Barcoin

Titolo del progetto: Barcoin
Alunni: Nadir Barlozzo
Classe: I4AC
Anno scolastico: 2018/2019
Docente responsabile: Ugo Bernasconi

1 Introduzione

1.1 Informazioni sul progetto

Docente Responsabile: Ugo Bernasconi
Scuola: Scuola d'Arti Mestieri Trevano
Sezione: Informatica
Materia: Modulo 306
Data inizio progetto: 04.09.2018
Data consegna progetto: 12.12.2018

1.2 Abstract

As the demand for micro-credits financial management systems, in poor or financially unstable countries, increases every day, new software solutions have to rise in order to prevent frauds and secure these transactions. Barcoin aims to be a budget solution for this problem in a rather small environment.

1.3 Scopo

Lo scopo del progetto consiste nel creare un applicativo per la gestione di un sistema micro-finanziario, con la possibilità di visualizzare tutte le linee di credito attive, le relative transazioni in dettaglio con i calcoli degli interessi ed infine la bilancia del prestito con l'appropriato grafico a linee.
Barcoin supporta inoltra l'esportazione dei dati immagazzinati in diverse forme, permettendo così il facile cambiamento di piattaforma.

Analisi

1.4 Analisi del dominio

Ad oggi esistono diversi software che permettono il micro-management di finanze, vengono usati specialmente nei paesi più poveri, dove le persone "*non riescono ad ottenere credito e altri servizi dalle istituzioni finanziarie tradizionali per due ragioni: vengono reputati non solvibili (unbanked) e/o i costi legati all'offerta di questi servizi sono eccessivi e rendono l'operazione non conveniente economicamente*." **Wikipedia**

Molti di questi applicativi risultano però proprietari e quindi non aperti al dominio pubblico, quelli che invece sono resi al pubblico risultano confusionari o poco ergonomici per l'utilizzo di tutti i giorni.
Barcoin cerca di sorpassare questo standard offrendo le funzioni basilari con estrema facilità.

1.5 Analisi dei costi e benefici

Categoria	**Costo**
Personale	210 ore * 1 persona * 60 CHF/h = 12600 CHF
TOT	**12600 CHF**

1.6 Analisi e specifica dei requisiti

ID: REQ-01	
Nome	Gestione anteprima delle linee di credito
Priorità	1
Versione	1.0
Note	
Sotto requisiti	
001	Deve esserci una griglia contenente ogni linea di credito separata
002	Deve esserci un'etichetta per il nome del creditore di tipo testo
003	Deve esserci un'etichetta per il cognome del creditore di tipo testo
004	Deve esserci un'etichetta per la descrizione della linea di credito di tipo testo

ID: REQ-02	
Nome	Gestione in dettaglio delle linee di credito
Priorità	1
Versione	1.0
Note	
Sotto requisiti	
001	Ogni linea di credito in anteprima deve avere una versione in dettaglio da visualizzare
002	Deve esserci un'etichetta per il tasso d'interesse di default di tipo testo
003	Deve esserci un'etichetta per il formato del tasso giornaliero di tipo testo
004	Deve esserci una tabella per la visualizzazione di ogni singola transazione di tipo **BarcoinTable**
005	Deve esserci un grafico per la visualizzazione del bilancio di tipo **BarcoinGraph**

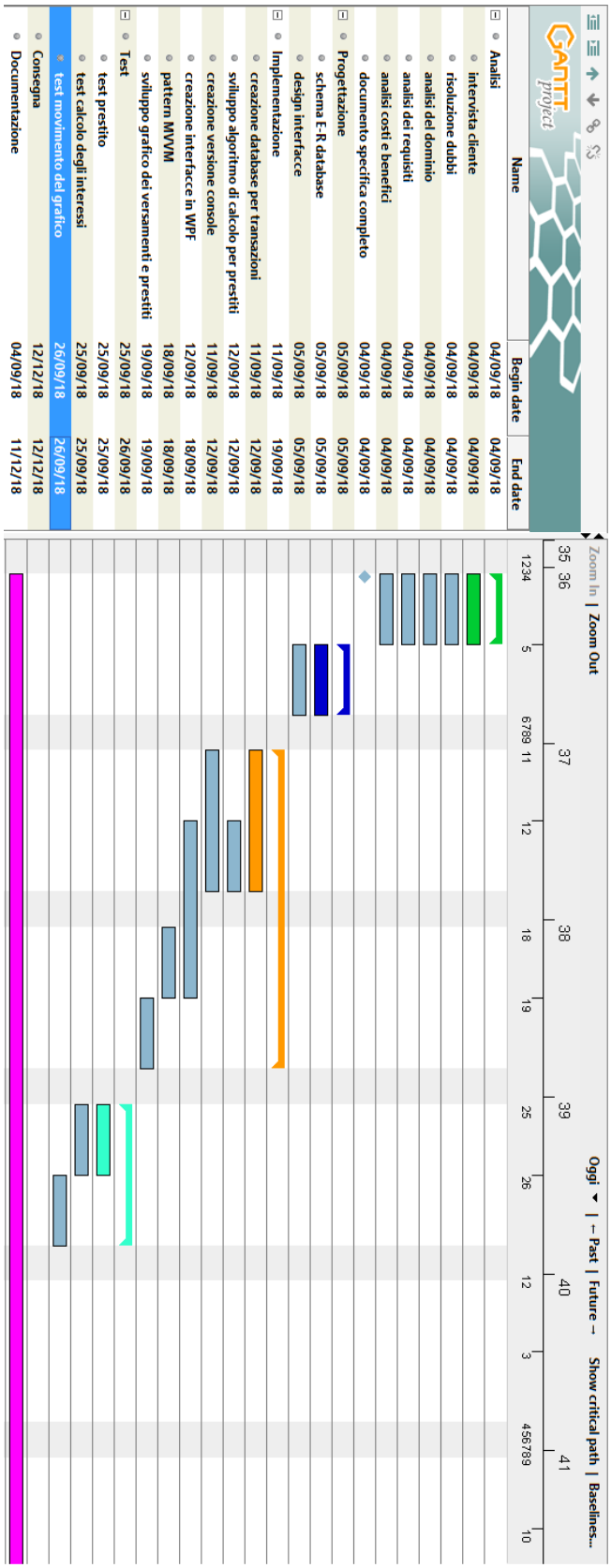
ID: REQ-03	
Nome	Gestione BarcoinTable
Priorità	1
Versione	1.2
Note	
Sotto requisiti	
001	Ogni transazione deve essere univoca e su righe diverse
002	Deve esserci un'etichetta per la data di prelievo/versamento di tipo testo
003	Deve esserci un'etichetta per il tasso di interesse effettivo di tipo testo
004	Deve esserci un'etichetta per l'importo di prelievo/versamento di tipo testo
005	Deve esserci un'etichetta per gli interessi maturati di tipo testo
006	Deve esserci un'etichetta per gli interessi pagati di tipo testo
007	Deve esserci un'etichetta per il bilancio degli interessi di tipo testo
008	Deve esserci un'etichetta per il principale pagato di tipo testo
009	Deve esserci un'etichetta per il bilancio del principale di tipo testo
010	Deve esserci un'etichetta per il totale in valuta dovuto di tipo testo

ID: REQ-04	
Nome	Gestione inserimento creditori
Priorità	2
Versione	1.3
Note	
Sotto requisiti	
001	Tramite un bottone della pagina iniziale deve essere possibile raggiungere l'interfaccia per nuovi creditori
002	Deve esserci un campo per il nome del nuovo creditore di tipo testo
003	Deve esserci un campo per il cognome del nuovo creditore di tipo testo
004	Deve esserci un campo per la descrizione della nuova linea di credito di tipo testo
005	Deve esserci un campo per il tasso giornaliero di tipo numero intero
006	Deve esserci un campo per il tasso d'interesse di default di tipo numero decimale

ID: REQ-05	
Nome	Gestione inserimento transazioni
Priorità	2
Versione	1.1
Note	
Sotto requisiti	
001	All'interno dell'interfaccia in dettaglio deve esserci un form per nuove transazioni
002	Deve esserci un campo per la data della nuova transazione di tipo G/M/A
003	Deve esserci un campo per il tasso d'interesse della nuova transazione di tipo numero decimale
004	Deve esserci un campo per il valore prelevato della nuova transazione di tipo numero interno.

ID: REQ-06	
Nome	Gestione esportazioni dati
Priorità	2
Versione	1.0
Note	
Sotto requisiti	
001	Devono esserci due key binding, ognuno dei quali per effettuare l'esportazione dei dati presenti all'interno della BarcoinTable in formato CSV o PDF. Questi devono chiaramente essere utilizzabili solamente quando ci si trova in un'interfaccia di dettaglio.
002	Premendo il tasto 'e' i dati devono essere esportati in formato CSV
003	Premendo il tasto 'p' i dati devono essere esportati in formato PDF

1.7 Pianificazione



1.7.1 Software

Microsoft Word 2013
Gantt Project 2.8.5
PowerPoint 2013
Google Chrome 64.0
TortoiseGit 2.5.0
DbSchema 6.0.3
Visual Studio 2017 Community
DevExpress 18.1.5
DB Browser for SQLite 3.10.1

1.7.2 Hardware

Laptop PC – Apple Macbook Air

2 Progettazione

2.1 Design dei dati e database

Il database è relativamente semplice, composto da 2 tabelle (**creditline** e **transaction**).

- La tabella **creditline** contiene alcuni dei dati anagrafici del creditore come nome e cognome (**creditorName** e **creditorSurname**), il motivo dell'apertura di tale credito (**description**) ed infine il valore percentuale del tasso d'interesse di base (**defaultInterestRate**) insieme al formato di tasso giornaliero (**diemRate**).
- La tabella **transaction** contiene i dati relativi alle transazioni dei creditori, la data di prelievo/versamento (**date**) con l'importo (**principal**), il tasso di interesse effettivo (**interestRate**) e la descrizione relativa (**description**).

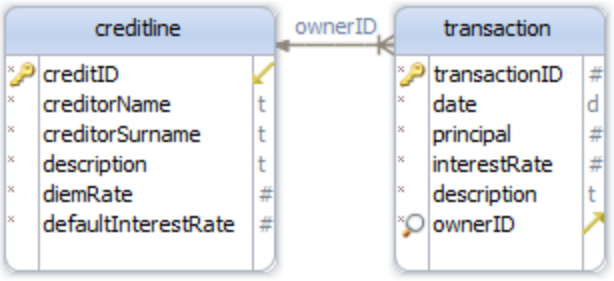


Figura 1 Schema database creato con DbSchema

2.2 Design delle interfacce

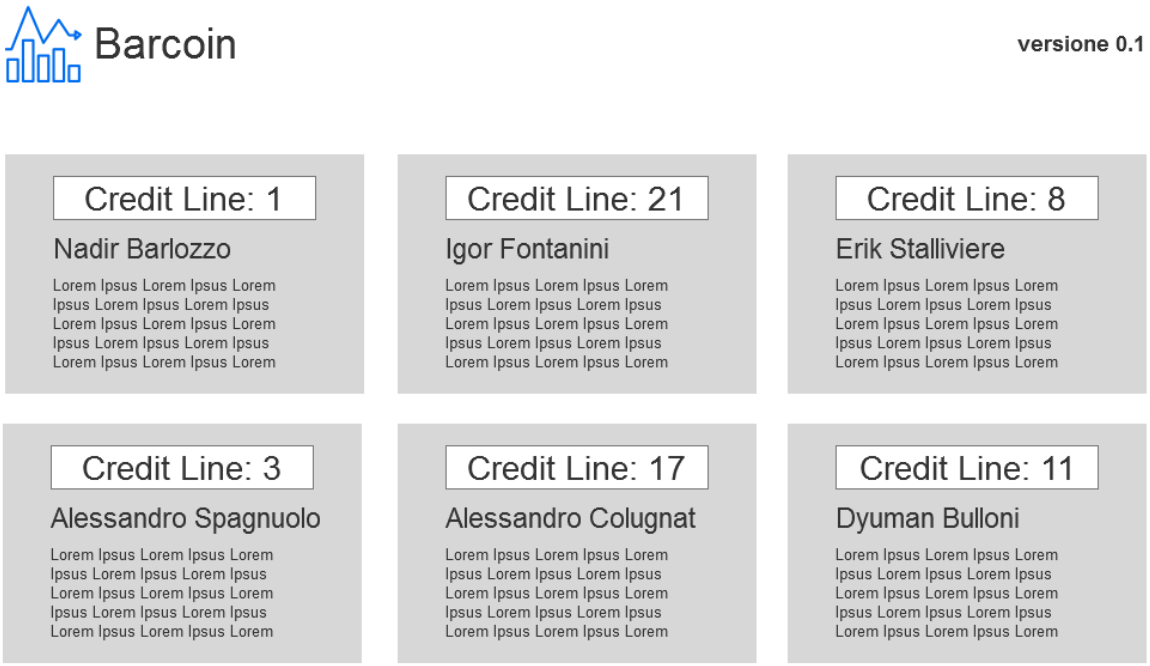


Figura 2 Interfaccia in anteprima

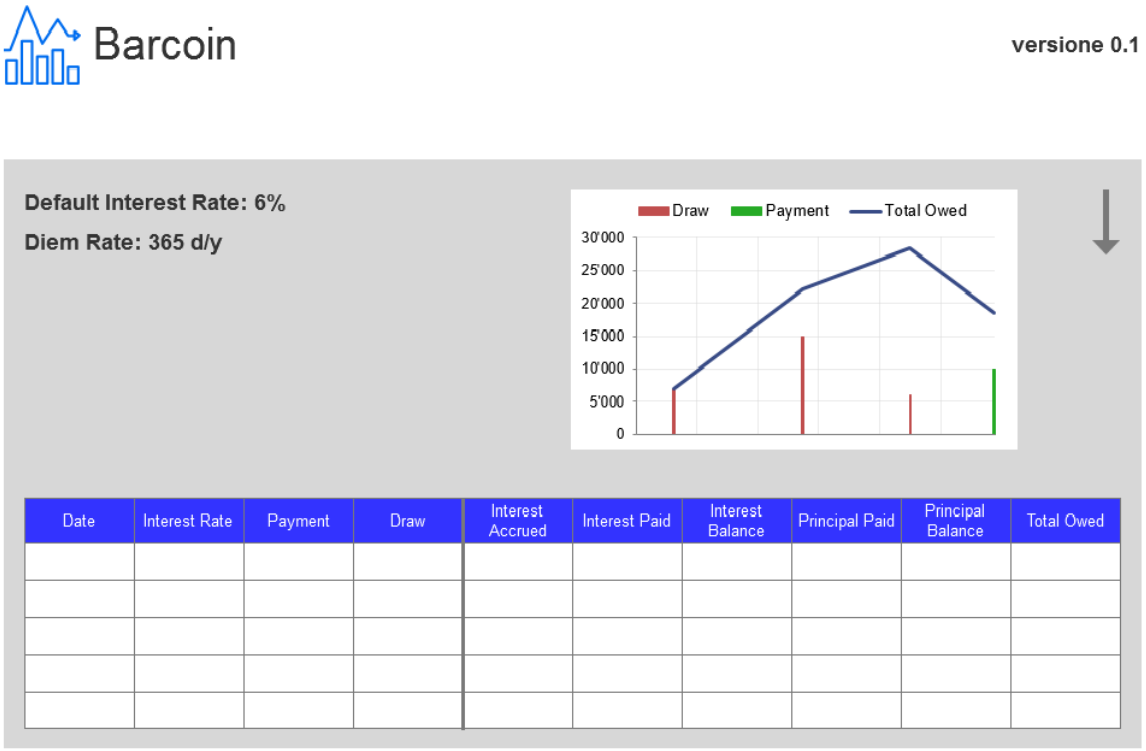


Figura 3 Interfaccia in dettaglio

3 Implementazione

3.1 Grafica / XAML

L'implementazione grafica è una parte consistente del progetto, considerando la relazione al modello MVVM nel mio progetto con i relativi binding è stato necessario ragionare in modo accurato l'ordine di lavoro per ottenere delle interfacce leggere e funzionanti, seguendo comunque le linee guida tracciate nei mockup.

L'applicazione è composta da 3 interfacce principali ognuna delle quali serve un utilità diversa, andandole ad elencare:

- **MainView:** Interfaccia principale, contiene il titolo del progetto, versione corrente, un immagine con il comando per riportarsi allo stato inziale ed infine un controllore di contenuto per le altre due interfacce. All'avvio la DashboardView sarà selezionata come attiva.

```
<DataTemplate DataType="{x:Type viewmodel:DashboardViewModel}">
    <local:DashboardView></local:DashboardView>
</DataTemplate>
<DataTemplate DataType="{x:Type viewmodel:DetailViewModel}">
    <local:DetailView></local:DetailView>
</DataTemplate>
```

Figura 4 Binding alle altre due interfacce

```
<Style TargetType="{x:Type ContentControl}">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type ContentControl}">
                <ContentPresenter Margin="{TemplateBinding Padding}"/>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

Figura 5 Stile per ottenere un margine sul controllore di contenuto

```
<Image x:Key="LogoLink" Source="{Binding Logo}"/>
```

Figura 6 Binding per il logo di Barcoin

```
<ContentControl Grid.Row="1" Margin="20"
                Content="{Binding Path=CurrentViewModel}"/>
```

Figura 7 Binding per visualizzare l'interfaccia attiva

```
<Button Grid.Column="2" Content="{StaticResource AddLink}"
        ToolTip="Add a new credit line" Background="Transparent"
        BorderThickness="0" Width="64" Height="64"
        Command="{Binding Path=AddCommand}" Margin="60,30,60,18">
</Button>
```

Figura 8 Bottone con binding per raggiungere AddCreditorView

- **DashboardView:** Interfaccia per visualizzare le anteprime delle linee di credito contenute nel database, viene creato un bottone personalizzato per ognuna di queste e assegnato il comando per raggiungere la propria interfaccia DetailView.

```
<DataTemplate x:Key="CreditorTemplate">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="*"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <Button ToolTip="Detail" Background="CadetBlue"
                Command="{Binding Path=DataContext.DetailCommand,
                RelativeSource={RelativeSource AncestorType={x:Type ListBox}}}" CommandParameter="{Binding}">
            <Grid Height="200" Width="300">
                <Grid.RowDefinitions>
                    <RowDefinition Height="0.3*"></RowDefinition>
                    <RowDefinition Height="0.3*"></RowDefinition>
                    <RowDefinition Height="0.7*"></RowDefinition>
                </Grid.RowDefinitions>
                <TextBlock VerticalAlignment="Center" HorizontalAlignment="Center" Foreground="White"
                           FontSize="30" Text="{Binding Path=CreditLineID}"></TextBlock>
                <TextBlock Grid.Row="1" VerticalAlignment="Center" Foreground="White" Padding="20"
                           FontSize="20" Text="{Binding Path=CreditorName}"></TextBlock>
                <TextBlock Grid.Row="2" Padding="20" Foreground="White" TextWrapping="Wrap"
                           FontSize="15" Text="{Binding Path=Description}"></TextBlock>
            </Grid>
        </Button>
    </Grid>
</DataTemplate>
```

Figura 9 Template applicato ad ogni bottone della Dashboard

```
<ListBox Background="Transparent" BorderThickness="0" ItemTemplate="{StaticResource CreditorTemplate}"
         ItemsSource="{Binding Path=Creditors}">
    <ListBox.ItemsPanel>
        <ItemsPanelTemplate>
            <WrapPanel Width="1000">

            </WrapPanel>
        </ItemsPanelTemplate>
    </ListBox.ItemsPanel>
</ListBox>
```

Figura 10 Lista resa a ‘blocco’ dove vengono inseriti i bottoni

- **DetailView:** Interfaccia che implementa la maggior parte del progetto a livello grafico, oltre che alle informazioni secondarie, mostra infatti tutte le transazioni eseguite dal creditore scelto in una tabella filtrabile e ha al suo interno un grafico quale riassume il bilancio attuale.

```
<DataGrid ItemsSource="{Binding Path=BalanceData, Mode=TwoWay}" Grid.Row="3" Grid.ColumnSpan="2"
          AutoGenerateColumns="False" Background="Transparent" FontSize="20" FontFamily="Montserrat"
          IsReadOnly="True" HorizontalAlignment="Left" VerticalAlignment="Center">
    <DataGrid.Columns>
        <DataGridTextColumn Header="Date"
            Binding="{Binding Path=Transaction.RawDate, StringFormat=yyyy/MM/dd}" Width="*"/>
        <DataGridTextColumn Header="Principal"
            Binding="{Binding Path=Transaction.Principal, StringFormat={}{0} CHF}" Width="*"/>
        <DataGridTextColumn Header="Interest Rate"
            Binding="{Binding Path=Transaction.InterestRate, StringFormat={}{0}%}" Width="*"/>
        <DataGridTextColumn Header="Principal Balance"
            Binding="{Binding Path=PrincipalBalance, StringFormat={}{0} CHF}" Width="*"/>
        <DataGridTextColumn Header="Accrued Interest"
            Binding="{Binding Path=AccruedInterest, StringFormat={}{0} CHF}" Width="*"/>
        <DataGridTextColumn Header="Interest Balance"
            Binding="{Binding Path=InterestBalance, StringFormat={}{0} CHF}" Width="*"/>
        <DataGridTextColumn Header="Principal Paid"
            Binding="{Binding Path=InterestPaid, StringFormat={}{0} CHF}" Width="*"/>
        <DataGridTextColumn Header="Interest Paid"
            Binding="{Binding Path=PrincipalPaid, StringFormat={}{0} CHF}" Width="*"/>
        <DataGridTextColumn Header="Total Owed"
            Binding="{Binding Path=TotalOwed, StringFormat={}{0} CHF}" Width="*"/>
    </DataGrid.Columns>
</DataGrid>
```

Figura 11 Griglia contenente le transazioni del creditore

```
<lvc:CartesianChart Grid.Column="1" Grid.RowSpan="3" Series="{Binding Path=GraphData}"
                    LegendLocation="Right">
    <lvc:CartesianChart.AxisY>
        <lvc:Axis Title="Balance"></lvc:Axis>
    </lvc:CartesianChart.AxisY>
    <lvc:CartesianChart.AxisX>
        <lvc:Axis Title="Date" Labels="{Binding Dates}"></lvc:Axis>
    </lvc:CartesianChart.AxisX>
</lvc:CartesianChart>
```

Figura 12 Grafico del bilancio fatto con Live Charts

- **AddCreditorView:** Interfaccia per aggiungere nuovi creditori all'interno del sistema di Barcoin.

```
<TextBox Grid.Row="1" Style="{StaticResource PlaceHolder}"
         Tag="Creditor Name e.g Nadir" Height="20" Width="250"
         HorizontalAlignment="Left" Text="{Binding Path=Name, Mode=TwoWay}">
</TextBox>
```

Figura 13 I TextBox all'interno di questa interfaccia utilizzano tutti lo stile PlaceHolder, per rendere l'inserimento semplificato

```
<Style x:Key="PlaceHolder" TargetType="{x:Type TextBox}" BasedOn="{StaticResource {x:Type TextBox}}">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type TextBox}">
                <Grid>
                    <TextBox Text="{Binding Path=Text,
                        RelativeSource={RelativeSource TemplatedParent},
                        Mode=TwoWay,
                        UpdateSourceTrigger=PropertyChanged}"
                        x:Name="textSource"
                        Background="Transparent"
                        Panel.ZIndex="2" />
                    <TextBox Text="{TemplateBinding Tag}" Background="{TemplateBinding Background}" Panel.ZIndex="1">
                        <TextBox.Style>
                            <Style TargetType="{x:Type TextBox}">
                                <Setter Property="Foreground" Value="Transparent"/>
                                <Style.Triggers>
                                    <DataTrigger Binding="{Binding Path=Text, Source={x:Reference textSource}}" Value="">
                                        <Setter Property="Foreground" Value="LightGray"/>
                                    </DataTrigger>
                                </Style.Triggers>
                            </Style>
                        </TextBox.Style>
                    </TextBox>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

Figura 14 Stile PlaceHolder applicato globalmente nel file App.xaml per ottenere l'effetto di testo fittizio

- **AboutView**

```
<StackPanel Margin="20">
    <TextBlock TextWrapping="WrapWithOverflow" FontSize="20"
        FontFamily="Montserrat" Text="{Binding Path=Description}"
        Margin="0,0,150,0"/>
    <Button Command="{Binding Path=VideoCommand}" Margin="0,20,0,0"
        Width="150" HorizontalAlignment="Left">
        About microfinance
    </Button>
</StackPanel>
```

Figura 15 La descrizione viene bindata dal ViewModel per lasciare lo XAML più pulito

3.2 Database / MySQL

Impostate le interfacce dell'applicazione bisogna creare il database per il salvataggio dei dati relativi ai creditori.
Seguendo lo schema *barcoinschema.png* risulta molto facile la trascrittura in codice SQL per tabelle e relazioni, il file *barcoin.sql* se eseguito creerà una copia identica del database usato in sviluppo locale.

3.3 Logica / C#

Preparata la banca dati MySQL ed avendo completato la prima fase manca ora la parte finale che metta in connessione tutti i moduli, adottando il modello MVVM (Model, View, ViewModel) ed applicando alcune altre convenzioni si ottengono i seguenti namespace:

Helper: Namespace contenente le risorse per i servizi.

MVVM: Namespace contenente tutti i file alla base della struttura MVVM.

MySqlDatabaseHelper: Classe per l'aiuto alla connessione della banca dati MySQL ed esecuzione delle relative query.

```
public MySqlDatabaseHelper()
{
    string connectionString = "datasource=localhost;port=3306;username=root;password=;database=barcoin;SslMode=none";

    databaseConnection = new MySqlConnection(connectionString);

    commandController = new MySqlCommand("", databaseConnection)
    {
        CommandTimeout = 10
    };
}
```

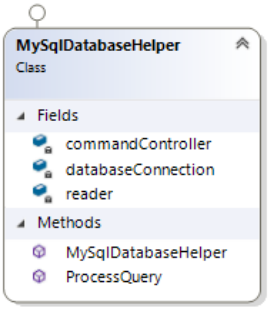
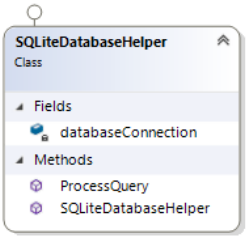


Figura 16 Costruttore della classe per stabilire una connessione con il database.

```
public static List<string[]> ProcessQuery()
{
    List<string[]> result = new List<string[]>();

    try
    {
        databaseConnection.Open();

        reader = commandController.ExecuteReader();

        if (reader.HasRows)
        {
            while (reader.Read())
            {
                string[] row = { reader.GetString(0), reader.GetString(1), reader.GetString(2), reader.GetString(3),
                    reader.GetString(4), reader.GetString(5) };
                result.Add(row);
            }
        }
        else
        {
            MessageBox.Show("No rows found.");
        }

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

    return result;
}
```

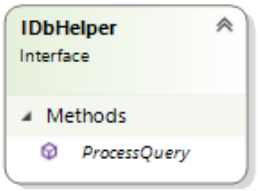
Figura 17 Metodo per eseguire la query corrente e ritornarne il risultato

SQLiteDatabaseHelper: Copia logica di MySqlDatabaseHelper ma mirata alla connessione con una banca dati SQLite.



IDbHelper: Interfaccia implementata da entrambe le classi di aiuto.

```
interface IDbHelper
{
    List<string[]> ProcessQuery(string query);
}
```



Service: Namespace contenente i servizi dell'applicazione.

CreditorDataRepository: Classe per l'esecuzione di interrogazioni alla banca dati relative ai creditori, ritorna le informazioni in forma di oggetti.

```
public Creditor Get(int id)
{
    string query = "SELECT * FROM creditline WHERE creditID = " + id;

    List<string[]> result = helper.ProcessQuery(query);

    Creditor c = new Creditor()
    {
        CreditLineID = int.Parse(result[0][0]),
        CreditorName = result[0][1],
        CreditorSurname = result[0][2],
        Description = result[0][3],
        DiemRate = int.Parse(result[0][4]),
        DefaultInterestRate = double.Parse(result[0][5])
    };

    return c;
}
```

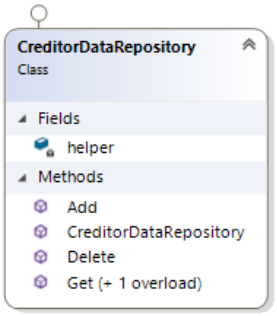


Figura 18 Metodo per ritornare le informazioni di un singolo creditore

```
public List<Creditor> Get()
{
    string query = "SELECT * FROM creditline";

    List<string[]> result = helper.ProcessQuery(query);
    List<Creditor> creditors = new List<Creditor>();

    for (int i = 0; i < result.Count; i++)
    {
        Creditor c = new Creditor()
        {
            CreditLineID = int.Parse(result[i][0]),
            CreditorName = result[i][1],
            CreditorSurname = result[i][2],
            Description = result[i][3],
            DiemRate = int.Parse(result[i][4]),
            DefaultInterestRate = double.Parse(result[i][5])
        };

        creditors.Add(c);
    }

    return creditors;
}
```

Figura 19 Metodo per ritornare le informazioni di tutti i creditori

```
public void Add(Creditor item)
{
    int lastId = 0;
    string query = "";

    if (Settings.Mode == Modality.Local)
    {
        query = $"SELECT * FROM `creditline` ORDER BY" +
            $" `creditline`.`creditID` DESC LIMIT 1";

        lastId = int.Parse(helper.ProcessQuery(query)[0][0]);

        query = $"INSERT INTO `creditline` (`creditID`," +
            $"`creditorName`, `creditorSurname`, `description`, `diemRate`, `defaultInterestRate`) VALUES" +
            $" ('{lastId+1}', '{item.CreditorName}', '{item.CreditorSurname}', '{item.Description}', '{item.DiemRate}'," +
            $" '{item.DefaultInterestRate}')";
    }
    else
    {
        query = $"INSERT INTO `creditline` (" +
            $"`creditorName`, `creditorSurname`, `description`, `diemRate`, `defaultInterestRate`) VALUES" +
            $" ('{item.CreditorName}', '{item.CreditorSurname}', '{item.Description}', '{item.DiemRate}'," +
            $" '{item.DefaultInterestRate}')";
    }


    List<string[]> result = helper.ProcessQuery(query);
}
```

Figura 20 Metodo per inserire un nuovo creditore, nel caso che l'utente stia utilizzando una banca dati SQLite andremo ad aggiungere l'id utente manualmente.

TransactionDataRepository: Classe per l'esecuzione di interrogazioni alla banca dati relative alle transazioni, ritorna le informazioni in forma di oggetti.

```
public List<Transaction> Get(int id)
{
    string query = $"SELECT * FROM `transaction` WHERE `transaction`.`ownerID` = '{id}' ORDER BY" +
        $" `transaction`.`date` ASC";

    List<string[]> result = helper.ProcessQuery(query);
    List<Transaction> transactions = new List<Transaction>();

    Debug.WriteLine(result[0][0]);

    for (int i = 0; i < result.Count; i++)
    {
        Transaction t = new Transaction()
        {
            TransactionID = int.Parse(result[i][0]),
            RawDate = DateTime.Parse(result[i][1]),
            Principal = int.Parse(result[i][2]),
            InterestRate = double.Parse(result[i][3]),
            Description = result[i][4]
        };

        transactions.Add(t);
    }

    return transactions;
}
```

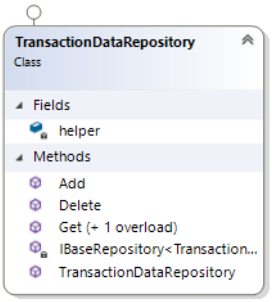


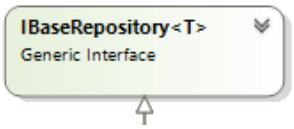
Figura 21 Metodo per ottenere tutte le transazioni di un creditore

```
public void Add(Transaction item)
{
    int lastId = 0;
    string query = "";

    if (Settings.Mode == Modality.Local)
    {
        query = $"SELECT * FROM `transaction` ORDER BY" +
            $" `transaction`.`transactionID` DESC LIMIT 1";

        lastId = int.Parse(helper.ProcessQuery(query)[0][0]);

        query = $"INSERT INTO `transaction` (`transactionID`, `date`, `principal`, `interestRate`, `description`," +
            $" `ownerID`) VALUES('{lastId+1}', '{item.RawDate.ToString("yyyy-MM-dd")}', '{item.Principal}', " +
            $"'{item.InterestRate}', '', '{item.TransactionID}')";
    }
    else
    {
        query = $"INSERT INTO `transaction` (`transactionID`, `date`, `principal`, `interestRate`, `description`," +
            $" `ownerID`) VALUES('{lastId}', '{item.RawDate.ToString("yyyy-MM-dd")}', '{item.Principal}', " +
            $"'{item.InterestRate}', '', '{item.TransactionID}')";
    }

    List<string[]> result = helper.ProcessQuery(query);
}
```

Figura 22 Metodo per aggiungere una transazione ad un creditore, nel caso che l'utente stia utilizzando una banca dati SQLite andremo ad aggiungere l'id della transazione manualmente.

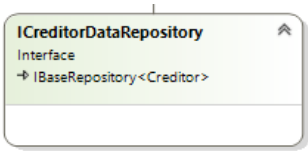
IBaseRepository: Interfaccia base per definire una repository di dati qualunque.

```
interface IBaseRepository<T>
{
    T Get(int id);
    List<T> Get();

    void Add(T item);

    void Delete(T item);
}
```



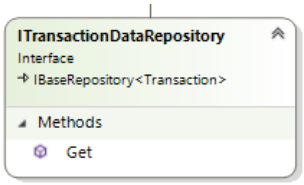
ICreditorDataRepository: Interfaccia specifica da implementare per dati dei creditori.

```
interface ICreditorDataRepository : IBaseRepository<Creditor> { }
```



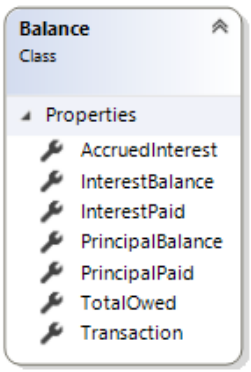
ITransactionDataRepository: Interfaccia specifica da implementare per dati delle transazioni.

```
interface ITransactionDataRepository : IBaseRepository<Transaction>
{
    new List<Transaction> Get(int id);
}
```



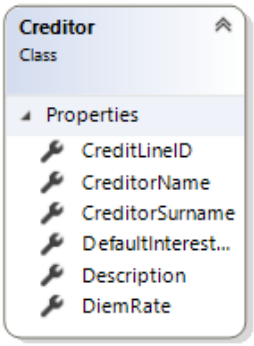
Model: Namespace dei modelli di dati.

Balance: Classe per il modello dati di un bilancio (creditore + transazione).



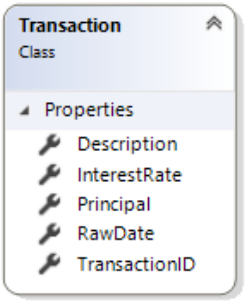

```
public class Balance
{
    public Transaction Transaction { get; set; }

    public double PrincipalBalance { get; set; }

    public double AccruedInterest { get; set; }

    public double InterestBalance { get; set; }

    public double PrincipalPaid { get; set; }

    public double InterestPaid { get; set; }

    public double TotalOwed { get; set; }
}
```

Creditor: Classe per il modello dati di un creditore.




```
public class Creditor
{
    public int CreditLineID { get; set; }

    public string CreditorName { get; set; }

    public string CreditorSurname { get; set; }

    public string Description { get; set; }

    public int DiemRate { get; set; }

    public double DefaultInterestRate { get; set; }
}
```

Transaction: Classe per il modello dati di una transazione.

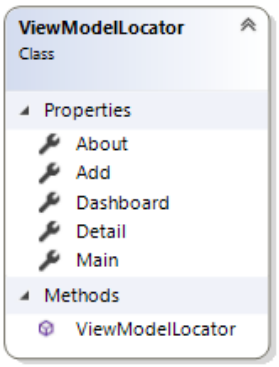



```
public class Transaction
{
    public int TransactionID { get; set; }

    public DateTime RawDate { get; set; }

    public int Principal { get; set; }

    public double InterestRate { get; set; }

    public string Description { get; set; }
}
```

ViewModel: Namespace per i controllori grafico/logici (View <> Model).

ViewModelLocator: Controllore per la gestione del ViewModel corrente.

```
public class ViewModelLocator
{
    public static MainViewModel Main
    {
        get { return ServiceLocator.Current.GetInstance<MainViewModel>(); }
    }

    public static DashboardViewModel Dashboard
    {
        get { return ServiceLocator.Current.GetInstance<DashboardViewModel>(); }
    }

    public static DetailViewModel Detail
    {
        get { return ServiceLocator.Current.GetInstance<DetailViewModel>(); }
    }

    public static AddCreditorViewModel Add
    {
        get { return ServiceLocator.Current.GetInstance<AddCreditorViewModel>(); }
    }

    public static AboutViewModel About
    {
        get { return ServiceLocator.Current.GetInstance<AboutViewModel>(); }
    }

    public ViewModelLocator()
    {
        ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);

        SimpleIoc.Default.Register<MainViewModel>();
        SimpleIoc.Default.Register<DashboardViewModel>();
        SimpleIoc.Default.Register<AddCreditorViewModel>();
        SimpleIoc.Default.Register<AboutViewModel>();
        SimpleIoc.Default.Register<DetailViewModel>(true);
    }
}
```



MainViewModel: Controllore per l'interfaccia MainView.

```
public string Logo
{
    get { return @"\Resource\logo.png"; }
}
```

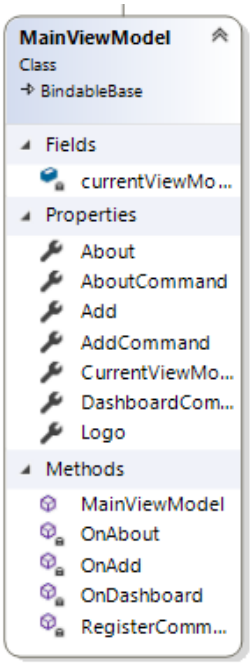
Figura 23 Proprietà per ottenere il link al logo di Barcoin

```
private BindableBase currentViewModelBase;
public BindableBase CurrentViewModel
{
    get { return currentViewModelBase; }
    set
    {
        SetProperty(ref currentViewModelBase, value);
    }
}
```

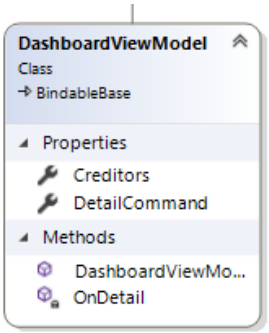
Figura 24 Proprietà per assegnare il controllore attuale a livello grafico

```
public IDelegateCommand DashboardCommand { get; private set; }
```

Figura 25 Comando per impostare il controllore Dashboard



DashboardViewModel: Controllore per l'interfaccia DashboardView.




```
public IDelegateCommand DetailCommand { get; private set; }
```

Figura 26 Comando per impostare il controllore Detail

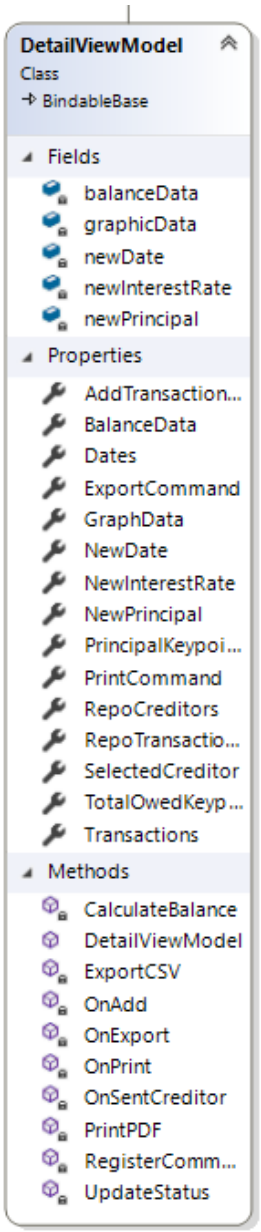
```
CreditorDataRepository repo = new CreditorDataRepository();

Creditors = new ObservableCollection<Creditor>(repo.GetCreditors());
```

Figura 27 Utilizzo del servizio CreditorDataRepository per ottenere tutti i creditori

```
private void OnDetail(object obj)
{
    Messenger.Default.Send((Creditor)obj);
}
```

Figura 28 Metodo per inviare il creditore scelto dall'utente nell'interfaccia grafica al controllore Detail

DetailViewModel: Controllore per l'interfaccia DetailView




```
public List<string> Dates { get; set; }
public List<double> TotalOwedKeypoints { get; set; }
public List<double> PrincipalKeypoints { get; set; }
```

Figura 29 Liste contenenti i dati per il grafico del bilancio

```
public Creditor SelectedCreditor { get; set; }
```

Figura 30 Proprietà per il creditore selezionato

```
Messenger.Default.Register<Creditor>(this, OnSentCreditor);
```

Figura 31 Metodo per ricevere l'oggetto Creditor inviato dal controllore Dashboard

```
Transactions = new ObservableCollection<Transaction>(
    repo.GetTransactionsByID(SelectedCreditor.CreditLineID)
);
```

Figura 32 Utilizzo del servizio CreditorDataRepository per ottenere tutte le transazioni del creditore attuale

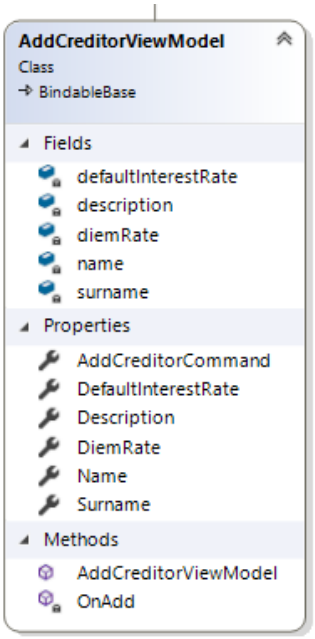
```
GraphData = new SeriesCollection
{
    new LineSeries
    {
        Values = new ChartValues<double>(TotalOwedKeypoints),
        Title = "Total Owed"
    },

    new LineSeries
    {
        Values = new ChartValues<double>(PrincipalKeypoints),
        Title = "Principal"
    }
};
```

Figura 33 Definizione delle linee con i loro valori e titoli, da inserire nel Live Chart

```
private void CalculateBalance()
```

Figura 34 Metodo per i calcoli da eseguire secondo l'algoritmo di credito

AddCreditorViewModel: Controllore per l'aggiunta di nuovi creditori.




```
public IDelegateCommand AddCreditorCommand { get; set; }
```

Figura 35 Comando per richiamare il metodo di aggiunta

```
private void OnAdd(object obj)
{
    CreditorDataRepository repo = new CreditorDataRepository();

    Creditor c = new Creditor()
    {
        CreditorName = Name,
        CreditorSurname = Surname,
        Description = Description,
        DiemRate = int.Parse(DiemRate),
        DefaultInterestRate = double.Parse(DefaultInterestRate)
    };

    repo.AddCreditor(c);

    ViewModelLocator.Main.CurrentViewModel = ViewModelLocator.Dashboard;
}
```

Figura 36 Metodo eseguito quando il comando AddCreditor viene effettuato

3.4 Algoritmo di credito

Fondamento logico principale su cui si basa Barcoin, il calcolo degli interessi e bilanci alla fine di ogni transazione per poter ottenere un sistema corretto e trasparente. Ha necessitato diversi ragionamenti ed analisi di alcuni fogli di calcolo Excel già esistenti, il risultato è il seguente:

B = Bilancio totale
C = Capitale accreditato totale
IA = Bilancio interesse
I = Interesse percentuale di una singola transazione
$\sum I$ = Interessi accumulati delle transazioni precedenti
T = Tasso giornaliero stabilito (360/365)
D_0 = Data accredito relativa
D_1 = Data corrente

$$B = C + IA$$
$$B = C + C * \left(\frac{I}{T}\right) + \sum_{I=0}^{n} I * (D_1 - D_0)$$

Esempio:

$$B = 120CHF + 120CHF * \left(\frac{6.5\%}{360}\right) + 0.34CHF * (23.03.18 - 18.01.18) + 0.12CHF * (23.03.18 - 29.10.17)$$

Per ottenere questo bilancio prendo il capitale totale di 120CHF, gli aggiungo il 6.5% di interesse diviso per il tasso giornaliero stabilito, infine sommo gli interessi calcolati nei bilanci precedenti moltiplicati per l'intervallo di giorni tra la data attuale e la data di accredito relativa.

3.5 Dipendenze

CommonServiceLocation v1.0.0 – Fornisce un livello astratto per l'inserimento di dipendenze logiche.

LiveCharts v0.9.7 – Visualizzazione dei dati semplice, flessibile ed interattiva per .NET.
LiveCharts.WPF v0.9.7 – Estensione di LiveCharts per WPF.

Interactivity.WPF v2.0.20525 – Pacchetto di Microsoft per favorire l'interattività WPF.

Microsoft.Data.SQLite v2.1.0 – Fornisce un DBMS e DB completamente offline in formato di file unico.

PDF Sharp v1.32.3057 – Fornisce una collezione di metodi e funzionalità per creare file in formato PDF.

4 Test

Test Case: **Riferimento**:	TC-01 REQ-01	**Nome:**	Controllo anteprima
Descrizione:	Visualizzare i creditori registrati in versione anteprima.		
Prerequisiti:	La banca dati deve contenere almeno un creditore.		
Procedura:	1. Aprire l'applicazione ed aspettare per lo splash screen di finire		
Risultati attesi:	Una serie di bottoni verranno visualizzati quanti sono i creditori registrati nella banca dati.		

Test Case: **Riferimento**:	TC-02 REQ-02	**Nome:**	Controllo dettaglio
Descrizione:	Visualizzare i creditori registrati in versione dettaglio.		
Prerequisiti:	TC-01.		
Procedura:	1. Premere su uno dei bottoni presenti nell'interfaccia di anteprima		
Risultati attesi:	L'utente viene mandato ad una nuova schermata con due dati testuali, un form per inserire nuove transazioni, una tabella che elenca tutte le transazioni effettuate ed un grafico del bilancio monetario. Ogni sessanta secondi deve essere effettuato un aggiornamento automatico dei dati mostrati.		

Test Case: **Riferimento**:	TC-03 REQ-04	**Nome:**	Controllo inserimento creditore
Descrizione:	Inserimento nella banca dati di un nuovo creditore.		
Prerequisiti:	Nessuno.		
Procedura:	1. Premere il bottone in alto a destra con l'immagine di un "+" 2. Compilare tutti i campi presenti nel form con i dati appartenenti al nuovo creditore. 3. Premere il bottone "Create"		
Risultati attesi:	L'utente viene riportato indietro alla schermata in anteprima con un nuovo bottone per l'utente creato.		

Test Case: TC-04 **Riferimento**: REQ-05		**Nome:**	Controllo inserimento transazione
Descrizione:	Inserimento nella banca dati di una nuova transazione.		
Prerequisiti:	TC-02.		
Procedura:	1. Compilare il form nell'interfaccia di dettaglio con i dati della nuova transazione. 2. Premere su "Add Transaction"		
Risultati attesi:	La tabella delle transazioni e il grafico dei bilanci vengono aggiornati per comprendere il nuovo record, l'utente avrà una nuova riga di tabella ed una coppia di colonne in più.		

Test Case: TC-05 **Riferimento**: REQ-06		**Nome:**	Controllo esportazioni PDF e CSV
Descrizione:	Esportare i vari dati in formato PDF e CSV.		
Prerequisiti:	TC-02.		
Procedura:	1. Premere il tasto "e". 2. Premere il tasto "p".		
Risultati attesi:	Andando nella cartella di compilazione del progetto l'utente troverà le cartelle "ExportedTransactions" e "PDFTransactions", al loro interno ci saranno gli omonimi file esportati con l'identificativo di *NomeCognome*.		

4.1 Risultati test

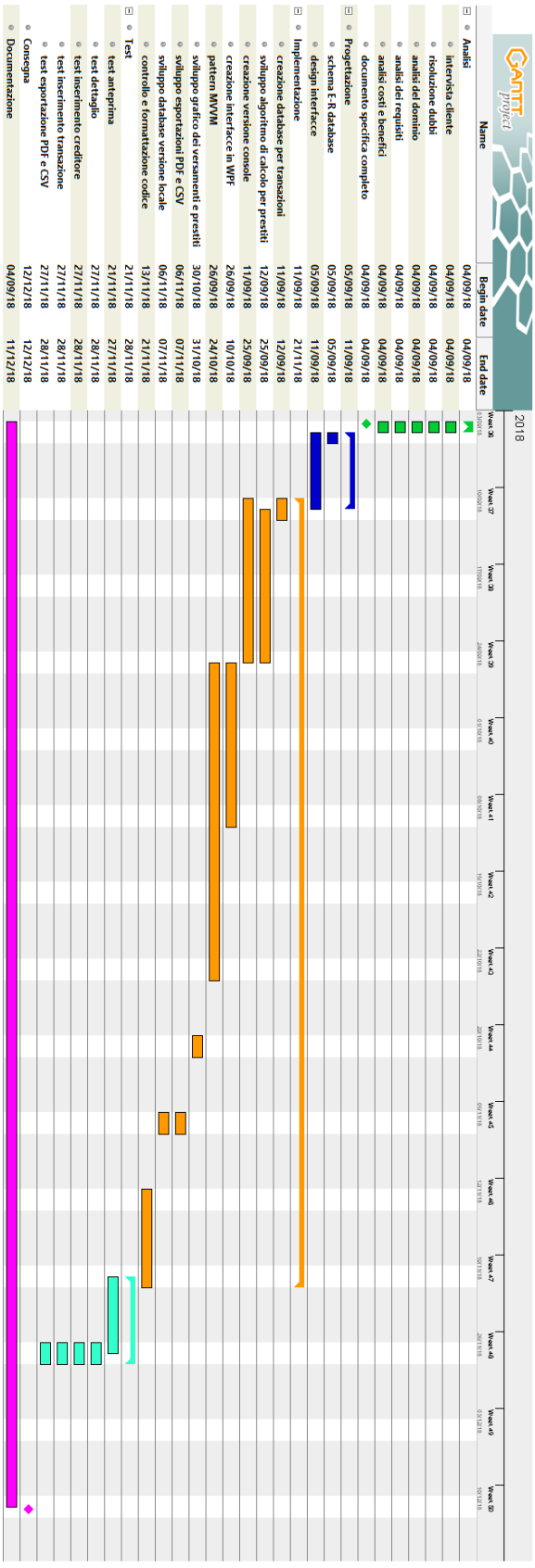
- TC-01 ✓
- TC-02 ✓
- TC-03 ✓
- TC-04 ✓
- TC-05 ✓

4.2 Mancanze/limitazioni conosciute

Come spiegato in modo abbastanza dettagliato all'inizio del documento, Barcoin non si presenta neanche lontanamente come un software per livelli aziendali o in grande volume, presenta infatti gli strumenti e funzionalità limitate per un campione di persone ristretto.

L'utilizzo del linguaggio C# porta inoltre delle limitazioni a livello di software e hardware, a differenza dei suoi simili C e C++ è ad alto livello di astrazione, con una compilazione più pesante e meno performante.

5 Consuntivo



6 Conclusioni

6.1 Sviluppi futuri

Come sviluppi possibili da implementare ci sarebbe in primo punto la versione web, disponibile a tutti aventi una connessione internet e in secondo punto la creazione di una valuta proprietaria per sorpassare i problemi di conversione, inflazione e truffa.

6.2 Considerazioni personali

Ho trovato personalmente il progetto divertente e particolarmente interessante verso le mie conoscenze economiche, con il professore è già stata discussa la possibilità di legare questo progetto con uno degli sviluppi futuri e renderlo basato su una valuta virtuale.
L'obiettivo sarebbe quello di creare una struttura basata su tecnologia blockchain per mantenere e controllare l'autenticità di questa valuta usata in Barcoin.

7 Bibliografia

Le risorse utilizzate come supporto e fondamenta per lo sviluppo di Barcoin, alcune di queste sono propri componenti software mentre altre sono testi, documentazioni o media.

7.1 Sitografia

https://lvcharts.net/, Sito di documentazione LiveCharts WPF
https://docs.microsoft.com/en-us/dotnet/framework/wpf/, Sito di documentazione Microsoft WPF
https://en.wikipedia.org/wiki/Microfinance, Wikipedia sulla micro finanza
https://www.vertex42.com/Calcators/line-of-credit-tracker.html, Base algoritmica di Barcoin
https://dev.mysql.com/doc/, Sito di documentazione MySQL
https://www.freepik.com, Creatore dell'immagine utilizzata come logo
https://www.flaticon.com/authors/eleonor-wang, Creatore dell'immagine rappresentante il punto di domanda
https://www.flaticon.com/authors/dave-gandy, Creatore dell'immagine rappresentante il segno di addizione.

8 Allegati

Allegato A: *I4_Diari_Barcoin.docx*

9 Glossario

Parola	Significato
WPF	Windows Presentation Foundation, modello di Microsoft per la costruzione di applicativi con interfaccia utente basata sul linguaggio XAML.
MVVM	Model View ViewModel, gruppo di regole e standard per definire una struttura dati da seguire negli sviluppi di applicativi.
Micro-Management	Stile di gestione in cui un manager osserva e / o controlla da vicino il lavoro dei suoi dipendenti o clienti.
Diem Rate	Tasso giornaliero prestabilito all'avvio di una linea di credito che indica quanti giorni effettivi vengono considerati dalla banca all'interno di un anno.
Unbanked	Che non ha la possibilità di pagare, non solvibile.
CSV	Comma-separated values, formato di dati separati da un carattere, solitamente dalla virgola.
PDF	Portable Document Format, formato di dati molto simile ad un documento Word ma senza la possibilità di modifica.