DESENVOLVIMENTO DE SISTEMAS COM C#

Cleverson Lopes Ledur

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Compreender e aplicar o desenvolvimento de aplicações em camadas

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Identificar o conceito de Model-View-Controller (MVC).
- Criar uma aplicação MVC no Visual Studio com C#.
- Organizar a arquitetura MVC em um projeto C#.

Introdução

Os sistemas em camadas são um assunto muito abordado nos últimos anos devido ao aumento da complexidade da criação de software. Essa divisão de sistemas em camadas proporcionou mais facilidade ao trabalho realizado por times de desenvolvimento. Além disso, ofereceu maior reuso de código. Muitos sistemas desenvolvidos atualmente utilizam o padrão MVC, baseado em três componentes básicos: o model, o view e o controller. Cada um com sua responsabilidade, eles agrupam o código necessário apenas para realizar o que é função do componente. Logo, eles reduzem a necessidade de reescrita de código em diferentes partes do sistema e organizam as responsabilidades dos componentes. O entendimento desse padrão e a aplicação dele no desenvolvimento de software é muito importante para engenheiros, analistas e desenvolvedores.

Neste texto, você irá entender melhor o conceito do padrão MVC. Também aprenderá a criar uma aplicação MVC no Visual Studio com C# e Windows Forms. Além disso, entenderá a organização e a arquitetura MVC em um projeto C#.

Introdução ao MVC

O Model-View-Controller (MVC) é um padrão de arquitetura de software para implementar interfaces de usuário em computadores. Ele divide uma determinada aplicação em três partes interligadas. Isso é feito para separar as representações internas de informações das formas como as informações são apresentadas e aceitas pelo usuário. O padrão de design MVC desacopla esses principais componentes, permitindo uma reutilização eficiente de código e desenvolvimento paralelo (FREEMAN, 2013).

Tradicionalmente, essa arquitetura era usada para interfaces de usuário gráficas de área de trabalho (GUIs). Mais tarde, ela se tornou popular para projetar aplicativos web e até mesmo clientes móveis, de desktop e outros. Linguagens de programação populares como Java, C#, Ruby, PHP e outras têm frameworks MVC populares que atualmente estão sendo usados no desenvolvimento de aplicativos da web diretamente fora da caixa.

Saiba mais

Tal como acontece com outras arquiteturas de software, a MVC expressa o "núcleo da solução" para um problema, permitindo que seja adaptado para cada sistema. As arquiteturas MVC particulares podem variar significativamente da descrição tradicional.

Componentes

O padrão MVC é composto por três elementos principais. Esses elementos são o modelo (*model*), a visualização (*view*) e o controlador (*controller*), como você pode ver na Figura 1.

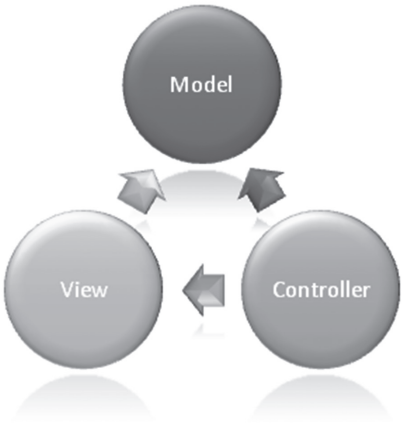

Figura 1. Componentes do padrão MVC.
Fonte: Smith (2016).

Que tal aprender um pouco mais sobre cada um deles? (FAGERBERG, 2015)

- O modelo é o componente central do padrão. Ele expressa o comportamento do aplicativo em termos do domínio do problema, independentemente da interface do usuário. Ele gerencia diretamente os dados, a lógica e as regras da aplicação.
- Uma visualização pode ser qualquer representação de saída de informações, como um gráfico ou um diagrama. Várias visualizações da mesma informação são possíveis, como um gráfico de barras para gerenciamento e uma visão tabular para contabilistas.
- A terceira parte ou seção, o controlador, aceita entrada e a converte em comandos para o modelo ou visualização.

Interações

Além de dividir o aplicativo em três tipos de componentes, o model-view-controller define as interações entre eles. Observe:

- Um modelo armazena dados que são recuperados de acordo com os comandos do controlador e exibidos na visualização.
- Uma visualização gera uma nova saída para o usuário com base em mudanças no modelo.

- Um controlador pode enviar comandos para o modelo a fim de atualizar o estado desse modelo (por exemplo, edição de um documento). Ele também pode enviar comandos para sua exibição associada a fim de alterar a apresentação da visão do modelo (por exemplo, rolar por meio de um documento, movimento de documento).

Vantagens do uso do MVC

A seguir, você pode ver as principais vantagens relacionadas à utilização do padrão model-view-controller.

- Desenvolvimento simultâneo: múltiplos desenvolvedores podem trabalhar simultaneamente no modelo, no controlador e nas visualizações (MICROSOFT, 2017).
- Alta coesão: o padrão MVC permite o agrupamento lógico de ações relacionadas em um controlador em conjunto. As visualizações para um modelo específico também são agrupadas.
- Acoplamento baixo: a própria natureza da estrutura MVC é tal que existe baixo acoplamento entre modelos, visualizações ou controladores.
- Facilidade de modificação: devido à separação de responsabilidades, o desenvolvimento ou a modificação futura é mais fácil.
- Múltiplas visualizações para um modelo: os modelos podem ter várias visualizações.

Desvantagens do uso do MVC

A seguir, você pode ver as principais desvantagens relacionadas à utilização do padrão model-view-controller.

- Navegabilidade do código: a navegação da estrutura pode ser complexa, porque introduz novas camadas de abstração e exige que os usuários se adaptem aos critérios de decomposição do MVC.
- Consistência de multiartefato: a decomposição de um recurso em três artefatos causa dispersão. Assim, exige que os desenvolvedores mantenham a consistência de múltiplas representações ao mesmo tempo.
- Curva de aprendizado pronunciada: o conhecimento em múltiplas tecnologias se torna necessário. Os desenvolvedores que usam MVC precisam ser especializados em várias tecnologias.

Objetivos ao utilizar MVC

Desenvolvimento em times: como o MVC desacopla os vários componentes de um aplicativo, os desenvolvedores podem trabalhar em paralelo em diferentes componentes sem afetar ou bloquear os outros. Por exemplo, uma equipe pode dividir seus desenvolvedores entre o front-end e o back-end. Os desenvolvedores de back-end podem projetar a estrutura dos dados e, como o usuário interage com ele, não precisam exigir a conclusão da interface do usuário. Por outro lado, os desenvolvedores front-end são capazes de projetar e testar o layout do aplicativo antes que a estrutura de dados esteja disponível.

Reutilização de código: ao criar componentes independentes um do outro, os desenvolvedores podem reutilizar componentes de forma rápida e fácil em outras aplicações. A mesma visão (ou similar) para um aplicativo pode ser refaturada para outro aplicativo com dados diferentes. Isso ocorre porque a visualização é simplesmente o modo como os dados são exibidos para o usuário.

Criando uma aplicação MVC

Nesta seção, você vai aprender a criar uma aplicação MVC no Visual Studio. Essa aplicação será um cadastro de funcionários. Nele, haverá uma entidade chamada `Funcionario` com nome, sobrenome, sexo, departamento e ID. Para isso, você precisa criar um projeto Windows Forms. Veja, no exemplo a seguir, o código da classe `Funcionario`.

```
public class Funcionario
    {
    public enum SexoPessoa
        {
            M   = 1,
            F   = 2
        }
    private string nome;
    private string sobrenome;
    private string id;
    private string departamento;

}
```

Para fins de organização do código, inicialmente você deve criar uma interface que o `controller` irá implementar. Essa interface deve ficar no diretório `controller`. Veja o código a seguir.

```
using System;
using WinFormMVC.Model;

namespace WinFormMVC.Controller
{
    public interface IUsersView
    {
        void SetController(FuncionariosController controller);
        void ClearGrid();
        void AddUserToGrid(Funcionario user);
        void UpdateGridWithChangedUser(Funcionario user);
        void RemoveUserFromGrid(Funcionario user);
        string GetIdOfSelectedUserInGrid();
        void SetSelectedUserInGrid(Funcionario user);

        string Nome     { get; set; }
        string Sobrenome      { get; set; }
        string ID            { get; set; }
        string Departamento    { get; set; }
        Funcionario.SexoPessoa Sexo { get; set; }
        bool CanModifyID     {      set; }
    }
}
```

Bem, agora você pode criar o `controller` com todos os métodos que irão manipular e consultar o modelo. Veja, no exemplo da próxima página, o código-fonte do `controller` com os métodos implementados.

```
using System;
using System.Collections;
using WinFormMVC.Model;

namespace WinFormMVC.Controller
{
    public class FuncionariosController
    {
        IUsersView _view;
        IList      _funcionarios;
        Funcionario       _funcionarioSelecionado;

        public FuncionariosController(IUsersView view, IList funcionarios)
        {
            _view = view;
            _funcionarios = funcionarios;
            view.SetController(this);
        }

        public IList Users
        {
           get { return ArrayList.ReadOnly(_funcionarios); }
        }

        private void updateViewDetailValues(Funcionario usr)
        {
           _view.Nome   =  usr.Nome;
           _view.Sobrenome    =  usr.Sobrenome;
           _view.ID          =  usr.ID;
           _view.Departamento  =  usr.Departamento;
           _view.Sexo         =  usr.Sexo;

        }
```

```
   private void updateUserWithViewValues(Funcionario usr)
   {
      usr.Nome     =  _view.Nome;
      usr.Sobrenome      =  _view.Sobrenome;
      usr.ID            =  _view.ID;
      usr.Departamento    =  _view.Departamento;
      usr.Sexo           =  _view.Sexo;
   }


   public void LoadView()
   {
       _view.ClearGrid();
       foreach (Funcionario usr in _funcionarios)
           _view.AddUserToGrid(usr);

       _view.SetSelectedUserInGrid((Funcionario)_funcionarios[0]);

   }

   public void SelectedUserChanged(string selectedUserId)
   {
       foreach (Funcionario usr in this._funcionarios)
       {
           if (usr.ID == selectedUserId)
           {
               _funcionarioSelecionado = usr;
               updateViewDetailValues(usr);
               _view.SetSelectedUserInGrid(usr);

}

public void SelectedUserChanged(string selectedUserId)
{
    foreach (Funcionario usr in this._funcionarios)
    {
        if (usr.ID == selectedUserId)
        {
            _funcionarioSelecionado = usr;
            updateViewDetailValues(usr);
            _view.SetSelectedUserInGrid(usr);
            this._view.CanModifyID = false;
            break;
        }
    }
}


public void AddNewUser()
{
    _funcionarioSelecionado = new Funcionario(""/*firstname*/,
                              "" /*lastname*/,
                              ""  /*id*/,
                              "" /*department*/,
                              Funcionario.SexoPessoa.M /*sex*/);

    this.updateViewDetailValues(_funcionarioSelecionado);
    this._view.CanModifyID = true;
}

public void RemoverFuncionario()
{
    string id = this._view.GetIdOfSelectedUserInGrid();
    Funcionario userToRemove = null;

    if (id != "")
    {
        foreach (Funcionario usr in this._funcionarios)
        {
```

```
                    if (usr.ID == id)
                    {
                        userToRemove = usr;
                        break;
                    }
                }

                if (userToRemove != null)
                {
                    int newSelectedIndex =
this._funcionarios.IndexOf(userToRemove);
                    this._funcionarios.Remove(userToRemove);
                    this._view.RemoveUserFromGrid(userToRemove);

                    if (newSelectedIndex > -1 && newSelectedIndex <
_funcionarios.Count)
                    {

this._view.SetSelectedUserInGrid((Funcionario)_funcionarios[newSelectedIndex]);
                    }
                }
            }
        }

        public void Salvar()
        {
            updateUserWithViewValues(_funcionarioSelecionado);
            if (!this._funcionarios.Contains(_funcionarioSelecionado))
            {
                // Add new user
                this._funcionarios.Add(_funcionarioSelecionado);
                this._view.AddUserToGrid(_funcionarioSelecionado);
            }
            else
            {
                // Update existing
                this._view.UpdateGridWithChangedUser(_funcionarioSelecionado);
            }
            _view.SetSelectedUserInGrid(_funcionarioSelecionado);
            this._view.CanModifyID = false;

        }
    }
}
```

Por fim, você deve criar a view. Nesse caso, pode utilizar uma janela do Windows Forms para permitir a manipulação dos funcionários. Veja a Figura 2, que mostra a janela (chamada *form*) criada no Windows Forms.

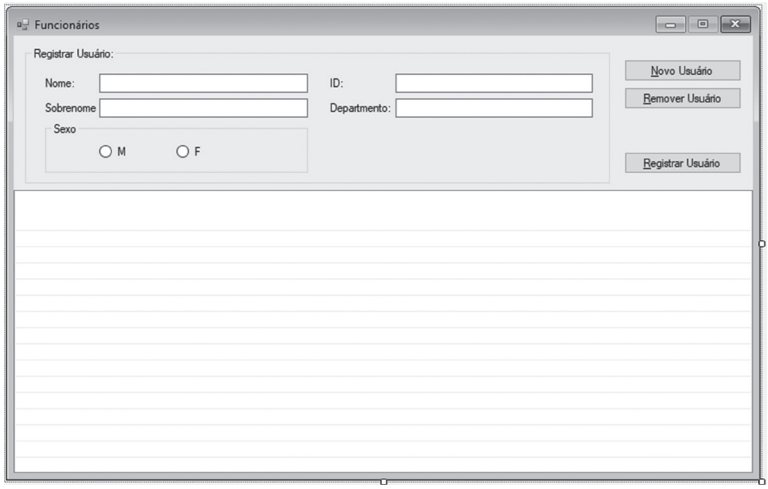

Figura 2. Componentes do padrão MVC.
Fonte: Smith (2016).

Debaixo dessa interface, há uma classe que contém os métodos chamados quando se efetua uma ação na janela. É possível implementar novos métodos nessa classe e modificar o comportamento de acordo com a necessidade. No caso, como você está utilizando MVC, vai chamar os métodos do `controller` sempre que uma ação for realizada. Veja, no exemplo a seguir, os métodos que foram criados e que chamam os métodos do `controller`.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using WinFormMVC.Controller;
using WinFormMVC.Model;

namespace WinFormMVC.View
{
    public partial class FuncionariosView : Form, IUsersView
    {
        public FuncionariosView()
        {
            InitializeComponent();
        }

        FuncionariosController _controller;

        #region Events raised  back to controller

        private void btnAdd_Click(object sender, EventArgs e)
        {
            this._controller.AddNewUser();
        }

        private void btnRemove_Click(object sender, EventArgs e)
        {
            this._controller.RemoverFuncionario();
        }

        private void btnRegister_Click(object sender, EventArgs e)
        {
            this._controller.Salvar();
        }

        private void grdUsers_SelectedIndexChanged(object sender, EventArgs e)
        {
            if (this.grdFuncionarios.SelectedItems.Count > 0)

this._controller.SelectedUserChanged(this.grdFuncionarios.SelectedItems[0].Text);
        }


        #endregion

        #region ICatalogView implementation

        public void SetController(FuncionariosController controller)
        {
            _controller = controller;
        }

        public void AddUserToGrid(Funcionario usr)
        {
```

```
            ListViewItem parent;
            parent = this.grdFuncionarios.Items.Add(usr.ID);
            parent.SubItems.Add(usr.Nome);
            parent.SubItems.Add(usr.Sobrenome);
            parent.SubItems.Add(usr.Departamento);
            parent.SubItems.Add(Enum.GetName(typeof(Funcionario.SexoPessoa),
usr.Sexo));
        }

        public void UpdateGridWithChangedUser(Funcionario usr)
        {
            ListViewItem rowToUpdate = null;

            foreach (ListViewItem row in this.grdFuncionarios.Items)
            {
                if (row.Text == usr.ID)
                {
                    rowToUpdate = row;
                }
            }

            if (rowToUpdate != null)
            {
                rowToUpdate.Text = usr.ID;
                rowToUpdate.SubItems[1].Text = usr.Nome;
                rowToUpdate.SubItems[2].Text = usr.Sobrenome;
                rowToUpdate.SubItems[3].Text = usr.Departamento;
                rowToUpdate.SubItems[4].Text =
Enum.GetName(typeof(Funcionario.SexoPessoa), usr.Sexo);
            }
        }

        public void RemoveUserFromGrid(Funcionario usr)
        {

            ListViewItem rowToRemove = null;

            foreach (ListViewItem row in this.grdFuncionarios.Items)
            {
                if (row.Text == usr.ID)
                {
                    rowToRemove = row;
                }
            }

            if (rowToRemove != null)
            {
                this.grdFuncionarios.Items.Remove(rowToRemove);
                this.grdFuncionarios.Focus();
            }
        }

        public string GetIdOfSelectedUserInGrid()
        {
            if (this.grdFuncionarios.SelectedItems.Count > 0)
                return this.grdFuncionarios.SelectedItems[0].Text;
            else
                return "";
        }
```

```
        public void SetSelectedUserInGrid(Funcionario usr)
        {
            foreach (ListViewItem row in this.grdFuncionarios.Items)
            {
                if (row.Text == usr.ID)
                {
                    row.Selected = true;
                }
            }
        }

        public string Nome
        {
            get { return this.txtFirstName.Text; }
            set { this.txtFirstName.Text = value; }
        }

        public string Sobrenome
        {
            get { return this.txtLastName.Text; }
            set { this.txtLastName.Text = value; }
        }

        public string ID
        {
            get { return this.txtID.Text; }
            set { this.txtID.Text = value; }
        }

        public string Departamento
        {
            get { return this.txtDepartamento.Text; }
            set { this.txtDepartamento.Text = value; }
        }

        public Funcionario.SexoPessoa Sexo
        {
            get
            {
                if (this.rdMale.Checked)
                    return Funcionario.SexoPessoa.M;
                else
                    return Funcionario.SexoPessoa.F;
            }
            set
            {
                if (value == Funcionario.SexoPessoa.M)
                    this.rdMale.Checked = true;
                else
                    this.rdFamele.Checked = true;
            }
        }

        public bool CanModifyID
        {
            set { this.txtID.Enabled = value; }
        }

        #endregion
    }
}
```

Portanto, você criou as principais classes do seu sistema utilizando o padrão MVC. Criou o seu model com a entidade `Funcionario` e o seu `controller` com todos os métodos de ação que fazem o intermédio entre a view e o model. Por fim, criou a sua view utilizando o Windows Forms. Outra opção para a view seria utilizar uma página ASP (caso fosse um projeto web) ou até mesmo relatórios.

Entendendo a arquitetura MVC

Agora, você vai ver como fica a arquitetura do seu sistema. No **Gerenciador de Soluções**, há uma breve visão de como ficou estruturado o sistema com o padrão MVC. Veja, na Figura 3, como ficou a estrutura de arquivos do seu sistema.

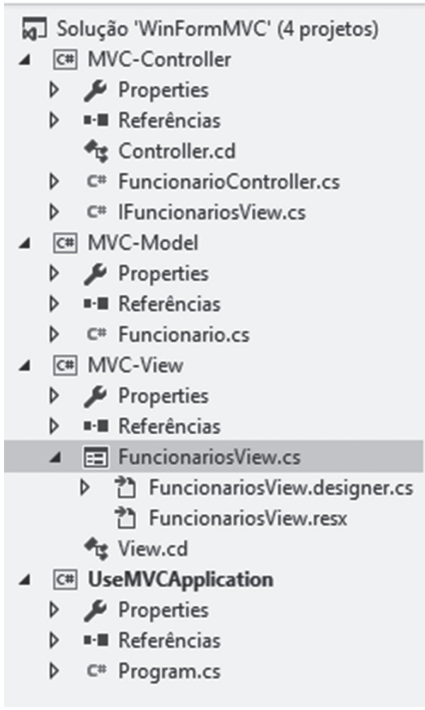

Figura 3. Estrutura de arquivos do sistema.

Há a divisão entre as classes de controller, model e view por meio dos pacotes. A seguir, na Figura 4, você pode verificar a estrutura por meio das classes.

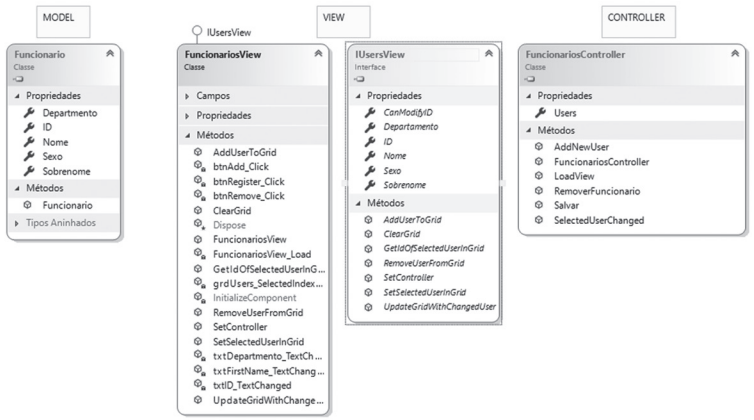

Figura 4. Estrutura organizada por meio das classes.

Referências

FAGERBERG, J. *C# for beginners*: the tactical guidebook-learn CSharp by coding. [S.l.]: CreateSpace Independent, 2015.

FREEMAN, A. *Pro ASP.NET MVC 5*. Berkeley: Apress, 2013.

MICROSOFT. *Learn about ASP.NET MVC*. Redmond, 2017. Disponível em: <https://www.asp.net/mvc/>. Acesso em: 10 nov. 2017.

SMITH, S. *Visão geral do núcleo do ASP.NET MVC*. Redmond: Microsoft, 2016. Disponível em: <https://docs.microsoft.com/pt-br/aspnet/core/mvc/overview>. Acesso em: 10 nov. 2017.

Leitura recomendada

SLACK, N. et al. *Gerenciamento de operações e de processos*: princípios e práticas de impacto estratégico. 2. ed. Porto Alegre: Bookman, 2013.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS