

**CURSO DE CAPACITAÇÃO TECNOLÓGICA EM ENGENHARIA E
DESENVOLVIMENTO DE SOFTWARE**

**GRUPO 4
HECTOR SHIN EMURA
HIDEKI GOECKING HANAOKA**

**PROJETO FINAL
Aplicação para controle de finanças pessoais**

**SÃO PAULO - SP
2022**

SUMÁRIO

1. OBJETIVO	3
2. DESCRIÇÃO DO PROJETO	3
Design	4
Banco de dados	8
Back-end	8
Front-end	13
3. CRONOGRAMA COMPLETO	16
4. REQUISITOS FUNCIONAIS	17
5. REQUISITOS NÃO FUNCIONAIS	17
6. SEMANA 1	17
O que deu certo	17
O que deu errado	17
Aprendizado	17
Auto-avaliação	17
7. SEMANA 2	18
O que deu certo	18
O que deu errado	18
Aprendizado	18
Auto-avaliação	18
8. SEMANA 3	19
O que deu certo	19
O que deu errado	19
Aprendizado	19
Auto-avaliação	19
9. SEMANA 4	19
O que deu certo	19
O que deu errado	19
Aprendizado	19
Auto-avaliação	20
10. SEMANA 5	20
O que deu certo	20
O que deu errado	20
Aprendizado	20
Auto-avaliação	20

1. OBJETIVO

A equipe decidiu desenvolver um Aplicativo para Controle de Finanças. O aplicativo será em plataforma desktop e usaremos, sobretudo, os conhecimentos adquiridos ao longo das aulas para sua execução.

2. DESCRIÇÃO DO PROJETO

O aplicativo terá uma interface em que o usuário poderá visualizar como seu dinheiro foi gasto no mês, para isso ele deverá registrar seus recebimentos e gastos, que serão armazenados em um banco de dados. Com isso, o aplicativo realizará os cálculos e facilitará a visualização dos gastos por categoria do usuário.

Com isso, decidiu-se que os principais requisitos que tornam o aplicativo funcional são:

- a. Controle de gastos total do usuário - Visualização geral dos gastos por tempo, sendo possível a alternância do intervalo (Por semana, por mês, por ano, etc). O gasto total será calculado automaticamente a partir dos itens adicionados pelo próprio usuário, e cada item terá informações como: Data, Valor, Descrição, Categoria.
- b. Controle de gastos por categoria - Lazer, Alimentação, Moradia, Supermercado, Mobilidade, Saúde, Educação, Investimentos, Outros. Assim como em (a), será possível a visualização dos gastos em determinada categoria por intervalo de tempo especificado pelo usuário.
- c. Controle de recebimentos - Haverá um mecanismo de carteira virtual, em que o usuário poderá especificar o salário, e esse valor será adicionado mensalmente. Além disso, o usuário terá a liberdade de adicionar rendas adicionais. Assim como em (a), os recebimentos também poderão ser controlados como itens.
- d. Estabelecimento de limite - Assim como em aplicativos de banco, o usuário poderá estabelecer um limite para seus gastos. Esse limite poderá ser estabelecido por categoria.

Para implementar essas funções serão utilizadas as seguintes plataformas e linguagens de programação:

Azure DevOps

Linguagem C# para o backend

WPF para o Front-end

Python para algumas determinadas funcionalidades , usando biblioteca

MATPLOTLIB

Banco de Dados SQLite

Metodologia Agile

Figma

Design

Para a construção do design, utilizou-se a ferramenta "Figma".

Criou-se, um diagrama do aplicativo, que estabelece o fluxo de interface vista pelo usuário e que permite a visualização geral do projeto, contendo todas as funcionalidades (tanto as funcionais e não funcionais) que serão implementadas no aplicativo.

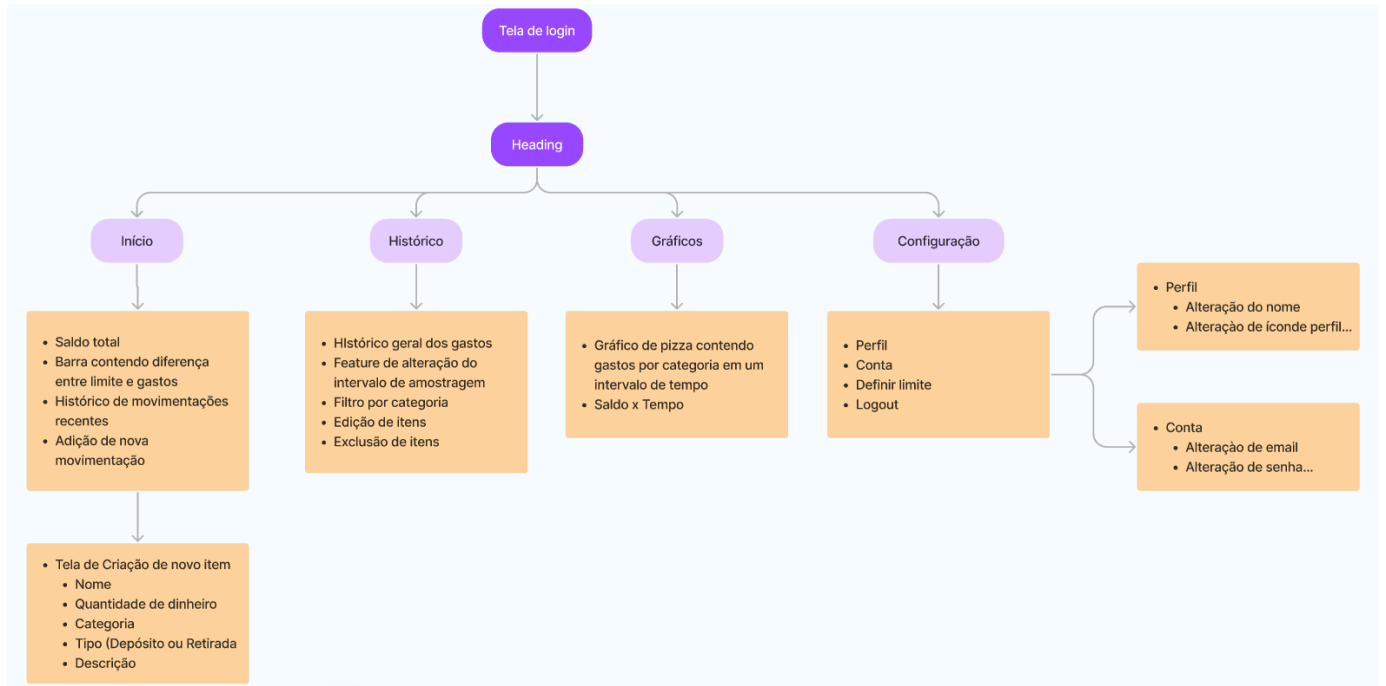


Figura 1 - Fluxo da interface


Link para o Diagrama:

<https://www.figma.com/file/P4o1Gulj8NQmKt3R5CXv8k/Untitled?node-id=0%3A1&t=igNUSBc6QO0WuWV3-1>

A partir das informações estabelecidas no diagrama, conseguiu-se montar uma interface para o aplicativo condizente com os propósitos do projeto

Link para o design das interfaces do aplicativo:

<https://www.figma.com/file/hu67Y75e2EBqj1O7IgEBgv/Untitled?node-id=0%3A1&t=zBCq6dWqSX6efbp0-1>



CashIn

Login

Email


Senha

Entrar

ou

Criar Conta

Figura 2- Tela de Login



CashIn

Criar Conta

Nome de Usuário

Email

Senha

Registrar

Figura 3- Tela de Registro



Figura 6- Tela de Gráficos

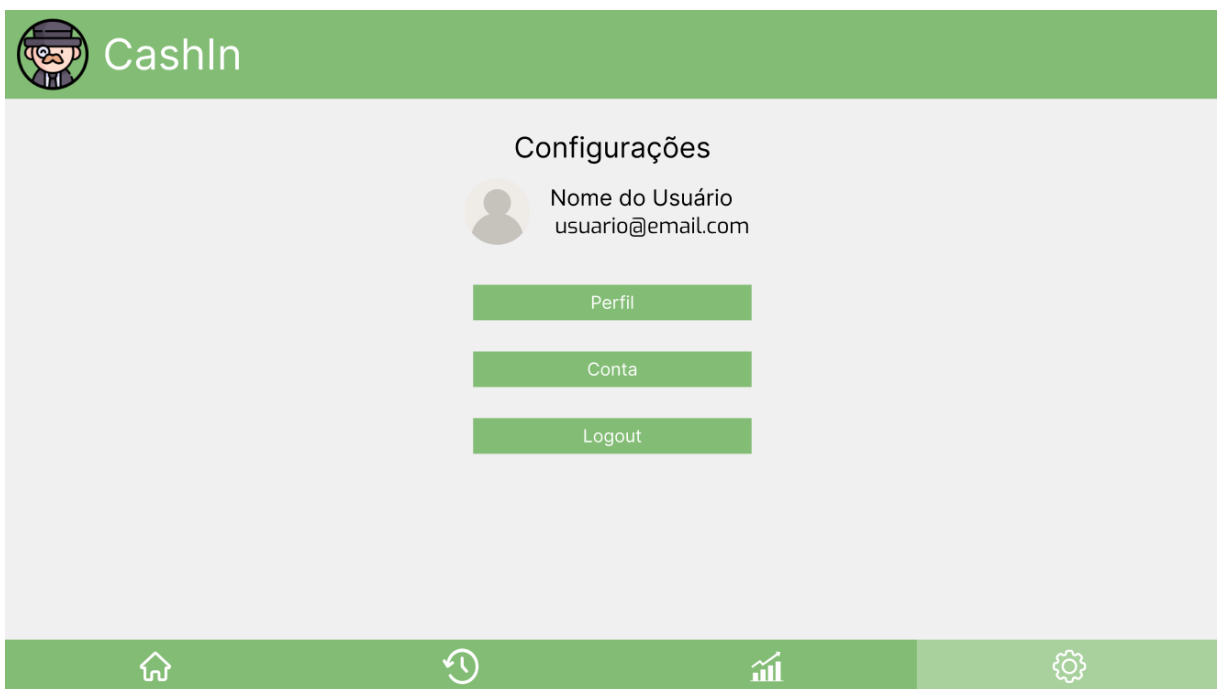


Figura 7 - Tela de Configurações

Banco de dados

O serviço de banco de dados utilizado no projeto foi o SQL Server. O servidor utilizado foi do próprio CTEDS e montamos uma estrutura utilizando o usuário 12.

A estrutura do banco foi montada em duas tabelas:

- Uma contendo informações de usuário, tais como : ID, Nome, Email e Senha
- Outra contendo informações de todas as transações: Título, Data, Descrição, Tipo, categoria, Valor e ID do usuário autor.

	ID	Name	Email	Password	Balance	Limit
►	6164d578-ade2...	Hideki	hideki@email.c...	senha	1000	500
	561fbc3a-6fc0-...	Admin	admin@email.c...	admin123	1500	500
*	NULL	NULL	NULL	NULL	NULL	NULL

Figura 8 - Tabela "Users"

/

	ID	Title	Value	Description	Date	Type	Category
►	d24-ae4629217492	Casas Bahia	3000	Notebook	14/12/2022 16:5...	Gasto	Lazer
	561fbc3a-6fc0-...	Dívida Leandro	500	Empréstimo pa...	14/12/2022 16:5...	Recebimento	Mercado
	6164d578-ade2...	Almoço	30	McDonalds	14/12/2022 17:3...	Gasto	Mercado
	6164d578-ade2...	Casas Bahia	110	Ventilador	14/12/2022 17:3...	Gasto	Lazer
	6164d578-ade2...	Aposta	300	Aposta em Mar...	14/12/2022 17:3...	Recebimento	Lazer
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 9 -Tabela "Transactions"

Back-end

A autenticação do usuário foi realizada da seguinte forma, utilizando-se do banco de dados acima.

```
private void Login_Click(object sender, RoutedEventArgs e)
{
    string stringConexao = "Server=labsoft.pcs.usp.br; Initial Catalog=db_12; User Id=usuario_12; Password=9077879676;";
    SqlConnection con = new SqlConnection(stringConexao);
    try
    {
        if (con.State == System.Data.ConnectionState.Closed)
            con.Open();
        String query = "SELECT COUNT(*) FROM [dbo].[Users] WHERE Email=@Email AND Password=@Password";
        SqlCommand sqlCmd = new SqlCommand(query, con);
        sqlCmd.CommandType = System.Data.CommandType.Text;
        sqlCmd.Parameters.AddWithValue("@Email", EmailForm.Text);
        sqlCmd.Parameters.AddWithValue("@Password", PasswordForm.Password);
        int count = Convert.ToInt32(sqlCmd.ExecuteScalar());
    }
}
```



```

if (count == 1)
{
    this.Opacity = 0.5;
    Sucess.IsOpen = true;

    String query2 = "SELECT ID, Name, Email, Balance, Limit FROM [dbo].[Users] WHERE Email=@Email AND Password=@Password";
    SqlCommand cmd = new SqlCommand(query2, con);
    cmd.Parameters.AddWithValue("@Email", EmailForm.Text);
    cmd.Parameters.AddWithValue("@Password", PasswordForm.Password);
    SqlDataReader rdr;

    using (cmd)
    {
        rdr = cmd.ExecuteReader();
        while (rdr.Read())
        {
            User.Id = (Guid)rdr["ID"];
            User.Nome = rdr["Name"].ToString();
            User.Email = rdr["Email"].ToString();
            User.Saldo = Convert.ToSingle(rdr["Balance"]);
            User.Limite = Convert.ToSingle(rdr["Limit"]);
        }
        rdr.Close();
    }

    String query3 = "SELECT * FROM [Transactions] WHERE ID=@Id";
    SqlCommand cmd2 = new SqlCommand(query3, con);
    cmd2.Parameters.AddWithValue("@Id", User.Id);
    SqlDataReader rdr2;

```

```

        using (cmd2)
        {
            rdr2 = cmd2.ExecuteReader();
            while (rdr2.Read())
            {
                Item NewItem = new Item()
                {
                    Id = (Guid)rdr2["ID"],
                    Titulo = rdr2["Title"].ToString(),
                    Valor = Convert.ToSingle(rdr2["Value"]),
                    Descricao = rdr2["Description"].ToString(),
                    Tipo = rdr2["Type"].ToString(),
                    Categoria = rdr2["Category"].ToString()
                };
                User.ListaItens.Add(NewItem);
            }
            rdr2.Close();
        }
    }
    else
    {
        this.Opacity = 0.5;
        Fail.IsOpen = true;
    }
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
finally
{
    con.Close();
}
}

```

Figura 10 - Código de Autenticação

Nesse código, estabeleceu-se uma conexão com o banco de dados a partir da variável `stringConexão`. Vale ressaltar que o método utilizado não é uma boa prática, pois a string de Conexão está totalmente visível e exposta a ataques.

Em seguida, abre-se a conexão e cria-se a query SQL com os parâmetros `@Email` e `@Password`. Com isso, conseguiu-se criar uma proteção para ataques de SQL Injection. Os

parâmetros mencionados são lidos do TextBox, que é uma entrada do usuário. Esse valor, então, é armazenado e, então, roda-se a query de busca no banco de dados pelos valores inseridos pelo usuário

A lógica do if e else indica se os dados digitados pelo usuário foram encontrados ou não no banco de dados. Caso a resposta seja afirmativa, adquire-se os dados como: Email, Nome, Saldo e Limite e coloca-se em uma variável global que pode ser acessada por outras janelas. Além dos dados da conta, faz-se também uma busca na tabela de “Transactions” e adquire-se todas as transações cadastradas pelo usuário. Esses itens são, então, armazenados em uma lista que também é global e pode ser acessada por outras janelas

```
private void RegisterDB_Click(object sender, RoutedEventArgs e)
{
    string stringConexao = "Server=labsoft.pcs.usp.br; Initial Catalog=db_12; User Id=usuario_12; Password=9877879676;";
    SqlConnection con = new SqlConnection(stringConexao);
    try
    {
        if (con.State == System.Data.ConnectionState.Closed)
            con.Open();
        String query = "SELECT COUNT(*) FROM [dbo].[Users] WHERE Email=@Email";
        SqlCommand sqlCommand = new SqlCommand(query, con);
        sqlCommand.CommandType = System.Data.CommandType.Text;
        sqlCommand.Parameters.AddWithValue("@Email", EmailForm.Text);
        int count = Convert.ToInt32(sqlCmd.ExecuteScalar());
        if (count == 1)
        {
            this.Opacity = 0.5;
            AlreadyRegistered.IsOpen = true; //Popup
        }
        else
        {
            this.Opacity = 0.5;
            SuccessRegister.IsOpen = true; // Popup
            String queryCreateUser = "INSERT INTO [dbo].[Users]([ID],[Name],[Email],[Password],[Balance],[Limit]) VALUES (@ID, @Name, @Email, @Password, @Balance, @Limit)";
            SqlCommand sqlCommandRegisterUser = new SqlCommand(queryCreateUser, con);
            sqlCommandRegisterUser.CommandType = System.Data.CommandType.Text;
            sqlCommandRegisterUser.Parameters.AddWithValue("@ID", Guid.NewGuid());
            sqlCommandRegisterUser.Parameters.AddWithValue("@Name", NameForm.Text);
            sqlCommandRegisterUser.Parameters.AddWithValue("@Email", EmailForm.Text);
            sqlCommandRegisterUser.Parameters.AddWithValue("@Password", PasswordForm.Password);
            sqlCommandRegisterUser.Parameters.AddWithValue("@Balance", Convert.ToSingle(0));
            sqlCommandRegisterUser.Parameters.AddWithValue("@Limit", Convert.ToSingle(0));
            sqlCommandRegisterUser.ExecuteNonQuery();
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        con.Close();
    }
}
```

Figura 11 - Código de Registro

Para o registro do usuário foi montado o código acima, que é IDEM à lógica de Login, porém, agora, com a query de inserção no banco de dados.

```
private void NewItemDB_Click(object sender, RoutedEventArgs e)
{
    string stringConexao = "Server=labsoft.pcs.usp.br; Initial Catalog=db_12; User Id=usuario_12; Password=9877879676;";
    SqlConnection con = new SqlConnection(stringConexao);
    try
    {
        if (con.State == System.Data.ConnectionState.Closed)
            con.Open();
        String queryCreateTransaction = "INSERT INTO [dbo].[Transactions]([ID],[Title],[Value],[Description],[Date],[Type],[Category]) VALUES (@ID, @Title, @Value, @Description, @Date, @Type, @Category)";
        SqlCommand sqlCommandCreateTransaction = new SqlCommand(queryCreateTransaction, con);
        sqlCommandCreateTransaction.CommandType = System.Data.CommandType.Text;
        Guid id = User.Id;
        string data = DateTime.Now.ToString("dd/MM/yyyy HH:mm:ss");
        string titulo = TituloForm.Text;
        string categoria = CategoriaForm.Text;
        string desc = DescricaoForm.Text;
        string tipo = TipoForm.Text;
        float valor = Convert.ToSingle(ValorForm.Text);
        sqlCommandCreateTransaction.Parameters.AddWithValue("@ID", id);
        sqlCommandCreateTransaction.Parameters.AddWithValue("@Date", data);
        sqlCommandCreateTransaction.Parameters.AddWithValue("@Title", titulo);
        sqlCommandCreateTransaction.Parameters.AddWithValue("@Value", valor);
        sqlCommandCreateTransaction.Parameters.AddWithValue("@Category", categoria);
        sqlCommandCreateTransaction.Parameters.AddWithValue("@Description", desc);
        sqlCommandCreateTransaction.Parameters.AddWithValue("@Type", tipo);
        int count = 0;
    }
}
```

```

try
{
    sqlCmdCreateTransaction.ExecuteScalar();
    count = 1;
}
catch
{
    count = 0;
}
if (count == 1)
{
    BalanceController.ActualBalance(tipo, valor);
    LimitController.ActualLimit(tipo, valor);
    Item newItem = new()
    {
        Id = id,
        Titulo = titulo,
        Valor = valor,
        Descricao = desc,
        Tipo = tipo,
        Categoria = categoria
    };
    User.ListaItens.Add(newItem);
    SucessTransaction.IsOpen = true;
}
else
{
    SucessTransaction.IsOpen = false;
}
}

catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
finally
{
    con.Close();
}
}

```

Figura 12 - Código de Cadastro de NovoItem

A lógica para cadastro de novo item também é IDEM às anteriores. No entanto, dessa vez, deve-se atualizar, em tempo real, a lista de itens cadastrados pelo usuário logado. Portanto, cria-se uma lógica de adição do novo item à essa lista.

Além disso, foi construído uma lógica para a atualização do valor do saldo e do limite do usuário no banco de dados

```
private void CadastrarLimite_Click(object sender, RoutedEventArgs e)
{
    string stringConexao = "Server=labsoft.pcs.usp.br; Initial Catalog=db_12; User Id=usuario_12; Password=9077879676;";
    SqlConnection con = new SqlConnection(stringConexao);
    try
    {
        if (con.State == System.Data.ConnectionState.Closed)
            con.Open();
        float newLimite = (float)Convert.ToSingle(LimiteForm.Text);
        User.Limite = newLimite;

        string query = "UPDATE [dbo].[Users] SET Limit=@Limit WHERE ID=@ID";
        SqlCommand sqlCmd = new SqlCommand(query, con);
        sqlCmd.CommandType = System.Data.CommandType.Text;
        sqlCmd.Parameters.AddWithValue("@Limit", Convert.ToSingle(LimiteForm.Text));
        sqlCmd.Parameters.AddWithValue("@ID", User.Id);
        int count = 0;
        try
        {
            sqlCmd.ExecuteNonQuery();
            count = 1;
        }
        catch
        {
            count = 0;
        }
        if (count == 1)
        {
            SucessLimit.IsOpen = true;
        }
        else
        {
            SucessLimit.IsOpen = false;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        con.Close();
    }
}
```

Figura 13 - Código de Atualização de saldo

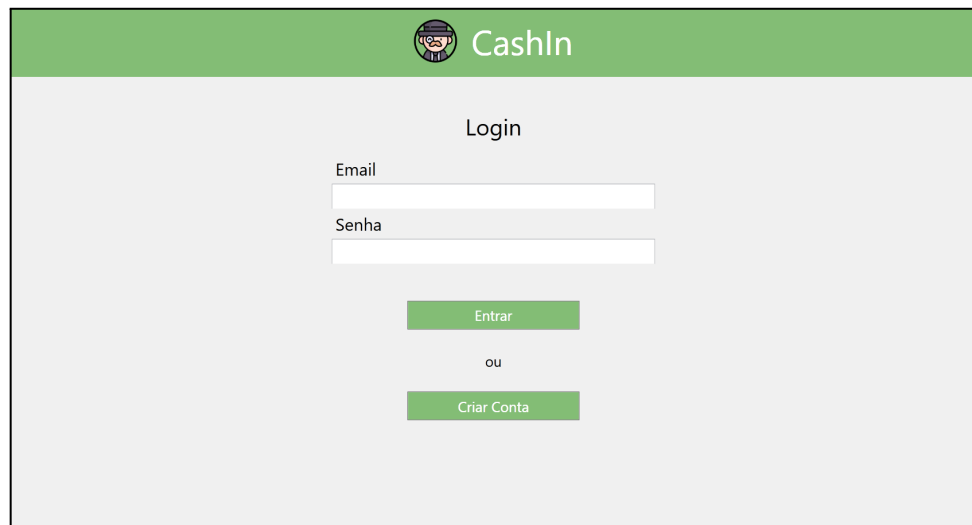
```
private void CadastrarSaldo_Click(object sender, RoutedEventArgs e)
{
    string stringConexao = "Server=labsoft.pcs.usp.br; Initial Catalog=db_12; User Id=usuario_12; Password=9077879676;";
    SqlConnection con = new SqlConnection(stringConexao);
    try
    {
        if (con.State == System.Data.ConnectionState.Closed)
            con.Open();
        float newSaldo = Convert.ToSingle(SaldoForm.Text);
        User.Saldo = newSaldo;

        string query = "UPDATE [dbo].[Users] SET Balance=@Balance WHERE ID=@ID";
        SqlCommand sqlCmd = new SqlCommand(query, con);
        sqlCmd.CommandType = System.Data.CommandType.Text;
        sqlCmd.Parameters.AddWithValue("@Balance", Convert.ToSingle(SaldoForm.Text));
        sqlCmd.Parameters.AddWithValue("@ID", User.Id);
        int count = 0;
        try
        {
            sqlCmd.ExecuteNonQuery();
            count = 1;
        }
        catch
        {
            count = 0;
        }
        if (count == 1)
        {
            SucessBalance.IsOpen = true;
        }
        else
        {
            SucessBalance.IsOpen = false;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        con.Close();
    }
}
```

Figura 14 - Código de Atualização de limite

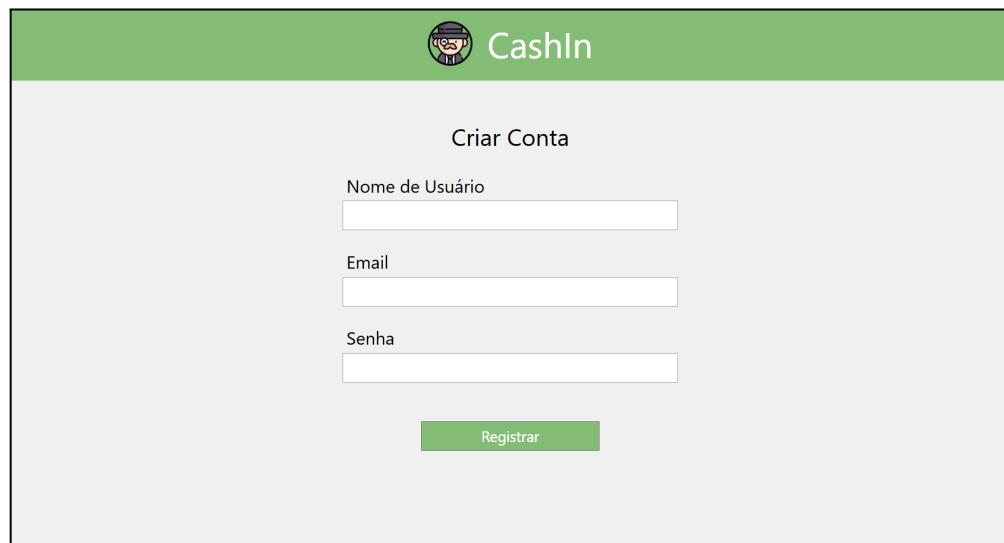
Front-end

Utilizando o XAML, montamos as seguintes interfaces



The login screen features a green header bar with a circular icon of a person with glasses and the text "CashIn". Below the header, the title "Login" is centered. The form consists of two input fields: "Email" and "Senha" (Password). Below the password field is a green button labeled "Entrar" (Login). Underneath the button is the word "ou" (or), followed by another green button labeled "Criar Conta" (Create Account).

Figura 15 - Tela de login



The create account screen features a green header bar with a circular icon of a person with glasses and the text "CashIn". Below the header, the title "Criar Conta" (Create Account) is centered. The form consists of three input fields: "Nome de Usuário" (Username), "Email", and "Senha" (Password). Below the password field is a green button labeled "Registrar" (Register).

Figura 16 - Tela de cadastro



Figura 17 - Tela inicial

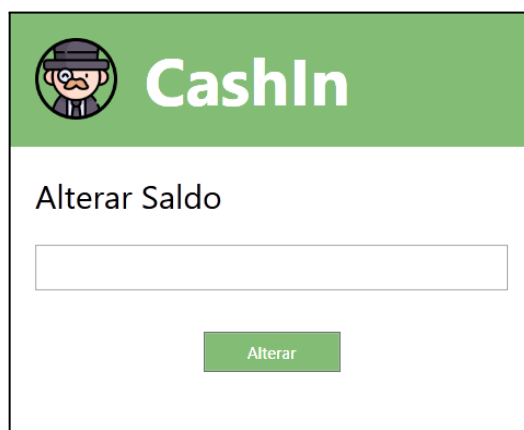


Figura 18 - Tela de Alteração de Saldo

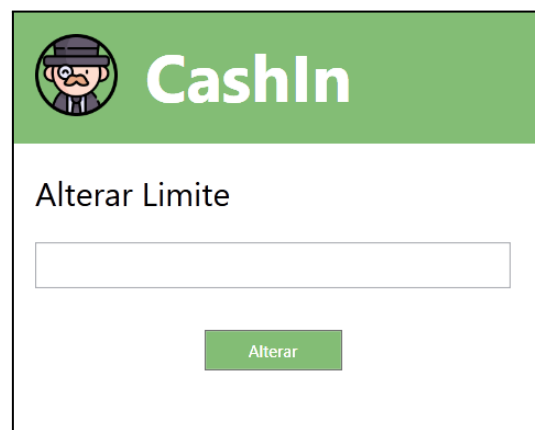


Figura 19 - Tela de Alteração de Limite



CashIn

Título

Quantidade de Dinheiro

Descrição

Tipo de Transação Categoria

Escolha uma opção Escolha uma opção

Adicionar Item

Figura 20 - Tela de Adição de Nova Transação

Além das telas, foi necessário escrever uma lógica para a atualização em tempo real da tela principal do usuário.

```
public partial class AccountWindow : Window
{
    public AccountWindow()
    {
        InitializeComponent();
        BoxLimit.Text = $"R$ {User.Limite}";
        BoxBalance.Text = $"R$ {User.Saldo}";
        User.ListaItens.Reverse();
        foreach (Item _item in User.ListaItens)
        {
            Button _button = new Button();
            _button.Height = 75;
            _button.FontSize = 20;
            _button.Background = Brushes.White;
            _button.Cursor = Cursors.Hand;
            _button.HorizontalAlignment = HorizontalAlignment.Stretch;
            PainelDeTransacoes.Children.Add(_button);

            StackPanel _stackPanel1 = new StackPanel();
            _stackPanel1.Orientation = Orientation.Horizontal;
            _stackPanel1.Margin = new Thickness(0, 0, 0, 0);
            _stackPanel1.HorizontalAlignment = HorizontalAlignment.Stretch;
            _button.Content = (_stackPanel1);

            StackPanel _stackPanel2 = new StackPanel();
            _stackPanel2.Orientation = Orientation.Vertical;
            _stackPanel2.Margin = new Thickness(20, 0, 0, 0);
            _stackPanel2.Width = 250;
            _stackPanel2.VerticalAlignment = VerticalAlignment.Center;
            _stackPanel2.HorizontalAlignment = HorizontalAlignment.Left;
            _stackPanel1.Children.Add(_stackPanel2);

            TextBlock _textbox1 = new TextBlock();
            _textbox1.Text = $"{_item.Titulo}";
            _textbox1.FontSize = 26;
            _stackPanel2.Children.Add(_textbox1);

            TextBlock _textbox2 = new TextBlock();
            _textbox2.Text = $"{_item.Categoria}";
            _textbox2.FontSize = 14;
            _stackPanel2.Children.Add(_textbox2);
        }
    }
}
```

```

TextBlock _textbox3 = new TextBlock();
if (_item.Tipo == "Recebimento")
{
    _textbox3.Text = $" + R$ {_item.Valor}";
}
else
{
    _textbox3.Text = $" - R$ {_item.Valor}";
}
_textbox3.FontSize = 30;
_textbox3.VerticalAlignment = VerticalAlignment.Center;
_textbox3.HorizontalAlignment = HorizontalAlignment.Center;
_textbox3.Width = 300;
_textbox3.Margin = new Thickness(0, 0, 0, 0);
_stackPanel1.Children.Add(_textbox3);

TextBlock _textbox4 = new TextBlock();
_textbox4.Text = $"{_item.Descricao}";
_textbox4.Width = 300;
_textbox4.FontSize = 14;
_textbox4.TextWrapping = TextWrapping.Wrap;
_textbox4.VerticalAlignment = VerticalAlignment.Center;
_textbox4.HorizontalAlignment = HorizontalAlignment.Right;
_textbox4.Margin = new Thickness(40, 0, 0, 0);
_stackPanel1.Children.Add(_textbox4);
}
User.ListaItens.Reverse();
}

```

Figura 21 - Código para atualização em tempo real das transações

3. CRONOGRAMA COMPLETO

SEMANA 1:

- Identificação das funcionalidades básicas do aplicativo
- Criação e montagem do ambiente de desenvolvimento: Ferramentas e plataformas

SEMANA 2

- Construção do design do aplicativo utilizando o figma
- Construção da arquitetura do projeto utilizando os serviços da Azure

SEMANA 3

- Desenvolvimento das funcionalidades definidas em (1)
- Montagem do ambiente de testes - Pipelines e automação

SEMANA 4

- Desenvolvimento do front-end básico
- Automação de deploys

SEMANA 5

- Implementação dos requisitos não funcionais

4. REQUISITOS FUNCIONAIS

- Controle de gastos total
- Controle de gastos por categoria
- Controle de recebimentos
- Estabelecimento de limite

5. REQUISITOS NÃO FUNCIONAIS

- Visualização de gráficos e tabelas
- Permissão de edição mais detalhada, como ícones, cores e etc
- Criação e remoção de categorias
-

6. SEMANA 1

O que deu certo

A partir das discussões entre a equipe, conseguiu-se montar um cronograma viável com os propósitos do projeto e com o tempo disponível para a realização do mesmo. Além disso, foi possível cumprir com as atividades descritas para a semana.

A identificação das funcionalidades exigiu bastante comunicação, e a equipe conseguiu propor requisitos que estão além daquelas propostas de antemão.

Quanto à criação e montagem do ambiente de desenvolvimento, a equipe montou um projeto utilizando o Azure DevOps. A partir disso, conseguiu-se transferir toda a proposta, cronograma e repositórios para uma única plataforma.

O que deu errado

Nessa semana, acredita-se que nada deu errado, tendo em vista que a equipe conseguiu se organizar bem e cumprir com todas as atividades da semana.

Aprendizado

Nessa primeira semana, notamos a dificuldade e importância de organizar um projeto do zero. Isso exigiu bastante comunicação e compreensão entre os membros da equipe, visto que cada integrante vinha com visões diferentes a respeito do aplicativo. Sendo assim, aprendemos que é crucial unificar e padronizar todas as ideias, a fim de direcionar todos os projetistas a um mesmo propósito.

Auto-avaliação

O processo de conversação para decidir o cronograma do projeto, os requisitos funcionais e as ferramentas foi bastante produtivo e saudável.

A equipe acredita que as atividades estão seguindo em um ritmo agradável e que se encaixa dentro da disponibilidade de cada um. Todos os integrantes estão motivados a se dedicar além do horário proposto para as aulas e estão com grandes expectativas.

Sendo assim, essa primeira semana foi extremamente proveitosa e sentimos que conseguimos adotar os conhecimentos adquiridos durante o curso, em especial, nas disciplinas de “Metodologias Ágeis” e “Introdução à DevOps”, visto que ambas as disciplinas recorrem à cultura de organização da equipe de desenvolvimento.

7. SEMANA 2

Para a Semana 2, foram definidas as seguintes atividades:

- Construção do design do aplicativo utilizando o figma
- Construção da arquitetura do projeto utilizando os serviços da Azure

O que deu certo

Antes de realizar a construção da interface em si, foi feito um diagrama do aplicativo para solidificar o fluxo das funcionalidades do aplicativo. Nesse primeiro momento, a construção foi feita considerando tanto as funcionalidades essenciais e não essenciais, que serão adicionadas conforme o cronograma previsto.

Posteriormente, foi projetado o design do aplicativo utilizando a ferramenta “Figma”. Utilizamos o site <https://colorhunt.co/> para definir as cores tema da interface

Por fim, foi realizada a construção da interface em si. Para isso, a equipe se reuniu para montar um modelo base a ser usado por todas as telas. Além disso, definiu-se alguns padrões como: a fonte a ser utilizada, tamanho das letras, margens e etc. Em seguida, dividiu-se as telas entre os integrantes para otimizar o processo.

A equipe não conseguiu terminar as atividades durante o horário de aula. Nesse sentido, foi necessária disciplina dos membros para a continuação do trabalho semanal em suas residências.

O que deu errado

Infelizmente, não foi possível determinar a arquitetura do projeto utilizando os serviços da Azure, visto que não foi prevista a grande carga de trabalho necessária para a construção da interface.

Aprendizado

A semana 2 exigiu muito do grupo. Isso porque nenhum dos integrantes possuía conhecimento prévio da ferramenta “Figma”. Nesse sentido, a familiarização com todos os recursos da ferramenta demandou mais tempo do que o previsto. Além disso, nenhum dos membros possuía conhecimento de design, o que tornou necessária uma pesquisa para se obter referências.

Tendo isso em vista, acredita-se que o aprendizado foi proveitoso, já que os integrantes foram capacitados para a construção de interfaces simples no “Figma” e adquiriram noções básicas de design.

Auto-avaliação

O processo de pesquisa e de familiarização com o “Figma” foi extremamente dinâmico e a equipe aprendeu muito com isso. Ademais, os resultados finais foram muito satisfatórios. No entanto, não foi possível terminar todas as atividades contidas no cronograma devido à ingenuidade do grupo no momento de construir o cronograma.

8. SEMANA 3

O que deu certo

O grupo conseguiu entender tudo sobre o armazenamento no banco de dados SQLite, que será utilizado para armazenar as informações dos usuários em um arquivo local.

A partir desses estudos, conseguiu-se montar uma base para o sistema de conexão com o banco de dados. Essas conexões são necessárias para o funcionamento de diversas funcionalidades do aplicativo, tais como a autenticação do usuário, cálculo da carteira e armazenamento dos itens relativos aos gastos e recebimentos.

Além disso, o grupo conseguiu modelar a estrutura do código e montar as principais classes a serem utilizadas.

O que deu errado

Novamente, o grupo não conseguiu concluir com o cronograma planejado, devido à grande complexidade das tarefas executadas. Nesse sentido, ainda faltam muitas implementações a serem escritas.

Aprendizado

O maior aprendizado da semana 3 foi em relação ao SQLite, já que nenhum dos integrantes do grupo tinha conhecimento sobre essa linguagem declarativa para banco de dados.

Além disso, o grupo conseguiu compreender as bases de estruturação de um código, a importância dos “Design Patterns” e as principais dificuldades enfrentadas ao se construir um código do zero.

Auto-avaliação

No geral, o grupo está desempenhando bem suas funções, ainda que esteja atrasado em relação ao cronograma previamente estabelecido.

9. SEMANA 4

O que deu certo

Nessa semana começou-se a escrever o código do front-end. A tela inicial de login foi feita e os botões para acessar a conta e registrar já estão funcionais, com as devidas conexões ao banco de dados.

O que deu errado

Infelizmente ainda não foi concluída a etapa de construção do front-end e nem a automação de deploys.

Aprendizado

O aprendizado dessa semana se deu à montagem do front-end junto com as associações às funcionalidades do aplicativo. Felizmente conseguiu-se compreender bem sobre o uso da linguagem de programação e o andamento deve ser mais rápido agora, uma vez que os conceitos estão bem fixados

Auto-avaliação

O grupo está atrasado em relação ao cronograma. No entanto, o aprendizado está sendo muito satisfatório, o que irá permitir que o desenvolvimento do aplicativo ande de forma mais fluida ao longo das próximas semanas.

10. SEMANA 5

O que deu certo

A tela de login e de cadastro foram finalizadas, com o tratamento de possíveis erros decorrentes de falhas de preenchimento. Além disso, a tela inicial foi feita, que exibe o saldo, o limite de gastos do usuário e suas últimas transações.

As funcionalidades de ajustar limite e saldo e de adição de item estão em funcionamento, comunicando com o banco de dados.

O que deu errado

O grupo não conseguiu concluir as tarefas planejadas, devido à falta de tempo, pouco domínio das ferramentas e má organização.

Dentre as tarefas que não foram realizadas estão a remoção e edição dos itens, criação e remoção de categorias e funcionamento das outras abas.

Aprendizado

A equipe aprendeu bastante sobre o XAML, SQL e a integração de ambos feita com C# e também o quanto a organização e divisão de tarefas contribuem para o resultado final de um projeto

Auto-avaliação

Essa semana foi a mais produtiva do grupo, conseguindo realizar diversas atividades necessárias para o funcionamento do projeto. Em relação ao projeto como um todo, a experiência agregou muito para os membros desenvolverem trabalho em equipe e perceberem que a constância e organização são fatores importantes para a boa execução de um projeto.

Além disso, a equipe desenvolveu um conhecimento prático relacionada a todas as ferramentas utilizadas durante o projeto, sendo elas o Azure DevOps, C#, XAML, SQL Server, Git e Figma.

Link do repositório Azure: https://dev.azure.com/HectorShin/CTEDS-CashIn/_git/Cashin