אפליקציה לניהול קבוצת כדורגל

**עבודת גמר**

**בתכנון ותכנות מערכות**

**במסלול פיתוח שרותי אינטרנט, תכנות אסינכרוניות ומסדי נתונים**

מוגש על ידי ליאב צדקיהו

תעודת זהות 216793604

העבודה בוצעה בהנחיית יאנה זמוסטיאנו

התשפ"ה 2025

תוכן

[מבוא 3](#_Toc196597188)

[ניתוח המערכת 4](#_Toc196597189)

[בסיס הנתונים 5](#_Toc196597190)

[מימוש הפרויקט 6](#_Toc196597191)

[מדריך למשתמש 7](#_Toc196597192)

[רפלקציה/סיכום אישי 8](#_Toc196597193)

[ביבליוגרפיה 9](#_Toc196597194)

[נספחים 10](#_Toc196597195)

## מבוא

***רקע לפרויקט:***

***שם הפרויקט:***

אפליקציית ניהול קבוצת כדורגל.

***קהל היעד:***

מנהלים ומאמנים של קבוצות כדורגל מקצועיות.

***תיאור קצר של הפרויקט:***

האפליקציה פונה למנהלי קבוצות כדורגל שצריכים ארגון וסדר בניהול המקצועי שלהם

ובתקשורת בין הצוות. יש מספר תוכנות כאלו בשוק כמו SportEasy ו-TeamSnap אך מה

שמייחד את התוכנה שלי זה שהיא לא בנוייה לעבוד עם קבוצות חובבניות )חברים שבאים

למגרש עירוני כדי לשחק לכיף(, אלא לקבוצות מקצועניות בדרגים הכי גבוהים באירופה.

תהיה אפשרות להיכנס לאפליקציה דרך שני סוגים של משתמשים: מנהל קבוצה ומאמן קבוצה.

פונקציונליות של מנהל: קביעת תקציב (תקציב העברות ותקציב שכר), דרישה לרכש מסוים,

צפייה ברשימת שחקנים, תקשורת עם המאמן (קבלת רשימת רכש מהמאמן בשביל

אישור\דחייה של שחקנים מסוימים).

פונקציונליות של מאמן: חיפוש שחקנים במבנה נתונים גדול לפי סינון: סטטיסטיקה, מחיר, גיל, לאום ועוד, תהייה מוצגת רשימת השחקנים הנוכחים בקבוצה.

***הסיבות לבחירת הנושא:***

1. אהבה אישית לעולם הכדורגל.
2. חוסר במערכות לניהול קבוצות מקצועניות.
3. רצון ליצירת פיתרון קל לשימוש עבור משתמשים שאין להם רקע טכני.

***מטרות המערכת:***

***מטרות על:***

לאפשר ניהול מקצועי ומסודר של קבוצות כדורגל.

לאפשר תקשורת נוחה בין המאמן ומנהל.

***מטרות נלוות:***

מעקב אחרי שחקנים(נתונים אישיים וסטטיסטיקה).

ניהול תקציבים.

הנגשת חיפוש שחקנים ורכש.

***תיאור המערכת:***

האפליקציה כוללת דפים ייעודיים לניהול תקציב, צפייה בפרופילי שחקנים, בניית הרכבי משחק אופטימליים וחיפוש שחקנים לרכש לפי קריטריונים שונים. היא מותאמת במיוחד לקבוצות מקצועניות הזקוקות לכלים פשוטים אך עוצמתיים.

***גבולות המערכת:***

הנתונים לא מתעדכנים בזמן אמת ביחס לעונה הנוכחית.

אין אפשרות של העברת שחקנים מקבוצה אחת לאחרת.

***סביבות פיתוח:***

ויז'ואל סטודיו 2022.

***שפות תכנות:***

Xaml

C#

SQL

***שכבות:***

Model

View

ViewModel

***פלטפורמות:***

Windows – נבחרה כי זאת מערכת ההפעלה הביתית הכי נפוצה בימינו.

Android – הרצת האפליקציה דרך מכשיר בעל מערכת הפעלה android. נבחרה בגלל הפשטות בהרצת דברים עליה וגם בגלל השכיחות שלה בעולם הסלולרי

***אתגרים מרכזיים:***במהלך פיתוח הפרויקט התמודדתי עם מספר אתגרים מרכזיים. האתגר הראשון היה בניית **מערכת הרשאות** למשתמשים שונים – מאמנים ומנהלים – כך שכל אחד מהם יוכל לגשת רק לפונקציונליות הרלוונטית לו. נדרשתי לתכנן מבנה הרשאות גמיש, שמונע גישה לא מורשית ועדיין שומר על חוויית משתמש חלקה.

אתגר נוסף היה השימוש בטכנולוגיית **.NET MAUI**, שנחשבת יחסית חדשה בשוק. עקב כך, לעיתים היה קשה למצוא תיעוד, פתרונות לבעיות ופורומים פעילים, מה שהוביל אותי לחקור לעומק את התשתית ולפתור תקלות בעצמי דרך ניסוי וטעייה.

כמו כן, תהליך **עיצוב מסכי ניהול התקציב** דרש מחשבה רבה. חיפשתי דרכים להפוך את הממשק לפשוט, ברור ואינטואיטיבי – כך שגם משתמשים שאינם טכנולוגיים יוכלו להבין את המערכת במהירות. השקעתי רבות בבניית גרפים, איזון תקציבי באמצעות סליידר, ושימוש בטקסטים מפורשים כדי לשפר את הנגישות.

הפרויקט בא לתת מענה לצורך ממשי של ניהול קבוצת כדורגל בצורה מקצועית ודיגיטלית – כולל שליטה בתקציבים, מעקב אחר שחקנים, וניהול הרשאות. בשלב המחקר הראשוני בחנתי פתרונות אחרים לניהול קבוצות אך מצאתי כי רובם אינם מותאמים לעברית או אינם כוללים לוגיקה כלכלית מותאמת אישית, ולכן פיתחתי פתרון שמבוסס על הצרכים המדויקים שזיהיתי.

***חידושים, התאמות ועדכונים טכנולוגיים:***

במהלך הפיתוח שילבתי מספר **חידושים טכנולוגיים ועיצוביים** על מנת להפוך את המערכת ליעילה, מתקדמת ונוחה יותר לשימוש.

אחד החידושים המשמעותיים היה פיתוח **מערכת גרפים אינטראקטיבית** לניהול תקציב עונתי וחודשי. הגרפים מציגים בצורה ברורה את התפלגות ההוצאות וההכנסות לאורך זמן, ומאפשרים למאמן להבין מגמות תקציביות ולהגיב בהתאם.

בנוסף, שילבתי **רכיבי שליטה דינמיים** כמו סליידר לחלוקת תקציב בין רכישות לשכר, מה שאיפשר התאמה אישית של התקציב בזמן אמת תוך שמירה על שלמות המידע.

המערכת עושה שימוש ב־**Web API חיצוני** בגישת JSON, מה שמאפשר גישה אוניברסלית מכל מכשיר – מבלי להיות תלוי בקובץ מקומי. זהו פתרון חיוני לאפליקציה שצריכה לתפקד בסביבה מרובת משתמשים מרוחקים.

לבסוף, ביצעתי **התאמות עיצוביות** מלאות לממשק המשתמש – כולל עיצוב רספונסיבי שמתאים למסכים קטנים, שימוש בצבעים אחידים, טקסטים ברורים, והיררכיה ויזואלית שמכוונת את המשתמש באופן אינטואיטיבי.

## ניתוח המערכת

***מסמך ייזום:***

בעידן הדיגיטלי של היום, קבוצות כדורגל מקצועניות מוצאות את עצמן זקוקות לפתרונות ניהול מודרניים. מספר הקבוצות בעולם, חוסר תקשורת מסודר, ניהול תקציבי לקוי ומעקב לא מסודר אחר נתוני שחקנים. כל אלו יוצרים את הצורך באפליקציה ייעודית שתאפשר ניהול יעיל ומקצועי של קבוצות כדורגל. האפליקציה "ניהול קבוצת כדורגל" באה לתת מענה לצורך זה, תוך שהיא מותאמת בדיוק לקבוצות מקצועניות שמחפשות דרך להתנהל בצורה מסודרת וחכמה.

***מצב קיים:***

קבוצות מקצועניות רבות מתנהלות כיום באמצעות:

* קבוצות וואטסאפ לא מסודרות.
* ניהול תקציב על דפי אקסל פשוטים.
* מעקב ידני אחר שחקנים.
* חוסר בכלי פשוט לבניית הרכב משחק אופטימלי.

***מטרות המערכת העתידית:***

המערכת צפויה להתפתח ולהציע כלים מתקדמים שירחיבו את היכולות הניהוליות והמקצועיות של קבוצה.  
בין המטרות המרכזיות לעתיד:

* פיתוח **אלגוריתם לבניית הרכב אופטימלי** שיתחשב בפרמטרים כמו סטטיסטיקות עדכניות, עייפות שחקנים, התאמה טקטית ורגל מועדפת.
* יצירת **מנגנון העברות שחקנים** דינמי, שיאפשר מו"מ, רכישה, מכירה והשאלה של שחקנים בין קבוצות – כולל ניהול חוזים, שכר, ותקציב העברה.
* שילוב **מערכת התרעות אוטומטית** שתעדכן את המאמן או המנהל באירועים חשובים – כמו ירידה בביצועים, חרגות תקציב או פציעות.
* בניית **מערכת אנליטיקה חכמה** לניתוח מגמות בקבוצה – למשל עלייה/ירידה בביצועים לאורך זמן, שחקנים בולטים, ותכנון עונה עתידי.
* פיתוח **לוח עבודה שיתופי** לאנשי הצוות: ניהול משימות, תזכורות, שיחות פנימיות, ותיעוד החלטות – הכל מתוך המערכת.
* הרחבת **הגישה לענן** כך שהמידע יסונכרן בין מכשירים שונים בזמן אמת, כולל גיבוי אוטומטי ואבטחת מידע מתקדמת.

***מצב עתידי:***

בעתיד, עם הרחבת היכולות, המערכת תאפשר:

* למאמן **לבנות הרכב אוטומטי** ומבוסס נתונים שמותאם ליריב הקרוב, תוך שקלול ביצועי עבר, עמדות והתאמה קבוצתית.
* למנהל לבצע **מהלכי רכש ומכירה של שחקנים**, לעקוב אחר השוק, להציע חוזים ולנהל מו”מ באופן מלא מתוך הפלטפורמה.
* לצוות כולו לקבל **התראות בזמן אמת** על אירועים כמו חריגה מהתקציב, פציעות שחקנים או סטייה מיעדי ביצוע.
* להפיק **דו"חות אנליטיים מתקדמים** שיעזרו להבין מגמות קבוצתיות ולקבל החלטות אסטרטגיות.
* לשתף פעולה על גבי **מערכת משימות ולוחות זמנים**, שתאפשר תקשורת רציפה בין המאמן, המנהל, אנשי הרפואה, והצוות המקצועי.
* לנהל את כל המידע של הקבוצה **בענן מאובטח**, עם סנכרון אוטומטי בין מכשירים – כך שאנשי הצוות יוכלו להתחבר למערכת מכל מקום, בכל זמן.

***UserCase:***

תמונה שמכילה צילום מסך, קו, אסטרונומיה

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

***DFD-0:***

תמונה שמכילה צילום מסך, תרשים

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

**תמונות של העמודים:**

**דף כניסה:**

תמונה שמכילה טקסט, צילום מסך, תוכנה, תכונות מולטימדיה

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

**דף ראשי:**

**תמונה שמכילה טקסט, צילום מסך, תוכנה, תכונות מולטימדיה

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**

**צפייה במשתמשים:**

תמונה שמכילה צילום מסך, טקסט, תוכנה, תכונות מולטימדיה

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

**חיפוש רכש:**

תמונה שמכילה צילום מסך, תוכנה, תכונות מולטימדיה, מחשב

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

**דף קבוצה:**

תמונה שמכילה צילום מסך, תוכנה, תכונות מולטימדיה, מחשב

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

**דף תקציב:**

תמונה שמכילה טקסט, צילום מסך, תוכנה, תכונות מולטימדיה

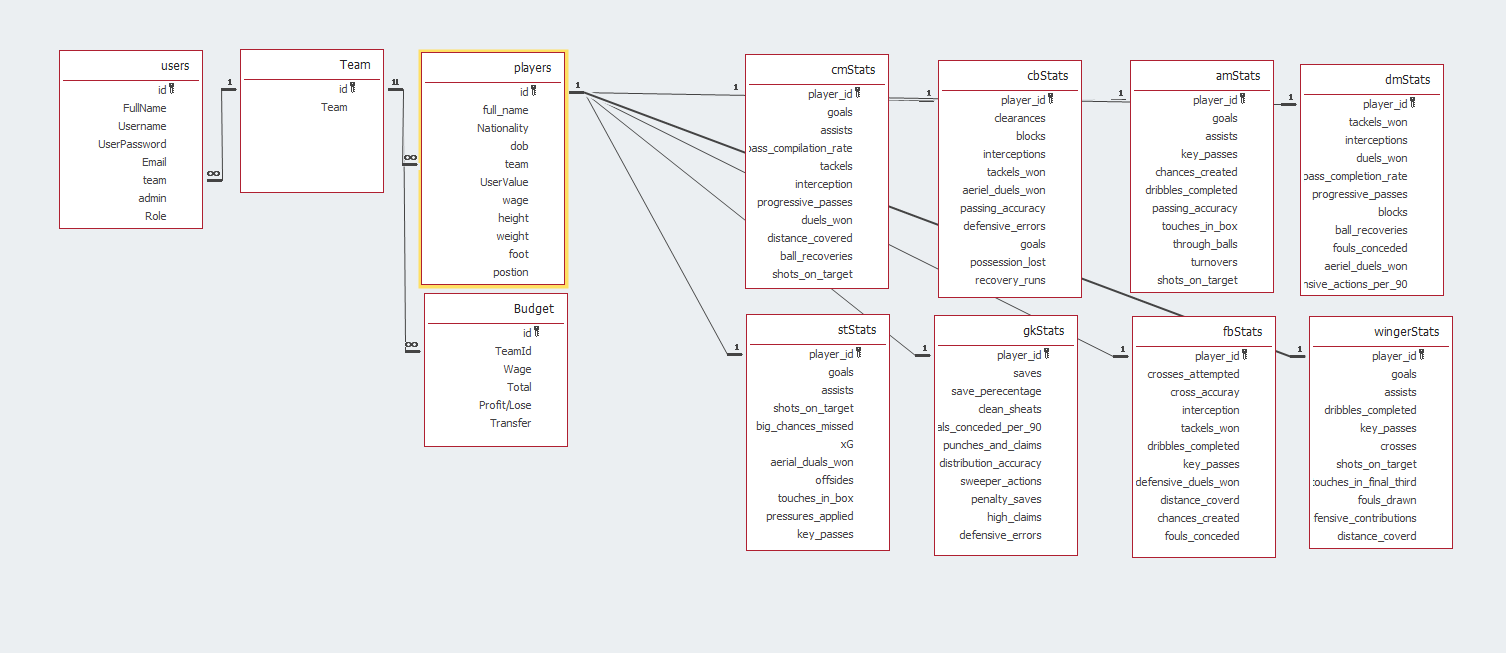
תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

**פרופיל של שחקן:**

תמונה שמכילה צילום מסך, טקסט, תוכנה, תכונות מולטימדיה

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

## בסיס הנתונים

***תרשים DSD:***

***טבלאות ראשיות:***

*טבלת users(משתמשים)*

*תפקיד הטבלה: לנהל מידע על משתמשי המערכת - מנהלים, מאמנים וכדומה.*

| *שם שדה* | *הסבר* |
| --- | --- |
| *id* | *מזהה ייחודי למשתמש* |
| *FullName* | *שם מלא של המשתמש* |
| *Username* | *שם משתמש להתחברות למערכת* |
| *UserPassword* | *סיסמה מוצפנת להתחברות* |
| *Email* | *כתובת דואר אלקטרוני של המשתמש* |
| *team* | *מזהה קבוצה אליה שייך המשתמש* |
| *admin* | *האם המשתמש הוא מנהל מערכת (ערך בוליאני)* |
| *Role* | *תפקיד המשתמש (כגון: מאמן, עוזר מאמן, מנהל מקצועי וכו')* |

*טבלת Team(קבוצה)*

*תפקיד הטבלה: ייצוג קבוצות כדורגל במערכת.*

| *שם שדה* | *הסבר* |
| --- | --- |
| *id* | *מזהה ייחודי לקבוצה* |
| *Team* | *שם הקבוצה* |

*טבלת Players(שחקנים)*

*תפקיד הטבלה: לשמור מידע אישי ומקצועי על כל שחקן.*

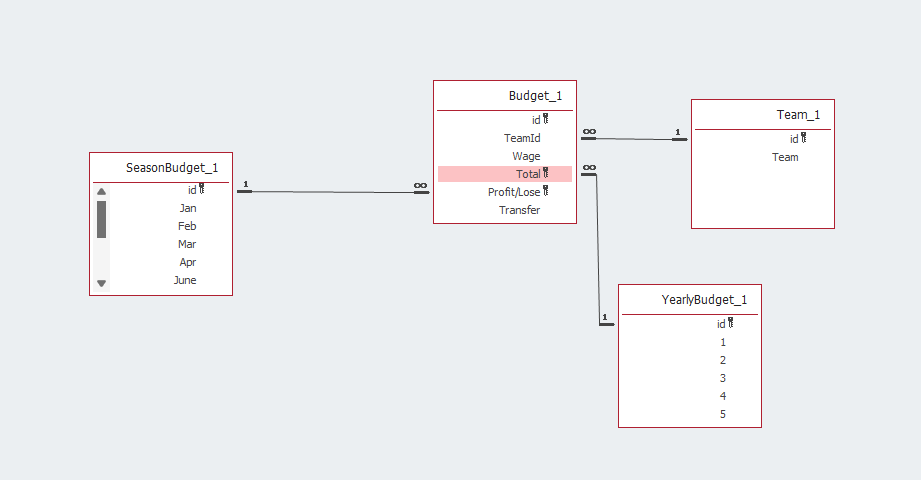
| *שם שדה* | *הסבר* |
| --- | --- |
| *id* | *מזהה ייחודי לשחקן* |
| *full\_name* | *שם מלא של השחקן* |
| *Nationality* | *לאום של השחקן* |
| *dob* | *תאריך לידה* |
| *team* | *מזהה קבוצה בה השחקן משחק* |
| *UserValue* | *ערך שוק נוכחי של השחקן* |
| *wage* | *שכר חודשי או שנתי של השחקן* |
| *height* | *גובה השחקן בס"מ* |
| *weight* | *משקל השחקן בק"ג* |
| *foot* | *רגל דומיננטית (ימין/שמאל)* |
| *position* | *תפקיד במגרש (שוער, בלם, קשר, חלוץ וכו')* |

*טבלת Budget(תקציב)*

*תפקיד הטבלה: ניהול תקציב והוצאות/הכנסות של קבוצות.*

| *שם שדה* | *הסבר* |
| --- | --- |
| *id* | *מזהה ייחודי לרישום תקציב* |
| *TeamId* | *מזהה קבוצה (מקושר לטבלת Team)* |
| *Wage* | *סך כל תשלומי השכר* |
| *Total* | *סכום כולל של התקציב* |
| *Profit/Lose* | *רווח או הפסד כספי* |
| *Transfer* | *תקציב שהוקצה להעברות שחקנים* |

***טבלאות קישור:***

***רבים לרבים:  
***

החיבור בין התקציבים לעונות מתבצע באמצעות השדות BudgetId ו־SeasonId, אשר מזהים כל תקציב ועונה בהתאמה. כל שורה מייצגת שיוך מסוים של תקציב לעונה, כאשר ייתכן שתקציב אחד ישויך למספר עונות שונות, ועונה אחת תכיל מספר תקציבים.

הקשר מתבסס על הטבלאות Budget\_1, האחראית על ניהול נתוני התקציבים, ו־SeasonBudget\_1, המאגדת את פרטי העונות. הקשר ביניהן משקף מצבים בהם חלוקה תקציבית מתבצעת לאורך כמה עונות, או כאשר עונה מסוימת מתוקצבת ממקורות שונים.

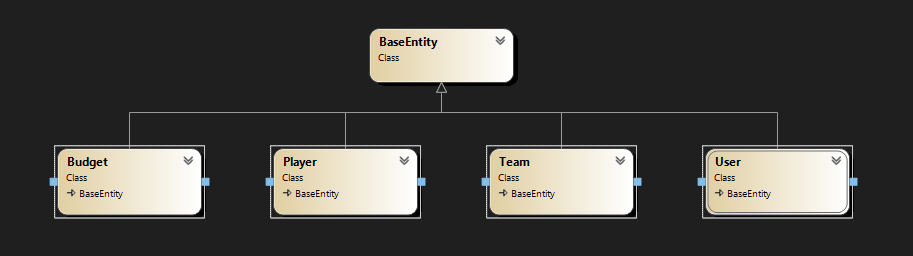
לכל צירוף Budget–Season יש משמעות ייחודית, ולכן לא נדרש מפתח מזהה נוסף***.***

## מימוש הפרויקט

***צד שרת:***

***שכבת ה-Model:***

שכבת ה - Modelאחראית לייצג את הנתונים והישויות המרכזיות של המערכת.

******

**Base entity:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FootballProject.Model

{

public class BaseEntity

{

private int id;

public int Id

{

get { return id; }

set { id = value; }

}

}

}

המחלקה BaseEntity היא מחלקת בסיס (abstract logic base) שמייצגת ישות עם מזהה (ID) ייחודי.

**Player:**

using System;

namespace FootballProject.Model

{

public class Player : BaseEntity

{

private string? fullName;

private string? nationality;

private DateTime dateOfBirth;

private Team team;

private int userValue;

private int wage;

private int height;

private int weight;

private string? foot;

private string? position;

private List<Stat>? stats;

public string FullName { get => fullName; set => fullName = value; }

public string Nationality { get => nationality; set => nationality = value; }

public DateTime DateOfBirth { get => dateOfBirth; set => dateOfBirth = value; }

public Team Team { get => team; set => team = value; }

public int UserValue { get => userValue; set => userValue = value; }

public int Wage { get => wage; set => wage = value; }

public int Height { get => height; set => height = value; }

public int Weight { get => weight; set => weight = value; }

public string Foot { get => foot; set => foot = value; }

public string Position { get => position; set => position = value; }

public List<Stat>? Stats { get => stats; set => stats = value; }

}

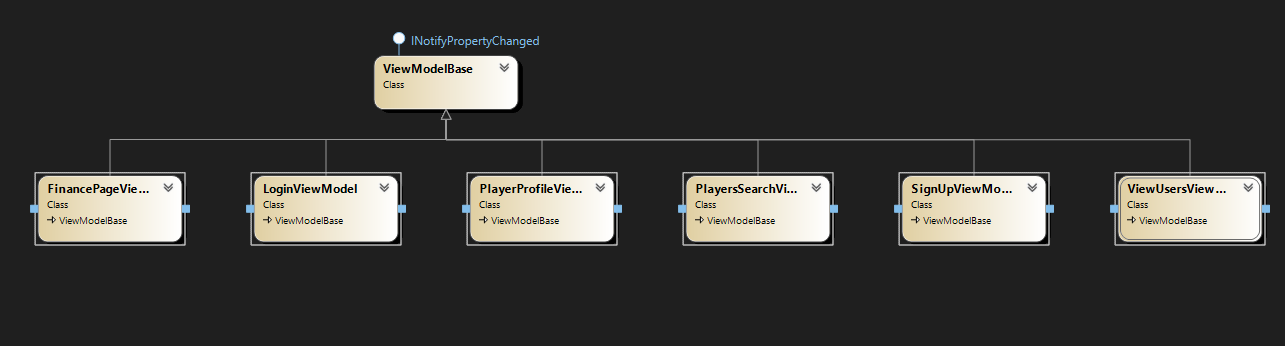
}

המחלקה Player מייצגת שחקן כדורגל ויורשת מהמחלקה BaseEntity, כך שהיא כוללת גם שדה Id.

בפרויקט שלי השתמשתי בעקרון הירושה (Inheritance) מתכנות מונחה עצמים, כדי לייעל את העבודה עם ישויות שונות במערכת. יצרתי מחלקת בסיס בשם BaseEntity, שמכילה מזהה ייחודי (Id), וממנה הורשתי מחלקות נוספות כמו Player, Team, ו־User. כך כל ישות בפרויקט מקבלת מזהה אחיד באופן אוטומטי, בלי צורך להגדיר אותו בכל מחלקה מחדש. הירושה עוזרת לי לשמור על אחידות במבנה הנתונים, לצמצם כפילויות בקוד, ולהפוך את המערכת לגמישה וקלה יותר לתחזוקה.

***שכבת ה-viewmodel:***

שכבת ה- ViewModel מהווה את המוח מאחורי התצוגה באפליקציה, ומיישמת את עקרונות ה־MVVM (Model-View-ViewModel).  
היא אחראית על קישור בין הנתונים (Model) לבין הממשק הגרפי (View), תוך שמירה על הפרדה מלאה בין לוגיקת העסקים להצגת הנתונים.



תמונה שמכילה טקסט, צילום מסך, מלבן, מספר

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

**ClubPlayersSearchViewModel:**

using FootballProject.Model;

using FootballProject.Services;

using FootballProject.ViewModel.DB;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Threading.Tasks;

using System.Windows.Input;

using Microsoft.Maui.Controls;

namespace FootballProject.ViewModel

{

public class ClubPlayersSearchViewModel : ViewModelBase

{

private readonly PlayerDB playerDB;

private readonly IUser userService;

private ObservableCollection<Player> players;

private string selectedField;

private string filterValue;

private string sortOrder;

public ClubPlayersSearchViewModel(IUser service)

{

playerDB = new PlayerDB();

players = new ObservableCollection<Player>();

userService = service;

SelectedField = "Any";

SortOrder = "None";

SearchCommand = new Command(async () => await SearchPlayersAsync());

NavigateToPlayerProfileCommand = new Command<Player>(async (player) => await NavigateToPlayerProfileAsync(player));

}

public ObservableCollection<Player> Players

{

get => players;

set

{

players = value;

OnPropertyChanged();

}

}

public string SelectedField

{

get => selectedField;

set

{

selectedField = value;

OnPropertyChanged();

}

}

public string FilterValue

{

get => filterValue;

set

{

filterValue = value;

OnPropertyChanged();

}

}

public string SortOrder

{

get => sortOrder;

set

{

sortOrder = value;

OnPropertyChanged();

}

}

public ICommand SearchCommand { get; }

public ICommand NavigateToPlayerProfileCommand { get; }

private async Task SearchPlayersAsync()

{

var manager = (userService as UserService)?.GetCurrentUser(); // Only if local UserService

if (manager?.Team == null)

{

await Application.Current.MainPage.DisplayAlert("Error", "No team associated with this user.", "OK");

return;

}

var teamId = manager.Team.Id;

if (!string.IsNullOrWhiteSpace(FilterValue))

{

Players = new ObservableCollection<Player>(

await playerDB.SelectTeamPlayersByFirstName(teamId, FilterValue));

}

else

{

Players = new ObservableCollection<Player>(

await playerDB.SelectPlayersByTeam(teamId));

}

}

private async Task NavigateToPlayerProfileAsync(Player player)

{

var data = new Dictionary<string, object>

{

{ "player", player },

{ "source", "ClubPlayersSearch" }

};

await Shell.Current.GoToAsync("/rPlayerProfile", data);

}

}

}

מחלקת ClubPlayersSearchViewModel אחראית ללוגיקת חיפוש והצגת שחקנים של הקבוצה המשויכת למשתמש, תוך תמיכה בסינון, הצגה וניווט לעמוד הפרופיל של שחקן – בהתאם למבנה ה־MVVM.

***HomePageViewModel:***

using FootballProject.Model;

using FootballProject.Services;

using System.Windows.Input;

using Microsoft.Maui.Controls;

namespace FootballProject.ViewModel

{

public class HomePageViewModel : ViewModelBase

{

private readonly IUser \_userService;

private User user;

public User User

{

get => user;

set

{

if (user != value)

{

user = value;

OnPropertyChanged();

}

}

}

public ICommand LogoutCommand { get; }

public HomePageViewModel(IUser userService)

{

\_userService = userService;

User = (\_userService as UserService)?.GetCurrentUser(); // Cast only if using UserService

LogoutCommand = new Command(async () => await OnLogoutAsync());

}

private async Task OnLogoutAsync()

{

// Optional: clear current user if applicable

if (\_userService is UserService us)

{

us.SetCurrentUser(null);

}

// Navigate to login page

await Shell.Current.GoToAsync("///rLogIn");

}

}

}

המחלקה HomePageViewModel מייצגת את לוגיקת התצוגה של דף הבית באפליקציה, בהתאם למבנה MVVM.

***הרחבות:***

***הליך א-סנכרוני:***

הפונקציה GetAllUsers היא פונקציה אסינכרונית השולחת בקשת GET לשרת במטרה לקבל את כל המשתמשים. לאחר קבלת התגובה, היא ממירה את תוכן ה-JSON לרשימת אובייקטים מסוג User. השימוש במילת המפתח async וב־await מאפשר להריץ את הפעולה ברקע מבלי לחסום את התוכנית הראשית, מה שמייעל ביצועים ומונע קפיאות בממשק המשתמש.

public async Task<List<User>> GetAllUsers()

{

var response = await client.GetAsync($"{baseUrl}/api/users");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<List<User>>(json, options);

}

***שכבת שירותי רשת:***

שכבת שירותי הרשת ממוקמת בתיקיית Services וכוללת את הממשק IUser ואת המימוש שלו WebService.  
הממשק IUser מגדיר את כלל הפעולות הקשורות לשירותי משתמש הפועלות מול השרת, בעוד ש־ WebService מממש את הפעולות האלו בפועל.  
התקשורת בין האפליקציה לשרת נעשית באמצעות פרוטוקול HTTP, תוך סיריאליזציה של הנתונים לפורמט JSON והמרתם חזרה לאובייקטים תואמים בצד הלקוח.

הגישה למסד הנתונים נעשית באמצעות שירות רשת חיצוני (כמו Swagger), אשר מאפשר חיבור דרך Host מרוחק במקום חיבור ישיר ממחשב מקומי.  
בנוסף, נעשה שימוש ב־Dev Tunnel שמספק כתובת רשת גלובלית, המאפשרת לכל מכשיר שבו

מותקנת האפליקציה לגשת למסד הנתונים שבשרת.  
שכבה זו חיונית לתקשורת בזמן אמת בין שני שחקנים, כך שיוכלו לשחק זה מול זה ממכשירים שונים ובמיקומים שונים – ללא צורך להיות על אותו מכשיר.

***Interface:***

הממשק IUser מגדיר את כל הפעולות שניתן לבצע מול השרת עבור ניהול משתמשים, שחקנים וסטטיסטיקות – מבלי לקבוע כיצד בדיוק הן מתבצעות.

using FootballProject.Model;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace FootballProject.Services

{

public interface IUser

{

Task<List<User>> GetAllUsers();

Task<User> GetUser(string username);

Task<User> GetUserByID(int id);

Task<bool> AddUser(User user);

Task<bool> UpdateUser(User user);

Task<bool> DeleteUser(User user);

Task<List<Stat>> GetAllStats(string position, int playerId);

}

}

using FootballProject.Model;

using System;

using System.Collections.Generic;

using System.Net.Http;

using System.Text;

using System.Text.Json;

using System.Text.Json.Serialization;

using System.Threading.Tasks;

namespace FootballProject.Services

{

public class WebService : IUser

{

private readonly HttpClient client;

private readonly JsonSerializerOptions options;

private const string baseUrl = "http://localhost:5026";

public WebService()

{

client = new HttpClient();

options = new JsonSerializerOptions

{

PropertyNameCaseInsensitive = true,

DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull,

WriteIndented = true

};

}

// ---------------------- USER METHODS ----------------------

public async Task<List<User>> GetAllUsers()

{

var response = await client.GetAsync($"{baseUrl}/api/users");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<List<User>>(json, options);

}

public async Task<User> GetUser(string username)

{

var response = await client.GetAsync($"{baseUrl}/api/users/by-username/{username}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<User>(json, options);

}

public async Task<User> GetUserByID(int id)

{

var response = await client.GetAsync($"{baseUrl}/api/users/{id}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<User>(json, options);

}

public async Task<bool> AddUser(User user)

{

var content = new StringContent(JsonSerializer.Serialize(user, options), Encoding.UTF8, "application/json");

var response = await client.PostAsync($"{baseUrl}/api/users", content);

return response.IsSuccessStatusCode;

}

public async Task<bool> UpdateUser(User user)

{

var content = new StringContent(JsonSerializer.Serialize(user, options), Encoding.UTF8, "application/json");

var response = await client.PutAsync($"{baseUrl}/api/users/{user.Id}", content);

return response.IsSuccessStatusCode;

}

public async Task<bool> DeleteUser(User user)

{

var response = await client.DeleteAsync($"{baseUrl}/api/users/{user.Id}");

return response.IsSuccessStatusCode;

}

public async Task InitStats(int id, string position = null)

{

await Task.CompletedTask;

}

public async Task<List<Stat>> GetAllStats(string position, int id)

{

var response = await client.GetAsync($"{baseUrl}/api/stats/{position}/{id}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<List<Stat>>(json, options);

}

// ---------------------- BUDGET METHODS ----------------------

public async Task<Budget> GetBudgetByTeamId(int teamId)

{

var response = await client.GetAsync($"{baseUrl}/api/budgets/team/{teamId}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<Budget>(json, options);

}

public async Task<bool> UpdateBudget(Budget budget)

{

var content = new StringContent(JsonSerializer.Serialize(budget, options), Encoding.UTF8, "application/json");

var response = await client.PutAsync($"{baseUrl}/api/budgets/{budget.TeamId}", content);

return response.IsSuccessStatusCode;

}

public async Task<bool> UpdateYearlyBudget(Budget budget)

{

var content = new StringContent(JsonSerializer.Serialize(budget, options), Encoding.UTF8, "application/json");

var response = await client.PutAsync($"{baseUrl}/api/yearlybudgets/{budget.Total}", content);

return response.IsSuccessStatusCode;

}

public async Task<bool> UpdateSeasonBudget(Budget budget)

{

var content = new StringContent(JsonSerializer.Serialize(budget, options), Encoding.UTF8, "application/json");

var response = await client.PutAsync($"{baseUrl}/api/seasonbudgets/{budget.ProfitLose}", content);

return response.IsSuccessStatusCode;

}

public async Task SaveChanges()

{

await Task.CompletedTask; // No-op in web context

}

// ---------------------- PLAYER METHODS ----------------------

public async Task<List<Player>> SelectAllPlayers()

{

var response = await client.GetAsync($"{baseUrl}/api/players");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<List<Player>>(json, options);

}

public async Task<List<Player>> SelectPlayersByTeam(int teamId)

{

var response = await client.GetAsync($"{baseUrl}/api/players/team/{teamId}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<List<Player>>(json, options);

}

public async Task<List<Player>> SelectTeamPlayersByFirstName(int teamId, string name)

{

var response = await client.GetAsync($"{baseUrl}/api/players/team/{teamId}/filter?name={name}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<List<Player>>(json, options);

}

public async Task<List<Player>> SelectByFilter(string field, string value)

{

var response = await client.GetAsync($"{baseUrl}/api/players/filter?field={field}&value={value}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<List<Player>>(json, options);

}

public async Task<List<Player>> SelectAndSort(string field, string order)

{

var response = await client.GetAsync($"{baseUrl}/api/players/sort?field={field}&order={order}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<List<Player>>(json, options);

}

}

}

***שכבת BusinessLogic:***

שכבת **Business Logic** (לוגיקת עסקית) מהווה את מוח הפעולה של האפליקציה, כאשר היא אחראית על עיבוד נתונים, ביצוע חישובים, הפעלת כללים פונקציונליים ושמירה על עקביות המידע, בהתאם לדרישות המערכת. בדף FinancePageViewModel, לוגיקת העסקים מתמקדת בניהול תקציב קבוצתי, חישוב איזון תקציבי בין שכר לרכש, ניתוח חודשי ושנתי והצגת נתונים למאמן בזמן אמת.

**טעינת תקציב קיים**

בעת פתיחת העמוד, מופעלת הפונקציה LoadBudgetData, אשר טוענת את נתוני התקציב מהמסד באמצעות BudgetDB. הטעינה מתבצעת לפי מזהה הקבוצה של המשתמש הנוכחי. אם נמצאו נתונים, הערכים כגון תקציב שכר (Wage), תקציב רכש (TransferBudget), יתרה בבנק (BankBalance) ורווח/הפסד עונתי (ProfitOrLoss) נטענים למשתנים מקומיים.

במקרה שלא קיימים נתונים – הערכים מאותחלים ל־0, ואם מתרחשת שגיאה – הערכים מאותחלים ל־-1.

**איזון התקציב לפי אחוזים – חישוב מתמטי**

אחד המאפיינים המרכזיים בעמוד הוא היכולת **לאזן תקציב** בין שכר לבין תקציב העברות, באמצעות רכיב גרפי מסוג Slider. המשתמש מזיז את הסליידר בין 0 ל־100 כדי לשלוט באחוז מתוך התקציב הכולל שיוקצה לרכישות שחקנים.

TransferBudget = (int)(totalBudget \* ((float)sliderValue / 100f));

Wage = totalBudget - TransferBudget;

דוגמה: אם התקציב הכולל הוא 1,000,000 € והמשתמש בחר 30% לרכש:

* תקציב רכש = 300,000 €
* תקציב שכר = 700,000 €

השיטה הזו מאפשרת גמישות רבה בניהול משאבים, ומדמה מצבי קבלת החלטות אמיתיים של מאמן או מנהל מקצועי.

**גרפים שנתיים וחודשיים – ניתוח חזותי**

לצורך תצוגה נוחה וברורה של התקציב לאורך זמן, מוצגים שני גרפים באמצעות ספריית **Microcharts**:

**גרף שנתי (LoadYearlyChart)**

* מציג נתוני תקציב לחמש השנים הקרובות (One עד Five).
* כל עמודה מייצגת שנה עם צבע שונה, לצורך ניתוח מגמות עתידיות.

**גרף חודשי (LoadMonthlyChart)**

* מציג נתוני תקציב לכל אחד מחודשי השנה (ינואר–דצמבר).
* מאפשר מעקב אחר תנודות פיננסיות במהלך העונה, כמו חודשים של רכש, שכר מוגבר או חיסכון.

שני הגרפים מספקים למשתמש ראייה רחבה ואינטואיטיבית של מצב התקציב.

**תצוגת נתונים מותאמת**

מחרוזות ייעודיות מציגות את הנתונים הכלכליים בפורמט קריא וברור, לדוגמה:

public string BankBalanceString => $"Overall Bank Balance: {BankBalance:N0} €";

שימוש ב־:N0 מוסיף הפרדת אלפים, ותוספת סימן המטבע (€) משדר מקצועיות.  
כך המשתמש רואה: Overall Bank Balance: 1,500,000 € במקום 1500000.

**שמירת שינויים – עדכון במסד**

הפונקציה SaveBudgets מאחדת את כל הנתונים שנשמרו בזיכרון ומשדרת אותם חזרה לשלוש טבלאות במסד הנתונים:

* BudgetDB – תקציב כולל.
* YearlyBudgetDB – תקציב חמש-שנתי.
* SeasonBudgetDB – תקציב חודשי/עונתי.

העדכון מתבצע בצורה אסינכרונית, בשימוש בטכנולוגיות כמו HTTP ו־JSON, כדי להבטיח תקשורת חלקה עם השרת ותגובה מהירה למשתמש.

**צד לקוח:**

האפליקציה נבנתה בטכנולוגיית **.NET MAUI**, המאפשרת יצירת אפליקציה אחת הפועלת על מגוון פלטפורמות: דסקטופ (Windows), אנדרואיד ואפילו כדף אינטרנט בהרחבות עתידיות.  
ממשקי המשתמש עוצבו מתוך מטרה להיות **נוחים, ברורים ונגישים**, עם דגש על חוויית שימוש פשוטה גם למשתמשים ללא רקע טכנולוגי.

במסכים השונים נעשה שימוש בטכניקות של **בדיקת תקינות קלט** – לדוגמה: אימות שדות התחברות, חיפוש שחקנים לפי טקסט, ואזהרות במקרים של שדות ריקים או קלט לא תקין.  
בנוסף, האינטראקציה עם המשתמש מבוססת על **אירועים ופקודות (Commands)** – כך שלחיצה על כפתור מבצעת פעולה ב־ViewModel בצורה מודולרית ומופרדת מהתצוגה.

**תמיכה בסוגי משתמשים:**

האפליקציה תומכת **בלפחות שני סוגי משתמשים**:

* **מאמן** – יכול לראות נתונים על שחקנים, לבחון סטטיסטיקות, לעצב הרכב ולבצע חיפושים מתקדמים.
* **מנהל קבוצה** – שולט במידע הפיננסי של הקבוצה, מנהל תקציבים ומבצע עדכונים מול בסיס הנתונים.

לכל משתמש מוצגות **אפשרויות מותאמות** לפי הרשאותיו, ונעשה שימוש בתפריטים דינמיים וממשקי ניהול שונים בהתאם לתפקיד.

**תיאור הדפים/מסכים:**

* **- LogInPage** – קלט לשם משתמש + תפקיד, כולל בדיקת תקינות.
* **HomePage** – מציג תפריט מותאם אישית לפי סוג המשתמש.
* **ClubPlayersSearch** – חיפוש שחקנים, סינון לפי שדות, והצגה ברשימה.
* **PlayerProfile** – תצוגת נתונים אישיים וסטטיסטיים של השחקן.
* **FinancePage** – מסך לניהול תקציב שנתי וחודשי, כולל גרפים אינטראקטיביים, סליידר לאיזון תקציב בין שכר לרכש, ופקודת שמירה מול המסד.

**ירושה ממחלקות שכתבתי:**

המערכת עושה שימוש בירושה כדי לארגן את מבנה הישויות.  
למשל, כל המחלקות Player, User, Team, Budget יורשות מהמחלקה BaseEntity שמכילה את השדה Id.  
באמצעות הירושה, נוצרה אחידות בין הישויות ומבנה קוד שקל להרחיב ולתחזק.

**שימוש ב־:Delegate**

בפרויקט יישמתי Delegate ייעודי בשם CreateOleDb, אשר מקבל BaseEntity ומחזיר מחרוזת.  
ה־Delegate משמש ליצירת שאילתות SQL מותאמות לכל ישות – ומאפשר שליטה בהתנהגות הדינמית של פעולות מול בסיס הנתונים.  
השימוש ב־Delegate מופרד למחלקה ChangeEntity, אשר מאגדת ישות יחד עם הפונקציה המתאימה לה, בהתאם להקשר.

כך לדוגמה, אפשר ליצור פעולת Insert או Update שונה לכל ישות, מבלי לכתוב תנאים בתוך הפונקציה עצמה – אלא ע"י הזרקת הפונקציה המתאימה בזמן ריצה.

**קוד אסינכרוני:**

בפרויקט נעשה שימוש נרחב בקוד אסינכרוני (async/await) למטרות כמו:

* שליפת נתונים מהשרת (למשל GetAllUsers, SelectPlayersByTeam)
* טעינת גרפים ועדכון תקציב בזמן אמת
* ניווט בין מסכים תוך כדי שמירה על תגובתיות מלאה של האפליקציה

הודות לכך, הממשק נשאר מהיר, ולא נתקע גם כאשר מתבצעות פעולות רקע כמו טעינת נתונים או שמירה למסד.

**ריבוי פלטפורמות:**

באמצעות טכנולוגיית **MAUI**, האפליקציה ניתנת להפעלה על מכשירי Windows ו־Android עם אותו בסיס קוד.  
בעתיד, ניתן יהיה להרחיב את התמיכה גם ל־ iOS ודפדפנים (Web), מבלי לשכתב את הלוגיקה העסקית או ממשקי המשתמש.

## 

## מדריך למשתמש

## האפליקציה לניהול קבוצת כדורגל נועדה לשמש כלי עבודה חכם, אינטואיטיבי ויעיל עבור מאמנים ו־מנהלי קבוצות מקצועניות. הממשק פשוט ונוח, וכל משתמש רואה רק את האפשרויות הרלוונטיות

## לתפקידו. מדריך זה מתאר את השימוש באפליקציה – מרגע הכניסה ועד הפעולות האפשריות בכל מסך.

## גרסאות עליהן נבדק הפרויקט

## Windows 10 – אפליקציית שולחן עבודה (MAUI).

## Android 13 – אמולטור Pixel 5 באורך מסך סטנדרטי.

## רזולוציות שנבדקו 1366x768, 1920x1080.

## שלב ההתחברות למערכת

## מסך: LoginPage

## בעת פתיחת האפליקציה מוצג מסך התחברות עם שדות:

## שם משתמש (ללא מגבלת תווים אך חובה להזין)

## סיסמה (שדה מוגן)

## לחיצה על כפתור "התחבר" מפעילה בדיקת תקינות.

## אם אחד השדות ריק – תופיע הודעת שגיאה

## אם ההתחברות מצליחה:

## המשתמש מזוהה כמנהל או מאמן ומופנה למסך הבית.

## לאחר ההתחברות – ניווט בין מסכים (Use Case כללי)

## המשתמש נכנס למערכת

## מזהה את תפקידו (מאמן/מנהל)

## מופנה ל־HomePage

## אם המשתמש הוא מנהל הוא יוכל להיכנס לדף תקציב ולדף רכש, אם המשתמש הוא מאמן הוא יכול להיכנס לכל הדפים(לא יכול לערוך תקציב) ואם הוא אדמין מופיע לו גם הדף שמאפשר למחוק, לעדכן וליצור משתמשים

## עבור משתמש מסוג: מאמן

## מסך: HomePage

* צפייה בתקציב
* צפייה בשחקנים בקבוצה
* חיפוש רכש

## מסך : ClubPlayersSearch

## מוצגת רשימת שחקנים בקבוצה.

## חיפוש לפי:

## שם פרטי

## בחירה בשחקן תפתח את מסך פרופיל השחקן.

## מסך : PlayerProfile

## מציג:

## שם מלא

## גיל

## גובה, משקל, רגל מועדפת, שווי, שכר, קבוצה, לאום

## סטטיסטיקות (שערים, בישולים וכו’)

## עבור משתמש מסוג: מנהל

## מסך: HomePage

## מוצגות אפשרויות:

## ניהול תקציב

## חיפוש רכש

## מסך : FinancePage

## מציג:

## יתרת בנק

## רווח/הפסד עונתי

## תקציב העברות ושכר

## ניתן:

## להחליף בין תקציב רכש ל־שכר שחקנים באמצעות Slider

## לצפות בגרפים:

## תקציב חודשי (ינואר עד דצמבר)

## תקציב שנתי (עד 5 שנים)

## לשמור שינויים מול המסד

צילומי מסך נמצאים בתחילת הספר.

## 

## רפלקציה/סיכום אישי

**רפלקציה/סיכום אישי**

פיתוח האפליקציה לניהול קבוצת כדורגל היה עבורי מסע אישי ומקצועי משמעותי. מעבר לרכישת כלים טכניים חדשים, נדרשתי ליכולות ניהול עצמי, התמדה ופתרון בעיות יצירתי.

**אתגרים מרכזיים:**

במהלך העבודה התמודדתי עם מספר אתגרים חשובים. ראשית, בניית ארכיטקטורה נכונה שמבוססת על עקרונות ה-MVVM דרשה תכנון מוקפד והבנה עמוקה יותר של חלוקת אחריות בקוד. שנית, ניהול הרשאות וגישה מבוססת תפקידים חייבה אותי להעמיק בלוגיקה של בקרת גישה, אבטחת מידע, והבנת חוויית משתמש (UX) עבור סוגים שונים של משתמשים (מאמן מול מנהל).

בנוסף, האתגר של בניית אלגוריתם לבחירת הרכב אופטימלי, על סמך נתוני שחקנים סטטיסטיים, אילץ אותי לשלב חשיבה אלגוריתמית עם שיקולים מעשיים של קלות שימוש ומהירות חישוב.

**למידה והתפתחות:**

במהלך הפיתוח העמקתי את הידע שלי בשפות כמו C#, SQL ו-XAML, ושיפרתי את היכולת שלי להפריד בין שכבות שונות בפרויקט, בצורה שתשמור על קוד נקי, קריא ומודולרי. כמו כן, למדתי לעבוד בסביבת Visual Studio בצורה מעמיקה יותר, לנצל כלים מתקדמים כמו ניפוי שגיאות (Debugging) ולבצע אופטימיזציה לקוד.

**תובנות עיקריות:**

* **חשיבות התכנון המוקדם**: בניית מסמך ייזום ברור ותרשימי ERD ו-DFD סייעו לי להבין את המערכת טוב יותר לפני תחילת הפיתוח, ולחסוך טעויות בשלב מאוחר יותר.
* **פשטות היא כוח**: גיליתי שלפעמים עדיף להעדיף פתרונות פשוטים אך יציבים מאשר לנסות לבנות פונקציות מורכבות מדי שלא בהכרח תורמות לחוויית המשתמש.
* **לימוד עצמי תוך כדי עבודה**: נאלצתי ללמוד טכנולוגיות חדשות באופן עצמאי, תהליך שחיזק בי את הביטחון להתמודד עם נושאים לא מוכרים.

**חוזקות וחולשות שזיהיתי:**

* **חוזקות**: ראייה מערכתית, ירידה לפרטים קטנים, התמדה ויכולת למידה מהירה.
* **חולשות**: נטייה לפרפקציוניזם שגרמה לעיתים להאריך משימות מעבר למה שנדרש בפועל; צורך בשיפור נוסף בניהול זמן בפרויקטים גדולים.

**מחשבות לעתיד:**

לו הייתה לי אפשרות להמשיך לפתח את הפרויקט, הייתי מוסיף:

* עדכוני נתונים בזמן אמת (Live Updates) באמצעות WebSocket או SignalR.
* הרחבת מנגנוני האבטחה, כולל אימות דו-שלבי (2FA).

**סיכום אישי:**

בסיום הפרויקט אני מרגיש שסגרתי מעגל שלם - מהרעיון הראשוני דרך האפיון, הבנייה ועד להשלמת מוצר שלם ומתפקד. זו הייתה חוויה מעשירה ביותר שפיתחה אצלי יכולות שלא הייתי מגיע אליהן במסגרת לימודית עיונית בלבד. אני גאה בתוצאה שהצלחתי להגיע אליה, ורואה בפרויקט הזה אבן דרך משמעותית בדרכי המקצועית לעולם פיתוח המערכות.

.

## ביבליוגרפיה

כל החומר נלקח מהמצגות בקלאסרום ומהאתר של מייקרוסופט שמכיל את התיעוד הרשמי של הטכנולוגיה ASP .NET.

[learn.microsoft.com/en-us/dotnet/framework/](file:///C:\Users\אבי\Downloads\learn.microsoft.com\en-us\dotnet\framework\)

## נספחים

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FootballProject.Model

{

public class BaseEntity

{

private int id;

public int Id

{

get { return id; }

set { id = value; }

}

}

}

namespace FootballProject.Model

{

public class Budget : BaseEntity

{

// From Budget table

public int Id { get; set; }

public int TeamId { get; set; }

public long Wage { get; set; }

public int Total { get; set; }

public int ProfitLose { get; set; }

public long Transfer { get; set; }

// Foreign keys (needed for JOINs)

public int YearId { get; set; }

public int SeasonId { get; set; }

// From YearlyBudget table (fields named [1] to [5] in Access)

public int One { get; set; }

public int Two { get; set; }

public int Three { get; set; }

public int Four { get; set; }

public int Five { get; set; }

// From SeasonBudget table (monthly data)

public int Jan { get; set; }

public int Feb { get; set; }

public int Mar { get; set; }

public int Apr { get; set; }

public int June { get; set; }

public int July { get; set; }

public int Aug { get; set; }

public int Sep { get; set; }

public int Oct { get; set; }

public int Nov { get; set; }

public int Dec { get; set; }

}

}

using FootballProject.Model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FootballProject.Model

{

public delegate string CreateOleDb(BaseEntity entity);

public class ChangeEntity

{

private BaseEntity entity;

private CreateOleDb createOleDb;

public ChangeEntity(CreateOleDb createOleDb, BaseEntity entity)

{

this.createOleDb = createOleDb;

this.entity = entity;

}

public BaseEntity Entity { get => entity; set => entity = value; }

public CreateOleDb CreateOleDb { get => createOleDb; set => createOleDb = value; }

}

}

using System;

namespace FootballProject.Model

{

public class Player : BaseEntity

{

private string? fullName;

private string? nationality;

private DateTime dateOfBirth;

private Team team;

private int userValue;

private int wage;

private int height;

private int weight;

private string? foot;

private string? position;

private List<Stat>? stats;

public string FullName { get => fullName; set => fullName = value; }

public string Nationality { get => nationality; set => nationality = value; }

public DateTime DateOfBirth { get => dateOfBirth; set => dateOfBirth = value; }

public Team Team { get => team; set => team = value; }

public int UserValue { get => userValue; set => userValue = value; }

public int Wage { get => wage; set => wage = value; }

public int Height { get => height; set => height = value; }

public int Weight { get => weight; set => weight = value; }

public string Foot { get => foot; set => foot = value; }

public string Position { get => position; set => position = value; }

public List<Stat>? Stats { get => stats; set => stats = value; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FootballProject.Model

{

public class Stat : BaseEntity

{

private string name;

private int value;

public string Name { get => name; set => name = value; }

public int Value { get => value; set => this.value = value; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FootballProject.Model

{

public class Team : BaseEntity

{

private string? team;

public string? team1 { get => team; set => team = value; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FootballProject.Model

{

public class User : BaseEntity

{

private string? name;

private string? username;

private string? password;

private string? email;

private Team team;

private string? isAdmin;

private string? role;

public string Name { get => name; set => name = value; }

public string? Email { get => email; set => email = value; }

public string? Username { get => username; set => username = value; }

public string? Password { get => password; set => password = value; }

public Team Team { get => team; set => team = value; }

public string? IsAdmin { get => isAdmin; set => isAdmin = value; }

public string? Role { get => role; set => role = value; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FootballProject.Model

{

public class UserList : List<User>

{

public UserList() { }

public UserList(IEnumerable<User> list) : base(list) { }

public UserList(IEnumerable<BaseEntity> list) : base(list.Cast<User>().ToList()) { }

}

}

using FootballProject.Model;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace FootballProject.Services

{

public interface IUser

{

Task<List<User>> GetAllUsers();

Task<User> GetUser(string username);

Task<User> GetUserByID(int id);

Task<bool> AddUser(User user);

Task<bool> UpdateUser(User user);

Task<bool> DeleteUser(User user);

Task<List<Stat>> GetAllStats(string position, int playerId);

public Task initUsers();

}

}

using FootballProject.Model;

using System;

using System.Collections.Generic;

using System.Net.Http;

using System.Text;

using System.Text.Json;

using System.Text.Json.Serialization;

using System.Threading.Tasks;

namespace FootballProject.Services

{

public class WebService : IUser

{

private readonly HttpClient client;

private readonly JsonSerializerOptions options;

private const string baseUrl = "http://localhost:5026";

public WebService()

{

client = new HttpClient();

options = new JsonSerializerOptions

{

PropertyNameCaseInsensitive = true,

DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull,

WriteIndented = true

};

}

// ---------------------- USER METHODS ----------------------

public async Task<List<User>> GetAllUsers()

{

var response = await client.GetAsync($"{baseUrl}/api/users");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<List<User>>(json, options);

}

public async Task<User> GetUser(string username)

{

var response = await client.GetAsync($"{baseUrl}/api/users/by-username/{username}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<User>(json, options);

}

public async Task<User> GetUserByID(int id)

{

var response = await client.GetAsync($"{baseUrl}/api/users/{id}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<User>(json, options);

}

public async Task<bool> AddUser(User user)

{

var content = new StringContent(JsonSerializer.Serialize(user, options), Encoding.UTF8, "application/json");

var response = await client.PostAsync($"{baseUrl}/api/users", content);

return response.IsSuccessStatusCode;

}

public async Task<bool> UpdateUser(User user)

{

var content = new StringContent(JsonSerializer.Serialize(user, options), Encoding.UTF8, "application/json");

var response = await client.PutAsync($"{baseUrl}/api/users/{user.Id}", content);

return response.IsSuccessStatusCode;

}

public async Task<bool> DeleteUser(User user)

{

var response = await client.DeleteAsync($"{baseUrl}/api/users/{user.Id}");

return response.IsSuccessStatusCode;

}

public async Task initUsers()

{

await Task.CompletedTask;

}

public async Task InitStats(int id, string position = null)

{

await Task.CompletedTask;

}

public async Task<List<Stat>> GetAllStats(string position, int id)

{

var response = await client.GetAsync($"{baseUrl}/api/stats/{position}/{id}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<List<Stat>>(json, options);

}

// ---------------------- BUDGET METHODS ----------------------

public async Task<Budget> GetBudgetByTeamId(int teamId)

{

var response = await client.GetAsync($"{baseUrl}/api/budgets/team/{teamId}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<Budget>(json, options);

}

public async Task<bool> UpdateBudget(Budget budget)

{

var content = new StringContent(JsonSerializer.Serialize(budget, options), Encoding.UTF8, "application/json");

var response = await client.PutAsync($"{baseUrl}/api/budgets/{budget.TeamId}", content);

return response.IsSuccessStatusCode;

}

public async Task<bool> UpdateYearlyBudget(Budget budget)

{

var content = new StringContent(JsonSerializer.Serialize(budget, options), Encoding.UTF8, "application/json");

var response = await client.PutAsync($"{baseUrl}/api/yearlybudgets/{budget.Total}", content);

return response.IsSuccessStatusCode;

}

public async Task<bool> UpdateSeasonBudget(Budget budget)

{

var content = new StringContent(JsonSerializer.Serialize(budget, options), Encoding.UTF8, "application/json");

var response = await client.PutAsync($"{baseUrl}/api/seasonbudgets/{budget.ProfitLose}", content);

return response.IsSuccessStatusCode;

}

public async Task SaveChanges()

{

await Task.CompletedTask; // No-op in web context

}

// ---------------------- PLAYER METHODS ----------------------

public async Task<List<Player>> SelectAllPlayers()

{

var response = await client.GetAsync($"{baseUrl}/api/players");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<List<Player>>(json, options);

}

public async Task<List<Player>> SelectPlayersByTeam(int teamId)

{

var response = await client.GetAsync($"{baseUrl}/api/players/team/{teamId}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<List<Player>>(json, options);

}

public async Task<List<Player>> SelectTeamPlayersByFirstName(int teamId, string name)

{

var response = await client.GetAsync($"{baseUrl}/api/players/team/{teamId}/filter?name={name}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<List<Player>>(json, options);

}

public async Task<List<Player>> SelectByFilter(string field, string value)

{

var response = await client.GetAsync($"{baseUrl}/api/players/filter?field={field}&value={value}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<List<Player>>(json, options);

}

public async Task<List<Player>> SelectAndSort(string field, string order)

{

var response = await client.GetAsync($"{baseUrl}/api/players/sort?field={field}&order={order}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<List<Player>>(json, options);

}

}

}

<?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"

xmlns:vm="clr-namespace:FootballProject.ViewModel"

x:Class="FootballProject.View.ClubPlayersSearch"

Title="ClubPlayersSearch"

BackgroundColor="#1E1E1E">

<Grid Padding="30">

<!-- Define Rows for each filter, Search Bar and Results -->

<Grid.RowDefinitions>

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

<RowDefinition Height="10\*" />

</Grid.RowDefinitions>

<!-- Define Columns for Labels and Inputs -->

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto" />

<ColumnDefinition Width="\*" />

</Grid.ColumnDefinitions>

<!-- Filter Selection -->

<Label Grid.Row="0" Grid.Column="0" Text="Select Field:" TextColor="White" VerticalOptions="Center" Margin="5"/>

<Picker Grid.Row="0" Grid.Column="1" x:Name="FilterPicker" TextColor="White" BackgroundColor="#2E2E2E" Margin="5"

SelectedItem="{Binding SelectedField, Mode=TwoWay}">

<Picker.ItemsSource>

<x:Array Type="{x:Type x:String}">

<x:String>Any</x:String>

<x:String>full\_name</x:String>

</x:Array>

</Picker.ItemsSource>

</Picker>

<!-- Sort Selection -->

<Label Grid.Row="1" Grid.Column="0" Text="Sort By:" TextColor="White" VerticalOptions="Center" Margin="5"/>

<Picker Grid.Row="1" Grid.Column="1" x:Name="SortPicker" TextColor="White" BackgroundColor="#2E2E2E" Margin="5"

SelectedItem="{Binding SortOrder, Mode=TwoWay}">

<Picker.ItemsSource>

<x:Array Type="{x:Type x:String}">

<x:String>None</x:String>

<x:String>Asc</x:String>

<x:String>Desc</x:String>

</x:Array>

</Picker.ItemsSource>

</Picker>

<!-- Filter Input -->

<Label Grid.Row="2" Grid.Column="0" Text="Enter Value:" TextColor="White" VerticalOptions="Center" Margin="5"/>

<Entry Grid.Row="2" Grid.Column="1" x:Name="FilterValue" Placeholder="Enter value" TextColor="White" BackgroundColor="#2E2E2E" Margin="5"

Text="{Binding FilterValue, Mode=TwoWay}"/>

<!-- Search Button -->

<Button Grid.Row="3" Grid.ColumnSpan="2" Text="Search" BackgroundColor="#8294C4" TextColor="White" Margin="5"

Command="{Binding SearchCommand}"/>

<!-- Search Results -->

<Frame Grid.Row="4" Grid.ColumnSpan="2" CornerRadius="30" BackgroundColor="#8294C4"

Margin="20">

<ScrollView>

<CollectionView ItemsSource="{Binding Players}">

<CollectionView.ItemTemplate>

<DataTemplate>

<Frame CornerRadius="15" BackgroundColor="#1E1E1E">

<Frame.GestureRecognizers>

<TapGestureRecognizer Command="{Binding Source={RelativeSource AncestorType={x:Type vm:ClubPlayersSearchViewModel}}, Path=NavigateToPlayerProfileCommand}" CommandParameter="{Binding .}" />

</Frame.GestureRecognizers>

<StackLayout Orientation="Vertical">

<StackLayout Orientation="Horizontal">

<Label Text="Full Name: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding FullName}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Nationality: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding Nationality}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Date of Birth: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding DateOfBirth}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Team: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding Team.team1}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="User Value: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding UserValue}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Wage: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding Wage}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Height: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding Height}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Weight: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding Weight}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Foot: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding Foot}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Position: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding Position}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

</StackLayout>

</Frame>

</DataTemplate>

</CollectionView.ItemTemplate>

</CollectionView>

</ScrollView>

</Frame>

</Grid>

</ContentPage>

<?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"

xmlns:charts="clr-namespace:Microcharts.Maui;assembly=Microcharts.Maui"

x:Class="FootballProject.View.FinancePage"

Title="Finance"

BackgroundColor="#1E1E1E">

<Grid Padding="30">

<Grid.RowDefinitions>

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="500" />

</Grid.ColumnDefinitions>

<!-- Bank Balance + Monthly Chart -->

<Frame Grid.ColumnSpan="2" Grid.Row="0" Grid.Column="0" CornerRadius="30" BackgroundColor="#8294C4"

HorizontalOptions="FillAndExpand" VerticalOptions="FillAndExpand" Margin="20">

<VerticalStackLayout Padding="20">

<Label Text="{Binding BankBalanceString}"

TextColor="White" FontSize="Large" FontAttributes="Bold"

HorizontalTextAlignment="Start" />

<charts:ChartView Chart="{Binding YearlyChart}"

HeightRequest="300"

WidthRequest="1200" />

</VerticalStackLayout>

</Frame>

<!-- Profit/Loss + Yearly Chart -->

<Frame Grid.ColumnSpan="2" Grid.Row="1" Grid.Column="0" CornerRadius="30" BackgroundColor="#8294C4"

HorizontalOptions="FillAndExpand" VerticalOptions="FillAndExpand" Margin="20">

<VerticalStackLayout Padding="20">

<Label Text="{Binding ProfitOrLossString}"

TextColor="White" FontSize="Large" FontAttributes="Bold"

HorizontalTextAlignment="Start" />

<charts:ChartView Chart="{Binding MonthlyChart}"

HeightRequest="300"

WidthRequest="1200" />

</VerticalStackLayout>

</Frame>

<!-- Right Side Frame: Transfer and Wage in 2 Sections -->

<Frame Grid.Row="0" Grid.RowSpan="2" Grid.Column="2" CornerRadius="30" BackgroundColor="#8294C4"

HorizontalOptions="FillAndExpand" VerticalOptions="FillAndExpand" Margin="20">

<Grid RowDefinitions="\*,\*">

<!-- Transfer Section -->

<VerticalStackLayout Grid.Row="0" Spacing="10" Padding="20,30,20,10">

<Label Text="Transfer Budget:" TextColor="White" FontSize="Large" FontAttributes="Bold"/>

<Label Text="{Binding TransferBudgetString}" TextColor="White" FontSize="Large"/>

<Label Text="Wage:" TextColor="White" FontSize="Large" FontAttributes="Bold"/>

<Label Text="{Binding WageString}" TextColor="White" FontSize="Large"/>

<!-- Single Slider to adjust percentage of Transfer and Wage -->

<Slider Minimum="0" Maximum="100"

Value="{Binding SliderValue, Mode=TwoWay}"

ThumbColor="White"

MinimumTrackColor="#FF8000"

MaximumTrackColor="Gray"

HorizontalOptions="FillAndExpand"/>

</VerticalStackLayout>

<Button Text="Save" BackgroundColor="#1E1E1E" TextColor="White" HorizontalOptions="Center" VerticalOptions="End"

Command="{Binding SaveCommand}" Margin="20"/>

</Grid>

</Frame>

</Grid>

</ContentPage>

<?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="FootballProject.View.HomePage"

xmlns:model="clr-namespace:FootballProject.Model;assembly=FootballProject"

xmlns:viewmodel="clr-namespace:FootballProject.ViewModel;assembly=FootballProject"

Title="HomePage"

BackgroundColor="#1E1E1E">

<Grid Margin="10">

<!-- Define two rows, one for the content and one for the logout button -->

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<!-- Log Out Button on the top right -->

<Button Grid.Row="0"

BackgroundColor="#8294C4"

TextColor="White"

FontSize="Medium"

Text="Log Out"

HeightRequest="35"

CornerRadius="10"

Margin="5"

Command="{Binding LogoutCommand}"

HorizontalOptions="End"

VerticalOptions="Start" />

<!-- User Information displayed below the logout button -->

<ScrollView Grid.Row="1">

<StackLayout Spacing="20" Padding="10">

<Label Text="User Information"

FontSize="Large"

FontAttributes="Bold"

TextColor="White"

HorizontalOptions="Center"/>

<!-- User Details with proper binding -->

<Label Text="{Binding User.Name, StringFormat='Your Name: {0}'}"

TextColor="White"

FontSize="Medium"

HorizontalOptions="Start" />

<Label Text="{Binding User.Username, StringFormat='Your User Name: {0}'}"

TextColor="White"

FontSize="Medium"

HorizontalOptions="Start" />

<Label Text="{Binding User.Email, StringFormat='Your Email: {0}'}"

TextColor="White"

FontSize="Medium"

HorizontalOptions="Start" />

<Label Text="{Binding User.Team.team1, StringFormat='Your Team: {0}'}"

TextColor="White"

FontSize="Medium"

HorizontalOptions="Start" />

<Label Text="{Binding User.IsAdmin, StringFormat='Your Admin Status: {0}'}"

TextColor="White"

FontSize="Medium"

HorizontalOptions="Start" />

<Label Text="{Binding User.Role, StringFormat='Your Role: {0}'}"

TextColor="White"

FontSize="Medium"

HorizontalOptions="Start" />

</StackLayout>

</ScrollView>

</Grid>

</ContentPage>

<?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="FootballProject.View.LogIn"

BackgroundColor="#1E1E1E">

<ScrollView>

<VerticalStackLayout

VerticalOptions="CenterAndExpand"

HorizontalOptions="CenterAndExpand"

Spacing="20"

Padding="30">

<Label

Text="The Human Before The Player"

TextColor="White"

FontSize="40"

HorizontalOptions="Center"

VerticalOptions="CenterAndExpand"

FontAttributes="Bold"

Margin="0,20,0,20"/>

<Label

Text="Welcome to "

TextColor="White"

FontSize="36"

HorizontalOptions="Center"

VerticalOptions="CenterAndExpand"

FontAttributes="Bold"

Margin="0,20,0,40"/>

<Label

Text="Your ultimate football management tool"

TextColor="LightGray"

FontSize="20"

HorizontalOptions="Center"

VerticalOptions="CenterAndExpand"

Margin="0,0,0,20"/>

<Label

Text="Join us to manage your football team and stay updated with the latest news and stats."

TextColor="LightGray"

FontSize="16"

HorizontalOptions="Center"

VerticalOptions="CenterAndExpand"

Margin="0,40,0,0"

HorizontalTextAlignment="Center"/>

<!-- Login Form Section -->

<Frame WidthRequest="400" HeightRequest="300" CornerRadius="20" BorderColor="#8294C4" BackgroundColor="#1E1E1E" HorizontalOptions="Center" VerticalOptions="Center" Margin="0,20">

<VerticalStackLayout FlowDirection="LeftToRight" Padding="20">

<Grid RowDefinitions="\*,\*,\*">

<HorizontalStackLayout Grid.Row="0">

<Label Padding="15" Text="Username:" FontSize="15" TextColor="white" />

<Entry Placeholder="Enter your username"

Text="{Binding Username}"

TextColor="White"

WidthRequest="250"

ReturnType="Next"/>

</HorizontalStackLayout>

<HorizontalStackLayout Grid.Row="1">

<Label Padding="15" Text="Password:" FontSize="15" TextColor="White" />

<Entry Placeholder="Enter your password"

Text="{Binding Password}"

TextColor="White"

WidthRequest="250"

IsPassword="True"

ReturnType="Done"/>

</HorizontalStackLayout>

<Button Text="Login"

BorderColor="#8294C4"

BackgroundColor="#8294C4"

Command="{Binding LoginUserCommand}"

Grid.Row="2"

HorizontalOptions="Center"/>

</Grid>

</VerticalStackLayout>

</Frame>

<StackLayout Orientation="Horizontal" HorizontalOptions="Center" VerticalOptions="End" Spacing="20" Margin="0,40,0,0">

<Image Source="psagot.png" WidthRequest="100" HeightRequest="100" />

<Image Source="karmiel.png" WidthRequest="100" HeightRequest="100" />

<Image Source="israel.png" WidthRequest="100" HeightRequest="100" />

</StackLayout>

</VerticalStackLayout>

</ScrollView>

</ContentPage>

<?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"

xmlns:vm="clr-namespace:FootballProject.ViewModel"

x:Class="FootballProject.View.PlayerProfile"

Title="PlayerProfile"

BackgroundColor="#1E1E1E">

<Grid Padding="30" RowSpacing="20">

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="\*" />

</Grid.RowDefinitions>

<!-- TOP PROFILE CARD -->

<Frame Grid.Row="0" CornerRadius="50" BackgroundColor="#8294C4"

HorizontalOptions="FillAndExpand" VerticalOptions="Start" Padding="30,40,30,30">

<Grid ColumnSpacing="20">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<!-- Name & Nation -->

<StackLayout Grid.Column="1" Spacing="10" VerticalOptions="Center">

<Label Text="{Binding Player.FullName}" TextColor="White" FontSize="Large" FontAttributes="Bold"/>

<Label Text="{Binding Player.Nationality}" TextColor="White" FontSize="Medium" FontAttributes="Bold"/>

<Label Text="{Binding Player.DateOfBirth, StringFormat='{0:dd/MM/yyyy} - {1} Y/O'}"

TextColor="White" FontSize="Medium" FontAttributes="Bold"/>

</StackLayout>

<!-- Team & Financial -->

<StackLayout Grid.Column="2" Spacing="10" VerticalOptions="Center">

<Label Text="{Binding Player.Team.team1}" TextColor="White" FontSize="Medium" FontAttributes="Bold"/>

<Label Text="{Binding Player.UserValue, StringFormat='{}{0:N0} €'}" TextColor="White" FontSize="Medium" FontAttributes="Bold"/>

<Label Text="{Binding Player.Wage, StringFormat='{}{0:N0} € p/m'}" TextColor="White" FontSize="Medium" FontAttributes="Bold"/>

</StackLayout>

<!-- Physical Attributes -->

<StackLayout Grid.Column="3" Spacing="10" VerticalOptions="Center">

<Label Text="{Binding Player.Height, StringFormat='{0} cm'}" TextColor="White" FontSize="Medium" FontAttributes="Bold"/>

<Label Text="{Binding Player.Weight, StringFormat='{0} kg'}" TextColor="White" FontSize="Medium" FontAttributes="Bold"/>

<Label Text="{Binding Player.Foot}" TextColor="White" FontSize="Medium" FontAttributes="Bold"/>

<Button Grid.Column="4" Text="Back" BackgroundColor="White" TextColor="#8294C4"

Command="{Binding BackCommand}" HorizontalOptions="Start"

VerticalOptions="Start" Margin="0,10,0,0" />

</StackLayout>

</Grid>

</Frame>

<!-- POSITION + STATS -->

<Grid Grid.Row="1" ColumnSpacing="20">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="3\*" />

<!-- 30% -->

<ColumnDefinition Width="7\*" />

<!-- 70% -->

</Grid.ColumnDefinitions>

<!-- Left Frame: Position Visualization -->

<Frame Grid.Column="0" CornerRadius="50" BackgroundColor="#8294C4" BorderColor="#6B7DBB"

HorizontalOptions="FillAndExpand" VerticalOptions="FillAndExpand" Padding="30,40">

<StackLayout HorizontalOptions="Center" VerticalOptions="Center" Spacing="30">

<StackLayout Orientation="Horizontal" HorizontalOptions="Center" Spacing="10">

<Label Text="Your Position:" TextColor="White" FontSize="Medium" FontAttributes="Bold" VerticalOptions="Center"/>

<Label Text="{Binding Player.Position}" TextColor="White" FontSize="Medium" FontAttributes="Bold" VerticalOptions="Center"/>

</StackLayout>

<Grid x:Name="PositionGrid" RowSpacing="40">

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Ellipse Grid.Row="0" WidthRequest="40" HeightRequest="40" Fill="LightGray" HorizontalOptions="Center"/>

<StackLayout Grid.Row="1" Orientation="Horizontal" HorizontalOptions="Center" Spacing="80">

<Ellipse WidthRequest="40" HeightRequest="40" Fill="LightGray"/>

<Ellipse WidthRequest="40" HeightRequest="40" Fill="LightGray"/>

<Ellipse WidthRequest="40" HeightRequest="40" Fill="LightGray"/>

</StackLayout>

<Ellipse Grid.Row="2" WidthRequest="40" HeightRequest="40" Fill="LightGray" HorizontalOptions="Center"/>

<Ellipse Grid.Row="3" WidthRequest="40" HeightRequest="40" Fill="LightGray" HorizontalOptions="Center"/>

<StackLayout Grid.Row="4" Orientation="Horizontal" HorizontalOptions="Center" Spacing="80">

<Ellipse WidthRequest="40" HeightRequest="40" Fill="LightGray"/>

<Ellipse WidthRequest="40" HeightRequest="40" Fill="LightGray"/>

<Ellipse WidthRequest="40" HeightRequest="40" Fill="LightGray"/>

</StackLayout>

<Ellipse Grid.Row="5" WidthRequest="40" HeightRequest="40" Fill="LightGray" HorizontalOptions="Center"/>

</Grid>

</StackLayout>

</Frame>

<!-- Right Frame: Stats -->

<Frame Grid.Column="1" CornerRadius="50" BackgroundColor="#8294C4"

HorizontalOptions="FillAndExpand" VerticalOptions="FillAndExpand" Padding="30,60">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="\*" />

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

<!-- Top: Title -->

<Label Grid.Row="0" Text="Stats - games played/started"

TextColor="White" FontSize="Medium" FontAttributes="Bold"

HorizontalOptions="Center" VerticalOptions="Start" />

<!-- Bottom: 10 Stats -->

<ScrollView Grid.Row="1" Orientation="Vertical">

<FlexLayout Wrap="Wrap" Direction="Row"

JustifyContent="Center" AlignItems="Center"

BindableLayout.ItemsSource="{Binding PlayerStats}"

Margin="0,20"

AlignContent="Center">

<BindableLayout.ItemTemplate>

<DataTemplate>

<Frame BackgroundColor="#6B7DBB" CornerRadius="10" Padding="10" Margin="5">

<StackLayout Spacing="5" HorizontalOptions="Center">

<Label Text="{Binding Name}" TextColor="White" FontSize="Small" HorizontalOptions="Center"/>

<Label Text="{Binding Value}" TextColor="White" FontSize="Large" FontAttributes="Bold" HorizontalOptions="Center"/>

</StackLayout>

</Frame>

</DataTemplate>

</BindableLayout.ItemTemplate>

</FlexLayout>

</ScrollView>

</Grid>

</Frame>

</Grid>

</Grid>

</ContentPage>

<?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"

xmlns:vm="clr-namespace:FootballProject.ViewModel"

x:Class="FootballProject.View.PlayersSearch"

Title="Players Search"

BackgroundColor="#1E1E1E">

<ContentPage.BindingContext>

<vm:PlayersSearchViewModel />

</ContentPage.BindingContext>

<Grid Padding="30">

<!-- Define Rows for each filter, Search Bar and Results -->

<Grid.RowDefinitions>

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

<RowDefinition Height="10\*" />

</Grid.RowDefinitions>

<!-- Define Columns for Labels and Inputs -->

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto" />

<ColumnDefinition Width="\*" />

</Grid.ColumnDefinitions>

<!-- Filter Selection -->

<Label Grid.Row="0" Grid.Column="0" Text="Select Field:" TextColor="White" VerticalOptions="Center" Margin="5"/>

<Picker Grid.Row="0" Grid.Column="1" x:Name="FilterPicker" TextColor="White" BackgroundColor="#2E2E2E" Margin="5"

SelectedItem="{Binding SelectedField, Mode=TwoWay}">

<Picker.ItemsSource>

<x:Array Type="{x:Type x:String}">

<x:String>Any</x:String>

<x:String>full\_name</x:String>

<x:String>Nationality</x:String>

<x:String>dob</x:String>

<x:String>team</x:String>

<x:String>UserValue</x:String>

<x:String>wage</x:String>

<x:String>height</x:String>

<x:String>weight</x:String>

<x:String>foot</x:String>

<x:String>position</x:String>

</x:Array>

</Picker.ItemsSource>

</Picker>

<!-- Sort Selection -->

<Label Grid.Row="1" Grid.Column="0" Text="Sort By:" TextColor="White" VerticalOptions="Center" Margin="5"/>

<Picker Grid.Row="1" Grid.Column="1" x:Name="SortPicker" TextColor="White" BackgroundColor="#2E2E2E" Margin="5"

SelectedItem="{Binding SortOrder, Mode=TwoWay}">

<Picker.ItemsSource>

<x:Array Type="{x:Type x:String}">

<x:String>None</x:String>

<x:String>Asc</x:String>

<x:String>Desc</x:String>

</x:Array>

</Picker.ItemsSource>

</Picker>

<!-- Filter Input -->

<Label Grid.Row="2" Grid.Column="0" Text="Enter Value:" TextColor="White" VerticalOptions="Center" Margin="5"/>

<Entry Grid.Row="2" Grid.Column="1" x:Name="FilterValue" Placeholder="Enter value" TextColor="White" BackgroundColor="#2E2E2E" Margin="5"

Text="{Binding FilterValue, Mode=TwoWay}"/>

<!-- Search Button -->

<Button Grid.Row="3" Grid.ColumnSpan="2" Text="Search" BackgroundColor="#8294C4" TextColor="White" Margin="5"

Command="{Binding SearchCommand}"/>

<!-- Search Results -->

<Frame Grid.Row="4" Grid.ColumnSpan="2" CornerRadius="30" BackgroundColor="#8294C4"

Margin="20">

<ScrollView>

<CollectionView ItemsSource="{Binding Players}">

<CollectionView.ItemTemplate>

<DataTemplate>

<Frame CornerRadius="15" BackgroundColor="#1E1E1E">

<Frame.GestureRecognizers>

<TapGestureRecognizer Command="{Binding Source={RelativeSource AncestorType={x:Type vm:PlayersSearchViewModel}}, Path=NavigateToPlayerProfileCommand}" CommandParameter="{Binding .}" />

</Frame.GestureRecognizers>

<StackLayout Orientation="Vertical">

<StackLayout Orientation="Horizontal">

<Label Text="Full Name: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding FullName}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Nationality: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding Nationality}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Date of Birth: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding DateOfBirth}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Team: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding Team.team1}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="User Value: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding UserValue}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Wage: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding Wage}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Height: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding Height}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Weight: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding Weight}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Foot: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding Foot}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Position: " TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

<Label Text="{Binding Position}" TextColor="White" FontSize="Medium"

VerticalOptions="Center" HorizontalOptions="StartAndExpand"/>

</StackLayout>

</StackLayout>

</Frame>

</DataTemplate>

</CollectionView.ItemTemplate>

</CollectionView>

</ScrollView>

</Frame>

</Grid>

</ContentPage>

<?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="FootballProject.View.SignUp"

Title="SignUp"

BackgroundColor="#1E1E1E">

<VerticalStackLayout FlowDirection="LeftToRight">

<Grid RowDefinitions="\*,\*,\*,\*,\*,\*,\*">

<Label TextColor="Red" Grid.Row="0"

Text="{Binding ErrorMessage}"

IsVisible="{Binding HasError}"

FontSize="15"

Margin="0,10,0,0"

HorizontalTextAlignment="Center" />

<HorizontalStackLayout Grid.Row="1">

<Label TextColor="White" Padding="15" Text="Name:" FontSize="15" />

<Entry Placeholder="Enter your name:"

Text="{Binding Name}"

PlaceholderColor="White"

TextColor="White"

WidthRequest="250"

ReturnType="Next"/>

</HorizontalStackLayout>

<HorizontalStackLayout Grid.Row="2">

<Label TextColor="White" Padding="15" Text="Username:" FontSize="15" />

<Entry x:Name="username"

PlaceholderColor="White"

TextColor="White"

WidthRequest="200"

Text="{Binding Username}"/>

</HorizontalStackLayout>

<HorizontalStackLayout Grid.Row="3">

<Label TextColor="White" Padding="15" Text="Password:" FontSize="15" />

<Entry x:Name="pass"

PlaceholderColor="White"

TextColor="White"

WidthRequest="200"

Keyboard="Text"

IsPassword="True"

Text="{Binding Password}"/>

</HorizontalStackLayout>

<HorizontalStackLayout Grid.Row="4">

<Label TextColor="White" Padding="15" Text="Email:" FontSize="15" />

<Entry x:Name="email"

PlaceholderColor="White"

TextColor="White"

WidthRequest="200"

Keyboard="Email"

Text="{Binding Email}"/>

</HorizontalStackLayout>

<HorizontalStackLayout Grid.Row="5">

<Label TextColor="White" Padding="15" Text="Team:" FontSize="15" />

<Entry x:Name="team"

PlaceholderColor="White"

TextColor="White"

WidthRequest="200"

Text="{Binding Team}"/>

</HorizontalStackLayout>

<HorizontalStackLayout Grid.Row="6" RadioButtonGroup.GroupName="yn" RadioButtonGroup.SelectedValue="{Binding IsAdmin}">

<Label TextColor="White" Padding="15" Text="isAdmin:" FontSize="15" />

<RadioButton Content="Yes" Value="Yes"/>

<RadioButton Content="No" Value="No" IsChecked="True"/>

</HorizontalStackLayout>

<Button Text="Register"

BorderColor="#8294C4"

BackgroundColor="#8294C4"

TextColor="White"

Command="{Binding AddUserCommand}"

IsEnabled="{Binding CanAddUser}"

CommandParameter="{Binding Text, Source={x:RelativeSource Mode=Self}}"

Grid.Row="5"

HorizontalOptions="Center"/>

</Grid>

</VerticalStackLayout>

</ContentPage>

<?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="FootballProject.View.ViewUsers"

xmlns:model="clr-namespace:FootballProject.Model;assembly=FootballProject"

xmlns:viewmodel="clr-namespace:FootballProject.ViewModel;assembly=FootballProject"

Title="ViewUsers">

<Grid Margin="10">

<ScrollView>

<RefreshView Command="{Binding RefreshCommand}" IsRefreshing="{Binding IsRefreshing}" Grid.Row="2">

<CollectionView ItemsSource="{Binding ObservableUsers}">

<CollectionView.ItemTemplate>

<DataTemplate x:DataType="model:User">

<Border StrokeThickness="1" StrokeShape="RoundRectangle 10" Padding="10" Margin="5">

<Grid ColumnDefinitions="\*,\*,\*,\*,\*,\*,\*" RowDefinitions="\*,\*,\*" >

<Label Grid.Column="0" Text="{Binding Name}" TextColor="#8294C4" FontSize="Medium"/>

<Label Grid.Column="1" Text="{Binding Username}" TextColor="#8294C4" FontSize="Medium"/>

<Label Grid.Column="2" Text="{Binding Password}" TextColor="#8294C4" FontSize="Medium"/>

<Label Grid.Column="3" Text="{Binding Email}" TextColor="#8294C4" FontSize="Medium"/>

<Label Grid.Column="4" Text="{Binding Team.team1}" TextColor="#8294C4" FontSize="Medium"/>

<Label Grid.Column="5" Text="{Binding IsAdmin}" TextColor="#8294C4" FontSize="Medium"/>

<Button

Grid.Row="2" Grid.Column="0" BackgroundColor="#8294C4" TextColor="White" FontSize="Medium" Text="Delete user" HeightRequest="35" CornerRadius="10" Margin="5"

Command="{Binding DeleteCommand, Source={x:RelativeSource AncestorType={x:Type viewmodel:ViewUsersViewModel},Mode=FindAncestorBindingContext}}"

CommandParameter="{Binding .}"/>

<Button Grid.Row="2" Grid.Column="1" BackgroundColor="#8294C4" TextColor="White" FontSize="Medium" Text="Edit user" HeightRequest="35" CornerRadius="10" Margin="5"

Command="{Binding EditCommand, Source={x:RelativeSource AncestorType={x:Type viewmodel:ViewUsersViewModel},Mode=FindAncestorBindingContext}}"

CommandParameter="{Binding .}"/>

</Grid>

</Border>

</DataTemplate>

</CollectionView.ItemTemplate>

</CollectionView>

</RefreshView>

</ScrollView>

<Button BackgroundColor="#8294C4" VerticalOptions="End" TextColor="White" FontSize="Medium" Text="Add User" HeightRequest="35" CornerRadius="10" Margin="5"

Command="{Binding AddCommand, Source={x:RelativeSource AncestorType={x:Type viewmodel:ViewUsersViewModel}, Mode=FindAncestorBindingContext}}"

CommandParameter="{Binding .}"/>

</Grid>

</ContentPage>

using System;

using System.Collections.Generic;

using System.Data.OleDb;

using FootballProject.Model;

namespace FootballProject.ViewModel.DB

{

public abstract class BaseDB

{

private string connectionString;

protected OleDbConnection connection;

protected OleDbCommand command;

protected OleDbDataReader reader;

protected List<ChangeEntity> inserted = new List<ChangeEntity>();

protected List<ChangeEntity> deleted = new List<ChangeEntity>();

protected List<ChangeEntity> updated = new List<ChangeEntity>();

protected abstract BaseEntity newEntity();

protected abstract string CreateInsertOleDb(BaseEntity entity);

protected abstract string CreateUpdateOleDb(BaseEntity entity);

protected abstract string CreateDeleteOleDb(BaseEntity entity);

protected abstract BaseEntity CreateModel(BaseEntity entity);

public BaseDB()

{

//connectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=D:\Users\User\Documents\GitHub\FootballProject\FootballProject\ViewModel\DB\Football2.accdb;Persist Security Info=True";

connectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Users\אבי\Documents\GitHub\FootballProject\FootballProject\ViewModel\DB\FootballDB.accdb;Persist Security Info=True";

connection = new OleDbConnection(connectionString);

command = new OleDbCommand();

}

protected async Task<List<BaseEntity>> Select(string query)

{

List<BaseEntity> list = new List<BaseEntity>();

try

{

command.Connection = connection;

command.CommandText = query;

connection.Open();

reader = (OleDbDataReader)await command.ExecuteReaderAsync();

while (reader.Read())

{

BaseEntity entity = newEntity();

list.Add(CreateModel(entity));

}

}

catch (Exception e)

{

System.Diagnostics.Debug.WriteLine(e.Message + "\nOleDb: " + command.CommandText);

}

finally

{

if (reader != null) reader.Close();

if (connection.State == System.Data.ConnectionState.Open)

connection.Close();

}

return list;

}

public virtual void Insert(BaseEntity entity)

{

BaseEntity reqEntity = newEntity();

if (entity != null && entity.GetType() == reqEntity.GetType())

{

inserted.Add(new ChangeEntity(this.CreateInsertOleDb, entity));

}

}

public virtual void Update(BaseEntity entity)

{

BaseEntity reqEntity = newEntity();

if (entity != null && entity.GetType() == reqEntity.GetType())

{

updated.Add(new ChangeEntity(this.CreateUpdateOleDb, entity));

}

}

public virtual void Delete(BaseEntity entity)

{

BaseEntity reqEntity = newEntity();

if (entity != null && entity.GetType() == reqEntity.GetType())

{

deleted.Add(new ChangeEntity(this.CreateDeleteOleDb, entity));

}

}

public async Task<int> SaveChanges()

{

int records\_affected = 0;

OleDbCommand cmd = new OleDbCommand();

try

{

cmd.Connection = connection;

connection.Open();

foreach (var item in inserted)

{

cmd.CommandText = item.CreateOleDb(item.Entity);

records\_affected += await cmd.ExecuteNonQueryAsync();

command.CommandText = "Select @@Identity";

item.Entity.Id = (int)command.ExecuteScalarAsync().Result;

}

foreach (var item in updated)

{

cmd.CommandText = item.CreateOleDb(item.Entity);

records\_affected += await cmd.ExecuteNonQueryAsync();

}

foreach (var item in deleted)

{

cmd.CommandText = item.CreateOleDb(item.Entity);

records\_affected += await cmd.ExecuteNonQueryAsync();

}

}

catch (Exception e)

{

System.Diagnostics.Debug.WriteLine(e.Message + "\nOleDb: " + cmd.CommandText);

}

finally

{

inserted.Clear();

updated.Clear();

deleted.Clear();

if (connection.State == System.Data.ConnectionState.Open) connection.Close();

}

return records\_affected;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using FootballProject.Model;

namespace FootballProject.ViewModel.DB

{

public class BudgetDB : BaseDB

{

protected override BaseEntity newEntity()

{

return new Budget();

}

protected override BaseEntity CreateModel(BaseEntity entity)

{

var budget = (Budget)entity;

budget.Id = reader.GetInt32(reader.GetOrdinal("id"));

budget.TeamId = reader.GetInt32(reader.GetOrdinal("TeamId"));

budget.Wage = reader.GetInt32(reader.GetOrdinal("Wage"));

budget.Transfer = reader.GetInt32(reader.GetOrdinal("Transfer"));

budget.YearId = reader.GetInt32(reader.GetOrdinal("Total"));

budget.SeasonId = reader.GetInt32(reader.GetOrdinal("Profit/Lose"));

// YearlyBudget fields

budget.One = reader.GetInt32(reader.GetOrdinal("1"));

budget.Two = reader.GetInt32(reader.GetOrdinal("2"));

budget.Three = reader.GetInt32(reader.GetOrdinal("3"));

budget.Four = reader.GetInt32(reader.GetOrdinal("4"));

budget.Five = reader.GetInt32(reader.GetOrdinal("5"));

// SeasonBudget fields

budget.Jan = reader.GetInt32(reader.GetOrdinal("Jan"));

budget.Feb = reader.GetInt32(reader.GetOrdinal("Feb"));

budget.Mar = reader.GetInt32(reader.GetOrdinal("Mar"));

budget.Apr = reader.GetInt32(reader.GetOrdinal("Apr"));

budget.June = reader.GetInt32(reader.GetOrdinal("June"));

budget.July = reader.GetInt32(reader.GetOrdinal("July"));

budget.Aug = reader.GetInt32(reader.GetOrdinal("Aug"));

budget.Sep = reader.GetInt32(reader.GetOrdinal("Sep"));

budget.Oct = reader.GetInt32(reader.GetOrdinal("Oct"));

budget.Nov = reader.GetInt32(reader.GetOrdinal("Nov"));

budget.Dec = reader.GetInt32(reader.GetOrdinal("Dec"));

budget.ProfitLose = budget.Jan + budget.Feb + budget.Mar + budget.Apr + budget.June + budget.July + budget.Aug + budget.Sep + budget.Oct + budget.Nov + budget.Dec;

budget.Total = budget.One + budget.Two + budget.Three + budget.Four + budget.Five;

return budget;

}

protected override string CreateInsertOleDb(BaseEntity entity)

{

throw new NotImplementedException();

}

protected override string CreateUpdateOleDb(BaseEntity entity)

{

var budget = (Budget)entity;

return $@"

UPDATE Budget

SET

[Wage] = {budget.Wage},

[Transfer] = {budget.Transfer},

[Total] = {budget.Total},

[Profit/Lose] = {budget.ProfitLose}

WHERE TeamId = {budget.TeamId}";

}

protected override string CreateDeleteOleDb(BaseEntity entity)

{

throw new NotImplementedException();

}

public async Task<Budget> SelectByTeamId(int teamId)

{

string query = @"

SELECT

b.\*,

y.[1], y.[2], y.[3], y.[4], y.[5],

s.[Jan], s.[Feb], s.[Mar], s.[Apr], s.[June], s.[July], s.[Aug], s.[Sep], s.[Oct], s.[Nov], s.[Dec],

b.[Profit/Lose] AS ProfitLose

FROM

((Budget AS b

INNER JOIN YearlyBudget AS y ON b.Total = y.[id])

INNER JOIN SeasonBudget AS s ON b.[Profit/Lose] = s.[id])

WHERE

b.TeamId = " + teamId;

List<BaseEntity> list = await Select(query);

return list.FirstOrDefault() as Budget;

}

public async Task UpdateBudget(Budget budget)

{

Update(budget); // queue the update

await SaveChanges(); // save all queued changes

}

}

}

using FootballProject.Model;

using System.Collections.Generic;

using System.Data.OleDb;

using System.Threading.Tasks;

namespace FootballProject.ViewModel.DB

{

public class PlayerDB : BaseDB

{

protected override BaseEntity newEntity()

{

return new Player();

}

protected override string CreateInsertOleDb(BaseEntity entity)

{

throw new NotImplementedException();

}

protected override string CreateUpdateOleDb(BaseEntity entity)

{

throw new NotImplementedException();

}

protected override string CreateDeleteOleDb(BaseEntity entity)

{

throw new NotImplementedException();

}

protected override BaseEntity CreateModel(BaseEntity entity)

{

var player = (Player)entity;

player.Id = reader.GetInt32(0);

player.FullName = reader.GetString(1);

player.Nationality = reader.GetString(2);

player.DateOfBirth = reader.GetDateTime(3);

player.Team = new Team();

player.Team.Id = reader.GetInt32(4);

player.Team.team1 = reader.GetString(11);

player.UserValue = reader.GetInt32(5);

player.Wage = reader.GetInt32(6);

player.Height = reader.GetInt32(7);

player.Weight = reader.GetInt32(8);

player.Foot = reader.GetString(9);

player.Position = reader.GetString(10);

return player;

}

public async Task<List<Player>> SelectAllPlayers()

{

string query = @"

SELECT p.\*, t.Team

FROM players p

INNER JOIN Team t ON p.team = t.id";

List<BaseEntity> list = await Select(query);

return list.Cast<Player>().ToList();

}

public async Task<Player> SelectById(int id)

{

string query = $@"

SELECT p.\*, t.Team

FROM players p

INNER JOIN Team t ON p.team = t.id

WHERE p.id = {id}";

List<BaseEntity> list = await Select(query);

if (list != null && list.Count == 1)

return list[0] as Player;

return null;

}

public async Task<List<Player>> SelectByFilter(string field, string value)

{

string query;

if (field == "Any")

{

query = $@"

SELECT p.\*, t.Team

FROM players p

INNER JOIN Team t ON p.team = t.id

WHERE p.FullName LIKE '%{value}%' OR p.Nationality LIKE '%{value}%' OR p.DateOfBirth LIKE '%{value}%'

OR p.Team LIKE '%{value}%' OR p.UserValue LIKE '%{value}%' OR p.Wage LIKE '%{value}%'

OR p.Height LIKE '%{value}%' OR p.Weight LIKE '%{value}%' OR p.Foot LIKE '%{value}%'

OR p.Position LIKE '%{value}%'";

}

else

{

query = $@"

SELECT p.\*, t.Team

FROM players p

INNER JOIN Team t ON p.team = t.id

WHERE p.{field} LIKE '{value}'";

}

List<BaseEntity> list = await Select(query);

return list.Cast<Player>().ToList();

}

public async Task<List<Player>> SelectAndSort(string field, string order)

{

string query = $@"

SELECT p.\*, t.Team

FROM players p

INNER JOIN Team t ON p.team = t.id

ORDER BY p.{field} {order}";

List<BaseEntity> list = await Select(query);

return list.Cast<Player>().ToList();

}

public async Task<List<Player>> SelectPlayersByTeam(int teamId)

{

string query = $@"

SELECT p.\*, t.Team

FROM players p

INNER JOIN Team t ON p.team = t.id

WHERE p.team = {teamId}";

List<BaseEntity> list = await Select(query);

return list.Cast<Player>().ToList();

}

public async Task<List<Player>> SelectTeamPlayersByFirstName(int teamId, string firstName)

{

string query = $@"

SELECT p.\*, t.Team

FROM players p

INNER JOIN Team t ON p.team = t.id

WHERE p.team = {teamId} AND p.FullName LIKE '%{firstName}%'";

List<BaseEntity> list = await Select(query);

return list.Cast<Player>().ToList();

}

}

}

using FootballProject.Model;

namespace FootballProject.ViewModel.DB

{

public class SeasonBudgetDB : BaseDB

{

protected override BaseEntity newEntity() => new Budget();

protected override string CreateUpdateOleDb(BaseEntity entity)

{

var budget = (Budget)entity;

return $@"

UPDATE SeasonBudget

SET

[Jan] = {budget.Jan}, [Feb] = {budget.Feb}, [Mar] = {budget.Mar}, [Apr] = {budget.Apr},

[June] = {budget.June}, [July] = {budget.July}, [Aug] = {budget.Aug}, [Sep] = {budget.Sep},

[Oct] = {budget.Oct}, [Nov] = {budget.Nov}, [Dec] = {budget.Dec}

WHERE id = {budget.ProfitLose}";

}

protected override BaseEntity CreateModel(BaseEntity entity) => entity;

protected override string CreateInsertOleDb(BaseEntity entity) => throw new NotImplementedException();

protected override string CreateDeleteOleDb(BaseEntity entity) => throw new NotImplementedException();

public async Task UpdateBudget(Budget budget)

{

Update(budget); // queue the update

await SaveChanges(); // save all queued changes

}

}

}

using System;

using System.Collections.Generic;

using System.Data.OleDb;

using System.Threading.Tasks;

using FootballProject.Model;

namespace FootballProject.ViewModel.DB

{

public class StatsDB : BaseDB

{

protected override BaseEntity newEntity()

{

return new Stat();

}

protected override BaseEntity CreateModel(BaseEntity entity)

{

Stat stat = (Stat)entity;

// Skip id and player\_id and just get the first stat column

for (int i = 0; i < reader.FieldCount; i++)

{

string name = reader.GetName(i);

if (name != "id" && name != "player\_id")

{

stat.Name = name;

stat.Value = Convert.ToInt32(reader[i]);

break; // Only take the first stat

}

}

return stat;

}

protected override string CreateInsertOleDb(BaseEntity entity)

{

throw new NotImplementedException();

}

protected override string CreateUpdateOleDb(BaseEntity entity)

{

throw new NotImplementedException();

}

protected override string CreateDeleteOleDb(BaseEntity entity)

{

throw new NotImplementedException();

}

public async Task<List<Stat>> SelectStatsByPosition(string position, int playerId)

{

string table = position.ToLower() + "Stats";

string query = $"SELECT \* FROM [{table}] WHERE player\_id = {playerId}";

List<Stat> stats = new List<Stat>();

try

{

command.Connection = connection;

command.CommandText = query;

connection.Open();

reader = (OleDbDataReader)await command.ExecuteReaderAsync();

if (reader.Read())

{

for (int i = 0; i < reader.FieldCount; i++)

{

string col = reader.GetName(i);

if (col == "id" || col == "player\_id") continue;

stats.Add(new Stat

{

Name = col,

Value = Convert.ToInt32(reader[i])

});

}

}

}

finally

{

if (reader != null) reader.Close();

if (connection.State == System.Data.ConnectionState.Open) connection.Close();

}

return stats;

}

}

}

using FootballProject.Model;

using System.Collections.Generic;

using System.Data.OleDb;

namespace FootballProject.ViewModel.DB

{

public class UserDB : BaseDB

{

protected override BaseEntity newEntity()

{

return new User();

}

protected override string CreateInsertOleDb(BaseEntity entity)

{

var user = (User)entity;

return $"INSERT INTO users (FullName, Username, UserPassword, Email, team, admin) VALUES ('{user.Name}', '{user.Username}', '{user.Password}', '{user.Email}','{user.Team}','{user.IsAdmin}')";

}

protected override string CreateUpdateOleDb(BaseEntity entity)

{

var user = (User)entity;

return $"UPDATE users SET FullName = '{user.Name}', Username = '{user.Username}', UserPassword = '{user.Password}', Email = '{user.Email}', team = '{user.Team}', admin = '{user.IsAdmin}' WHERE id = {user.Id}";

}

protected override string CreateDeleteOleDb(BaseEntity entity)

{

var user = (User)entity;

return $"DELETE FROM users WHERE id = {user.Id}";

}

protected override BaseEntity CreateModel(BaseEntity entity)

{

var user = (User)entity;

user.Id = reader.GetInt32(0);

user.Name= reader.GetString(1);

user.Username = reader.GetString(2);

user.Password = reader.GetString(3);

user.Email = reader.GetString(4);

user.Team = new Team();

user.Team.Id = reader.GetInt32(5);

user.Team.team1 = reader.GetString(8);

user.IsAdmin = reader.GetString(6);

user.Role = reader.GetString(7);

return user;

}

public async Task<List<User>> SelectAllUsers()

{

string query = @"

SELECT u.\*, t.Team

FROM users u

INNER JOIN Team t ON u.team = t.id";

List<BaseEntity> list = await Select(query);

return list.Cast<User>().ToList();

}

public async Task<User> SelectById(int id)

{

string query = $"SELECT \* FROM users WHERE id = {id}";

List<BaseEntity> list = await Select(query);

if(list != null && list.Count == 1)

return list[0] as User;

return null;

}

public async Task<User> SelectByUsername(string username)

{

// Escape single quotes in the username to avoid SQL injection

string safeUsername = username.Replace("'", "''");

string query = $@"

SELECT u.\*, t.Team

FROM users u

LEFT JOIN Team t ON u.team = t.id

WHERE u.Username = '{safeUsername}'";

List<BaseEntity> list = await Select(query);

if (list != null && list.Count == 1)

return list[0] as User;

return null;

}

public override void Insert(BaseEntity entity)

{

User user = entity as User;

if (user != null)

{

inserted.Add(new ChangeEntity(this.CreateInsertOleDb, entity));

}

}

public override void Update(BaseEntity entity)

{

User user = entity as User;

if (user != null)

{

updated.Add(new ChangeEntity(this.CreateUpdateOleDb, entity));

}

}

public override void Delete(BaseEntity entity)

{

User user = entity as User;

if (user != null)

{

deleted.Add(new ChangeEntity(this.CreateDeleteOleDb, entity));

}

}

}

}

using FootballProject.Model;

namespace FootballProject.ViewModel.DB

{

public class YearlyBudgetDB : BaseDB

{

protected override BaseEntity newEntity() => new Budget(); // or a specific model if needed

protected override string CreateUpdateOleDb(BaseEntity entity)

{

var budget = (Budget)entity;

return $@"

UPDATE YearlyBudget

SET

[1] = {budget.One},

[2] = {budget.Two},

[3] = {budget.Three},

[4] = {budget.Four},

[5] = {budget.Five}

WHERE id = {budget.Total}";

}

protected override BaseEntity CreateModel(BaseEntity entity) => entity;

protected override string CreateInsertOleDb(BaseEntity entity) => throw new NotImplementedException();

protected override string CreateDeleteOleDb(BaseEntity entity) => throw new NotImplementedException();

public async Task UpdateBudget(Budget budget)

{

Update(budget); // queue the update

await SaveChanges(); // save all queued changes

}

}

}

using FootballProject.Model;

using FootballProject.Services;

using FootballProject.ViewModel.DB;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Threading.Tasks;

using System.Windows.Input;

using Microsoft.Maui.Controls;

namespace FootballProject.ViewModel

{

public class ClubPlayersSearchViewModel : ViewModelBase

{

private readonly PlayerDB playerDB;

private readonly IUser userService;

private ObservableCollection<Player> players;

private string selectedField;

private string filterValue;

private string sortOrder;

public ClubPlayersSearchViewModel(IUser service)

{

playerDB = new PlayerDB();

players = new ObservableCollection<Player>();

userService = service;

SelectedField = "Any";

SortOrder = "None";

SearchCommand = new Command(async () => await SearchPlayersAsync());

NavigateToPlayerProfileCommand = new Command<Player>(async (player) => await NavigateToPlayerProfileAsync(player));

}

public ObservableCollection<Player> Players

{

get => players;

set

{

players = value;

OnPropertyChanged();

}

}

public string SelectedField

{

get => selectedField;

set

{

selectedField = value;

OnPropertyChanged();

}

}

public string FilterValue

{

get => filterValue;

set

{

filterValue = value;

OnPropertyChanged();

}

}

public string SortOrder

{

get => sortOrder;

set

{

sortOrder = value;

OnPropertyChanged();

}

}

public ICommand SearchCommand { get; }

public ICommand NavigateToPlayerProfileCommand { get; }

private async Task SearchPlayersAsync()

{

var manager = (userService as UserService)?.GetCurrentUser(); // Only if local UserService

if (manager?.Team == null)

{

await Application.Current.MainPage.DisplayAlert("Error", "No team associated with this user.", "OK");

return;

}

var teamId = manager.Team.Id;

if (!string.IsNullOrWhiteSpace(FilterValue))

{

Players = new ObservableCollection<Player>(

await playerDB.SelectTeamPlayersByFirstName(teamId, FilterValue));

}

else

{

Players = new ObservableCollection<Player>(

await playerDB.SelectPlayersByTeam(teamId));

}

}

private async Task NavigateToPlayerProfileAsync(Player player)

{

var data = new Dictionary<string, object>

{

{ "player", player },

{ "source", "ClubPlayersSearch" }

};

await Shell.Current.GoToAsync("/rPlayerProfile", data);

}

}

}

using FootballProject.Model;

using FootballProject.Services;

using FootballProject.ViewModel.DB;

using Microcharts;

using SkiaSharp;

using System;

using System.Threading.Tasks;

using System.Windows.Input;

using Windows.System;

namespace FootballProject.ViewModel

{

public class FinancePageViewModel : ViewModelBase

{

private readonly BudgetDB budgetDB;

private readonly YearlyBudgetDB yearlyBudgetDB;

private readonly SeasonBudgetDB seasonBudgetDB;

private long bankBalance;

private readonly IUser userService;

public Chart YearlyChart { get; private set; }

public Chart MonthlyChart { get; private set; }

public long BankBalance

{

get => bankBalance;

set { bankBalance = value; OnPropertyChanged(); OnPropertyChanged(nameof(BankBalanceString)); }

}

private long profitOrLoss;

public long ProfitOrLoss

{

get => profitOrLoss;

set { profitOrLoss = value; OnPropertyChanged(); OnPropertyChanged(nameof(ProfitOrLossString)); }

}

private long transferBudget;

public long TransferBudget

{

get => transferBudget;

set

{

transferBudget = value;

OnPropertyChanged();

OnPropertyChanged(nameof(TransferBudgetString));

}

}

private long wage;

public long Wage

{

get => wage;

set

{

wage = value;

OnPropertyChanged();

OnPropertyChanged(nameof(WageString));

}

}

private int sliderValue;

public int SliderValue

{

get => sliderValue;

set

{

sliderValue = value;

UpdateBudgetsBasedOnSlider();

OnPropertyChanged();

}

}

private long totalBudget;

public int One { get; set; }

public int Two { get; set; }

public int Three { get; set; }

public int Four { get; set; }

public int Five { get; set; }

public int Jan { get; set; }

public int Feb { get; set; }

public int Mar { get; set; }

public int Apr { get; set; }

public int June { get; set; }

public int July { get; set; }

public int Aug { get; set; }

public int Sep { get; set; }

public int Oct { get; set; }

public int Nov { get; set; }

public int Dec { get; set; }

public int YearlyId { get; set; }

public int SeasonId { get; set; }

public string BankBalanceString => $"Overall Bank Balance: {BankBalance:N0} €";

public string ProfitOrLossString => $"Profit/Loss This Season: {ProfitOrLoss:N0} €";

public string TransferBudgetString => $"{TransferBudget:N0} €";

public string WageString => $"{Wage:N0} €";

public FinancePageViewModel(IUser service)

{

userService = service;

budgetDB = new BudgetDB();

yearlyBudgetDB = new YearlyBudgetDB();

seasonBudgetDB = new SeasonBudgetDB();

int teamId = (userService as UserService)?.GetCurrentUser().Team.Id ?? 0;

LoadBudgetData(teamId);

SaveCommand = new Command(async () => await SaveBudgets());

YearlyChart = new LineChart

{

Entries = new[]

{

new ChartEntry(20) { Label = "No Data", ValueLabel = "0", Color = SKColors.Gray }

}

};

MonthlyChart = new LineChart

{

Entries = new[]

{

new ChartEntry(20) { Label = "No Data", ValueLabel = "0", Color = SKColors.Gray }

}

};

LoadYearlyChart();

LoadMonthlyChart();

}

public ICommand SaveCommand { get; }

private async void LoadBudgetData(int teamId)

{

try

{

Budget budget = await budgetDB.SelectByTeamId(teamId);

if (budget != null)

{

BankBalance = budget.Total;

ProfitOrLoss = budget.ProfitLose;

TransferBudget = budget.Transfer;

Wage = budget.Wage;

YearlyId = budget.YearId;

SeasonId = budget.SeasonId;

One = budget.One;

Two = budget.Two;

Three = budget.Three;

Four = budget.Four;

Five = budget.Five;

Jan = budget.Jan;

Feb = budget.Feb;

Mar = budget.Mar;

Apr = budget.Apr;

June = budget.June;

July = budget.July;

Aug = budget.Aug;

Sep = budget.Sep;

Oct = budget.Oct;

Nov = budget.Nov;

Dec = budget.Dec;

totalBudget = TransferBudget + Wage;

SliderValue = (int)((TransferBudget \* 100) / totalBudget);

}

else

{

BankBalance = 0;

ProfitOrLoss = 0;

TransferBudget = 0;

Wage = 0;

totalBudget = 0;

SliderValue = 0;

}

}

catch (Exception)

{

BankBalance = -1;

ProfitOrLoss = -1;

TransferBudget = -1;

Wage = -1;

totalBudget = -1;

SliderValue = 0;

}

}

private void LoadYearlyChart()

{

YearlyChart = new LineChart

{

Entries = new[]

{

new ChartEntry(One) { Label = "Year 1", ValueLabel = One.ToString(), Color = SKColors.Red },

new ChartEntry(Two) { Label = "Year 2", ValueLabel = Two.ToString(), Color = SKColors.Orange },

new ChartEntry(Three) { Label = "Year 3", ValueLabel = Three.ToString(), Color = SKColors.Yellow },

new ChartEntry(Four) { Label = "Year 4", ValueLabel = Four.ToString(), Color = SKColors.Green },

new ChartEntry(Five) { Label = "Year 5", ValueLabel = Five.ToString(), Color = SKColors.Blue }

},

LabelOrientation = Orientation.Horizontal,

ValueLabelOrientation = Orientation.Horizontal

};

OnPropertyChanged(nameof(YearlyChart));

}

private void LoadMonthlyChart()

{

MonthlyChart = new LineChart

{

Entries = new[]

{

new ChartEntry(Jan) { Label = "Jan", ValueLabel = Jan.ToString(), Color = SKColors.Red },

new ChartEntry(Feb) { Label = "Feb", ValueLabel = Feb.ToString(), Color = SKColors.Orange },

new ChartEntry(Mar) { Label = "Mar", ValueLabel = Mar.ToString(), Color = SKColors.Yellow },

new ChartEntry(Apr) { Label = "Apr", ValueLabel = Apr.ToString(), Color = SKColors.Green },

new ChartEntry(June) { Label = "Jun", ValueLabel = June.ToString(), Color = SKColors.Teal },

new ChartEntry(July) { Label = "Jul", ValueLabel = July.ToString(), Color = SKColors.Blue },

new ChartEntry(Aug) { Label = "Aug", ValueLabel = Aug.ToString(), Color = SKColors.Purple },

new ChartEntry(Sep) { Label = "Sep", ValueLabel = Sep.ToString(), Color = SKColors.Magenta },

new ChartEntry(Oct) { Label = "Oct", ValueLabel = Oct.ToString(), Color = SKColors.Brown },

new ChartEntry(Nov) { Label = "Nov", ValueLabel = Nov.ToString(), Color = SKColors.Gray },

new ChartEntry(Dec) { Label = "Dec", ValueLabel = Dec.ToString(), Color = SKColors.Black }

},

LabelOrientation = Orientation.Horizontal,

ValueLabelOrientation = Orientation.Horizontal

};

OnPropertyChanged(nameof(MonthlyChart));

}

private void UpdateBudgetsBasedOnSlider()

{

if (totalBudget <= 0) return;

System.Diagnostics.Debug.WriteLine($"slider value: {sliderValue}");

TransferBudget = (int)(totalBudget \* ((float)sliderValue / 100f));

Wage = totalBudget - TransferBudget;

System.Diagnostics.Debug.WriteLine($"TB value: {TransferBudget}");

System.Diagnostics.Debug.WriteLine($"Wage value: {Wage}");

OnPropertyChanged(nameof(TransferBudgetString));

OnPropertyChanged(nameof(WageString));

}

private async Task SaveBudgets()

{

if (totalBudget <= 0) return;

try

{

var currentUser = (userService as UserService)?.GetCurrentUser();

Budget updatedBudget = new Budget

{

TeamId = currentUser.Team.Id,

Transfer = this.TransferBudget,

Wage = this.Wage,

Total = this.YearlyId,

ProfitLose = this.SeasonId,

One = this.One,

Two = this.Two,

Three = this.Three,

Four = this.Four,

Five = this.Five,

Jan = this.Jan,

Feb = this.Feb,

Mar = this.Mar,

Apr = this.Apr,

June = this.June,

July = this.July,

Aug = this.Aug,

Sep = this.Sep,

Oct = this.Oct,

Nov = this.Nov,

Dec = this.Dec

};

await budgetDB.UpdateBudget(updatedBudget);

await yearlyBudgetDB.UpdateBudget(updatedBudget);

await seasonBudgetDB.UpdateBudget(updatedBudget);

await budgetDB.SaveChanges();

await yearlyBudgetDB.SaveChanges();

await seasonBudgetDB.SaveChanges();

}

catch (Exception ex)

{

Console.WriteLine($"Error saving budget: {ex.Message}");

}

}

}

}

using FootballProject.Model;

using FootballProject.Services;

using System.Windows.Input;

using Microsoft.Maui.Controls;

namespace FootballProject.ViewModel

{

[QueryProperty(nameof(User), "User")]

public class HomePageViewModel : ViewModelBase

{

private readonly IUser \_userService;

private User user;

public User User

{

get => user;

set

{

if (user != value)

{

user = value;

OnPropertyChanged();

}

}

}

public ICommand LogoutCommand { get; }

public HomePageViewModel(IUser userService)

{

\_userService = userService;

LogoutCommand = new Command(async () => await OnLogoutAsync());

}

private async Task OnLogoutAsync()

{

// Optional: clear current user if applicable

if (\_userService is UserService us)

{

us.SetCurrentUser(null);

}

// Navigate to login page

await Shell.Current.GoToAsync("///rLogIn");

}

}

}

using FootballProject.Model;

using FootballProject.Services;

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using System.Windows.Input;

using Microsoft.Maui.Controls;

namespace FootballProject.ViewModel

{

public class LoginViewModel : ViewModelBase

{

private string username;

private string password;

private readonly IUser service; // Now works with WebService or UserService

private User user;

public ICommand NavToRegisterCommand { get; }

public ICommand LoginUserCommand { get; }

public LoginViewModel(IUser service)

{

this.service = service;

LoginUserCommand = new Command(async () => await LoginUser());

NavToRegisterCommand = new Command(async () => await Shell.Current.GoToAsync("RegisterPage"));

Username = "Bruh"; // default for testing

Password = "Bruh"; // default for testing

}

public string Username

{

get => username;

set

{

if (username != value)

{

username = value;

OnPropertyChanged();

OnPropertyChanged(nameof(CanLogin));

}

}

}

public string Password

{

get => password;

set

{

if (password != value)

{

password = value;

OnPropertyChanged();

OnPropertyChanged(nameof(CanLogin));

}

}

}

public User User

{

get => user;

set

{

if (user != value)

{

user = value;

OnPropertyChanged();

}

}

}

public bool CanLogin => !string.IsNullOrWhiteSpace(Username) && !string.IsNullOrWhiteSpace(Password);

private async Task LoginUser()

{

try

{

var fetchedUser = await service.GetUser(Username);

if (fetchedUser == null || fetchedUser.Password != Password)

{

await Application.Current.MainPage.DisplayAlert("Login Failed", "Invalid username or password.", "OK");

return;

}

User = fetchedUser;

await Application.Current.MainPage.DisplayAlert("Login Success", $"Welcome, {User.Name}!", "OK");

App.Current.MainPage = new AppShell();

await Shell.Current.GoToAsync("///rHomePage", new Dictionary<string, object>

{

{ "User", User }

});

}

catch (Exception ex)

{

await Application.Current.MainPage.DisplayAlert("Error", $"Something went wrong: {ex.Message}", "OK");

}

}

}

}

using FootballProject.Model;

using FootballProject.Services;

using FootballProject.ViewModel.DB;

using System.Collections.ObjectModel;

using System.Threading.Tasks;

using System.Windows.Input;

using Microsoft.Maui.Controls;

namespace FootballProject.ViewModel

{

[QueryProperty(nameof(Player), "player")]

[QueryProperty(nameof(Source), "source")]

public class PlayerProfileViewModel : ViewModelBase

{

private Player player;

private ObservableCollection<Stat> playerStats;

private readonly StatsDB statsDB;

private readonly IUser userService;

private string source;

public PlayerProfileViewModel(IUser service)

{

userService = service;

statsDB = new StatsDB();

BackCommand = new Command(async () => await OnBackCommandExecuted());

}

public ObservableCollection<Stat> PlayerStats

{

get => playerStats;

set

{

playerStats = value;

OnPropertyChanged();

}

}

public string Source

{

get => source;

set

{

source = value;

OnPropertyChanged();

}

}

public Player Player

{

get => player;

set

{

player = value;

OnPropertyChanged();

if (player != null)

{

\_ = LoadPlayerStatsAsync(); // fire-and-forget

}

}

}

public ICommand BackCommand { get; }

private async Task LoadPlayerStatsAsync()

{

var stats = await userService.GetAllStats(player.Position, player.Id);

player.Stats = stats;

PlayerStats = new ObservableCollection<Stat>(stats);

}

private async Task OnBackCommandExecuted()

{

switch (Source)

{

case "ClubPlayersSearch":

await Shell.Current.GoToAsync("///rClubPlayersSearch");

break;

case "PlayersSearch":

default:

await Shell.Current.GoToAsync("///rPlayersSearch");

break;

}

}

}

}

using FootballProject.Model;

using FootballProject.ViewModel.DB;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Threading.Tasks;

using System.Windows.Input;

using Microsoft.Maui.Controls;

namespace FootballProject.ViewModel

{

public class PlayersSearchViewModel : ViewModelBase

{

private readonly PlayerDB playerDB;

private ObservableCollection<Player> players;

private string selectedField;

private string filterValue;

private string sortOrder;

public PlayersSearchViewModel()

{

playerDB = new PlayerDB();

players = new ObservableCollection<Player>();

SelectedField = "Any"; // Default field

SortOrder = "None"; // Default sort

SearchCommand = new Command(async () => await SearchPlayersAsync());

NavigateToPlayerProfileCommand = new Command<Player>(async (player) => await NavigateToPlayerProfileAsync(player));

}

public ObservableCollection<Player> Players

{

get => players;

set

{

players = value;

OnPropertyChanged();

}

}

public string SelectedField

{

get => selectedField;

set

{

selectedField = value;

OnPropertyChanged();

}

}

public string FilterValue

{

get => filterValue;

set

{

filterValue = value;

OnPropertyChanged();

}

}

public string SortOrder

{

get => sortOrder;

set

{

sortOrder = value;

OnPropertyChanged();

}

}

public ICommand SearchCommand { get; }

public ICommand NavigateToPlayerProfileCommand { get; }

private async Task SearchPlayersAsync()

{

if (SelectedField == "Any" || string.IsNullOrEmpty(SelectedField))

{

if (!string.IsNullOrWhiteSpace(FilterValue))

{

Players = new ObservableCollection<Player>(

await playerDB.SelectByFilter("Any", FilterValue));

}

else if (!string.IsNullOrWhiteSpace(SortOrder) && SortOrder != "None")

{

Players = new ObservableCollection<Player>(

await playerDB.SelectAndSort("Any", SortOrder));

}

else

{

Players = new ObservableCollection<Player>(

await playerDB.SelectAllPlayers());

}

}

else

{

if (!string.IsNullOrWhiteSpace(FilterValue))

{

Players = new ObservableCollection<Player>(

await playerDB.SelectByFilter(SelectedField, FilterValue));

}

else if (!string.IsNullOrWhiteSpace(SortOrder) && SortOrder != "None")

{

Players = new ObservableCollection<Player>(

await playerDB.SelectAndSort(SelectedField, SortOrder));

}

else

{

Players = new ObservableCollection<Player>(

await playerDB.SelectAllPlayers());

}

}

}

private async Task NavigateToPlayerProfileAsync(Player player)

{

var data = new Dictionary<string, object>

{

{ "player", player },

{ "source", "PlayersSearch" }

};

await Shell.Current.GoToAsync("/rPlayerProfile", data);

}

}

}

using FootballProject.Model;

using FootballProject.Services;

using FootballProject.ViewModel.DB;

using System.Collections.Generic;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Windows.Input;

using Microsoft.Maui.Controls;

namespace FootballProject.ViewModel

{

[QueryProperty(nameof(EditUser), "user")]

public class SignUpViewModel : ViewModelBase

{

private readonly IUser userService;

private List<User> users = new List<User>();

private User user = new User();

private User? editUser;

private string name, username, password, email;

private Team team;

private string isAdmin = "No";

private string errorMessage;

public SignUpViewModel(IUser service)

{

userService = service;

\_ = userService.initUsers();

AddUserCommand = new Command(async () => await AddOrUpdateUser());

}

public string Name

{

get => name;

set { name = value; OnPropertyChanged(); HandleError(); }

}

public string Username

{

get => username;

set { username = value; OnPropertyChanged(); HandleError(); }

}

public string Password

{

get => password;

set { password = value; OnPropertyChanged(); HandleError(); }

}

public string Email

{

get => email;

set { email = value; OnPropertyChanged(); HandleError(); }

}

public Team Team

{

get => team;

set { team = value; OnPropertyChanged(); }

}

public string IsAdmin

{

get => isAdmin;

set { isAdmin = value; OnPropertyChanged(); }

}

public string ErrorMessage

{

get => errorMessage;

set { errorMessage = value; OnPropertyChanged(); }

}

public bool HasError => !ValidName() || !ValidUsername() || !ValidPassword() || !ValidEmail();

public bool CanAddUser => !HasError;

public ICommand AddUserCommand { get; }

public User? EditUser

{

get => editUser;

set

{

editUser = value;

if (editUser != null)

{

user = new User

{

Id = editUser.Id,

Name = editUser.Name,

Username = editUser.Username,

Password = editUser.Password,

Email = editUser.Email,

Team = editUser.Team,

IsAdmin = editUser.IsAdmin

};

Name = editUser.Name;

Username = editUser.Username;

Password = editUser.Password;

Email = editUser.Email;

Team = editUser.Team;

IsAdmin = editUser.IsAdmin;

}

else

{

user = new User();

Name = Username = Password = Email = null;

Team = null;

IsAdmin = "No";

}

HandleError();

OnPropertyChanged();

}

}

private bool ValidName() => !string.IsNullOrWhiteSpace(Name) && Name.Length >= 3;

private bool ValidUsername() => !string.IsNullOrWhiteSpace(Username) && Username.Length >= 3;

private bool ValidPassword() => !string.IsNullOrWhiteSpace(Password) && Password.Length >= 6;

private bool ValidEmail()

{

if (string.IsNullOrWhiteSpace(Email)) return false;

var pattern = @"^[^@\s]+@[^@\s]+\.[^@\s]+$";

return Regex.IsMatch(Email, pattern);

}

private void HandleError()

{

var errors = new List<string>();

if (!ValidName()) errors.Add("Name is too short.");

if (!ValidUsername()) errors.Add("Username is too short.");

if (!ValidPassword()) errors.Add("Password must be at least 6 characters.");

if (!ValidEmail()) errors.Add("Invalid email address.");

ErrorMessage = string.Join("\n", errors);

OnPropertyChanged(nameof(HasError));

OnPropertyChanged(nameof(CanAddUser));

}

private async Task AddOrUpdateUser()

{

users = await userService.GetAllUsers();

HandleError();

if (HasError) return;

// Editing existing user

if (EditUser != null && user != null && EditUser.Id == user.Id)

{

var existing = users.Find(u => u.Id == user.Id);

if (existing != null)

{

existing.Name = Name;

existing.Username = Username;

existing.Password = Password;

existing.Email = Email;

existing.Team = Team;

existing.IsAdmin = IsAdmin;

bool updated = await userService.UpdateUser(existing);

if (updated)

{

await Shell.Current.DisplayAlert("Success", "User updated successfully.", "OK");

await Shell.Current.GoToAsync("///rViewUsers");

EditUser = null;

}

else

{

await Shell.Current.DisplayAlert("Error", "Failed to update user.", "Cancel");

}

}

}

else

{

// Check for duplicate username

if (users.Exists(u => u.Username.Equals(Username, StringComparison.OrdinalIgnoreCase)))

{

await Shell.Current.DisplayAlert("Error", "Username already exists.", "OK");

return;

}

user = new User

{

Id = users.Count + 1,

Name = Name,

Username = Username,

Password = Password,

Email = Email,

Team = Team,

IsAdmin = IsAdmin

};

bool added = await userService.AddUser(user);

if (added)

{

await Shell.Current.DisplayAlert("Success", "User added successfully.", "OK");

await Shell.Current.GoToAsync("///rViewUsers");

}

else

{

await Shell.Current.DisplayAlert("Error", "Failed to add user.", "Cancel");

}

EditUser = null;

}

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

namespace FootballProject.ViewModel

{

public class ViewModelBase : INotifyPropertyChanged

{

public event PropertyChangedEventHandler? PropertyChanged;

protected void OnPropertyChanged([CallerMemberName] string propertyName = null)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

}

}

using FootballProject.Model;

using FootballProject.Services;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Threading.Tasks;

using System.Windows.Input;

using Microsoft.Maui.Controls;

namespace FootballProject.ViewModel

{

public class ViewUsersViewModel : ViewModelBase

{

private readonly IUser userService;

private bool isRefreshing;

private bool isAdmin;

private ObservableCollection<User> observableUsers;

public ObservableCollection<User> ObservableUsers

{

get => observableUsers;

set

{

if (observableUsers != value)

{

observableUsers = value;

OnPropertyChanged();

}

}

}

public ICommand RefreshCommand { get; }

public ICommand EditCommand { get; }

public ICommand DeleteCommand { get; }

public ICommand AddCommand { get; }

public bool IsRefreshing

{

get => isRefreshing;

set

{

if (isRefreshing != value)

{

isRefreshing = value;

OnPropertyChanged();

}

}

}

public bool IsAdmin

{

get => isAdmin;

set

{

if (isAdmin != value)

{

isAdmin = value;

OnPropertyChanged();

}

}

}

public ViewUsersViewModel(IUser service)

{

userService = service;

ObservableUsers = new ObservableCollection<User>();

RefreshCommand = new Command(async () => await RefreshAsync());

AddCommand = new Command(async () => await GoToAddUserPage());

DeleteCommand = new Command<User>(async (u) => await DeleteUserAsync(u));

EditCommand = new Command<User>(async (u) => await EditUserAsync(u));

\_ = LoadUsersAsync(); // fire and forget

}

public async Task LoadUsersAsync()

{

var users = await userService.GetAllUsers();

ObservableUsers = new ObservableCollection<User>(users);

}

public async Task RefreshAsync()

{

IsRefreshing = true;

await LoadUsersAsync();

IsRefreshing = false;

}

private async Task DeleteUserAsync(User user)

{

bool deleted = await userService.DeleteUser(user);

if (deleted)

{

await RefreshAsync();

}

}

private async Task EditUserAsync(User user)

{

var data = new Dictionary<string, object>

{

{ "user", user }

};

await Shell.Current.GoToAsync("rSignUp", data);

}

private async Task GoToAddUserPage()

{

await Shell.Current.GoToAsync("rSignUp");

}

public User GetCurrentUser()

{

if (userService is UserService us)

return us.GetCurrentUser();

return null;

}

}

}

using FootballProject.Services;

using FootballProject.View;

using FootballProject.ViewModel;

using Microcharts.Maui;

using Microsoft.Extensions.Logging;

namespace FootballProject

{

public static class MauiProgram

{

public static MauiApp CreateMauiApp()

{

var builder = MauiApp.CreateBuilder();

builder

.UseMauiApp<App>()

.UseMicrocharts()

.ConfigureFonts(fonts =>

{

fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");

fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");

});

#if DEBUG

builder.Logging.AddDebug();

#endif

builder.Services.AddSingleton<MainPage>();

builder.Services.AddSingleton<SignUp>();

builder.Services.AddSingleton<SignUpViewModel>();

builder.Services.AddSingleton<LogIn>();

builder.Services.AddSingleton<LoginViewModel>();

builder.Services.AddSingleton<ViewUsers>();

builder.Services.AddSingleton<ViewUsersViewModel>();

builder.Services.AddSingleton<HomePage>();

builder.Services.AddSingleton<HomePageViewModel>();

builder.Services.AddSingleton<FinancePage>();

builder.Services.AddSingleton<FinancePageViewModel>();

builder.Services.AddSingleton<PlayersSearch>();

builder.Services.AddSingleton<PlayersSearchViewModel>();

builder.Services.AddSingleton<ClubPlayersSearch>();

builder.Services.AddSingleton<ClubPlayersSearchViewModel>();

builder.Services.AddSingleton<PlayerProfile>();

builder.Services.AddSingleton <PlayerProfileViewModel>();

builder.Services.AddSingleton<IUser, WebService>();

return builder.Build();

}

}

}

using FootballServerGood.DataAccess;

using FootballServerGood.Model;

using Microsoft.AspNetCore.Mvc;

using System.Threading.Tasks;

namespace FootballServerGood.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class BudgetsController : ControllerBase

{

private readonly BudgetDB budgetDB = new BudgetDB();

[HttpGet("team/{teamId}")]

public async Task<ActionResult<Budget>> GetByTeamId(int teamId)

{

var budget = await budgetDB.SelectByTeamId(teamId);

if (budget == null) return NotFound();

return Ok(budget);

}

[HttpPut("{teamId}")]

public async Task<IActionResult> UpdateBudget(int teamId, [FromBody] Budget budget)

{

if (budget.TeamId != teamId) return BadRequest("Mismatched team ID.");

await budgetDB.UpdateBudget(budget);

return NoContent();

}

}

}

using FootballServerGood.Model;

using FootballServerGood.DataAccess;

using FootballServerGood.Model;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace FootballServerGood.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class UsersController : ControllerBase

{

private readonly UserDB userDB = new UserDB();

[HttpGet]

public async Task<ActionResult<List<User>>> GetAll()

{

var users = await userDB.SelectAllUsers();

return Ok(users);

}

[HttpGet("by-username/{username}")]

public async Task<ActionResult<User>> GetByUsername(string username)

{

var user = await userDB.SelectByUsername(username);

if (user == null) return NotFound();

return Ok(user);

}

[HttpGet("{id}")]

public async Task<ActionResult<User>> GetById(int id)

{

var user = await userDB.SelectById(id);

if (user == null) return NotFound();

return Ok(user);

}

[HttpPost]

public IActionResult AddUser([FromBody] User user)

{

userDB.Insert(user);

userDB.SaveChanges();

return CreatedAtAction(nameof(GetById), new { id = user.Id }, user);

}

[HttpPut("{id}")]

public IActionResult UpdateUser(int id, [FromBody] User user)

{

if (id != user.Id)

return BadRequest("ID mismatch");

userDB.Update(user);

userDB.SaveChanges();

return NoContent();

}

[HttpDelete("{id}")]

public IActionResult DeleteUser(int id)

{

var user = userDB.SelectById(id).Result;

if (user == null) return NotFound();

userDB.Delete(user);

userDB.SaveChanges();

return NoContent();

}

}

}

using FootballServerGood.DataAccess;

using FootballServerGood.Model;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace FootballServerGood.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class PlayersController : ControllerBase

{

private readonly PlayerDB playerDB = new PlayerDB();

[HttpGet]

public async Task<ActionResult<List<Player>>> GetAll()

{

return Ok(await playerDB.SelectAllPlayers());

}

[HttpGet("team/{teamId}")]

public async Task<ActionResult<List<Player>>> GetByTeam(int teamId)

{

return Ok(await playerDB.SelectPlayersByTeam(teamId));

}

[HttpGet("team/{teamId}/filter")]

public async Task<ActionResult<List<Player>>> FilterByTeamName(int teamId, [FromQuery] string name)

{

return Ok(await playerDB.SelectTeamPlayersByFirstName(teamId, name));

}

[HttpGet("filter")]

public async Task<ActionResult<List<Player>>> Filter([FromQuery] string field, [FromQuery] string value)

{

return Ok(await playerDB.SelectByFilter(field, value));

}

[HttpGet("sort")]

public async Task<ActionResult<List<Player>>> Sort([FromQuery] string field, [FromQuery] string order)

{

return Ok(await playerDB.SelectAndSort(field, order));

}

}

}

using FootballServerGood.DataAccess;

using FootballServerGood.Model;

using Microsoft.AspNetCore.Mvc;

using System.Threading.Tasks;

namespace FootballServerGood.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class SeasonBudgetsController : ControllerBase

{

private readonly SeasonBudgetDB seasonDB = new SeasonBudgetDB();

[HttpPut("{id}")]

public async Task<IActionResult> Update(int id, [FromBody] Budget budget)

{

if (budget.ProfitLose != id) return BadRequest("Mismatched season budget ID.");

await seasonDB.UpdateBudget(budget);

return NoContent();

}

}

}

using FootballServerGood.DataAccess;

using FootballServerGood.Model;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace FootballServerGood.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class StatsController : ControllerBase

{

private readonly StatsDB statsDB = new StatsDB();

// GET: api/stats/{position}/{id}

[HttpGet("{position}/{id}")]

public async Task<ActionResult<List<Stat>>> GetStatsByPosition(string position, int id)

{

var stats = await statsDB.SelectStatsByPosition(position, id);

if (stats == null || stats.Count == 0)

return NotFound();

return Ok(stats);

}

}

}

using FootballServerGood.DataAccess;

using FootballServerGood.Model;

using Microsoft.AspNetCore.Mvc;

using System.Threading.Tasks;

namespace FootballServerGood.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class YearlyBudgetsController : ControllerBase

{

private readonly YearlyBudgetDB yearlyDB = new YearlyBudgetDB();

[HttpPut("{id}")]

public async Task<IActionResult> Update(int id, [FromBody] Budget budget)

{

if (budget.Total != id) return BadRequest("Mismatched yearly budget ID.");

await yearlyDB.UpdateBudget(budget);

return NoContent();

}

}

}