1. Naam van de initiatiefnemer

Robin Roeland

1. Werknaam van het project

BaseballScoringApp

1. Documentatie van het project

Hieronder bevindt zich alle documentatie die ik bijgehouden heb bij het maken van de server API en MAUI client app zijde.

Het server project is BaseballCalcASP (folder BaseballCalcASP-Localizers)

Het client project is BaseballScoringApp

Het server project is het bestaande project dat ik voor .NET Advanced en .Net Frameworks heb gebouwd. Ik heb dit verder uitgebreid met API controllers vanaf dat bestaande project.

\*\*\*\* server side \*\*\*\*

De server zijde werkt nog steeds en is uitgebreid met een aantal API controllers die reageren op aanvragen vanaf de client App (MAUI).

De code hiervoor bevindt zich in het ASP.Net project BaseballCalcASP (folder BaseballCalcASP-Localizers). De onderliggende database is een SQLServer databank.

Het project bevat de server pages en een uitbreiding met API controllers die zich allemaal in de subfolder "APIControllers" bevinden.

Elke entiteit heeft zijn eigen controller class zoals bvb: GamesControllerAPI, GameStatisticsControllerAPI, PlayersControllerAPI, TeamsControllerAPI en AccountController (voor authenticatie)

\*\*\*\* client en server side authenticatie \*\*\*\*

In mijn bestaand project voor .NET Advanced en .NET Frameworks had ik voordien al authenticatie ingebouwd.

Er is reeds een login-page aanwezig en verschillende behandeling voor een gewone gebruiker en een administrator.

Voor de realisatie van mijn MAUI project heb ik de bestaande login procedure uitgebreid met een WJT-token-based authenticatie.

Mijn .NET project maakte nog gebruik van .NET6 en voor het implementeren van de JWT-Token moest ik eerst een upgrade naar .NET9 uitvoeren.

Na deze upgrade kreeg ik problemen met de scaffolding.

Ik kreeg een hardnekkige foutmelding die zelfs met het weggooien van bestaande migraties niet op te lossen viel:

*An error was generated for warning 'Microsoft.EntityFrameworkCore.Migrations.PendingModelChangesWarning': The model for context 'BaseballCalcASPContext' has pending changes. Add a new migration before updating the database. This exception can be suppressed or logged by passing event ID 'RelationalEventId.PendingModelChangesWarning' to the 'ConfigureWarnings' method in 'DbContext.OnConfiguring' or 'AddDbContext'. PS C:\Projects\BaseballCalcASP-Localizers\baseballcalcasp>"*

Uiteindelijk heb ik aan ChatGPT raad gevraagd en ik kreeg 2 opties: Een nieuwe cleane migratie maken – wat niet werkte – of een aanpassing doen aan de warnings:

--- ChatGPT Begin ---

In your DbContext class, modify the OnConfiguring method (or in AddDbContext if using Dependency Injection) to handle the warning:

*protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)  
{  
 optionsBuilder  
 .ConfigureWarnings(warnings =>   
 warnings.Ignore(RelationalEventId.PendingModelChangesWarning));  
}*

Alternatively, if you prefer to log the warning, use:

*optionsBuilder  
 .ConfigureWarnings(warnings =>   
 warnings.Log(RelationalEventId.PendingModelChangesWarning));*

--- ChatGPT End ---

Deze Ignore loste het probleem op en daarom heb ik dit toegepast in de BaseballCalcASPContext.OnConfiguring method.

De JWT-token wordt bij succesvol inloggen aangemaakt door de AccountController in de .NET API en wordt teruggestuurd naar de MAUI-app.

De token wordt aan de Client-Side in SecureStorage opgeslagen voor een betere veiligheid.

Bij een volgende opstarten van de MAUI-app, wordt (in App.xaml.cs) getest op de aanwezigheid en geldigheidsduur van de token.

Als de token niet bestaat of niet meer geldig is, komt het login-scherm en moet de gebruiker opnieuw inloggen zodat er weer een nieuwe JWT-Token aangemaakt en teruggestuurd wordt.

Bij een geldige token wordt de gebruiker meteen doorgestuurd naar het hoofdscherm.

Bij elke API-call vanuit de MAUI-app wordt de JWT-Token meegegeven in de Authorization Header.

De secret-key voor het aanmaken van de JWT-Token wordt centraal bewaard in appsettings.json in de .NET API

In de Windows HTML-login-page wordt de JWT-token ook gebruikt en opgeslagen in de HttpContext.Session.

(Dit gebeurt in de BaseballCalcASP-Localizers\BaseballCalcASP\Areas\Identity\Pages\Account\Login.cshtml.cs)

\*\*\*\* seeding in de .NET API \*\*\*\*

Initieel had ik enkel een paar kleinere teams en een paar spelers met wat fake data toegevoegd om te testen.

Uiteindelijk heb ik de teams en spelers van MLB gedownload en wou ik deze in de seeding verwerken zodat bij een nieuwe opstart meteen goede data voorhanden is.

Om de seeding van meer dan 1000 spelers wat eenvoudiger en praktischer te maken heb ik een SeedDataGenerator functie geschreven die een CSV file inleest om de seed functie te vullen.

Hiervoor moest ik het CsvHelper NuGet package installeren.

In de utils folder heb ik een SeedDataGenerator toegevoegd. Deze functie genereert SeedData voor spelers op basis van een CSV file players.csv

De oproep van de functie BaseballCalcASPContext.GeneratePlayerSeedData gebeurt eenmalig tijdelijk in program.cs en resulteert in een tekst-file 'playerSeedData.txt'.

Deze file bevat een lijn per speler:

*new Player { Id = 458677, Name = "Justin Wilson", TeamId = 111, DOB = DateTime.ParseExact("27-02-2003 12:00 am", "dd-MM-yyyy hh:mm tt", CultureInfo.InvariantCulture), Deleted = false },*

*new Player { Id = 500743, Name = "Miguel Rojas", TeamId = 119, DOB = DateTime.ParseExact("11-03-2000 12:00 am", "dd-MM-yyyy hh:mm tt", CultureInfo.InvariantCulture), Deleted = false },*

*new Player { Id = 502171, Name = "Alex Cobb", TeamId = 116, DOB = DateTime.ParseExact("08-11-2006 12:00 am", "dd-MM-yyyy hh:mm tt", CultureInfo.InvariantCulture), Deleted = false },*

Deze lijnen kunnen dan handmatig gecopy-paste worden in de BaseballCalcASPContext.SeedPlayers method.

Aangezien de SeedPlayers functie hierdoor veel te lang werd (meer dan 1000 lijnen), heb ik uiteindelijk de SeedPlayers aangepast zodat deze rechtstreeks de CSV file leest en het resultaat meteen in de Players tabel wegschrijft.

Hierdoor is de SeedPlayers functie beter te onderhouden wanneer er een nieuwe CSV file komt met spelers.

Dezelfde methode heb ik toegepast voor ScoreStatistics om bij een cleane opstart op zijn minst al een game met scores en statistieken voorhanden te hebben.

Hiervoor is er een Statistics.csv voorzien die in de BaseballCalcASPContext.SeedStatistics method ingelezen wordt en het resultaat meteen in de ScoreStatistics tabel wegschrijft.

Op die manier kan de statistics-functie van mijn MOA app ook getoond worden zonder eerst een game te moeten starten en te scoren.

De teams die in de geseede data aanwezig zijn komen van een game tussen "Los Angeles Angels" en "Kansas City Royals".

Er zijn na initiële seeding meteen statistics aanwezig voor volgende spelers:

Los Angeles Angels (id 108)

543294 Kyle Hendricks  
677347 Kyren Paris  
681351 Logan O'Hoppe  
694384 Nolan Schanuel  
621493 Taylor Ward  
650859 José Quijada  
664068 Scott Kinergy  
666160 Mickey Moniak  
676116 Ryan Noda

Kansas City Royals (id 118):

521692 Salvador Perez  
669004 MJ Melendez  
672580 Maikel Garcia  
677951 Bobby Witt Jr.  
679845 Nick Loftin  
686469 Vinnie Pasquantino  
672582 Angel Zerpa  
664728 Kyle Isbel  
676369 Nelson Velazquez  
686681 Michael Massey

Hier zijn nog eens te stappen om van een volledig leeg systeem te starten zonder DB en zonder aanwezige migrations:

1. Indien DB aanwezig: Drop DB via Microsoft SQL Server Management Studio
2. Indien Migrations folder niet leeg: leegmaken
3. In Visual Studio - Menu View - Terminal
4. In terminal (Developer Powershell): ga naar BasballCalcASP folder  
    cd BaseballCalcASP
5. Start migrations  
    dotnet-ef migrations add InitialDBSetup
6. Start db-update  
    dotnet-ef database update
7. In Visual Studio BaseballCalcASP (.NET API): Start BaseballCalcASP
8. In Visual Studio BaseballScoringApp (MAUI): check in globals.cs of serverURL goed staat (windows-app of phone of emulator)  
   Check ook via Command window ipconfig voor correct IP-adres voor serverURL  
   Voor emulator: start "Android Device Manager" en check of Android-emulator (pixel-7) gestart is
9. In Visual Studio BaseballScoringApp (MAUI): Start voor gekozen output (windows-app, phone of emulator)

\*\*\*\* packages en libraries in de MAUI app \*\*\*\*

Gebruikte NUGET packages:

- community toolkit  
- Newtonsoft JSON  
- Plugin.Maui.Audio  
- Metrolog.Maui  
- CsvHelper

Het project bestaat uit het main program in namespace BaseballScoringApp en maakt gebruikt van 2 libraries : BaseballModelsLib en MLBRestAPI.

BaseballModelsLib is de gedeelde library die de java pojo classes bevat die zowel in de api als de app gebruikt worden.

MLBRestAPI is een class library die nodig is om de initiele teams en players uit een api call om te zetten in objecten en in de database in te lezen.

Elke contentpage heeft een xaml pagina en code behind class. Elke class heeft een viewmodel class die de lokale data bevat.

De gedeelde data die overal gebruikt wordt bevindt zich in een data repository BBDataRepository (zie models subfolder)

Omdat er bijkomende applicatie-specifieke members en functionaliteiten geschreven werden voor de pojo object uit BaseballModelsLib bevat models ook specialisatie-klasses die afleiden van de pojo basis classes.

In de app (tijdens een game) wordt gebruik gemaakt van de Audio plugin om geluiden in MP3 formaat af te spelen.

Het object dat hiervoor gemaakt wordt is een singleton static instance van de class SoundManager.

De audiomanager instance als member van de SoundManager class gebruikt een filestream om voor elke MP3 file een audioplayer te maken.

Omdat meerdere sounds kort na mekaar kunnen afgespeeld worden voordat de vorige klaar is met afspelen worden de players bijgehouden in een List.

Elke soundplayer krijgt een PlayBackEnded callback functie die de player terug uit de lijst verwijdert wanneer het afspelen klaar is.

Initieel eenmalig inlezen van teams en players: -> initieel geinitialiseerd via een api call (zie testpage), geparsed uit Json en hiervoor afzonderlijk library gemaakt:

* MLBRestAPI sql server aangemaakt en in app import via tijdelijke testpagina via sqlserver
* Via postman call gedaan en resultaat gepaste, gebruik gemaakt van pastespecial, paste json as classes om lib classes aan te maken.
* Dit leest alle teams en spelers in vanaf de officiele MLB api's en slaat ze in de sqlserver op als master data.

\*\*\*\* GamestartContentPage \*\*\*\*

Bij het starten van de contentpage wordt gewacht tot de repository de teams en spelers ingeladen heeft via de backend API.

Het inlezen van de teams en players gebeurt asynchroon met http calls naar de backend.

Dit gebeurt met behulp van "await" die de uitvoerende functie onderbreekt tot de aangeroepen functie klaar is.

De eerste keer dat de startpage in onappearing komt, wordt er via de repository een call gestuurd om te checken of er nog een lopende game is gelinked aan de ingelogde user.

Indien een lopende game gevonden is wordt de vraag gesteld of de gebruiker de game will herstarten of een nieuw game wil kiezen.

Indien een bestaande game herstart wordt worden alle voordien opgeslagen statistics van die game gewist op de server en herstart de game.

Bij starten van een nieuwe game wordt de bestaande open game gewist op de server kant. Deze calls gebeuren via de repository.

Indien een game in progress was, wordt automatisch doorgeschakeld naar de ScoringContentPage.

Bij een nieuwe game kiezen wordt gebruik gemaakt van Carrouselview en IndicatorViews om de beide teams te selecteren.

Bij de currentItemChanged functies van de teams worden de teamlogo's van het nieuwe geselecteerde team ingeladen uit images in de resources/images.

De pitcherlijst wordt aangepast met enkel de spelers die Pitcher zijn voor het geselecteerde team.

Bij start game wordt een nieuwe game aangemaakt op de server-zijde via de repository (mRepo.StartNewGame(game)) na overname van de invoervelden in het game object.

Start new game roep de api call /api/games/addgame aan en geeft het game object door in JSON.

Bij succesvolle server afhandeling start het game object een nieuwe game: startgame().

De start game zal een line up bepalen uit de player roster van elk team en de gekozen pitcher initialiseren.

Daarna start de ScoreContentPage om het scoring te starten.

\*\*\*\* ScoringContentPage \*\*\*\*

De scoring page is opgebouwd als een grid met 4 rijen.

Rij 1: scoreboard grid

Rij 2: een overlay van een image (baseball veld), een graphicsview (gebruikt om lijnen te tekenen tussen de buttons van de spelers, en een absolutelayout "buttoncontainer" die alle buttons bevat.

1 button voor elke positie op het veld

Rij 3: een grid met daarin de informatie over de current batter en current pitcher.

Rij 4: een scrollview met actie buttons, afhankelijk van de situatie in het spel is scrollview PitchingPlay\_ButtonGroup zichtbaar of InfieldPlay\_ButtonGroup

Elke player/positie button is opgebouwd als een frame met daarin een imagebutton, een border en een Label om het speler-nummer af te beelden.

De image button zal gevuld worden met de foto van de speler die online gedownload wordt via de officiele MLB.com api website.

De scorecontentpage heeft twee modes: "Pitching" of "InFieldPlay". De actieve mode wordt bijgehouden in het BBGameProgress object binnen de current game.

Bij opstart is de mode Pitching. zodra er een actie gestart wordt die een infieldplay is wisselt de mode naar InFieldPlay.

Bij pitching mode kunnen enkel pitching gerelateerde acties uitgevoerd worden (ball, strike, walk batter, catcher error, of een infield actie gestart zoals 'single', 'double',...

Bij infieldplay kan een defensive play gekozen worden en kunnen verschillende andere offensive player acties of errors voor defensive players gekozen worden.

De infieldplay mode wordt beeindigd door de EndOfPlay button te kiezen.

Rij 3 wisselt de zichtbare button groep in de scrollview op basis van de actieve mode.

De buttons worden dynamisch gepositioneerd op basis van relatieve posities ten opzichte van de baseball field image.

Om dit generiek te maken is er een dictionary aangemaakt met als key het frame object (positiebutton) en als value de relatieve x en y positie in percentages van de grootte van de image.

(setButtonRelativePositionsOfFieldPlayers, deze wordt aangemaakt bij onpageappearing)

Bij resize worden de absolute button posities herberekend in de OnImageSizeChanged callback. Voor elke dictionary entry (frame) gaan we de bijpassende relatieve layout coordinaten ophalen en de nieuwe positie bepalen op basis van de nieuwe image width en height. Voor elke frame wordt dan de SetLayoutBounds en SetLayoutFlags gezet. Hierdoor krijgen de buttons hun nieuwe positie relatief aan de image size.

In de OnImageSizeChanged worden ook de posities en de getekende lijnen (defensive plays) herberekend en hertekend. (dit is enkel nodig bij infieldplay mode)

De gameplay main functie (of een simulatie van een gameloop) wordt bepaald door de functie: HandleGameStatus\_AtEndOfCurrentPlay

Deze bepaalt de huidige situatie van het spel zoals bvb: naar volgende inning, einde game.

Na elke operatie die uitgevoerd wordt zal deze functie aangeroepen worden.

\*\*\*\* UpdateScreenData \*\*\*\*

Na elke operatie of actie worden er wijzigingen doorgevoerd in het game en gameprogress object. Deze worden zichtbaar gemaakt aan de gebruiker door aanroepen van een game-teken-functie.

Hierin worden alle visuele elementen geupdate. Deze wordt aangeroepen in de gameloop functie : HandleGameStatus\_AtEndOfCurrentPlay

\*\*\*\* Beheer van statistieken en scoring \*\*\*\*

Tijdens elke inning worden scores bijgehouden in een central ScoreManager object. Dit is ook een singleton instance en bevat kennis over hoe een score geregistreerd moet worden (single of cummulative).

De score manager bestaat uit een dictionary met als key het spelerobject en als value een BBScoreStatisticDictEntry.

Een BBScoreStatisticDictEntry is een List<BBScoreStatistic> of met andere woorden: een lijst van alle scores die voor die speler geregistreerd werden.

De key functie voor het toevoegen van scores is registerScore() in het scoremanager object

Aan het einde van elke side van een inning (bvb einde van top of the 1st) worden alle scores in de score manager doorgestuurd naar de serverside via de API call.

Dit gebeurt via de functie publishScoreList() in score manager die dan via de Repository sendScoreList oproept en via de http call /api/statistics/bookStatistics een JSON met lijst van scores doorgeeft en in de database boekt. Daarna wordt de lijst leeggemaakt en start een nieuwe inning.

\*\*\*\* concept van GameActions en uitvoeren van Actions \*\*\*\*

Elke actie die kan gebeuren bij een speler in de scoring is gemodelleerd als een GameAction.

De basis klasse is GameActionBase die de interface IGameAction implementeert.

Elke actie die kan gebeuren is geschreven als een specialisatie van de basis klasse (inheritence) en heeft zijn eigen specifieke implementatie van de DoAction functie waarin de actie uitgevoerd wordt.

Bvb: GameAction\_Ball behandelt het gooien van een "ball", de DoAction functie hier verhoogt de pitch count, telt het aantal balls, en speelt een sound af. Indien er meer dan 3 balls zijn wordt een nieuwe action aangemaakt en uitgevoerd : GameAction\_WalkBatter. Dus een action kan ook andere actions starten en uitvoeren.

Ander vb GameAction\_MoveToBase: action om een speler naar een andere base te verplaatsen. indien naar home wordt de scoring verhoogd en de bijhorende scores geregistreerd.

Bij een klik op een speler tijdens gameplay wordt voor die speler positie een lijst met mogelijke acties opgevraagd. (zie bvb OnDefenderButton\_DoubleTapped en OnPlayerBase1Clicked)

Die List<IGameAction> wordt dan doorgegeven aan een generieke popup dialog die de acties uit de lijst laat zien, een keuze laat maken en bij keuze de actie uitvoert.

Dit gebeurt in de functie: handleChoiceFromActionList(possibleActionList);

Deze functie maakt een instance van DynamicButtonActionList popup dialog en laat die zien. De teruggegeven en gekozen actie wordt meteen uitgevoerd.

Na het uitvoeren van de actie wordt de game loop functie opnieuw uitgevoerd: HandleGameStatus\_AtEndOfCurrentPlay

\*\*\*\* messages \*\*\*\*

Gedurende het spel wordt feedback gegeven aan de gebruiker van het resultaat van een play-actie. Bvb : "Batter is out", "Ball 4, take a base", "Change Field!"

Dit zijn messages die als een soort overlay getoond worden bovenop het game scherm en gedurende een paar seconden zichtbaar zijn.

Soms zijn er meerdere messages nodig als resultaat van uitgevoerde acties. De messages worden dan ook geregistreerd in een stack (mMessagesToDisplayStack) binnen het BBGameProgress in BBGame.

Aan het einde van de gameloop functie (HandleGameStatus\_AtEndOfCurrentPlay) worden de boodschappen 1 voor 1 getoond tot de stack leeg is met behulp van de klasse PopupDialog\_ShowBoard.

Dit is een dialog afgeleid van Popup die in de constructor een string boodschap en duration in milliseconden neemt.

Bij ShowAsync wordt de message getoond als Label in het centrum van het scherm. Een Task.Delay zorgt er voor dat de Popup nadien automatisch terug sluit.

Het tonen van messages gebeurt ook in de HandleGameStatus\_AtEndOfCurrentPlay functie. Door await voor de showAsync te plaatsen worden boodschappen 1 voor 1 getoond tot ze allemaal getoond werden.

\*\*\*\* caching images \*\*\*\*

Player roster images headshots worden ingeladen vanaf de MLB.com webpagina https://img.mlbstatic.com/mlb-photos/image/upload/d\_people:generic:headshot:silo:current.png/r\_max/w\_180,q\_auto:best/v1/people/{MLBPersonId}/headshot/silo/current

waarbij {MLBPersonId} vervangen wordt door de player id van het BBPlayer object. Het telkens opnieuw inladen van images verliep te traag bij het hertekenen, daarom worden ingeladen images bijgehouden in het BBPLayer object (mPlayerImage) nadat de image is ingeladen. Eens ingeladen wordt het gecachte image object opnieuw gebruikt en niet opnieuw ingeladen.

Om het inladen van image standaard te maken gebruik ik de functie SetValidatedImageUrl(). Deze functie kijkt of de player image al bestaat of niet. Indien ze niet bestaat dan proberen we online de foto te vinden. We halen dan de url op voor de player en roepen ValidateImageUrl(imageurl) aan. Hierin openen we de url en kijken of er een error is. Indien er een error is laden we de defaultplayreimage.png uit de lokale resources. Indien de url benaderd kan worden, dan laden we de image in met ImageSource.FromUri(new Uri(url)). Daarna wordt die image gezet in het aanroepende GUI object (imagebutton of image)

\*\*\*\* OnPageTapped en OnTapGestureRecognizerTapped voor drawline hit \*\*\*\*

In InFieldPlay mode in het spel is het nodig om een punt op de image te kunnen aangeven waar de bal geslagen (of gevallen) is.

Om dat te kunnen bepalen moet je kunnen klikken op de image van het veld en dan een lijn tekenen van de batter positie naar het geklikte punt.

In OnImageResize ga ik dit punt ook herberekenen om bij size van window het punt mee te laten scalen met de nieuwe image size.

De tap detectie gebeurt op 2 manieren:

1. OnPageTapped functie gedeclareerd in de xaml als Grid.GestureRecognizers op de master grid layout van de scoringview.
2. OnTapGestureRecognizerTapped functie op de GraphicsView waarop de lijnen getekend worden die bovenop de image liggen.

In beide functies wordt de member mLastTappedPosition gezet in de contentpage.

De mLastTappedPosition wordt gebruikt in de tekenfunctie om DrawableLine objecten aan te maken die getekend worden op de graphicsview (mHitLine van LinesDrawable)

Graphics lijnen worden ook aangemaakt voor defensive plays, dit zijn bewegingen van de bal door de defending players.

Je maakt die door in InFieldPlay mode op de imagebuttons van de verdedigende ploeg te klikken.

Elke klik op een defender voegt die geklikte image button toe aan de collection mDefensiveButtonsSelected.

In de drawDefensiveplay() functie worden de geklikte buttons gebruikt om de lijnen ertussen aan te maken als lijn-objecten die dan getekend worden. (Lines collection in LinesDrawable)

De functie drawDefensiveplay wordt aangeroepen in UpdateScreenData() voor het aanmaken en tekenen van de lijnen en ook in OnImageSizeChanged om in geval van resize de posities te herberekenen op basis van de nieuw button posities.

Gebruikte info:

https://www.syncfusion.com/blogs/post/draw-2d-graphics-maui-graphicsview

https://learn.microsoft.com/en-us/dotnet/maui/user-interface/graphics/draw?view=net-maui-9.0

\*\*\*\* gestureRecognizer double click button \*\*\*\*

Voor een player is er zowel een single als een double click event op de ImageButton.

Single click wordt gebruikt voor het tekenen van defensive play lijnen.

Double click wordt gebruikt voor de popup van het possible actions menu.

Om een double click op een button te kunnen krijgen heb ik een gesturerecogniser aangemaakt die in zijn properties NumberOfClicksRequired = 2 heeft.

De event functie gaat pas af indien er 2 clicks geweest zijn. Elke ImageButton gebruikt dezelfde event functie OnDefenderButton\_DoubleTapped. Op basis van sender in de functie parameters wordt gekeken welke knop de dubbel klik kreeg.

Info gevonden via: https://learn.microsoft.com/en-us/dotnet/api/microsoft.maui.controls.clickgesturerecognizer.numberofclicksrequired?view=net-maui-9.0#microsoft-maui-controls-clickgesturerecognizer-numberofclicksrequired

\*\*\*\* end game -> EndBallGameContentPage \*\*\*\*

Wanneer in HandleGameStatus\_AtEndOfCurrentPlay vastgesteld wordt dat de ballgame voorbij is, dan wordt een nieuwe content page getoond met het resultaat van de wedstrijd.

Het vaststellen of de game over is gebeurt door aan BBGameProgress te vragen Handle\_OutsInInning(). De return value van deze functie zegt of de inning nog InProgress is, EndOfInning of EndOfBallGame.

Vanuit de EndBallGameContentPage keer je met een button terug naar GamestartContentPage om een nieuw spel te starten.

De game.BallGame() functie sluit intern het spel af. Elke speler krijgt nog een bijkomende GamesPlayed statistic toegewezen en deze worden opgeslagen samen met de scores uit de laatste inning (publishScoreList).

De game wordt in de database als Finished = true geboekt en via sendGameUpdate wordt /api/games/updategame aan de serverkant aangeroepen.