FEJLESZTŐI DOKUMENTÁCIÓ

🎒 Osztálykirándulás Tervező Alkalmazás (OKKT25)

# 1. PROJEKT ÁTTEKINTÉS

## 1.1 Alkalmazás célja

Az OKKT25 egy mobilalkalmazás, amely célja, hogy megkönnyítse az osztálykirándulások tervezését, költségmegosztását és pénzügyi előkészületeit. Az alkalmazás segítségével pedagógusok és szülők hatékonyan tervezhetik meg a kirándulás költségeit, elemezhetik a diákok zsebpénzének fedezettségét, és dokumentálhatják a kapcsolódó kiadásokat.

## 1.2 Főbb funkciók

Költségtervezés részletes kedvezménykezeléssel

Zsebpénz elemzés egyéni és csoportos módban

Vizuális jelentések diagramokkal

Intelligens javaslatok költségfedezeti problémákra

Fotókezelés számlák dokumentálásához

PDF export professzionális jelentésekhez

Adatmentés lokális tárolásban

## 1.3 Célközönség

Osztályfőnökök és tanárok

Szülők és iskolai koordinátorok

Diákcsoportok vezetői

# 2. TECHNOLÓGIAI STACK

## 2.1 Fő technológiák

.NET MAUI 8.0 - Platformfüggetlen keretrendszer

C# 11 - Backend programozási nyelv

XAML - UI definíciók

MVVM minta - Architektúra

## 2.2 Külső könyvtárak

xml

<PackageReference Include="Microsoft.Maui.Controls" Version="$(MauiVersion)" />

<PackageReference Include="Microsoft.Extensions.Logging.Debug" Version="9.0.0" />

<PackageReference Include="PdfSharpCore" Version="1.3.67" />

## 2.3 Platform támogatás

Android (API 21+)

Windows (10.0.19041.0+)

# 3. ARCHITEKTÚRA ÉS DESIGN MINTÁK

## 3.1 Alkalmazás szerkezet

text

OKKT25/

├── Models/ // Adatmodellek

│ ├── TripData.cs

│ ├── TripSummary.cs

│ ├── CostItem.cs

│ └── MauiEntryExtensions.cs

├── Views/ // UI oldalak

│ ├── MainPage.xaml

│ ├── PastTripsPage.xaml

│ ├── TripDetailPage.xaml

│ └── ImageViewPage.xaml

├── ViewModels/ // Üzleti logika (implicit)

├── Resources/ // Erőforrások

└── Services/ // Szolgáltatások (implicit)

## 3.2 Design minták

3.2.1 MVVM (Model-View-ViewModel)

Az alkalmazás implicit MVVM mintát követ, ahol:

Model: TripData, CostItem, TripSummary osztályok

View: XAML fájlok (MainPage.xaml, stb.)

ViewModel: Code-behind fájlok üzleti logikával

3.2.2 Data Binding

xml

<Label Text="{Binding TripName}"

FontSize="18"

FontAttributes="Bold"

TextColor="#FFD700"/>

3.2.3 Command Pattern

csharp

public ICommand TripTappedCommand { get; }

TripTappedCommand = new Command<TripSummary>(async (trip) => await OpenTripAsync(trip));

# 4. ADATMODEL LEÍRÁS

## 4.1 TripData osztály

public class TripData

{

// Kirándulás alapadatok

public string TripName { get; set; } = string.Empty; // Kirándulás egyedi azonosítója

public string TripDestination { get; set; } = string.Empty; // Célállomás

public DateTime TripDateStart { get; set; } = DateTime.Now; // Kezdő dátum

public DateTime TripDateEnd { get; set; } = DateTime.Now; // Záró dátum

// Részvételi adatok

public int Participants { get; set; } // Résztvevők száma

public bool IsPerPersonMode { get; set; } // Költség számítás módja (főre vagy összesen)

// Pénzügyi adatok

public List<CostItem> Costs { get; set; } = new List<CostItem>(); // Költségtételek listája

public List<double> PocketMoney { get; set; } = new List<double>(); // Diákok havi zsebpénze

public double AveragePocketMoney { get; set; } // Átlagos zsebpénz számítva

// Műszaki adatok

public List<string> PhotoPaths { get; set; } = new List<string>(); // Csatolt képek útvonalai

public DateTime LastSaved { get; set; } = DateTime.Now; // Utolsó mentés időpontja

public bool Calculated { get; set; } = false; // Számítás elvégzése státusza

}

A TripData osztály a kirándulás összes adatát tartalmazza. A Costs lista a különböző kiadási kategóriákat (pl. szállás, étkezés), míg a PocketMoney lista a diákok egyéni pénzügyi lehetőségeit tárolja. Az IsPerPersonMode tulajdonság dönti el, hogy a költségeket főnként vagy összesítve kell számolni.

## 4.2 CostItem osztály

csharp

public class CostItem

{

public string Type { get; set; } = string.Empty;

public double Amount { get; set; }

public int NumberOfPeople { get; set; }

public bool HasDiscount { get; set; }

public double DiscountAmount { get; set; }

public int DiscountNumberOfPeople { get; set; }

}

Az osztály egy költségtételt reprezentál, amely tartalmazza a költség típusát, az összeget, a hozzá kapcsolódó személyek számát, valamint a kedvezményekre vonatkozó információkat.

**Tulajdonságok:**

* **Type**: A költségtétel típusa vagy kategóriája (például "Étkezés", "Szállítás").
* **Amount**: A költségtétel teljes összege kedvezmény nélkül.
* **NumberOfPeople**: Az a személyek száma, akikhez a költség kapcsolódik.
* **HasDiscount**: Logikai érték, amely jelzi, hogy a költségtételhez van-e kedvezmény.
* **DiscountAmount**: A kedvezmény összege, ha van alkalmazva.
* **DiscountNumberOfPeople**: Annak a személyeknek a száma, akikre a kedvezmény érvényes.

Ez a dokumentáció segít a fejlesztőknek megérteni, hogyan lehet egy költségtételt kezelni és a kedvezményeket nyomon követni.

## 4.3 TripSummary osztály

csharp

public class TripSummary

{

public string FileName { get; set; }

public string TripName { get; set; }

public string TripDestination { get; set; }

public DateTime TripDateStart { get; set; }

public DateTime TripDateEnd { get; set; }

public DateTime LastSaved { get; set; }

public int Participants { get; set; }

public double TotalCost { get; set; }

}

Az osztály egy utazás összegzését reprezentálja, tartalmazva az alapvető információkat, a résztvevők számát és a teljes költséget.

**Tulajdonságok:**

* **FileName**: Az utazás adatainak mentett fájlneve.
* **TripName**: Az utazás neve vagy címe.
* **TripDestination**: Az utazás célállomása.
* **TripDateStart**: Az utazás kezdő dátuma.
* **TripDateEnd**: Az utazás befejező dátuma.
* **LastSaved**: Az utazás adatainak utolsó mentésének időpontja.
* **Participants**: A résztvevők száma az utazáson.
* **TotalCost**: Az utazás teljes költsége minden kiadással együtt.

Ez az osztály lehetővé teszi a fejlesztők számára, hogy gyorsan áttekintsék egy utazás legfontosabb adatait, és nyomon kövessék a résztvevőket és a költségeket.

# 5. FELHASZNÁLÓI FELÜLET ÉS NAVIGÁCIÓ

## 5.1 Alkalmazás szerkezet

### 5.1.1 AppShell navigáció

xml

<TabBar>

<ShellContent

Title="Új kirándulás"

Icon="travel.png"

ContentTemplate="{DataTemplate local:MainPage}" />

<ShellContent

Title="Tervezetek"

Icon="folder.png"

ContentTemplate="{DataTemplate local:PastTripsPage}" />

</TabBar>

A TabBar a felhasználói felületen egy lapfül-sávot (tab bar) definiál, amely lehetővé teszi a felhasználó számára, hogy különböző nézetek között váltson.

**Elemei (ShellContent):**

1. **Új kirándulás**
   * **Title**: "Új kirándulás" – a tab címe.
   * **Icon**: "travel.png" – a tab ikonjának fájlneve.
   * **ContentTemplate**: MainPage – a tabhoz tartozó tartalomoldal sablonja, amely a felhasználó új kirándulást hozhat létre.
2. **Tervezetek**
   * **Title**: "Tervezetek" – a tab címe.
   * **Icon**: "folder.png" – a tab ikonjának fájlneve.
   * **ContentTemplate**: PastTripsPage – a tabhoz tartozó tartalomoldal sablonja, ahol a korábbi tervezett utazások listája jelenik meg.

**Leírás:**  
Ez a TabBar felépítés lehetővé teszi, hogy az alkalmazás főoldalán két fő nézetet biztosítson: az új kirándulások létrehozását és a már tervezett utazások megtekintését. Az ikonok és címek segítik a felhasználót a gyors navigációban.

### 5.1.2 Fő oldalak

MainPage - Új kirándulás létrehozása

PastTripsPage - Mentett kirándulások böngészése

TripDetailPage - Kirándulás részletes nézete

ImageViewPage - Kép megtekintése és kezelése

## 5.2 UI/UX Design elvek

### 5.2.1 Színpaletta

csharp

Primary: #121212 (háttér)

Secondary: #1E1E1E (kártyák)

Accent: #FFD700, #FF9800, #FFA500 (kiemelések)

Text: #FFFFFF, #C8C8C8 (szövegek)

### 5.2.2 Tipográfia

Betűcsalád: Arial

Méretek: 11px - 24px

Súlyok: Regular, Bold

# 6. ÜZLETI LOGIKA RÉSZLETEZÉS

## 6.1 Költségszámítás algoritmus

### 6.1.1 Teljes költség számítás

csharp

double totalCost = tripData.Costs.Sum(c =>

(c.Amount \* c.NumberOfPeople) +

(c.DiscountAmount \* c.DiscountNumberOfPeople));

### 6.1.2 Fedezettségi elemzés

csharp

bool canPay = monthlyTotal >= costPerPerson;

double shortage = costPerPerson - monthlyTotal;

6.1.3 Intelligens javaslatok

csharp

// Költségcsökkentés

double neededReduction = totalShortage;

// Több diák fizet

double extraPerPerson = totalShortage / (participants - cantPayList.Count);

// Több idő

int neededMonths = (int)Math.Ceiling(costPerPerson / pocketMoneyList.Min());

## 6.2 Adatvalidáció

### 6.2.1 Bemeneti ellenőrzések

csharp

if (!double.TryParse(pocketMoneyEntries[i].Text, out double amount) || amount < 0)

{

await ShowError($"Kérlek add meg a {i + 1}. diák érvényes zsebpénzét!");

return;

}

### 6.2.2 Dátum validáció

csharp

private void OnEndDateChanged(object sender, DateChangedEventArgs e)

{

if (TripDateEnd.Date < TripDateStart.Date)

{

TripDateEnd.Date = TripDateStart.Date;

}

}

# 7. ADATKEZELÉS ÉS TÁROLÁS

## 7.1 JSON szerializáció

csharp

public async Task SaveTripDataAsync()

{

string json = System.Text.Json.JsonSerializer.Serialize(tripData,

new System.Text.Json.JsonSerializerOptions { WriteIndented = true });

string tripFileName = $"{tripData.TripName}.json";

string filePath = Path.Combine(FileSystem.Current.AppDataDirectory, tripFileName);

await File.WriteAllTextAsync(filePath, json);

}

Ez az aszinkron metódus az aktuális utazás adatait menti JSON formátumban a helyi fájlrendszerre.

**Működés:**

1. A metódus a tripData objektumot JSON formátumba alakítja, emberileg olvasható formázással (WriteIndented = true).
2. A fájl neve az utazás nevéből (TripName) generálódik, .json kiterjesztéssel.
3. A fájl a rendszer által biztosított alkalmazásadat-könyvtárba kerül elmentésre (FileSystem.Current.AppDataDirectory).
4. A tényleges írás a fájlba aszinkron módon történik (File.WriteAllTextAsync).

**Használat:**

* A metódus lehetővé teszi, hogy az utazás adatai offline is elérhetők legyenek, és később betölthetők legyenek az alkalmazásba.
* Az aszinkron végrehajtás biztosítja, hogy a felhasználói felület ne akadjon meg a mentés során.

**Megjegyzés:**

* A metódus feltételezi, hogy a tripData objektum érvényes és megfelelően inicializált.
* A fájlnév generálásnál figyelni kell a fájlrendszer által tiltott karakterekre a TripName mezőben.

## 7.2 Fájlkezelés

### 7.2.1 Mentés helye

csharp

string appDataDir = FileSystem.Current.AppDataDirectory;

var jsonFiles = Directory.GetFiles(appDataDir, "\*.json");

### 7.2.2 Fotókezelés

csharp

string targetPath = Path.Combine(FileSystem.Current.AppDataDirectory,

Path.GetFileName(result.FullPath));

## 7.3 Adatbetöltés

csharp

private async void LoadTrips()

{

trips.Clear();

var appDataDir = FileSystem.Current.AppDataDirectory;

var jsonFiles = Directory.GetFiles(appDataDir, "\*.json");

foreach (var file in jsonFiles)

{

var json = await File.ReadAllTextAsync(file, Encoding.UTF8);

var tripData = JsonSerializer.Deserialize<TripData>(json);

// Feldolgozás...

}

}

Ez az aszinkron metódus az alkalmazás korábban mentett utazásait tölti be a helyi fájlrendszerből.

**Működés:**

1. A meglévő trips lista kiürül, hogy friss adatok kerüljenek betöltésre.
2. Lekérdezi az alkalmazás adatkönyvtárát (FileSystem.Current.AppDataDirectory).
3. Az adatkönyvtárban keres minden .json kiterjesztésű fájlt, amelyek az utazások adatait tartalmazzák.
4. Minden fájlt aszinkron módon beolvas UTF-8 kódolással.
5. A JSON tartalmat deszerializálja TripData objektummá.
6. A deszerializált objektum további feldolgozásra kerül (például hozzáadása a trips listához).

**Használat:**

* A metódus lehetővé teszi az összes mentett utazás adatának betöltését az alkalmazás indításakor vagy frissítéskor.
* Aszinkron működésének köszönhetően nem blokkolja a felhasználói felületet a fájlok olvasása közben.

**Megjegyzés:**

* A metódus feltételezi, hogy minden JSON fájl érvényes és megfelel a TripData szerkezetének.
* Hibakezelés (például sérült vagy hiányzó fájlok esetén) érdemes implementálni a stabil működés érdekében.

# 8. GRAFIKUS ELEMEK ÉS DIAGRAMOK

## 8.1 Egyéni rajzolás (IDrawable)

### 8.1.1 PieChartDrawable

csharp

public class PieChartDrawable : IDrawable

{

public void Draw(ICanvas canvas, RectF dirtyRect)

{

// Kördiagram rajzolása

DrawPieSlice(canvas, centerX, centerY, radius, -90, canPayAngle, Color.FromArgb("#4CAF50"));

}

}

Ez az osztály egyéni rajzolást valósít meg a IDrawable interfészen keresztül, konkrétan kördiagram (pie chart) megjelenítésére.

**Fő funkciók:**

* **Draw metódus**: Ez a metódus felelős a kördiagram tényleges megjelenítéséért a megadott ICanvas felületen.
  + A metódus a diagram szeleteit (pie slices) rajzolja ki az adott középpont (centerX, centerY) és sugár (radius) alapján.
  + A szeletek színezése az alkalmazás logikája szerint történik (például a fizethető rész zöld színnel: #4CAF50).
  + A szeletek kezdő szöge és kiterjedése (pl. -90 kezdőszög, canPayAngle szelet) a kördiagram adatainak vizualizációját szolgálja.

**Használat:**

* A PieChartDrawable osztály alkalmas grafikus felületeken (pl. MAUI vagy Xamarin alkalmazásokban) a pénzügyi vagy statisztikai adatok vizuális megjelenítésére.
* Bármilyen ICanvas típusú rajzoló felülethez hozzárendelhető, így a kördiagram dinamikusan frissíthető a felhasználói interakciók vagy adatmódosítások hatására.

**Megjegyzés:**

* Az osztály implementálja az IDrawable interfészt, így biztosítja az egységes rajzolási szerkezetet más egyéni rajzoló elemekkel.
* A diagram szeletenkénti színezése és kiterjedése testreszabható az alkalmazás logikájának megfelelően.

### 8.1.2 CostBreakdownDrawable

csharp

public class CostBreakdownDrawable : IDrawable

{

public void Draw(ICanvas canvas, RectF dirtyRect)

{

// Költségmegoszlási diagram

foreach (var cost in costs)

{

float percentage = (float)(costValue / totalCost);

float sweepAngle = percentage \* 360f;

DrawPieSlice(canvas, centerX, centerY, radius, startAngle, sweepAngle, sliceColor);

}

}

}

Ez az osztály egyéni rajzolást valósít meg a IDrawable interfészen keresztül, és költségmegoszlási diagram (pie chart) létrehozására szolgál.

**Fő funkciók:**

* **Draw metódus**: A metódus felelős a költségtételek vizuális megjelenítéséért egy kördiagram formájában a megadott ICanvas felületen.
  + A diagram minden költségtételhez egy szeletet (pie slice) rajzol a teljes költséghez viszonyított arány alapján.
  + A szeletek kezdő szöge (startAngle) és kiterjedése (sweepAngle) a költségtétel arányát tükrözi a teljes költséghez képest.
  + A szeletek színezése az alkalmazás logikája szerint történik (sliceColor).
* A diagram minden költségtételhez egy külön szeletet jelenít meg, így vizuálisan könnyen értelmezhető a költségek megoszlása.

**Használat:**

* Alkalmas grafikus felületeken (például MAUI vagy Xamarin alkalmazásokban) a költségadatok vizuális elemzésére.
* Az ICanvas típusú rajzoló felületekhez csatlakoztatva a költségdiagram dinamikusan frissíthető az adatok változásakor.

**Megjegyzés:**

* Az osztály implementálja az IDrawable interfészt, így kompatibilis más egyéni rajzoló elemekkel.
* A szeletek színezése és a diagram arányai testreszabhatók az alkalmazás logikájának megfelelően.

## 8.2 GraphicsView használata

csharp

public class PieChartView : GraphicsView

{

public PieChartView(List<double> pocketMoney, double cost, int months)

{

Drawable = new PieChartDrawable(pocketMoney, cost, months);

}

}

Ez az osztály a GraphicsView leszármazottja, és egy kördiagram (pie chart) megjelenítésére szolgál a felhasználói felületen.

**Konstruktor:**

* **Paraméterek:**
  + pocketMoney (List<double>): A zsebpénz vagy rendelkezésre álló összeg adatai, amelyek a diagram szeleteit meghatározzák.
  + cost (double): Az összehasonlítandó költség, amelyhez a zsebpénzt viszonyítjuk.
  + months (int): Az időtartam hónapokban, amely segít a diagram értékeinek arányosításában.

**Fő funkciók:**

* A konstruktor inicializálja a Drawable tulajdonságot egy PieChartDrawable objektummal, amely felelős a kördiagram tényleges rajzolásáért.
* A PieChartView automatikusan megjeleníti a diagramot a felületen, és vizuálisan ábrázolja a pénzügyi adatokat.

**Használat:**

* Alkalmas grafikus felületeken (például MAUI vagy Xamarin alkalmazásokban) a pénzügyi adatok gyors és vizuális áttekintésére.
* A diagram dinamikusan frissíthető az adatok változásakor, ha új PieChartDrawable kerül hozzárendelésre.

# 9. PDF EXPORT FUNKCIONALITÁS

## 9.1 PDF generálás PdfSharpCore-dal

### 9.1.1 Dokumentum létrehozása

csharp

using (var document = new PdfSharpCore.Pdf.PdfDocument())

{

var page = document.AddPage();

page.Size = PdfSharpCore.PageSize.A4;

using (var gfx = XGraphics.FromPdfPage(page))

using (var xImage = XImage.FromFile(tempImagePath))

{

gfx.DrawImage(xImage, 0, 0, pageWidth, pageHeight);

}

document.Save(filePath);

}

Ez a kódrészlet egy PDF dokumentum létrehozását és egy kép beszúrását valósítja meg a PdfSharpCore könyvtár segítségével.

**Működés:**

1. Létrehoz egy új PDF dokumentumot (PdfDocument).
2. Hozzáad egy új oldalt a dokumentumhoz, és beállítja az oldal méretét A4-re.
3. Az oldalon egy XGraphics objektumot hoz létre a rajzoláshoz.
4. Betölt egy képfájlt (XImage.FromFile) a megadott tempImagePath elérési út alapján.
5. A képet az oldalra rajzolja, a megadott szélességgel és magassággal (pageWidth, pageHeight).
6. A dokumentumot elmenti a megadott filePath helyre.

**Használat:**

* Alkalmas PDF-jelentések, utazási összefoglalók vagy bármilyen dokumentum generálására, ahol képek és grafikus tartalom szükséges.
* A kód automatikusan kezeli a PDF oldalak létrehozását és a kép beszúrását a megfelelő méretezéssel.

**Megjegyzés:**

* Fontos biztosítani, hogy a tempImagePath érvényes fájlra mutasson, különben hibát dob a XImage.FromFile.
* A pageWidth és pageHeight paramétereknek meg kell felelniük az oldal arányainak, hogy a kép ne torzuljon.
* A using blokkok gondoskodnak a PDF és a kép erőforrásainak helyes felszabadításáról.

### 9.1.2 Platformfüggetlen útvonalak

csharp

#if ANDROID

var downloadsPath = Android.OS.Environment

.GetExternalStoragePublicDirectory(Android.OS.Environment.DirectoryDownloads)

.AbsolutePath;

#elif WINDOWS

var downloadsPath = Path.Combine(

Environment.GetFolderPath(Environment.SpecialFolder.UserProfile),

"Downloads");

#endif

## 9.2 Képernyőkép készítése

csharp

var screenshot = await pdfView.CaptureAsync();

# 10. FOTÓKEZELÉS ÉS MEDIA

## 10.1 Képfeltöltés lehetőségek

csharp

string action = await DisplayActionSheet("Fénykép hozzáadása", "Mégse", null,

"📷 Kamera", "🖼️ Galéria");

## 10.2 Kép nagyítás és navigáció

csharp

public partial class ImageViewPage : ContentPage

{

private double currentScale = 1;

private double startScale = 1;

private void OnPinchUpdated(object sender, PinchGestureUpdatedEventArgs e)

{

// Pinch zoom implementáció

}

}

## 10.3 Gesture kezelés

csharp

var pinchGesture = new PinchGestureRecognizer();

pinchGesture.PinchUpdated += OnPinchUpdated;

FullImage.GestureRecognizers.Add(pinchGesture);

var doubleTap = new TapGestureRecognizer { NumberOfTapsRequired = 2 };

doubleTap.Tapped += (s, e) => ResetZoom();

Ez a kódrészlet a felhasználói interakciókhoz kapcsolódó gesztusokat valósít meg egy képen (FullImage), lehetővé téve a nagyítást és az alaphelyzetbe állítást.

**Működés:**

1. **Pinch (csippentés) gesztus:**
   * Létrehoz egy PinchGestureRecognizer objektumot.
   * A PinchUpdated eseményhez hozzáad egy kezelőt (OnPinchUpdated), amely a képre vonatkozó nagyítási műveleteket kezeli.
   * A gesztus felismerő hozzáadódik a FullImage elem GestureRecognizers gyűjteményéhez.
   * Lehetővé teszi, hogy a felhasználó két ujjal húzva nagyítsa vagy kicsinyítse a képet.
2. **Double-Tap (kétszeri koppintás) gesztus:**
   * Létrehoz egy TapGestureRecognizer objektumot, amely két egymást követő koppintást igényel (NumberOfTapsRequired = 2).
   * A Tapped eseményhez egy inline kezelőt ad, amely a ResetZoom() metódust hívja.
   * A gesztus felismerő lehetővé teszi, hogy a felhasználó kétszeri koppintással visszaállítsa a kép nagyítását az alapértelmezett állapotra.

**Használat:**

* Ezek a gesztusok interaktív felhasználói élményt biztosítanak képek böngészésekor.
* A pinch zoom és a double-tap reset kombinációja intuitív módon teszi lehetővé a kép nagyítását és visszaállítását.

# 11. PLATFOROMFÜGGŐ IMPLEMENTÁCIÓK

## 11.1 Android specifikus beállítások

### 11.1.1 StatusBar szín beállítás

csharp

#if ANDROID

Microsoft.Maui.Handlers.WindowHandler.Mapper.AppendToMapping(

"StatusBarColor",

(handler, view) =>

{

var activity = Platform.CurrentActivity;

var window = activity.Window;

window.SetStatusBarColor(Android.Graphics.Color.ParseColor("#121212"));

});

#endif

### 11.1.2 RadioButton testreszabás

csharp

#if ANDROID

radioButton.HandlerChanged += (s, e) =>

{

if (radioButton.Handler.PlatformView is Android.Widget.RadioButton androidRadioButton)

{

var colorStateList = new ColorStateList(states, colors);

androidRadioButton.ButtonTintList = colorStateList;

}

};

#endif

# 12. HIBAKEZELÉS ÉS LOGOLÁS

## 12.1 Globális hibakezelés

csharp

try

{

// Művelet végrehajtása

}

catch (Exception ex)

{

await DisplayAlert("Hiba", $"Nem sikerült a művelet: {ex.Message}", "OK");

}

## 12.2 Logolás konfiguráció

csharp

#if DEBUG

builder.Logging.AddDebug();

#endif

## 12.3 Felhasználóbarát hibaüzenetek

csharp

await DisplayAlert("Hiba", "Kérlek adj meg érvényes költség összeget minden tételhez!", "OK");

# 13. ANIMÁCIÓK ÉS FELHASZNÁLÓI ÉLMÉNY

## 13.1 Egyszerű animációk

csharp

private async void AnimateView(View view)

{

view.Opacity = 0;

await view.FadeTo(1, 300);

}

## 13.2 Hibajelzés animáció

csharp

private async Task ShowError(string message)

{

await DisplayAlert("Hiba", message, "OK");

await BtnCalculate.TranslateTo(-15, 0, 50);

await BtnCalculate.TranslateTo(15, 0, 50);

// Reszponzív shake animáció

}

# 14. TESZTELÉSI STRATÉGIA

## 14.1 Tesztelendő komponensek

### 14.1.1 Unit tesztek

Költségszámítás logika

Adatvalidációk

JSON serializáció/deserializáció

### 14.1.2 UI tesztek

Navigáció működése

Adatkötések helyessége

Reszponzív design

### 14.1.3 Integrációs tesztek

Fájl I/O műveletek

PDF generálás

Fotókezelés

### 14.2 Tesztkörnyezet

Android Emulátor - API 21+

Windows Desktop - 10.0.19041.0+

Fizikai eszközök - Különböző képernyőméretek

# 15. BEVEZETÉSI ÚTMUTATÓ

## 15.1 Fejlesztői környezet beállítása

### 15.1.1 Előfeltételek

Az alkalmazás fejlesztéséhez és futtatásához az alábbi szoftverek és eszközök szükségesek:

1. **Visual Studio 2022 17.8 vagy újabb**
   * A fejlesztői környezet, amely támogatja a .NET MAUI és cross-platform fejlesztést.
2. **.NET 8.0 SDK**
   * A futtatókörnyezet és fordító, amely szükséges a projekt buildeléséhez és futtatásához.
3. **Android SDK (API 21 vagy újabb)**
   * Az Android alkalmazások buildeléséhez és futtatásához szükséges fejlesztői eszközök és könyvtárak.
4. **Windows 10 SDK (19041 vagy újabb)**
   * A Windows platformra történő buildeléshez szükséges fejlesztői eszközök és könyvtárak.

### 15.1.2 Projekt betöltése

bash

git clone <repository-url>

cd OKKT25

dotnet restore

## 15.2 Build és deploy

### 15.2.1 Android

bash

dotnet build -f net8.0-android -c Release

dotnet publish -f net8.0-android -c Release

### 15.2.2 Windows

bash

dotnet build -f net9.0-windows10.0.19041.0 -c Release

dotnet publish -f net9.0-windows10.0.19041.0 -c Release

# 16. ISMERT KORLÁTOZÁSOK ÉS JÖVŐBELI FEJLESZTÉSEK

## 16.1 Jelenlegi korlátozások

Nincs felhőszinkronizáció

Limitált képméretek kezelése

Csak lokális adattárolás

## 16.2 Jövőbeli bővítési lehetőségek

Felhő integráció - OneDrive, Google Drive

Többnyelvűség - Angol, német támogatás

Haladó jelentések - Excel export, statisztikák

Collaboration - Több felhasználós szerkesztés

Offline mód - Teljes offline funkcionalitás

# 17. BIZTONSÁGI MEGFONTOLÁSOK

## 17.1 Adatvédelem

Minden adat lokálisan tárolódik

Nincs adatgyűjtés külső szerverekre

Fájlok titkosítva tárolhatók jövőben

## 17.2 Engedélyek

xml

<!-- Android engedélyek -->

<uses-permission android:name="android.permission.CAMERA" />

<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

Az alkalmazás működéséhez az alábbi Android engedélyek szükségesek:

1. **CAMERA**
   * Engedélyezés: android.permission.CAMERA
   * Leírás: Lehetővé teszi az alkalmazás számára a készülék kamerájának használatát, például fényképek készítéséhez vagy QR-kód beolvasáshoz.
2. **READ\_EXTERNAL\_STORAGE**
   * Engedélyezés: android.permission.READ\_EXTERNAL\_STORAGE
   * Leírás: Lehetővé teszi az alkalmazás számára a felhasználó eszközén található külső tárolóról történő fájlok olvasását, például képek vagy dokumentumok betöltéséhez.
3. **WRITE\_EXTERNAL\_STORAGE**
   * Engedélyezés: android.permission.WRITE\_EXTERNAL\_STORAGE
   * Leírás: Lehetővé teszi az alkalmazás számára fájlok írását vagy módosítását a felhasználó eszközének külső tárolójában, például képek vagy PDF-ek mentéséhez.

# 18. TELJESÍTMÉNYOPTIMALIZÁLÁS

## 18.1 Memóriakezelés

csharp

// IDisposable objektumok proper kezelése

using (var stream = await screenshot.OpenReadAsync())

using (var fileStream = File.Create(tempImagePath))

{

await stream.CopyToAsync(fileStream);

}

Ez a kódrészlet a memóriakezelés és az IDisposable objektumok helyes kezelésének példáját mutatja, különösen fájlok és adatfolyamok másolásánál.

**Működés:**

1. A using blokk biztosítja, hogy az IDisposable típusú objektumok (például stream és fileStream) automatikusan felszabaduljanak a használat után.
2. A screenshot.OpenReadAsync() aszinkron módon nyitja meg a képernyőkép olvasására szolgáló adatfolyamot.
3. A File.Create(tempImagePath) létrehoz egy új fájlt a megadott ideiglenes elérési úton (tempImagePath).
4. A stream.CopyToAsync(fileStream) másolja az adatfolyam tartalmát a fájlba aszinkron módon.
5. A using blokkok lezárásakor mindkét adatfolyam automatikusan lezáródik és felszabadítja a memóriát, megelőzve az erőforrás-szivárgást.

**Használat:**

* Az IDisposable objektumok, mint például fájl- és hálózati adatfolyamok kezelése során mindig javasolt a using blokk alkalmazása a biztonságos memória- és erőforrás-kezelés érdekében.
* Aszinkron műveleteknél (await) is fontos a megfelelő using struktúra, hogy az objektumok a művelet befejeztével felszabaduljanak.

**Megjegyzés:**

* Helytelen memória- vagy erőforrás-kezelés memória- és fájl-leakhez vezethet, ami teljesítménycsökkenést vagy hibákat okozhat.
* Az aszinkron using (await using) is alkalmazható, ha az objektum IAsyncDisposable interfészt valósít meg.

## 18.2 UI reszponzivitás

csharp

// Hosszú műveletek háttérszálon

await Task.Run(() => {

// CPU intenzív műveletek

});

# 19. KÓD MINŐSÉG ÉK KARBANTARTÁS

## 19.1 Kódstílus és konvenciók

C# naming conventions - camelCase, PascalCase

XAML coding standards - konzisztens formázás

Comment policy - komplex logika dokumentálása

# 20. ÖSSZEFOGLALÁS

Az OKKT25 alkalmazás egy átfogó megoldást kínál osztálykirándulások tervezésére és költségkezelésére. A .NET MAUI keretrendszer használata lehetővé teszi a platformfüggetlen fejlesztést, miközben natív felhasználói élményt biztosít. Az alkalmazás moduláris architektúrája, részletes üzleti logikája és felhasználóbarát felülete ideális eszközt kínál pedagógusok és szülők számára a kirándulások hatékony megszervezéséhez.

A jelen dokumentáció részletesen bemutatja az alkalmazás technikai megvalósítását, architektúráját és fejlesztési folyamatait, amely alapot nyújt a további fejlesztésekhez és karbantartáshoz.