AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

Dokumentacja

Autor:
Michał Bernardy
Adrian Gargula

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań

Aplikacja mobilna **CountYourDollars** w skrócie "**CYD**"ma oferować użytkownikom łatwą i szybką możliwość zarządzania i monitorowania miesięcznych wydatków użytkwnika. Narzędzie jest wyposażone w wiele funkcji takich jak logowanie, dokumentowanie zakupów, miesięczne podsumowanie wydatków oraz powiadomienia. Dzięki prostemu i intuicyjnemu interfejsowi, aplikacja jest przyjazna dla wszystkich użytkowników, w tym osób starszych, którzy mogą korzystać z niej bez obaw o trudności w obsłudze. **CountYourDollars** to kompleksowe rozwiązanie, które wspiera użytkowników w efektywnym zarządzaniu finansami osobistymi dzięki czemu mogą oni zapomnieć o magicznie znikajacych pieniądzach.

1.1. Konto

Aby skorzystać z pełnych możliwości aplikacji CountYourDollars, użytkownik musi najpierw założyć konto. Proces rejestracji jest prosty i intuicyjny, co pozwala na szybkie rozpoczęcie korzystania z narzędzia. Utworzenie konta gwarantuje bezpieczeństwo danych użytkownika, zapewniając, że wszystkie informacje finansowe oraz osobiste są odpowiednio chronione.

1.2. Logowanie biometryczne

CountYourDollars zapewnia najwyższy poziom bezpieczeństwa, umożliwiając użytkownikom logowanie za pomocą danych biometrycznych, takich jak odciski palców. To innowacyjne rozwiązanie gwarantuje, że dostęp do wrażliwych informacji finansowych jest ściśle ograniczony do uprawnionych użytkowników, co znacząco minimalizuje ryzyko nieautoryzowanego dostępu. Wprowadzenie logowania biometrycznego nie tylko podnosi standardy bezpieczeństwa, ale także znacząco zwiększa wygodę korzystania z aplikacji. Dzięki szybkiej i intuicyjnej metodzie uwierzytelniania użytkownicy mogą łatwo i bezpiecznie uzyskać dostęp do swojego konta, eliminując konieczność zapamiętywania skomplikowanych haseł.

1.3. Dokumentacja zakupów

Aplikacja będzie wymagała od użytkowników dołączenia dokumentu potwierdzającego wydatek. Dzięki temu każdy zapis będzie dokładnie udokumentowany, co pozwoli na lepszą kontrolę finansów oraz umożliwi łatwiejsze śledzenie wydatków. Użytkownicy będą mieli możliwość załączania zdjęć paragonów lub innych dowodów zakupu, co zapewni pełną transparentność i wiarygodność w zarządzaniu osobistymi finansami.

1.4. Powiadomienia

Oprogramowanie ma wbudowany system powiadomień push, który na bieżąco informuje użytkowników o zbliżaniu się do indywidualnie ustalonego limitu wydatków. Dzięki możliwości dostosowania limitu do własnych potrzeb, użytkownicy mogą efektywnie zarządzać swoim budżetem, a system powiadomień przypomina im o nadchodzących ograniczeniach. W ten sposób **CountYourDollars** nie tylko pomaga w monitorowaniu wydatków, ale także wspiera w osiąganiu finansowych celów, oferując na czas przypomnienia, które pozwalają na lepszą kontrolę nad osobistymi finansami.

1.5. Funkcjonalność aplikacji

Na początku korzystania z aplikacji użytkownik wprowadza swój miesięczny dochód, co stanowi fundament efektywnego zarządzania finansami. Interfejs aplikacji został zaprojektowany w sposób intuicyjny, umożliwiając łatwe przechodzenie między poszczególnymi miesiącami. Po kliknięciu na konkretny miesiąc, użytkownicy zyskują pełny wgląd w swoje wydatki, co pozwala na szczegółową analizę finansową. Na zakończenie każdego miesiąca aplikacja generuje podsumowanie wydatków, prezentując użytkownikowi informacje o tym, ile zostało wydane na różne kategorie, takie jak samochód, jedzenie, rozrywka, czynsz i inne. Dzięki temu użytkownicy mają możliwość skutecznego monitorowania swoich wydatków oraz podejmowania świadomych decyzji finansowych, co przyczynia się do lepszego zarządzania budżetem.

1.6. Tryb offline

CountYourDollars została zaprojektowana z myślą o zapewnieniu użytkownikom maksymalnej wygody i dostępności, nawet w sytuacjach, gdy nie mają połączenia internetowego. Funkcjonalność trybu offline oferuje użytkownikom podgląd do wydatków w obecenym miesiącu.

1.7. Baza danych

Aplikacja korzysta z chmurowej bazy danych Firebase, co pozwala na skuteczne przechowywanie, zarządzanie i dostęp do różnorodnych informacji związanych z finansami użytkowników. Firebase jest wszechstronnym rozwiązaniem, które oferuje wiele funkcji niezbędnych do prawidłowego funkcjonowania aplikacji.

1.8. Design logo

Rys. 1.1. Logo

Logo przedstawia symboliczny emblemat w kształcie okręgu z wyraźnym napisem "CYD"w jego wnętrzu. Dominująca kolorystyka w odcieniach niebieskiego nadaje mu nowoczesny i profesjonalny wygląd, co jest często kojarzone z technologią i finansami. Napis "COUNTYOURDOLLARS"poniżej skrótu to nazwa aplikacji która już na wstępie sugeruje że będzie ona służyć do zarządzania finansami. Cały projekt jest schludny i estetyczny, wykorzystując prostą, ale elegancką typografię.

2. Określenie wymagań szczegółowych

2.1. Specyfikacja techniczna .NET MAUI

- Język programowania: .NET MAUI pozwala programować w języku C#.
- Wspierane platformy: .NET MAUI umożliwia tworzenie aplikacji na platformy iOS, Android, macOS oraz Windows.
- Jednolity kod interfejsu: Dzięki .NET MAUI można tworzyć interfejsy użytkownika za pomocą wspólnego kodu dla wielu platform.
- Natywny dostęp: Pozwala na dostęp do natywnych API oraz funkcji systemowych każdej z platform przy użyciu specyficznych narzędzi i komponentów.
- Wspólne API: .NET MAUI oferuje dostęp do wspólnego zestawu API, co pozwala na udostępnienie logiki biznesowej między platformami.
- Narzędzia IDE: Do pracy z .NET MAUI można wykorzystać środowisko Visual Studio 2022 oraz Visual Studio for Mac.
- Biblioteki: .NET MAUI integruje się z ekosystemem .NET, umożliwiając dostęp do szerokiej gamy bibliotek oraz narzędzi, takich jak NuGet.
- Testowanie: Możliwość testowania aplikacji zarówno na rzeczywistych urządzeniach, jak i przy użyciu emulatorów oraz narzędzi takich jak .NET MAUI Test. [1]

2.2. Specyfikacja techniczna Aplikacji CYD

Aplikacja CYD to kompleksowe narzędzie stworzone w technologii .NET MAUI, dostępne na platformy Android, które ma na celu ułatwienie życia oraz dostarczenie istotnych informacji związanych z wydatkami. Aplikacja będzie wyposażona w funkcje powiadomień push, logowanie biometryczne, kategoryzowanie wydatków, podsumowywanie miesięczne oraz dokumentowanie wydatków poprzez zdjęcie. Poniżej przedstawiamy bardziej szczegółowe wymagania dotyczące każdej z tych funkcji

[1] Programowanie urządzeń mobilnych [**wyklady**]

2.3. Funkcje i Moduły

2.3.1. Logowanie i Weryfikacja

- Aplikacja CYD umożliwia użytkownikom zalogowanie się za pomocą unikatowego identyfikatora oraz hasła, a także wykorzystanie zaawansowanych mechanizmów logowania biometrycznego, takich jak rozpoznawanie odcisków palców .

- W trakcie procesu logowania następuje weryfikacja tożsamości użytkownika, co zapewnia bezpieczeństwo danych oraz dostęp tylko do odpowiednich zasobów

2.3.2. Powiadomienia push

- Funkcja "Powiadomienia push"powiadamia użytkowników o kończącym się budżecie w danym miesiącu.

- Powiadomienia są wysyłane po przekroczeniu danych limitów ustawionych przez użytkownika.

2.3.3. Dokumentacja wydatków

- Każdy wydatek może być potwierdzany zdjęciem paragonu.

- Dzięki temu użytkownicy mogą dokładnie uwieczniac swoje wydatki nie musząc zbierać paragonów.

2.3.4. Kategorie wydatków

- Kategoryzowanie pozwala na podział obciążeń, użytkownik może sprawdzać jakie produkty pochłaniają daną ilość pieniędzy w skali miesiąca.

2.3.5. Miesięczne zestawienie

- Aplikacja dostarcza informacje na miesięcznego zestawienia wydatków.

- Dane do zestawienia są pobierane z bazy danych i prezentowane w tabeli z sumą wydatków podzieloną na kategorię.

2.3.6. Baza Danych

- Aplikacja korzysta z bazy danych do przechowywania informacji o uŻytkownikach, wydatkach i innych danych.
- Baza danych jest odpowiednio zabezpieczona i regularnie tworzone są jej kopie zapasowe, aby zapewnić integralność i dostępność danych.
- W aplikacji będzie wykorzystana baza danych Google Firebase. Firebase oferuje bazę danych typu NoSQL o nazwie Firebase Realtime Database oraz Firebase Firestore. Oba rozwiązania są skalowalne i oferują możliwość współpracy w czasie rzeczywistym.[2].

[2]Google Firebase [1].

3. Projektowanie

3.1. Przygotowanie Narzędzi do Tworzenia Aplikacji "CYD"w .NET MAUI

Tworzenie aplikacji mobilnych jest zadaniem wymagającym odpowiednich narzędzi i środowiska programistycznego. W przypadku aplikacji "CYD"w języku .NET MAUI, musimy odpowiednio skonfigurować narzędzia, aby móc sprawnie rozwijać projekt. W poniższym opisie przedstawiamy kroki przygotowania narzędzi, aby ułatwić proces tworzenia aplikacji dla platformy Android.

3.1.1. Środowisko programistyczne .NET MAUI

Rozpoczynamy od zainstalowania środowiska .NET MAUI na naszym komputerze. .NET MAUI to nowoczesne narzędzie umożliwiające tworzenie aplikacji wieloplatformowych z wykorzystaniem języka C oraz jednolitego kodu dla iOS, Androida, macOS i Windows.

3.1.2. Visual Studio

Następnie, jeśli nie mamy jeszcze zainstalowanego Visual Studio, pobieramy i instalujemy najnowszą wersję programu. Podczas instalacji zaleca się zaznaczenie opcji "Mobile development with .NET", co pozwoli na aktywację wsparcia dla .NET MAUI. To środowisko programistyczne dostarcza narzędzi do projektowania interfejsu, edycji kodu, debugowania i testowania aplikacji.

3.1.3. Git

Git to system kontroli wersji. Git to system, który pozwala programistom zapisywać wszystkie zmiany w pisanym kodzie – w taki sposób, aby niczego nie stracili. Programista w każdej chwili może wrócić do poprzedniej wersji, jeżeli zajdzie taka potrzeba. Dzięki podglądowi poprzednich wersji można prześledzić, jak program się rozwijał, cofnąć się, odzyskać przypadkowo utracone zmiany czy powrócić do wcześniejszych pomysłów. Z systemu kontroli wersji Git korzystają zarówno programiści w korporacyjnych zespołach, jak i ci pracujący samodzielnie. Warto wyrobić sobie ten dobry nawyk korzystania z systemu kontroli wersji – nie raz okaże się, że uratuje nam to skórę i zapobiegnie marnowaniu czasu[3].

[3]CODECOOL www[2].

3.1.4. Emulator

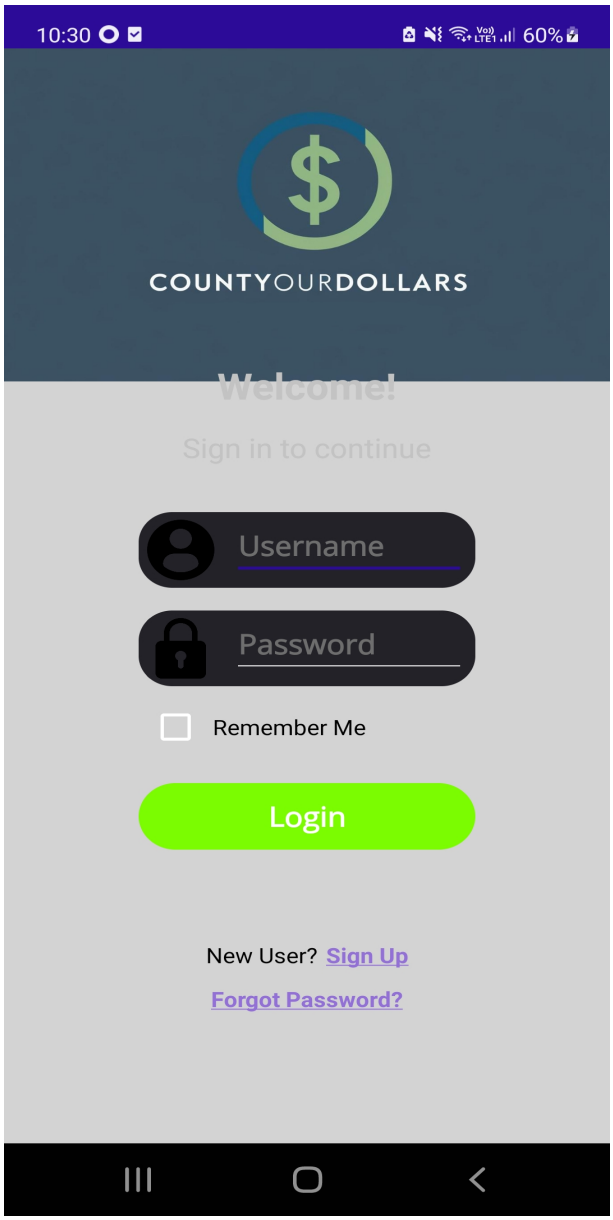
Warto także zadbać o narzędzia potrzebne podczas tworzenia aplikacji .NET MAUI. Przydatnym narzędziem jest emulator Androida do testowania aplikacji na różnych urządzeniach oraz fizyczne urządzenie Android.

3.1.5. Tworzenie Nowego Projektu .NET MAUI

Po zainstalowaniu narzędzi i skonfigurowaniu środowiska, możemy przejść do tworzenia nowego projektu .NET MAUI. W Visual Studio wybieramy opcję utworzenia nowego projektu .NET MAUI App. Możemy wybrać odpowiedni szablon aplikacji, który najlepiej odpowiada naszym potrzebom, na przykład Blank App, .NET MAUI Blazor App lub inne dostępne opcje. Szablony te dostarczają podstawową strukturę aplikacji i pozwalają na łatwe dodawanie nowych widoków i funkcjonalności.

3.2. Szkic layouta

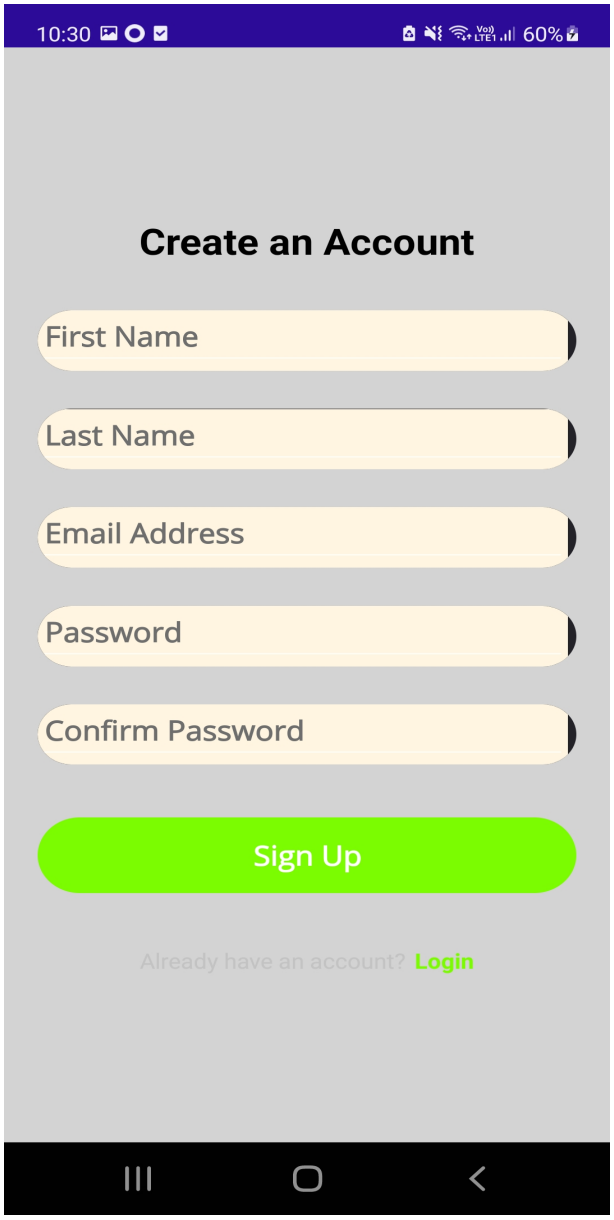
3.2.1. Ekran Logowania



Rys. 3.1. Ekran logowania

Na rysunku.3.1(s.12). Użytkownik zobaczy ekran po uruchomieniu aplikacji. Na górze ekranu znajduje się logo aplikacji. Poniżej logotypu jest formularz logowania, z polami na wprowadzenie loginu użytkownika i hasła oraz przyciskiem "Login"oraz "Sign Up"czyli logowanie oraz rejerstrowanie się jest także opcja "Forgot Password?"czyli odzyskiwanie hasła. Opcja "Remember me"pozwala na zapamiętywanie użytkownika.

3.2.2. Ekran Rejestrowania



Rys. 3.2. Ekran Sign Up

Na rysunku.3.2(s.13). Użytkownik zobaczy ekran po kliknięciu sing up na stronie logowania. Na górze ekranu znajduje się informacja co użytkownik robi w danym momencie . Poniżej jest formularz rejestrowania, z niezbędnymi do utworzenia konta danymi. Gdy wypełnimy formularz możemy kliknac "Sign Up"dzięki czemu utworzymy konto.

4. Implementacja

4.1. Klasa AppShell

```
using CommunityToolkit.Maui.Alerts;
using CommunityToolkit.Maui.Core;
using Font = Microsoft.Maui.Font;
using Firebase.Auth;
namespace CYD
{
    public partial class AppShell : Shell
    {
        private FirebaseAuthService _authService;

        public AppShell()
        {
            InitializeComponent();
            _authService = new FirebaseAuthService();
            var currentTheme = Application.Current!.UserAppTheme;
            ThemeSegmentedControl.SelectedIndex = currentTheme ==
    AppTheme.Light ? 0 : 1;
        }

        public static async Task DisplaySnackbarAsync(string
    message)
        {
            CancellationTokenSource cancellationTokenSource = new
    CancellationTokenSource();

            var snackbarOptions = new SnackbarOptions
            {
                BackgroundColor = Color.FromArgb("#FF3300"),
                TextColor = Colors.White,
                ActionButtonTextColor = Colors.Yellow,
                CornerRadius = new CornerRadius(0),
                Font = Font.SystemFontOfSize(18),
                ActionButtonFont = Font.SystemFontOfSize(14)
            };

            var snackbar = Snackbar.Make(message, visualOptions:
    snackbarOptions);

            await snackbar.Show(cancellationTokenSource.Token);
        }

```

```
38          public static async Task DisplayToastAsync(string message)
39          {
40              // Toast is currently not working in MCT on Windows
41              if (OperatingSystem.IsWindows())
42                  return;
43
44              var toast = Toast.Make(message, textSize: 18);
45
46              var cts = new CancellationTokenSource(TimeSpan.
    FromSeconds(5));
47              await toast.Show(cts.Token);
48          }
49
50          private void SfSegmentedControl_SelectionChanged(object
    sender, Syncfusion.Maui.Toolkit.SegmentedControl.
    SelectionChangedEventArgs e)
51          {
52              Application.Current!.UserAppTheme = e.NewIndex == 0 ?
    AppTheme.Light : AppTheme.Dark;
53          }
54
55          private async void OnLogoutButtonClicked(object sender,
    EventArgs e)
56          {
57              await _authService.LogoutAsync();
58
59              // Przej Żcie do ekranu logowania po wylogowaniu
60              Application.Current.MainPage = new NavigationPage(new
    LoginPage());
61              SecureStorage.Remove("user_email");
62              SecureStorage.Remove("user_uid");
63          }
64      }
65  }
```

Listing 1. Klasa AppShell

4.1.1. Wyjaśnienie kodu

- **Deklaracja klasy AppShell** na **Listing 1** (s. 14):

```
1      public partial class AppShell : Shell
2
```

Listing 2. Deklaracja klasy

Klasa **AppShell** dziedziczy po klasie **Shell** z frameworka .NET MAUI i jest odpowiedzialna za główny interfejs aplikacji. Zawiera ona funkcjonalności związane z autoryzacją użytkownika oraz obsługą interfejsu użytkownika.

- **Konstruktor klasy AppShell()** na **Listing 3** (s. 16):

```
1     public AppShell()
2     {
3         InitializeComponent();
4         _authService = new FirebaseAuthService();
5         var currentTheme = Application.Current!.UserAppTheme;
6         ThemeSegmentedControl.SelectedIndex = currentTheme ==
    AppTheme.Light ? 0 : 1;
7     }
8
```

Listing 3. Konstruktor klasy

Konstruktor klasy **AppShell** inicjalizuje komponenty strony przy pomocy **InitializeComponent()** oraz ustawia bieżący motyw aplikacji (jasny lub ciemny) na podstawie ustawień systemowych. Używa do tego obiektu **UserAppTheme**.

- **Metoda DisplaySnackbarAsync** na **Listing 4** (s. 16):

```
1     public static async Task DisplaySnackbarAsync(string
    message)
2
```

Listing 4. Metoda DisplaySnackbarAsync

Ta statyczna metoda wyświetla powiadomienie typu snackbar z określonym komunikatem. Można dostosować wygląd snackbar, np. kolor tła, kolor tekstu, czcionkę itp. Powiadomienie jest wyświetlane przez określony czas, aż użytkownik je zamknie lub upłynie czas oczekiwania.

- **Metoda DisplayToastAsync** na **Listing 5** (s. 16):

```
1     public static async Task DisplayToastAsync(string message)
2
```

Listing 5. Metoda DisplayToastAsync

Metoda ta wyświetla powiadomienie typu toast, które pojawia się na krótki czas na ekranie i znika automatycznie. Jeśli aplikacja działa na systemie Windows, ta funkcjonalność jest wyłączona, ponieważ obecnie nie działa ona poprawnie na tej platformie.

- **Zmiana motywu aplikacji** na **Listing 6** (s. 17):

```
    private void SfSegmentedControl_SelectionChanged(object
sender, Syncfusion.Maui.Toolkit.SegmentedControl.
SelectionChangedEventArgs e)
    {
        Application.Current!.UserAppTheme = e.NewIndex == 0 ?
AppTheme.Light : AppTheme.Dark;
    }

```

Listing 6. Zmiana motywu aplikacji

Ta metoda jest wywoływana po zmianie wyboru w kontrolce **SegmentedControl**. Na podstawie nowego wyboru (indeksu) motyw aplikacji jest zmieniany na jasny lub ciemny.

- **Metoda OnLogoutButtonClicked** na **Listing 7** (s. 17):

```
    private async void OnLogoutButtonClicked(object sender,
EventArgs e)
    {
        await _authService.LogoutAsync();

        // Przejście do ekranu logowania po wylogowaniu
        Application.Current.MainPage = new NavigationPage(new
LoginPage());
        SecureStorage.Remove("user_email");
        SecureStorage.Remove("user_uid");
    }

```

Listing 7. Metoda OnLogoutButtonClicked

Metoda **OnLogoutButtonClicked** jest wywoływana po kliknięciu przycisku wylogowania. Używa ona metody **LogoutAsync** z klasy **FirebaseAuthService** do wylogowania użytkownika. Następnie, użytkownik zostaje przekierowany do ekranu logowania, a dane dotyczące użytkownika są usuwane z **SecureStorage**.

4.2. XAML Strony Logowania

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="CYD.Pages.LoginPage"
```

```
            Title="LoginPage"
            NavigationPage.HasNavigationBar="False">
    <Grid BackgroundColor="LightGray">
        <Grid.RowDefinitions>
            <RowDefinition Height="220"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="40"/>
        </Grid.RowDefinitions>

        <StackLayout Grid.Row="0" HeightRequest="335">
            <Image Source="Resources/Images/logo.png"
                   HorizontalOptions="FillAndExpand"
                   VerticalOptions="FillAndExpand"
                   Aspect="AspectFill"/>
        </StackLayout>

        <Grid RowSpacing="5" Grid.Row="1" Margin="20,20,20,0">
            <Grid.RowDefinitions>
                <RowDefinition Height="*"/>
                <RowDefinition Height="50"/>
                <RowDefinition Height="50"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="40"/>
                <RowDefinition Height="40"/>
                <RowDefinition Height="*"/>
            </Grid.RowDefinitions>

            <Label Grid.Row="1" Text="Welcome!"
                   HorizontalOptions="Center"
                   FontSize="Title"
                   FontAttributes="Bold"
                   Padding="10"
                   Margin="0"/>
            <Label Grid.Row="2" Text="Sign in to continue"
                   HorizontalOptions="Center"
                   FontSize="Medium"/>

            <StackLayout Grid.Row="3" Orientation="Vertical"
HorizontalOptions="CenterAndExpand">
                <Border BackgroundColor="Transparent" Stroke="
LightGreen" Padding="0" HorizontalOptions="FillAndExpand">
                    <StackLayout Orientation="Horizontal"
HorizontalOptions="FillAndExpand" VerticalOptions="
CenterAndExpand" Spacing="5">
                        <Border BackgroundColor="SkyBlue"
```

```
    HeightRequest="40" WidthRequest="40" Padding="0" Margin="5">
46                          <Image Source="Resources/Images/
    profile_user.png" Aspect="AspectFill" Margin="0"/>
47                      </Border>
48                      <Entry x:Name="EmailEntry" Placeholder="
    Email" TextColor="Black" FontAttributes="Bold" VerticalOptions="
    Center"
49                             HorizontalOptions="FillAndExpand"
    HeightRequest="40" Margin="0,0,5,0"/>
50                  </StackLayout>
51              </Border>
52
53              <Border BackgroundColor="Transparent" Stroke="
    LightGreen" Margin="0,15,0,0" Padding="0" HorizontalOptions="
    FillAndExpand">
54                  <StackLayout Orientation="Horizontal"
    HorizontalOptions="FillAndExpand" VerticalOptions="
    CenterAndExpand" Spacing="5">
55                      <Border BackgroundColor="SkyBlue"
    HeightRequest="40" WidthRequest="40" StrokeShape="RoundRectangle
    " Padding="0" Margin="5">
56                          <Image Source="Resources/Images/
    password.png" Aspect="AspectFill" Margin="0"/>
57                      </Border>
58                      <Entry x:Name="PasswordEntry" Placeholder="
    Password" IsPassword="True" HorizontalOptions="FillAndExpand"
59                             VerticalOptions="Center" TextColor="
    Black" FontAttributes="Bold" HeightRequest="40" Margin="0,0,5,0"
    />
60                  </StackLayout>
61              </Border>
62
63              <StackLayout Orientation="Horizontal" Margin="
    0,5,0,0" Padding="0">
64                  <CheckBox IsChecked="False"/>
65                  <Label Text="Remember Me" TextColor="Black"
    FontSize="Small" VerticalTextAlignment="Center"
    HorizontalTextAlignment="Center"/>
66              </StackLayout>
67
68              <Button Text="Login" Clicked="OnLoginButtonClicked"
     BackgroundColor="LawnGreen" TextColor="White" FontAttributes="
    Bold"
69                      CornerRadius="30" WidthRequest="200" Margin
    ="0,15,0,0"/>
```

```
70
71                  <StackLayout Orientation="Horizontal"
    HorizontalOptions="Center" VerticalOptions="Center" Margin="
    0,60,0,0" Padding="10,0" Spacing="5">
72                      <Label Text="New User?" TextColor="Black"
    FontSize="Small" VerticalOptions="Center" HorizontalOptions="
    Start"/>
73                      <Label Text="Sign Up" TextColor="MediumPurple"
    FontAttributes="Bold" TextDecorations="Underline" FontSize="
    Small"
74                             VerticalOptions="Center"
    HorizontalOptions="Start">
75                          <Label.GestureRecognizers>
76                              <TapGestureRecognizer Tapped="
    OnSignUpTapped"/>
77                          </Label.GestureRecognizers>
78                      </Label>
79                  </StackLayout>
80
81                  <StackLayout Orientation="Horizontal"
    HorizontalOptions="Center" VerticalOptions="Center" Margin="
    0,10,0,0" Padding="10,0" Spacing="5">
82                      <Label Text="Forgot Password?" TextColor="
    MediumPurple" FontSize="Small" FontAttributes="Bold"
    TextDecorations="Underline"
83                             VerticalOptions="Center"
    HorizontalOptions="Center">
84                          <Label.GestureRecognizers>
85                              <TapGestureRecognizer Tapped="
    OnForgotPasswordTapped"/>
86                          </Label.GestureRecognizers>
87                      </Label>
88                  </StackLayout>
89
90              </StackLayout>
91          </Grid>
92      </Grid>
93  </ContentPage>
```

Listing 8. XAML Strony Logowania

4.2.1. Wyjaśnienie kodu XAML

- **Struktura Layoutu**: Strona logowania jest oparta na układzie **Grid**, który dzieli ekran na trzy główne części: nagłówek, sekcję logowania oraz stopkę.

  - Wiersz pierwszy (`Grid.Row="0"`) zawiera logo aplikacji.
  - Wiersz drugi (`Grid.Row="1"`) zawiera pola logowania oraz inne elementy interfejsu, takie jak przyciski i pola wyboru.
  - Wiersz trzeci (`Grid.Row="2"`) jest przeznaczony na przyciski oraz dodatkowe informacje o rejestracji i odzyskiwaniu hasła.

- **Logo Aplikacji**: Logo znajduje się w górnej części strony, a jego źródło to `Resources/Images/logo.png`. Jest ono wyświetlane w sposób responsywny, wypełniając dostępne miejsce, dzięki ustawieniu `HorizontalOptions="FillAndExpand"` oraz `VerticalOptions="FillAndExpand"`.

- **Formularz logowania**: Formularz składa się z dwóch głównych pól wejściowych: jedno dla adresu e-mail oraz drugie dla hasła. Każde z tych pól znajduje się w **StackLayout**, który zawiera również ikony (np. ikona użytkownika przy polu e-mail i ikona kłódki przy polu hasła).
  - `Entry` to pole wprowadzania danych, atrybut `IsPassword="True"` dla pola hasła zapewnia ukrycie tekstu.
  - `Border` wokół każdego pola wejściowego daje efekt konturu.

- **Opcje logowania i rejestracji**: Poza polami logowania, aplikacja oferuje możliwość zapamiętania danych użytkownika (z pomocą `CheckBox`), a także przyciski:
  - `Button` - przycisk logowania.
  - `Label` z gestem dotknięcia, który przekierowuje użytkownika do strony rejestracji (`OnSignUpTapped`).
  - `Label` z gestem dotknięcia, który umożliwia odzyskiwanie hasła (`OnForgotPasswordTap`

4.2.2. Funkcjonalność

- **OnLoginButtonClicked**: Metoda ta zostanie wywołana po kliknięciu przycisku logowania. Odpowiada za wykonanie logowania, weryfikację danych użytkownika i dalsze przekierowanie w aplikacji. - **OnSignUpTapped**: Akcja odpowiedzialna za przekierowanie do strony rejestracji. - **OnForgotPasswordTapped**: Akcja odpowiedzialna za przekierowanie do strony odzyskiwania hasła.

4.3. Implementacja klasy FirebaseAuthService

```
public class FirebaseAuthService
{
    private readonly FirebaseAuthProvider _authProvider;
    private readonly string _firebaseApiKey = "
   AIzaSyCsPbeDGIi0slQWhIcPr2hI0NShLn7dE7M";
    private FirebaseAuthLink _authLink;

    public FirebaseAuthService()
    {
        _authProvider = new FirebaseAuthProvider(new FirebaseConfig
   (_firebaseApiKey));
    }

    public async Task<string> SignInWithEmailAndPasswordAsync(
   string email, string password)
    {
        try
        {
            _authLink = await _authProvider.
   SignInWithEmailAndPasswordAsync(email, password);
            await SecureStorage.SetAsync("user_email", email);
            await SecureStorage.SetAsync("user_uid", _authLink.User
   .LocalId);
            string token = _authLink.FirebaseToken;
            return token;
        }
        catch (Exception ex)
        {
            return $"Error: {ex.Message}";
        }
    }
}
```

Listing 9. Klasa FirebaseAuthService

4.3.1. Wyjaśnienie kodu

- **Deklaracja klasy FirebaseAuthService** na **Listing 9** (s. 22):

```
    public class FirebaseAuthService

```

Listing 10. Deklaracja klasy

Klasa **FirebaseAuthService** odpowiada za autentykację użytkownika w Firebase przy użyciu emaila i hasła.

- **Konstruktor klasy FirebaseAuthService()** na **Listing 11** (s. 23):

```
1    public FirebaseAuthService()
2    {
3        _authProvider = new FirebaseAuthProvider(new
   FirebaseConfig(_firebaseApiKey));
4    }
5
```

Listing 11. Konstruktor klasy

Konstruktor tworzy obiekt **FirebaseAuthProvider**, który będzie używany do wykonywania operacji autoryzacji. Przekazuje mu klucz API Firebase, który pozwala na dostęp do usług Firebase.

- **Metoda SignInWithEmailAndPasswordAsync** na **Listing 12** (s. 23):

```
1    public async Task<string> SignInWithEmailAndPasswordAsync(
   string email, string password)
2
```

Listing 12. Metoda SignInWithEmailAndPasswordAsync

Metoda ta umożliwia użytkownikowi logowanie się przy użyciu adresu e-mail i hasła. Zwraca token autoryzacyjny, jeśli logowanie zakończy się powodzeniem.

- **Proces logowania** na **Listing 13** (s. 23):

```
1    try
2    {
3        _authLink = await _authProvider.
   SignInWithEmailAndPasswordAsync(email, password);
4        await SecureStorage.SetAsync("user_email", email);
5        await SecureStorage.SetAsync("user_uid", _authLink.User
   .LocalId);
6        string token = _authLink.FirebaseToken;
7        return token;
8    }
9    catch (Exception ex)
10   {
11       return $"Error: {ex.Message}";
12   }
13
```

Listing 13. Proces logowania

W tej części kodu:

- **SignInWithEmailAndPasswordAsync** jest wywoływana na obiekcie **_authProvider**, który próbuje zalogować użytkownika.
- Jeśli logowanie jest pomyślne, przechowujemy dane użytkownika (adres e-mail oraz unikalny identyfikator **user_uid**) w **SecureStorage**.
- Na końcu zwracamy token autoryzacyjny **_authLink.FirebaseToken**.

- **Obsługa błędów** na **Listing 14** (s. 24):

```
    catch (Exception ex)
    {
        return $"Error: {ex.Message}";
    }

```

Listing 14. Obsługa błędów

W przypadku wystąpienia błędu, zostanie on przechwycony przez blok **catch**, a metoda zwróci komunikat o błędzie.

4.4. Klasa SignUpPage

```
using Microsoft.Maui.Controls;

namespace CYD.Pages;

public partial class SignUpPage : ContentPage
{
    private FirebaseAuthService _authService;
    public SignUpPage()
    {
        InitializeComponent();
        _authService = new FirebaseAuthService();
    }

    private async void OnSignUpClicked(object sender, EventArgs e)
    {
        string firstName = FirstNameEntry.Text;
        string lastName = LastNameEntry.Text;
        string email = EmailEntry.Text;
        string password = PasswordEntry.Text;
        string confirmPassword = ConfirmPasswordEntry.Text;

        if (string.IsNullOrEmpty(firstName) || string.IsNullOrEmpty
   (lastName) ||
```

```
23            string.IsNullOrEmpty(email) || string.IsNullOrEmpty(
    password) ||
24            string.IsNullOrEmpty(confirmPassword))
25        {
26            await DisplayAlert("Error", "Please fill in all fields
    .", "OK");
27            return;
28        }
29
30        if (password != confirmPassword)
31        {
32            await DisplayAlert("Error", "Passwords do not match.",
    "OK");
33            return;
34        }
35
36        if (password.Length < 6)
37        {
38            await DisplayAlert("Error","Password must be at least 6
     signs!","OK");
39            return;
40        }
41
42        string result = await _authService.\textbf{
    RegisterWithEmailAndPasswordAsync}(email, password);
43
44        if (!string.IsNullOrEmpty(result) && result.StartsWith("
    Error"))
45        {
46            await DisplayAlert("Error", "Unable to register account
    ", "OK");
47            return;
48        }
49        else
50        {
51            await DisplayAlert("Registration Success", "Account
    created successfully!", "OK");
52            await Navigation.PopAsync();
53
54            EmailEntry.Text = string.Empty;
55            PasswordEntry.Text = string.Empty;
56            ConfirmPasswordEntry.Text = string.Empty;
57        }
58
59    }
```

```
60
61      private async void OnLoginTapped(object sender, EventArgs e)
62      {
63          await Navigation.PopAsync();
64      }
65  }
```

Listing 15. Klasa SignUpPage

4.4.1. Wyjaśnienie kodu

- **Deklaracja klasy SignUpPage** na **Listing 15** (s. 24):

```
1       public partial class SignUpPage : ContentPage
2
```

Listing 16. Deklaracja klasy

Klasa **SignUpPage** odpowiada za formularz rejestracji użytkownika w aplikacji. Dziedziczy po klasie **ContentPage**, co oznacza, że jest stroną w aplikacji opartej na technologii Xamarin (MAUI).

- **Konstruktor klasy SignUpPage()** na **Listing 17** (s. 26):

```
1       public SignUpPage()
2       {
3           InitializeComponent();
4           _authService = new FirebaseAuthService();
5       }
6
```

Listing 17. Konstruktor klasy

Konstruktor inicjalizuje komponenty strony przy pomocy **InitializeComponent()** oraz tworzy obiekt **FirebaseAuthService**, który będzie odpowiedzialny za logikę rejestracji użytkownika za pomocą Firebase.

- **Metoda OnSignUpClicked** na **Listing 18** (s. 26):

```
1       private async void OnSignUpClicked(object sender, EventArgs
        e)
2
```

Listing 18. Metoda OnSignUpClicked

Metoda ta jest wywoływana, gdy użytkownik kliknie przycisk rejestracji. Wykonuje ona szereg sprawdzeń:

- Sprawdza, czy wszystkie pola zostały wypełnione.
- Sprawdza, czy hasła się zgadzają.
- Sprawdza, czy hasło ma odpowiednią długość (przynajmniej 6 znaków).

- **Sprawdzanie i rejestracja użytkownika** na **Listing 19** (s. 27):

```
1    string result = await _authService.\textbf{
     RegisterWithEmailAndPasswordAsync}(email, password);
2
3    if (!string.IsNullOrEmpty(result) && result.StartsWith("
     Error"))
4    {
5        await DisplayAlert("Error", "Unable to register account
     ", "OK");
6        return;
7    }
8    else
9    {
10       await DisplayAlert("Registration Success", "Account
     created successfully!", "OK");
11       await Navigation.PopAsync();
12
13       EmailEntry.Text = string.Empty;
14       PasswordEntry.Text = string.Empty;
15       ConfirmPasswordEntry.Text = string.Empty;
16   }
17
```

Listing 19. Sprawdzanie i rejestracja użytkownika

Jeśli wszystkie dane są poprawne, wywoływana jest metoda **RegisterWithEmailAndPasswordAsync** z klasy **FirebaseAuthService**, która próbuje zarejestrować użytkownika. Jeśli rejestracja zakończy się niepowodzeniem, wyświetli się komunikat o błędzie. W przeciwnym razie użytkownik zostaje poinformowany o sukcesie i strona rejestracji zostaje zamknięta.

- **Metoda OnLoginTapped** na **Listing 20** (s. 27):

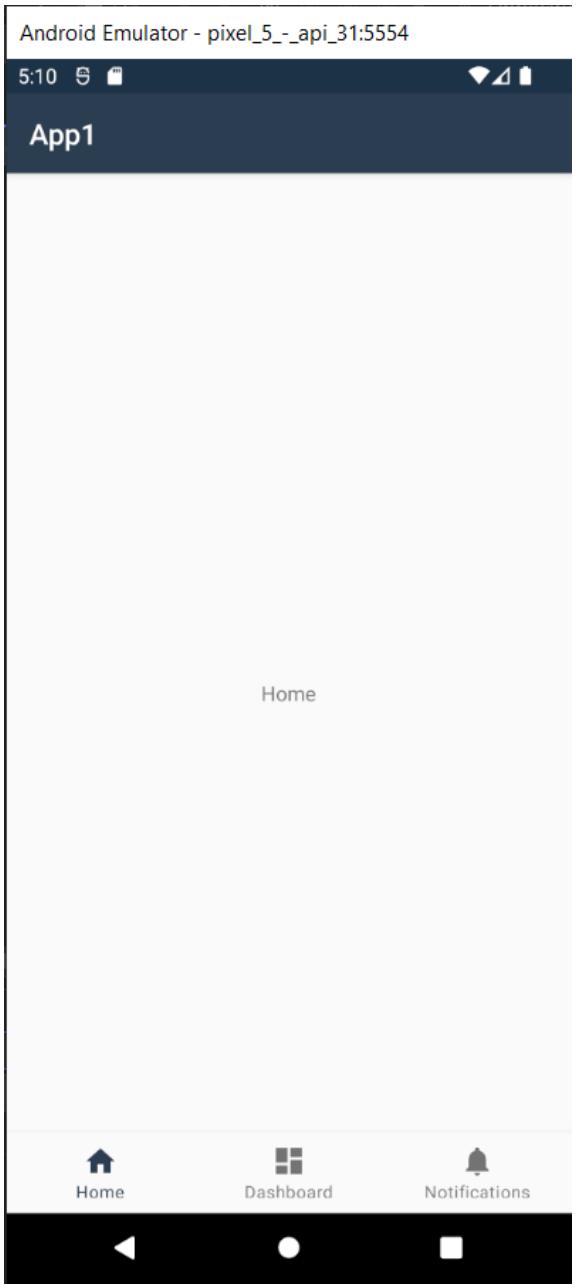
```
1    private async void OnLoginTapped(object sender, EventArgs e
     )
2    {
3        await Navigation.PopAsync();
4    }
5
```

Listing 20. Metoda OnLoginTapped

Metoda **OnLoginTapped** umożliwia użytkownikowi powrót do strony logowania, klikając w odpowiedni element interfejsu.

5. Testowanie

ale dziala eng.(but working)



Rys. 5.1. Test1

6. Podręcznik użytkownika

Bibliografia

[1] *Firebase*. URL: https://firebase.google.com/.

[2] *CODECOOL*. URL: https://codecool.com/pl/blog/github-czym-jest-i-do-czego-sluzy/.

Spis rysunków

Spis tabel

Spis listingów