Projektuppgift: Automatiserad RSS-läsare
Kurs: Objektorienterad programmering med C#, fortsättningskurs, HT20, IK202G
3.5 hp
Last updated: 2020-10-23

Table of Contents

Introduktion

Bolaget AutomateEverything tror att det finns en marknad för automatiserad RSS-läsare för privatpersoner. ”Tid=pengar” tänker de, ”och det här systemet kommer våra kunder spara tid på!”.

Bolaget tänker också att det är dyrt att anställa en konsult för att utföra jobbet.

- Vi anställer några studenter istället! De är billiga i drift, menar chefen på ett möte med utvecklingsteamet, och alla håller med.

Chefen planerar att projektet ska genomföras av grupper på 3 studenter, och de kommer få drygt två veckor på sig att genomföra uppgiften. Lansering av produkten sker 26 november så projektet måste vara inlämnat, med källkod, videodemonstration och rapport, senast **tisdag 3 november kl. 23:59** för att all funktionalitet ska hinna verifieras. Återkoppling och omdöme kommer att meddelas senast 24 november innan lanseringen.

Projektupplägg

I ert projekt får ni välja om ni vill använda Windows Forms eller WPF. På föreläsningarna har vi gått igenom Windows Forms men inte WPF (och WPF tas inte heller upp praktiskt i kursboken). WPF är således valfritt för de som vill utmana sig själva.

Arkitektur

Projektet ska struktureras i en tre-lagersarkitektur. Boken (2015 upplagan) tar upp detta kortfattat på sidan 362 och i föreläsningen för KT9 och KT12 ges exempel på det. Varje lager av applikationen ska vara en ”class library” och inte en mapp (se respektiva KT:en och sidorna 518-522 av boken).

Designmönstret ”repository pattern” ska implementeras i applikationen. Genom att använda repository pattern kan man minska duplicering av kod som annars ofta uppstår men också ha ett abstraktionslager mellan Business Logic och Data Access Layer. Repository pattern gör att samma funktioner kan användas för att hämta olika typer av data.

Ni ska också arbeta enligt Single Responsibility Principle när ni designerar er kod. Detta innebär att varje klass/metod ska ha en uppgift, inte fler.

Krav

Uppgiften är alltså att skapa en automatiserad RSS-läsare, som används för att prenumerera på podcasts (RSS-flöden, läs mer om detta i materialet för KT8). Det ska göras genom en desktop-applikation för Windows, där en användare kan mata in en URL till ett RSS-flöde för en podcast-prenumeration, namnge flödet samt ange hur ofta data ska hämtas via ett uppdateringsintervall.

Detta flöde ska sedan hämtas in och visas i en lista, som också ska lagras som XML på den egna datorn. Under tiden programmet körs ska det enligt ett angett uppdateringsintervall uppdatera listan och spara ner detta.

Er applikation behöver inte kunna spela upp nedladdade podcasts i fall finns media data (video eller bild) kopplad till podcasts avsnitt. För att utveckla applikationen ska ni använda Visual Studio och programmeringsspråket C#.

Tekniska krav

Förutom strukturering av applikationen i en tre-lagersarkitektur och användning av repository pattern i erat program ni ska använda följande:

- `Virtual`
    - `Virtual` är ett nyckelord som anger att en metod eller property i en superklass kan utökas av dess subklass(er).

    - Det skall finnas minst en superklass och subklass. Den subklass(en) ska kunna överlagra virtuella metoder från superklass(en).
    - Tänk om hur en sån arv-design kan implementeras innan ni börjar skriva själva koden!
- `Interface`
    - Ett `interface` representeras ett uppförandekontrakt (contract of behaviour). Ett interface innehåller endast deklarationer av metoder och egenskaper. Klasser som implementerar ett interface måste definiera metodkropp för de medlemmar som interface har deklarerat.
    - Det går bra att implementera de inbyggda interface som redan finns i .NET biblioteket men det skall finnas minst ett eget interface som ska implementeras också.
- `Async`
    - Att göra asynkrona anrop är viktigt när anropen kan ta lång tid tex att läsa in en Feed eller ladda ner en podcast. Tips: kolla på klasserna SyndicationFeed, XMLReader och Task för att lösa detta.
- `LINQ`
    - LINQ är ett sätt att hantera samlingar. Man kan filtrerar/sortera/räkna beroende på olika villkor.
    - Samlingar ska hanteras med LINQ.
- Exception handling och validering
    - Applikationen får inte ge några byggfel eller varningar vid körning och all felhantering ska hanteras med `Exception`.
    - Det ska finnas minst en egen klass som sköter all validering av det data användaren matar in och den klassen ska kasta passande Exceptions om inmatningen är felaktig.
    - Ni ska skapa egna `Exception` klasser (Custom Exception classes) som ska hantera fel i applikationen.
- Method overloading
    - Innebär att man kan definiera metoder med samma namn men med olika parametrar. Vid metodanropet kallas den metod som har en överensstämmande paramaterlista.

Funktionella krav

Applikationen kommer främst att hantera tre typer av entiteter: *feeds*, *avsnitt* och *kategorier*. De ska vara persistenta (kom ihåg object persistance med serialisering i KT8) dvs. att de ska sparas på användarnas dator och laddas in igen där man startar applikationen på nytt.

En feed är en kanal där webbinnehåll publiceras där användare via en länk kan prenumerera på innehållet. I vårt fall ska applikationen kunna hantera RSS-feeds i form av XML, där innehållet är podcast-avsnitt. Tips: KT8 föreläsning slides går genom `SyndicationFeed` som kan användas för att underlätta läsning av RSS-feeds för XML dokumenten.

Varje podcast ska tillhöra en kategori samt har ett antal avsnitt knutna till sig.

Följande funktionalitet ska vara möjlig med applikationen:

- Podcast feeds
- Lägg till en ny podcast feed som:
    - Har en URL (dvs. adressen som feeden är tillgänglig via).
    - Hör till en kategori
    - Har ett uppdateringsintervall (dvs hur ofta applikationen ska kolla på länken för att se om det publicerats några nya podcasts).
    - Har ett antal avsnitt
    - Har ett namn (tex. SVT-nyheter) som kan användaren skriva in.
- Användaren skall kunna klicka på en podcast för att sedan göra följande funktionalitet:
    - Ändra en befintligt podcast namn

    - Ändra kategori och updateringsintervall på vald podcast
    - Ta bort podcast med all tillhörande data
    - Visa en lista av alla avsnitt av den vald podcast
- När programmet körs ska applikationen kolla efter nya avsnitt i XML feeden enligt det tidsintervall som användaren angett och lägga till i podcast:en. Tre uppdateringsintervall skulle räcka tex 10 sekunder, 30 sekunder och 1 minut.

- Podcastavsnitt

Avsnittsinformation ska visas i en lista av valfritt utseende. Information ska inkludera beskrivning (dvs. <description> av XML feed dokumentet) till varje avsnitt. Detta ska utföras genom att användaren klickar på podcast elementet i en lista för att se en detaljerad vy av innehållet.

- Kategorier

Kategorier används för att sortera podcasts efter ämne, till exempel hälsa, musik, programmering, osv.
Det ska vara möjligt att användaren:

- Lägga till en kategori
- Skall kunna ändra namn på en kategori. När användaren ändrar namn på en kategori skall tillhörande podcast kategori också ändras.
- Skall kunna ta bort en kategori. Applikationen ska också ta bort podcast(s) som har den kategori. För att utföra detta ska applikationen fråga användaren om hen är säker på att hen vill göra detta och om svaret är JA ska applikationen ta bort kategorin och alla podcasts som har den kategorin.
- Skall kunna sortera podcast utifrån kategori. Väljs “Musik” skall applikationen endast visa alla podcasts som hör till musik kategori.

Övrigt

Ni ska skriva er kod i enlighet med principen Clean Code, vilket innebär att koden ska vara tydlig och lätt att förstå, utan att några kommentarer behövs. Det handlar inte om att bara sluta kommentera koden, utan att se till så namnsättningen är tydlig och metoder och klasser är korta, koncisa och har ett tydligt syfte.

Det räcker alltså inte med att det fungerar utan ni måste se till så det inte finns onödig redundans, alltför komplexa metoder, klasser som har fler ansvarsområden osv i er kod. Läs på om Clean Code, Single Responsibility Principle, tre-lagersarkitektur och repository pattern innan ni börjar koda!

Grupparbetsformer

Uppgiften skall genomföras i grupper om tre personer. Det är viktigt att alla medlemmar är delaktiga i framställandet av applikationen. Det är därför alla individers ansvar att såväl ansvara för sin egen insats som att ansvara för att bereda möjligheter för övriga gruppdeltagarnas delaktighet. Arbetsformen innebär *samarbete* och sådant kräver kontinuerlig kommunikation och interaktion mellan gruppmedlemmarna. Att helt enkelt dela upp arbetet och sedan sammanstråla på slutet för att sätta ihop slutresultatet räknas inte som en samarbetsform under det här projektarbetet och kommer heller inte leda till att man som individ uppnår lärandemålen.

Det uppmuntras att söka lösningar på problem på Internet (tex StackOverflow) samt annan litteratur än kurslitteraturen. Viktigt att tänka på är dock att detta är en formell examinationsuppgift som förutsätter att det bedömningsunderlag man lämnar in är ett originalarbete.

Instruktioner för inlämning

Inlämningen ska innehålla följande:

- Komprimerad solution till en zip/rar
- Rapport (se instruktioner nedan)
- Länk till redovisnings/demonstrationsfilm. Detta ska göras genom att spela in video med demonstration av och redovisning om sin egen applikation (se instruktioner nedan).

Inlämningen sker **senast tisdag 3 november kl. 23:59**. Skicka inlämningen via Blackboardmeddelande till Panagiota Chatzipetrou. Skicka allt material i en komprimerad mapp som heter ”GRUPP_XX” där XX står för erat gruppnummer.

Sent inskickade inlämning kommer inte bedömas under ordinarie examination utan ni får då vänta till omexamination av projektet.

Rapport

En rapport skall bifogas med färdigt projekt, fokus är dock på projektet och koden. Rapporten ska skrivas gemensamt av gruppen, dvs. en rapport per projekt, och bör inte vara längre än 3 sidor. Det finns inga formalia i rapporten – det räcker med att helt enkelt svara på frågorna nedan men kom ihåg att referera korrekt om ni använder annat material i rapporten.

Rapporten ska innehålla följande delar:

- Era namn
- Erfarenheter och reflektioner från projektet
    - Var har varit bra eller dåligt?
    - Vad kunde ni ha gjort bättre?
- Kodreflektioner
    - Vad skulle ni vilja gjort annorlunda om ni haft mer tid?
    - Vilka genvägar har ni varit tvungna att ta?

Videodemonstrationen

En del av examinationen består av en videoinspelning (en s.k. skärminspelning).

Filmen bör vara 10-20 minuter lång och skapas med ett fritt valt verktyg för skärminspelning. Förslag från vår sida är att använda *QuickTime* (Mac) eller gratisversionen av *BandiCam* (PC). Skulle det inspelningsverktyg ni använder ha tidsbegränsning så går det bra att spela in flera kortare filmer. Filmen/erna kan distribueras genom att man lägger upp en länk till en plats där man lagt filmen, t.ex., YouTube, Google Drive.

Nedan följer instruktioner om att ni tre ska prata på videon och vad ni ska prata om.

- Börja videon med att berätta vad ni heter och vem som är vem.
- Genomgång av funktionella krav:
    - Utvecklare 1:
        - Ge en snabb överblick av applikationens funktionalitet (max 2 min)
        - Lägga till en ny kategori
        - Lägga till en ny podcast och tilldela detta till den nya kategorien
    - Utvecklare 2:
        - Klicka på en podcast
            - Ändra sitt namn, kategori och uppdateringsintervall
            - Ta bort en podcast
            - Visa en lista av alla avsnitt av den vald podcast
    - Utvecklare 3:
        - Klicka på ett podcastavsnitt för att visa sin information
        - Välj en kategori och ändra namn på en kategori
        - Ta bort en kategori som innehåller podcast(s)

        - Sortera podcast(s) utifrån en kategori
- Genomgång av tekniska krav:
    - Utvecklare 1:
        - Visa och förklara hur har ni använt er `virtual` och `interface` nyckelordet.
    - Utvecklare 2:
        - Visa och förklara hur har ni använt er `Async` och `LINQ`
    - Utvecklare 3:
        - Visa och förklara hur har ni använt er Exception handling och validering, och Method overloading

Betygsättning

Som slutbetyg på projektet ges betyg: Godkänd (G) och Underkänd (U).

- För att få G ett projekt ska inte innehålla mer än 1 teknisk (förutom `virtual` och `interface`) och 1 funktionellt krav som är bristande.

- Om ett projekt har mer än 2 tekniska och 3 funktionella krav som är bristande detta ska betygsättas med U.

- Om ett projekt har mer än 1 teknisk och 1 funktionellt krav som är bristande och max 2 tekniska och 3 funktionella krav som är bristande Rest kan ges för detta. Rest kan ges en gång per projekt, om resten inte är godkänd får studenten göra om projektet nästa gång kursen ges. Grupp ska få en vecka på sig att skicka in en reviderad version av projektet.

Saker att tänka på

- Snygga till koden innan ni skickar in projektet! Vi vill inte se filer som inte hör till projektet, funktioner som inte används och liknande.
- Bra kod är oftast enkel kod.
- Det finns inga krav på den grafiska designen av programmet. Försök dock att skapa ett tydligt och lättanvänt GUI till er applikation (se Bild 1 nedan för inspiration). Men kom ihåg att hellre snygg kod än snyggt UI.

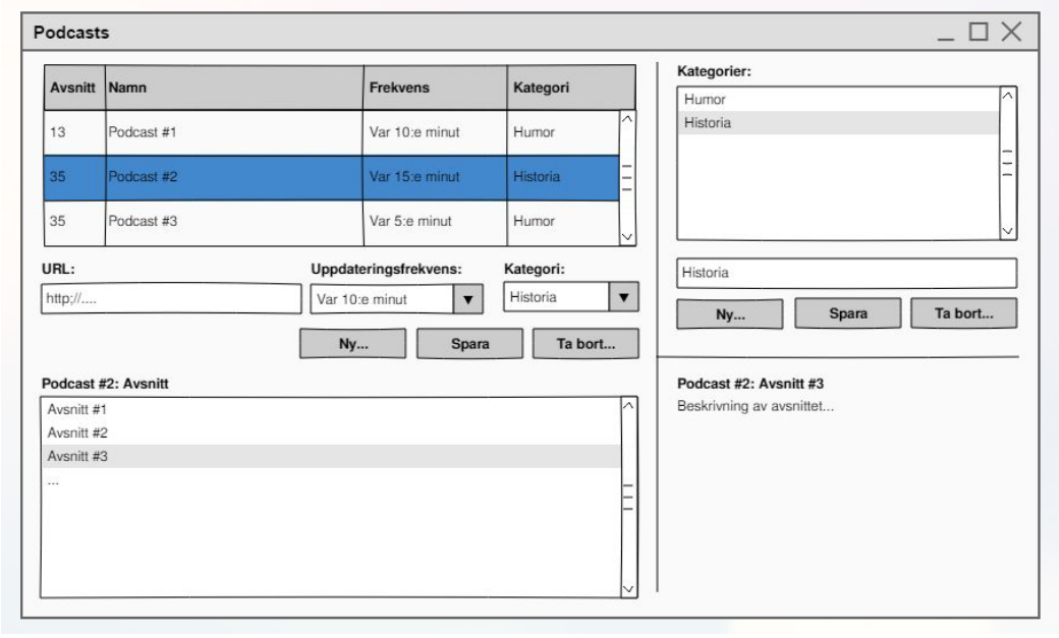

Bild 1. Exempel vy av applikationen RSS-läsare