hogeschool Windesheim

Eindopdracht WFHBOICT.CSP.22

Regels

- Inlevering is per groep (twee studenten) één opdracht, met vermelding van de individuele leden van de groep. Schrijf je hiervoor in als groep op BrightSpace.
- De deadlines wordt vermeld bij de inlevertaak op Brightspace.
- Inleveren via inlevermap op BrightSpace op basis van de GitHub repository.
- Per groep werk je samen in één GitHub repository en lever je dus ook maar één repo in.
- In de GitHub repository van de hele Visual Studio solutions. Laat de *.vs* en *packages*-folders weg uit de repository. (bij .Net Core is eigenlijk geen packages folder meer, want die wordt los van je solution opgeslagen. Maar check toch even.)
- GitHub repository moet een versiegeschiedenis bevatten die past bij een normaal ontwikkeltraject. (dus niet één commit met je eindproduct).
- De commits zijn voorzien van een kort, professioneel, lading dekkend commentaar (bv: **Wel:** “Bugfix Inlogfunctionaliteit” **Niet:** “changes”/”aanpassing”/”Het werkt”/…).
- De applicaties moeten door ons te *builden* en te *runnen* zijn zonder verdere aanpassingen.
- Applicaties die niet *builden* of niet *runnen* worden niet nagekeken en zijn dus automatisch onvoldoende.
- Bij een Runtime error worden er punten afgetrokken op de onderwerpen waar deze effect op heeft. (Zorg ervoor dat je applicatie voldoende robuust is.)
- De applicatie moet voldoen aan alle functionele, maar vooral aan alle technische requirements.
- De implementatie moet overeenkomen met FO en TO. Houdt deze dan ook in overeenstemming. Gebruik bij voorkeur het C4-model aan voor de visualisatie. Afwijken mag, zolang het maar duidelijk is.
- Extra dingen toevoegen op eigen initiatief is op eigen risico.
- Er wordt altijd gekeken naar overeenkomsten tussen applicaties: als twee groepjes (grotendeels) dezelfde code inleveren wordt dit **altijd** als fraude gemeld bij de examencommissie.

Opdracht

De opdracht bestaat uit twee onderdelen: een WPF MVVM applicatie (beheerapplicatie) en een MVC-webapplicatie (gebruikersapplicatie) deze moeten ingeleverd en voldoende zijn.

BELANGRIJK: Voor alles geldt "Comply or Explain". Met andere woorden, je **moet** alles doen zoals het in het colleges en lesmateriaal wordt beschreven. Dezelfde methoden en technieken. Je **mag** er wel van afwijken, maar alleen als je die keuze uitlegt en onderbouwt (in commentaar in de code). Dus bijvoorbeeld: "*Wij hebben hier Business Logic in het model, ipv in het ViewModel, omdat wij het niet eens zijn met het college en de afhankelijkheid van het ViewModel met de DbContext in onze ogen slecht onderhoudbare code oplevert*".

Afwijken zonder uitleg of afwijken uit gemakzucht, zonder écht goede reden wordt niet goed gerekend!

Deel 1: WPF MVVM-applicatie

Functionele requirements (Wat)

De applicatie die gebouwd moet worden is een beheermodule voor beheerders en medewerkers van een muziekstreamingdienst (denk Spotify, Apple Music, etc). Het is dus niet bedoeld voor de eindgebruikers van de muziekstreamingdienst. Denk hier bij aan een uiteindelijk domeinmodel voor nummers (Songs), artiesten, albums en afspeellijsten. **De beheerders en medewerkers moeten alle data voor Songs en Albums kunnen lezen, invoeren en bewerken**.

Het staat vrij om naast de minimumeisen zelf invulling te geven, maar tenminste wordt verwacht dat er *Songs (nummers)* en *Albums* in het model zitten met gegeven details. Hierbij zijn enkele basisregels met betrekking tot de relaties: een *Album* kan meerdere *Songs* hebben. *Songs* kunnen op meerdere *Albums* voorkomen. Aangevuld met alles wat je zelf nog verzint. Je mag het model uitbreiden, maar houdt rekening met de tijd die je nodig hebt om het te implementeren.

Je mag van het muziekstreamingdienst-thema afwijken, maar dan moet je dat wel vervangen door een idee wat past bij de opdrachtcriteria en dit idee voldoende uitwerken (Comply or explain).

De applicatie moet tenminste voldoen aan de acceptatiecriteria van de user stories (zie bijlage A op Brightspace voor een lijst van User Stories met acceptatiecriteria).

Kort overzicht van de functionaliteiten:

- *Songs* kunnen aanmaken, bekijken, bewerken en verwijderen. (CRUD)
- *Albums* kunnen aanmaken, bekijken, bewerken en verwijderen. (CRUD)
- Kunnen aangeven welke *Songs* bij welke *Albums* horen (master-detail view).
- Je moet een overzicht van alle *Songs* kunnen zien (denk aan het bijhouden van een losse lijst met alle *Songs* is hier waarschijnlijk noodzakelijk, ERD en/of processchema en/of andere nuttige schema's)
- Functioneel moet de applicatie robuuster en uitgewerkter zijn dan de voorbeelden in de workshops en lesmateriaal. Denk bijvoorbeeld aan checks zodat een *Song* niet meerdere keren aan een *Album* toegevoegd kan worden. Sommige elementen van het programma worden hier niet benoemd, maar het kan wel zijn dat die elementen in de opdracht wel (binnen redelijke grenzen) aanwezig moeten zijn.
- Mogelijkheid tot sorteren en filteren van de lijst.
- Minimaal de mogelijkheid van zoeken op Song en *Album.*
- GUI naar eigen inzicht (daar gaat deze cursus niet over), maar gebruiksvriendelijke, mooie, slimme interface levert wel punten op (zie studiegids).

Technische requirements (Hoe)

De WPF-applicatie is in deze opzet de baas over de database. In deze applicatie komen alle handelingen met betrekking tot het databasebeheer, zoals Migrations en Seeding.

Bovenstaande functionele requirements moeten op de volgende manier gemaakt worden:

- Maak een logisch data-model. Tenminste moet het *Songs* en *Albums* bevatten (comply or explain). Deze Classes hebben **Properties** en (als dat nodig is voor de werking van de applicatie) **Methods** en **Constructors**.
- Maak een GUI in WPF met XAML om de CRUD-operaties van deze objecten mogelijk te maken.
- Houd verzamelingen van deze objecten bij op basis van ObservableCollection
- Gebruik databinding in de XAML om de WPF-controls te koppelen aan je data en implementeer **INotifyPropertyChanged** en **RelayCommand**
- Gebruik een ViewModel voor ieder scherm (Window, Page of ContentControl) dat je laat zien.
- Er zijn Migrations: De database moet door een ander te installeren zijn door Update-Database uit te voeren.
- Er is Seeding-Data (niet test1, test2 etc., maar wel enigszins logische, bij de casus passende data).
- Alle code moet voldoen aan de Microsoft coding conventions zoals behandeld. https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/naming-guidelines
- **Enum** types gebruiken en dropdowns ervoor in de GUI.
- Gebruik **CollectionViewSource** voor filter en sortering, of maak een **Linq** query waarmee je een deel van de data laat zien.

Beoordelingscriteria

Zie beoordelingsmatrix WPF

Deel 2: MVC webapplicatie:

Functionele requirements (Wat)

De applicatie die gebouwd moet worden is de website voor klanten van de streamingdienst. Het haalt de data uit de database van de beheerapplicatie.

Als klant moet je met de applicatie details van *Songs* en *Albums* kunnen bekijken en *Playlists* kunnen maken en deze ook kunnen bekijken en 'afspelen'. Er moet een audiospeler gemaakt worden om *Songs* af te spelen. (*Een implementatie met echte audio is niet vereist, maar mag wel*). Er moet een log-in-systeem zijn (*maak **géén** gebruik van de authenticatie die door .NET geleverd wordt, deze geven op dit punt in de opleiding meer problemen dan ze oplossen*)
De applicatie moet tenminste voldoen aan de acceptatiecriteria van de user stories (zie bijlage B op Brightspace voor een lijst van User Stories met acceptatiecriteria).
Kort overzicht van de functionaliteiten:

- Er moet een welkomstscherm zijn, vanaf waar je verder kunt
- Er moet een overzichtsscherm zijn waar je Songs, Albums of Artiesten of genre kunt zoeken en waar suggesties op gedaan worden die passen bij de voorkeur, afluisterhistorie van de gebruiker.
- Als je op een Song, Album of Artiest klikt moet je de details kunnen zien en van daar uit kunnen afspelen en/of aan een *Playlist* kunnen toevoegen.
- Een gebruiker moet in staat zijn om een *Song, Album, Artist, Playlist* te kunnen 'liken'.
- In een *Playlist* moet je een naam kunnen opgeven en of deze privé of openbaar is.
- De *Playlist* moet opgeslagen worden
- Je krijgt een overzicht van je *Playlists* (alles wat is opgeslagen en misschien de kosten – als er ook een *Price* zit in je Model bijvoorbeeld).
- De weergave van een playlists, albums, artiesten en songs moeten statistieken weergeven zoals afspeelduur, aantal
- **Optioneel**: Voor degenen die een werkende player willen maken. Spotify levert een Web Playback API die je eventueel zou kunnen gebruiken. Je bent vrij om een andere technische implementatie te bedenken. Dit is een onderdeel waar je je technische uitdaging in mag zoeken. Neem het wel mee in het TO als je hier iets mee doet.

Technische requirements (Hoe)

Bovenstaande functionele requirements moeten gemaakt worden op de volgende manier:

- Breidt het Model uit deel 1 (WPF-applicatie) uit met: *User* en *Playlist* (of iets soortgelijks, waarmee je bovenstaande functionaliteit kunt bereiken).
- Een *Playlist* moet tenminste bevatten:
  - *Naam*
  - *Timestamp*
- De applicatie is een ASP.NET MVC applicatie in .NET 7.
- De applicatie gebruikt een database met Entity Framework.
- Gebruik zoveel mogelijk tag-helpers (en/of HTML-helpers) in de Razor-Syntax
- Maak ViewModels voor de Views
- Gebruik annotaties op het ViewModel / Model
- Alle code moet voldoen aan de Microsoft coding conventions zoals behandeld in les 5. https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/naming-guidelines
- **Optioneel**: gebruik WebAPI uit les 14 om de data voor een View op te halen i.p.v. via de Controller+ViewModel+View

Beoordelingscriteria

Zie beoordelingsmatrix MVC

Succes!
Team C#