PROJETO INTERDISCIPLINAR

Projeto de Sistema Bancário

Protótipo de sistema bancário baseado no seu funcionamento geral.

Alunos:

RGM	Nome
30114896	Kaique Melo de Jesus Ferreira
29644666	Mateus Mendes de Jesus
29822106	Kaynan Jonathan da Silva França
30084300	Diogo Henrique Santana de Jesus
28813456	Wallace Faustino Cuer
29971993	Hiago Pereira dos Santos

São Paulo

2023

UNIVERSIDADE CRUZEIRO DO SUL

PROJETO INTERDISCIPLINAR

Projeto de Sistema Bancário

Protótipo de sistema bancário baseado no seu funcionamento geral.

Trabalho apresentado como parte do requisito para aprovação na Disciplina de Projeto Interdisciplinar do curso de Tecnologia em Analise e Desenvolvimento de Sistemas da Universidade Cruzeiro do Sul.

Orientadores: Prof. Agnaldo Silibert Mota e Prof. Fabio Silva

São Paulo
2023

Sumário

1. APRESENTAÇÃO:

1.1 Justificativa e Motivação

Este projeto nos desafia a colocar em prática boa parte do que foi ensinado nos semestres passados, nos levando a ter uma motivação para realmente viver e colocar em prática como realmente é feito um planejamento de um software, realizando desde o levantamento dos requisitos funcionais como também os requisitos não funcionais, realização dos diagramas, regras de negócio e o desenvolvimento do códigos.

1.2 Dados do Sistema.

O sistema desenvolvido foi baseado em um sistema bancário, nele conseguimos demonstrar como funciona o sistema desde o cadastro do cliente até as movimentações na conta que será possível realizar, e também gerenciar perfis de acesso para que cada perfil exerça a sua devida função. O Projeto é importante para ter uma base de como funciona o sistema interno de um banco, dando a possibilidade de conhecer apenas uma parte da imensa diversidade de opções e funcionalidades que um sistema bancário pode nos oferecer.

2 REQUISITOS DE TÉCNICA DE DESENVOLVIMENTO DE ALGORITMOS

2.1 Classe Cadastro

Para começar a descrever os códigos foi feita a condição para o carregamento da página dentro da Classe cadastro.

```
public partial class cadastro : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)//serve para o primeiro carregamento da pagina
        {

        }
    }
```

Logo após os requisitos do cadastro foi desenvolvido da seguinte maneira ainda dentro da classe cadastro:

```
protected void btncadastro_Click(object sender, EventArgs e)
{
    //captura todas as regiões sem espaços vazios e faz a validação de todos os espaços
    string nome = txtnome.Text.Trim();
    string cpf = txtcpf.Text.Trim();
    string senha = txtsenha.Text.Trim();
    string senhac = txtsenhac.Text.Trim();
    string email = txtemail.Text.Trim();
    if(nome == ""){
        resposta.Text = "PREENCHA O NOME";

    }
    else{
    if (senha.Length < 6)
    {
        resposta.Text = "A SENHA DEVE CONTER 6 DIGITOS";
    }
    else
    {
        if(senha == senhac){
        if (cpf.Length < 11)
        {

            resposta.Text = "O CPF DEVE CONTER 11 DIGITOS";
        }
        else {

            if (email.Contains('@'))
            {
                Validate Vcpf = new Validate();
                bool cpfc = Vcpf.Verifica_Cpf(cpf);
                if (cpfc == true)
                {
                    PORC cripto = new PORC();
                    string Senhac = cripto.criptografar(senha);
                    verificarsenha(nome,cpf,email,Senhac);

                }
```

```
        else
        {
            resposta.Text = "O CPF DIGITADO NÃO É VALIDO";
        }

    }
    else {


        resposta.Text = "INSIRA UM EMAIL VÁLIDO";
    }
}
    }
else{
resposta.Text="SENHAS PRECISAM SER IGUAIS";
```

Foram feitas condições de segurança para que não haja conflito nos dados informados pelo usuário durante o cadastro.

Logo depois foi criado o método verificarsenha que serve para verificar a condição da senha informada pelo usuário e verificar no banco de dados se ela já está cadastrada para que não haja conflito de dados e também que a senha seja segura o suficiente.

```
private void verificarsenha(string nome, string cpf, string email, string Senhac)
{
    //metodo para verificar se a senha que foi inserida pelo usuario já existe (impossibilitando bugs no sistema posteriormente)
    Validate consultar_senha = new Validate();
    string Scon = "Data Source=DESKTOP-01QK6E3;Initial Catalog=PJI_BCB;Integrated Security=True";
    string Squery = "SELECT SENHA FROM BCB_INFO WHERE SENHA= '" + Senhac + "'";
    DataTable dta = consultar_senha.Tabela_Generica(Scon, Squery);
    if (dta.Rows.Count == 0)
    {
        //caso a senha digita não exista chama o metodo de cadastro para que seja realizado o insert no banco
        cadastrar(nome, cpf, email, Senhac);
    }
    else
    {
        //se exister pede ao usuario uma nova senha
        //senhas duplicadas neste sistema causam divergencia de transferencia fazendo com que seja transferido entre senhas e não cpf
        resposta.Text = "Escolha uma Senha Mais Segura";
    }


}
```

Logo depois temos o método cadastrar que é responsável pelo cadastro do cliente no banco de dados.Caso o cadastro tenha êxito, automaticamente é adicionado 10 reais para o usuário, caso tenha alguma informação que esteja divergente aparecerá uma mensagem pedindo para que o usuário revise as informações.

```
private void cadastrar (string nome, string cpf,string email, string Senhac){
    // query de cadastro no banco de dados
    string Squery = "insert into BCB_INFO (NOME,CPF,EMAIL,SENHA) VALUES ('"+nome+"','"+cpf+"','"+email+"','"+Senhac+"')";
    string Scon = "Data Source=DESKTOP-01QK6E3;Initial Catalog=PJI_BCB;Integrated Security=True";
    //instancia a classe para fazer a query e retornar uma tabela
    Validate Cadastrar = new Validate();
    DataTable dta = Cadastrar.Tabela_Generica(Scon, Squery);
    if(dta.Rows.Count == 0){
        //se a tabela retornar vazia o insert deu certo ele confirma fazendo um select de cpf e senha
        Squery = "SELECT CPF,SENHA FROM BCB_INFO WHERE CPF= '" + cpf + @"'AND SENHA= '" + Senhac + "'";
    DataTable dtal = Cadastrar.Tabela_Generica(Scon, Squery);
    if (dtal.Rows.Count != 0)
    {
        //apos a confirmação adiciona automaticamente 10 reais ao usuario
        Squery = "INSERT INTO BCB_PESSOA (CPF,SALDO) VALUES ('"+cpf+"','10.00')";
        DataTable dtala = Cadastrar.Tabela_Generica(Scon, Squery);
        resposta.Text = "Cadastro Realizado Com Sucesso";
        btnvoltar.Visible= true;
        btncadastro.Visible = false;
    }
    else
    {
        //caso contrario a falha significa que o cpf não foi registrado em outra tabela ou já existe na tabela de informações
        resposta.Text = "Falha No Cadastro (Revise as Informações)";
    }
    }
    else{
        //exceção final de cadastro somente em falta de dados

    resposta.Text="Exceção contate o administrador do sistema";
    }
```

2.2 Classe Login

A classe login se inicia com um método chamado Page_Load que é a pagina de carregamento.

```
public partial class Login : System.Web.UI.Page
{

    protected void Page_Load(object sender, EventArgs e)
    {
        if(!IsPostBack){
          //PRIMEIRA VEZ SENDO CARREGADA OU SENDO CHAMADO UM CALLBACK
        }
    }
```

Em seguida temos a função btnentrar_Click é a parte de inserção de dados no front para que seja verificado no banco de dados para que a autenticação no banco seja feita com êxito. Nela também foi feita medidas de segurança para que os dados informados não tenha divergência e o login seja feito com sucesso.

```
#region EVENTO DE LOGIN
protected void btnentrar_Click(object sender, EventArgs e)
{
    string cpf = txtcpf.Text.Trim();//CAPTURA O CPF E A SENHA SEM ESPAÇOS VAZIOS
    string senha = txtsenha.Text.Trim();
    if (senha.Length < 6)
    {
        //SENHA MENOR QUE 6 DIGITOS
        resposta.Text = "A SENHA DEVE CONTER 6 DIGITOS";
    }
    else {
        if (cpf.Length < 11)
        {
            //CPF MENOR QUE 11 DIGITOS
            resposta.Text = "O CPF DEVE CONTER 11 DIGITOS";
        }

        else
        {
            //INSTANCIA UMA CLASSE EXISTENTE
            Validate Vcpf = new Validate();
            //VARIAVEL DE FALSE OR TRUE PARA VALIDADR CPF
            bool cpfc = Vcpf.Verifica_Cpf(cpf);
            if (cpfc == true)
            {
                //CPF VALIDO INSTANCIA A CLASSE DE CRIPTOGRAFIA
                PORC cripto = new PORC();
                //ENVIA A SENHA PARA A CRIPTOGRAFIA
                string Senhac = cripto.criptografar(senha);
                //CHAMA O METODO DE LOGIN COM O CPF E A SENHA AGORA CRIPTOGRAFADA
                verificalogin(cpf, Senhac);
            }
            else {
                resposta.Text = "O CPF DIGITADO NÃO É VALIDO";
            }

        }
    }

}
```

Em seguida, temos o método verificalogin que é responsável pela verificação do login no banco de dados que também tem uma verificação de segurança caso os dados informados não esteja cadastrado no banco de dados

```
#region VERIFICAR LOGIN NO BANCO DE DADOS
private void verificalogin(string cpf, string senhac)
{
  //INSTANCIA A CLASSE DE VALIDAÇÃO
    Validate validar = new Validate();
    //STRING DE CONEXÃO E QUERY DOS SISTEMA
    string Scon = "Data Source=DESKTOP-01QK6E3;Initial Catalog=PJI_BCB;Integrated Security=True";
    string Squery = "SELECT CPF,SENHA FROM BCB_INFO WHERE CPF= '" + cpf + @"'AND SENHA= '" + senhac + "'";
    //CRIA UMA NOVA TABELA COM NOME DTA AONDE DENTRO DA CLASSE DE VALIDAÇÃO ELA SERA USADA PARA INSERIR O RESULTADO DA QUERY ACIMA
    DataTable dta = validar.Tabela_Generica(Scon, Squery);
    if (dta.Rows.Count != 0)
    {
        //APOS A TABELA CONTER ALGUMA LINHA COM AS INFORMAÇÕES DA QUERY REDIRECIONA PARA O PERFIL INICIAL INDICANDO A SENHA CRIPTOGRADA COMO ID DE INDENTIFICAÇÃO
        Response.Redirect("~/Perfil.aspx?id=" + senhac + "");


    }
    else
    {
    //CASO CONTRARIO
        resposta.Text = "USUÁRIO NÃO CADASTRADO OU SENHA INCORRETA";
    }

}
```

2.3 Classe Senha

A classe senha tem as funções de alguns botões do front-end e nele tem a função alterar que é usada para abrir uma solicitação de alteração de dados no banco de dados que no caso seria a alteração da senha. Logo no final tem um callback verificarstatus para ver se está tudo certo com os caracteres informados na senha.

```
protected void btnalterar_Click(object sender, EventArgs e)
{
    string cpfd = Request.QueryString["id"];
    alterar(cpfd);
}
private void alterar(string cpfd)
{
    string snova = txtsenha.Text;

    PORC cripto = new PORC();
    string Senhac = cripto.criptografar(snova);
    string Scon = "Data Source=DESKTOP-01QK6E3;Initial Catalog=PJI_BCB;Integrated Security=True";
    string Squery = "UPDATE BCB_INFO SET SENHA = '" + Senhac + "'  WHERE senha = '" + cpfd + "'";
    try
    {
        using (SqlConnection sqlcnn = new SqlConnection(Scon))
        {
            //ABRE UMA CONEXÃO
            sqlcnn.Open();
            //CRIA UM NOVO CMD PASSANDO A QUERY E A CONEXÃO PRA ELE

            using (SqlCommand cmd = new SqlCommand(Squery, sqlcnn))
            {
                cmd.ExecuteScalar().ToString();

            }

        }
    }
    catch (Exception e)
    {
        verificarstatus(Senhac);
    }
}
```

O método verificarstatus permite que verifique no banco de dados o status da senha e dando a possibilidade de alterá-la, e também mostra um status caso a senha for alterada.

```
private void verificarstatus(string Senhac) {
    string Scon = "Data Source=DESKTOP-01QK6E3;Initial Catalog=PJI_BCB;Integrated Security=True";
    string Squery = "SELECT SENHA FROM BCB_INFO WHERE SENHA = '" + Senhac + "'";
    Validate verificar = new Validate();
    DataTable dta = verificar.Tabela_Generica(Scon,Squery);
    if (dta.Rows.Count != 0)
    {
        btnalterar.Visible = false;
        btnvoiltar.Visible = true;
        resultado.Text = "SENHA ALTERADA";
    }
    else {
        resultado.Text = "ALTERAÇÃO PERMITIDA APENAS UMA VEZ";

    }
}
```

2.4 Classe Perfil

A classe perfil começa com a parte de login, logo após ter feito o login aparece o nome da pessoa mais o saldo disponível dela na conta. O método identificação pesquisa o CPF pelo id no banco de dados e consegue identificar a senha mesmo ela estando criptografada.

```
private void indentificação(string id)
{
    //pesquisa o cpf pelo id (senha criptografada e recupera nome e saldo atual do usuario para exibir na tela)
    string Scon = "Data Source=DESKTOP-01QK6E3;Initial Catalog=PJI_BCB;Integrated Security=True";
    string Squery = "SELECT NOME FROM BCB_INFO WHERE SENHA= '" + id + "'";
    using (SqlConnection sqlcnn = new SqlConnection(Scon))
    {
        //ABRE UMA CONEXÃO
        sqlcnn.Open();
        //CRIA UM NOVO CMD PASSANDO A QUERY E A CONEXÃO PRA ELE
        using (SqlCommand cmd = new SqlCommand(Squery, sqlcnn))
        {
            string pessoa = cmd.ExecuteScalar().ToString();
            nome.Text = pessoa;
            Dispose();


        }
        Squery = "SELECT CPF FROM BCB_INFO WHERE SENHA= '" + id + "'";
        using (SqlCommand cmds = new SqlCommand(Squery, sqlcnn))
        {
            string cpf = cmds.ExecuteScalar().ToString();
            saldoatual(cpf);

        }
    }
}
```

Logo depois temos o método saldoatual onde ele consulta no banco de dados qual o saldo disponível do cliente para que possa ser mostrado na tela.

```
private void saldoatual(string cpf)
{
    string Scon = "Data Source=DESKTOP-01QK6E3;Initial Catalog=PJI_BCB;Integrated Security=True";
    string Squery = "SELECT SALDO FROM BCB_PESSOA WHERE CPF= '" + cpf + "'";
    using (SqlConnection sqlcnn = new SqlConnection(Scon))
    {
        //ABRE UMA CONEXÃO
        sqlcnn.Open();
        //CRIA UM NOVO CMD PASSANDO A QUERY E A CONEXÃO PRA ELE
        using (SqlCommand cmd = new SqlCommand(Squery, sqlcnn))
        {
            string SALDO = cmd.ExecuteScalar().ToString();
            lblsaldo.Text = "R$" + SALDO;
            Dispose();


        }

    }
}
```

No meio de todos os códigos apresentados, o projeto em si contém métodos para facilitar a comunicação do backend e front com o banco de dados. Foi desenvolvido o método PORC que é a criptografia que recebe uma string que é a senha e ele me devolve uma senha criptografada.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

/// <summary>
/// CRIPTOGRAFIA MD5
/// ATENÇÃO: NÃO REVERSIVEL (BASE64 É REVERSIVEL)
/// FONTE: MICROSOFT,STACKOVERFLOW
/// </summary>
public class PORC
{
    public PORC()
    {

    }
    public string criptografar(string senha){
        try
        {
            System.Security.Cryptography.MD5 md5 = System.Security.Cryptography.MD5.Create();
            byte[] inputBytes = System.Text.Encoding.ASCII.GetBytes(senha);
            byte[] hash = md5.ComputeHash(inputBytes);
            System.Text.StringBuilder sb = new System.Text.StringBuilder();
            for (int i = 0; i < hash.Length; i++)
            {
                sb.Append(hash[i].ToString("X2"));
            }
            return sb.ToString(); // Retorna senha criptografada
        }
        catch (Exception)
        {
            return null; // Caso encontre erro retorna nulo
        }
    }
}
```

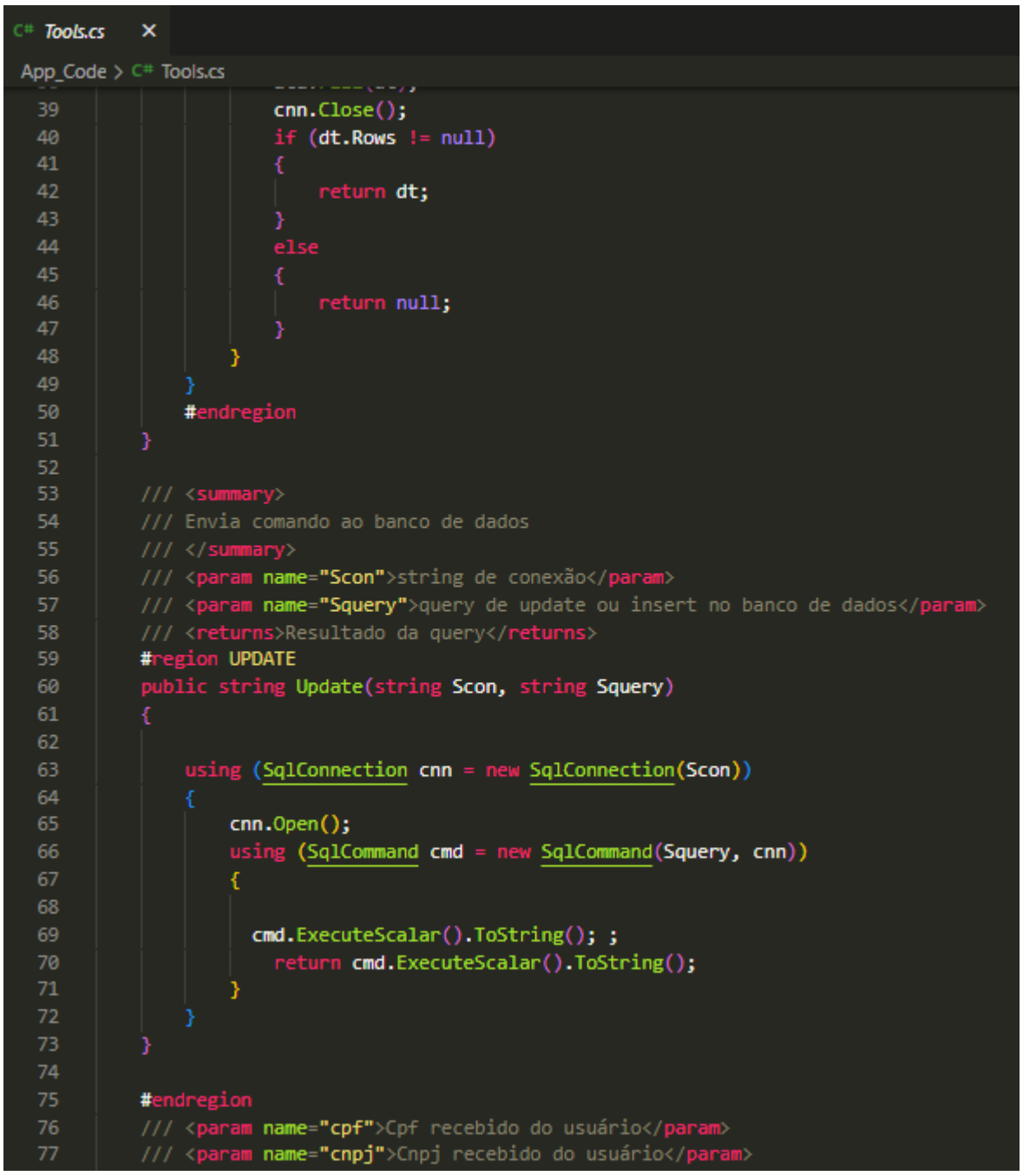
2.5 Classe tools

A classe Tools que é onde foi feito os métodos, variáveis as consultas no SQL e também consegue resolver alguns problemas caso acontece algo anormal nas informações dadas pelo usuário. Também nela é possível verificar se o usuário está conectado a internet o CPF, CNPJ, elimina as chances de ter dígitos iguais em sequência e também captura cada dígito do CPF.
Visão geral do código

C# Tools.cs

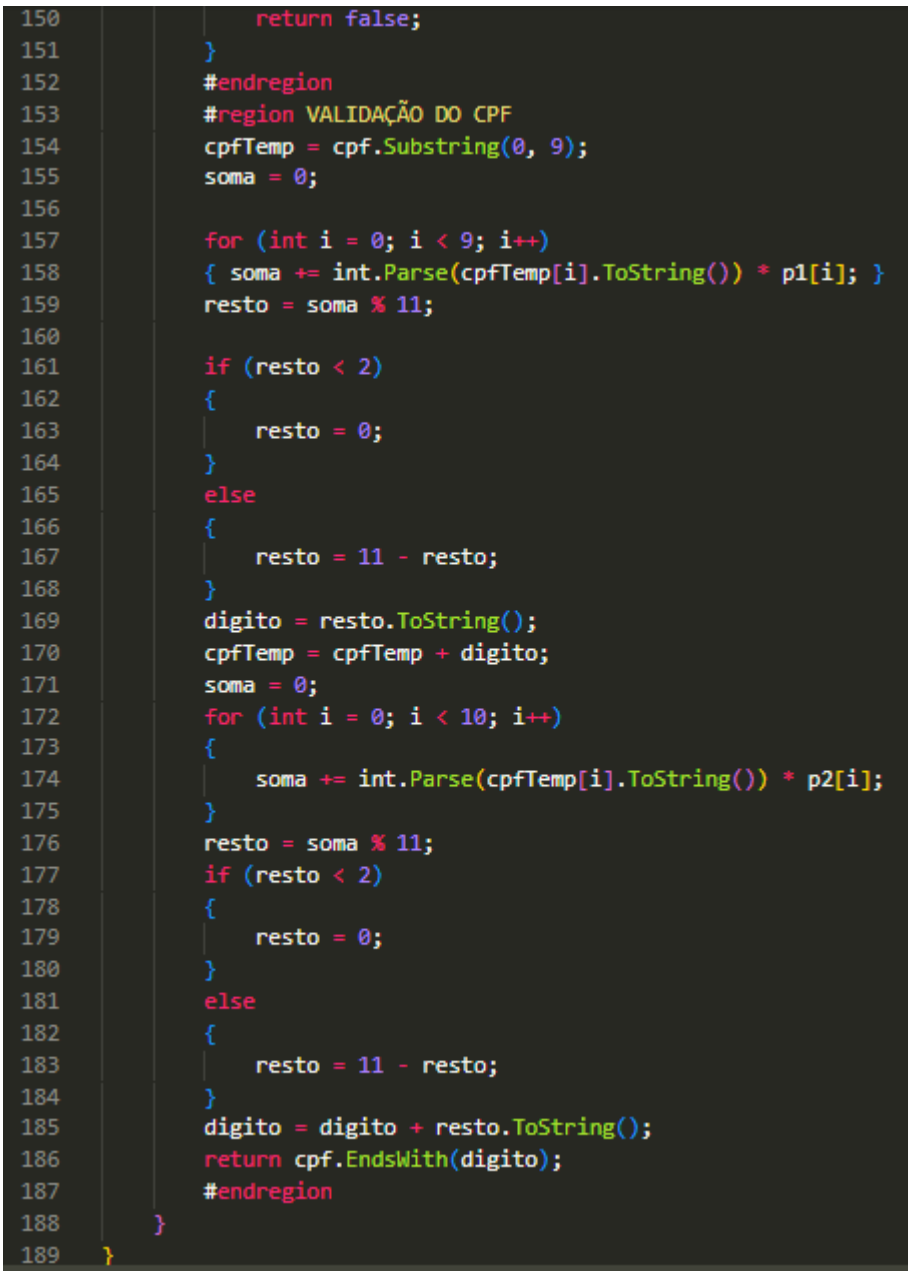
App_Code > C# Tools.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data;
using System.Net;
using System.Data.Sql;
using System.Data.SqlClient;
using System.Data.SqlTypes;
public class Validate
{
    public Validate()
    {
        //
        //
        //
    }
    /// <summary>
    /// Envia uma query pro banco de dados
    /// </summary>
    /// <param name="Scon">string de conexão ao banco</param>
    /// <param name="Squery">query do usuário</param>
    /// <returns>Tabela com os dados</returns>
    public DataTable Tabela_Generica(string Scon, string Squery)
    {
        #region CONEXÃO E QUERY
        DataTable dt = new DataTable();
        SqlDataAdapter dta = new SqlDataAdapter();


        using (SqlConnection cnn = new SqlConnection(Scon))
        {
            cnn.Open();

            using (SqlCommand cmd = new SqlCommand(Squery, cnn))
            {
                dta.SelectCommand = cmd;
                dta.Fill(dt);
                cnn.Close();
                if (dt.Rows != null)
```


Tools.cs
App_Code > Tools.cs
cnn.Close();
if (dt.Rows != null)
{
return dt;
}
else
{
return null;
}
}
}
#endregion
}
/// <summary>
/// Envia comando ao banco de dados
/// </summary>
/// <param name="Scon">string de conexão</param>
/// <param name="Squery">query de update ou insert no banco de dados</param>
/// <returns>Resultado da query</returns>
#region UPDATE
public string Update(string Scon, string Squery)
{
using (SqlConnection cnn = new SqlConnection(Scon))
{
cnn.Open();
using (SqlCommand cmd = new SqlCommand(Squery, cnn))
{
cmd.ExecuteScalar().ToString(); ;
return cmd.ExecuteScalar().ToString();
}
}
}
#endregion
/// <param name="cpf">Cpf recebido do usuário</param>
/// <param name="cnpj">Cnpj recebido do usuário</param>

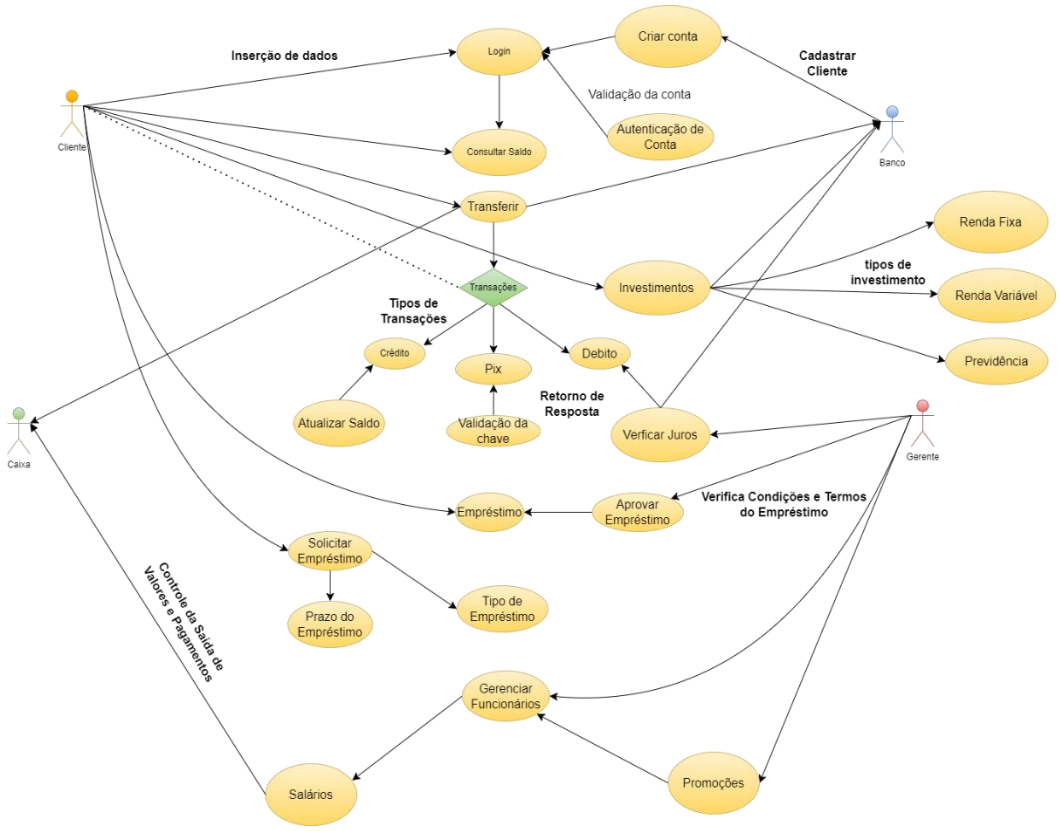
```
75      #endregion
76      /// <param name="cpf">Cpf recebido do usuário</param>
77      /// <param name="cnpj">Cnpj recebido do usuário</param>
78      /// <param name="Scon">String de conexão ao banco de dados</param>
79      /// <returns>Verdadeiro ou falso</returns>
80      public bool Verifica_Cpf(string cpf)
81      {
82
83          #region VERIFICA VALOR VAZIO
84          if (cpf == "")
85          {
86              return false;
87          }
88          cpf = cpf.Trim();
89          cpf = cpf.Replace("-", "").Replace(".", "");
90          #endregion
91          #region VERIFICA QUANTIDADE DE DIGITOS
92          if (cpf.Length != 11) { return false; }
93          #endregion
94          #region VARIAVEIS PARA CALCULOS
95          ///VARIAVEIS PARA SOMA E VALIDAÇÃO DOS NUMEROS
96          int soma = 0;
97          string digito = "";
98          string cpfTemp;
99          int resto;
100
101         int[] p1 = new int[9] { 10, 9, 8, 7, 6, 5, 4, 3, 2 };
102         int[] p2 = new int[10] { 11, 10, 9, 8, 7, 6, 5, 4, 3, 2 };
103         int[] n = new int[11];
104         ////////////////////////////////////////////
105         #endregion
106         #region ELIMINA CASOS DE DIGITOS IGUAIS
107         switch (cpf)
108         {
109             case "11111111111":
110                 return false;
111             case "00000000000":
112                 return false;
113             case "22222222222":
```

```
            case "00000000000":
                return false;
            case "22222222222":
                return false;
            case "33333333333":
                return false;
            case "44444444444":
                return false;
            case "55555555555":
                return false;
            case "66666666666":
                return false;
            case "77777777777":
                return false;
            case "88888888888":
                return false;
            case "99999999999":
                return false;
        }
        #endregion
        #region CAPTURA CADA DIGITO DOS 11 DIGITOS DO CPF
        try
        {
            //converte para int e captura o numero de uma determinada posição da variavel cpf
            n[0] = Convert.ToInt32(cpf.Substring(0, 1));
            n[1] = Convert.ToInt32(cpf.Substring(1, 1));
            n[2] = Convert.ToInt32(cpf.Substring(2, 1));
            n[3] = Convert.ToInt32(cpf.Substring(3, 1));
            n[4] = Convert.ToInt32(cpf.Substring(4, 1));
            n[5] = Convert.ToInt32(cpf.Substring(5, 1));
            n[6] = Convert.ToInt32(cpf.Substring(6, 1));
            n[7] = Convert.ToInt32(cpf.Substring(7, 1));
            n[8] = Convert.ToInt32(cpf.Substring(8, 1));
            n[9] = Convert.ToInt32(cpf.Substring(9, 1));
            n[10] = Convert.ToInt32(cpf.Substring(10, 1));
            ////////////////////////////////////////////////////////////////////////////////////////
        }
        catch
        {
            return false;
```



```
150             return false;
151         }
152         #endregion
153         #region VALIDAÇÃO DO CPF
154         cpfTemp = cpf.Substring(0, 9);
155         soma = 0;
156 
157         for (int i = 0; i < 9; i++)
158         { soma += int.Parse(cpfTemp[i].ToString()) * p1[i]; }
159         resto = soma % 11;
160 
161         if (resto < 2)
162         {
163             resto = 0;
164         }
165         else
166         {
167             resto = 11 - resto;
168         }
169         digito = resto.ToString();
170         cpfTemp = cpfTemp + digito;
171         soma = 0;
172         for (int i = 0; i < 10; i++)
173         {
174             soma += int.Parse(cpfTemp[i].ToString()) * p2[i];
175         }
176         resto = soma % 11;
177         if (resto < 2)
178         {
179             resto = 0;
180         }
181         else
182         {
183             resto = 11 - resto;
184         }
185         digito = digito + resto.ToString();
186         return cpf.EndsWith(digito);
187         #endregion
188     }
189 }
```

3 REQUISITOS DE PROGRAMAÇÃO ORIENTADA A OBJETOS



O Diagrama acima apresenta os requisitos de movimentações (histórico, depósito, e transferência) da conta bancária que foram elaborado pelo grupo. Começando com o histórico:

3.1Classe Histórico:

```
15  public partial class Historico : System.Web.UI.Page
16  {
17      protected void Page_Load(object sender, EventArgs e)
18      {
19          string cpf = Request.QueryString["id"];
20          carregatabela(cpf);
21      }
22      protected void btnvoltar_Click(object sender, EventArgs e)
23      {
24            string cpf = Request.QueryString["id"];
25          string  Scon = "Data Source=DESKTOP-01QK6E3;Initial Catalog=PJI_BCB;Integrated Security=True";
26              string  Squery = "SELECT SENHA FROM BCB_INFO WHERE CPF = '" + cpf+"'";
27           using (SqlConnection sqlcnn = new SqlConnection(Scon))
28             {
29                 //ABRE UMA CONEXÃO
30                 sqlcnn.Open();
31                 //CRIA UM NOVO CMD PASSANDO A QUERY E A CONEXÃO PRA ELE
32
33                 using (SqlCommand cmd = new SqlCommand(Squery, sqlcnn))
34                 {
35                     string cpft = cmd.ExecuteScalar().ToString();
36
37                     string id = cpft;
38                     Response.Redirect("~/Perfil.aspx?id="+id);
39                     Dispose();
40                 }
41      }
42  }
```

Aqui na classe (Histórico), quando o cliente busca a situação da sua conta pelo o seu CPF, o sistema faz uma requisição no banco para retornar o seu histórico de transferências.

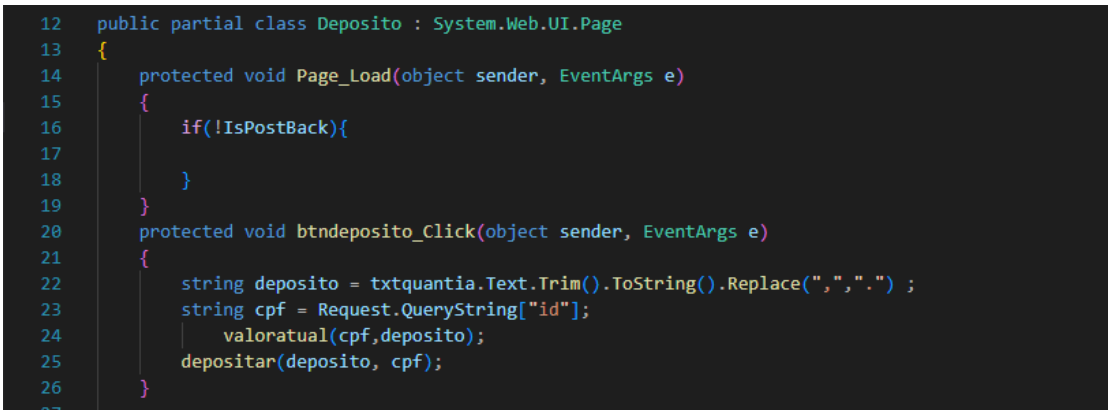
Após a requisição ser concluída, ele o retorna para a página que mostra o histórico de transações, e os dados desse evento, como o CPF, nome da pessoa, o valor que foi transferido.

```
        ///inicio do corpo///
        sql.AppendLine("<tbody>");
        foreach (DataRow dtr in dta.Rows)
        {
            sql.AppendLine("<tr class='linhatabela'>");
            sql.AppendLine("<div class='divtbl'><td>" + dtr["CPF"] + "</td></div>");
            sql.AppendLine("<div class='divtbl'><td>" + dtr["VALOR"] + "</td></div>");
            sql.AppendLine("<div class='divtbl'><td>" + dtr["NOME"] + "</td></div>");
            sql.AppendLine("<div class='divtbl'><td>" + dtr["TRANSAÇÃO"] + "</td></div>");
            sql.AppendLine("</tr>");
        }
        sql.AppendLine("</tbody>");
        sql.AppendLine("</table>");

        if (dta != null)
        {
            string rsql = sql.ToString();
            pnl.Controls.Clear();
            pnl.Controls.Add(new LiteralControl(rsql));
            pnl.Visible = true;
        }
    }
}
```

O sistema separa todos os dados da transição por linhas, para que fique o mais clara possível a compreensão do usuário.

Após isso vem funcionalidade do usuário realizar um depósito em sua conta.

3.1 Classe Depósito:



```
public partial class Deposito : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if(!IsPostBack){

        }
    }
    protected void btndeposito_Click(object sender, EventArgs e)
    {
        string deposito = txtquantia.Text.Trim().ToString().Replace(",",".") ;
        string cpf = Request.QueryString["id"];
            valoratual(cpf,deposito);
        depositar(deposito, cpf);
    }
```

O código acima mostra a classe (Deposito), onde assim que o usuário clicar no botão de depósito (btndeposito_Click). O Banco vai fazer a requisição dos dados do cliente a partir do seu CPF. Onde mostra o valor que ele tem na sua conta e um campo do quanto ele vai querer depositar.

```
28      private void depositar(string deposito, string cpf)
29      {
30          //
31          string Scon = "Data Source=DESKTOP-01QK6E3;Initial Catalog=PJI_BCB;Integrated Security=True";
32          string Squery = "UPDATE BCB_PESSOA SET SALDO = SALDO +" + deposito + "  WHERE CPF = '" + cpf + "'";
33          try
34          {
35              using (SqlConnection sqlcnn = new SqlConnection(Scon))
36              {
37                  //ABRE UMA CONEXÃO
38                  sqlcnn.Open();
39                  //CRIA UM NOVO CMD PASSANDO A QUERY E A CONEXÃO PRA ELE
40
41                  using (SqlCommand cmd = new SqlCommand(Squery, sqlcnn))
42                  {
43                      cmd.ExecuteScalar().ToString();
44                      if (cmd.ToString().Contains("-"))
45                      {
46                          resultado.Text = "SALDO NEGATIVO";
47                      }
48                      else
49                      {
50                          valoratual(cpf,deposito);
51                      }
52                      Dispose();
53                  }
54              }
55          }
56          catch(Exception e){
57          }
58      }
```

Na classe (depositar), assim que fizer a conexão com o banco, o sistema retorna o documento do usuário e a situação do seu saldo bancário.

```
60      private void registrar(string deposito, string cpf)
61      {
62          string Scon = "Data Source=DESKTOP-01QK6E3;Initial Catalog=PJI_BCB;Integrated Security=True";
63          string Squery = "SELECT NOME FROM BCB_INFO WHERE CPF= '" + cpf + "'";
64          using (SqlConnection sqlcnn = new SqlConnection(Scon))
65          {
66              //ABRE UMA CONEXÃO
67              sqlcnn.Open();
68              //CRIA UM NOVO CMD PASSANDO A QUERY E A CONEXÃO PRA ELE
69              using (SqlCommand cmd = new SqlCommand(Squery, sqlcnn))
70              {
71                  string pessoa = cmd.ExecuteScalar().ToString();
72                  string nome = pessoa;
73                  Scon = "Data Source=DESKTOP-01QK6E3;Initial Catalog=PJI_BCB;Integrated Security=True";
74                  Squery = "INSERT INTO BCB_HIST (CPF,VALOR,NOME,TRANSAÇÃO) VALUES ('"+cpf+"','"+deposito+"','"+nome+"','DEPOSITO')";
75
76                  Validate Cadastrar = new Validate();
77                  DataTable dta = Cadastrar.Tabela_Generica(Scon, Squery);
78                  if (dta.Rows.Count == 0)
79                  {
80                      Scon = "Data Source=DESKTOP-01QK6E3;Initial Catalog=PJI_BCB;Integrated Security=True";
81                      Squery = "SELECT * FROM BCB_HIST where NOME = '" + pessoa + "'";
82
83                      Validate consultarregistro = new Validate();
84                      DataTable dtal = consultarregistro.Tabela_Generica(Scon, Squery);
85                      if (dtal.Rows.Count != 0)
86                      {
87                          resultado.Text = "DEPOSITO REALIZADO";
88                      }
89                      else {
90                          resultado.Text = "exceção";
91                      }
92                  }
93              }
94          }
95      }
```

A situação de deposito é retornada com duas possibilidades, “DEPOSITO REALIZADO” ou “exceção”, se caso houver algum problema nesse processo.

```
97      private void valoratual(string cpf,string deposito) {
98
99          string Scon = "Data Source=DESKTOP-01QK6E3;Initial Catalog=PJI_BCB;Integrated Security=True";
100         string Squery = "SELECT SALDO FROM BCB_PESSOA WHERE CPF = '" + cpf + "'";
101         using (SqlConnection sqlcnn = new SqlConnection(Scon))
102         {
103             //ABRE UMA CONEXÃO
104             sqlcnn.Open();
105             //CRIA UM NOVO CMD PASSANDO A QUERY E A CONEXÃO PRA ELE
106             using (SqlCommand cmd = new SqlCommand(Squery, sqlcnn))
107             {
108                 string pessoa = cmd.ExecuteScalar().ToString();
109
110                 btnvoltar.Visible = true;
111                 btndeposito.Visible = false;
112                 registrar(deposito, cpf);
113             }
114         }
115     }
```

Depois do processo de depósito ser concluído, ele retorna para a tela em que mostra o quanto o usuário tem na conta dele.

Depois, disso caso o usuário queira fazer uma transferência

3.2 Classe Transferência:

A conexão com o banco é feita, buscando os dados da conta do usuário.

```
11  public partial class Transferencias : System.Web.UI.Page
12  {
13      protected void Page_Load(object sender, EventArgs e)
14      {
15          if(!IsPostBack){
16
17          }
18      }
19      protected void btnconsulta_Click(object sender, EventArgs e)
20      {
21          string chave = txtchave.Text.Trim();
22          if (chave != "")
23          {
24              consultarchave(chave);
25          }
26          else {
27              status.Text = "Campo Vazio";
28          }
29      }
```

A partir, disso, abre a tela para que o usuário passe a chave PIX de transferência.

```
31      private void consultarchave(string chave) {
32          string Scon = "Data Source=DESKTOP-01QK6E3;Initial Catalog=PJI_BCB;Integrated Security=True";
33          string Squery = "select cpf from bcb_info where cpf = '"+chave+"'";
34          Validate vchave = new Validate();
35          DataTable dta = vchave.Tabela_Generica(Scon, Squery);
36          if (dta.Rows.Count != 0)
37          {
38              using (SqlConnection sqlcnn = new SqlConnection(Scon))
39              {
40                  //ABRE UMA CONEXÃO
41                  sqlcnn.Open();
42                  //CRIA UM NOVO CMD PASSANDO A QUERY E A CONEXÃO PRA ELE
43                  Scon = "Data Source=DESKTOP-01QK6E3;Initial Catalog=PJI_BCB;Integrated Security=True";
44                  Squery = "SELECT NOME FROM BCB_INFO WHERE CPF = '" + chave + "'";
45                  using (SqlCommand cmd = new SqlCommand(Squery, sqlcnn))
46                  {
47                      string cpft = cmd.ExecuteScalar().ToString();
48
49                      status.Text = cpft;
50                      Dispose();
51                  }
52                  Squery = "SELECT EMAIL FROM BCB_INFO WHERE CPF = '" + chave + "'";
53                  using (SqlCommand cmd = new SqlCommand(Squery, sqlcnn))
54                  {
55
56                      string cpft = cmd.ExecuteScalar().ToString();
57                      email.Text = cpft;
58                      btnfazerpix.Visible = true;
59                      Dispose();
60                  }
61              }
62          }
63          else {
64              btnfazerpix.Visible = false;
65              email.Text = "";
66              status.Text = "CHAVE NÃO EXISTENTE";
67          }
68      }
```

Após o usuário inserir a chave PIX, e o sistema abrir a conexão com o banco para verificar a chave, e os dados da pessoa que receberá a transferência. Imprime o resultado da consulta na tela para que o usuário confirme se os dados estão certo, e informar o valor da transferência.

```
71      protected void btnfazerpix_Click(object sender, EventArgs e)
72      {
73          string cpfn = Request.QueryString["id"];
74          string cpft = txtchave.Text.ToString();
75
76          Response.Redirect("~/Transação.aspx?cpfd=" + cpft + "&cpfn=" + cpfn);
77      }
78
```

Após isso, o usuário pode clicar no botão para efetuar a transferência.

4 CONSIDERAÇÃO FINAIS

O desenvolvimento do projeto foi um desafio pois foi feito em uma linguagem que tivemos pouco contato mais que pode cobrir com as nossas necessidades. O grupo desenvolveu bem as ideias e grande parte foi colocado no desenvolvimento do projeto. O sistema bancario em si é muito complexo e com muitas funções que precisam ser bem planejadas e bem levantadas para ser desenvolvidas. O grupo procurou fazer o desenvolvimento mais basico possível perante o que foi abordado no tema.

5 BIBLIOGRAFIA

- https://engsoftmoderna.info/
- https://www.youtube.com/@bosontreinamentos
- https://creately.com/blog/pt/diagrama/guia-de-tipos-de-diagramas-uml-aprenda-sobre-todos-os-tipos-de-diagramas-uml-com-exemplos/
- https://www.c-sharpcorner.com/
- Livro The C# Programming Yellow Book
- youtube/cursoemvideo
- https://blog.nubank.com.br/sistema-financeiro-nacional/
- https://ri.bb.com.br/o-banco-do-brasil/estrutura-organizacional/

APENSO 1 – CRONOGRAMA DE ENTREGA DE ATIVIDADES.

#	Descrição	Data		Prazo do cronograma em semanas																	
		Início	Término		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	Definição do tema e planejamento inicial	12/03/2023	-------	P	X																
				R	X																
2	Levantamento dos requisitos	04/04/2023	07/04/2023	P	X																
				R	X																
3	Desenvolvimento do código	18/04/2023	04/05/2023	P				X													
				R							X										
4	Documentação	05/05/2023	12/05/2023	P		X															
				R			X														
5				P																	
				R																	
6				P																	
				R																	
7	Entrega do projeto final e apresentação	12/05/2023	16/05/2023	P	X																
				R	X																

OBS:

1) **P =** previsto; **R =** realizado