figura 12. Página Cadastro em ASP.Net.

Fonte: Autor.

4. PROTÓTIPO EM ANDROID

Foi implementada utilizando a linguagem Java no Android Studio, uma IDE que permite a criação de layouts por meio de recursos de arrastar e soltar (Drag and Drop), conforme mostrado na Figura 13.

Dentro dessa camada, criamos um protótipo de interface gráfica para dispositivos Android, oferecendo ao usuário as funcionalidades CRUD e permitindo a interação com os dados modelados nessa parte específica do banco de dados. Além disso, na Figura 14, temos uma parte do código XML desse protótipo, que define a estrutura do layout da interface.

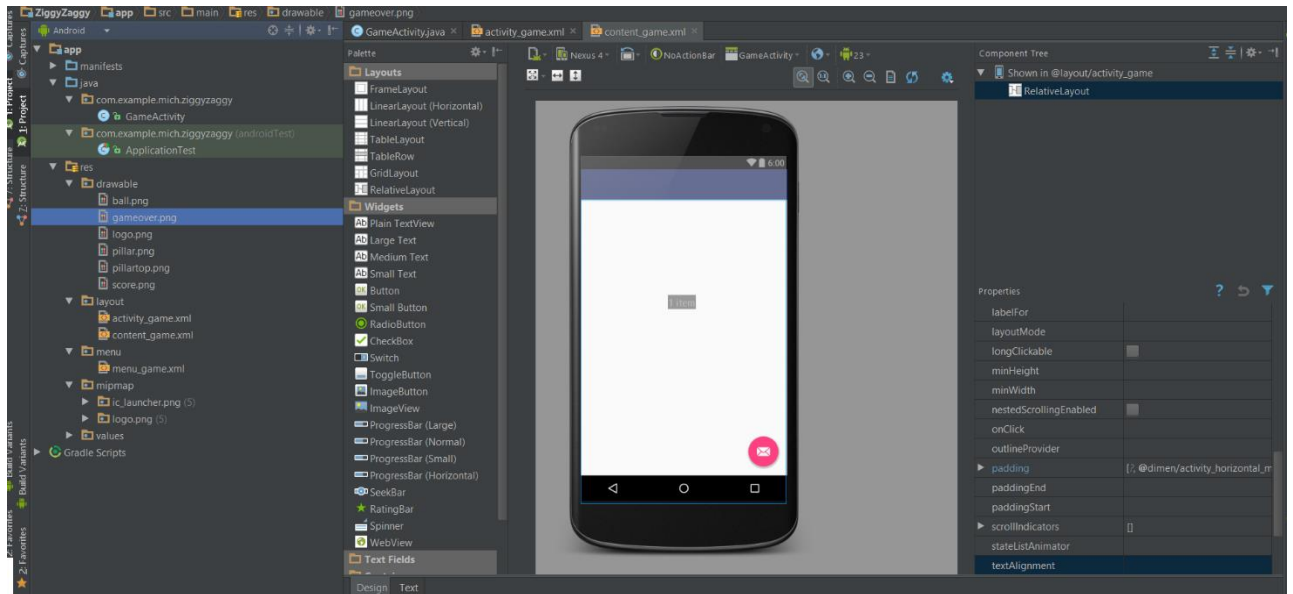


Figura 13. IDE Android Studio.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingTop="16dp"
    android:paddingRight="16dp"
    android:paddingBottom="16dp"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textViewTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Protótipo Android"
        android:textSize="20sp"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="16dp"/>

    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@id/textViewTitle"
        android:layout_marginTop="16dp">
    </ListView>
</RelativeLayout>
```

Figura 14. Parte do código XML do protótipo em Android.

Fonte: Autor.

CONCLUSÃO

Neste projeto, implementamos um mecanismo de acesso a um segmento de um banco de dados. Desenvolvemos esse mecanismo utilizando a linguagem de programação C# e seguindo a arquitetura MVC, com suporte para as operações CRUD. Essa abordagem técnica nos permitiu criar uma estrutura sólida e eficiente para o desenvolvimento de um projeto MVC com camadas bem definidas, incluindo camadas de negócio, modelagem, acesso ao banco de dados e interface com o usuário.

Para o banco de dados, utilizamos o MySQL Workbench e definimos cuidadosamente a estrutura necessária para que o mecanismo pudesse acessar e manipular com sucesso as informações armazenadas. Isso nos permitiu realizar as operações CRUD de forma consistente e confiável.

A aplicação web foi desenvolvida em ASP.Net utilizando o Visual Studio, proporcionando uma interface simples e responsiva. Com ela, os usuários podem interagir com os dados por meio das operações CRUD, enquanto o sistema realiza as devidas chamadas ao banco de dados.

A aplicação Android, desenvolvida no Android Studio, também oferece uma interface intuitiva e responsiva. Ela permite aos usuários realizar as operações CRUD, enquanto o mecanismo de acesso ao banco de dados é utilizado para garantir a integridade e consistência dos dados.

Em suma, concluímos que o projeto foi executado com sucesso, abrangendo desde a base do sistema em C#, passando pela implementação do banco de dados, até o desenvolvimento de protótipos de interface web e Android. O resultado final é uma aplicação funcional e eficiente, pronta para atender às necessidades dos usuários.

REFERÊNCIAS

- ANDROID STUDIO]. Página inicial. Disponível em: <<https://developer.android.com/studio>>. Acesso em: 30 de abril de 2021.
- [VISUAL STUDIO]. Página Inicial. Disponível em: <<https://visualstudio.microsoft.com>>. Acesso em: 30 de abril de 2021.
- [MYSQL]. Página inicial. Disponível em: <<https://www.mysql.com/>>. Acesso em: 30 de abril de 2021.
- [GUNJI], José Cassiano Grassi. Tópicos especiais de programação orientada a objetos. São Paulo: Editora Sol, 2015.
- [MARINHO], Salatiel Luz. Programação Orientada a Objetos II. São Paulo: Editora Sol, 2015.
- VERSOLATTO, Fábio Rossi. Projeto de Sistemas Orientado a Objetos. São Paulo: Editora Sol, 2015.
- [BRITO, 2020] BRITO, Robison Cris. Android com Android Studio - Passo a Passo. Rio de Janeiro, Ciência Moderna, 2020.
- [DENNIS, 2005] DENNIS, Alan. Análise e Projeto de Sistemas. Rio de Janeiro, LTC, 2005.
- [LOTAR], Alfredo. Como Programar com ASP.NET e C#, 2ª edição. São Paulo: Editora Novatec, 2010.
- [MANZANO, 2015] MANZANO, José Augusto N. G. Estudo dirigido: Microsoft VisualC# community 2015. São Paulo, Editora Érica, 2015.
- [PRESSMAN, 2002] PRESSMAN, R. Engenharia de software. Rio de Janeiro: McGraw-Hill, 2002.