Bibliotekshanteringssystem

Lawko Abdullah

NET24

1. Introduktion

Detta Projekts syfte var att utveckla en konsolapplikation som simulerar ett funktionellt bibliotekshanteringssystem. Målet var att skapa en applikation där användare kan hantera en samling utav böcker, inklusive att lägga till, ta bort och söka efter böcker samt checka ut och returnera böcker. Bibliotekshanteringssystemet skulle även vara användarvänlig och implementera korrekt datavalidering och felhantering.

2. Problembeskrivning

Projektet var att skapa och implementera ett bibliotekshanteringssystem som skulle lagra och hantera information om böcker. Systemet skall inkludera en bokklass med egenskaper som titel, författare, ISBN samt en indikation om boken var utlånad alternativ returnerad. Användaren skulle kunna:

- Lägga till nya böcker.
- Ta bort böcker baserat på ISBN,Titel.
- Söka efter böcker med hjälp av författarens namn, ISBN eller titel.
- Visa alla böcker i samlingen.
- Checka ut och returnera böcker.
- Säkerställa att alla användarinmatningar var korrekt validerade om inte dök det upp ett fel meddelande .

Dessa krav utgjorde en kritisk del för projektets struktur och funktionalitet. Detta är även de kravställningarna vi hade för att leverera produkten.

3. Lösning och Genomförande

Arbetet med bibliotekshanteringssystemet inleddes med en noggrant planerad fördelning av uppgifter. Vi delade upp ansvaret så att varje gruppmedlem kunde fokusera på olika aspekter av systemet. Dorsa Moradi agerade som scrum master och ansvarade för att fördela uppgifterna i samråd med resten av klassen. Isak fokuserade på att skapa en User-klass, vilket innebar att han utvecklade en lista för alla böcker i samlingen samt en metod för att söka efter böcker baserat på författare. Mikaels uppgift var att implementera en metod för att skriva ut tillgängliga böcker, samt funktioner för att lägga till och ta bort böcker. Han arbetade också med en switch-sats för att söka efter böcker och stänga av programmet. Ikran ansvarade för att säkerställa att program.cs fungerade korrekt och för att skapa en lista över lediga böcker, samt funktioner för att söka efter, checka ut och återlämna böcker. Lawkos del var att implementera en funktion för att checka ut böcker man vill låna samt böcker man vill returnera.

Bokklass och Datastruktur

Skapandet av en bokklass, Book.cs, som innehöll attribut för varje bok, inklusive titel, författare, ISBN och status för om boken var utlånad. Denna klass blev central för vår

datalagring. Vi valde att använda en List<Book> för att möjliggöra en dynamisk hantering av böcker.

Användargränssnitt och Interaktion

I User-klassen implementerade vi användargränssnittet som styr programflödet och användarinteraktionen. Klassen innehöll en meny som presenterade olika alternativ för användaren, vilket gjorde det enkelt att navigera i bibliotekshanteringssystemet.

Implementering av Funktioner

Varje funktion i menyn hanterades i Meny-metoden. Användaren kan:

- **Lägga till en ny bok**: Denna metod kontrollerade att alla obligatoriska fält var ifyllda innan en ny bok lades till listan.
- **Söka efter böcker**: Vi implementerade en sökmetod där användaren kunde söka böcker baserat på författarens namn.
- **Visa alla böcker**: Denna funktion skriver ut en lista över alla böcker i samlingen.
- **Ta bort böcker**: Användaren kan ange titeln på en bok för att ta bort den från samlingen.

Checka Ut och Returnera Böcker

Denna delen av projektet blev jag tilldelad att komplettera , för att checka ut och returnera böcker. Dessa funktioner säkerställde att en bok bara kunde checkas ut om den var tillgänglig och kunde returneras om den var utlånad.

```
public void CheckOutBook(string title)
{
   var book = books.FirstOrDefault(b => b.Title.Equals(title,
StringComparison.OrdinalIgnoreCase));
   if (book != null && !book.IsCheckedOut)
   {
      book.IsCheckedOut = true;
      Console.WriteLine($"Boken '{book.Title}' har checkats ut.");
   }
   else
   {
      Console.WriteLine("Boken är antingen redan utlånad eller finns inte i systemet.");
```

```
    }
}

public void ReturnBook(string title)
{
    var book = books.FirstOrDefault(b => b.Title.Equals(title,
StringComparison.OrdinalIgnoreCase));
    if (book != null && book.IsCheckedOut)
    {
        book.IsCheckedOut = false;
        Console.WriteLine($"Boken '{book.Title}' har returnerats.");
    }
    else
    {
        Console.WriteLine("Boken är antingen inte utlånad eller finns inte i systemet.");
    }
}
```

Testning och Felsökning

Under utvecklingsprocessen genomförde vi en del omfattande tester för att säkerställa att alla funktioner fungerade som avsett. Vi gjorde detta genom att varje gruppmedlem testade sina egna delar av koden, Detta ledde till en effektiv felsökningsprocess där vi snabbt kunde identifiera och åtgärda eventuella problem.

Genom att följa en strukturerad arbetsprocess och hålla god kommunikation kunde vi framgångsrikt bygga och implementera ett bibliotekshanteringssystem.

4. Teknik och Verktyg

Vi använde Visual Studio som utvecklingsmiljö och .NET 6.0 för att bygga applikationen. För versionshantering använde vi GitHub, vilket underlättade samarbetet och gjorde det enkelt att spåra förändringar. Applikationen var skriven i C#, ett språk som vi har blivit bekanta med under kursen. Dessa verktyg bidrog till en effektiv utvecklingsprocess och hjälpte oss att hålla koden organiserad.

5. Utmaningar och Lärdomar

En av de största utmaningarna var att implementera datavalidering på ett effektivt sätt. Det var viktigt att säkerställa att användarinmatningar var korrekta. Vi lärde oss också vikten av undantagshantering för att fånga och hantera eventuella fel som kunde uppstå under användning av systemet.

För min del av projektet var jag under resande fot på grund utav oförutsedda personliga förändringar vilket presenterade nya utmaningar och att det fick mig vara tillgänglig efter middagen vilket var svårt att få helt ihop med gruppen men vi gjort vårt bästa med de förutsättningarna vi haft framför oss. Det jag tyckte saknades var avstämningar av kodgranskningar när en del skulle pushas det hade förhindrat errors samt missen i min kod när jag pushade sista delen vilket inte dök upp i user klassen. Detta är en av de saker jag kommer ta med mig som en nyckel del till nästa projekt samt vikten av kommunikation otroligt viktigt! , om vi haft bättre koll från allas håll hade vi kunde lösa problem snabbare och mer effektivt.

6. Slutsats & Reflektion

Uppgiften genomfördes framgångsrikt och vi lyckades skapa ett fungerande bibliotekshanteringssystem till slut. Ett tips jag skulle ge till mig själv är att planera noggrant i början av projektet för att undvika missförstånd om arbetsfördelningen samt andra fel som dök upp. Områden för förbättring inkluderar att arbeta på vår kommunikation och dokumentation under projektets gång, vilket skulle ha underlättat vårt arbete för alla.

7. Kodexempel

För att ytterligare illustrera viktiga delar av koden inkluderar jag exempel på hur vi hanterade utlåning av böcker och användarinteraktionen.

```
public void CheckOutBook(string title)
{
   var book = books.FirstOrDefault(b => b.Title.Equals(title,
StringComparison.OrdinalIgnoreCase));
   if (book != null && !book.IsCheckedOut)
   {
      book.IsCheckedOut = true;
      Console.WriteLine($"Boken '{book.Title}' har checkats ut.");
   }
   else
   {
      Console.WriteLine("Boken är antingen redan utlånad eller finns inte i systemet.");
```

```
    }
}


// Exempel på hur man returnerar en bok
public void ReturnBook(string title)
{
    var book = books.FirstOrDefault(b => b.Title.Equals(title,
StringComparison.OrdinalIgnoreCase));
    if (book != null && book.IsCheckedOut)
    {
        book.IsCheckedOut = false;
        Console.WriteLine($"Boken '{book.Title}' har returnerats.");
    }
    else
    {
        Console.WriteLine("Boken är antingen inte utlånad eller finns inte i systemet.");
    }
}
```