



İSPARTA UYGULAMALI BİLİMLER ÜNİVERSİTESİ

TEKNOLOJİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BLG-228 BİLGİSAYAR PROGRAMLAMA II DERSİ

YIL SONU PROJE RAPORU

Dersin Öğretim Üyesi
Dr. Öğr. Üyesi Serdar PAÇACI

Dersin Öğretim Elemanı
Arş. Gör. Rafet GÖZBAŞI

Proje Açıklaması: Cardify – Kişisel Finans Yönetim Uygulaması

Cardify, kullanıcıların tüm banka kartlarını, gelir ve giderlerini, kategori bazlı harcamalarını, varlıklarını kolayca takip edip yönetebilecekleri kullanıcı dostu bir kişisel finans yönetim uygulamasıdır. Projenin temel amacı, finansal verilerin detaylı analizini görsel ve mantıklı bir biçimde sunarak kullanıcıların bütçelerini etkin yönetmelerini sağlamaktır.

Temel Özellikler

1. Çoklu Banka ve Kart Takibi

Kullanıcılar sisteme birden fazla banka kartını kolayca ekleyebilir. Her kart, benzersiz renk kodlarıyla tanımlanarak grafiklerde ve analizlerde görsel ayrımlı sağlanır. Her banka için ayrı renkler kullanılır. Bu sayede kullanıcılar hangi kartla ne kadar harcama veya gelir sağladığını rahatlıkla görebilir.

2. Gelir ve Gider Takibi – Kart Bazlı

Kullanıcılar kart bazında aylık gelir ve gider toplamlarını görebilir. Harcamalar kırmızı, gelirler yeşil renklerle grafiklerde vurgulanır. Örneğin, "Garanti kartımla bu ay 1200 TL harcadım, 400 TL kazandım" gibi özet bilgilere kolayca ulaşılır.

3. Tüm Kartların Toplam Finansal Görünümü

Tüm kartlardan gelen toplam varlık ve harcama bilgileri tek bir özet sayfada sunulur. Kullanıcılar toplam mevcut bakiyelerini ve belirli dönemlerdeki harcama tutarlarını kolayca takip edebilir.

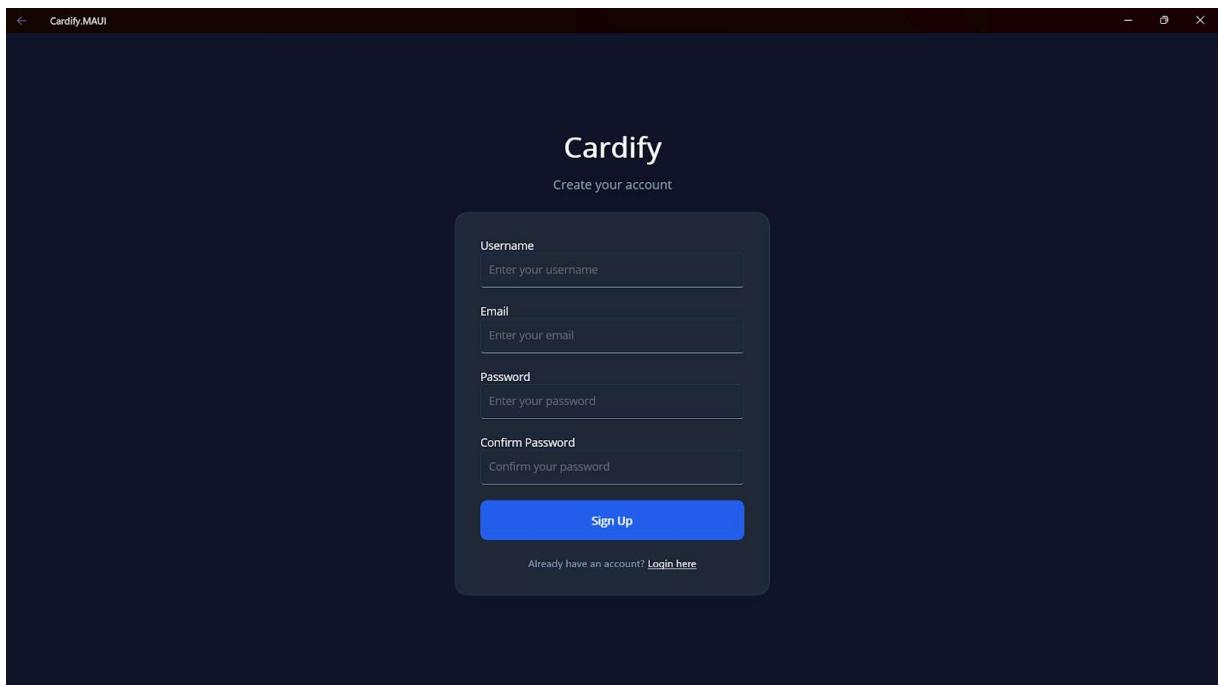
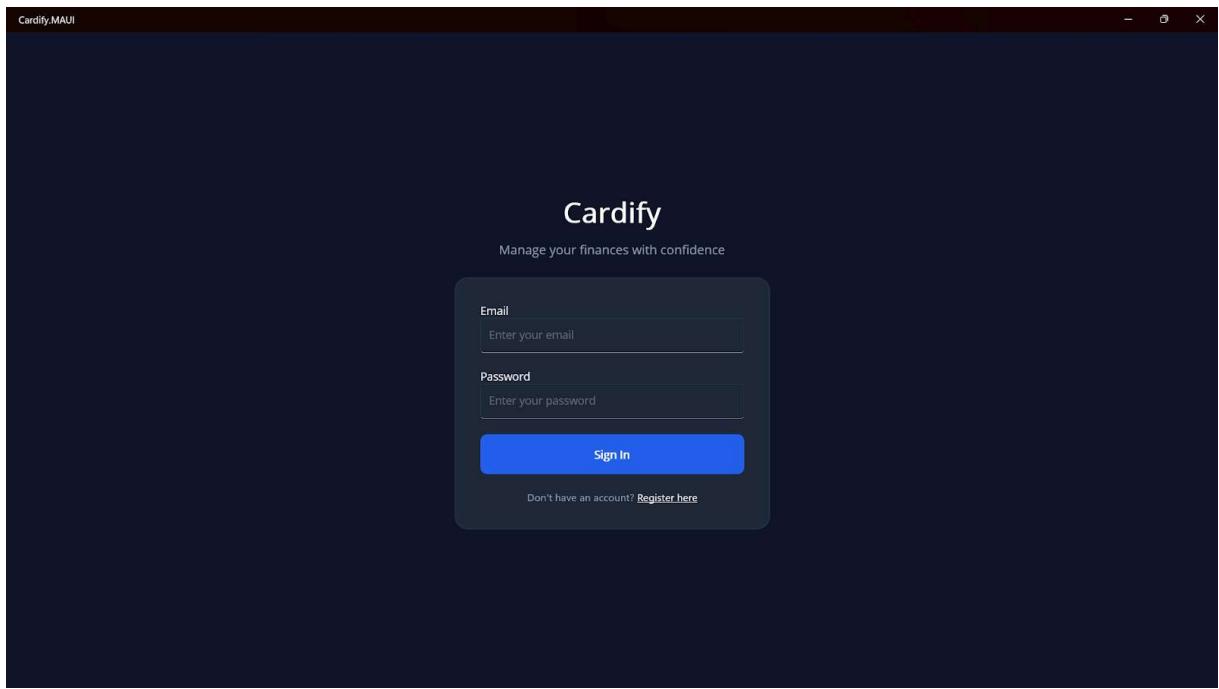
Teknik Detaylar ve Kullanılan Teknolojiler

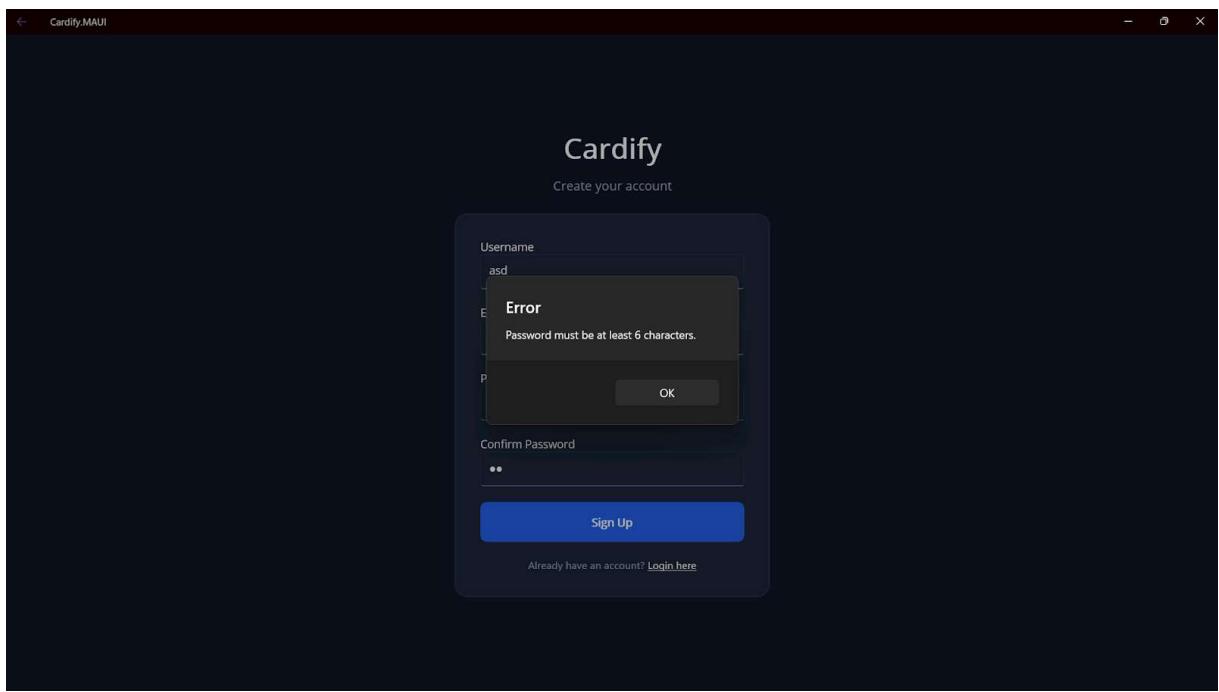
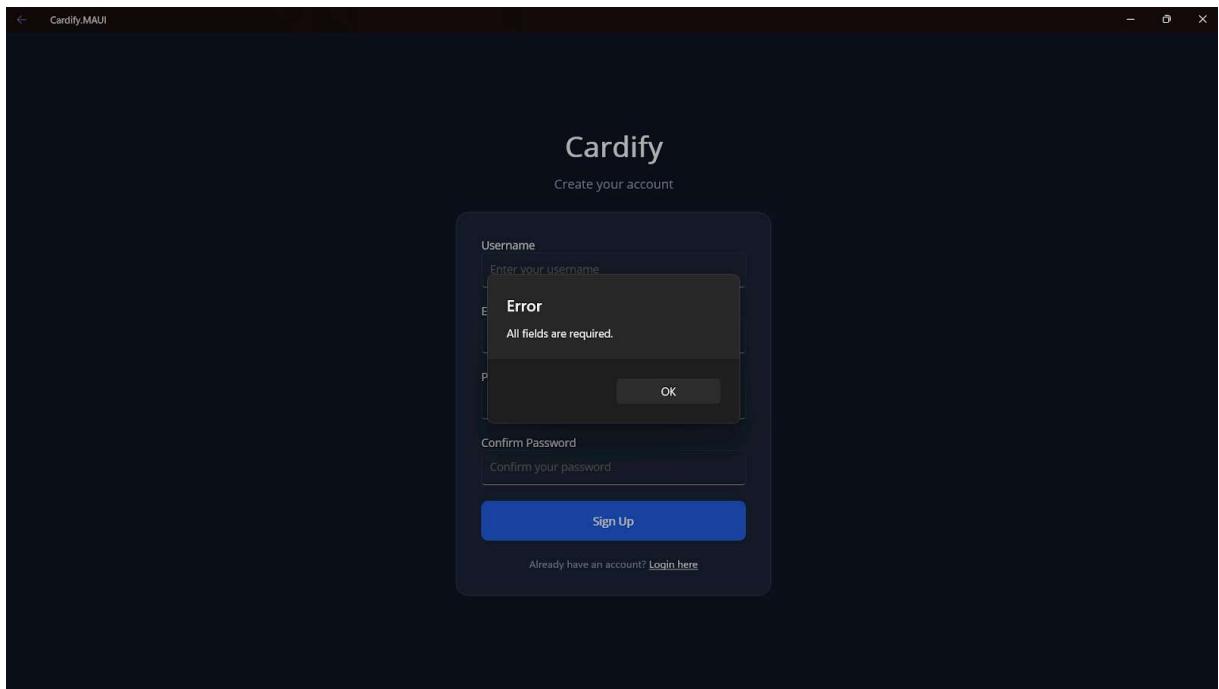
- **MAUI (Multi-platform App UI)** kullanılarak hem mobil hem masaüstü platformlarda çalışabilen çapraz platform uygulama geliştirildi.
- **Minimal API** yapısı ile backend tarafında hafif, hızlı ve esnek bir API tasarlandı.
- **Entity Framework Core Code First** yaklaşımı ile veritabanı modelleri tasarılanıp, migrationlar kullanılarak veritabanı dinamik şekilde oluşturuldu ve yönetildi.
- Veritabanı bağlantısı güvenli ve performanslı şekilde sağlandı.
- **CRUD (Create, Read, Update, Delete)** işlemleri kapsamlı olarak uygulandı; kullanıcı kart, işlem, kategori ve hedef verileri üzerinde tam kontrol sağlandı.
- **Kullanıcı Yönetimi:** Giriş, çıkış, şifre değiştirme gibi işlemler güvenli şekilde yapıldı.
- **UI Bileşenleri:** CollectionView, Picker, DatePicker, RadioButton gibi MAUI kontrolleri kullanılarak zengin ve kullanıcı dostu arayüz oluşturuldu.
- **OOP prensipleri** projenin her katmanında uygulanarak kodun sürdürülebilirliği ve genişletilebilirliği sağlandı.
- **Veri doğrulama** ile kullanıcıdan alınan verilerin doğruluğu ve tutarlılığı sağlandı.
- **LINQ sorguları** ile veritabanı işlemleri optimize edildi ve filtreleme, sıralama gibi işlemler etkin şekilde gerçekleştirildi.

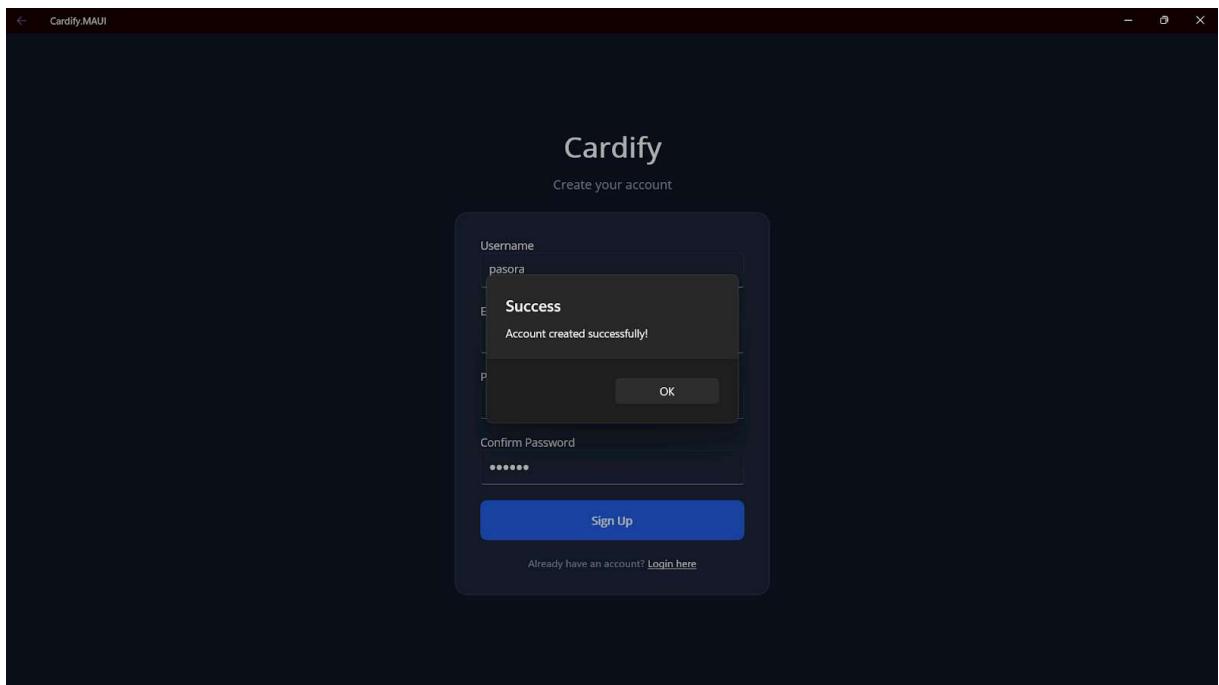
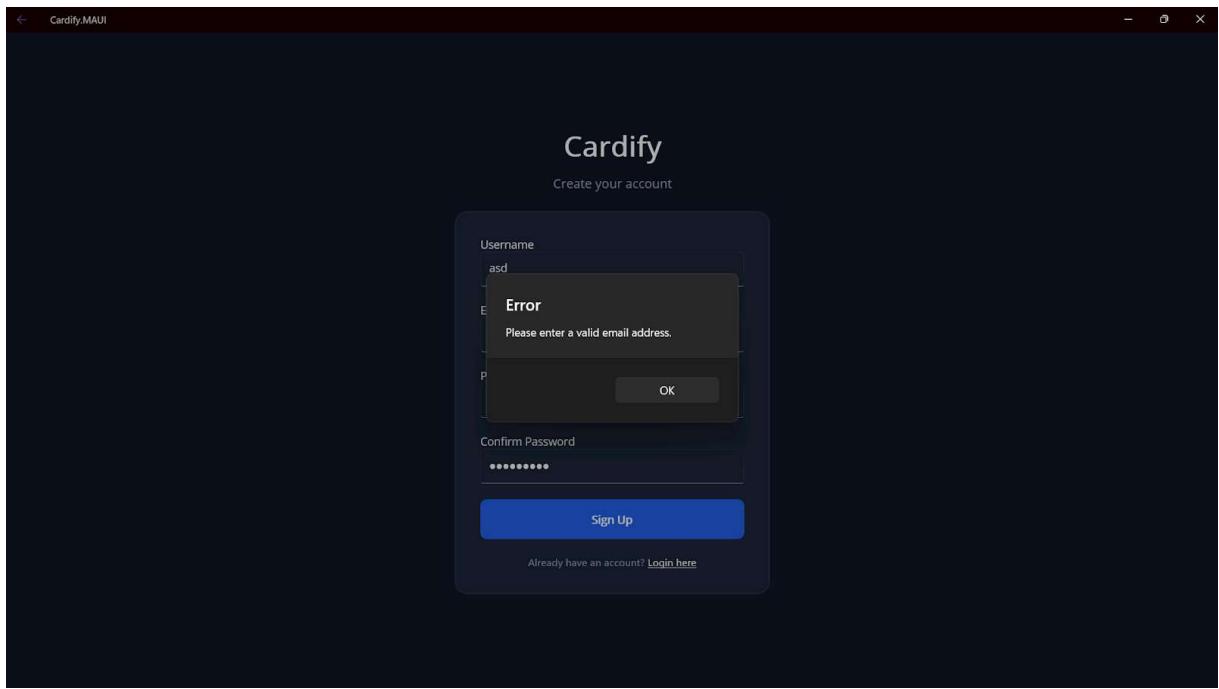
Sonuç olarak, Cardify uygulaması kullanıcılarına:

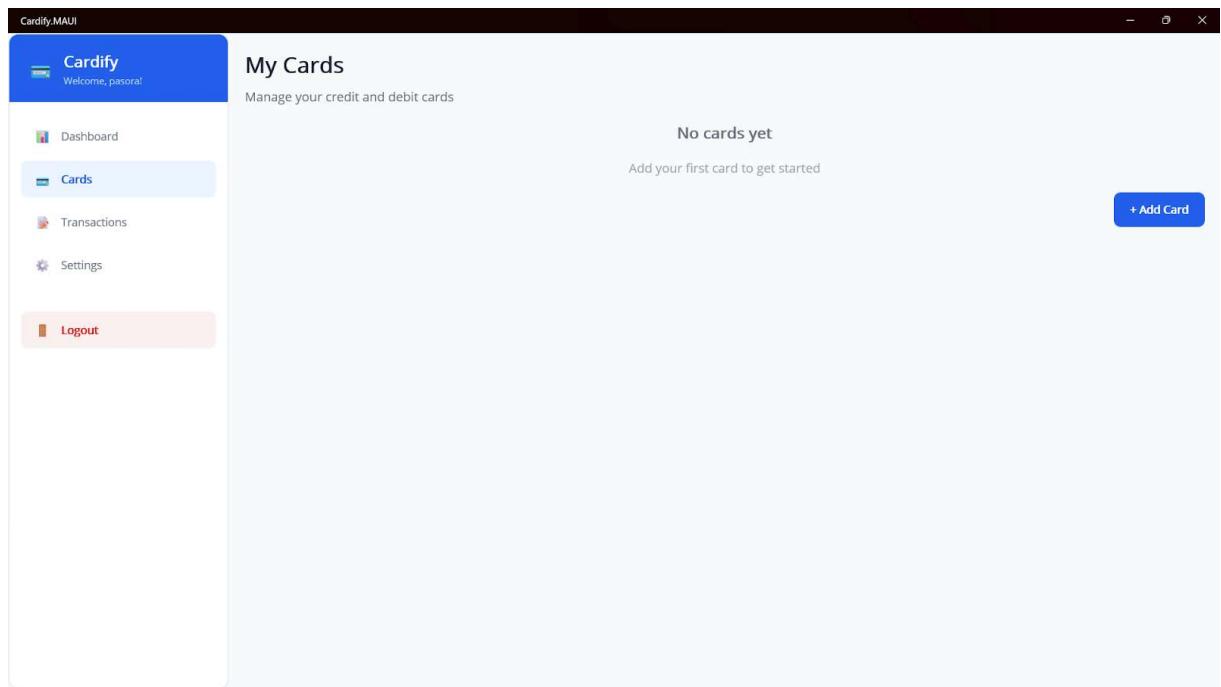
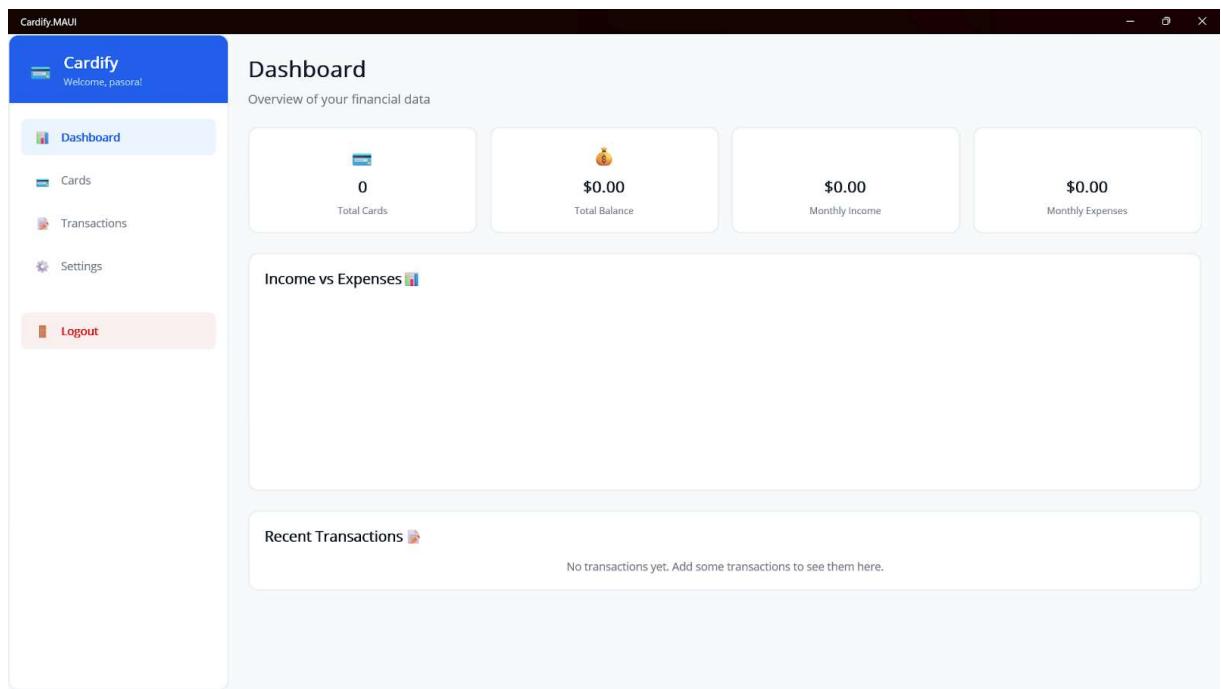
- Finansal durumlarını tek bir platformda şeffaf şekilde takip etme,
- Kart bazlı analiz ve görselleştirme,
- Kategori bazlı harcama yönetimi,
- Hızlı, güvenli ve kullanıcı dostu finansal işlemler yapabilme imkanı sunmaktadır.

Bu proje, gerçek dünya finans yönetimi ihtiyaçlarını karşılamak üzere tasarlanmış ve modern teknolojilerle geliştirilmiş kapsamlı bir kişisel bütçe yönetim sistemidir.

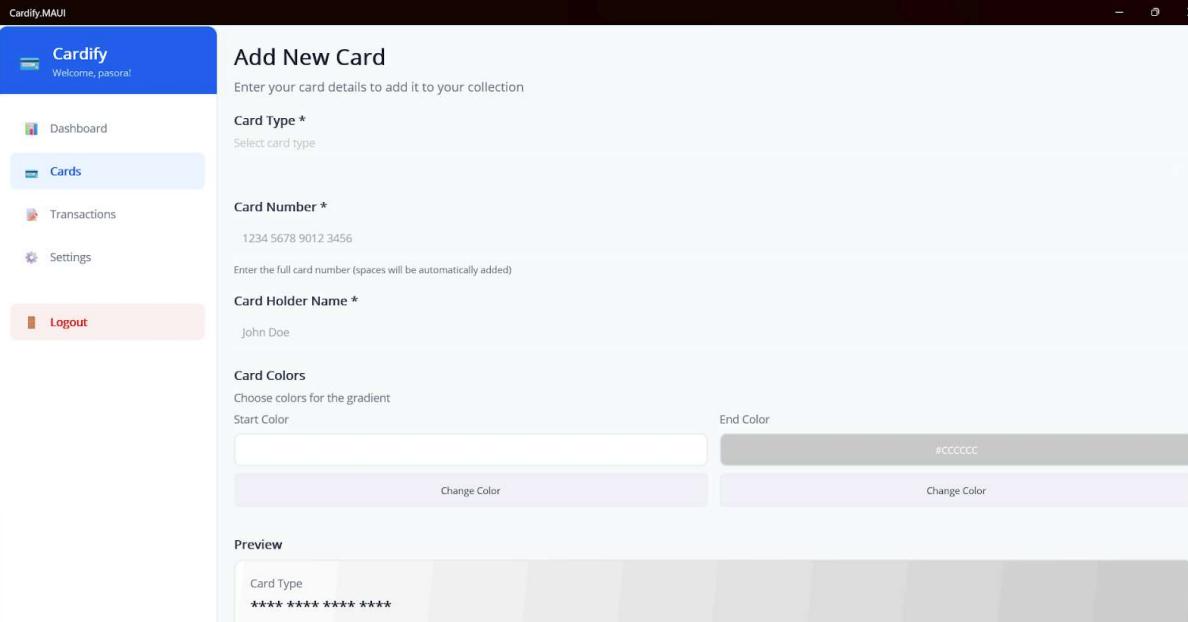








Cardify.MAU



The screenshot shows the Cardify.MAU application interface. On the left is a sidebar with a blue header "Cardify" and a sub-header "Welcome, pasoral". Below the header are five menu items: "Dashboard", "Cards" (which is highlighted in blue), "Transactions", and "Settings", each with an icon. At the bottom of the sidebar is a red button labeled "Logout". The main content area has a white background. At the top, the title "Add New Card" is displayed in large black font. Below it is a subtitle "Enter your card details to add it to your collection". A form starts with a section "Card Type *". It includes a dropdown placeholder "Select card type". Next is a section "Card Number *". It contains a text input field with the value "1234 5678 9012 3456" and a note below it: "Enter the full card number (spaces will be automatically added)". Then is a section "Card Holder Name *". It contains a text input field with the value "John Doe". Following this is a "Card Colors" section with a note "Choose colors for the gradient". It features two color swatches: "Start Color" (white) and "End Color" (gray). Each swatch has a "Change Color" button below it. Finally, there is a "Preview" section showing a small representation of the card with "Card Type" and "*****" for the number, and "Card Holder Name".

Add New Card

Enter your card details to add it to your collection

Card Type *

Select card type

Card Number *

1234 5678 9012 3456

Enter the full card number (spaces will be automatically added)

Card Holder Name *

John Doe

Card Colors

Choose colors for the gradient

Start Color

End Color

#CCCCCC

Change Color

Change Color

Preview

Card Type
***** ***** ***** *****

Card Holder Name

Cardify.MAUI

The screenshot shows the Cardify.MAUI application interface. On the left is a sidebar with icons for Dashboard, Cards (selected), Transactions, and Settings, along with a Logout button. The main content area has a title 'Add New Card' and a subtitle 'Enter your card details to add it to your collection'. It includes fields for 'Card Type *' (Credit Card selected), 'Card Number' (1234 5678 9012 3456), 'Card Holder Name *' (John Doe), 'Card Colors' (gradient from light blue to dark grey), and a preview section showing the card details.

Add New Card

Enter your card details to add it to your collection

Card Type *

Select card type

Credit Card

Debit Card

1234 5678 9012 3456

Enter the full card number (spaces will be automatically added)

Card Holder Name *

John Doe

Card Colors

Choose colors for the gradient

Start Color

End Color

#CCCCCC

Change Color

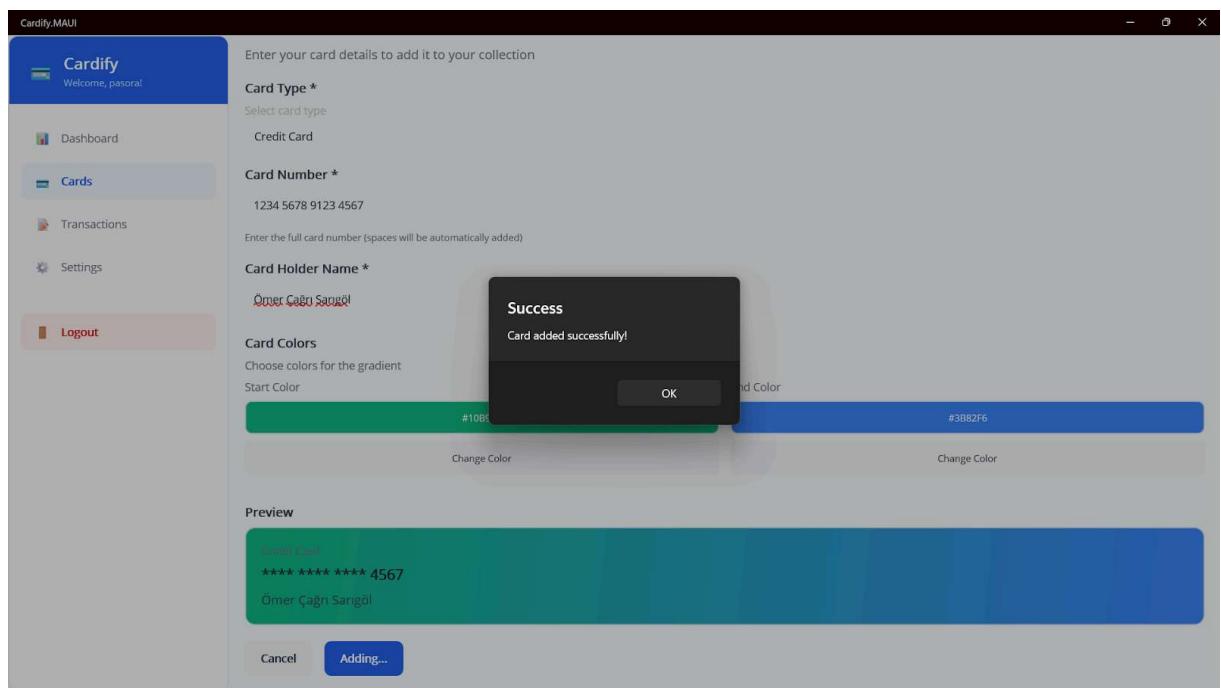
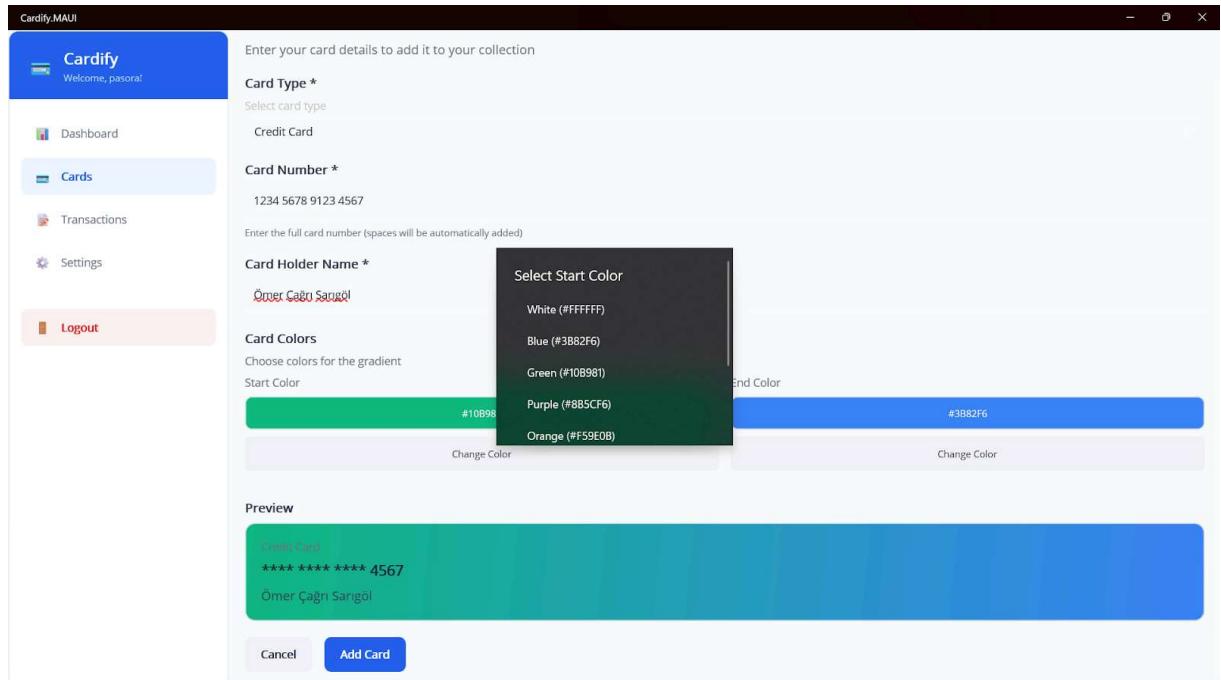
Change Color

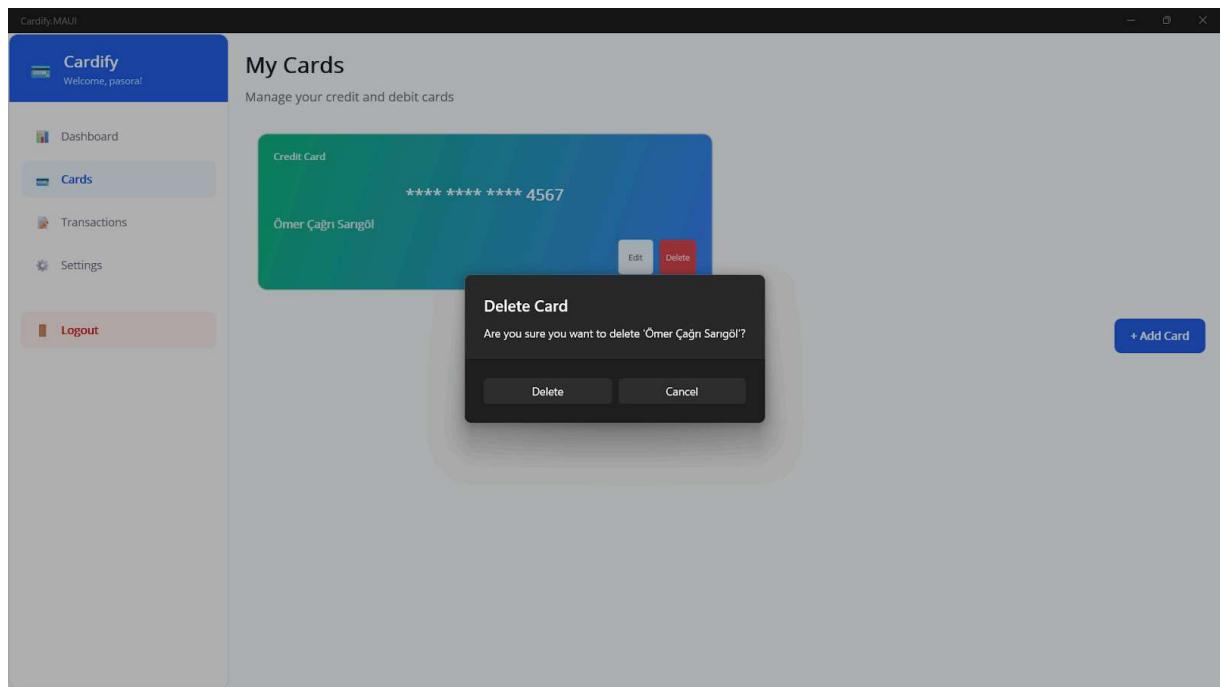
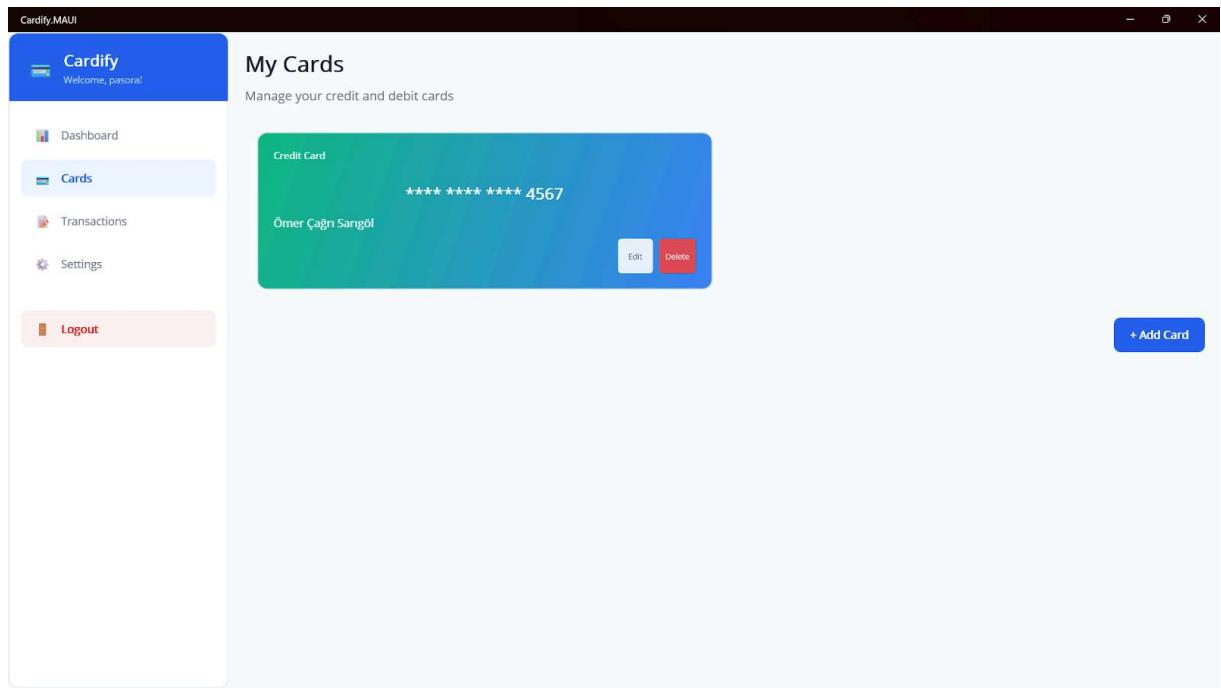
Preview

Credit Card

***** ***** ***** *****

Card Holder Name





Cardify.MAUI

Edit Card

Welcome, pasora!

Update your card details

Card Type *
Select card type
Credit Card

Card Number *
1234 5678 9123 4567
Enter the full card number (spaces will be automatically added)

Card Holder Name *
Ömer Çağrı Sarıgöl

Card Colors
Choose colors for the gradient

Start Color: #10B981 | End Color: #3B82F6

Change Color

Preview

Credit Card
***** * 4567
Ömer Çağrı Sarıgöl

Cardify.MAUI

Add New Transaction

Welcome, pasora!

Enter transaction details to track your finances

Transaction Name *
e.g., Grocery Shopping, Salary, etc.

Amount *
0.00

Transaction Type *
 Income Expense Debt

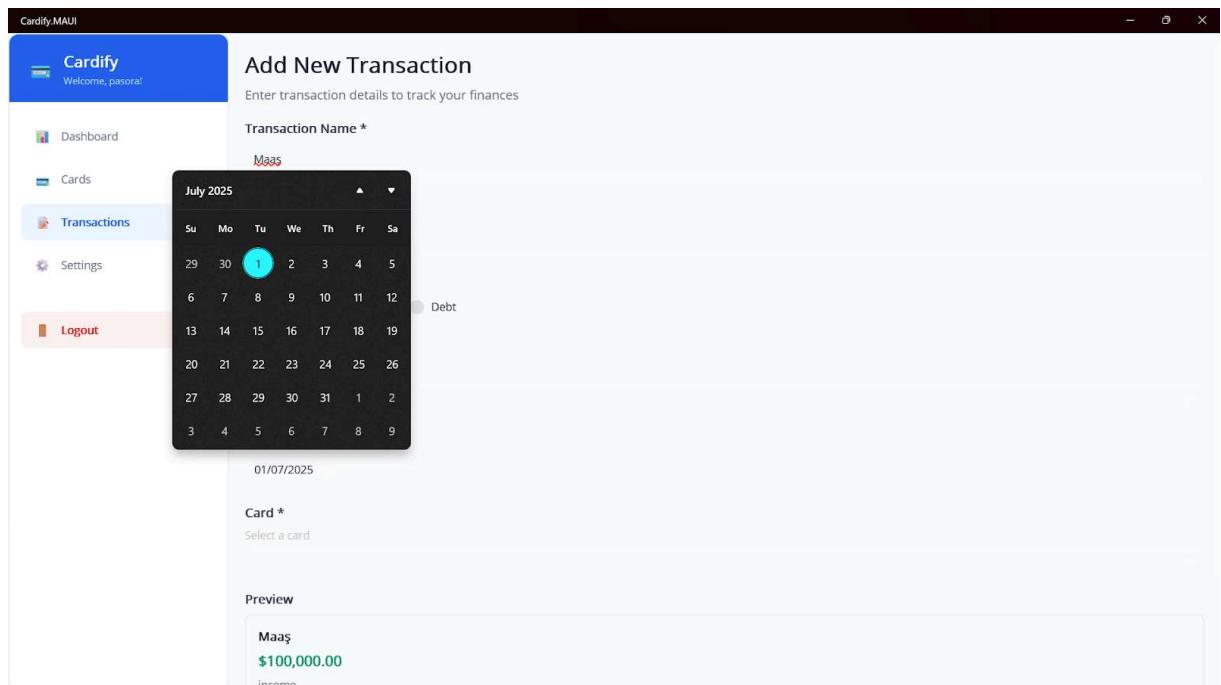
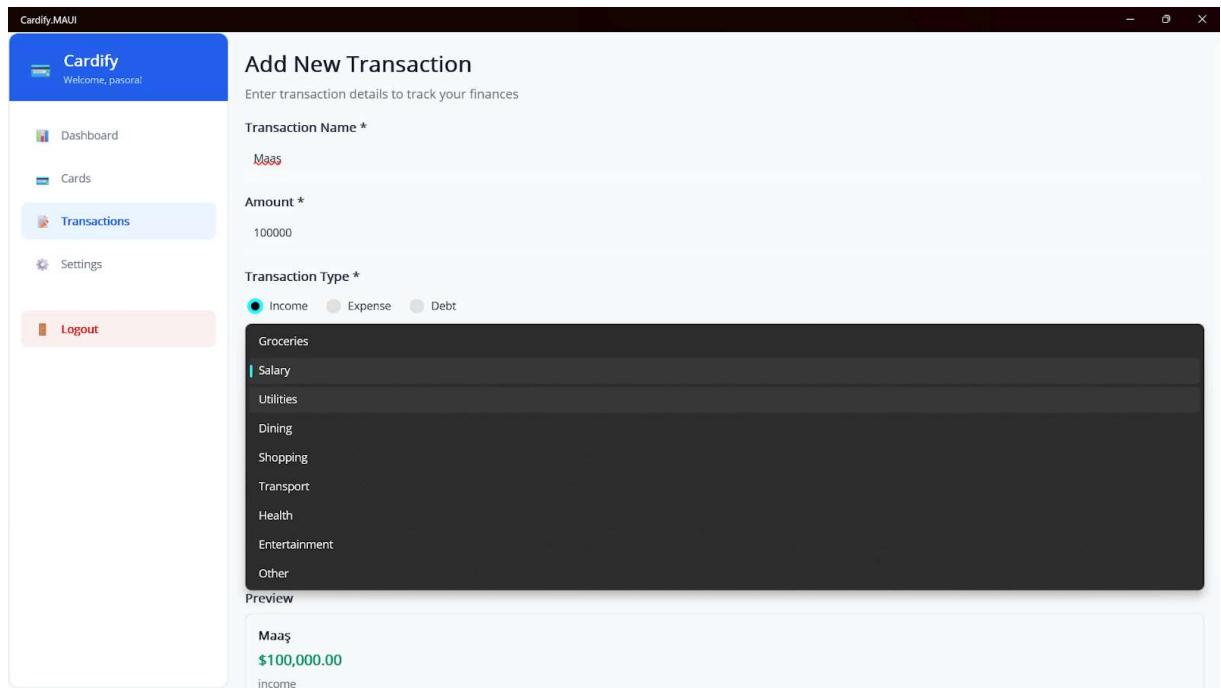
Category *
Select a category

Date *
01/07/2025

Card *
Select a card

Preview

Transaction Name
\$0.00
Type



Cardify.MAUI

Cardify
Welcome, pasora!

Maaş

Amount *
100000

Transaction Type *
 Income Expense Debt

Category *
Select a category
Salary

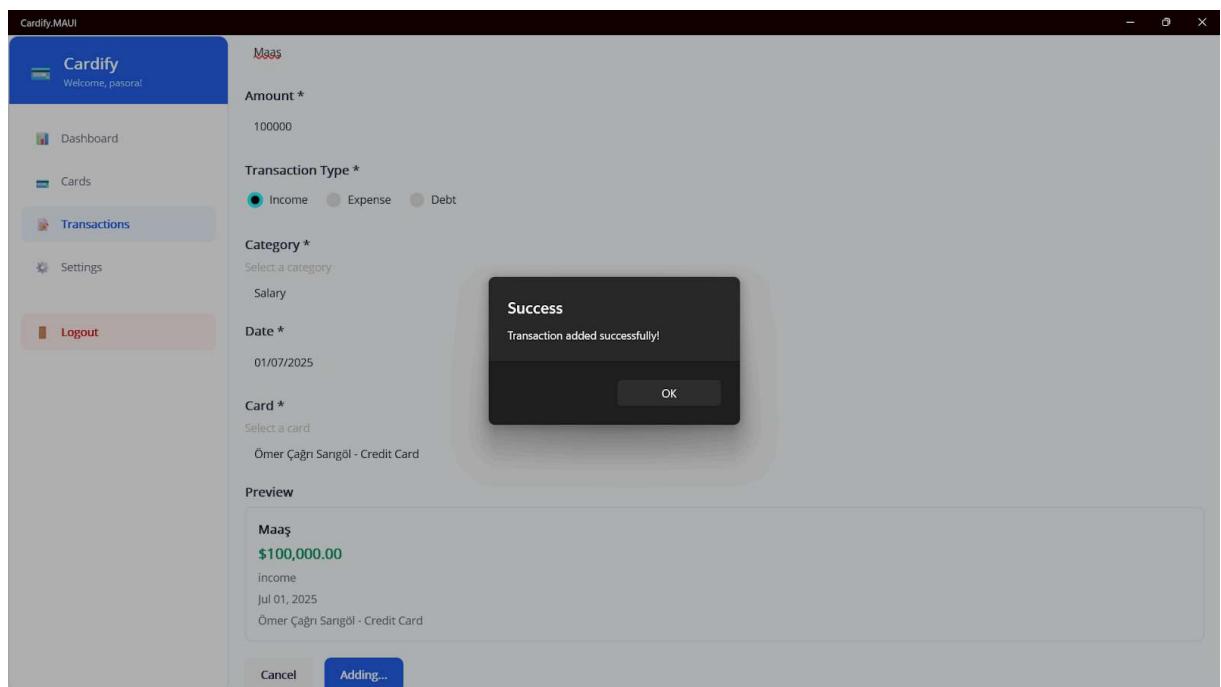
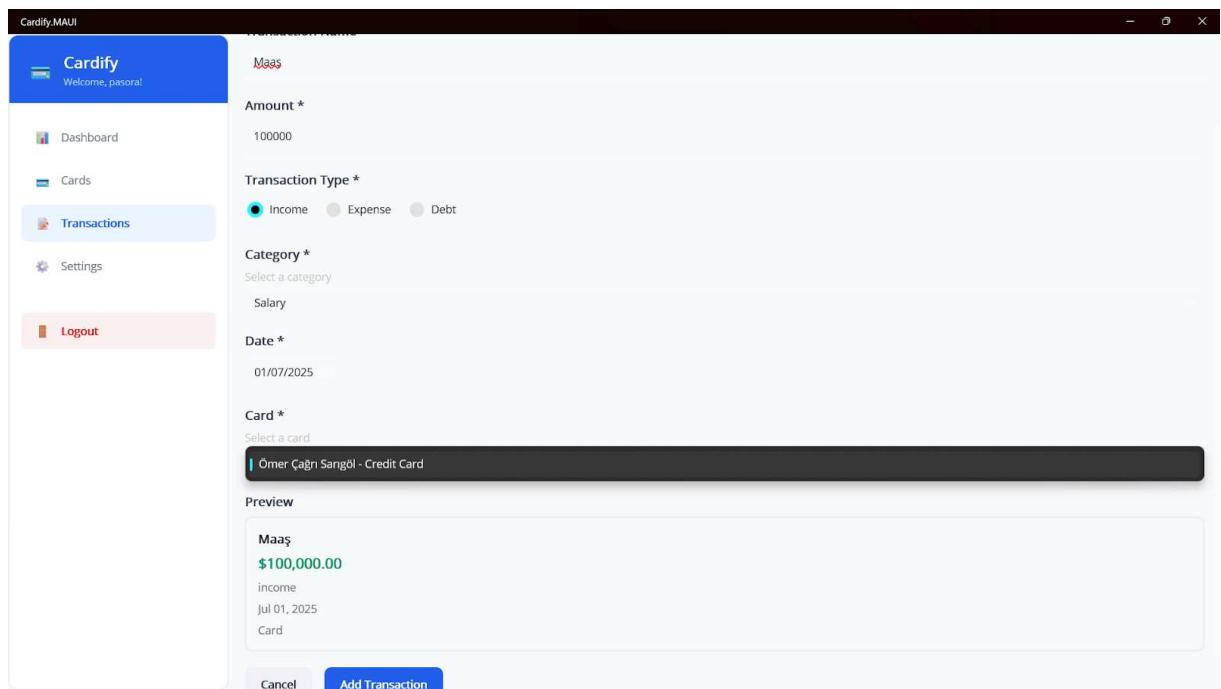
Date *
01/07/2025

Card *
Select a card
Ömer Çağrı Sangöl - Credit Card

Preview

Maaş
\$100,000.00
income
Jul 01, 2025
Card

Add Transaction



Cardify.MAUI

Edit Transaction

Update transaction details

Transaction Name *

Maaş

Amount *

100000.00

Transaction Type *

Income Expense Debt

Category *

Select a category

Salary

Date *

01/07/2025

Card *

Select a card

Ömer Çağrı Sarıgöl - Credit Card

Preview

Maaş	\$100,000.00
income	

Cardify.MAUI

My Transactions

Manage your income, expenses, and debts

Maaş	Jul 01, 2025	\$100,000.00
Card: Ömer Çağrı Sarıgöl		Income
Edit	Delete	

Delete Transaction

Are you sure you want to delete 'Maaş'?

[Delete](#) [Cancel](#)

+ Add Transaction

Cardify.MAUI

My Transactions

Welcome, pasora!

Manage your income, expenses, and debts

Ev kirası
Jul 16, 2025
Card: Omer Çağrı Sangol

\$30,000.00 Debt

Edit Delete

Araba tamir
Jul 01, 2025
Card: Omer Çağrı Sangol

\$20,000.00 Expense

Edit Delete

Yemek
Jul 01, 2025
Card: Omer Çağrı Sangol

\$200.00 Expense

Edit Delete

Maaş
Jul 01, 2025
Card: Omer Çağrı Sangol

\$100,000.00 Income

Edit Delete

Logout

Cardify.MAUI

Dashboard

Welcome, pasora!

Overview of your financial data

1 Total Cards

\$79,800.00 Total Balance

\$100,000.00 Monthly Income

\$20,200.00 Monthly Expenses

Income vs Expenses

Recent Transactions

Ev kirası
Jul 16, 2025

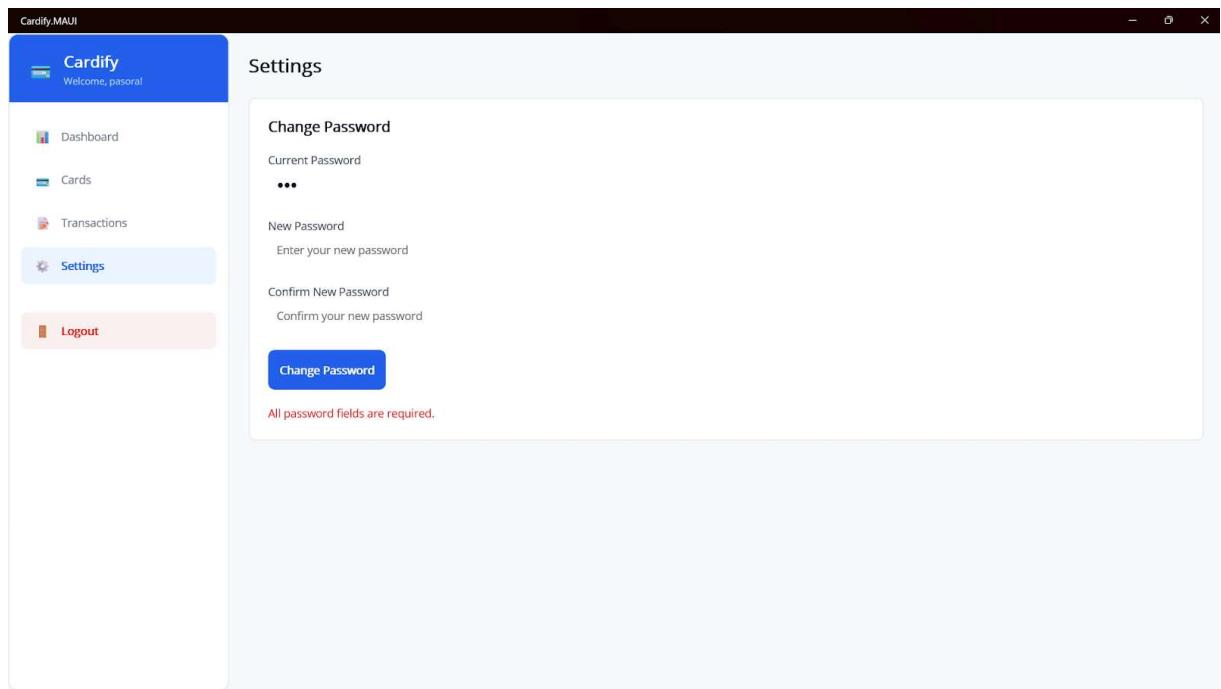
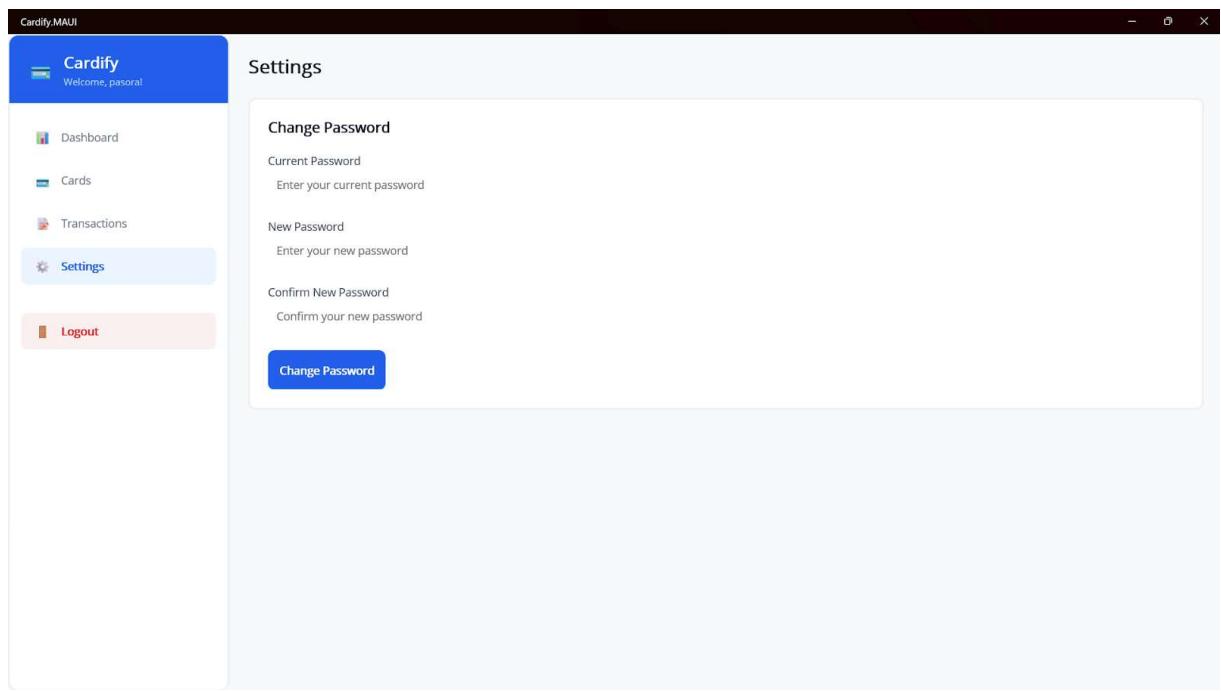
\$30,000.00 Debt

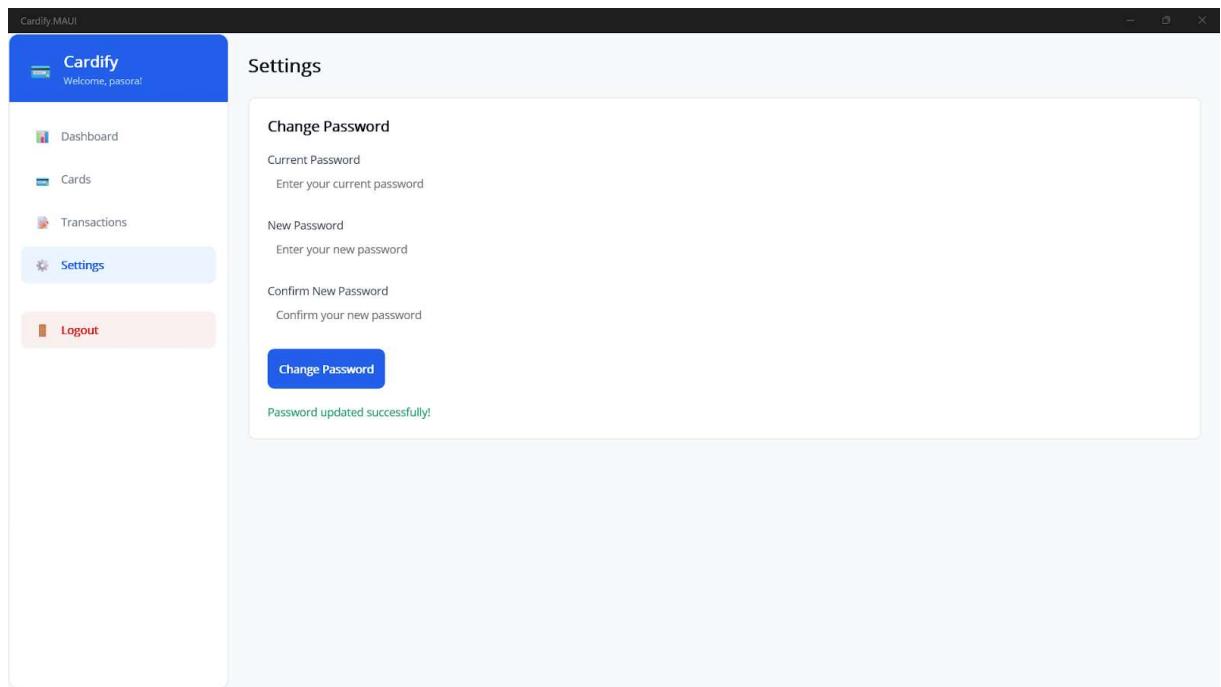
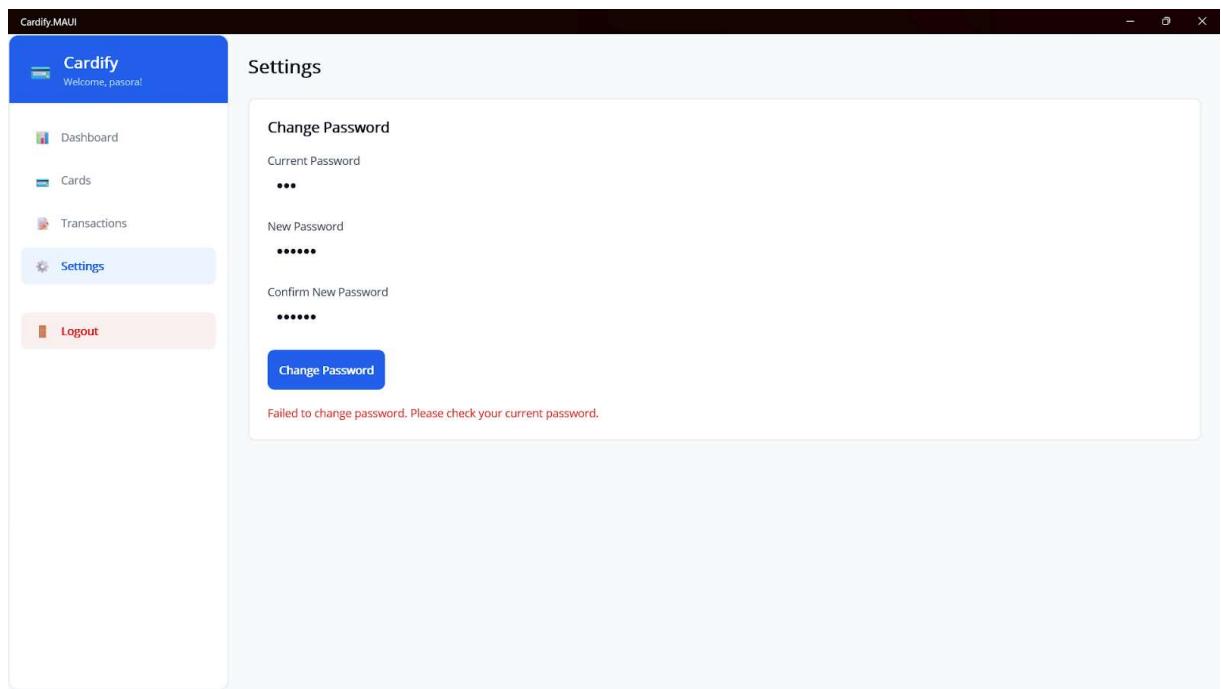
Edit Delete

Araba tamir

Expense

Logout





Bilgisayar Programlama II Dersi Yıl Sonu Projesi İsterleri

1- Proje raporu hazırlanmış mı? Evet.

2- Proje MAUI ile mi yapılmış? Evet.

```
using Cardify.Core.Services;
using Cardify.MAUI.Services;
using Microsoft.Extensions.Logging;
using Microsoft.Maui.LifecycleEvents;
using SkiaSharp.Views.Maui.Controls.Hosting;
using LiveChartsCore.SkiaSharpView.Maui;

namespace Cardify.MAUI;

public static class MauiProgram
{
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder();
        builder
            .UseMauiApp<App>()
            .ConfigureFonts(fonts =>
            {
                fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
                fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
            })
            .UseSkiaSharp()
            .UseLiveCharts()
            ;

#if DEBUG
        builder.Logging.AddDebug();
#endif
        builder.Services.AddSingleton<ILoginService, ApiService>();
    }
}
```

```

namespace Cardify.MAUI.WinUI;

/// <summary>
/// Provides application-specific behavior to supplement the default Application class.
/// </summary>
public partial class App : MauiWinUIApplication
{
    /// <summary>
    /// Initializes the singleton application object. This is the first line of authored code
    /// executed, and as such is the logical equivalent of main() or WinMain().
    /// </summary>
    public App()
    {
        this.InitializeComponent();
    }

    protected override MauiApp CreateMauiApp() => MauiProgram.CreateMauiApp();
}

```

```

Cardify.MAUI > App.xaml
1  <?xml version = "1.0" encoding = "UTF-8" ?>
2  <Application xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
3      xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4      xmlns:local="clr-namespace:Cardify.MAUI"
5      xmlns:converters="clr-namespace:Cardify.MAUI.Converters"
6      x:Class="Cardify.MAUI.App">
7      <Application.Resources>
8          <ResourceDictionary>
9              <ResourceDictionary.MergedDictionaries>
10                 <ResourceDictionary Source="Resources/Styles/Colors.xaml" />
11                 <ResourceDictionary Source="Resources/Styles/Styles.xaml" />
12             </ResourceDictionary.MergedDictionaries>
13
14             <!-- Value Converters -->
15             <converters:BoolToColorConverter x:Key="BoolToColorConverter" />
16             <converters:TypeToColorConverter x:Key="TypeToColorConverter" />
17         </ResourceDictionary>
18     </Application.Resources>
19 </Application>

```

Bu proje, Microsoft'un çapraz platform mobil ve masaüstü uygulama geliştirme çerçevesi olan .NET MAUI ile geliştirilmiştir.

Cardify.MAUI/MauiProgram.cs dosyasında, MAUI uygulamalarına özgü MauiApp.CreateBuilder() ve .UseMauiApp<App>() gibi yapılandırmalar yer almaktadır.

Android, Windows ve iOS platformlarına özel dosyalarda MauiAppCompatActivity, MauiWinUIApplication ve MauiUIApplicationDelegate gibi yalnızca MAUI'de bulunan sınıflar kullanılmıştır.(Raporda windows dosyası kodunu paylaştık.)

Ayrıca App.xaml dosyasında kullanılan <Application xmlns="http://schemas.microsoft.com/dotnet/2021/maui"> bildirimi de MAUI'ye özgü XAML yapısını göstermektedir.

Tüm bu unsurlar, projenin .NET MAUI teknolojisiyle geliştirildiğini açıkça kanıtlamaktadır.

3- Minimal Api projesi oluşturulmuş mu? Evet, oluşturulmuştur.

```
Cardify.API > C# Program.cs
 1  using Cardify.Core;
 2  using Cardify.Core.Models;
 3  using Cardify.Core.Services;
 4  using Microsoft.EntityFrameworkCore;
 5
 6  var builder = WebApplication.CreateBuilder(args);
 7
 8
 9  builder.Services.AddOpenApi();
10  builder.Services.AddDbContext<CardifyDbContext>(options =>
11      options.UseSqlServer(
12          builder.Configuration.GetConnectionString("DefaultConnection"),
13          b => b.MigrationsAssembly("Cardify.API")
14      ));
15  builder.Services.AddEndpointsApiExplorer();
16  builder.Services.AddSwaggerGen();
17
18 // Register services
19 builder.Services.AddScoped<IUserService, UserService>();
20 builder.Services.AddScoped<ICardService, CardService>();
21 builder.Services.AddScoped<ITransactionService, TransactionService>();
22
23 var app = builder.Build();
24
25 // Configure the HTTP request pipeline.
26 if (app.Environment.IsDevelopment())
27 {
28     app.MapOpenApi();
29 }

```



```
app.MapPost("/api/users/register", async (UserCreateDto dto, IUserService userService) => { ... });
app.MapPost("/api/users/login", async (UserLoginDto dto, IUserService userService) => { ... });
app.MapGet("/api/cards", async (int userId, ICardService cardService) => { ... });
app.MapGet("/api/cards/{id}", async (int id, int userId, ICardService cardService) => { ... });
app.MapPost("/api/cards", async (CardCreateDto dto, int userId, ICardService cardService) => { ... });
app.MapPut("/api/cards/{id}", async (int id, CardUpdateDto dto, int userId, ICardService cardService) => { ... });
app.MapDelete("/api/cards/{id}", async (int id, int userId, ICardService cardService) => { ... });
app.MapGet("/api/transactions", async (int userId, int? cardId, string? type, DateTime? fromDate, DateTime? toDate, ITransactionService transactionService) => { ... });
app.MapGet("/api/transactions/{id}", async (int id, int userId, ITransactionService transactionService) => { ... });
app.MapPost("/api/transactions", async (TransactionCreateDto dto, int userId, ITransactionService transactionService) => { ... });
app.MapPut("/api/transactions/{id}", async (int id, TransactionUpdateDto dto, int userId, ITransactionService transactionService) => { ... });
app.MapDelete("/api/transactions/{id}", async (int id, int userId, ITransactionService transactionService) => { ... });
app.MapPost("/api/users/logout", async (int userId, IUserService userService) => { ... });
app.MapPost("/api/users/change-password", async (PasswordChangeDto dto, int userId, IUserService userService) => { ... });
```

```

Cardify.API > Program.cs
35     app.MapPost("/api/users/register", async (UserCreateDto dto, IUserService userService) =>
36     {
37         var result = await userService.RegisterUserAsync(dto);
38         if (!result)
39             return Results.BadRequest("Registration failed. Username or email may already exist.");
40
41         return Results.Ok(new { message = "User registered successfully." });
42     });
43
44
45     app.MapPost("/api/users/login", async (UserLoginDto dto, IUserService userService) =>
46     {
47         var result = await userService.LoginUserAsync(dto);
48         if (result == null)
49             return Results.BadRequest("Invalid username/email or password.");
50
51         return Results.Ok(result);
52     });
53
54     // Card CRUD Endpoints
55     app.MapGet("/api/cards", async (int userId, ICardService cardService) =>
56     {
57         var cards = await cardService.GetUserCardsAsync(userId);
58         return Results.Ok(cards);
59     });
60
61     app.MapGet("/api/cards/{id}", async (int id, int userId, ICardService cardService) =>
62     {
63         var card = await cardService.GetCardByIdAsync(id, userId);
64         if (card == null)
65             return Results.NotFound("Card not found.");
66
67         return Results.Ok(card);
68     });
69
70     app.MapPost("/api/cards", async (CardCreateDto dto, int userId, ICardService cardService) =>
71     {
72         var cardId = await cardService.CreateCardAsync(dto, userId);
73
74         app.MapPut("/api/cards/{id}", async (int id, CardUpdateDto dto, int userId, ICardService cardService) =>
75         {
76             var result = await cardService.UpdateCardAsync(id, dto, userId);
77             if (!result)
78                 return Results.NotFound("Card not found.");
79
80             return Results.Ok(new { message = "Card updated successfully." });
81         });
82
83         app.MapDelete("/api/cards/{id}", async (int id, int userId, ICardService cardService) =>
84         {
85             var result = await cardService.DeleteCardAsync(id, userId);
86             if (!result)
87                 return Results.NotFound("Card not found.");
88
89             return Results.Ok(new { message = "Card deleted successfully." });
90         });
91     });

```

Bu projede, .NET 6 ile gelen Minimal API yapısı kullanılmıştır. Tüm API endpoint tanımlamaları ve yapılandırma kodları Cardify.API/Program.cs dosyasında yer almaktadır.

Minimal API yapısında, klasik Controller sınıfları yerine doğrudan kod içerisinde app.MapGet, app.MapPost, app.MapPut ve app.MapDelete gibi metodlarla HTTP istekleri tanımlanır. Bu yöntemle her route bir lambda fonksiyonuna bağlanır ve isteği işler. Ayrıca, servisler ve veri transfer objeleri (DTO) doğrudan parametre olarak enjekte edilir.

Minimal API, az kodla sade ve okunabilir API'ler oluşturmak için modern bir yaklaşımındır.

Projenin backend tarafı, .NET 6 ve sonrası sürümlerde desteklenen Minimal API mimarisi kullanılarak geliştirilmiştir.

4- Entity Framework Code First yaklaşımı kullanılmış mı? Evet, kullanılmış.

```
Cardify.Core > CardifyDbContext.cs
1  using Microsoft.EntityFrameworkCore;
2  using Cardify.Core.Models;
3
4  namespace Cardify.Core
5  {
6      public class CardifyDbContext : DbContext
7      {
8          public CardifyDbContext(DbContextOptions<CardifyDbContext> options) : base(options) { }
9
10         public DbSet<User> Users { get; set; }
11         public DbSet<Card> Cards { get; set; }
12         public DbSet<Transaction> Transactions { get; set; }
13
14         protected override void OnModelCreating(ModelBuilder modelBuilder)
15         {
16             base.OnModelCreating(modelBuilder);
17
18             // Set decimal precision for Amount
19             modelBuilder.Entity<Transaction>()
20                 .Property(t => t.Amount)
21                 .HasColumnType("decimal(18,2)");
22
23             // Prevent multiple cascade paths
24             modelBuilder.Entity<Transaction>()
25                 .HasOne(t => t.User)
26                 .WithMany(u => u.Transactions)
27                 .HasForeignKey(t => t.UserId)
28                 .OnDelete(DeleteBehavior.Restrict);

```

```
Cardify.Core > Models > User.cs
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Cardify.Core.Models
8  {
9      public class User
10     {
11         public int Id { get; set; }
12         public string Username { get; set; } = string.Empty;
13         public string PasswordHash { get; set; } = string.Empty;
14         public string Email { get; set; } = string.Empty;
15         public string Name { get; set; } = string.Empty;
16         public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
17         public DateTime? UpdatedAt { get; set; }
18         public bool IsActive { get; set; } = true;
19
20         public ICollection<Card>? Cards { get; set; }
21         public ICollection<Transaction>? Transactions { get; set; }
22     }
23 }
24 }
```

```
Cardify.Core > Models > Card.cs
1  using System;
2  using System.Collections.Generic;
3
4  namespace Cardify.Core.Models
5  {
6      public class Card
7      {
8          public int Id { get; set; }
9          public string CardType { get; set; } = string.Empty;
10         public string CardNumber { get; set; } = string.Empty;
11         public string LastFourDigits { get; set; } = string.Empty;
12         public string CardHolderName { get; set; } = string.Empty;
13         public string CardColorStart { get; set; } = "#FFFFFF";
14         public string CardColorEnd { get; set; } = "#CCCCCC";
15         public int UserId { get; set; }
16         public User? User { get; set; }
17         public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
18         public DateTime? UpdatedAt { get; set; }
19         public int? CreatedBy { get; set; }
20         public int? UpdatedBy { get; set; }
21
22         public ICollection<Transaction>? Transactions { get; set; }
23     }
24 }
```

```
namespace Cardify.Core.Models
{
    public class Transaction
    {
        public int Id { get; set; }
        public string Name { get; set; } = string.Empty;
        public DateTime Date { get; set; }
        public decimal Amount { get; set; }
        public string Type { get; set; } = string.Empty; // income, expense, debt, etc.
        public int CardId { get; set; }
        public Card? Card { get; set; }
        public int UserId { get; set; }
        public User? User { get; set; }
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
        public DateTime? UpdatedAt { get; set; }
        public int? CreatedBy { get; set; }
        public int? UpdatedBy { get; set; }
        public string Category { get; set; } = string.Empty;
    }
}
```

```

namespace Cardify.API.Migrations
{
    [DbContext(typeof(CardifyDbContext))]
    [Migration("20250630192552_AddTransactionCategory")]
    partial class AddTransactionCategory
    {
        /// <inheritdoc />
        protected override void BuildTargetModel(ModelBuilder modelBuilder)
        {
#pragma warning disable 612, 618
            modelBuilder
                .HasAnnotation("ProductVersion", "9.0.6")
                .HasAnnotation("Relational:MaxIdentifierLength", 128);

           SqlServerModelBuilderExtensions.UseIdentityColumns(modelBuilder);

            modelBuilder.Entity("Cardify.Core.Models.Card", b =>
            {
                b.Property<int>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("int");

               SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

                b.Property<string>("CardColorEnd")
                    .IsRequired()
                    .HasColumnType("nvarchar(max)");
            });
        }
    }
}

```

Bu projede veritabanı işlemleri **Entity Framework Core (EF Core)** kullanılarak gerçekleştirılmıştır ve yaklaşım olarak **Code First** yöntemi tercih edilmiştir. Bu yaklaşımın, veritabanı yapısı doğrudan C# dilinde tanımlanmış sınıflar üzerinden oluşturulmuştur.

Projenin veritabanı bağlantı, Cardify.Core dizini altında yer alan CardifyDbContext sınıfı aracılığıyla tanımlanmıştır. Bu sınıf, DbContext sınıfından türetilmiş olup, veritabanında karşılık gelecek tabloları temsil eden DbSet özelliklerini barındırmaktadır. Aynı dosya içerisinde override edilen OnModelCreating metodu ile tablo ilişkileri Fluent API kullanılarak yapılandırılmıştır. Örneğin, Transaction sınıfının Amount özelliği için ondalık hassasiyet belirlenmiş ve User ile olan ilişkisi üzerinde silme davranışı özelleştirilmiştir.

Projede veritabanı tablolarını temsil eden veri modelleri Cardify.Core/Models klasörü altında yer almaktır, User, Card ve Transaction sınıfları bu amaçla oluşturulmuştur. Her bir model sınıfında temel özellikler tanımlanmış ve EF Core tarafından tablo sütunlarına dönüştürülecek biçimde yapılandırılmıştır.

Bu yapıların oluşturulmasının ardından veritabanı tarafında değişiklikleri yansıtmak için migration dosyaları oluşturulmuştur. Bu dosyalar, Cardify.API/Migrations/ klasörü altında yer almaktır, örnek olarak InitialCreate, AddCardNumberField ve AddTransactionCategory gibi migration adımları net bir şekilde görülmektedir. Bu klasörde ayrıca EF Core'un otomatik olarak ürettiği .Designer.cs dosyaları ve CardifyDbContextModelSnapshot.cs dosyası da yer almaktadır. Bu yapı, Code First yaklaşımının yalnızca başlatılmadığını, aynı zamanda aktif olarak kullanıldığını da göstermektedir.

5- Migrationlar eklenmiş mi? Evet, eklenmiş.

```
Cardify.API > Migrations > 20250627102120_InitialCreate.cs
1  using System;
2  using Microsoft.EntityFrameworkCore.Migrations;
3
4  #nullable disable
5
6  namespace Cardify.API.Migrations
7  {
8      /// <inheritdoc />
9      public partial class InitialCreate : Migration
10     {
11         /// <inheritdoc />
12         protected override void Up(MigrationBuilder migrationBuilder)
13         {
14             migrationBuilder.CreateTable(
15                 name: "Users",
16                 columns: table => new
17                 {
18                     Id = table.Column<int>(type: "int", nullable: false)
19                         .Annotation("SqlServer:Identity", "1, 1"),
20                     Username = table.Column<string>(type: "nvarchar(max)", nullable: false),
21                     PasswordHash = table.Column<string>(type: "nvarchar(max)", nullable: false),
22                     Email = table.Column<string>(type: "nvarchar(max)", nullable: false),
23                     Name = table.Column<string>(type: "nvarchar(max)", nullable: false),
24                     CreatedAt = table.Column<DateTime>(type: "datetime2", nullable: false),
25                     UpdatedAt = table.Column<DateTime>(type: "datetime2", nullable: true),
26                     IsActive = table.Column<bool>(type: "bit", nullable: false)
27                 },
28             );
29         }
30     }
31 }
```

```
Cardify.API > Migrations > 20250628142740_AddCardNumberField.cs
1  using Microsoft.EntityFrameworkCore.Migrations;
2
3  #nullable disable
4
5  namespace Cardify.API.Migrations
6  {
7      /// <inheritdoc />
8      public partial class AddCardNumberField : Migration
9      {
10         /// <inheritdoc />
11         protected override void Up(MigrationBuilder migrationBuilder)
12         {
13             migrationBuilder.AddColumn<string>(
14                 name: "CardNumber",
15                 table: "Cards",
16                 type: "nvarchar(max)",
17                 nullable: false,
18                 defaultValue: "");
19         }
20
21         /// <inheritdoc />
22         protected override void Down(MigrationBuilder migrationBuilder)
23         {
24             migrationBuilder.DropColumn(
25                 name: "CardNumber",
26                 table: "Cards");
27         }
28     }
29 }
```

```
namespace Cardify.API.Migrations
{
    [DbContext(typeof(CardifyDbContext))]
    partial class CardifyDbContextModelSnapshot : ModelSnapshot
    {
        protected override void BuildModel(ModelBuilder modelBuilder)
        {
#pragma warning disable 612, 618
            modelBuilder
                .HasAnnotation("ProductVersion", "9.0.6")
                .HasAnnotation("Relational:MaxIdentifierLength", 128);

           SqlServerModelBuilderExtensions.UseIdentityColumns(modelBuilder);

            modelBuilder.Entity("Cardify.Core.Models.Card", b =>
            {
                b.Property<int>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("int");

                SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

                b.Property<string>("CardColorEnd")
                    .IsRequired()
                    .HasColumnType("nvarchar(max)");

                b.Property<string>("CardColorStart")
                    .IsRequired()
                    .HasColumnType("nvarchar(max)");

                b.Property<string>("CardHolderName")
                    .IsRequired()
                    .HasColumnType("nvarchar(max)");
            });
        }
    }

    /// <inheritdoc />
    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(
            name: "Transactions");

        migrationBuilder.DropTable(
            name: "Cards");

        migrationBuilder.DropTable(
            name: "Users");
    }
}
```

Bu projede Entity Framework Core (EF Core) kullanılarak veritabanı yapısı Code First yaklaşımıyla oluşturulmuş ve bu yapının sürümlenmesi için migration dosyaları oluşturulmuştur. İlk migration, "dotnet ef migrations add InitialCreate --project Cardify.API" komutu ile başlatılmıştır. Bu komut, projenin başlangıçtaki veritabanı şemasını tanımlayan InitialCreate adında bir migration dosyası oluşturur.

Oluşturulan migration'lar veritabanına "dotnet ef database update --project Cardify.API" komutu ile uygulanır. Bu komut, migration dosyalarında tanımlanan değişiklikleri alıp veritabanının yapısını günceller.

Migration dosyaları, Cardify.API/Migrations/ klasörü içerisinde yer almaktadır. Bu klasörde, projedeki veritabanı yapısının evrimini yansıtan çeşitli migration adımları bulunmaktadır. Örnek olarak: InitialCreate.cs, AddCardNumberField.cs ve AddTransactionCategory.cs isimli migration dosyaları ve bu adımlara ait .Designer.cs dosyaları yer almaktadır. EF Core'un migration sistemine özel olarak oluşturduğu CardifyDbContextModelSnapshot.cs dosyası da aynı klasörde bulunur. Bu dosya, veritabanı modelinin anlık durumunu yansıtarak, yeni migration'ların neye göre oluşturulacağını belirler.

Migration dosyalarında yer alan Up() metodları, veritabanında hangi tabloların, sütunların ve ilişkilerin oluşturulacağını açıkça tanımlar. Örneğin InitialCreate.cs adlı dosyada, kullanıcılar, kartlar ve işlemlerle ilgili temel tabloların nasıl oluşturulacağı detaylandırılmıştır.

6- Veri tabanı bağlantısı yapılmış mı? Evet, yapılmış.

```
Cardify.API > { appsettings.json > ...
1  {
2    "Logging": {
3      "LogLevel": {
4        "Default": "Information",
5        "Microsoft.AspNetCore": "Warning"
6      }
7    },
8    "AllowedHosts": "*",
9    "ConnectionStrings": {
10      "DefaultConnection": "Server=DESKTOP-8I2A02N;Database=CardifyDb;Trusted_Connection=True;MultipleActiveResultSets=true"
11    }
12  }

8  builder.Services.AddOpenApi();
9  builder.Services.AddDbContext<CardifyDbContext>(options =>
10  options.UseSqlServer(
11    builder.Configuration.GetConnectionString("DefaultConnection"),
12    b => b.MigrationsAssembly("Cardify.API")
13  ));
14  );
15  builder.Services.AddEndpointsApiExplorer();
16  builder.Services.AddSwaggerGen();
17
```

```

Cardify.Core > CardifyDbContext.cs
1  using Microsoft.EntityFrameworkCore;
2  using Cardify.Core.Models;
3
4  namespace Cardify.Core
5  {
6      public class CardifyDbContext : DbContext
7      {
8          public CardifyDbContext(DbContextOptions<CardifyDbContext> options) : base(options) { }
9
10         public DbSet<User> Users { get; set; }
11         public DbSet<Card> Cards { get; set; }
12         public DbSet<Transaction> Transactions { get; set; }
13
14         protected override void OnModelCreating(ModelBuilder modelBuilder)
15         {
16             base.OnModelCreating(modelBuilder);
17
18             // Set decimal precision for Amount
19             modelBuilder.Entity<Transaction>()
20                 .Property(t => t.Amount)
21                 .HasColumnType("decimal(18,2)");
22
23             // Prevent multiple cascade paths
24             modelBuilder.Entity<Transaction>()
25                 .HasOne(t => t.User)
26                 .WithMany(u => u.Transactions)
27                 .HasForeignKey(t => t.UserId)
28                 .OnDelete(DeleteBehavior.Restrict);
29         }
30     }
31 }
32
33 var builder = WebApplication.CreateBuilder(args);
34
35
36 builder.Services.AddOpenApi();
37 builder.Services.AddDbContext<CardifyDbContext>(options =>
38     options.UseSqlServer(
39         builder.Configuration.GetConnectionString("DefaultConnection"),
40         b => b.MigrationsAssembly("Cardify.API")
41     );
42
43 builder.Services.AddEndpointsApiExplorer();
44 builder.Services.AddSwaggerGen();
45
46 // Register services
47 builder.Services.AddScoped<IUserService, UserService>();
48 builder.Services.AddScoped<ICardService, CardService>();
49 builder.Services.AddScoped<ITransactionService, TransactionService>();
50
51 var app = builder.Build();
52
53 // Configure the HTTP request pipeline.
54 if (app.Environment.IsDevelopment())
55 {
56     app.MapOpenApi();
57 }
58
59 app.UseSwagger();
60 app.UseSwaggerUI();
61
62 app.UseHttpsRedirection();
63
64 app.MapPost("/api/users/register", async (UserCreateDto dto, IUserService userService) =>

```

Bu projede Entity Framework Core aracılığıyla bir veritabanı bağlantısı yapılandırılmıştır. Bağlantı dizesi, Cardify.API projesi içerisinde yer alan appsettings.json dosyasının "ConnectionStrings" bölümünde "DefaultConnection" anahtarı altında tanımlanmıştır.

Bu bağlantı, uygulamanın Program.cs dosyasında builder.Services.AddDbContext<CardifyDbContext>() metodu aracılığıyla kullanıma sunulmuştur. Bu kod ile CardifyDbContext sınıfı dependency injection sistemine dahil edilmiştir. Aynı satırda .UseSqlServer(...) metodu kullanılarak veritabanı sağlayıcısı olarak SQL Server seçilmiştir.

CardifyDbContext sınıfı, Cardify.Core projesinde tanımlanmış olup tüm veritabanı işlemlerinin merkezi olarak yürütüldüğü sınıfıdır. Bu yapı, hem bağlantının kurulduğunu hem de Code First mimarisiyle entegre edildiğini göstermektedir.

7- Veri tabanından listeleme işlemi yapılmış mı? (Çeşitli kriterlere göre sorgulama yapılabiliyor mu?) Evet, yapılmış.

```
namespace Cardify.Core.Services
{
    public class CardService : ICardService
    {
        private readonly CardifyDbContext _context;

        public CardService(CardifyDbContext context)
        {
            _context = context;
        }

        public async Task<IEnumerable<object>> GetUserCardsAsync(int userId)
        {
            var cards = await _context.Cards
                .Where(c => c.UserId == userId)
                .Select(c => new
                {
                    c.Id,
                    c.CardType,
                    c.CardNumber,
                    c.LastFourDigits,
                    c.CardHolderName,
                    c.CardColorStart,
                    c.CardColorEnd,
                    c.CreatedAt
                })
                .ToListAsync();

            return cards;
        }

        public async Task<object?> GetCardByIdAsync(int id, int userId)
    }
}
```

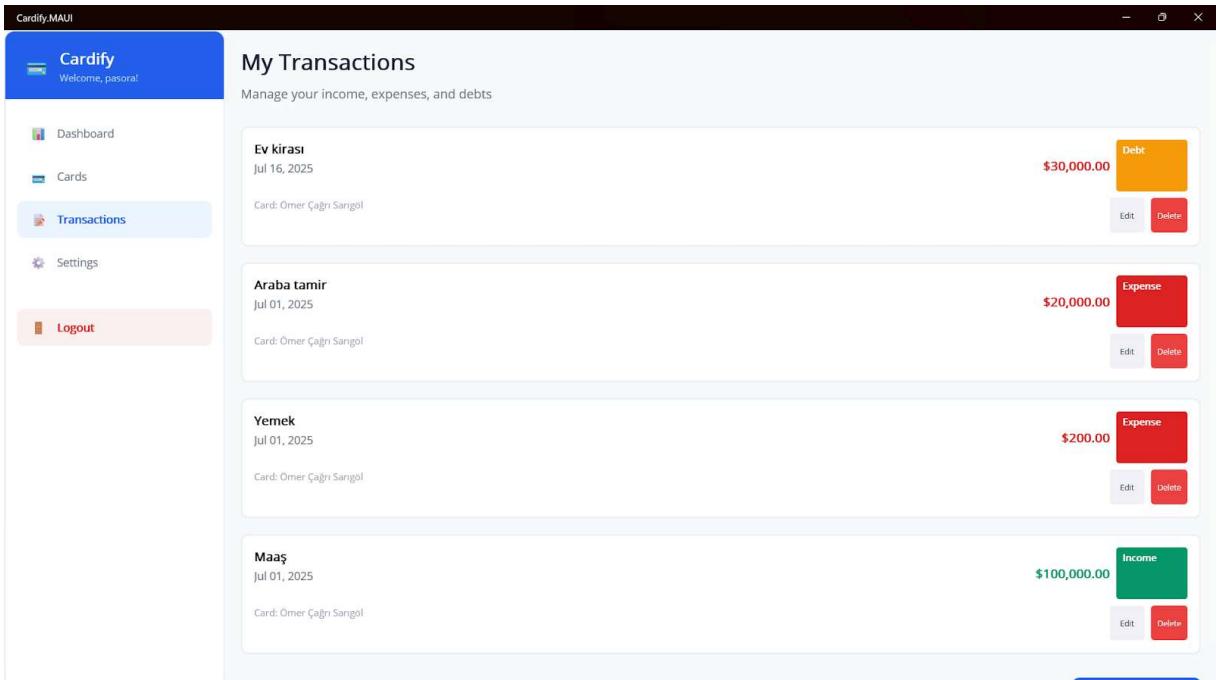
```
Cardify.Core > Services > TransactionService.cs
1  using Cardify.Core.Models;
2  using Microsoft.EntityFrameworkCore;
3
4  namespace Cardify.Core.Services
5  {
6      public class TransactionService : ITransactionService
7      {
8          private readonly CardifyDbContext _context;
9
10         public TransactionService(CardifyDbContext context)
11         {
12             _context = context;
13         }
14
15         public async Task<IEnumerable<object>> GetUserTransactionsAsync(int userId, int? cardId, string? type, DateTime? fromDate, DateTime? toDate)
16         {
17             var query = _context.Transactions.Where(t => t.UserId == userId);
18
19             // Apply filters
20             if (cardId.HasValue)
21                 query = query.Where(t => t.CardId == cardId.Value);
22             if (!string.IsNullOrWhiteSpace(type))
23                 query = query.Where(t => t.Type == type);
24             if (fromDate.HasValue)
25                 query = query.Where(t => t.Date >= fromDate.Value);
26             if (toDate.HasValue)
27                 query = query.Where(t => t.Date <= toDate.Value);
28
29             var transactions = await query
30                 .Include(t => t.Card)
31                 .OrderByDescending(t => t.Date)
32                 .Select(t => new
33                 {
34                     t.Id,
35                     t.Name,
36                     t.Type
37                 })
38                 .ToListAsync();
39
40             return transactions;
41         }
42     }
43 }
```

```
// Transaction CRUD Endpoints
app.MapGet("/api/transactions", async [int userId, int? cardId, string? type, DateTime? fromDate, DateTime? toDate, ITransactionService transactionService] =>
{
    var transactions = await transactionService.GetUserTransactionsAsync(userId, cardId, type, [fromDate], [toDate]);
    return Results.Ok(transactions);
});

app.MapGet("/api/transactions/{id}", async [int id, int userId, ITransactionService transactionService] =>
{
    var transaction = await transactionService.GetTransactionByIdAsync(id, userId);
    if (transaction == null)
        return Results.NotFound("Transaction not found.");

    return Results.Ok(transaction);
});

app.MapPost("/api/transactions", async [TransactionCreateDto dto, int userId, ITransactionService transactionService] =>
{
    var transactionId = await transactionService.CreateTransactionAsync(dto, userId);
    return Results.Created($"/api/transactions/{transactionId}", transactionId);
});
```



Bu projede veritabanından veri listeleme işlemleri Entity Framework Core ve LINQ kullanılarak gerçekleştirilmiştir. Bu işlemler, servis katmanında tanımlanmış ve Minimal API tarafından doğrudan servis çağrılarıyla kullanılmıştır.

Özellikle CardService sınıfındaki GetUserCardsAsync(int userId) metodu, ilgili kullanıcıya ait tüm kartları listelemektedir. Daha gelişmiş filtreleme işlemleri ise TransactionService içerisinde GetUserTransactionsAsync(...) metodunda uygulanmaktadır. Bu metot; kullanıcı ID'si, kart ID'si, işlem türü (örneğin gelir/gider) ve tarih aralığı gibi opsiyonel parametrelerle çalışmaktadır.

Her iki servis sınıfı da LINQ sorguları kullanarak filtreleme ve veri şekillendirme işlemlerini gerçekleştirilmektedir. Bu yapı, hem performanslı, hem de temiz kod prensiplerine uygun bir yapı sunar. Ayrıca veri erişimi servis katmanında toplandığı için projede ayrık sorumluluklar (separation of concerns) prensibine uyulmuştur.

8- Veri tabanına veri ekleme işlemi yapılmış mı? Evet, yapılmış.

```
public async Task<int> CreateCardAsync(CardCreateDto dto, int userId)
{
    // Validate input
    if (string.IsNullOrWhiteSpace(dto.CardType) || string.IsNullOrWhiteSpace(dto.CardNumber) || string.IsNullOrWhiteSpace(dto.CardHolderName))
        return 0;

    // Check if user exists
    var user = await _context.Users.FindAsync(userId);
    if (user == null)
        return 0;

    var card = new Card
    {
        CardType = dto.CardType,
        CardNumber = dto.CardNumber,
        LastFourDigits = dto.LastFourDigits,
        CardHolderName = dto.CardHolderName,
        CardColorStart = dto.CardColorStart,
        CardColorEnd = dto.CardColorEnd,
        UserId = userId,
        CreatedAt = DateTime.UtcNow,
        CreatedBy = userId
    };

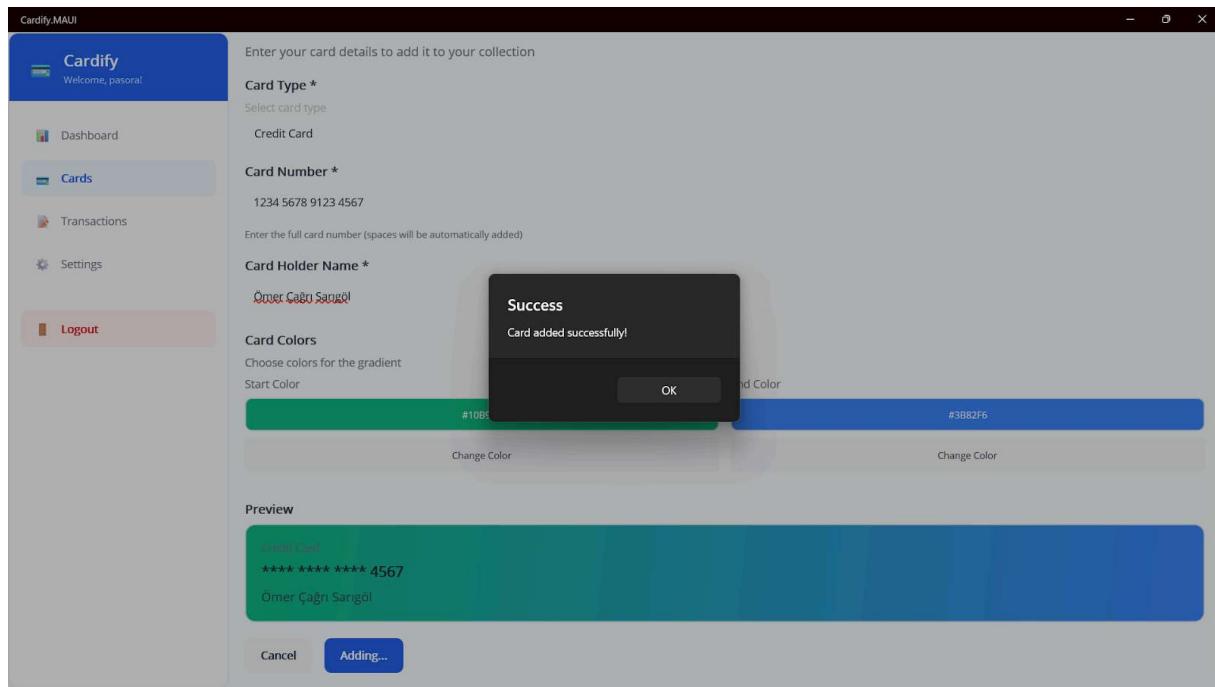
    _context.Cards.Add(card);
    await _context.SaveChangesAsync();

    return card.Id;
}

public async Task<IList<object>> GetUserTransactionsAsync(int userId, int? cardId, string? type, DateTime? fromDate, DateTime? toDate)
{
    var query = _context.Transactions.Where(t => t.UserId == userId);

    // Apply filters
    if (cardId.HasValue)
        query = query.Where(t => t.CardId == cardId.Value);
    if (!string.IsNullOrWhiteSpace(type))
        query = query.Where(t => t.Type == type);
    if (fromDate.HasValue)
        query = query.Where(t => t.Date >= fromDate.Value);
    if (toDate.HasValue)
        query = query.Where(t => t.Date <= toDate.Value);

    var transactions = await query
        .Include(t => t.Card)
        .OrderByDescending(t => t.Date)
        .Select(t => new
    {
        t.Id,
        t.Name,
        t.Date,
        t.Amount,
        t.Type,
        Card = new
        {
            t.Card!.Id,
            t.Card!.CardType,
            t.Card!.LastFourDigits,
            t.Card!.CardHolderName
        },
        t.CreatedAt
    })
    .ToListAsync();
}
```



Projede Entity Framework Core kullanılarak veritabanına veri ekleme işlemi yapılmıştır. Kart ekleme işlemi, CardService sınıfındaki CreateCardAsync metodu ile; işlem (transaction) ekleme ise TransactionService sınıfındaki CreateTransactionAsync metodu ile gerçekleştirilmiştir.

Her iki metod da DTO (örneğin CardCreateDto) kullanarak kullanıcıdan gelen verileri alır, gerekli doğrulamaları yapar, kullanıcı veya kartın varlığını kontrol eder ve uygun entity'yi oluşturarak DbContext üzerinden Add() ve SaveChangesAsync() metotları ile veritabanına kaydeder.

Ekleme işlemleri servis katmanında tanımlandığı için katmanlı mimari ve temiz kod prensiplerine uygun bir yapı sunulmuştur.

9- Veri tabanındaki veri üzerinde güncelleme işlemi yapılmış mı? Evet, yapılmış.

```
public async Task<bool> UpdateCardAsync(int id, CardUpdateDto dto, int userId)
{
    var card = await _context.Cards.FirstOrDefaultAsync(c => c.Id == id && c.UserId == userId);
    if (card == null)
        return false;

    // Update only provided fields
    if (!string.IsNullOrWhiteSpace(dto.CardType))
        card.CardType = dto.CardType;
    if (!string.IsNullOrWhiteSpace(dto.CardNumber))
        card.CardNumber = dto.CardNumber;
    if (!string.IsNullOrWhiteSpace(dto.LastFourDigits))
        card.LastFourDigits = dto.LastFourDigits;
    if (!string.IsNullOrWhiteSpace(dto.CardHolderName))
        card.CardHolderName = dto.CardHolderName;
    if (!string.IsNullOrWhiteSpace(dto.CardColorStart))
        card.CardColorStart = dto.CardColorStart;
    if (!string.IsNullOrWhiteSpace(dto.CardColorEnd))
        card.CardColorEnd = dto.CardColorEnd;

    card.UpdatedAt = DateTime.UtcNow;
    card.UpdatedBy = userId;

    await _context.SaveChangesAsync();
    return true;
}

public async Task<bool> UpdateTransactionAsync(int id, TransactionUpdateDto dto, int userId)
{
    var transaction = await _context.Transactions.FirstOrDefaultAsync(t => t.Id == id && t.UserId == userId);
    if (transaction == null)
        return false;

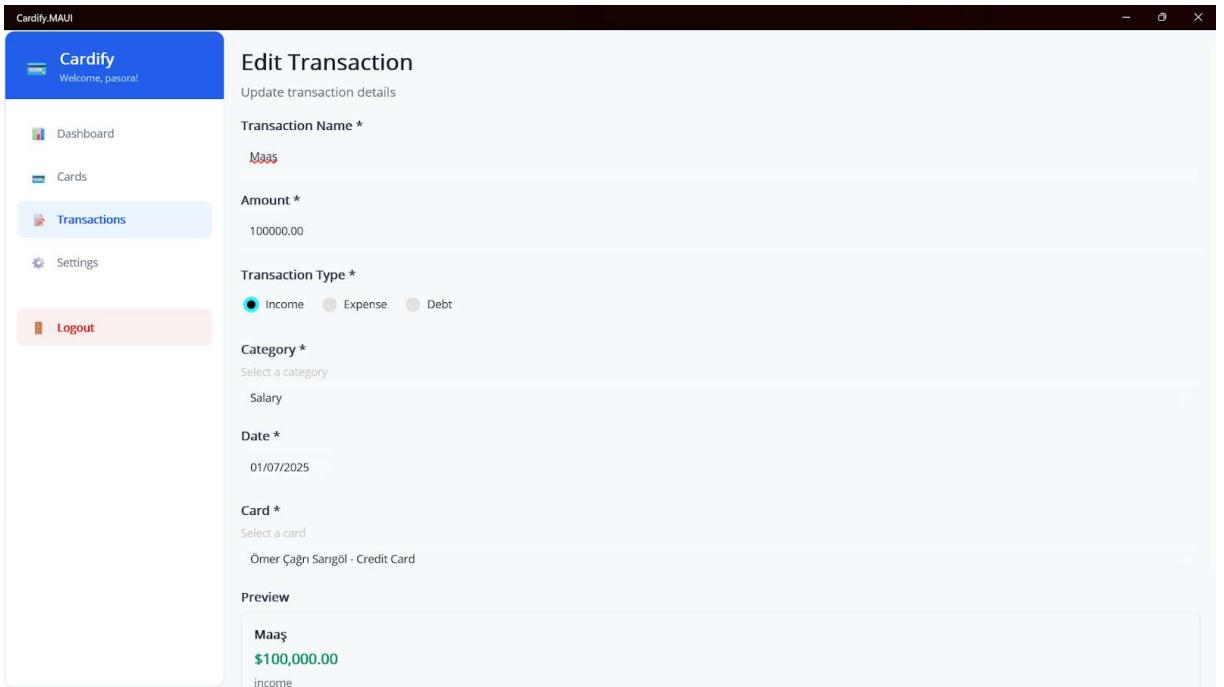
    // Update only provided fields
    if (!string.IsNullOrWhiteSpace(dto.Name))
        transaction.Name = dto.Name;
    if (dto.Date.HasValue)
        transaction.Date = dto.Date.Value;
    if (dto.Amount.HasValue && dto.Amount.Value > 0)
        transaction.Amount = dto.Amount.Value;
    if (!string.IsNullOrWhiteSpace(dto.Type))
        transaction.Type = dto.Type;

    // If CardId is being updated, validate it belongs to user
    if (dto.CardId.HasValue && dto.CardId.Value != transaction.CardId)
    {
        var card = await _context.Cards.FirstOrDefaultAsync(c => c.Id == dto.CardId.Value && c.UserId == userId);
        if (card == null)
            return false;

        transaction.CardId = dto.CardId.Value;
    }

    transaction.UpdatedAt = DateTime.UtcNow;
    transaction.UpdatedBy = userId;

    await _context.SaveChangesAsync();
    return true;
}
```



Projede veritabanı kayıtları üzerinde güncelleme işlemleri yapılmıştır. CardService sınıfındaki UpdateCardAsync ve TransactionService sınıfındaki UpdateTransactionAsync metotları ile sırasıyla kart ve işlem kayıtları güncellenmektedir.

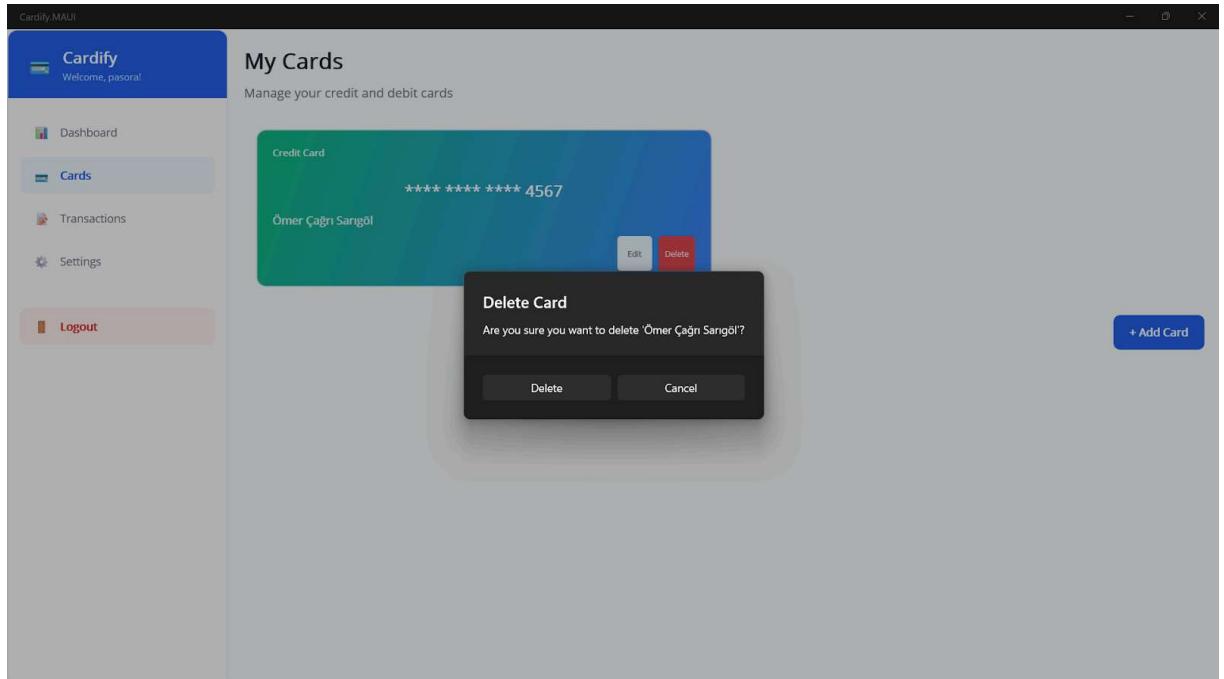
Her iki metotta da DTO kullanılarak alınan veriler, güncellenecek alanlara uygulanmadan önce gerekli kontroller yapılır (kayıt var mı, kullanıcıya ait mi vb.). Ayrıca UpdatedAt ve UpdatedBy alanları güncellenerek işlem kaydı tutulur.

Güncellemeler EF Core aracılığıyla DbContext kullanılarak gerçekleştirilir ve işlem mantığı servis katmanında kapsüllenmiştir.

- 10- Veri tabanındaki veri silme işlemi yapılmış mı? Evet, yapılmış.

```
public async Task<bool> DeleteTransactionAsync(int id, int userId)
{
    var transaction = await _context.Transactions.FirstOrDefaultAsync(t => t.Id == id && t.UserId == userId);
    if (transaction == null)
        return false;

    _context.Transactions.Remove(transaction);
    await _context.SaveChangesAsync();
    return true;
}
```



Projede veritabanı kayıtları üzerinde silme işlemleri yapılmıştır. Bu işlemler CardService ve TransactionService sınıflarında yer alan DeleteCardAsync ve DeleteTransactionAsync metotları ile gerçekleştirilmiştir.

Silme işlemine başlamadan önce, ilgili kart veya işlem kaydının gerçekten mevcut olup olmadığı ve kullanıcıya ait olup olmadığı kontrol edilir. Koşullar sağlanıyorsa, ilgili kayıt EF Core üzerinden DbContext.Remove() metodu ile silinir ve SaveChangesAsync() ile değişiklikler veritabanına uygulanır.

İşlem mantığı servis katmanında tutulduğu için, uygulama temiz mimari ve sorumluluk ayrimı ilkelerine uygun olarak tasarlanmıştır.

- 11-** Ekleme ve güncelleme işlemlerinde hangi kullanıcı ne zaman ekledi ya da güncelledi verileri tutulmuş mu? Evet, tutulmuş.

```
Cardify.Core > Models > C# Card.cs
1  using System;
2  using System.Collections.Generic;
3
4  namespace Cardify.Core.Models
5  {
6      public class Card
7      {
8          public int Id { get; set; }
9          public string CardType { get; set; } = string.Empty;
10         public string CardNumber { get; set; } = string.Empty;
11         public string LastFourDigits { get; set; } = string.Empty;
12         public string CardHolderName { get; set; } = string.Empty;
13         public string CardColorStart { get; set; } = "#FFFFFF";
14         public string CardColorEnd { get; set; } = "#CCCCCC";
15         public int UserId { get; set; }
16         public User? User { get; set; }
17         public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
18         public DateTime? UpdatedAt { get; set; }
19         public int? CreatedBy { get; set; }
20         public int? UpdatedBy { get; set; }
21
22         public ICollection<Transaction>? Transactions { get; set; }
23     }
24 }
```

```
Cardify.Core > Models > C# User.cs
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Cardify.Core.Models
8  {
9      public class User
10     {
11         public int Id { get; set; }
12         public string Username { get; set; } = string.Empty;
13         public string PasswordHash { get; set; } = string.Empty;
14         public string Email { get; set; } = string.Empty;
15         public string Name { get; set; } = string.Empty;
16         public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
17         public DateTime? UpdatedAt { get; set; }
18         public bool IsActive { get; set; } = true;
19
20         public ICollection<Card>? Cards { get; set; }
21         public ICollection<Transaction>? Transactions { get; set; }
22     }
23 }
```

```
Cardify.Core > Models > Transaction.cs
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;
 5  using System.Threading.Tasks;
 6
 7  namespace Cardify.Core.Models
 8  {
 9    public class Transaction
10    {
11      public int Id { get; set; }
12      public string Name { get; set; } = string.Empty;
13      public DateTime Date { get; set; }
14      public decimal Amount { get; set; }
15      public string Type { get; set; } = string.Empty; // income, expense, debt, etc.
16      public int CardId { get; set; }
17      public Card? Card { get; set; }
18      public int UserId { get; set; }
19      public User? User { get; set; }
20      public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
21      public DateTime? UpdatedAt { get; set; }
22      public int? CreatedBy { get; set; }
23      public int? UpdatedBy { get; set; }
24      public string Category { get; set; } = string.Empty;
25    }
26  }
27
```

Projede ekleme ve güncelleme işlemlerinde hangi kullanıcı tarafından ve ne zaman işlem yapıldığı izlenmektedir.

Bu denetim (audit) bilgileri başlıca User, Card ve Transaction entity'lerinde tutulur. Ortak olarak CreatedAt, UpdatedAt, CreatedBy ve UpdatedBy alanları mevcuttur.

CardService ve TransactionService servis sınıflarında, ilgili metodlar (CreateCardAsync, UpdateCardAsync, CreateTransactionAsync, UpdateTransactionAsync) içinde bu alanlar, işlem yapan kullanıcının kimliği ve işlem zamanı olarak kaydedilir.

User entity'sinde ise sadece CreatedAt ve UpdatedAt alanları vardır; kullanıcı kayıt işlemlerinde CreatedBy ve UpdatedBy kullanılmamaktadır.

- 12-** Kullanıcı girişi yapılmış mı? Evet, yapılmış.

```
// Register services
builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<ICardService, CardService>();
builder.Services.AddScoped<ITransactionService, TransactionService>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.MapOpenApi();
}

app.UseSwagger();
app.UseSwaggerUI();

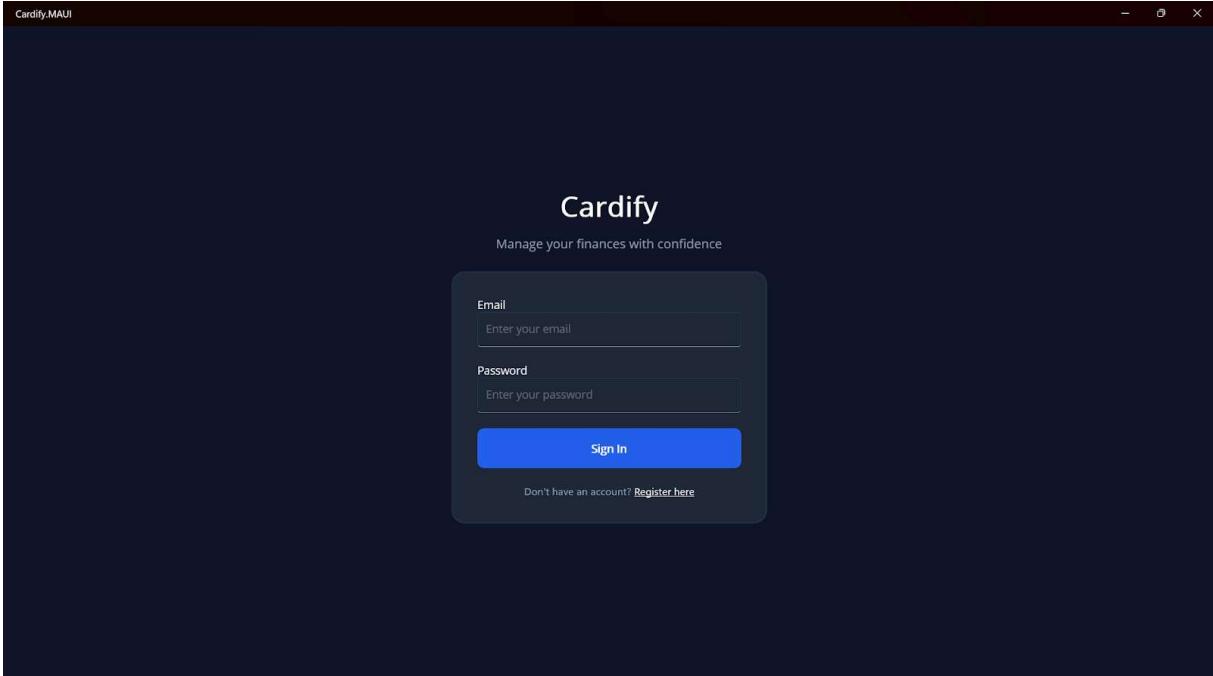
app.UseHttpsRedirection();

app.MapPost("/api/users/register", async [UserCreateDto dto, IUserService userService] =>
{
    var result = await userService.RegisterUserAsync(dto);
    if (!result)
        return Results.BadRequest("Registration failed. Username or email may already exist.");

    return Results.Ok(new { message = "User registered successfully." });
});

app.MapPost("/api/users/login", async (UserLoginDto dto, IUserService userService) =>
{
    var result = await userService.LoginUserAsync(dto);
    if (result == null)
        return Results.BadRequest("Invalid username/email or password.");

    return Results.Ok(result);
});
```



Kullanıcı girişi, Cardify.API projesindeki Program.cs dosyasında, POST /api/users/login endpoint'i olarak tanımlanmıştır. Bu endpoint, kullanıcı adı veya e-posta ile parola bilgilerini alan UserLoginDto veri transfer nesnesini kullanır.

Kimlik doğrulama işlemi UserService sınıfındaki LoginUserAsync metodu tarafından gerçekleştirilir. Bu metot, kullanıcı bilgilerini kontrol eder, kullanıcı adı veya e-posta ile eşleşme arar ve parola doğruluğunu inceler. Başarılıysa kullanıcıya ait oturum bilgileri döner, başarısızsa null döner.

Bu yapı, temel kullanıcı doğrulama sürecini kapsamakta ve projenin kullanıcı giriş gereksinimini karşılamaktadır.

13- Kullanıcı çıkışı yapılmış mı? Evet, yapılmış.

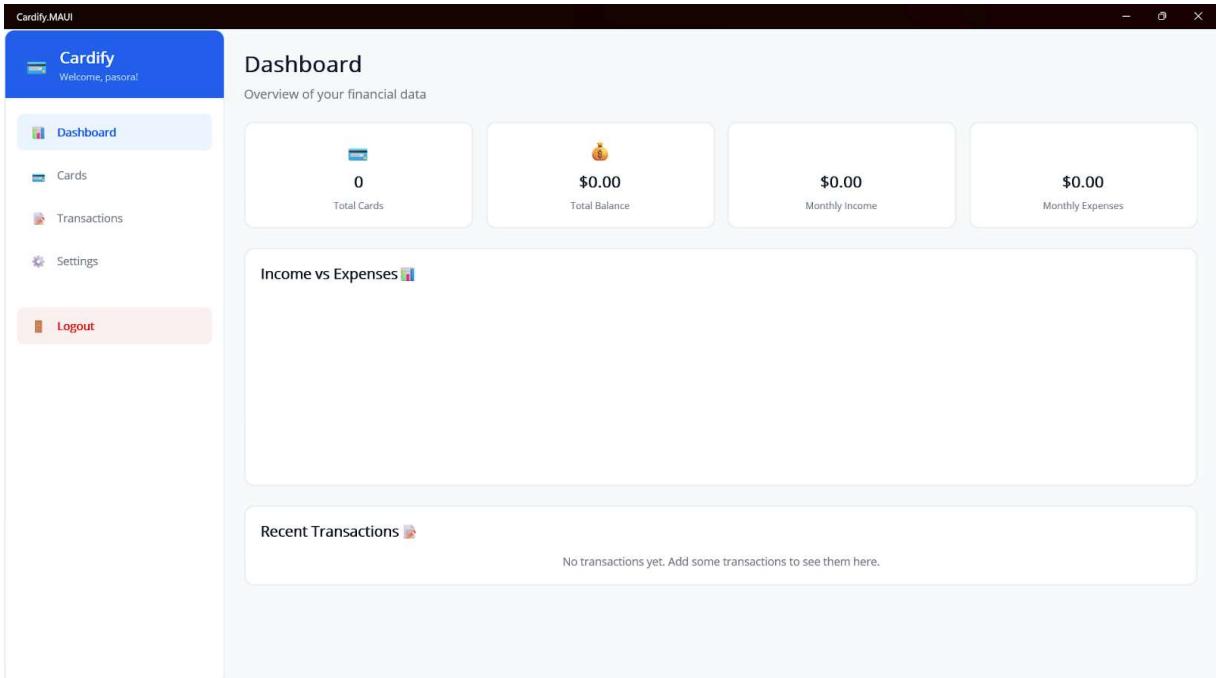
```
// User Logout Endpoint
app.MapPost("/api/users/logout", async (int userId, IUserService userService) =>
{
    var result = await userService.LogoutUserAsync(userId);
    if (!result)
        return Results.BadRequest("User not found.");

    return Results.Ok(new { message = "Logout successful." });
});

// Password Change Endpoint
app.MapPost("/api/users/change-password", async (PasswordChangeDto dto, int userId, IUserService userService) =>
{
    var result = await userService.ChangePasswordAsync(userId, dto);
    if (!result)
        return Results.BadRequest("Password change failed. Please check your current password.");

    return Results.Ok(new { message = "Password changed successfully." });
});

app.Run();
```



Projede kullanıcı çıkışı fonksiyonu mevcuttur.

Kullanıcı çıkışı işlemi, Cardify.API projesindeki Program.cs dosyasında, POST /api/users/logout endpoint'i olarak tanımlanmıştır. Bu endpoint, kullanıcı kimliğini (userId) parametre olarak alır ve UserService sınıfındaki LogoutUserAsync metodu tarafından işlenir.

LogoutUserAsync metodu, genellikle veritabanında ilgili kullanıcı veya oturum kaydını güncelleyerek çıkış işlemini gerçekleştirir. Kodda oturum sonlandırma ya da token iptali ile ilgili özel bir yönetim görünmemekte; dolayısıyla çıkış işlemi büyük ihtimalle bir durum (status) güncellemesidir.

Girdi olarak herhangi bir DTO kullanılmaz; sadece kullanıcı ID'si parametre olarak geçer. Çıkış işlemi başarılı olduğunda, basit bir mesaj içeren anonim bir nesne döndürülür.

14- Şifre değiştirme sayfası yapılmış mı?

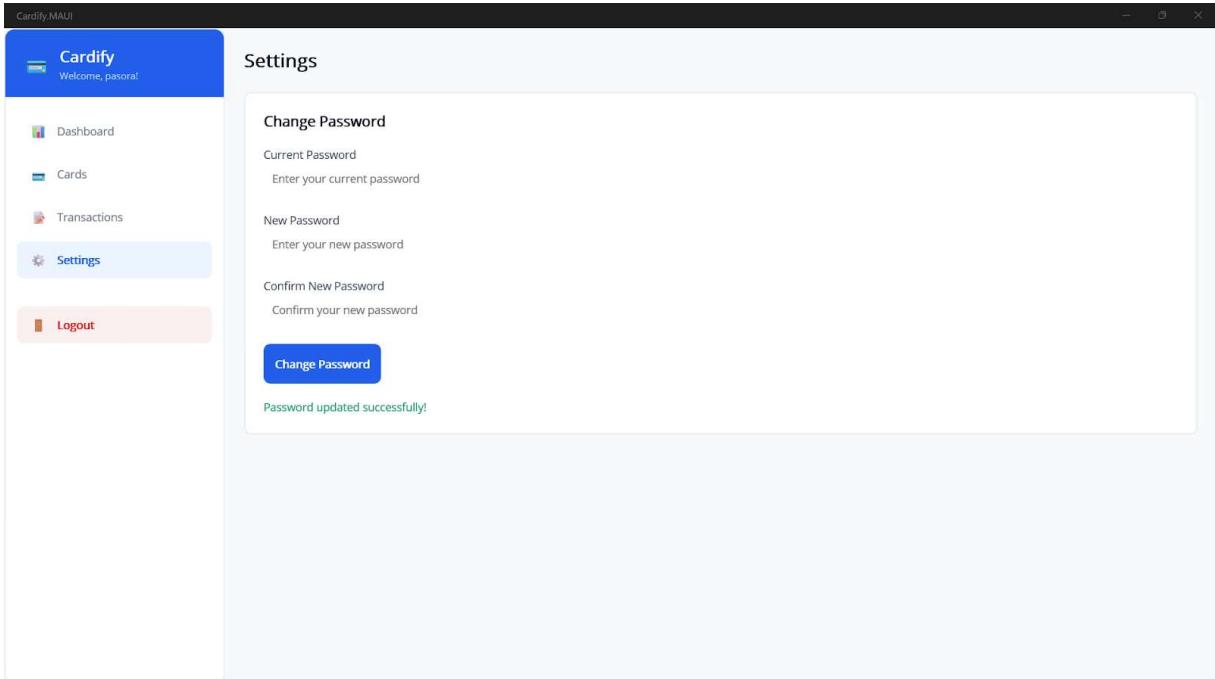
```
// User Logout Endpoint
app.MapPost("/api/users/logout", async (int userId, IUserService userService) =>
{
    var result = await userService.LogoutUserAsync(userId);
    if (!result)
        return Results.BadRequest("User not found.");

    return Results.Ok(new { message = "Logout successful." });
});

// Password Change Endpoint
app.MapPost("/api/users/change-password", async (PasswordChangeDto dto, int userId, IUserService userService) =>
{
    var result = await userService.ChangePasswordAsync(userId, dto);
    if (!result)
        return Results.BadRequest("Password change failed. Please check your current password.");

    return Results.Ok(new { message = "Password changed successfully." });
});

app.Run();
```



Projede şifre değiştirme fonksiyonu hem backend API'de hem de MAUI istemci tarafında uygulanmıştır.

Backend tarafından, Cardify.API/Program.cs dosyasında POST /api/users/change-password endpoint'i tanımlanmıştır. Bu endpoint, genellikle eski ve yeni şifre bilgilerini içeren PasswordChangeDto adlı bir DTO kullanır.

MAUI istemci tarafında ise, Cardify.MAUI/Models/PasswordChangeDto.cs dosyasında şifre değiştirme işlemi için gerekli model bulunmaktadır. Ayrıca, API çağrılarını yöneten ApiUserService.cs servisinde şifre değiştirme işlemini gerçekleştirecek bir metodun mevcut olduğu görülmektedir.

Şifre değiştirme özelliği API tarafından /api/users/change-password endpoint'i ile sağlanmakta; istemci tarafından ise ApiUserService.cs sınıfındaki ChangePasswordAsync metodu ile bu API çağrılmaktadır. PasswordChangeDto modeli veri transferi için kullanılmaktadır.

15- CollectionView kullanılmış mı? Evet, kullanılmış.

```
Cardify.MAUI > Views > TransactionsView.xaml
2   <ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
6     <Grid x:Name="MainGrid">
9       <Grid x:Name="TransactionsListGrid" RowDefinitions="Auto,* ,Auto"
13
14       <!--TransactionsCollectionView-->
15       <CollectionView Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="2"
16         x:Name="TransactionscollectionView"
17         ItemsSource="{Binding Transactions}"
18        SelectionMode="None">
19         <CollectionView.ItemsLayout>
20           <LinearItemsLayout Orientation="Vertical" ItemSpacing="10"/>
21         </CollectionView.ItemsLayout>
22
23         <CollectionView.ItemTemplate>
24           <DataTemplate>
25             <views:TransactionCardTemplate EditClicked="OnEditTransactionClicked"
26               DeleteClicked="OnDeleteTransactionClicked"/>
27           </DataTemplate>
28         </CollectionView.ItemTemplate>
29
30         <CollectionView.EmptyView>
31           <VerticalStackLayout HorizontalOptions="Center"
32             VerticalOptions="Center"
33             Spacing="20">
34             <Label Text="No transactions yet"
35               FontSize="20"
36               FontAttributes="Bold"
37               TextColor="#6B7280"
38               HorizontalOptions="Center"/>
39             <Label Text="Add your first transaction to get started"
40               FontSize="16"
41               TextColor="#9CA3AF"
42               HorizontalOptions="Center"/>
43           </VerticalStackLayout>
44         </CollectionView.EmptyView>
45       </CollectionView>
46   
```

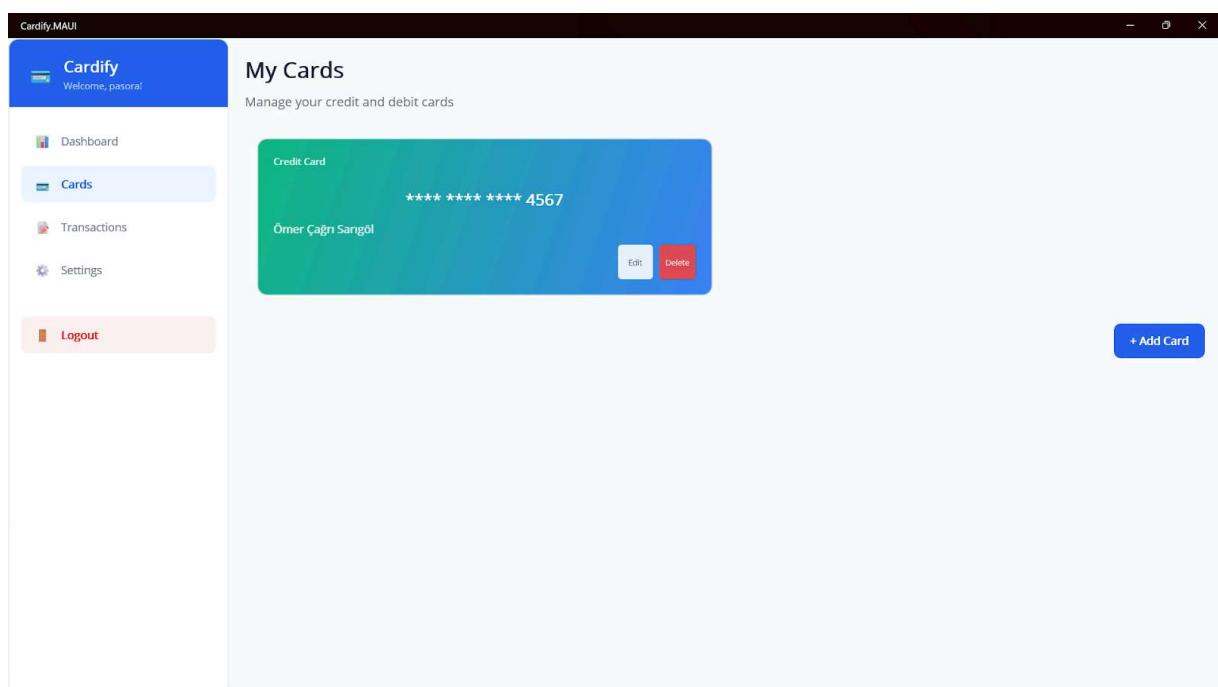
```

<!-- Cards CollectionView -->
<CollectionView Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="2"
    x:Name="CardsCollectionView"
    ItemsSource="{Binding Cards}"
    SelectionMode="None">
    <CollectionView.ItemsLayout>
        <GridItemsLayout Orientation="Vertical"
            Span="2"
            HorizontalItemSpacing="15"
            VerticalItemSpacing="15"/>
    </CollectionView.ItemsLayout>

    <CollectionView.ItemTemplate>
        <DataTemplate>
            <views:CardItemView EditCardClicked="OnEditCardClicked"
                DeleteCardClicked="OnDeleteCardClicked"/>
        </DataTemplate>
    </CollectionView.ItemTemplate>

    <CollectionView.EmptyView>
        <VerticalStackLayout HorizontalOptions="Center"
            VerticalOptions="Center"
            Spacing="20">
            <Label Text="No cards yet"
                FontSize="20"
                FontAttributes="Bold"
                TextColor="#6B7280"
                HorizontalOptions="Center"/>
            <Label Text="Add your first card to get started"
                FontSize="16"
                TextColor="#9CABAF"
                HorizontalOptions="Center"/>
        </VerticalStackLayout>
    </CollectionView.EmptyView>

```

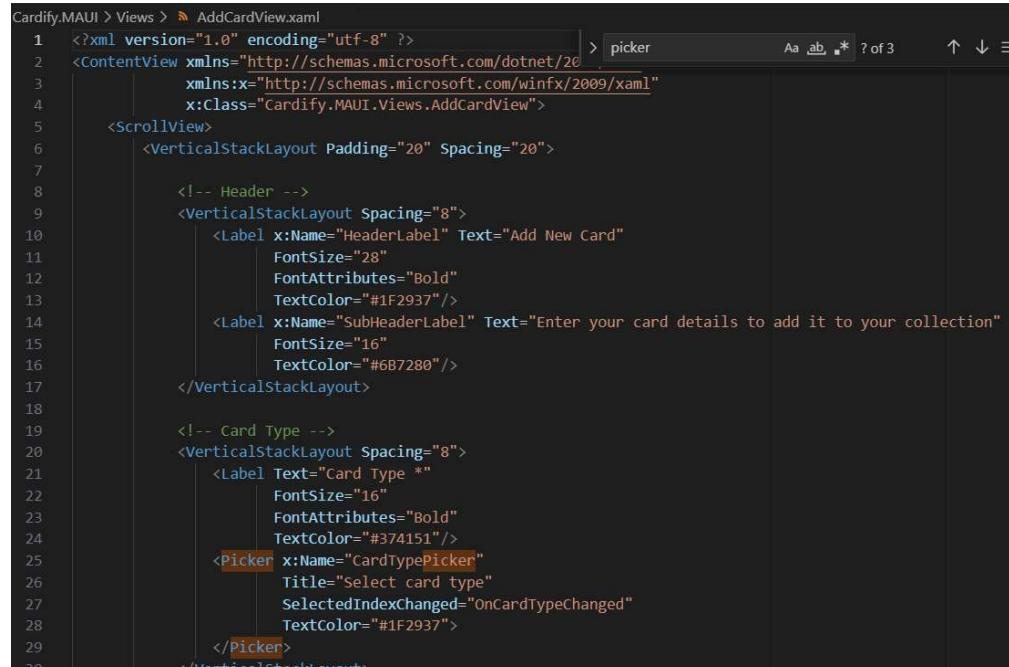


Projede .NET MAUI CollectionView kontrolü aktif şekilde kullanılmaktadır. CardsView.xaml dosyasında kullanıcıya ait kartlar, TransactionsView.xaml dosyasında ise seçilen karta ait işlemler listelenmektedir.

Her iki sayfa da veri kaynağı olarak kod-behind içinde tanımlı ObservableCollection kullanmakta ve ItemTemplate ile her ögenin görünümü özelleştirilmektedir. Veriler, ilgili servislerden çekilerek bu koleksiyonlara aktarılmakta ve sayfaların BindingContext'i doğrudan kendisine atanmıştır.

Böylece kullanıcı arayüzünde kartlar ve işlemler dinamik olarak görüntülenebilmektedir.

16- Picker kullanılmış mı? Evet, kullanılmış.

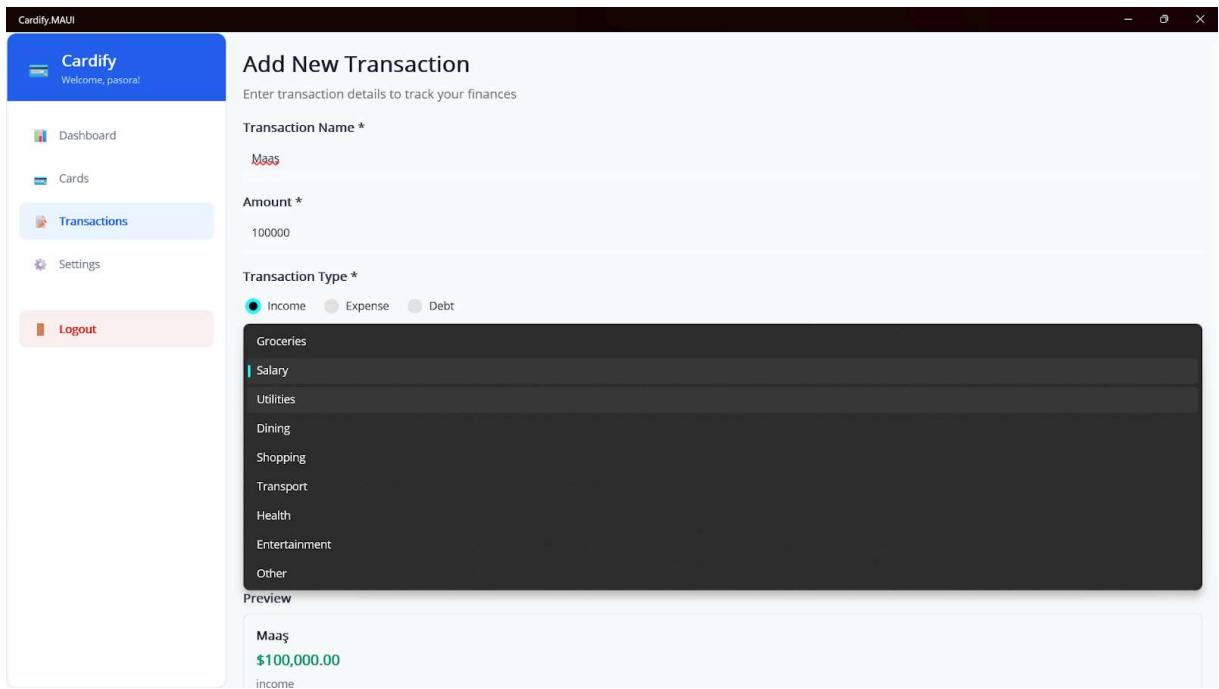


The screenshot shows a Windows Forms application window titled "picker". The search bar at the top contains the text "picker". The main content area displays the XAML code for "AddCardView.xaml". The code includes a `<Picker x:Name="CardtypePicker" ...>` element, which is highlighted in orange. The code is as follows:

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/06/xaml"
3   xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4   x:Class="Cardify.MAUI.Views.AddCardView">
5   <ScrollView>
6     <VerticalStackLayout Padding="20" Spacing="20">
7       <!-- Header -->
8       <VerticalStackLayout Spacing="8">
9         <Label x:Name="HeaderLabel" Text="Add New Card"
10            FontSize="28"
11            FontAttributes="Bold"
12            TextColor="#1F2937"/>
13         <Label x:Name="SubHeaderLabel" Text="Enter your card details to add it to your collection"
14            FontSize="16"
15            TextColor="#6B7280"/>
16       </VerticalStackLayout>
17
18       <!-- Card Type -->
19       <VerticalStackLayout Spacing="8">
20         <Label Text="Card Type *"
21            FontSize="16"
22            FontAttributes="Bold"
23            TextColor="#374151"/>
24         <Picker x:Name="CardtypePicker"
25           Title="Select card type"
26           SelectedIndexChanged="OnCardTypeChanged"
27           TextColor="#1F2937">
28       </Picker>
29     </VerticalStackLayout>
30   </ScrollView>
```

Below the XAML editor, another code editor window is visible, titled "AddTransactionView.xaml.cs". It shows C# code with several `CategoryPicker` instances highlighted in orange. The code is as follows:

```
7 {
19
20   public event EventHandler? TransactionAdded;
21   public event EventHandler? TransactionUpdated;
22   public event EventHandler? Cancelled;
23
24   public AddTransactionView()
25   {
26     InitializeComponent();
27     _transactionService = new ApiTransactionService();
28     _cardService = new ApicardService();
29
30     // Set default date
31     TransactionDatePicker.Date = DateTime.Now;
32
33     // Set category picker items
34     var categories = new List<string> { "Groceries", "Salary", "Utilities", "Dining", "Shopping", "Transport", "Entertainment" };
35     foreach (var category in categories)
36     {
37       CategoryPicker.Items.Add(category);
38     }
39
40     LoadCards();
41     UpdatePreview();
42 }
```



Projede .NET MAUI Picker kontrolü birden fazla yerde kullanılmıştır.

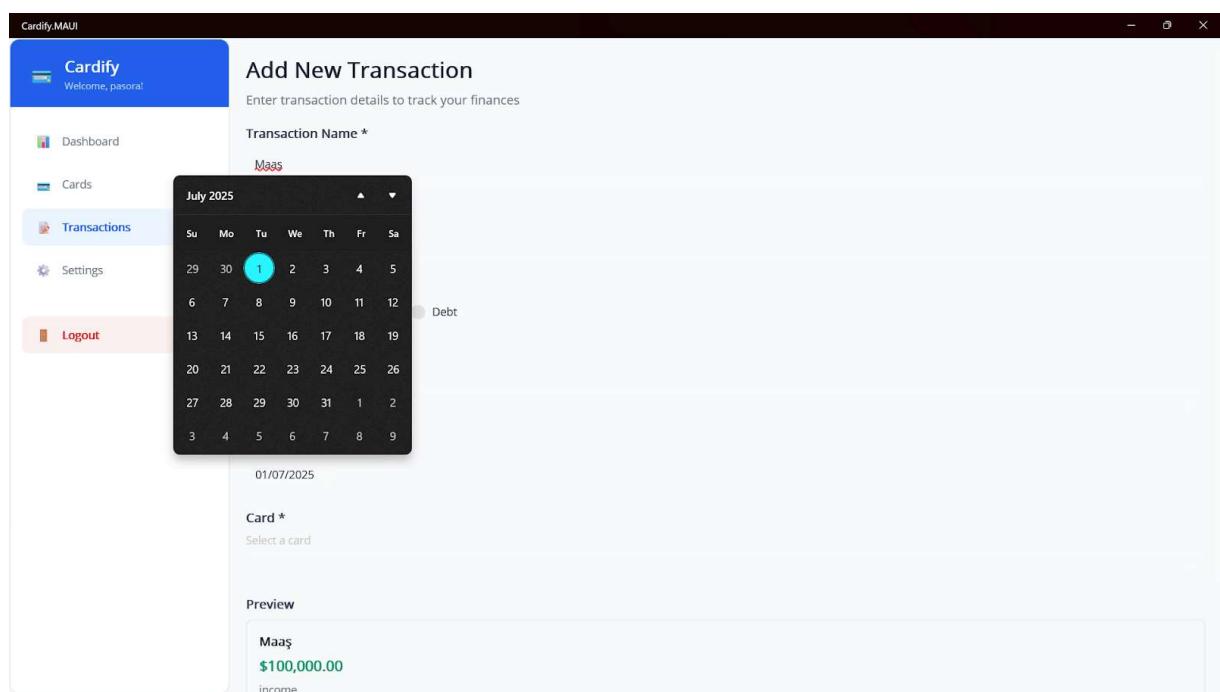
AddTransactionView.xaml ve AddCardView.xaml sayfalarında CategoryPicker, CardPicker ve CardTypePicker gibi Picker bileşenleri yer almaktadır. Bu Picker'lar bazı durumlarda sabit veri (örneğin kategori veya kart türleri), bazı durumlarda ise dinamik veri (kullanıcının kartlarını servis aracılığıyla alma) ile doldurulmuştur.

ItemsSource bağlama yöntemi yerine öğeler doğrudan kod tarafında Picker'ın Items koleksiyonuna eklenmiş, seçilen değerler _selectedCategory, _selectedCardId ve _selectedCardType gibi özel değişkenler aracılığıyla takip edilmiştir.

Seçim değişiklikleri SelectedIndexChanged olayı ile izlenmiş ve düzenleme senaryolarında seçimler programatik olarak atanmıştır. Bu yapı sayesinde, hem sabit hem de dinamik verilerle esnek bir kullanıcı arayüzü sağlanmıştır.

- 17- DatePicker veya TimePicker kullanılmış mı? Evet, DatePicker kullanılmıştır.

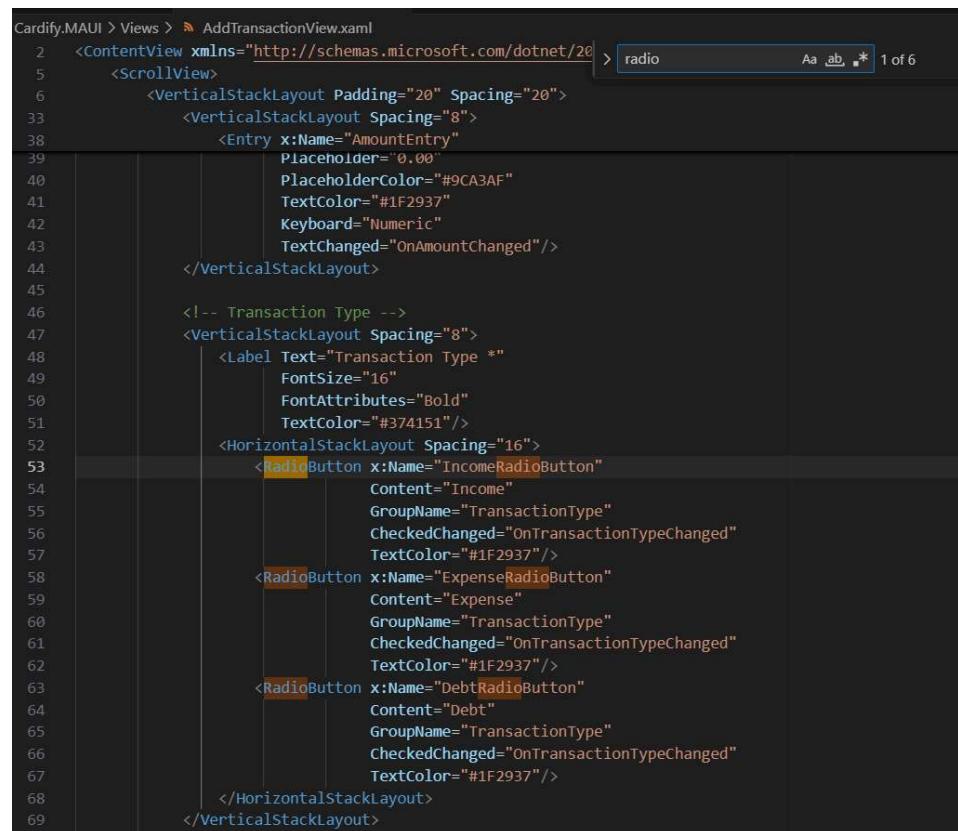
```
Cardify.MAUI > Views > AddTransactionView.xaml
  2   <ContentView xmlns="http://schemas.microsoft.com/dotnet/2023/maui" > DatePicker
  5     <ScrollView>
  6       <VerticalStackLayout Padding="20" Spacing="20">
  7         <!-- Date -->
  8         <VerticalStackLayout Spacing="8">
  9           <Label Text="Date *"
  10             FontSize="16"
  11             FontAttributes="Bold"
  12             TextColor="#374151"/>
  13           <DatePicker x:Name="TransactionDatePicker"
  14             DateSelected="OnDateSelected"
  15             TextColor="#1F2937"/>
  16         </VerticalStackLayout>
  17
  18         <!-- Card Selection -->
  19         <VerticalStackLayout Spacing="8">
  20           <Label Text="Card *"
  21             FontSize="16"
  22             FontAttributes="Bold"
  23             TextColor="#374151"/>
  24           <Picker x:Name="CardPicker"
  25             Title="Select a card"
  26             SelectedIndexChanged="OnCardChanged"
  27             TextColor="#1F2937"/>
  28         </VerticalStackLayout>
  29
  30       </VerticalStackLayout>
  31     </ScrollView>
  32   </ContentView>
```



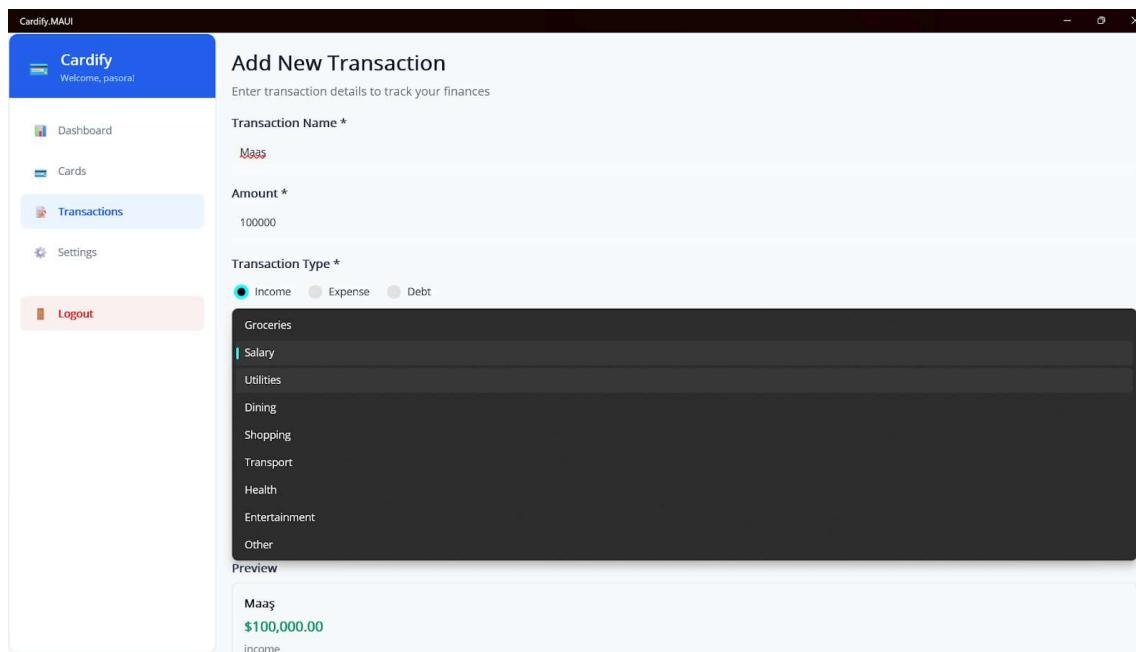
Bu projede DatePicker bileşeni aktif olarak kullanılmaktadır, ancak TimePicker için doğrudan bir kullanım bulunmamaktadır.

DatePicker, AddTransactionView.xaml dosyasında bir işlem (transaction) tarihini seçmek için kullanılmıştır. Seçilen tarih, arka planda _selectedDate adlı bir değişkende saklanmaktadır ve OnDateSelected adlı olay tetikleyicisiyle güncellenmektedir. Bu tarih bilgisi doğrudan veri bağlama (data binding) ile değil, programatik olarak takip edilmektedir. Başlangıç değeri olarak DateTime.Now atanmıştır. MinimumDate veya MaximumDate gibi bir kısıtlama tanımlanmamıştır.

18- Checkbox ya da RadioButton kullanılmış mı? Evet, RadioButton kullanılmıştır.



```
Cardify.MAUI > Views > AddTransactionView.xaml
  2  <ContentView xmlns="http://schemas.microsoft.com/dotnet/2023/maui" x:Name="Root" >
  3      <ScrollView>
  4          <VerticalStackLayout Padding="20" Spacing="20">
  5              <VerticalStackLayout Spacing="8">
  6                  <Entry x:Name="AmountEntry" Placeholder="0.00" PlaceholderColor="#9CA3AF" TextColor="#1F2937" Keyboard="Numeric" TextChanged="OnAmountChanged"/>
  7              </VerticalStackLayout>
  8
  9              <!-- Transaction Type -->
 10             <VerticalStackLayout Spacing="8">
 11                 <Label Text="transaction Type *" FontSize="16" FontAttributes="Bold" TextColor="#374151"/>
 12                 <HorizontalStackLayout Spacing="16">
 13                     <RadioButton x:Name="IncomeRadioButton" Content="Income" GroupName="TransactionType" CheckedChanged="OnTransactionTypeChanged" TextColor="#1F2937"/>
 14                     <RadioButton x:Name="ExpenseRadioButton" Content="Expense" GroupName="TransactionType" CheckedChanged="OnTransactionTypeChanged" TextColor="#1F2937"/>
 15                     <RadioButton x:Name="DebtRadioButton" Content="Debt" GroupName="TransactionType" CheckedChanged="OnTransactionTypeChanged" TextColor="#1F2937"/>
 16                 </HorizontalStackLayout>
 17             </VerticalStackLayout>
 18         </VerticalStackLayout>
 19     </ScrollView>
 20 
```



Bu projede RadioButton kontrolleri kullanılmıştır. Özellikle AddTransactionView.xaml dosyasında, işlem türünü (örneğin gelir, gider, borç) seçmek için birden fazla RadioButton yer almaktadır.

Seçilen değer, kod-behind'da `_selectedType` adlı özel bir değişkende tutulmakta ve her RadioButton için tanımlanan `CheckedChanged` olaylarıyla güncellenmektedir.

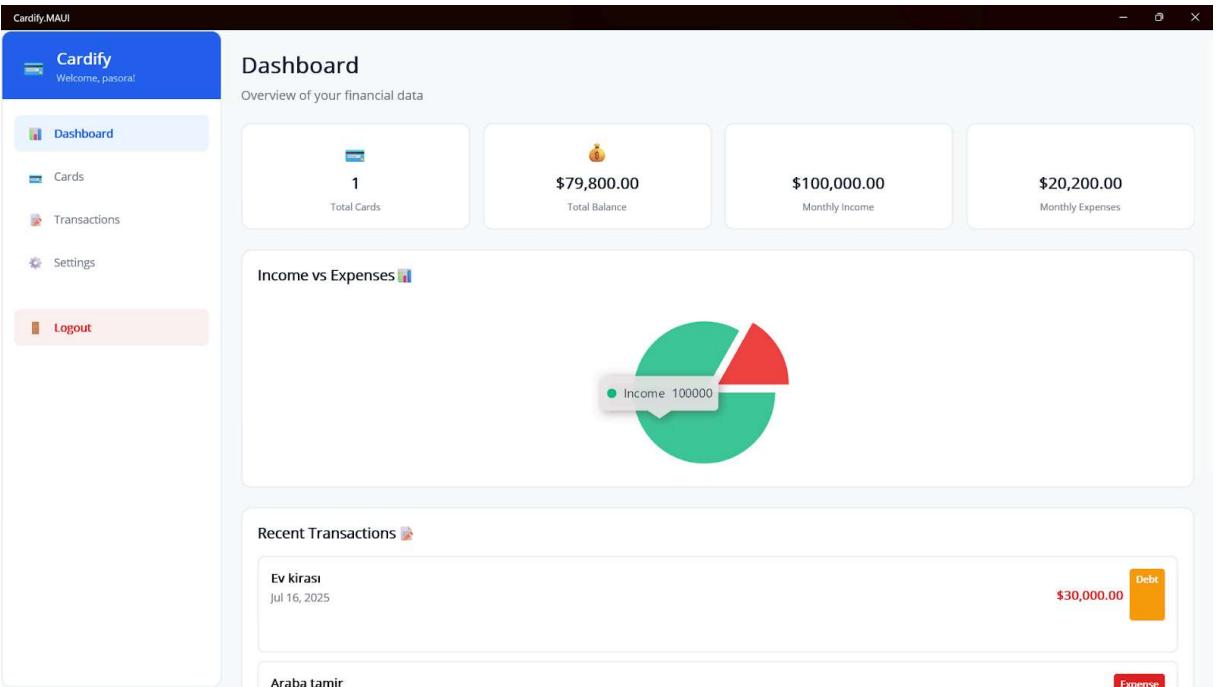
Checkbox kontrolü ise ana görünüm dosyalarında bulunmamaktadır. RadioButton kullanımı, kullanıcı seçimlerinin etkin şekilde izlenmesini sağlamaktadır.

- 19- Derste anlatılmayan MAUi kontrollerinden biri kullanılmış mı? Evet, kullanılmış.

```
Cardify.MAUI > Views > DashboardView.xaml
  2   <ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
108   </ScrollView>
  8     <VerticalStackLayout Spacing="24">
74       <!-- Income vs Expenses Chart -->
75       <Border BackgroundColor="White"
76           Stroke="#E5E7EB" StrokeThickness="1"
77           StrokeShape="RoundRectangle 12" Padding="20">
78         <VerticalStackLayout Spacing="16">
79           <HorizontalStackLayout>
80             <Label Text="Income vs Expenses"
81                 FontSize="18" FontAttributes="Bold"
82                 TextColor="#1F2937" />
83             <Image Source="income_expenses_piechart.png" />
84           </HorizontalStackLayout>
85           <lvc:PieChart x:Name="IncomeExpensesPieChart" HeightRequest="220"/>
86         </VerticalStackLayout>
87       </Border>
```

```
Cardify.MAUI > Views > DashboardView.xaml.cs
10  {
11    {
54      private void UpdateStats(ApiDashboardService.DashboardData data)
55      {
56        TotalCardsLabel.Text = data.ActiveCards.ToString();
57        TotalBalanceLabel.Text = $"{data.TotalBalance:N2}";
58        MonthlyIncomeLabel.Text = $"{data.MonthlyIncome:N2}";
59        MonthlyExpensesLabel.Text = $"{data.MonthlyExpenses:N2}";

60        // Update PieChart for Income vs Expenses
61        IncomeExpensesPieChart.Series = new ISeries[]
62        {
63          new PieSeries<double> { Values = new double[] { data.MonthlyIncome }, Name = "Income", Fill = new SolidColorBrush(new SkiaSharp.SKColor(16, 185, 129)) }, // Green
64          new PieSeries<double> { Values = new double[] { data.MonthlyExpenses }, Name = "Expenses", Fill = new SolidColorBrush(new SkiaSharp.SKColor(239, 68, 68)) } // Red
65        };
66      }
67    }
68  }
```



LiveCharts kütüphanesinin bir parçası olan PieChart kontrolü eklenmiştir. Bu kontrol, MAUI'nin standart kontrolleri arasında yer almaz ve derste anlatılmamıştır. PieChart'ın verileri dinamik olarak kod tarafında atanır. PieSeries ile gelir ve gider değerleri pasta dilimi olarak gösterilir.

- 20- Service Interface'leri kullanılmış mı? Evet, kullanılmış.

```
Cardify.Core > Services > ICardService.cs
1  using Cardify.Core.Models;
2
3  namespace Cardify.Core.Services
4  {
5      public interface ICardService
6      {
7          Task<IEnumerable<object>> GetUserCardsAsync(int userId);
8          Task<object?> GetCardByIdAsync(int id, int userId);
9          Task<int> CreateCardAsync(CardCreateDto dto, int userId);
10         Task<bool> UpdateCardAsync(int id, CardUpdateDto dto, int userId);
11         Task<bool> DeleteCardAsync(int id, int userId);
12     }
13 }
```

```
Cardify.Core > Services > ILoginService.cs
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Cardify.Core.Services
8  {
9      public interface ILoginService
10     {
11         Task<bool> LoginAsync(string email, string password);
12     }
13 }
```

```

Cardify.Core > Services > ITransactionService.cs
1  using Cardify.Core.Models;
2
3  namespace Cardify.Core.Services
4  {
5      public interface ITransactionService
6      {
7          Task<IEnumerable<object>> GetUserTransactionsAsync(int userId, int? cardId, string? type, DateTime? date);
8          Task<object?> GetTransactionByIdAsync(int id, int userId);
9          Task<int> CreateTransactionAsync(TransactionCreateDto dto, int userId);
10         Task<bool> UpdateTransactionAsync(int id, TransactionUpdateDto dto, int userId);
11         Task<bool> DeleteTransactionAsync(int id, int userId);
12     }
13 }

```

```

Cardify.MAUI > Services > ApiCardService.cs
1  using System.Text;
2  using System.Text.Json;
3  using Cardify.MAUI.Models;
4
5  namespace Cardify.MAUI.Services;
6
7  public class ApiCardService : ICardService
8  {
9      private readonly HttpClient _httpClient;
10     private readonly string _baseUrl = "http://localhost:5144/api";
11
12     public ApiCardService()
13     {
14         _httpClient = new HttpClient();
15         _httpClient.DefaultRequestHeaders.Add("Accept", "application/json");
16     }
17
18     public async Task<List<Card>> GetCardsAsync()
19     {
20         try
21         {
22             var userId = ApiLoginService.CurrentUserId;
23             if (userId == null)
24             {
25                 throw new Exception("User not logged in");
26             }
27
28             var response = await _httpClient.GetAsync($"{_baseUrl}/cards?userId={userId}");
29             response.EnsureSuccessStatusCode();
30
31             var content = await response.Content.ReadAsStringAsync();
32             var cards = JsonSerializer.Deserialize<List<Card>>(content, new JsonSerializerOptions
33             {
34                 PropertyNameCaseInsensitive = true
35             });
36
37             return cards ?? new List<Card>();
38         }
39     }
40 }

```

Projede servis arayızları (service interfaces) başarıyla tanımlanmış ve kullanılmıştır. Tüm arayızlar Cardify.Core/Services dizininde yer almaktır, başlıklarını ICardService, ITransactionService, IUserService ve ILoginService’dir.

Her arayüzün somut bir implementasyonu bulunmaktadır (örneğin, ICardService → CardService). Bu arayızlar, API tarafında Cardify.API/Program.cs içinde bağımlılık enjeksiyonu (dependency injection) ile kayıt edilerek, minimal API endpointlerinde parametre olarak kullanılmaktadır.

Ayrıca MAUI istemci uygulamasında da benzer servis arayızları ApiCardService, ApiTransactionService ve ApiLoginService gibi sınıflar tarafından uygulanmaktadır ve API iletişiminde kullanılmaktadır. Bu yapı, projenin hem backend hem frontend katmanlarında sürdürilebilirlik ve test edilebilirlik açısından iyi bir mimari sağlamaktadır.

- 21-** Service sınıfları oluşturulmuş mu? Evet, oluşturulmuş.

The screenshot shows a file explorer or solution browser interface with the following structure:

- KISISELBTUCENEW** (Expanded)
 - Cardify.Core** (Expanded)
 - Services** (Expanded)
 - CardService.cs
 - ICardService.cs
 - ILoginService.cs
 - ITransactionService.cs
 - IUserService.cs
 - TransactionService.cs
 - UserService.cs
 - Cardify.Core.csproj
 - CardifyDbContext.cs
 - Cardify.MAUI** (Expanded)
 - bin
 - Converters
 - Models
 - obj
 - Pages
 - Platforms
 - Properties
 - Resources
 - Services** (Expanded)
 - ApiCardService.cs
 - ApiDashboardService.cs
 - ApiLoginService.cs
 - ApiTransactionService.cs
 - ApiUserService.cs
 - ICardService.cs
 - ITransactionService.cs
 - MockLoginService.cs
 - Views**

Projede servis sınıfları oluşturulmuş ve aktif olarak kullanılmaktadır. Backend tarafında, Cardify.Core/Services klasöründe CardService, TransactionService ve UserService gibi temel servis sınıfları bulunmaktadır. Bu sınıflar iş mantığını ve veri işlemlerini yönetmektedir. Ön ucta (MAUI) ise Cardify.MAUI/Services dizininde API ile iletişim kuran karşılık gelen servis sınıfları yer almaktadır. Bu yapı, projenin katmanlı mimarisi içinde sorumlulukların net ayrılmasını sağlar ve kodun bakımını kolaylaştırır.

22- Proje genelinde OOP prensipleri uygulanmış mı? Evet, uygulanmış.

```
public partial class TransactionsView : ContentView
{
    private readonly ITransactionService _transactionService;

    public ObservableCollection<Transaction> Transactions { get; set; } = new();

    // Events for transaction modifications
    public event EventHandler? TransactionModified;

    public TransactionsView()
    {
        InitializeComponent();
        _transactionService = new ApiTransactionService();
        BindingContext = this;
        LoadTransactions();
    }

    public void OnTabChanged(object sender, string section)
    {
        // Refresh data when transactions tab becomes active
        if (section == "transactions")
        {
            LoadTransactions();
        }
    }

    private async void LoadTransactions()
    {
        try
        {
            var transactions = await _transactionService.GetTransactionsAsync();
            Transactions.Clear();
        }
    }
}

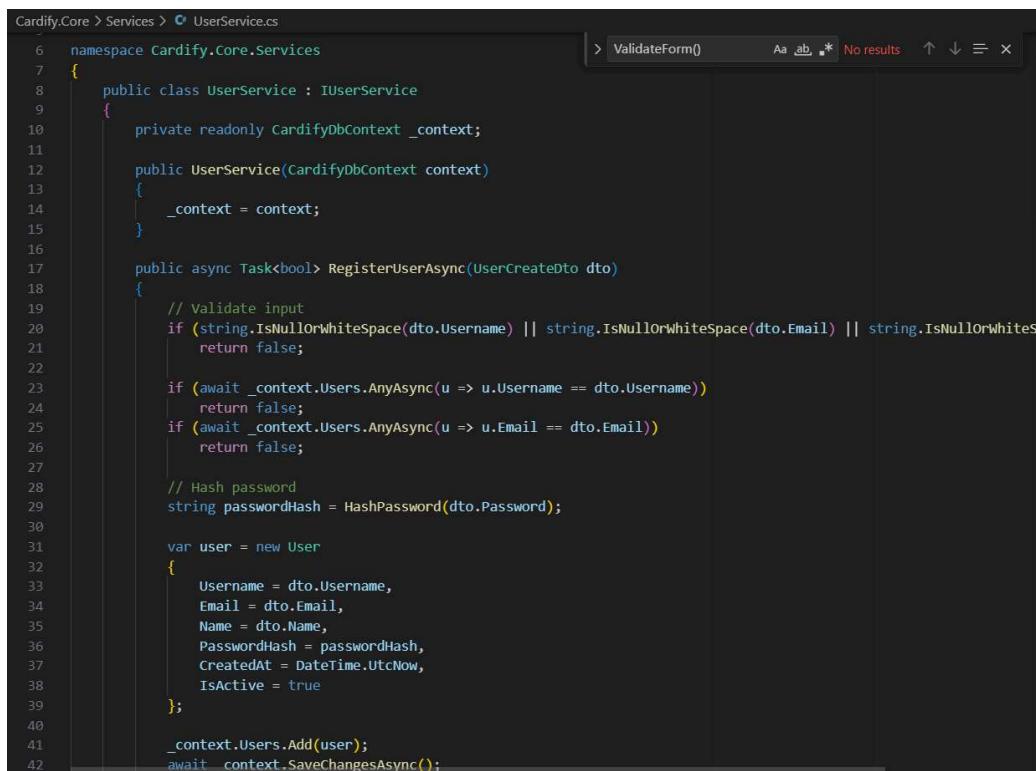
Cardify.API > Program.cs
5
6 var builder = WebApplication.CreateBuilder(args);
7
8
9 builder.Services.AddOpenApi();
10 builder.Services.AddDbContext<CardifyDbContext>(options =>
11     options.UseSqlServer(
12         builder.Configuration.GetConnectionString("DefaultConnection"),
13         b => b.MigrationsAssembly("Cardify.API")
14     ));
15 builder.Services.AddEndpointsApiExplorer();
16 builder.Services.AddSwaggerGen();
17
18 // Register services
19 builder.Services.AddScoped<UserService, UserService>();
20 builder.Services.AddScoped<ICardService, CardService>();
21 builder.Services.AddScoped<ITransactionService, TransactionService>();
22
23 var app = builder.Build();
24
25 // Configure the HTTP request pipeline.
26 if (app.Environment.IsDevelopment())
27 {
28     app.MapOpenApi();
29 }
30
31 app.UseSwagger();
32 app.UseSwaggerUI();
33
34 app.UseHttpsRedirection();
35
36 app.MapPost("/api/users/register", async (UserCreateDto dto, IUserService userService) =>
37 {
38     var result = await userService.RegisterUserAsync(dto);
39     if (!result)
40         return Results.BadRequest("Registration failed. Username or email may already exist.");
41     return Results.Ok(new { message = "User registered successfully." });
42 })
```

```
Cardify.Core > Services > ICardService.cs
1  using Cardify.Core.Models;
2
3  namespace Cardify.Core.Services
4  {
5      public interface ICardService
6      {
7          Task<IEnumerable<object>> GetUserCardsAsync(int userId);
8          Task<object?> GetCardByIdAsync(int id, int userId);
9          Task<int> CreateCardAsync(CardCreateDto dto, int userId);
10         Task<bool> UpdateCardAsync(int id, CardUpdateDto dto, int userId);
11         Task<bool> DeleteCardAsync(int id, int userId);
12     }
13 }
```

```
Cardify.Core > Services > CardService.cs
1  using Cardify.Core.Models;
2  using Microsoft.EntityFrameworkCore;
3
4  namespace Cardify.Core.Services
5  {
6      public class CardService : ICardService
7      {
8          private readonly CardifyDbContext _context;
9
10         public CardService([CardifyDbContext] context)
11         {
12             _context = context;
13         }
14
15         public async Task<IEnumerable<object>> GetUserCardsAsync(int userId)
16         {
17             var cards = await _context.Cards
18                 .Where(c => c.UserId == userId)
19                 .Select(c => new
20                 {
21                     c.Id,
22                     c.CardType,
23                     c.CardNumber,
24                     c.LastFourDigits,
25                     c.CardHolderName,
26                     c.CardColorStart,
27                     c.CardColorEnd,
28                     c.CreatedAt
29                 })
30                 .ToListAsync();
31
32             return cards;
33         }
34
35         public async Task<object?> GetCardByIdAsync(int id, int userId)
36         {
```

Projede Nesne Yönelimli Programlama (OOP) prensipleri uygulanmıştır. Model ve servis yapıları sınıflar üzerinden kurulmuş, kapsülleme ile veri gizliliği sağlanmış, DbContext gibi yapılarla kalıtım kullanılmıştır. Servisler interface'ler üzerinden çağrılmış, böylece çok biçimlilik ve bağımlılıkların esnek yönetimi mümkün olmuştur. Proje genelinde sınıf yapısı, encapsulation, inheritance ve interface kullanımı gibi OOP temelleri açık şekilde görülmektedir.

23- Kaydetme, güncelleme, kullanıcı girişi gibi sayfalardaki veriler doğrulanmış mı (Boş bırakılamaz, karakter sınırı vb.)? Evet, doğrulanmış.



The screenshot shows a code editor window with the following details:

- Project:** Cardify.Core
- Namespace:** Cardify.Core.Services
- File:** UserService.cs
- Code Content:**

```
6  namespace Cardify.Core.Services
7  {
8      public class UserService : IUserService
9      {
10         private readonly CardifyDbContext _context;
11
12         public UserService(CardifyDbContext context)
13         {
14             _context = context;
15         }
16
17         public async Task<bool> RegisterUserAsync(UserCreateDto dto)
18         {
19             // Validate input
20             if (string.IsNullOrWhiteSpace(dto.Username) || string.IsNullOrWhiteSpace(dto.Email) || string.IsNullOrWhiteSpace(dto.Password))
21                 return false;
22
23             if (await _context.Users.AnyAsync(u => u.Username == dto.Username))
24                 return false;
25             if (await _context.Users.AnyAsync(u => u.Email == dto.Email))
26                 return false;
27
28             // Hash password
29             string passwordHash = HashPassword(dto.Password);
30
31             var user = new User
32             {
33                 Username = dto.Username,
34                 Email = dto.Email,
35                 Name = dto.Name,
36                 PasswordHash = passwordHash,
37                 CreatedAt = DateTime.UtcNow,
38                 IsActive = true
39             };
40
41             _context.Users.Add(user);
42             await _context.SaveChangesAsync();
        }
```
- Search Bar:** ValidateForm()
- Toolbars:** Aa ab, No results, navigation icons

```
Cardify.MAUI > Pages > C# SignUpPage.xaml.cs
7  {
16      private async void OnSignUpClicked(object sender, EventArgs e)
17      {
18          string username = UsernameEntry.Text;
19          string email = EmailEntry.Text;
20          string password = PasswordEntry.Text;
21          string confirmPassword = ConfirmPasswordEntry.Text;
22
23          if (string.IsNullOrWhiteSpace(username) ||
24              string.IsNullOrWhiteSpace(email) ||
25              string.IsNullOrWhiteSpace(password) ||
26              string.IsNullOrWhiteSpace(confirmPassword))
27          {
28              await DisplayAlert("Error", "All fields are required.", "OK");
29              return;
30          }
31
32          if (password.Length < 6)
33          {
34              await DisplayAlert("Error", "Password must be at least 6 characters.", "OK");
35              return;
36          }
37
38          if (password != confirmPassword)
39          {
40              await DisplayAlert("Error", "Passwords do not match.", "OK");
41              return;
42          }
43
44          if (!email.Contains("@") || !email.Contains("."))
45          {
46              await DisplayAlert("Error", "Please enter a valid email address.", "OK");
47              return;
48          }
49
50 }
```

```
Cardify.MAUI > Views > C# AddCardView.xaml.cs
7  {
260
261
262     private void ValidateForm()
263     {
264         var isValid = !string.IsNullOrWhiteSpace(_selectedCardType) &&
265                     !string.IsNullOrWhiteSpace(_cardNumber) &&
266                     _cardNumber.Length >= 13 && // Minimum card number length
267                     _cardNumber.Length <= 16 && // Maximum card number length
268                     !string.IsNullOrWhiteSpace(_cardHolderName);
269
270         AddCardButton.IsEnabled = isValid;
271     }
272 }
```

Projede hem backend (API) hem de frontend (MAUI) tarafında kullanıcıdan alınan veriler doğrulanmaktadır. Kullanıcının boş alan bırakmaması, belirli karakter sınırlarına uyması ve geçerli girişler yapması için çeşitli kontroller uygulanmıştır.

MAUI tarafında örneğin AddCardView.xaml.cs dosyasında kart ekleme işlemi öncesi gerekli alanların (kart türü, numarası, isim vb.) boş bırakılmaması kontrol edilir. Aynı şekilde giriş ekranında kullanıcı adı ve şifre girilmeden işlem yapılmasına izin verilmez.

Backend tarafında ise CardService.cs ve UserService.cs gibi servis sınıflarında, API'ye gelen isteklerde DTO içeriği kontrol edilerek gerekli alanların dolu olup olmadığı ve verilerin uygun formatta olup olmadığı denetlenir. Örneğin kart eklerken string.IsNullOrWhiteSpace() gibi ifadelerle boşluk kontrolleri yapılır. Ayrıca girilen kart numarası uzunluğu gibi sınırlamalar da uygulanmıştır.

Bu kontroller sayesinde, sistemde veri bütünlüğünü korunmakta ve kullanıcı hataları en aza indirilmektedir.

24- LINQ aktif olarak kullanılmış mı? Evet, kullanılmış.

```
namespace Cardify.Core.Services
{
    public class CardService : ICardService
    {
        private readonly CardifyDbContext _context;

        public CardService(CardifyDbContext context)
        {
            _context = context;
        }

        public async Task<IEnumerable<object>> GetUserCardsAsync(int userId)
        {
            var cards = await _context.Cards
                .Where(c => c.UserId == userId)
                .Select(c => new
                {
                    c.Id,
                    c.CardType,
                    c.CardNumber,
                    c.LastFourDigits,
                    c.CardHolderName,
                    c.CardColorStart,
                    c.CardColorEnd,
                    c.CreatedAt
                })
                .ToListAsync();

            return cards;
        }

        public async Task<object?> GetCardByIdAsync(int id, int userId)
        {

```

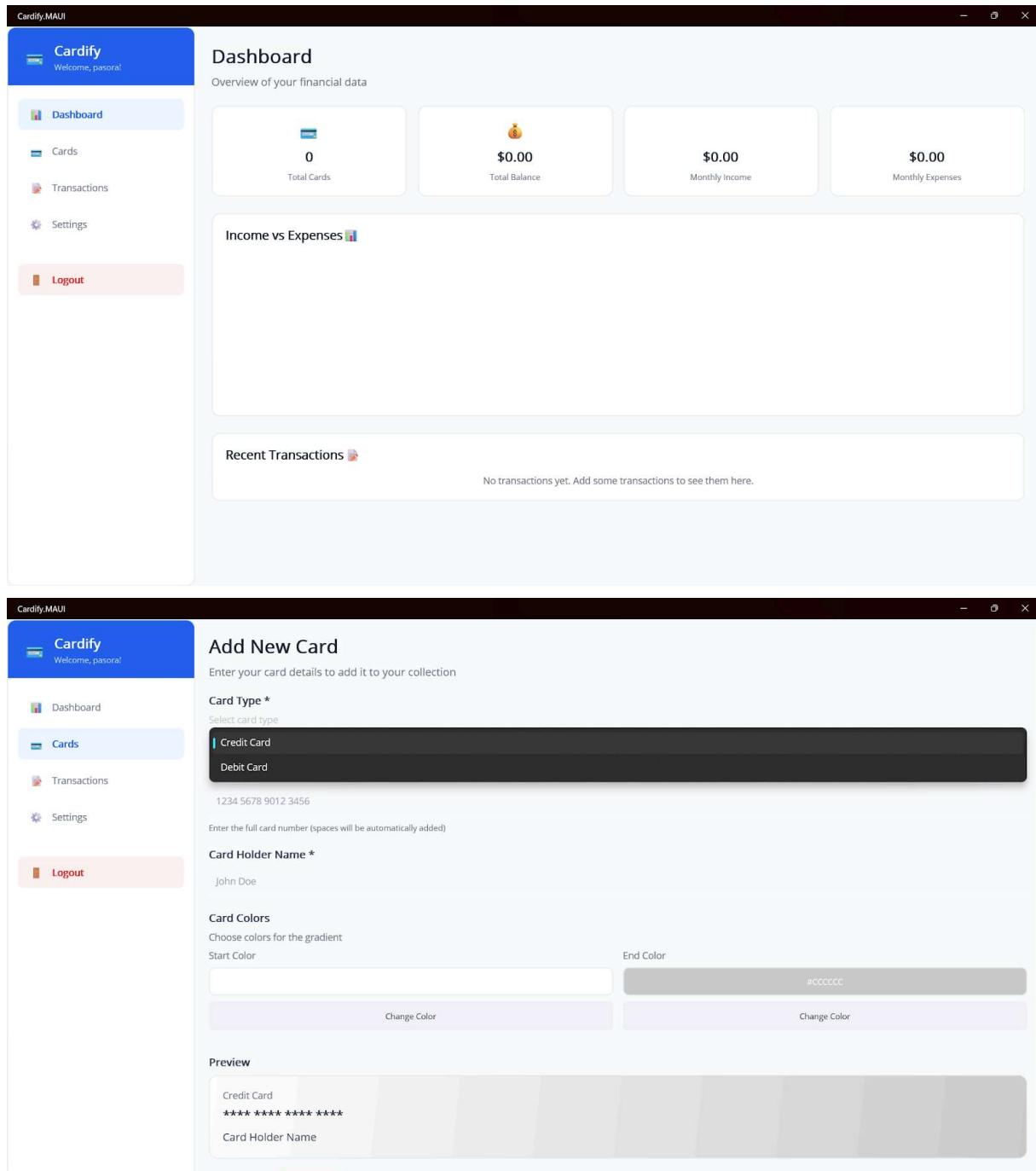
```
Cardify.Core > Services > TransactionService.cs
5  {
6    {
7      }
8
9      public async Task<IEnumerable<object>> GetUserTransactionsAsync(int userId, int? cardId, string? type, DateTime? fromDate, DateTime? toDate)
10     {
11       var query = _context.Transactions.Where(t => t.UserId == userId);
12
13       // Apply filters
14       if (cardId.HasValue)
15         query = query.Where(t => t.CardId == cardId.Value);
16       if (!string.IsNullOrWhiteSpace(type))
17         query = query.Where(t => t.Type == type);
18       if (fromDate.HasValue)
19         query = query.Where(t => t.Date >= fromDate.Value);
20       if (toDate.HasValue)
21         query = query.Where(t => t.Date <= toDate.Value);
22
23       var transactions = await query
24         .Include(t => t.Card)
25         .OrderByDescending(t => t.Date)
26         .Select(t => new
27         {
28           t.Id,
29           t.Name,
30           t.Date,
31           t.Amount,
32           t.Type,
33           Card = new
34             {
35               t.Card!.Id,
36               t.Card.CardType,
37               t.Card.LastFourDigits,
38               t.Card.CardHolderName
39             },
40             t.CreatedAt
41         })
42
43       .ToListAsync();
44     }
45   }
46 }
```

```
Cardify.MAUI > Views > AddCardView.xaml.cs
7  {
8    {
9      }
10
11      private void OnCardTypeChanged(object sender, EventArgs e)
12      {
13        if (CardTypePicker.SelectedItem != null)
14        {
15          _selectedCardType = CardTypePicker.SelectedItem.ToString() ?? string.Empty;
16          UpdatePreview();
17          ValidateForm();
18        }
19      }
20
21      private void OnCardNumberChanged(object sender, TextChangedEventArgs e)
22      {
23        var input = e.NewTextValue ?? string.Empty;
24
25        // Remove all non-digit characters
26        var digitsOnly = new string(input.Where(char.IsDigit).ToArray());
27
28        // Limit to 16 digits
29        if (digitsOnly.Length > 16)
30        {
31          digitsOnly = digitsOnly.Substring(0, 16);
32        }
33
34        // Format with spaces every 4 digits
35        var formatted = FormatCardNumber(digitsOnly);
36
37        // Update the entry if the formatted text is different
38        if (formatted != input)
39        {
40          CardNumberEntry.Text = formatted;
41          CardNumberEntryCursorPosition = formatted.Length;
42        }
43
44        cardNumber = digitsOnly; // Store only digits for validation
45      }
46    }
47  }
```

Projede LINQ sorguları etkin bir şekilde kullanılmaktadır. Özellikle Entity Framework Core ile gerçekleştirilen veritabanı işlemlerinde LINQ, servis katmanında veri filtreleme, projeksiyon, sıralama ve ilişkili tabloların dahil edilmesi işlemleri için tercih edilmiştir. Örneğin, CardService ve TransactionService sınıflarında Where, Select, Include, OrderBy gibi LINQ metotları kullanılarak kapsamlı sorgular oluşturulmuştur.

Ayrıca, veri bütünlüğü ve varlık kontrolü amacıyla FirstOrDefault, Any gibi metodlarla kayıtların varlığı denetlenmiştir. Bu sayede veritabanı erişimi hem okunabilirlik hem de performans açısından optimize edilmiştir. Projenin veri işleme süreçlerinde LINQ temel ve vazgeçilmez bir bileşen olarak yer almaktadır.

25- Proje çalışıyor mu? Evet, çalışıyor.



The image consists of two screenshots of the Cardify.MAUI application. The top screenshot shows the 'Dashboard' screen, which includes a sidebar with 'Dashboard', 'Cards', 'Transactions', 'Settings', and 'Logout'. The main area displays four cards: 'Total Cards' (0), 'Total Balance' (\$0.00), 'Monthly Income' (\$0.00), and 'Monthly Expenses' (\$0.00). Below these is a section titled 'Income vs Expenses' and then 'Recent Transactions' with a note 'No transactions yet. Add some transactions to see them here.' The bottom screenshot shows the 'Add New Card' screen, also with a sidebar. This screen has fields for 'Card Type *' (with 'Credit Card' selected and 'Debit Card' as an option), 'Card Number' (1234 5678 9012 3456), 'Card Holder Name *' (John Doe), and 'Card Colors' for a gradient. A preview section shows a sample card with the number and holder name.

Cardify.MAUI uygulaması başarıyla derlenmiş ve çalıştırılmıştır.

Giriş ekranı üzerinden geçerli bilgilerle yapılan login işlemi sonrası, kullanıcı başarılı bir şekilde uygulamanın ana ekranına yönlendirilmiştir.

Giriş sonrası açılan bu ekran, uygulamanın çalıştığını, sayfalar arası geçişin sorunsuz olduğunu ve kullanıcı doğrulama mantığının uygulandığını göstermektedir.

Bu durum, projenin temel işlevsellüğünün yerinde olduğunu açıkça kanıtlamaktadır.

26- Proje konusuna göre, olması gereken minimum işlemler yapılabiliyor mu? Evet, yapabiliyor.

The screenshot shows two code editor windows side-by-side. The left window displays `TransactionUpdateDto.cs` from the `Cardify.MAUI` project. The right window displays `Program.cs` from the `Cardify.API` project.

```
Cardify.MAUI > Models > TransactionUpdateDto.cs
1  namespace Cardify.MAUI.Models;
2
3  public class TransactionUpdateDto
4  {
5      public string? Name { get; set; }
6      public DateTime? Date { get; set; }
7      public decimal? Amount { get; set; }
8      public string? Type { get; set; } // income, expense, debt, etc.
9      public int? CardId { get; set; }
10     public string? Category { get; set; }
11 }
```

```
Cardify.API > Program.cs
1  using Cardify.Core.Models;
2  using Cardify.Core.Services;
3  using Microsoft.EntityFrameworkCore;
4
5  var builder = WebApplication.CreateBuilder(args);
6
7
8
9  builder.Services.AddOpenApi();
10 builder.Services.AddDbContext<CardifyDbContext>(options =>
11     options.UseSqlServer(
12         builder.Configuration.GetConnectionString("DefaultConnection"),
13         b => b.MigrationsAssembly("Cardify.API")
14     ));
15 builder.Services.AddEndpointsApiExplorer();
16 builder.Services.AddSwaggerGen();
17
18 // Register services
19 builder.Services.AddScoped<IUserService, UserService>();
20 builder.Services.AddScoped<ICardService, CardService>();
21 builder.Services.AddScoped<ITransactionService, TransactionService>();
22
23 var app = builder.Build();
24
25 // Configure the HTTP request pipeline.
26 if (app.Environment.IsDevelopment())
27 {
28     app.MapOpenApi();
29 }
30
31 app.UseSwagger();
32 app.UseSwaggerUI();
33
34 app.UseHttpsRedirection();
35
36 app.MapPost("/api/users/register", async (UserCreateDto dto, IUserService userService) =>
37 {
38     var result = await userService.RegisterUserAsync(dto);
39     if (!result)
```

```

        _context.Transactions.Add(transaction);
        await _context.SaveChangesAsync();

        return transaction.Id;
    }

    public async Task<bool> UpdateTransactionAsync(int id, TransactionUpdateDto dto, int userId)
    {
        var transaction = await _context.Transactions.FirstOrDefaultAsync(t => t.Id == id && t.UserId == userId);
        if (transaction == null)
            return false;

        // Update only provided fields
        if (!string.IsNullOrWhiteSpace(dto.Name))
            transaction.Name = dto.Name;
        if (dto.Date.HasValue)
            transaction.Date = dto.Date.Value;
        if (dto.Amount.HasValue && dto.Amount.Value > 0)
            transaction.Amount = dto.Amount.Value;
        if (!string.IsNullOrWhiteSpace(dto.Type))
            transaction.Type = dto.Type;

        // If CardId is being updated, validate it belongs to user
        if (dto.CardId.HasValue && dto.CardId.Value != transaction.CardId)

        // Password Change Endpoint
        app.MapPost("/api/users/change-password", async [PasswordChangeDto] dto, int userId, IUserService userService] =>
    {
        var result = await userService.ChangePasswordAsync(userId, dto);
        if (!result)
            return Results.BadRequest("Password change failed. Please check your current password.");

        return Results.Ok(new { message = "Password changed successfully." });
    });

    app.Run();
}

// User Logout Endpoint
app.MapPost("/api/users/logout", async (int userId, IUserService userService) =>
{
    var result = await userService.LogoutUserAsync(userId);
    if (!result)
        return Results.BadRequest("User not found.");

    return Results.Ok(new { message = "Logout successful." });
});

```

```
public async Task<IEnumerable<object>> GetUserCardsAsync(int userId)
{
    var cards = await _context.Cards
        .Where(c => c.UserId == userId)
        .Select(c => new
    {
        c.Id,
        c.CardType,
        c.CardNumber,
        c.LastFourDigits,
        c.CardHolderName,
        c.CardColorStart,
        c.CardColorEnd,
        c.CreatedAt
    })
    .ToListAsync();

    return cards;
}
```

Projede kişisel bütçe yönetimi uygulaması için gereken temel işlemler başarıyla gerçekleştirilemiştir.

Kullanıcı kayıt ve giriş işlemleri API üzerinde tanımlanmış, ilgili DTO ve servis metotları ile desteklenmiştir.

Kart yönetimi kapsamında kart ekleme, listeleme, güncelleme ve silme işlemleri API ve MAUI arayüzü ile sağlanmaktadır. Benzer şekilde, işlem yönetimi için de gerekli CRUD işlemleri eksiksiz uygulanmıştır.

Şifre değiştirme ve çıkış yapma fonksiyonları da kullanıcı deneyimini tamamlamaktadır.

Veri doğrulama hem backend hem frontend tarafında yapılmakta, filtreleme ve listeleme işlemleri LINQ sorguları ve CollectionView bileşeniyle desteklenmektedir.

Ayrıca, oluşturma ve güncelleme işlemlerinde yapılan değişiklikler kayıt altına alınarak izlenebilirlik sağlanmıştır.

Tüm bu özellikler, projenin amaçladığı kişisel bütçe yönetimi işlevsellliğini tam olarak karşıladığı göstermektedir.

27- C# isimlendirme kurallarına uyulmuş mu? Evet, uyulmuş.

The screenshot shows two code files in a dark-themed code editor:

CardService.cs

```
1  using Cardify.Core.Models;
2  using Microsoft.EntityFrameworkCore;
3
4  namespace Cardify.Core.Services
5  {
6      public class CardService : ICardService
7      {
8          private readonly CardifyDbContext _context;
9
10         public CardService(CardifyDbContext context)
11         {
12             _context = context;
13         }
14
15         public async Task<IEnumerable<object>> GetUserCardsAsync(int userId)
16         {
17             var cards = await _context.Cards
18                 .Where(c => c.UserId == userId)
19                 .Select(c => new
20                 {
21                     c.Id,
22                     c.CardType,
23                     c.CardNumber,
24                     c.LastFourDigits,
25                     c.CardHolderName,
26                     c.CardColorStart,
27                     c.CardColorEnd,
28                     c.CreatedAt
29                 })
30                 .ToListAsync();
31
32             return cards;
33         }
34
35         public async Task<object?> GetCardByIdAsync(int id, int userId)
36     }
```

Program.cs

```
// Register services
builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<ICardService, CardService>();
builder.Services.AddScoped<ITransactionService, TransactionService>();
```

```

public async Task<IEnumerable<object>> GetUserCardsAsync(int userId)
{
    var cards = await _context.Cards
        .Where(c => c.UserId == userId)
        .Select(c => new
    {
        c.Id,
        c.CardType,
        c.CardNumber,
        c.LastFourDigits,
        c.CardHolderName,
        c.CardColorStart,
        c.CardColorEnd,
        c.CreatedAt
    })
    .ToListAsync();

    return cards;
}

public async Task<object?> GetCardByIdAsync(int id, int userId)
{

```

```

Cardify.Core > Models > TransactionUpdateDto.cs > COLLECTION
1  using System.ComponentModel.DataAnnotations;
2
3  namespace Cardify.Core.Models
4  {
5      public class TransactionUpdateDto
6      {
7          [StringLength(100)]
8          public string? Name { get; set; }
9
10         public DateTime? Date { get; set; }
11
12         [Range(0.01, double.MaxValue)]
13         public decimal? Amount { get; set; }
14
15         [StringLength(20)]
16         public string? Type { get; set; } // income, expense, debt, etc.
17
18         [StringLength(30)]
19         public string? Category { get; set; }
20
21         public int? cardId { get; set; }
22     }
23 }
```

C# isimlendirme kurallarına genel olarak uyulmuştur.

Sınıf ve arayüz isimlendirmeleri PascalCase formatında yazılmış, arayüzler İ harfi ile başlamıştır. Metot ve değişken isimleri camelCase ya da PascalCase standartlarına uygun şekilde kullanılmıştır. Bu düzen, kodun okunabilirliğini ve bakımını kolaylaştırmaktadır.

