Projektuppgift

DT071G – Programmering i C#.NET

Ted's spelbutik

Navid Ghate

Mittuniversitetet
MID SWEDEN UNIVERSITY

Campus Härnösand Universitetsbacken 1, SE-871 88. **Campus Sundsvall** Holmgatan 10, SE-851 70 Sundsvall.
Campus Östersund Kunskapens väg 8, SE-831 25 Östersund.
Phone: +46 (0)771 97 50 00. Fax: +46 (0)771 97 50 01.

Ted’s spelbutik
Navid Ghate 2024-01-15

MITTUNIVERSITETET
Avdelningen för informationssystem och -teknologi

Författare: Navid Ghate, nagh2200@student.miun.se
Utbildningsprogram: Webbutveckling, 120 hp
Huvudområde: Datateknik
Termin, år: HT, 2023

Sammanfattning

Ted’s spelbutik har begärt en digitalisering av deras spelbutik för att effektivisera verksamheten samt att bli mer konkurrenskraftiga. Digitaliseringsresan har gjorts genom att skapa en konsolapplikation med hjälp av C#.NET och JSON för datahantering för att presentera butikens spel på ett lämpligt sätt för kunder och att de kan lämna recensioner och se varandras recensioner. AI används även för att placera recensioner som positiva eller negativa. För den administrativa effektiviseringen ger applikationen ett enkelt sätt att skapa, redigera eller ta bort spel, vidare kan även administratörer även ta bort recensioner.

Innehållsförteckning

1 Introduktion

1.1 Bakgrund och problemmotivering

Ted driver en spelbutik sedan flera år tillbaka och har länge funderat på hur han ska digitalisera sin verksamhet. Det finns både en avsaknad i verksamheten när det gäller administrationen av butiken och även en avsaknad för konsumenter att kunna hitta sig till det rätta köpet. Efter en workshop med personalen kommer Ted fram till att butiken behöver ett program för att kunna hantera spelen i butiken genom att lägga till, ta bort eller redigera spel så att kunderna snabbt och enkelt kan se vad som finns tillgängligt i deras butik. Ted genomför även en kundundersökning om vad kunderna tycker att butiken saknar och kommer fram till att kunderna har svårt att lista ut vilka spel som är hetast och att kunderna gärna vill lämna efter sig recensioner. Ted kontaktar nu en IT konsultfirma för att utforska vilka alternativ butiken har för att tillämpa en plattform där både administrationen förenklas samtidigt som plattformen kan utöka relevant information till kunderna.

1.2 Detaljerad problemformulering

Ted's spelbutik står inför ett stort dilemma där inflationen har ökat butikens fasta driftkostnader vilket har minskat lönsamheten avsevärt. Samtidigt är butikens intäkter pressade med en nedåtgående trend då konkurrenter är mer lätt tillgängliga, marknadsför bättre och på så vis tar marknadsandelar. Konsumenterna plånböcker är även pressade på grund av inflationen, men även kring stressen hur framtidstron är på ekonomin. I ett ledningsgruppsmöte kommer ledningen med ett förslag om att effektivisera verksamheten på ett sätt som kan dra ner de rörliga kostnaderna i verksamheten så som personalkostnader. Ledningsgruppen ställer sig fast vid att butiken behöver genomgå en omfattande digitaliseringsresa där butiken kan effektivisera administrationen i butiken och på så sätt utföra varsel på grund av arbetsbrist. Ledningsgruppen ställer även frågan kring hur intäkter kan ökas och kommer fram till att det även här kan lösas genom att digitalisera. Genom att erbjuda kunderna en tillgänglig plattform tror ledningsgruppen att de kan öka

sin konkurrenskraft genom att kunder bevakar och jämför utbudet mellan butiken och deras konkurrenter.

Ted får i uppdrag från ledningsgruppen och ser först över med medarbetare i en workshop hur man skulle kunna minska den administrativa arbetsbördan. Utfallet av workshoppen är att det krävs en plattform där administratörerna kan snabbt gå in och lägga till, revidera eller ta bort produkter och där produkter erhåller relevant information till kunden. Vidare utför Ted en kundundersökning för att identifiera vad kunder anser är viktigt för dem när de handlar. Kundundersökningen gav en indikation att priset var en viktig del då många jämför med konkurrenter innan köp. Det var även viktigt för konsumenterna att vägledas till att köpa rätt produkt och att digitala system inte ska vara för svåra.

Efter workshoppen och kundundersökningen kontaktar Ted en konsultfirma som ska ta sig an digitaliseringsuppdraget. Efter en diskussion mellan de resultat Ted har fått från sin undersökning kommer parterna överens om att butiken kräver en plattform för både kunden och administratörerna. Kunden ska kunna se alla produkter med relevant information för att kunna avgöra deras inköpsval. För att vägleda kunderna till rätt köp ska kunden utöver att kolla på spelets pris även kunna kolla på recensioner som är gjorda av andra konsumenter. På sådant sett ska kunderna kunna avgöra om spelet är ett köp eller ej. För administratörerna kommer det behövas ett sätt att säkert kunna komma åt det för att lägga till, ta bort eller revidera produkter. Administratörerna ska även kunna revidera eller ta bort kundernas recension på ett lätt sätt för att kunna avgränsa recensioner som är olämpliga.

Ted presenterar för ledningsgruppen de fynd som han har gjort under sin undersökning och förslaget som konsultfirman har gjort, ledningsgruppen tycker att förslaget låter bra men funderar om uppdraget kan även implementera AI. Ted kontaktar återigen konsultfirman och funderar om man kan implementera AI i plattformen och får ett givande svar till att recensionerna

som kunderna lämnar in kan utvärderas av AI/ML för att uppskatta om kundens recension är positiv eller negativ i dess natur. Ledningsgruppen blir exalterade kring förslaget och finansierar uppdraget.

1.3 Avgränsningar

Projektet är avgränsad till att förhålla sig till de kursmoment som har utfärdats i kursen DT071G med viss hjälp från internetsökningar.

1.4 Författarens bidrag

Författarens bidrag till projektet är att all kod är egenskriven förutom den del som berör AI som är tagen utifrån Microsofts egna algoritmer.

2 Teori

2.1 C#

C# är ett högnivå-programmeringsspråk som skapades av Anders Hejlsberg (Microsoft) år 2000. Till en början var C# källkod då Microsoft inte hade några öppna källkoder, men 2014 öppnades C# källkod. C# är ett programmeringsspråk som erbjuder OOP (objektorienterad programmering), är ett strikt ”type” språk som kontrollerar att rätt typer matas in och ut vilket ger koden hållbarhet. C# är även likt andra programmeringsspråk som C och C++ vilket underlättar språkbytet. C# är även ett effektivt programmeringsspråk, där koden kan optimeras för att minimera användningen av minne och processorkraft, men är inte lika effektiv som andra språk. Det nyaste versionen av C# är version 12 som släpptes ut november 2023 och trenden är att en ny version släpps ut varje år tillsammans med ramverket .NET, där de nya versionerna förbättrar, utvidgar och erbjuder nya, smartare sätt att programmera. [1]

2.2 .NET

.NET är ett ramverk med öppen källkod skapad av .NET Foundation som är en organisation under Microsoft och första versionen släpptes 2016. .NET är en plattform för att utveckla och exekvera applikationer. Den stödjer flera programmeringsspråk som C#, F# och Visual basic och erbjuder utöver programmeringsspråken ”libraries” för att till exempel kunna utveckla webbapplikationer, dator och/eller mobiltelefon applikationer och moln baserade tjänster. Innan .NET släppte sin första version så fanns .NET Framework som hade liknande funktionaliteter men var allmänt mer begränsad till operativsystemet Windows. .NET erbjuder ett bredare ekosystem för utveckling, erbjuder bättre prestanda, är mer modular samt flexibel plattform och är ”cross-platform”, där MacOS och Linux stöds. [2]

2.2.1 CLI

CLI (common language infrastructure) är en teknisk standard som utvecklades av Microsoft som möjliggöra att exekvera kod i flera högnivå programmeringsspråk i olika operativsystem utan att behöva skriva om källkoden. Med hjälp av CLI kan därmed programmeringsspråk som C# exekveras med hjälp av .NET på olika operativsystem. [3]

2.2.2 CIL

CIL (common intermediate language) är de instruktionerna som är definierade av källkoden som skickas från CLI. CIL instruktionerna är exekverade av en CLR (common language runtime). .NET skickar därmed instruktionerna från källkoden till exempel från C# genom CLI till CIL där CIL kompilerar C# koden till CIL kod som skickas vidare den kompilerade koden till CLR. CIL koden är oberoende av vilket operativsystem och/eller processor man använder och är därmed det som gör .NET kompatibel till flera plattformar. [4]

2.2.3 CLR

CLR är en virtuell maskin inom .NET som hanterar exekveringen av .NET program. När CLR får CIL koden så använder CLR av ”just-in-time” kompilering där koden från CLR konverteras till maskinkod som exekveras på enhetens processor. Alla programmeringsspråk som använder sig utav .NET ramverket kompileras på CLR oavsett vilket programmeringsspråk. [5]

2.3 AI

AI (artificiell intelligens) är datorprogram som har förmågan att förutspå utfall utifrån statistiska mått. AI ska främst kunna lösa kognitiva funktioner som förmågan att lära sig utifrån historiskt data, förstå språk, lösa problem, förutspå och hitta mönster och generalisera. Exempel på AI som används praktiskt är maskininläsning, röststyrning, chattbottar, business intelligence, ansiktsigenkänning, självkörande bilar etcetera. Aktuella delområden inom AI är maskininlärning, datamining och generativ AI. [6]

2.3.1 ML

ML (machine learning) är ett delområde inom AI som utgörs av att utveckla statistiska algoritmer som kan lära sig utav stora datakällor och genom datakällan kunna göra sin bästa bedömning. ML används för att bland annat för att lösa stora språkmodeller som ChatGPT, generera bilder som Midjourney, röstigenkänning som Alexa, e-post filtrering etcetera. [7]

3 Metod

3.1 Förarbete

3.1.1 Flödesschema

Ett flödesschema kommer att användas för att underlätta utvecklingsarbetet. Flödesschemat hjälper att detaljera flödet av applikationen, utifrån användarens val inom applikationen ska flödesschemat detaljera vart användaren ska komma och vad nästa steg är. Ett flödesschema underlättar utvecklingstiden avsevärt då utvecklaren vet vad som ska utvecklas och förkortar även felsökningen vid självaste konstruktionen av applikationen, flödesschemat kan även vara en fördelaktig för framtida underhåll. [8]

3.1.2 ER Diagram

Ett ER diagram kommer att användas för att kartlägga hur informationen ska lagras och vilka kopplingar som ska finnas mellan de olika tabellerna. ER diagrammet kommer detaljera tabellernas primärnycklar, sekundärnycklar och vilken kardinalitet som är mellan tabellerna. Genom ER diagrammet underlättas konstruktionen av informationshanteringen i applikationen och eventuella brister och/eller fel kan identifieras redan vid ER diagrammet och minska felsökningen vid implementeringen av informationslagringen. [9]

3.1.3 UML diagram

Ett UML diagram kommer att användas för att ge en ungefärlig bild på hur applikationens uppbyggnad ska vara. UML diagrammet kommer innehålla klasser och dess metoder med vilka parametrar som ska in och ut från metoden. Utifrån UML diagrammet ska det vara tydligt vad som krävs från applikationen så att en utomstående utvecklare ska kunna förstå vad som efterfrågas och överlämning av den färdiga applikationen kan underlätta förståelsen av applikationens uppbyggnad. [10]

3.2 Utvecklingsverktyg

3.2.1 Utvecklingsmiljö

Konstruktionen av applikationen kommer att ske på Visual studio code. Även konstruktionen av flödesschemat, ER diagram och UML diagram kommer att ske inom visual studio code tillsammans med tillägget Draw.io.

3.2.2 Programmeringsspråk

Under utvecklingen av applikationen kommer programmeringsspråket C# att användas tillsammans med .NET. För att lagra informationen kommer JSON filer att skapas.

För AI delen av projektet kommer Microsofts ML.NET att användas som är ett tillägg för .NET som tillåter machine learning. [11]

3.2.3 Versionshantering

För att underlätta utvecklingen kommer versionshanteringssystemet Git att användas successivt för det lokala repot. Det lokala repot backas även upp i ett molnbaserat repo i Github. [12]

4 Konstruktion

4.1 Förarbete

Förarbetet ska underlätta utvecklingen av applikationen genom att grena av hur flödet av applikationen ska vara, visa hur datastrukturen på applikationen ska vara och sist vilka funktionaliteter som krävs för att uppnå de efterfrågade kraven på applikationen. Tillsammans kan materialet av förarbetet ges till en utvecklare som ska kunna avläsa hur applikationens uppbyggnad är och genom material kunna utveckla applikationen i sin helhet.

4.1.1 Flödesschema

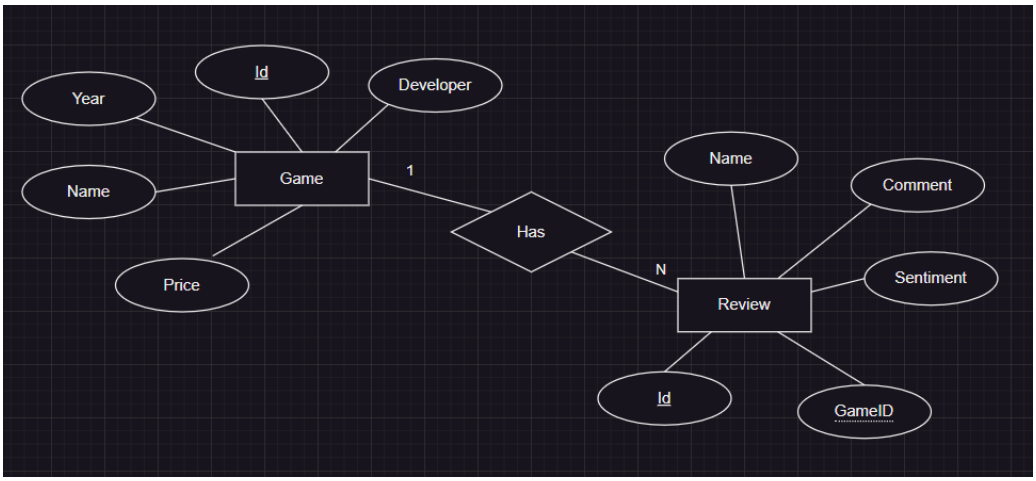
PÅ bilaga A alternativt på GitHub [12] finns ett flödesschema som ska underlätta utvecklingen av applikationen. Flödesschemat visar hur menyvalen ska vara både för en vanlig användare men även för en administratör. När applikationen startas ska programmet kontrollera om JSON filen innehåller några spel, om inga spel finns kan enbart administratörerna logga in för att lägga in spel, om spel finns så kan både användare och administratörer använda applikationen genom att bli hänvisad till en meny. Menyn för användare ger 2 (3 med att stänga applikationen) alternativ samtidigt som applikationen visar alla spel. Alternativ 1 är att se recensioner och alternativ 2 är att skapa en recension. Användaren får sedan välja spelets ID som den vill se eller skapa recension på och om ID är korrekt så visas recensionen eller så får användaren lägga in en recension. Administratörmenyn fungerar på liknande sätt där 4 (5 med att logga ut som administratör) alternativ:

1. Lägg till spel
2. Redigera spel
3. Ta bort spel
4. Recensioner

Lägg till spel gör så att administratören får lägga in information som behövs för att skapa ett spel i JSON filen. Funkar allt skapas spelet. Redigera spel och ta bort spel fungerar likt recensionerna där ett spel ID måste matas in, är det träff så får användaren redigera spelet eller så tas spelet bort. Recensioner går vidare till en annan meny för administratörer med 2 (3 med att gå tillbaka till administratörmenyn) alternativ där administratören kan antingen kolla på recensioner eller ta bort recensioner. Likt användare behöver administratören ge spelets ID och om det blir en träff så ser administratören recensionerna eller vid borttagning blir ombedd om vilket recension ID som ska tas bort, ges det ett rätt ID så tas recensionen bort.

4.1.2 ER diagram

På figur 1 kan vi se ER diagrammet för applikationen.



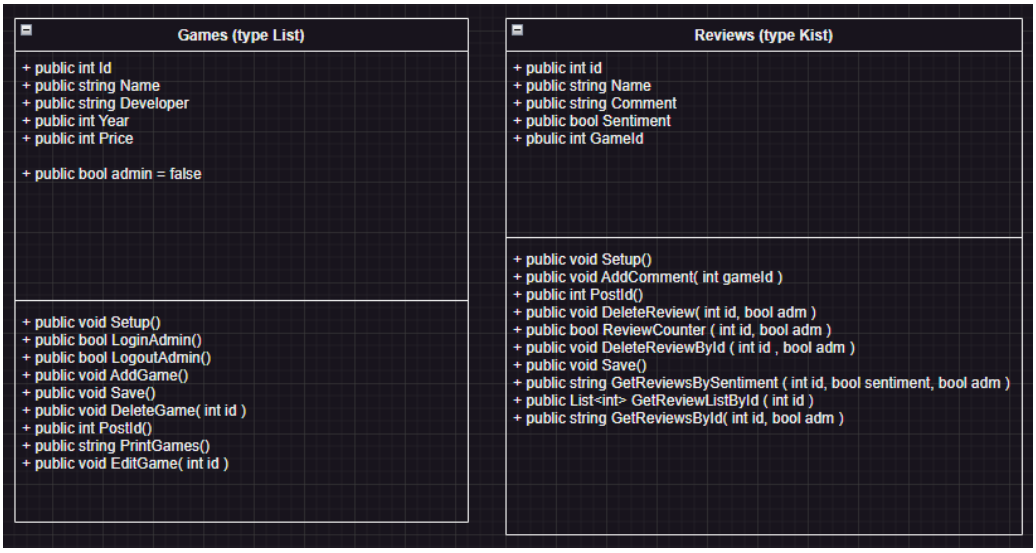
Figur 1 – ER diagram

Här ser vi att det ska finnas två tabeller, ”game” som innehåller information som en unik id som dess primärnyckel, spelets namn, spelets utvecklare, spelets pris och spelets utgivningsår. ”Review” lagrar information som användarnamn, recensionen/kommentaren, ”sentiment” som innehåller om recensionen är positiv eller negativ med hjälp av AI, en unik id som dess primärnyckel och en sekundärnyckel från tabellen ”game” som kopplar vilket spel recensionen är kopplat mot. Kardinaliteten mellan tabellerna är att ett spel kan ha flera kommentarer men att en kommentar kan bara vara riktat

mot ett spel. Tabellen Review visar även att om ett spel tas bort från en administratör så är den kopplat på ett sätt där alla recensioner som är kopplat till det spelets id kommer även tas bort.

4.1.3 UML diagram

På figur 2 kan vi se två UML diagram för applikationens klasser.



Figur 2 – UML diagram

Applikationen har två klasser, Games och Reviews. För Games ser vi att klassen kommer att ha properties som id, namn, utvecklare, utgivningsår och pris, samt en kontroll om administratör. Games har metoderna ”setup” som ska kontrollera att applikationen kan startas under rätta omständigheter, Login och logout admin som ska hantera inloggning och utloggning av administratörerna, AddGame som ska kunna möjliggöra att lägga till nya spel, Save som ska spara JSON filen, DeleteGame som ska ta bort ett spel givet rätt id nummer, PostId som ska ge ett index till nya spel, PrintGames som ska visa upp alla spel och EditGame som ska möjliggöra redigeringen av spel. När det gäller Reviews så har den properties id (för reviews), användarnamn, recensionen, sentiment och spelets id. Metoder som Reviews har är setup som på samma sätt kontrollerar att applikationen är redo att köras, AddComment som möjliggör för att lägga till recensioner, PostId som ger ett unikt id till recensionen, DeleteReview tar bort alla recensioner som är länkat till ett

spels id och går endast genom att vara inloggad som administratör, ReviewCounter som räknar antal inlägg inom ett givet spel, DeleteReviewById som tar bort recensionen baserat på recensionens id, Save sparar JSON filen, GetReviewsBySentimen ger antingen alla, positiva eller negativa recensioner, GetReviewListById ger en detaljerad lista med de valda recensionerna och GetReviewsById som ger alla recensioner baserad på spelets id.

4.2 Utvecklingen av applikationen

Utvecklingen av applikationen påbörjades genom att använda Visual studio code samt genom att installera C#.NET lokalt på datorn. När C#.NET var installerad kunde projektet påbörjas genom den inbyggda funktionen i visual studio code för att initiera ett nytt projekt. Två nya projekt skapades där den ena projektet är för att kunna köra applikationen i en konsol och där den andra ska hantera klasserna som ska brygga vad användaren ser och ”skickar” i terminalen till klassprojektet som levererar applikationens funktionaliteter. Vid senare tillfälle lades även det en till projekt som är för AI/ML delen av applikationen.

4.2.1 Utveckling av klasserna

Genom UML diagrammet så skapades ett C#.NET klassprojekt där två .cs filer skapades ”Games.cs” och ”Reviews.cs”. Dessa filer ska innehålla klasserna ”Games” och ”Reviews” som innehåller klassens egenskaper och metoder. Arbetet påbörjades först med att skapa Games filen för att utveckla bland annat inloggningen och utloggningen av administratörer, skapandet av JSON filen och kontroll att JSON filen existerar, samt funktionaliteter som efterfrågades för administratörer som att lägga till, ta bort och redigera spel. Vidare har även Games klassen en metod som skriver ut alla spel uppradade i en lista som används flitigt i applikationen. När utvecklingen av klassen Games var klar påbörjades därefter utvecklingen av klassen Reviews. Likt Games så användes även här UML diagrammet för att skapa egenskaper och metoder i Reviews för att uppnå målet. Metoder som skapades här är skapandes av JSON filen och kontroll att den existerar, sätt att lägga till eller ta bort recensioner, olika sätt att kunna se recensionerna på till exempel om

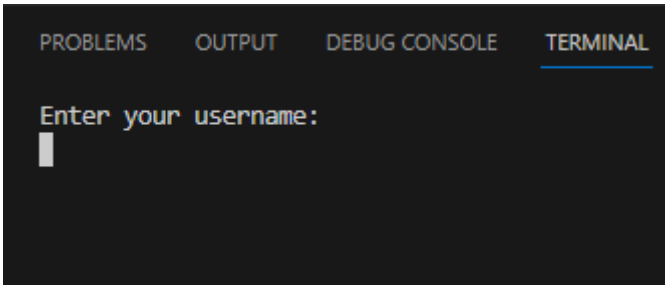
den är positiv eller negativ och interna kontroller som att kontrollera att spelet har några recensioner, vilket ID recensionen ska få etcetera.

4.2.2 Utvecklingen av konsolapplikationen

Konsolapplikationen skapades som ett separat projekt i en annan mapp än klassfilerna. Konsolapplikationen länkas därefter till klassfilerna genom att skapa en referens till klassfilerna i innehållet i konsolapplikationens projekt fil (.csproj) och på sådant sätt kunna bli tillkallad. Konsolapplikationens struktur är merparten skapad genom att visa användaren en output i konsolen och be om inputs. Genom en korrekt input från användaren utvärderar konsolapplikationen vad som ska tillkallas från klassfilerna. Figur 3 visar "huvudmenyn" när konsolapplikationen startas.

Figur 3 – Huvudmeny

Här presenteras användaren med alternativ och utifrån vad användaren lägger som input så kommer konsolapplikationen ge ett nytt output. Ett exempel ges på figur 4.



Figur 4 – Input "2" logga in som administratör

Här har användaren lagt in "2" som input och konsolen begär då ett användarnamn och sedan ett lösenord, är dessa korrekt kommer då användaren se administratörsmenyerna. Konsolapplikationen är därmed strukturerad i sin helhet på detta sätt. För de olika menyerna dvs. menyn vid start av applikationen och menyn vid inloggning som administratör eller som en användare så fungerar konsolapplikationen genom att hoppa mellan olika funktioner. Mellan funktionerna (menyerna) skickas även relevanta parametrar som vägleder vad som ska presenteras, ett exempel på detta kan vara om en användare vill se recensioner på ett spel som är positiva/negativa eller alla recensioner. Konsolapplikationen hanterar detta genom att skicka med användarens input som en parameter till en intern funktion som sedan tillkallar data från klassernas metoder. Figur 5 visar detta.

```
void revOutput(int id, bool sentiment = false, bool sentimentDummy = true, bool admin = false)
{
    Clear();


    // if sentimentdummy true, get reviews based on sentiment
    if (sentimentDummy)
    {

        WriteLine(reviews.GetReviewsBySentiment(id, sentiment, admin));
    }
    // if sentimentdummy false, get reviews based on id only I.E. all reviews on that id
    else
    {
        WriteLine(reviews.GetReviewsById(id, admin));
    }
    ReadKey();
    if (!admin)
    {
        // go back to review menu as user
        RevMenu(1);
    }
}
```

Figur 5 – Hantering för att hämta recensioner

Här ser vi att funktionen tar emot spelets id, om recensionen är positiv/negativ (default negativ), en dummy variabel som säger hämta recensionen baserat på positiv/negativ och om man är inloggad som administratör. Utifrån användarens input kommer då detta hanteras inom funktionen för att sedan kalla in rätt metod i klassen.

För alla administratörval finns det även en ”kontroll” parameter för funktioner som ska kalla på relevanta metoder. Från klassens instans hämtas informationen om användaren är inloggad som administratör, om användaren är det och vill till exempel lägga till ett spel så tillkallas funktionen i konsolapplikationen som kontrollerar att man faktiskt är inloggad och sedan tillkallar den klassmetoden som påbörjar processen för att lägga till ett spel. Figur 6 visar detta.

```
void AddGame(bool loggedIn)
{
    // check adminstatus
    if (loggedIn)
    {
        // add game method
        games.AddGame();
        // go back to adminmenu
        menu2(2);
    }
}
```

Figur 6 – AddGame, administrationskontroll

4.2.3 Utveckling av AI

Genom Microsofts egentillverkade maskininlärning tillägg ML.NET kan applikationen integrera maskininlärning på recensionerna. Microsoft erbjuder en stor datamängd på recensioner som kan användas tillsammans med ML.NET för att analysera om recensionerna i datamängden är positiva eller negativa. För denna applikation så körde ML.NET i 10 minuter där den analyserade datamängden och kom med en närmare 80% träffsäkerhet på recensionerna genom att använda sig utav ”linear logistic regression” vilket är en regression som förutsäger recensionen till bästa förmåga mellan intervallet 0 – 1, där den sedan avrundar till närmsta binära heltal och heltalet avgör då om recensionen är positiv eller negativ. Genom ML.NET skapades då ett separat projekt i en ny mapp som på liknade sätt som på konsolapplikationen länkades från klassprojektets .csproj fil till ML.NET projektet, där ett

referensnamn kunde tillkalla för att utvärdera om recensionens innehåll är positiv eller negativ. Figur 7 visar hur det praktiskt fungerar.

```
if (CommentChecker && NameChecker)
{

    // code integrated with AI
    // add inputted data
    var sampleData = new SentimentModel.ModelInput()
    {
        Col0 = CommentInp
    };
    // make a prediction based on the submitted review
    var result = SentimentModel.Predict(sampleData);
    // if AI gives 1, then the prediction of the review is positive return true else false
    var sentiment = result.PredictedLabel == 1 ? true : false;

    // create the new review
    review.Add(new Review(Id: PostId(), Name: NameInp, Comment: CommentInp, Sentiment: sentiment, GameId: gameId));

    // save
    Save();
    // clear
    Clear();
    // if the review was positive
    if (sentiment)
    {
        // positive review message
        WriteLine("Your positive review has been created, press any key to continue!");
        ReadKey();
    }
    else // negative
    {
        // negative review message
        WriteLine("Your negative review has been created, press any key to continue!");
        ReadKey();
```

Figur 7 – ML.NET i klassmetod AddComment

Här ser vi att när korrekt input har getts av en användare skickas recensionen till ML.NET projektet för att utvärdera recensionens innehåll. ML.NET gör en förutsägelse med ett binärt tal dvs. 0 eller 1. Utifrån det binära talet omvandlar vi den till ett boolean som sedan lagras i JSON filen. Vidare får användaren en output på att recensionen har skapats och att den är positiv eller negativ.

5 Resultat

Konstruktionen av konsolapplikationen har gett ett resultat som motsvarar verksamhetens krav. Verksamheten vill ha en plattform som skulle minska den administrativa bördan. Applikationen erbjuder de efterfrågade administratörsfunktionaliteterna som att lägga till, ta bort eller revidera spel som är lagrade i deras system. Vidare kan även administratörerna hantera användarnas recensioner genom att ta bort de. Applikationen har även haft säkerhet i åtanke så att funktionaliteter som administratörer ska komma åt inte kan kommas åt av obehöriga. För vanliga användare så erbjuder applikationen ett sätt för användare att se vilka spel som butiken har och ta del om information från andra användare genom recensioner. Recensioner som skapas av användare hanteras genom AI/ML för att förutsäga om recensionen är positiv eller negativ. Genom recensionen natur kan då användaren avgöra själv om spelet ska bli ett potentiellt köp. Vidare visar även applikationen alla spel som finns i butiken och relevant information kopplat till spelet såsom spelet titel, vilken utvecklare som har skapat spelet, utgivningsår och pris som ska underlätta för användaren att kunna jämför till exempel verksamhetens priser mot andra konkurrenter.

6 Slutsatser

Efter konstruktionen av applikationen så finns det några förbättringar som kunde ha applicerats på applikationen. Användandet av JSON som primär lagringsalternativ är nog inte klokt. Hade applikationen gjorts om hade det gjorts om antingen genom att använda JSON filerna för att skicka information till en riktig databas (MySQL) eller att man faktiskt använde C#.NET inbyggda funktionaliteter som kan kommunicera direkt med databaser genom inbyggda API:er. Vidare kunde även applikationen byggas som en hemsida i stället för en konsolapplikation. Då kursen har varit begränsad är min kompetens inte tillräcklig för att göra detta (än) men det skulle definitivt vara nästa steg och min misstanke är att back-end (klasserna) är någorlunda kompatibel med hemsidan, där viss revision behövs för att kunna kommunicera med en databas. Majoriteten behövs då göras om på front-end sidan.

En annan ”brist” är maskininlärningsdelen av projektet. Fastän påståendet från ML.NET om 80% träffsäkerhet så känns det snarare som att det invers dvs. 80% blir fel. En annan sak att påpeka är att maskininlärningen är tränad baserat på engelska data och inte svenska, recensioner som görs på svenska är därmed inte alls pålitliga.

Applikationen kunde även förbättras utifrån UX aspekten. Det finns möjlighet att utöka applikationen UX med att till exempel att ha sökningsfunktionaliteter som kan söka på namn, genre etcetera för om det är 100 spel i applikationen kommer det att bli svårt att hitta rätt spel. Samma gäller recensioner, kanske är man intresserad av en användare specifikt, då kanske man vill se just den användarens recensioner för att avgöra om ett spel är köpvärdigt eller ej. Som sagt, applikationen uppfyller dess syfte men har förbättringsutrymme.

Källförteckning

[1] Wikipedia, "C sharp (programming language)", https://en.wikipedia.org/wiki/C_Sharp_(programming_language), hämtad 2024-01-05, publicerad 2024-01-02.

[2] Wikipedia, ".NET", https://en.wikipedia.org/wiki/.NET, hämtad 2024-01-05, publicerad 2023-12-28.

[3] Wikipedia, "Common Language Infrastructure", https://en.wikipedia.org/wiki/Common_Language_Infrastructure, hämtad 2024-01-05, publicerad 2023-12-12.

[4] Wikipedia, "Common Intermediate Language", https://en.wikipedia.org/wiki/Common_Intermediate_Language, hämtad 2024-01-05, publicerad 2023-12-25.

[5] Wikipedia, "Common Language Runtime", https://en.wikipedia.org/wiki/Common_Language_Runtime, hämtad 2024-01-05, publicerad 2023-11-05.

[6] Wikipedia, "Artificiell intelligens", https://sv.wikipedia.org/wiki/Artificiell_intelligens, hämtad 2024-01-05, publicerad 2023-11-03.

[7] Wikipedia, "Machine learning", https://en.wikipedia.org/wiki/Machine_learning, hämtad 2024-01-05, publicerad 2024-01-04.

[8] Codesansar, "Flowcharts (Guidelines, advantages & disadvantages), https://www.codesansar.com/computer-basics/flowcharts.htm, hämtad 2024-01-05.

[9] Lucidchart, "What is an entity relationship diagram (ERD)?", https://www.lucidchart.com/pages/er-diagrams, hämtad 2024-01-05.

[10] Indeed, "What is a UML and what are its benefits?", https://ca.indeed.com/career-advice/career-development/what-is-a-uml, hämtad 2024-01-05, publicerad 2022-10-17.

[11] Microsoft, "ML.NET Tutorial – Get started in 10 minutes", https://dotnet.microsoft.com/en-us/learn/ml-dotnet/get-started-tutorial/evaluate, hämtad 2024-01-05.

[12] Github, "DT071G_Projekt", https://github.com/Cutecowboy/DT071G_Projekt, hämtad 2024-01-05.

Bilaga A: Flödesschema av applikationen

Flödesschema