**Media Shop**

Av: Fabian Fröding

# 1. Antaganden

* Användare registrerar sig med mail
* I klassen Product representeras varunumret av fältet ”id”.
* Data sparas i .txt-format.

# 2. Översikt

Vid programstart har användaren möjlighet att antingen logga in med existerande konto eller skapa nytt konto. Vid skapning av nytt konto anger användaren sin mail som användarnamn och lösenord. Användaren måste välja vilken department den tillhör.

Programmet består av tre huvudsaliga vyer:

* Inloggningsvy
* Lagervy
* Kassavy

Beroende på vilken ”department” en användare har så förs denne vidare till lager eller kassavyn. Lagervyn hanterar tillägning och bortagning av produkter, samt uppdatering av lagerstatus på produkter. Kassavyn hanterar försäljning av produkter. Båda vyerna har funktionalitet att se alla produkter och dess relaterade information.

# 3. Detaljerad beskrivning

Programmet använder sig av MVC-arkitekturen.

* Models består av domän-enheter som t.ex. Product, User.
* Views består av Windows Forms, i.e. “interfaces”.
* Controller-lagret är uppdelat i två ytterligare lager:
  + Controllers: Hanterar vilka vyer som ska visas baserat på vilka enheter som efterfrågas.
  + Repositories: Hanterar lagring och hämtning av data.

Programmet består av följande klasser:

* Domän-enheter:
  + Product: representationen av en fysisk produkt i lagret. Förutom grundläggande egenskaper som id, namn etc. så har en product även en ”type” som berättar vad för typ av produkt det är.
    - ProductType: En enum för att avgöra vilken kategori en Product tillhör.
  + User: representerar en användare. Förutom grundläggande egenskaper som id, mail etc. så har även användare en ”department” för att hjälpa systemet avgöra vilka funktionaliteter som ska finnas tillgängliga för användaren.
    - Department: En enum som avgör vilken department an User tillhör.
* Datalagring:
  + ”Repository”-klasser är det understa lagret som hanterar sparande och hämtning av data. Repositiy-klassernas ansvar och funktionalitet är baserat på CRUD-operationer (Create, read, update, delete).
  + ”Controller”-klasser använder datan som skickas från Repository-lagret för att sedan skicka den vidare till vyerna som visas för användaren.
  + Repositories är baserade på ”Interfaces” för att bevara det grundläggande ansvaret som klasserna har.
  + Data sparas i en extern textfil. En textfil för varje domän-enhet.
* Vyer:
  + Login: tillåter användare att logga in i systemet och ange sin roll.
  + Cashier View: visas efter login och när department ”SALES” valts.
  + Storage View visas efter login och när department ”STORAGE” valts.
* Grundläggande funktioner som getters/setters och liknande är inte inkluderat i klassdiagrammet. Detta för att hålla diagrammet minimalistiskt och begripligt.
* Metoder som ärvs från parent-klasser eller interfaces är inte representerade i den ärvande klassen. Detta för att reduca redundans.
* Varje domän-enhet (t.ex. Product och User) är POCOs (Plain Old C-Object), som är klasser som endast håller information.
* Somliga metoder som återger en bool gör detta för att indikera om operationen lyckades eller inte. T.ex. om metoden ”Add(Product)” återger *true* indikerar det att tilläget av produkten lyckades medans *false* hade indikerat att tilläget misslyckades. Denna sorts funktionalitet kan komma till nytta senare i utvecklingen.

# 4. Problem

Från början var det tänkt att ha ytterligare ett ”Service”-lager mellan Controller och Repository, men detta togs bort för att undvika over-engineering då lagret inte kändes så användbart (service-layers är till för att öka separation of concerns mellan Controller och Repository).