DOCUMENTAÇÃO COMPLETA
PROGRAMA CHURRASCARIA LLCART

DESENVOLVEDORES:

MATHEUS FERREIRA LOPES
DANIEL SENA SANTIAGO
GUSTAVO NINO

VERSÃO: 1.0

Diadema
2025

SUMÁRIO

1. ARQUITETURA DO SISTEMA

1.1 VISÃO GERAL

Sistema de gestão para a churrascaria LLCART, desenvolvido em C# .NET Windows Forms, utilizando arquitetura em camadas e padrões de projeto.

1.2 TECNOLOGIAS UTILIZADAS

- Linguagem: C# .NET
- Interface: Windows Forms
- Banco de Dados: MySQL
- Padrões: Repository, Service Layer, MVC

1.3 ESTRUTURA DE PASTAS

1. Data/…….(Conexão com banco)
2. Models/……(Entidades do sistema)
3. Repositories/……(Acesso a dados)
4. Services/……(Lógica)
5. UserControls/……..(Componentes de UI)
6. Views/……..(Formulários)
7. Arquivo Raiz……(Configuração)

1.4 FLUXO DE DADOS

Database -> Repositories -> Services -> UserControls/Views -> Usuário

2. CAMADA DATA

2.1 DatabaseConfig.cs - Configuração centralizada da conexão com MySQL

```
public static class DatabaseConfig
{
    // Variáveis de configuração (privadas e estáticas)
    private static string _server = "localhost";
    private static string _database = "churrascariadb";
    private static string _userId = "root";
    private static string _password = "";

    // Property que monta a string de conexão
    public static string ConnectionString
    {

get=>$"Server={_server};Database={_database};Uid={_userId};Pwd={_password};";
    }
}
```

2.2 DatabaseContext.cs - Gerenciamento de ciclo de vida da conexão

```
public class DatabaseContext : IDisposable
{
  private MySqlConnection _connection;

  // Construtor abre conexão automaticamente
  public DatabaseContext()
  {
    _connection = new MySqlConnection(DatabaseConfig.ConnectionString);
    _connection.Open(); // Ponto crítico - pode lançar exceção
  }

  // Fornece conexão para repositórios
  public MySqlConnection GetConnection() => _connection;

  // Libera recursos (padrão Disposable)
  public void Dispose()
  {
    _connection?.Close();
    _connection?.Dispose(); }}
```

3. CAMADA MODELS

3.1 Mesa.cs - Representa uma mesa da churrascaria com seu estado atual

```
namespace Forms_LLCART_Projeto.Models
{
    public class Mesa
    {
        public int Id { get; set; }                 // PK do banco
        public string Numero { get; set; }          // Número da mesa ("01", "02")
        public int Capacidade { get; set; }         // Quantidade de pessoas
        public StatusMesa Status { get; set; }      // Estado atual (Enum)
        public string ComandaAtual { get; set; }    // Comanda vinculada
    }

    public enum StatusMesa
    {
        Livre,      // 0 - Mesa disponível
        Ocupada,    // 1 - Mesa com clientes
        Reservada   // 2 - Mesa reservada
    }
}
```

3.2 Pedido.cs - Representa um pedido completo com todos os itens

```
public class Pedido
{
    public int Id { get; set; }
    public int MesaId { get; set; }                 // FK para Mesa
    public string Comanda { get; set; }             // Identificador único
    public DateTime DataAbertura { get; set; }
    public DateTime? DataFechamento { get; set; }   // Nullable = pedido aberto
    public StatusPedido Status { get; set; }
    public List<ItemPedido> Itens { get; set; }     // Composição - pedido tem itens
    public string Observacoes { get; set; }
    public string GarcomResponsavel { get; set; }

    // Property calculada - não persiste no banco
    public decimal Total => CalcularTotal();

    // Construtor inicializa valores padrão
    public Pedido()
    {
        Itens = new List<ItemPedido>();
        DataAbertura = DateTime.Now;
        Status = StatusPedido.Aberto;
    }

    // Método privado para cálculo do total
    private decimal CalcularTotal()
    {
        decimal total = 0;
        foreach (var item in Itens)
```

```
        {
            if (item.Status != StatusItem.Cancelado)
                total += item.Subtotal;
        }
        return total;
    }
}
```

3.3 Produto.cs - Representa um produto do cardápio

```
public class Produto
{
    public int Id { get; set; }
    public string Nome { get; set; }                // "Picanha", "Coca-Cola"
    public string Categoria { get; set; }           // "Carnes", "Bebidas"
    public decimal Preco { get; set; }
    public string Descricao { get; set; }
    public bool Ativo { get; set; }                 // Soft delete
    public int Estoque { get; set; }
    public string SetorPreparo { get; set; }        // "Churrasco", "Cozinha"
}
```

4. CAMADA REPOSITORIES

A camada Repositories é responsável por toda a comunicação direta com o banco de dados, implementando o padrão Repository para isolar o acesso a dados.

4.1 MesaRepo.cs - Operações CRUD (Create, Read, Update, Delete) para a entidade Mesa.

1. CONSTRUTOR E DEPENDÊNCIAS

```
public class MesaRepo
{
   // Não tem construtor específico - usa DatabaseContext diretamente
   // Padrão: Cada método gerencia seu próprio ciclo de conexão
}
```

2. MÉTODO OBTER TODAS - READ

```
public List<Mesa> ObterTodas()
{
   var mesas = new List<Mesa>();

   using (var context = new DatabaseContext())  //  Using garante Dispose()
   {
      var command = new MySqlCommand(
         "SELECT Id, Numero, Capacidade, Status, ComandaAtual FROM Mesas ORDER BY Numero",
         context.GetConnection());

      using (var reader = command.ExecuteReader())  //  Using no Reader também
      {
         while (reader.Read())  //  Lê cada linha do resultado
         {
            mesas.Add(new Mesa
            {
               Id = reader.GetInt32("Id"),
               Numero = reader.GetString("Numero"),
               Capacidade = reader.GetInt32("Capacidade"),
               Status = (StatusMesa)reader.GetInt32("Status"),  //  Cast do enum
               ComandaAtual = reader.IsDBNull("ComandaAtual") ? null : reader.GetString("ComandaAtual")
            });
         }
      }
   }
   return mesas;
}
```

3. MÉTODO OBTER POR ID - READ ESPECÍFICO

```
public Mesa ObterPorId(int id)
{
   using (var context = new DatabaseContext())
   {
      var command = new MySqlCommand(
         "SELECT Id, Numero, Capacidade, Status, ComandaAtual FROM Mesas WHERE Id = @Id",
         context.GetConnection());

      command.Parameters.AddWithValue("@Id", id);  //  Parâmetro evita SQL Injection

      using (var reader = command.ExecuteReader())
      {
         if (reader.Read())  //  if em vez de while - espera só um resultado
         {
            return new Mesa
            {
               // Mapeamento igual ao método anterior
            };
         }
      }
   }
   return null;  //  Retorna null se não encontrou
}
```

4. MÉTODO CRIAR - CREATE

```
public void Criar(Mesa mesa)
{
   using (var context = new DatabaseContext())
   {
      var command = new MySqlCommand(
         "INSERT INTO Mesas (Numero, Capacidade, Status, ComandaAtual) " +
         "VALUES (@Numero, @Capacidade, @Status, @ComandaAtual); " +
         "SELECT LAST_INSERT_ID();",  //  Retorna o ID gerado
         context.GetConnection());

      //  Parameters.AddWithValue - SEGURANÇA contra SQL Injection
      command.Parameters.AddWithValue("@Numero", mesa.Numero);
      command.Parameters.AddWithValue("@Capacidade", mesa.Capacidade);
      command.Parameters.AddWithValue("@Status", (int)mesa.Status);  //  Cast para INT
      command.Parameters.AddWithValue("@ComandaAtual",
         string.IsNullOrEmpty(mesa.ComandaAtual) ? DBNull.Value : (object)mesa.ComandaAtual);

      mesa.Id = Convert.ToInt32(command.ExecuteScalar());  //  ExecuteScalar para ID
   }
}
```

5. MÉTODO ATUALIZAR STATUS

```
public void AtualizarStatus(int mesaId, StatusMesa status, string comanda = null)
{
   using (var context = new DatabaseContext())
   {
      var command = new MySqlCommand(
         "UPDATE Mesas SET Status = @Status, ComandaAtual = @ComandaAtual WHERE Id =
@Id",
          context.GetConnection());

      command.Parameters.AddWithValue("@Id", mesaId);
      command.Parameters.AddWithValue("@Status", (int)status);
      command.Parameters.AddWithValue("@ComandaAtual",
         string.IsNullOrEmpty(comanda) ? DBNull.Value : (object)comanda);

      command.ExecuteNonQuery();  //  ExecuteNonQuery para INSERT/UPDATE/DELETE
   }
}
```

4.2 PedidoRepo.cs - Operações complexas para pedidos e itens

1. CRIAR PEDIDOS COM ITENS - TRANSACTION IMPLÍCITA

```
public void Criar(Pedido pedido)
{
   using (var context = new DatabaseContext())
   {
      // 1. INSERE PEDIDO
      var command = new MySqlCommand(
         @"INSERT INTO Pedidos (MesaId, Comanda, DataAbertura, Status, Observacoes,
GarcomResponsavel)
          VALUES (@MesaId, @Comanda, @DataAbertura, @Status, @Observacoes, @GarcomResponsavel);
          SELECT LAST_INSERT_ID();",
         context.GetConnection());

      // ... parameters do pedido
      pedido.Id = Convert.ToInt32(command.ExecuteScalar());

      // 2. INSERE CADA ITEM DO PEDIDO
      foreach (var item in pedido.Itens)
      {
         var itemCommand = new MySqlCommand(
            @"INSERT INTO ItensPedido (PedidoId, ProdutoId, NomeProduto, Quantidade, PrecoUnitario,
Observacoes, Status, DataHora, UsuarioResponsavel)
              VALUES (@PedidoId, @ProdutoId, @NomeProduto, @Quantidade, @PrecoUnitario,
@Observacoes, @Status, @DataHora, @UsuarioResponsavel);
              SELECT LAST_INSERT_ID();",
            context.GetConnection());

         // ... parameters do item
         item.Id = Convert.ToInt32(itemCommand.ExecuteScalar());
      }
   }
}
```

2. OBTER PEDIDO POR ID COM ITENS

```
public Pedido ObterPorId(int id)
{
   using (var context = new DatabaseContext())
   {
      // 1. BUSCA DADOS DO PEDIDO
      var command = new MySqlCommand(
         @"SELECT Id, MesaId, Comanda, DataAbertura, DataFechamento, Status, Observacoes,
GarcomResponsavel
           FROM Pedidos WHERE Id = @Id",
         context.GetConnection());
      command.Parameters.AddWithValue("@Id", id);

      using (var reader = command.ExecuteReader())
      {
         if (reader.Read())
         {
            var pedido = new Pedido
            {
               // Mapeamento dos dados básicos
            };

            // 2. CARREGA ITENS DO PEDIDO
            pedido.Itens = ObterItensPorPedidoId(pedido.Id);  //  Método auxiliar
            return pedido;
         }
      }
   }
   return null;
}
```

3. MÉTODO AUXILIAR - OBTER ITENS POR PEDIDO

```
private List<ItemPedido> ObterItensPorPedidoId(int pedidoId)
{
   var itens = new List<ItemPedido>();

   using (var context = new DatabaseContext())
   {
      var command = new MySqlCommand(
         @"SELECT Id, ProdutoId, NomeProduto, Quantidade, PrecoUnitario, Observacoes, Status, DataHora,
UsuarioResponsavel
           FROM ItensPedido WHERE PedidoId = @PedidoId",
         context.GetConnection());
      command.Parameters.AddWithValue("@PedidoId", pedidoId);

      using (var reader = command.ExecuteReader())
      {
         while (reader.Read())
         {
            itens.Add(new ItemPedido
            {
               // Mapeamento completo do item
            });
         }
      }
```

```
    }
    return itens;
}
```

4. OBTER ITENS POR STATUS - PARA COZINHA

```
public List<ItemPedido> ObterItensPorStatus(StatusItem status)
{
    var itens = new List<ItemPedido>();

    using (var context = new DatabaseContext())
    {
        var command = new MySqlCommand(
            @"SELECT ip.Id, ip.PedidoId, ip.ProdutoId, ip.NomeProduto, ip.Quantidade, ip.PrecoUnitario,
                    ip.Observacoes, ip.Status, ip.DataHora, ip.UsuarioResponsavel,
                    p.Comanda, p.MesaId
              FROM ItensPedido ip
              INNER JOIN Pedidos p ON ip.PedidoId = p.Id          --  JOIN para dados extras
              WHERE ip.Status = @Status AND p.Status = 0          --  Só pedidos abertos
              ORDER BY ip.DataHora",
            context.GetConnection());

        command.Parameters.AddWithValue("@Status", (int)status);
        // ... execução e mapeamento
    }
    return itens;
}
```

4.3 ProdutoRepo.cs - Operações para produtos do cardápio

1. OBTER CATEGORIAS DISTINTAS

```
public List<string> ObterCategorias()
{
    var categorias = new List<string>();

    using (var context = new DatabaseContext())
    {
        var command = new MySqlCommand(
            "SELECT DISTINCT Categoria FROM Produtos WHERE Ativo = 1 ORDER BY Categoria",
            context.GetConnection());

        using (var reader = command.ExecuteReader())
        {
            while (reader.Read())
            {
                categorias.Add(reader.GetString("Categoria"));  //  Apenas uma coluna
            }
        }
    }
    return categorias;
}
```

5. CAMADA SERVICES

A camada Services contém a lógica de negócio do sistema, coordenando operações entre múltiplos repositórios e aplicando regras de negócio.

5.1 MesaService.cs - Coordena relações relacionadas a mesas e aplicar regras de negócio

1. INJEÇÃO DE DEPENDÊNCIAS DE CONSTRUTOR

```
public class MesaService
{
   private readonly MesaRepo _mesaRepo;  //  readonly - só atribui no construtor

   public MesaService()
   {
      _mesaRepo = new MesaRepo();  //  Composição - Service "tem um" Repository
   }
}
```

2. MÉTODOS DE DELEGAÇÃO SIMPLES

```
public List<Mesa> ObterMesas()
{
   return _mesaRepo.ObterTodas();  //  Apenas delega para o Repository
}

public Mesa ObterMesaPorId(int id)
{
   return _mesaRepo.ObterPorId(id);
}
```

3. MÉTODO COM LÓGICA DE NEGÓCIO

```
public void AtualizarStatusMesa(int mesaId, StatusMesa status, string comanda = null)
{
   //  ANTES: Poderia ter validações como:
   // - Verificar se mesa existe
   // - Validar transições de estado (Livre → Ocupada → Livre)
   // - Logar a alteração

   _mesaRepo.AtualizarStatus(mesaId, status, comanda);  //  Delega para Repository

   //  DEPOIS: Poderia notificar outros sistemas
   // - Atualizar interface em tempo real
   // - Enviar para display da cozinha
}
```

5.2 PedidoService.cs - Coordena o fluxo completo de pedidos com múltiplos repositórios.

1. MÚLTIPLAS DEPENDÊNCIAS

```
public class PedidoService
{
   private readonly PedidoRepo _pedidoRepo;
   private readonly MesaRepo _mesaRepo;  //  Precisa atualizar mesa também

   public PedidoService()
   {
      _pedidoRepo = new PedidoRepo();
      _mesaRepo = new MesaRepo();
   }
}
```

2. SALVAR PEDIDO - COORDENAÇÃO ENTRE REPOSITORIES

```
public void SalvarPedido(Pedido pedido)
{
   if (pedido.Id == 0)  //  Pedido novo
   {
      _pedidoRepo.Criar(pedido);  // 1. Salva pedido no banco

      // 2. Atualiza status da mesa para ocupada
      _mesaRepo.AtualizarStatus(pedido.MesaId, StatusMesa.Ocupada, pedido.Comanda);
   }
   //  FUTURO: else para atualizar pedido existente
}
```

3. FECHAR PEDIDO - FLUXO COMPLETO

```
public void FecharPedido(int pedidoId)
{
   // 1. Busca pedido para validar existência
   var pedido = _pedidoRepo.ObterPorId(pedidoId);

   if (pedido != null)
   {
      // 2. Fecha pedido no banco
      _pedidoRepo.FecharPedido(pedidoId);

      // 3. Libera mesa associada
      _mesaRepo.AtualizarStatus(pedido.MesaId, StatusMesa.Livre, null);

      //  FUTURO: Poderia gerar nota fiscal, enviar email, etc.
   }
}
```

4. OBTER PEDIDO POR MESA - LÓGICA DE BUSCA

```
public Pedido ObterPedidoPorMesa(int mesaId)
{
   var pedidosAtivos = _pedidoRepo.ObterPedidosAtivos();  //  Todos pedidos abertos

   //  Encontra pedido específico da mesa
   return pedidosAtivos.Find(p => p.MesaId == mesaId);

   //  ALTERNATIVA: Repository específico para esta busca
   // return _pedidoRepo.ObterPorMesa(mesaId);
}
```

5. ATUALIZAR STATUS DO PEDIDO - PARA COZINHA

```
public void AtualizarStatusItem(int pedidoId, int itemId, StatusItem novoStatus)
{
   _pedidoRepo.AtualizarStatusItem(pedidoId, itemId, novoStatus);

   //  FUTURO: Notificar garçom quando item estiver pronto
   //  FUTURO: Verificar se todos itens foram entregues para liberar mesa
}
```

6. OBTER ITENS POR STATUS - PARA PAINEL DA COZINHA

```
public List<ItemPedido> ObterItensPorStatus(StatusItem status)
{
   return _pedidoRepo.ObterItensPorStatus(status);
}
```

5.3 ProdutoService.cs - Gerencia produtos e categorias do cardápio.

1. FILTRAGEM POR CATEGORIA

```
public List<Produto> ObterProdutosPorCategoria(string categoria)
{
   var produtos = _produtoRepo.ObterTodos();  //  Pega todos do banco

   //  Filtra localmente - poderia ser no banco com WHERE
   return produtos.FindAll(p => p.Categoria == categoria);
}
```

2. OBTER CATEGORIAS

```
public List<string> ObterCategorias()
{
   return _produtoRepo.ObterCategorias();  //  Delega para Repository especializado
}
```

5.4 GerenciadorDados.cs - Camada em memória para testes/demonstrações

```
namespace Forms_LLCART_Projeto.Services
{
   public static class GerenciadorDados
   {
      private static MesaRepo _mesaRepo = new MesaRepo();
      private static PedidoRepo _pedidoRepo = new PedidoRepo();

      public static List<Mesa> ObterMesas()
      {
         return _mesaRepo.ObterTodas();
      }

      public static Mesa ObterMesaPorId(int id)
      {
         return _mesaRepo.ObterPorId(id);
      }

      public static Mesa ObterMesaPorNumero(string numero)
      {
         var mesas = _mesaRepo.ObterTodas();
         return mesas.FirstOrDefault(m => m.Numero == numero);
      }

      public static void FecharPedido(int pedidoId)
      {
         Console.WriteLine($" FecharPedido: Procurando pedido ID={pedidoId}");

         var pedido = _pedidoRepo.ObterPorId(pedidoId);
         if (pedido != null)
         {
            Console.WriteLine($" Pedido encontrado - Comanda: {pedido.Comanda}, Mesa: {pedido.MesaId}");

            _pedidoRepo.FecharPedido(pedidoId);

            var mesa = _mesaRepo.ObterPorId(pedido.MesaId);
            if (mesa != null)
            {
               _mesaRepo.AtualizarStatus(mesa.Id, StatusMesa.Livre, null);
            }

            Console.WriteLine($" PEDIDO FECHADO: {pedido.Comanda} - Mesa {pedido.MesaId} liberada");
         }
         else
         {
            Console.WriteLine($" Pedido ID {pedidoId} NÃO ENCONTRADO para fechar!");
         }
      }


   }
}
```

6. CAMADA USER CONTROLS

A camada User Controls contém componentes de interface reutilizáveis que encapsulam funcionalidades específicas do sistema.

6.1 ucMesas.cs - Exibir e gerenciar visualmente as mesas da churrascaria

1. INICIALIZAÇÃO E DEPENDÊNCIAS

```
public partial class ucMesas : UserControl
{
   private MesaService mesaService;
   private List<Mesa> mesas;

   public ucMesas()
   {
      InitializeComponent();      //  Método gerado pelo Designer
      mesaService = new MesaService();
      CarregarMesas();          //  Carrega dados na inicialização
   }
}
```

2. CARREGAR MESAS - COMUNICAÇÃO COM SERVICE

```
private void CarregarMesas()
{
   mesas = mesaService.ObterMesas();  //  Busca dados do Service

   //  DEBUG: Log para console (útil durante desenvolvimento)
   foreach (var mesa in mesas)
   {
      Console.WriteLine($"DEBUG Mesa {mesa.Numero}: Status={mesa.Status},
Comanda={mesa.ComandaAtual}");
   }

   flowPanel.Controls.Clear();  //  Limpa mesas anteriores

   //  Cria visualização para cada mesa
   foreach (var mesa in mesas)
   {
      var panelMesa = CriarPanelMesa(mesa);
      flowPanel.Controls.Add(panelMesa);
   }
}
```

3. CRIAR PAINEL MESA - INTERFACE VISUAL DINÂMICA

```
private Panel CriarPanelMesa(Mesa mesa)
{
   var panel = new Panel
   {
      Width = 150,
      Height = 150,
      Margin = new Padding(10),
      BorderStyle = BorderStyle.FixedSingle,
```

```
    Cursor = Cursors.Hand,          //  Indica que é clicável
    Tag = mesa                      //  Armazena objeto para recuperar depois
};

//  CORES DINÂMICAS baseadas no status
Color corFundo, corStatus;
if (mesa.Status == StatusMesa.Livre)
{
    corFundo = Color.LightGreen;
    corStatus = Color.Green;
}
else if (mesa.Status == StatusMesa.Ocupada)
{
    corFundo = Color.LightCoral;
    corStatus = Color.Red;
}
else // Reservada
{
    corFundo = Color.LightYellow;
    corStatus = Color.Orange;
}

panel.BackColor = corFundo;

//  LABEL DO NÚMERO DA MESA
var lblNumero = new Label
{
    Text = $"Mesa {mesa.Numero}",
    Dock = DockStyle.Top,
    Height = 40,
    TextAlign = ContentAlignment.MiddleCenter,
    Font = new Font("Microsoft Sans Serif", 12, FontStyle.Bold)
};

//  LABEL DA CAPACIDADE
var lblCapacidade = new Label
{
    Text = $"{mesa.Capacidade} lugares",
    Dock = DockStyle.Top,
    Height = 30,
    TextAlign = ContentAlignment.MiddleCenter
};

//  LABEL DO STATUS
var lblStatus = new Label
{
    Text = mesa.Status.ToString().ToUpper(),
    Dock = DockStyle.Bottom,
    Height = 30,
    TextAlign = ContentAlignment.MiddleCenter,
    Font = new Font("Microsoft Sans Serif", 9, FontStyle.Bold),
    BackColor = corStatus,
    ForeColor = Color.White
};

//  LABEL DA COMANDA (se existir)
var lblComanda = new Label
```

```
    {
        Text = mesa.ComandaAtual ?? "Sem comanda",  //  Operador null-coalescing
        Dock = DockStyle.Top,
        Height = 20,
        TextAlign = ContentAlignment.MiddleCenter,
        Font = new Font("Microsoft Sans Serif", 7),
        ForeColor = Color.DarkBlue
    };

    //  ADICIONA TODOS OS CONTROLES AO PANEL
    panel.Controls.AddRange(new Control[] { lblStatus, lblComanda, lblCapacidade, lblNumero });

    //  CONFIGURA EVENTOS DE CLIQUE EM TODOS OS ELEMENTOS
    panel.Click += (s, e) => AbrirMesa(mesa);
    lblNumero.Click += (s, e) => AbrirMesa(mesa);
    lblCapacidade.Click += (s, e) => AbrirMesa(mesa);
    lblStatus.Click += (s, e) => AbrirMesa(mesa);
    lblComanda.Click += (s, e) => AbrirMesa(mesa);

    return panel;
}
```

4. ABRIR MESA - LÓGICA DE NEGÓCIO NA INTERFACE

```
private void AbrirMesa(Mesa mesa)
{
    Console.WriteLine($"Clicou na mesa {mesa.Numero} - Status: {mesa.Status}");

    if (mesa.Status == StatusMesa.Livre)
    {
        //  MESA LIVRE: Abre formulário para novo pedido
        var formPedido = new Views.frmPedido(mesa);
        var resultado = formPedido.ShowDialog();  //  Modal - trava tela atual

        if (resultado == DialogResult.OK)  //  Pedido foi criado com sucesso
        {
            CarregarMesas();  //  Atualiza interface

            //  Feedback para usuário
            var pedidoService = new PedidoService();
            var pedidoSalvo = pedidoService.ObterPedidoPorMesa(mesa.Id);

            if (pedidoSalvo != null)
            {
                MessageBox.Show($" PEDIDO CRIADO!\n\nMesa: {mesa.Numero}\nComanda:
{pedidoSalvo.Comanda}\nItens: {pedidoSalvo.Itens.Count}",
                    "Pedido Criado", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }
    }
    else if (mesa.Status == StatusMesa.Ocupada)
    {
        //  MESA OCUPADA: Tenta abrir pedido existente para edição
        var pedidoService = new PedidoService();
        var pedidoExistente = pedidoService.ObterPedidoPorMesa(mesa.Id);

        if (pedidoExistente != null)
```

```
        {
            var formPedido = new Views.frmPedido(mesa, pedidoExistente);
            var resultado = formPedido.ShowDialog();

            if (resultado == DialogResult.OK)
            {
                CarregarMesas();
                MessageBox.Show($" Pedido atualizado!\nMesa: {mesa.Numero}",
                    "Atualizado", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }
        else
        {
            MessageBox.Show($" Mesa {mesa.Numero} está marcada como ocupada mas não encontramos
pedido.\n\nLiberando mesa...",
                "Pedido Não Encontrado", MessageBoxButtons.OK, MessageBoxIcon.Warning);

            mesaService.AtualizarStatusMesa(mesa.Id, StatusMesa.Livre, null);
            CarregarMesas();
        }
    }
}
```

6.2 ucCozinha.cs - Painel Kanban para controle de produção na cozinha

1. TIMER PARA ATUALIZAÇÃO AUTOMÁTICA

```
private Timer timerAtualizacao;

private void ConfigurarTimer()
{
    timerAtualizacao = new Timer();
    timerAtualizacao.Interval = 5000;  //  5 segundos
    timerAtualizacao.Tick += (s, e) => CarregarPedidos();  //  Evento do timer
    timerAtualizacao.Start();
}
```

2. CARREGAR PEDIDOS - ORGANIZAÇÃO POR STATUS

```
private void CarregarPedidos()
{
    //  CARREGA TRÊS COLUNAS DIFERENTES
    CarregarItensPorStatus(StatusItem.Novo, flowNovos, "NOVOS PEDIDOS");
    CarregarItensPorStatus(StatusItem.EmPreparo, flowPreparo, "EM PREPARO");
    CarregarItensPorStatus(StatusItem.Pronto, flowProntos, "PRONTOS PARA ENTREGA");
}
```

3. CARREGAR ITENS POR STATUS - MÉTODO REUTILIZÁVEL

```
private void CarregarItensPorStatus(StatusItem status, FlowLayoutPanel flowPanel, string titulo)
{
    flowPanel.Controls.Clear();  //  Limpa itens anteriores

    //  TÍTULO DA COLUNA
    var lblTitulo = new Label
    {
        Text = titulo,
        Width = flowPanel.Width - 20,
        Height = 30,
        TextAlign = ContentAlignment.MiddleCenter,
        Font = new Font("Microsoft Sans Serif", 10, FontStyle.Bold),
        BackColor = GetCorStatus(status),  //  Cor baseada no status
        ForeColor = Color.White,
        Margin = new Padding(5)
    };
    flowPanel.Controls.Add(lblTitulo);

    //  BUSCA ITENS DO SERVICE
    var itens = pedidoService.ObterItensPorStatus(status);

    if (itens.Count == 0)
    {
        //  MENSAGEM QUANDO NÃO HÁ ITENS
        var lblVazio = new Label
        {
            Text = "Nenhum item",
            Width = flowPanel.Width - 20,
            Height = 40,
            TextAlign = ContentAlignment.MiddleCenter,
            Font = new Font("Microsoft Sans Serif", 9, FontStyle.Italic),
            ForeColor = Color.Gray
        };
        flowPanel.Controls.Add(lblVazio);
        return;
    }

    //  CRIA CARD PARA CADA ITEM
    foreach (var item in itens)
    {
        var panelItem = CriarPanelItemCozinha(item);
        flowPanel.Controls.Add(panelItem);
    }
}
```

4. CRIAR PAINEL ITEM - CARD INTERATIVO

```
private Panel CriarPanelItemCozinha(ItemPedido item)
{
    var panel = new Panel
    {
        Width = 250,
        Height = 100,
        Margin = new Padding(5),
        BorderStyle = BorderStyle.FixedSingle,
        BackColor = GetCorStatus(item.Status)  //  Fundo colorido por status
    };

    //  INFORMAÇÕES DO PRODUTO
    var lblProduto = new Label
    {
        Text = $"{item.Quantidade}x {item.NomeProduto}",
        Location = new Point(10, 10),
        Size = new Size(230, 20),
        Font = new Font("Microsoft Sans Serif", 10, FontStyle.Bold),
        TextAlign = ContentAlignment.MiddleLeft
    };

    //  BOTÕES DINÂMICOS baseados no status
    if (item.Status == StatusItem.Novo)
    {
        var btnIniciar = new Button
        {
            Text = "Iniciar",
            Location = new Point(150, 30),
            Size = new Size(80, 25),
            BackColor = Color.Orange,
            Tag = item  //  Armazena item para recuperar no evento
        };
        btnIniciar.Click += (s, e) => AtualizarStatusItem(item, StatusItem.EmPreparo);
        panel.Controls.Add(btnIniciar);
    }
    else if (item.Status == StatusItem.EmPreparo)
    {
        var btnFinalizar = new Button
        {
            Text = "Pronto",
            Location = new Point(150, 30),
            Size = new Size(80, 25),
            BackColor = Color.LightGreen,
            Tag = item
        };
        btnFinalizar.Click += (s, e) => AtualizarStatusItem(item, StatusItem.Pronto);
        panel.Controls.Add(btnFinalizar);
    }

    //  ADICIONA TODOS OS CONTROLES
    panel.Controls.AddRange(new Control[] { lblProduto, /* outros labels */ });

    return panel;
}
```

5. ATUALIZAR STATUS ITEM - FLUXO DA COZINHA

```
private void AtualizarStatusItem(ItemPedido item, StatusItem novoStatus)
{
   var pedidoService = new PedidoService();

   //  BUSCA PEDIDO ATIVO QUE CONTÉM O ITEM
   var pedidosAtivos = pedidoService.ObterPedidosAtivos();
   foreach (
```

7. CAMADA VIEWS

A camada Views contém os formulários principais dos sistemas que coordenam a exibição e navegação entre os User Controls.

7.1 frmMain.cs - Container principal com menu de navegação e área de conteúdo dinâmico.

1. CONSTRUTOR E INICIALIZAÇÃO

```
public frmMain()
{
   InitializeComponent();    //  Configura controles do Designer
   ConfigurarEventos();     //  Conecta eventos dos botões
   CarregarDashboard();     //  Tela inicial
}
```

2. CONFIGURAR EVENTOS - NAVEGAÇÃO

```
private void ConfigurarEventos()
{
   //  LAMBDA EXPRESSIONS para eventos de clique
   btnMesas.Click += (s, e) => CarregarMesas();
   btnPedidos.Click += (s, e) => CarregarPedidos();
   btnCozinha.Click += (s, e) => CarregarCozinha();
   btnCaixa.Click += (s, e) => CarregarCaixa();
   btnRelatorios.Click += (s, e) => CarregarRelatorios();
}
```

3. CARREGAR USER CONTROLS - PADRÃO DE NAVEGAÇÃO

```
private void CarregarMesas()
{
   panelContainer.Controls.Clear();  //  Limpa conteúdo anterior

   var ucMesas = new UserControls.ucMesas();
   ucMesas.Dock = DockStyle.Fill;    //  Preenche todo o espaço
   panelContainer.Controls.Add(ucMesas);
}

//  PADRÃO REPETIDO PARA TODOS OS USER CONTROLS:
// 1. Clear() no container
// 2. New UserControl()
// 3. Dock = Fill
// 4. Add ao container
```

4. CARREGAR PEDIDOS - LÓGICA ESPECÍFICA

```
private void CarregarPedidos()
{
    panelContainer.Controls.Clear();

    //  CRIA INTERFACE DINÂMICA (não usa UserControl)
    var panel = new Panel { Dock = DockStyle.Fill, BackColor = Color.White };
    var lblTitulo = new Label
    {
        Text = "Gestão de Pedidos - Mesas Ocupadas",
        Dock = DockStyle.Top,
        Height = 60,
        TextAlign = ContentAlignment.MiddleCenter,
        Font = new Font("Microsoft Sans Serif", 16, FontStyle.Bold),
        ForeColor = Color.FromArgb(51, 51, 76)  //  Cor corporativa
    };

    var flowPanel = new FlowLayoutPanel
    {
        Dock = DockStyle.Fill,
        AutoScroll = true,
        Padding = new Padding(20)
    };

    //  BUSCA DADOS DO SERVICE
    var pedidoService = new Services.PedidoService();
    var pedidosAtivos = pedidoService.ObterPedidosAtivos();
    var mesaService = new Services.MesaService();

    if (pedidosAtivos.Count == 0)
    {
        //  ESTADO VAZIO - FEEDBACK AO USUÁRIO
        var lblVazio = new Label
        {
            Text = "Nenhum pedido ativo no momento\n\nClique em 'Mesas' para abrir um novo pedido",
            Dock = DockStyle.Fill,
            TextAlign = ContentAlignment.MiddleCenter,
            Font = new Font("Microsoft Sans Serif", 12, FontStyle.Italic),
            ForeColor = Color.Gray
        };
        flowPanel.Controls.Add(lblVazio);
    }
    else
    {
        //  CRIA CARD PARA CADA PEDIDO ATIVO
        foreach (var pedido in pedidosAtivos)
        {
            var mesa = mesaService.ObterMesaPorId(pedido.MesaId);
            if (mesa != null)
            {
                var panelPedido = CriarPanelPedido(pedido, mesa);
                flowPanel.Controls.Add(panelPedido);
            }
        }
    }
```

```
    // MONTA A HIERARQUIA DE CONTROLES
    panel.Controls.Add(flowPanel);
    panel.Controls.Add(lblTitulo);
    panelContainer.Controls.Add(panel);
}
```

5. CRIAR PAINEL PEDIDO - CARD INTERATIVO

```
private Panel CriarPanelPedido(Pedido pedido, Mesa mesa)
{
    var panel = new Panel
    {
        Width = 250,
        Height = 140,
        Margin = new Padding(10),
        BorderStyle = BorderStyle.FixedSingle,
        Cursor = Cursors.Hand,
        BackColor = Color.LightCoral,
        Tag = new { Pedido = pedido, Mesa = mesa }  // DADOS COMPOSTOS no Tag
    };

    // BOTÃO VER/EDITAR
    var btnDetalhes = new Button
    {
        Text = "Ver/Editar Pedido",
        Dock = DockStyle.Bottom,
        Height = 30,
        BackColor = Color.LightBlue,
        Tag = new { Pedido = pedido, Mesa = mesa }  // Mesmos dados no botão
    };

    // EVENTO DINÂMICO com acesso aos dados
    btnDetalhes.Click += (s, e) =>
    {
        var dados = (dynamic)btnDetalhes.Tag;  // DYNAMIC para acesso fácil
        var formPedido = new Views.frmPedido(dados.Mesa, dados.Pedido);
        formPedido.ShowDialog();
        CarregarPedidos();  // ATUALIZA após fechar o form
    };

    panel.Controls.AddRange(new Control[] { btnDetalhes, /* labels */ });
    return panel;
}
```

7.2 frmPedido.cs - Criar e editar pedidos com interface de cardápio interativo

1. CONSTRUTORES SOBRECARREGADOS

```
// CONSTRUTOR PARA NOVO PEDIDO
public frmPedido(Mesa mesaSelecionada)
{
   InitializeComponent();
   mesa = mesaSelecionada;
   produtoService = new ProdutoService();
   InicializarPedido();  // Cria novo pedido
   CarregarProdutos();   // Carrega cardápio
}

// CONSTRUTOR PARA EDITAR PEDIDO EXISTENTE
public frmPedido(Mesa mesaSelecionada, Pedido pedidoExistente)
{
   InitializeComponent();
   mesa = mesaSelecionada;
   pedidoAtual = pedidoExistente;
   produtoService = new ProdutoService();

   // CONFIGURA INTERFACE PARA EDIÇÃO
   this.Text = $"Editando Pedido - Mesa {mesa.Numero} - Comanda: {pedidoAtual.Comanda}";
   lblComanda.Text = $"Comanda: {pedidoAtual.Comanda}";
   lblMesa.Text = $"Mesa: {mesa.Numero}";

   CarregarProdutos();
   AtualizarListaPedidos();  // Carrega itens existentes
}
```

2. INICIALIZAR PEDIDO - NOVO PEDIDO

```
private void InicializarPedido()
{
   pedidoAtual = new Pedido
   {
      MesaId = mesa.Id,
      Comanda = GerarNumeroComanda(),  // Número único
      GarcomResponsavel = "Garçom"     // Valor padrão
   };

   // ATUALIZA INTERFACE
   this.Text = $"Novo Pedido - Mesa {mesa.Numero} - Comanda: {pedidoAtual.Comanda}";
   lblComanda.Text = $"Comanda: {pedidoAtual.Comanda}";
   lblMesa.Text = $"Mesa: {mesa.Numero}";
}

private string GerarNumeroComanda()
{
   return DateTime.Now.ToString("ddMMyyyyHHmmss");  // Timestamp como ID único
}
```

3. CARREGAR PRODUTOS - CARDÁPIO INTERATIVO

```
private void CarregarProdutos()
{
    produtos = produtoService.ObterProdutos();
    var categorias = produtoService.ObterCategorias();

    flowCategorias.Controls.Clear();
    flowProdutos.Controls.Clear();

    //  BOTÃO "TODOS"
    var btnTodos = new Button
    {
        Text = "📋 Todos",
        Size = new Size(120, 40),
        Margin = new Padding(5),
        BackColor = Color.LightBlue,
        Tag = "Todos"
    };
    btnTodos.Click += (s, e) => FiltrarProdutosPorCategoria("Todos");
    flowCategorias.Controls.Add(btnTodos);

    //  BOTÕES DE CATEGORIA
    foreach (var categoria in categorias)
    {
        var btnCategoria = new Button
        {
            Text = GetEmojiCategoria(categoria) + " " + categoria,
            Size = new Size(120, 40),
            Margin = new Padding(5),
            BackColor = Color.LightGreen,
            Tag = categoria
        };
        btnCategoria.Click += (s, e) => FiltrarProdutosPorCategoria(categoria);
        flowCategorias.Controls.Add(btnCategoria);
    }

    FiltrarProdutosPorCategoria("Todos");  //  Carrega todos inicialmente
}
```

4. FILTRAR PRODUTOS - LÓGICA DE EXIBIÇÃO

```
private void FiltrarProdutosPorCategoria(string categoria)
{
    flowProdutos.Controls.Clear();

    //  FILTRA PRODUTOS
    var produtosFiltrados = categoria == "Todos"
        ? produtos
        : produtos.Where(p => p.Categoria == categoria).ToList();

    //  CRIA CARD PARA CADA PRODUTO
    foreach (var produto in produtosFiltrados)
    {
        var panelProduto = CriarPanelProduto(produto);
        flowProdutos.Controls.Add(panelProduto);
  } }
```

5. CRIAR PAINEL PRODUTO - CARD CLICÁVEL

```
private Panel CriarPanelProduto(Produto produto)
{
    var panel = new Panel
    {
        Width = 150,
        Height = 100,
        Margin = new Padding(5),
        BorderStyle = BorderStyle.FixedSingle,
        Cursor = Cursors.Hand,      //  Indica clicável
        BackColor = Color.LightYellow,
        Tag = produto               //  Armazena produto para evento
    };

    //  EVENTO DE CLIQUE NO PANEL INTEIRO
    panel.Click += (s, e) => AdicionarProdutoAoPedido(produto);

    //  LABEL DO NOME (também clicável)
    var lblNome = new Label
    {
        Text = produto.Nome,
        Dock = DockStyle.Top,
        Height = 40,
        TextAlign = ContentAlignment.MiddleCenter,
        Font = new Font("Microsoft Sans Serif", 9, FontStyle.Bold),
        BackColor = Color.LightBlue,
        Cursor = Cursors.Hand
    };
    lblNome.Click += (s, e) => AdicionarProdutoAoPedido(produto);

    //  LABEL DO PREÇO (também clicável)
    var lblPreco = new Label
    {
        Text = $"R$ {produto.Preco:F2}",
        Dock = DockStyle.Bottom,
        Height = 30,
        TextAlign = ContentAlignment.MiddleCenter,
        Font = new Font("Microsoft Sans Serif", 9, FontStyle.Bold),
        BackColor = Color.LightGreen,
        Cursor = Cursors.Hand,
        ForeColor = Color.DarkGreen
    };
    lblPreco.Click += (s, e) => AdicionarProdutoAoPedido(produto);

    panel.Controls.AddRange(new Control[] { lblPreco, lblNome });
    return panel;
}
```

6. ADICIONAR PRODUTO - FORMULÁRIO DE QUANTIDADE

```
private void AdicionarProdutoAoPedido(Produto produto)
{
    //  CRIA FORM MODAL PARA QUANTIDADE
    var formQuantidade = new Form
    {
        Text = $"Quantidade - {produto.Nome}",
        Size = new Size(300, 200),
        StartPosition = FormStartPosition.CenterParent,
        FormBorderStyle = FormBorderStyle.FixedDialog,
        MaximizeBox = false,
        BackColor = Color.White
    };

    var numericQuantidade = new NumericUpDown
    {
        Minimum = 1,
        Maximum = 20,
        Value = 1,
        Location = new Point(100, 78),
        Size = new Size(80, 25),
        Font = new Font("Microsoft Sans Serif", 10)
    };

    var btnConfirmar = new Button
    {
        Text = " Adicionar",
        Location = new Point(50, 120),
        Size = new Size(90, 35),
        BackColor = Color.LightGreen,
        Font = new Font("Microsoft Sans Serif", 9, FontStyle.Bold)
    };

    btnConfirmar.Click += (s, e) =>
    {
        //  CRIA NOVO ITEM
        var item = new ItemPedido
        {
            ProdutoId = produto.Id,
            NomeProduto = produto.Nome,
            Quantidade = (int)numericQuantidade.Value,
            PrecoUnitario = produto.Preco,
            Observacoes = "",
            Status = StatusItem.Novo,
            DataHora = DateTime.Now,
            UsuarioResponsavel = "Garçom"
        };

        //  ADICIONA AO PEDIDO
        pedidoAtual.Itens.Add(item);
        AtualizarListaPedidos();  //  Atualiza interface

        MessageBox.Show($" {item.Quantidade}x {item.NomeProduto} adicionado!\nSubtotal: R$
{item.Subtotal:F2}",
            "Item Adicionado", MessageBoxButtons.OK, MessageBoxIcon.Information);
```

```
        formQuantidade.Close();
    };

    formQuantidade.Controls.AddRange(new Control[] { numericQuantidade, btnConfirmar });
    formQuantidade.ShowDialog();
}
```

7. ATUALIZAR LISTA PEDIDOS - RESUMO DO PEDIDO

```
private void AtualizarListaPedidos()
{
    flowItensPedido.Controls.Clear();
    lblTotal.Text = $"Total: R$ {pedidoAtual.Total:F2}";  //  Atualiza total

    if (pedidoAtual.Itens.Count == 0)
    {
        //  ESTADO VAZIO
        var lblVazio = new Label
        {
            Text = "Nenhum item no pedido\n\nClique nos produtos à esquerda para adicionar",
            Size = new Size(350, 100),
            TextAlign = ContentAlignment.MiddleCenter,
            Font = new Font("Microsoft Sans Serif", 10, FontStyle.Italic),
            ForeColor = Color.Gray
        };
        flowItensPedido.Controls.Add(lblVazio);
        return;
    }

    //  CRIA ITEM PARA CADA PRODUTO NO PEDIDO
    foreach (var item in pedidoAtual.Itens)
    {
        var panelItem = CriarPanelItemPedido(item);
        flowItensPedido.Controls.Add(panelItem);
    }
}
```

8. CRIAR PAINEL ITEM PEDIDO - ITEM EDITÁVEL

```
private Panel CriarPanelItemPedido(ItemPedido item)
{
    var panel = new Panel
    {
        Width = 350,
        Height = 70,
        Margin = new Padding(5),
        BorderStyle = BorderStyle.FixedSingle,
        BackColor = Color.White
    };

    //  BOTÃO REMOVER
    var btnRemover = new Button
    {
        Text = "🗑️ Remover",
        Location = new Point(300, 20),
        Size = new Size(40, 30),
        BackColor = Color.LightCoral,
```

```
        Font = new Font("Microsoft Sans Serif", 8),
        Tag = item
    };

    btnRemover.Click += (s, e) =>
    {
        var result = MessageBox.Show($"Remover {item.Quantidade}x {item.NomeProduto}?",
            "Confirmar Remoção", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

        if (result == DialogResult.Yes)
        {
            pedidoAtual.Itens.Remove(item);
            AtualizarListaPedidos();
            MessageBox.Show("Item removido do pedido!", "Removido",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    };

    panel.Controls.AddRange(new Control[] { btnRemover, /* labels */ });
    return panel;
}
```

9. FINALIZAR PEDIDO - VALIDAÇÃO E CONFIRMAÇÃO

```
private void btnFinalizarPedido_Click(object sender, EventArgs e)
{
    // VALIDAÇÃO
    if (pedidoAtual.Itens.Count == 0)
    {
        MessageBox.Show(" Adicione itens ao pedido antes de finalizar!", "Pedido Vazio",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }

    // RESUMO PARA CONFIRMAÇÃO
    var resumo = $"**RESUMO DO PEDIDO**\n\n";
    resumo += $"Mesa: {mesa.Numero}\n";
    resumo += $"Comanda: {pedidoAtual.Comanda}\n";
    resumo += $"Itens: {pedidoAtual.Itens.Count}\n";
    resumo += $"Total: R$ {pedidoAtual.Total:F2}\n\n";
    resumo += "Confirmar finalização do pedido?";

    var result = MessageBox.Show(resumo, " CONFIRMAR PEDIDO",
        MessageBoxButtons.YesNo, MessageBoxIcon.Question);

    if (result == DialogResult.Yes)
    {
        try
        {
            // SALVA NO BANCO
            var pedidoService = new PedidoService();
            pedidoService.SalvarPedido(pedidoAtual);

            MessageBox.Show($" PEDIDO FINALIZADO COM SUCESSO!\n\n" +
                    $"Mesa: {mesa.Numero}\n" +
                    $"Comanda: {pedidoAtual.Comanda}\n" +
                    $"Itens: {pedidoAtual.Itens.Count}\n" +
```

```
                    $"Total: R$ {pedidoAtual.Total:F2}",
                    "SUCESSO",
                    MessageBoxButtons.OK, MessageBoxIcon.Information);

            this.DialogResult = DialogResult.OK;  //  Sinaliza sucesso
            this.Close();
        }
        catch (Exception ex)
        {
            MessageBox.Show($" Erro ao salvar pedido: {ex.Message}", "ERRO",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
}
```

COMUNICAÇÃO ENTRE FORMS:

```
//  PADRÃO MODAL COM RETORNO
var resultado = formPedido.ShowDialog();
if (resultado == DialogResult.OK)
{
    // Atualiza interface após fechar form
    CarregarPedidos();
}

//  PADRÃO DYNAMIC TAG para passar dados
button.Tag = new { Pedido = pedido, Mesa = mesa };
var dados = (dynamic)button.Tag;
```