CCWallet - A Crypto Currency Wallet

Sebastian Owuya

Innehåll

1 Introduktion **3**

2 NuGet-paket **3**
2.1 Newtonsoft.Json . . . 3
2.2 Microsoft.EntityFrameworkCore . . . 3
2.3 CountryFlag . . . 3

3 Klasser **4**
3.1 Crypto . . . 4
3.2 Wallet . . . 4
3.3 ApiQuery . . . 5
3.4 CCWalletContext . . . 5

4 Applikationen **5**
4.1 Konstruktor, Initialisering och Fält . . . 6
4.2 Menyn . . . 7
4.2.1 Refresh . . . 7
4.2.2 Wallets . . . 7
4.2.3 Vald Valuta . . . 8
4.3 Marknaden . . . 9
4.4 Plånboken . . . 9

5 Resultat **11**

6 Framtida förbättringar **11**

1 Introduktion

De senaste åren har det skett en radikal teknologisk utveckling och flera aspekter i vår vardag har digitaliserats. Räkningar och kvitton kommer nu i den digitala brevlådan, pengar förs över mellan människor via en enkel knapptryckning i telefonen och på snabbmatskedja kan man beställa sin mat utan att prata med en kassaansvarig. I samband med detta så har digitala valutor sk kryptovalutor sett en explosionsartad utveckling och allt fler individer visar intresse för den nya teknologin. Utvecklingen är fortfarande i ett tidigt skede och alla verktyg som man kan tänkas behöva finns inte riktigt på plats än. För att komma över kryptovalutor behöver man oftast ansluta sig till en börs som hanterar dessa valutor och inte ens då är det säkert att man har tillgång till alla tillgängliga valutor som existerar. Därför är det vanligt att man har konton på flera börser, somliga som finns på lättåtkomliga centraliserade börser och andra som är decentraliserade på diverse blockkedjor. Med saldo på flera olika platser kan det vara lämpligt med ett verktyg som kan bevaka ens totala balans. CCWallet är ett verktyg där en användare enkelt kan skapa en eller flera plånböcker, föra över valuta och spara lokalt i SQL Server. Dessa plånböcker har inget egentligt värde utan agerar som en portfolio som hämtar livedata från CoinMarketCaps API. På så sätt kan saldot från flera börser samlas och valutornas utveckling observeras.

2 NuGet-paket

2.1 Newtonsoft.Json

Detta och liknande paket är troligtvis A och O i de flesta applikationer som hanterar Json-datan, då det underlättar extremt vid avläsning av Json-data och konvertering. API:t som används i applikationen returnerar en Json respons i strängform. Med hjälp av Newtonsoft så kan denna repsons avläsas, parsas och mamnipuleras för att användas på efterfrågat vis. Paketets metoder används främst i klassen *ApiQuery*.

2.2 Microsoft.EntityFrameworkCore

Entity Framework (EF) Core installeras främst för att hantera databasaccess. Med EF Core så kan modeller för databsen som är länkad till applikationen skapas och underhållas. EF Core SqlServer är installerat för det är den aktuella databasen och EF Core Tools tillåter migrationer för databasemodellerna. Det sistnämnda paketet är egentligen ingen nödvändighet vid utvecklandet av denna relativt simpla applikation, men kan snabbt komma att bli ett kärnpaket när applikationens metoder expanderas.

2.3 CountryFlag

CountryFlag är ett mindre paket som innehåller en CountryFlag- och CountryCode klass. Dessa klasser är lättvikt anpassade för att kunna användas som komponenter i XAML och skrivna i Visual Basic. I denna applikation används främst CoutnryFlag klassen, ty det är ett enkelt sätt att implementera uniforma flaggikoner som kan användas som valutasymboler. Paketet saknade dock EU flaggan

så därav modifierade jag paketet genom att lägga till den saknade flaggan och även göra alla de 10 flaggorna som används i applikationen lika stora genom omformattering och adderandet av en transparent bakgrund där det behövdes. Med dessa ändringar så är inte NuGet-paketet installerat på normalt sätt utan en ny dll-fil har skapats i en relativ path och importerats manuellt via assemblies.

3 Klasser

3.1 Crypto

Klassen Crypto är superklassen som utgör en kryptovaluta. En Crypto innehåller Name (string), Id (int), Rank (int) och Symbol (string). Dessa fält har valts genom att analysera datan i Json-responsen från CoinMarketCap-Api:t och sedan se vad som är nödvändigt för att en Crypto ska vara unik, samtidigt som datan ska kunna användas i syftet för denna applikation. Innan applikationen kunde hämta data till 5000 olika valutor så behövdes inte Id-fältet, men efter att det hade lösts så blev fältet högst relevant som unik markör.

Eftersom att Crypto är en superklass så finns det givetvis subklasser som ärver från Crypto. Dessa subklasser är CryptoInfo för kryptovalutor på marknaden och CryptoInstance för kryuptovalutor i en plånbok. Från början så var dessa separata klasser, som sedan kom att vävas ihop då mycket information var snarlik mellan plånböckerna och markaden. Förutom den ärvda fälten så innehåller en CryptoInfo Price (double) och PriceStr (string), detta för att kunna hålla det numeriska priset för en valuta med Price och det senare för att kunna binda detta numeriska värde och representera som en sträng i applikationen. CryptoInstance har liknande metoder genom fälten Amount (double), AmountStr (string), Value (string) och ValueStr (string). Fälten med ändelserna "-Str" i respektive subklasser formateras genom att använda double klassens "ToString" metod. Förutom att återge visuella strängrepresentationer i GUI:t så kan dessa fält hålla antal av en kryptovaluta och det totala värdet av denna instans av Crypto i den valda fiatvalutan[1].

3.2 Wallet

Wallet klassen är containern för en portfolio. En Wallet innehåller fälten Id (Guid), Name (string), Cryptos (lista av CryptoInstance), FiatBalance (double) och FiatBalanceStr (string). Klassen implementerar INotifyPropertyChanged interfacet för att kunna notifiera MainWindow.xaml när förändringar sker i FiatBalanceStr fältet. Detta var ett iterativt beslut när det blev för många fält som kom att läggas till i MainWindow.cs som fungerade som notifiers. Grundtanken med Wallet klassen är att den ska kunna hålla flera olika CryptoInstances och återge det totala värdet av all krypto i plånboken i den valda valutan[1]. Klassens konstruktor tar ett strängargument och skapar en plånbok med argumentet som namn, en Guid som Id skapas och är det unika fältet i klassen (även om applikationen inte tilllåter skapandet av plånböcker med samma namn), fiatbalansen sätts till 0 och en tom lista med CryptoInstance skapas.

[1]Mer information om vald valuta i avsnitt **4.2.3 Vald Valuta**

3.3 ApiQuery

Detta är en klass innehåller ett fält för API nyckeln till CoinMarketCap, ett string fält som heter Error för att kunna fånga exceptions, en ObservableCollection kallad AllCryptos som håller alla Crypto objekt som hämtas ut från API:t och en metod med två string parametrar. Metoden hämtar data från CoinMarketCap och uppdaterar AllCryptos, currency används för att kunna välja till vilken fiat valuta värdeomvandling ska ske och fiatSymbol används som ändelse eller början av en pris sträng beroende på om strängen är "kr"eller någon annan valuta. I metoden rensas först AllCryptos sedan kallas en GET metod uppkopplat mot API:t som hämtar data om de 5000 högst rankade kryptovalutorna baserat på börsvärde. Svaret är en Json sträng som parsas till JObject för att kunna iterera över datan i detta objekt. En boolean "skip" skapas följt av att kryptovalutor läses av och skapas genom ett switch case baserat på relevanta fält. Kryptovalutor som ej lästs av korrekt läggs inte till i listan för alla CryptoInfo objekt, ty skulle ett error uppstå så sätts skip till true för den berörda valutan. Skulle det vara ett större problem så uppdateras Error fältet med Exception strängen och efter metoden kallats i MainWindow.xaml.cs sker en kontroll om fältet fortfarande bara innehåller en tom sträng, om inte så betyder det att något gått snätt i API accessen och relevanta promptsträngar i applikationen uppdateras för att visa information om problemet.

3.4 CCWalletContext

Applikationen hanterar databsoperationer genom CCWalletContext klassen som ärver från System.Data.Entity.DbContext klassen. Klassen är relativt simpel och består av en default konstruktor för att EF migrations ska fungera och en konstruktor som tar en connectionstring som argument för att upprätta en anslutning mot en databas server. Eftersom att denna applikationen skapar en localdb i SQL Server så är OnConfiguring metoden hårdkodad för att utföra denna setup. Vidare innehåller klassen enbart en modell för Wallet klassen och tanken är att Wallets ska sparas i databasen och då behöver listan med CryptoInstance objekt serialiseras, vilket görs med hjälp av Newtonsoft.

4 Applikationen

Det bör nämnas att storleken på fönstret är låst till en specifik storlek för att underlätta design och lägga fokus på funktionalitet. ResizeMode är satt till "CanMinimize" vilket gör att maximerings knappen är synlig men inaktiverad. Nästan alla TextBox komponenter är fokusbara för att kunna få fokus vid relevanta MouseUp event och lyssnar på KeyDown event för att kalla på Enter_Pressed och på så sätt kunna konfirmera med hjälp av return tangenten istället för att man alltid ska behöva klicka på en knapp. Vidare är text fälten för att föra in eller ut valuta kopplade till String_Is_Number metoden via PreviewTextInput eventet. String_Is_Number kontrollerar texten i en TextBox mot en regex som returnerar true om det är en sträng som går att omvandla till en double. Om inputten inte är godkänd så uppdateras en relevant prompt text i antingen eAddBalance eller eSubBalance, beroende på om sendern är från marknaden eller plånboken. Är inputten däremot okej så töms promptsträngarna, men eftersom att plånboken inte kan ha en oändlig mängd av en

given valuta så sätts det bundna värdet i textboxen till den valda valutans mängd om användaren skulle föra in ett för högt värde.

GridView kolumnerna i både marknaden och plånboken är sorterbara mha Header_Click, sorteringen är i stort sätt hämtad från Inlämningsuppgift 2 och så är även markaden och plånbokens design. Det som lagt till i sorteringen är att metoden kontrollerar vilken GridView det är som är sender för att sortera korrekt grid och så fungerar if/else klausulerna lite som guards för att sortera efter de numeriska fälten som är relaterade till "-Str" fälten och dessutom ge de fälten en omvänd sorteringsföljd.

4.1 Konstruktor, Initialisering och Fält

Applikationen implementerar INotifyPropertyChanged och använder en salig blandning av ObservableCollections och fält som använder sig av PropertyChanged eventet för att kunna binda och signalera till MainWindow när det är dags att uppdatera datan i fönstret. Det finns totalt tre stycken ObservableCollections, currentCurrency, wallets och walletCryptos. currentCurrency innehåller enbart ett objekt och det är en CountryFlag med den valda fiatvalutan, wallets innehåller wallets alla wallets tillgängliga i databasen och walletCryptos består av innehållet i den valda plånboken. Vidare finns det en connectionstring som hämtas från App.config filen för att kunna ansluta till databasen och en instans av ApiQuery klassen, _apiManager, vars primära funktion är allCryptos kollektionen som innehåller alla kryptovalutor som hämtats från API:t,. En lista, currencyFlags med 10 stycken utvalda valutor i form av MenuItem objekt med CountryFlag objekt som Header, finns som fält och härifrån kan en Dictionary, currencySymbols länkas och uppdatera currentFiatSymbol för att återge en fiatsymbol i värde strängarna som visas i ListViewsen. Fälten som kan notifiera med PropertyChanged eventet är currentWallet, eAddBalance, eSubBalance, addCryptoStr, removeValue och promptLabel. currentWallet är den aktiva plånboken, medan resterande fält antingen är prompt strängar eller numeriska värden som sätter eller hämtar värden från textboxar för att kunna uppdatera balansen i en plånbok.

I början av konstruktorn instantieras de fält som behöver instantieras och sedan läggs en handler till för GridView headers och en för ett ButtonDown event som definieras i MouseButtonDownHandler, vars syfte enbart är för att kunna ta bort eventuella prompt texter genom att klicka vart som helst med musen. Med hjälp av currentCurrency och dess symbol så kan _apiManager uppdatera sina kryptovalutor sedan initialiseras komponenten. Därefter länkas code-behind fält med sina respektive XAML komponenter, currencyFlags med FiatMenu, walletCryptos med WalletLV och allCryptos med MarketLV, sedan sätts en default sortering för objekten i marknads ListViewen och plånboks ListViewen, samt att MarketFilter metoden sätts som delegate filter för MarketLV. Efter det så skapas eller hämtas data från databasen och slutligen så kallas SetupWalletCollection metoden som tas upp i detalj i 4.2.2.

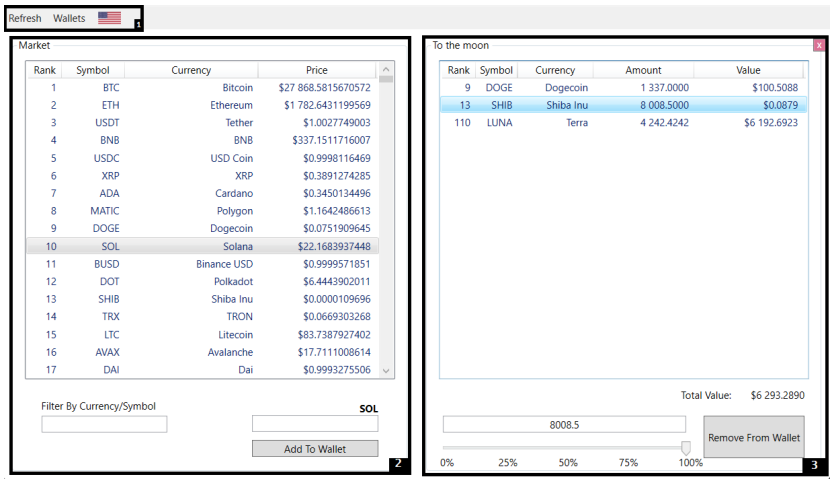

Fig. 1: En snapshot av applikationens GUI. Ruta 1 innehåller menyn, ruta 2 är marknaden och ruta 3 är plånboken.

4.2 Menyn

Menyn utgörs av en DockPanel med en Menu som innehåller tre stycken MenuItems som går att expandera i dropdowns.

4.2.1 Refresh

På menyvalet "Refresh" så kan man välja att antingen uppdatera marknaden genom att kalla på Refresh_Market metoden som i sin tur gör ett nytt anrop till API:t och hämtar uppdaterad data. Det andra alternativet är att uppdatera plånboken och då kallas Refresh_Wallet metoden som uppdaterar currentWallet, walletCryptos och databasen baserat på innehållet i plånboken och den rådande marknaden.

4.2.2 Wallets

Nästa objekt i menyn är "Wallets" som antingen används för att välja vilken plånboks innehåll man vill visa och redigera eller för att skapa en ny plånbok. Denna meny är uppbyggd med en CompositeCollection som källa som skapas code-behind i SetupWalletCollection. I XAML filen så hanteras klickeventen för alternativen genom att sätta MenuItem.ItemContainerStyle för alla MenuItems i kollektionen med ett clickevent som kopplas till Wallet_Click metoden och valje header binds till "Name" fältet för respektive element i källan. Anledningen till att en CompositeCollection används är för att få en dynamisk dropdown meny som modifieras när antalet plånböcker förändras, samtidigt som det statiska alternativet att lägga till en ny plånbok alltid ska vara tillgängligt, placerat längst ned och föregås av en Separator om minst en plånbok existerar.

I metoden så skapas först en ny global kollektion "wallets" genom att sortera den tidigare "wallets" kollektionen, följt av att sätta det första Wallet objektet

som currentWallet eller gömma och visa somliga XAML komponenter beroende på kollektionens storlek. De XAML komponenterna som påverkas av denna kontroll är knappen för att lägga till krypto från marknaden till plånboken, plånboken och den komponent som visas istället för plånboken när den är gömd.

Om kollektionen ej är tom så kallas Update_Wallet_Values metoden. Denna metod går igenom varenda CryptoInstance i currentWallet och letar rätt på motsvarande i _apiManager.AllCryptos för att uppdatera varje CryptoInstance mot den rådande marknaden, samtidigt som FiatBalance fältet i plånboken uppdateras. Som tidigare nämnts i avsnittet om Crypto klassen så formateras fälten med "-Str" ändelserna med hjälp av ToString metoden och valuta symboler läggs till på lämplig placering i strängen. Den nya uppdateringarna placeras in i wallets kollektionen på currentWallets plats och sedan så kallas Update_Wallet_Cryptos som uppdaterar walletCryptos i och med att den kollektionen är bunden mot plånboken i GUI:t, följt av att databasen uppdateras med de nya värdena i Update_Db_Wallets som även kan hantera uppladdning av nya Wallet objekt.

Wallet_Click metoden kontrollerar om det klickade menyobjektet är en Wallet eller om användaren vill skapa en ny plånbok. Har användaren klickat på en plånbok så kallas Open_Wallet metoden som bytter currentWallet mot den klickade plånboken och Refresh_Wallet kallas. Vill användaren istället skapa en ny plånbok så kallas New_Wallet_Prompt metoden. I denna metod så inaktiveras i princip alla komponenter och en prompt visas i mitten av applikationen där man kan fylla i ett plånboksnamn i en TextBox med fokus flr att skapa en ny plånbok och konfirmera med "Create Wallet" knappen eller klicka i "Cancel" för att kalla på Cancel_Prompt för att återaktivera alla komponenter och återigen dölja prompten.

Väljer användaren att fullfölja skapandet av den nya plånboken så kallas Create_Wallet metoden. I Create_Wallet kontrolleras först och främst om den relaterade TextBoxen är tom eller om texten redan är namnet på en tillgänglig plånbok. Misslyckas denna kontroll så uppdateras prompt texten med relevant information och processen avbryts. Är namnet dock okej så skapas en ny plånbok denna tomma plånbok blir currentWallet, laddas upp i databasen, läggs till i wallets kollektionen och Update_Wallet_Cryptos kallas för att uppdatera walletCryptos. Om den nya plånboken är den enda tillgängliga plånboken så kallas SetupWalletCollection för att skapa en CompCollection så att en Separator kan läggas in i menyn, annars så uppdateras wallets kollektionen och på så sätt även CompCollectionen i dropdown menyn. Detta följs av att alla XAML komponenter relaterade till prompt fönstret döljs, komponenten som visas istället för den aktiva plånboken döljs, knapparna relaterade till att föra ut valuta från plånboken döljs och att de inaktiva komponenterna aktiveras.

4.2.3 Vald Valuta

Det sista alternativet representeras av ett CountryFlag objekt som är bundet till currentCurrency[0]. Dropdown objekten är bundna via currencyFlags kollektionen bestående av CountyrFlag objet, har isCheckable satt till True och alla har ett clickevent kopplat till Select_Currency. I Select_Currency så hämtas landskoden ur det klickade objektet i dropdown menyn för att kunna skapa en

deep-copy för att byta ut currentCurrency[0], samtidigt som currentFiatSymbol uppdateras genom att hämta värdet ur currencySymbols dicionaryn. Sedan så itereras alla MenuItems i dropdownen och bockas av för att efter denna itering kunna bocka i det valda objektet igen. Slutligen kallas Refresh_Market för att uppdatera marknaden med värden i den valda valutanl, följt av ett kall till Refresh_Wallet för att uppdatera plånboken med de nya värdena.

4.3 Marknaden

Denna del av applikationen är huvudsakligen en ListView med en GridView som listar _apiManager.AllCryptos med bindningar till Rank, Symbol, Name och PriceStr fälten. Marknaden går att sortera efter kolumn, filtrera baserat på symbol och namn (samtidigt) och det går att föra över en given mängd kryptovaluta till den aktiva plånboken. Filtreringen sker genom delegate metoden MarketFilter som lyssnar den efter TextChanged events, därmed räcker det med att enbart föra in text i TextBoxen för att marknaden ska filtreras. Klickar man på en valuta så kallas MarketLv_MouseUp som visar en Grid med ett TextBlock som innehåller den valda valutans symbol, en TextBox för mängden valuta att föra till plånboken som får fokus och en knapp för att konfirmera mängden valuta att föra över till plånboken.

Vill användaren föra över valuta till plånboken så kallas AddCrypto_Click metoden. Här kontrolleras först att en plånbok är vald, att en valuta är vald och att det finns en numerisk mängd att föra över till plånboken. TextBoxen är kopplad till String_Is_Number metoden som åkallas vid PreviewTextInput eventet, så därav behövs ingen TryParse för att parsea mängden som förts in. Sedan kontrolleras det huruvida en CryptoInstance med samma namn som marknadens valda CryptoInfo existerar i plånboken. Finns det ingen CryptoInstance med samma namn så skapas det en ny som läggs till i plånboken och FiatBalance fältet uppdateras med värdet av den nya krypton. Finns däremot redan en matchande krypto så uppdateras krypton med ny data från marknaden samt mängden och värdet uppdateras baserat på TextBoxens input. Därefter så uppdateras databasen och walletCryptos som är bundet till plånboken, knappfältet för att plocka ut krypto från plånboken döljs i och med att plånboken har uppdaterats och det inte längre finns ett SelectedValue i WalletLV komponenten och eAddBalance texten uppdateras för att ge användaren feedback om att krypto har lagts till plånboken.

4.4 Plånboken

Titeln för plånboks GroupBoxen är bundet till namnet på currentWallet och högst upp till höger är det en knapp för att radera en plånbok. I övrigt så är plånboken likt marknaden huvudsakligen en ListView med en GridView som listar walletCryptos med bindningar till Rank, Symbol, Name, AmountStr och ValueStr fälten samt nedanför så återfinns en TextBox som är bunden till FiatBalanceStr för att visa det totala värdet för plånbokens innehåll. Här går det att sortera efter kolumn och en given mängd kryptovaluta kan föras ut från plånboken antingen genom att fylla i en mängd i TextBoxen eller använda Slider komponenten för att välja olika fjärdedels intervall. Klickar man på en valuta så kallas WalletLv_MouseUp som visar en Grid med en TextBox för mängden

valuta att föra från plånboken som får fokus, en Slider initialt satt till värde 50 för att välja förbestämda procentvärden att föra över och en knapp för att konfirmera mängden valuta att föra över till plånboken. Det ska tilläggas att om det inte finns någon plånbok så visas istället en blank ruta med en knapp som föreslår en att skapa sin första plånbok.

Knappen uppe till höger för att radera en plånbok har ett click event kopplat till Delete_Wallet_Prompt, vars funktionalitet liknar prompt metoden när man skapar en plånbok, skillnaden är att det inte är någon TextBox som förväntar sig input utan användaren blir mött av ett val där hen kan konfirmera huruvida plånboken verkligen ska raderas eller ej. Skulle användaren välja "No" så kallas Cancel_Prompt och annars så kallas Delete_Wallet. I Delete_Wallet tas currentWallet bort från wallets och förändringen appliceras även i databasen och Grid komponenten relaterat till att föra ut valuta från plånboken döljs. Om det finns fler plånböcker väljs bara den första i listan som currentWallet och Update_Wallet_Values kallas, annars så kallas SetupWalletCollection för att skapa en ny dropdown meny för "Wallets" med enbart alternativet att skapa en ny plånbok och den tomma rutan för att skapa en plånbok ersätter informationen för den borttagna plånboken.

TextBox komponenten för att plocka ut valuta från sin plånbok är lite mer komplex än den för att lägga till krypto. Texten i boxen är bunden till removeValue som i sin tur även förändras i metoden som är kopplad till hur Slider komponenten rör på sig. För att kontrollera källan till rörelsen i Slider komponenten så har Slider_ValueChanged först en loop som castar ints till Key objekt med hjälp av Enum klassens parse metod. På så sätt går det att avgöra om en siffra, punkt eller kommatecken varit källan till förändringen av värdet och positionen, vilket är nödvändigt i dagsläget eftersom att positionen inte är bunden till removeValue eftersom att det är en sträng och Slide.Value istället förändras manuellt i funktioner där removeValue förändras. Om förändringen orsakats av en tangent så behöver därför inte removeValue förändras om slider värdet förändras, däremot om användaren har klickat på en tick så förändras removeValue proportionellt utefter dess värde eller position. Metoden WithdrawValue_TextChanged sköter det som visas i TextBoxen, främst så ser den till att värdet inte är större än innehållet av valutan i plånboken, detta är dock en extra kontroll eftersom att även String_Is_Number gör denna kontroll. Sedan så måste indexet för textmarkören plockas ut när removeValue förändras för att sedan kunna stoppas in på samma plats, görs inte detta så blir beteendet inte särskilt användarvänligt. Är removeValue tomt så sätts Slide.Value till 0, är det inte tomt så sätts antingen värdet på slidern baserat på värdet i textboxen i förhållande till det totala värdet för den valda valutan i plånboken eller så sätts slidern till 100 och även värdet i boxen till CryptoInstance objektets Amount fält.

Klickar användaren på "Remove from Wallet" knappen åkallas Remove_Value metoden och processen för att radera en mängd krypto från plånboken startar. Den valda instansen hämtas ut ur WalletLV, priset för krypton hämtas ut ur _apiManager.AllCryptos och mängden är den mängd som står i TextBoxen. Är slidern på 0 så sätts en lämplig prompt text i eSubBalance och processen avbryts. Om slidern däremot är på 100 så tas hela CryptoInstance objektet bort från currentWallet och Grid komponenten som hanterar bort-

tagning av krypto döljs i och med att WalletLV kommer att uppdateras och det därmed inte kommer finnas ett SelectedValue. I normalfallet så är slidern dock på varken 0 eller 100 och då söks index för CryptoInstance objektet upp i currentWallet fälten uppdateras på lämpligt vis och removeValue sätts till hälften av den nya mängden för att slidern skall återgå till mitten när processen utförts. Därefter uppdateras prompt texten eSubBalance för att reflektera förändringen, Update_Wallet_Values uppdaterar walletCryptos, databasen mm och WalletLV.SelectedValue blir den tidigare valda krypton som är null om den inte längre finns i plånboken.

5 Resultat

CCWallet är helt klart över förväntan, sett till den lilla tid som utvecklingen forgått. I princip så hann jag med allt som var planerat och den resulterade applikationen är stilren och funktionaliteten är där. Hittills har jag inte heller hittat några gömda buggar som får applikationen att krascha. Något som jag är extra nöjd över är hur jag hade problem med att hämta ut data från API:t och jag därför till en början nöjde mig med att endast ha med ett urval av 25 stycken valutor. När jag hade lite tid över så gick jag tillbaka och justerade lite i ApiQuery klassen och på första försöket så gick jag från att lista 25 valutor till 5000. En begränsning som påverkade besluten i hur applikationen skulle byggas var att API-nyckeln var CoinMarketCaps gratisversion med bl a en begränsning på antal requests i minuten och per dygn. Resultatet hade kunnat bli aningen bättre redan nu om sparsamheten inte behövde tas i åtanke. Självfallet finns det förbättringar att göra, men som en första iteration är jag stolt över att presentera CCWallet.

6 Framtida förbättringar

Mjukvaruutveckling är en iterativ process och det finns ständigt förändringar och förbättringar tillgängliga. I detta avsnitt listas en del förbättringar som troligtvis legat högst upp i en potentiell backlog vid fortsatt arbete med denna applikation.

- Till att börja med så hade designen varit i fokus nu när funktionaliteten är på plats och det hade därför varit fördelaktigt med en design som är någorlunda responsiv till förändringar av storleken istället för att ha en låst storlek som nu.
- Somliga variabel eller klassnamn bör förändras för att bättre beskriva deras funktion och därmed läsbarheten. Som kanske märks om man tittar i koden så är jag inte så mycket för kommentarer utan föredrar metod- och variabelnamn som är beskrivande och talar om för utvecklaren vad syftet är och vad som sker. Kommentarer används av mig helst som undantag snarare än som standard.
- Under utvecklandet av applikationen så kom placeringen av Price att ifrågasättas som ett fält som eventuellt borde flyttas över till Crypto och därmed lämna CrytptoInfo tomt. Hade man lämnat CryptoInfo tomt, så

hade det dock inte betytt att subklassen i sig inte hade renderats som onödig, utan i samma anda så hade fortfarande klassen kunna utökas med andra datafält från API:t för att göra applikationen mer omfattande.

- Knappen för att uppdatera plånboken är möjligtvis överflödig då en uppdatering av marknaden istället även borde trigga en uppdatering av plånboken.
- I AddCrypto_Click kan det uppstå mindre problem när en ny krypto läggs till, eftersom att Rank fältet är beroende på den nuvarande marknaden, vilket kan leda till att två olika CryptoInstance objekt i plånboken kan ha samma Rank. Detta går att lösa genom att sätta igång en uppdatering av både marknaden och plånboken i slutet av metoden.
- Ett annat litet problem uppstår när användaren för in egna värden att föra ut från plånboken eftersom att vid ungefär 14 decimaler så kan inte längre .Net avgöra om en float är större eller mindre vid en jämförelse. Detta resulterar i att man kan skriva in ett tal som är marginellt större än det egentliga maxvärdet. Detta kan åtgärdas genom att begränsa antalet decimaler som visas i TextBoxen.