Namn: Ahmad Abo Warda
Version 3.

1. Antaganden

Under utvecklingsprocessen av detta program har jag gjort flera antaganden, främst när det gäller designen. Jag har valt att använda en DataGridView för att visa listan av produkter i både kassa- och lager-vyerna. Min antagande var att produkt-ID kan genereras automatiskt av systemet om användaren inte anger ett eget ID. Dessutom antog jag att en knapp för att skriva ut det senaste kvittot var nödvändig. Vidare antog jag att en enda klass (Product) skulle räcka och att det inte skulle behövas flera klasser för olika typer av produkter.

Det beslutades att inte använda metadata från API:et. Vidare valdes det att visa information om synkroniseringen i knapptexten istället för texten "Sync Now". Under synkroniseringsprocessen valdes det att låsa sync-knappen för att undvika att knappen trycks flera gånger efter varandra. Det antogs också att man behöver en textruta för API-länken, detta för att kunna testa olika API versioner vid framtida uppdateringar av API:et eller för att testa "API action=error". Denna textruta låses också när en synkroniseringprocess körs.

Det antogs att synkroniseringar körs också innan returer, inte bara försäljningar. Vid auto-sync antogs det att man inte kan trycka på knappen sync-now och att ”sync now”-knappen blir en meddelanderuta för att visa information om lyckad eller misslyckas auto synkronisering.

Det gjordes också en annan antagande, om det blir fel vid auto-synkronisering så väntas 5 minuter innan nästa synkronisering, detta är för att tillåta servern att återhämta sig om den ligger nere.

Ett antagande gjordes om hur man skulle logga historisk lagerstatus. Jag valde att spara datan som text i en fil med filformatet (.awStore). För att visa grafen valdes det att skapa en ny separat flik(tab).

Produkter som inte finns i centrallagret ignoreras vid synkronisering och ingen produkt-historik sparas eller graf ritas. Det antogs att programmet inte börjar synkronisera automatiskt när det startar.

2. Översikt

AW Supermarket App är en applikation som hanterar ett lager- och kassasystem för en matvarubutik/supermarket. Med hjälp av programmet kan butikspersonalen hantera produktinventering, lägga till och ta bort produkter från lagret.

Programmet har också en innbygd kassa system med ett varukorg, detta kan användas för att sälja och returnera produkter. Programmet har även en inbyggs utskrivnings funktion som skriver ut kvittot direkt efter varje försäljning eller retur, det är också möjligt att skriva ut senaste kvitto.

Programmet har flera funktioner, såsom att hantera lagret, lägga till produkter i varukorgen och sälja produkterna till kunderna. Programmet uppdaterar automatiskt lagersaldot när en

produkt säljs, returneras eller vid mottagning av en leverans. Det går också att beställa en leverans med de produkter som butiken behöver.

Det går också att söka efter produkter i lagret med hjälp av deras namn, produktID, pris, antal och mycket mer.

Produkternas priser och antal i lager uppdateras från ett API (central lager). Det uppdateras när knappen (Sync Now) trycks eller en gång varje minut om auto sync är påslagen.

Producternas historiska priser och antal sparas lokalt i filer av filformatet .awStore och används för att visa hur produkternas antal och pris ändras under tiden.

3. Detaljerad beskrivning

Programmet är utformat enligt MVC-mönstret och består av klasserna Product, ProductList, View och Controller. View innehåller allt som behöver hanteras av gränssnittet, medan Controller kontrollerar inmatningar från View, skickar dem till backend och returnerar resultaten till View. Min Model består av både klasserna Product och ProductList.

Jag började med att designa programmets utseende och sedan implementera funktionaliteten. Först skapade jag klasserna Product och ProductList, där Product innehåller getters och setters för relevant information som produktID, namn, författare, produkttyp, pris och annat. Därefter implementerade jag BindingList och BindingSource och kopplade dem till dataGridView.

Sedan implementerade jag funktionaliteten för knapparna (Lägg till i kundvagnen och Ta bort från kundvagnen) och här behövde jag inte komma åt backend eftersom jag lägger de i en listbox. Sedan implemeterades säljknappen och där behövde man komma åt backend för att kontrollera om det finns tillräckligt med denna produkt för att kunna sälja den och därefter uppdateras antalet (quantity) med hjälp av productList.UpdateQuantity() metod och om det inte finns tillräckligt med denna produkt i lagret så visas en meddelanderuta med medelande som säger att produkten är slut i lager. Jag har också implementerat funktionaliteten för en label i kassavyn som räknar det totala priset på produkterna.

Därefter implementerade jag returknappen, som är motsatsen till säljknappen. Vid en retur ökar antalet produkter i lager istället för att minska och de båda använder samma metode productList.UpdateQuantity().

Sedan implementerade jag lagerdelen, som ser nästan likadan ut som kassadelen. Det innehåller också en listbox som används för att lägga till produkter för att skapa en ny beställning/leverans. Lagerdelen/vyn har också en dataGridView som visar alla produkter. Till skillnad från dataGridView i kassadelen kan man här välja flera rader samtidigt och lägga till dem i beställningen.

Lagerdelen har också en del där personalen kan lägga till nya produkter genom att först välja produkttypen i en comboBox. De kan välja mellan böcker, spel och filmer. Därefter kan de fylla i mer information, som produktid, namn, pris och annan information om produkten, när de trycker på Add product knapp sker först flera kontroller som kontrollerar att de har fyllt i rätt. Från denna del kan man också ta bort befintliga produkter genom att markera dem i dataGridView och sedan trycka på knappen "Ta bort produkt".

Det finns också en sökruta där man kan söka efter produkter med hjälp av produktID, namn, pris och annat. Slutligen implementerade jag funktionaliteten för två knappar. Man kan trycka på radioknappen för att välja mellan att visa information för nuvarande året eller nuvarande månaden. Sedan kan man trycka på knappen "Top 10" för att visa de tio bästa produkterna eller trycka på knappen "Total Sales" för att se alla de bästa produkterna i månad eller år.

Det finns en sync now knapp som uppdaterar produkternas lagerstatus och pris mot ett central lager med hjälp av en API. Det går också att slå på auto Sync funktionen som uppdaterar priser och lager statusen varje minut.
Vid försäljningen eller retur av produkter genomförs sync först därefter justeras lagerstatusen efter försäljningen/returen.
I slutet görs en uppdatering åt andra hållet, från programmet till API:et med hjälp av en API action=update.
Vid varje synkusering loggas tiden, priset och lagerstatusen för varje product. För att därefter plottas i en graf.

Variabler:

- I productList klassen har jag använt flera variabler som till exempel BindingList<product> productList och här laddas alla data från filen vid program start och det är den här listan som sparas till fil när programmet stängs.
- Jag har också List<string> csvFile vilket är en list vars element är en linje av csv Filen.
- En boolean variable var autoSyncThreadIsRunning och det används för att stoppa while loopen för autosyncThread.
- histroyChart används för att rita grafen för produkternas pris historik.
- autoSync är en ny tråd som starts när auto sync funktionen körs.

Metoder:

```
internal List<string> getAllSoldProducts()
        /* Loads all the products that were sold from the database (database1.csv) file.
         * Returns as a list of strings */
private void loadCSVFile()
        /* loadCSVFile function -> loads a csv file (database.csv) which includes all information about the products in the supermarket.
        * The function loads the info to the product list and checks if a product does not have a product id (may be cased be adding the products manualy to the csv file) it requests a
        * new unique product ID.
        * If the file is missing a new file gets created!
        */
internal int generateProductID()
        /* This function generates a new random product ID and to make it unique it compares it to all other product id's */
internal object getDataSource()
        // Returns the dataSource
internal void saveFile()
        /* Saves all the data in the productList to the file/database (database.csv) */
internal void SaveSold()
        /* Saves the number of sold products with product id and date in the productList to the file/database (database1.csv) */
```

```
internal List<string> loadSeries(string productID, string param)
        // Load the data from the relevavnt file to plot them.

private string getXmlFromAPI(string api)
        // Gets the data from api and return it as a text containing the xml data

private void saveProductPriceHistory(List<Product> tmpList)
        // Create folder priceHistory and a file {ProductID}.awStore for each product
        // Save the current time, price and quantity for each product

private List<Product> extractAndLogProducts(string response)
        // Convert the product from xml format to List of products then calls the
saveProductPriceHistory method

internal bool syncNow(string api)
        // Get the products update form the api then update them in the productList
        // If success, return true
```

Det finns många andra metoder men jag nöjer mig med att nämna de ovanstående.

4. Problem

Under projektets gång har jag stött på flera problem. Ett av dessa problem uppstod när jag försökte spara data till en fil som saknades. För att lösa detta använde jag funktionen File.Create() för att skapa filen om den inte fanns och därefter spara datan till filen. Dock fick jag alltid ett exception-felmeddelande som indikerade att filen redan var i användning av en annan process. För att lösa detta problemet sökte jag på internet och hittade en lösning genom att använda File.Create("database.csv").Close().

Jag stötte också på problem med att jämföra text som jag läste in från fil med annan text. För någon anledning hade jag fått '\n' i början av varje element i filen. För att lösa detta problemet använde jag text1.Contains(text2) istället för text1 == text2.

En annan utmaning jag stötte på var att hantera dubbla for-loopar. För att hitta en lösning testade jag olika breakpoints och använde Immediate Window för att felsöka.

Vid körning av auto sync stötte jag på ett annat problem, det var att while loopen slutar alldrig att köra trotts att jag använde Thread.Abort(). För att lösa detta problem presenterades en ny boolean variabel autoSyncThreadIsRunning för att stoppa while loopen när den behöver stoppas.

Jag stötte också på problem när jag valde att köra drawChart vid DataGridView on selection changed. drawChart började köras flera gånger när programmet startades. För att lösa detta problemet försökte jag att använda dataGridView.ClearSelectin() i formLoad event, jag provade också att tömma alla listor och variabler vid varje anrop till metoden drawChart men problemet löstes alldrig. Därför valde jag att plotta grafen när en knapp trycks istället.

Ett sista problem uppstod när jag valde en produkt som inte fanns i central lager och försökte rita en graf för den. Det skapade en exception för att det fanns ingen file med data för den valda produkten. Föt att lösa detta problem använde jag en if sats som returneras en tom lista

om fil saknas och visar en messagebox för att informera användaren att den valda produkten inte har product historik.

5. Sammanfattning

Jag kan sammanfatta att det har gått ganska bra i detta projekt. Jag har lyckats implementera alla de funktioner som var planerade och skapat en fungerande applikation enligt MVC-mönstret.

Jag upptäckte fördelarna med att använda BindingList och BindingSource för att visa data i dataGridView, vilket underlättade arbetet med att ändra datan från två olika ställen (kassan och lagerdelen). En alternativ lösning för att implementera funktionaliteten för att ändra datan från två olika ställen (kassan och lagerdelen) skulle ha varit att använda en ArrayList eller en List. Nackdelen med det är att det skulle ha krävt mer kod och mer arbete att uppdatera gränssnittet när produkter läggs till eller tas bort.

En annan alternativ lösning skulle ha varit att använda flera klasser för olika typer av produkter (Böcker, spel, filmer) i stället för en enda Product-klass. Fördelen med detta skulle ha varit att det skulle ha varit lättare att lägga till nya produkter och att hantera olika typer av produkter. Nackdelen är att det skulle ha krävt mer tid att implementera och underhålla. Jag har även börjat med att använda tre olika klasser för böcker, Spel, och Filmer med jag ändrade mig därefter och bestämt mig att använda bara en klass (Product). Jag har också lärt mig hur man plottar data och rita garfer i c# .net framwork.

6. Framtida Uppdateringar

Jag har flera idéer för att uppdatera och utveckla programmet ännu mer i framtiden.

Här är några av de:

1- Skapa en kolumn på vänstra delan av designen där visas kortfattat information om hur man kan använda en viss funktion av programmets olika funktioner. Information ändras när man pekar på olika delar av programmet med pekaren.
2- Visa pris historik i en separat fönster när man väljer en produkt och trycker på knappen plot.
3- Flytta sync now och auto sync till en dropdown (tooltip) meny istället för att ha det som en knapp.
4- Addera återställ (Reset) funktion/knapp för produkt historik för att ge användaren möjligheten att ta bort gammal data och minska storleken av programmet i datorns hårddisk.
5- Implementera funktionalliteten för att optimera användandet av tangentbordet, genom att tillexempel (tryck på s för att sälja och r för att returnera) detta är för att underlätta och försnabba användandet av programmet för kassa personalen och skapa en bättre upplevelse för kunder.
6- Bestämma vilken valuta butiken använder och göra det möjligt att använda flera valutor.
7- Förbättra utseendet av statistiken genom att använda grafer för att visa de bästa produkterna i butiken.

8- Optimera sync funktionen genom att bara köra sync när priset och lagerstatusen har ändrats.
9- Visa Datum och tid i en av hörnen i programmet.

Ungefärlig tidsåtgång för de olika momenten:

- Design av gränssnittet: 1 dag
- Implementering av klasserna Product och ProductList: 1 dag
- Implementering av BindingList och BindingSource: 1 dag
- Implementering av funktioner för att lägga till och ta bort produkter från korgen: 1 dag
- Implementering av funktioner för att sälja och returnera produkter: 1 dagar
- Implementering av lagerdelen: 2 dagar
- Implementering av sökfunktionen och statistikfunktionerna: 1 dagar.
- Convertering från .NET Core to .NET Framework : 1 dagar
- Implementering av sync och auto sync funktioner: 1 dag
- Implementering av produkt historik och graf ritning: 1 dag