Projektarbete – Digital hyrfilmsapp Richard Zejak

Jag och min gruppkamrat Stefan fick i uppgift att utveckla en filmuthyrnings app där kunden skulle kunna logga in, hyra filmer, och därefter se sin hyr-historik.

Vi började med att se över den första versionen av appen som vi fick i samband med uppgiften, vi tyckte det var en bra grund att utgå ifrån men vi märkte också att den inte utnyttjade API’n till fullo.

API’n vi fick hade tillgång till: IMDB id, Filmtitel, IMDB Rating, Genre, och Bildlänk, men i context.cs hade bara Filmtitlen och bildlänken laddats in, så vi började med att påbygga model.cs med properties för all denna oanvända datan från API’n, och dessutom fler relevanta properties om customern: Password för att göra det mer säkert, Age ifall man skulle vilja göra en barnvänlig sidan för yngre customers i framtiden, och även E-mail.

Efter det kodade vi om Context.cs så att vi kunde ladda in all den oanvända datan från API’n till dom properties vi hade gjort, vilket gjorde att vi hade helt plötsligt en massa data i databasen som vi hade att arbeta med.

Vi bestämde oss först för att arbeta med att göra appen säkrare, det vill säga, vi tog bort alla standard customers som låg i Seeding.cs och började arbeta på en account page, vi lade till att du fick skriva Password två gånger för att du verkligen skulle skriva rätt också, vi arbetade också med att du skulle därefter kunna logga in med ditt nyaskapade konto, redan i början av utvecklandet med appen kom vi överens om att vi skulle arbeta med designen sist utav allt, och fokusera på funktionerna i appen.

När vi kände oss nöjda med create account och login delen av appen bestämde vi oss för att skriva ut lite mer info om filmerna, vi bestämde oss för att skriva ut både imdb-rating och genre under filmerna, När vi gjort detta kom vi överens om att fixa sorteringsfunktioner.

När vi fixade sorteringsfunktionerna gick vi till API.cs och tittade på den ursprungliga metoden av hur filmerna laddades in som standard, och såg direkt att de laddades in via namnet, då kunde vi bara skapa fler metoder som efterliknade den, fast istället köra ORDERBYDESCENDING om man ville sortera filmerna efter bokstavordning baklänges, vi gjorde samma sak fast med rating där vi sorterade efter högst nummer först och även en ORDERBYDESCENDING på det, så att man kunde sortera efter rating, lägst till högst.

När sorteringsfunktionerna var fixade, så kände vi att med så mycket data vi hade, så många filmer, så var vi tvungna att även ha en sökfunktion, vi tittade åter igen på metoderna vi hade använt för att sortera filmerna, vi tänkte att vi kunde använda en metod likt den metoden vi hade som sökte efter customernamn när man loggade in, som då istället först rensade State.Movies, och därefter lade till alla som den hittade i våran metod, vilket fungerade perfekt, Efter detta kände vi oss nöjda med sorterings och sökfunktionerna, nu återstod det bara att fixa ett ”my account” window.

Att hitta vilken användare det var, var enkelt vi hade redan vilken användare det var som var inloggad i State.User, och att ordna till en ”Change Password” funktion var inte svårt då vi istället bara gjorde en update på existerande customer istället för som när vi skapade en ny user.

Det svåra som också kom att bli det sista att tackla var att visa customerns rentals i accountpagen, här försökte vi komma åt det via:

Using (ctx context)

State.user.rentals , vilket inte fungerade så bra, problemet var att använder man sig av using ctx hela tiden så skapar man nya context vilket inte innehåller den ursprungliga informationen, vilket redan demot av appen hade haft i varenda metod i API.cs, där ersatte vi istället alla metoders using med en enda, Public static Context ctx = new Context(); längst upp, och på så vis kunde vi komma åt den överallt.

När denna funktion var fixad, kände vi oss nöjda med programmet, vi hade byggt på modellen, utökat context med mer data, byggt på appen efter nödvändiga funktioner, nu fixade vi till designen litegrann för att få en finare helhet och bättre känsla.

Det jag känner vi kunde gjort bättre och använt oss av, var för detta första istället för att byta fönster varje gång vi hoppar mellan fönster, utan använt oss av hide och show i wpf biblioteket, speciellt när man bytar sorteringssätt i filmlistan. Vi började med att skriva kod som tog bort alla children i grid, och att vi därefter skulle göra en ny filmlista istället för att hoppa till ett helt nytt fönster, problemet var att om vi skrev Grid.Children.Destroy, så tog vi även bort sorteringsfunktionsknapparna och account window. En lösning på detta var att omkonstruera alla knappar på nytt i koden, men det blev väldigt mycket kod och kändes inte alls tidseffektivt, en lösning jag hade i tanke som vi aldrig testade, hade varit att ta bort knapparna vi ville behålla från att vara child, sen förstöra alla children i gridet(filmerna och det) och därefter lägga till knapparna och de som child igen, och på så sätt unvika deras förstörelse, men av tidsskäl fick det bli att vi helt enkelt gjorde nya fönster för varje sorteringssätt.

Jag känner att databasdesignen hade kunnat vara bättre, men vi kände oss något begränsade av vad vi hade att pilla med från API’n, den hade inte mer data vi kunde plocka till databasen egentligen, det vi hade kunnat göra hade varit att splitta texten vid ”(” och ”)” och lagt in det inuti som element ifrån listan vilket var årtalen, då hade vi kunnat ge årtal till varje film i varsin record, istället för att årtalen skulle ingå i movie title recorden.

Något vi också hade kunnat göra hade varit att lägga till ännu fler filmer i listan vi hade, det vi kände var att man kunde ändå söka efter filmerna av alla som fanns i databasen, vilket eliminerade det problemet delvis.

Lazyloading var bra att ha i vårat fall eftersom det lät oss be om data enbart när vi behöver det, däremot hade det kanske inte varit lika bra ifall våran databas låg på en annan server, då vi hade fått skicka och hämta data online varje gång vi bad om det, vilket hade kunnat resultera i långsammare datalämning/datahämtning, i såna fall hade Eagerloading kunnat vara ett bättre alternativ då den laddar in allting i förväg och bara ligger och väntar på att användaren är beredd på det.

Allt som allt tycker jag det var ett roligt och inspirerande projekt, vi fick fria vingar, vi valde att använda API’n som begränsade oss något, men tycker vi ändå fick med många funktioner som en filmuthyrningsapp idag har.