Projektuppgift

Programmering i C#.NET

Projekt

Annika Gadman

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.

Projekt
Annika Gadman 2025-10-29

MITTUNIVERSITETET
Avdelningen för informationssystem och -teknologi

Författare: Annika Gadman, anga2403@student.miun.se
Utbildningsprogram: Webbutveckling, 120 hp
Huvudområde: Datateknik
Termin, år: HT, 2025

Sammanfattning

I detta slutprojekt i kursen C#.NET, hos Mittuniversitetet, har jag utvecklat en konsolbaserad TODO-applikation med C# och .NET 9. Applikationen låter användaren lägga till, visa, markera som klar och ta bort uppgifter, som lagras i en lokal SQLite-databas. Projektet har fokuserat på objektorienterad programmering och modulär struktur genom klasserna TodoItem, TodoDatabase, TodoApp och ConsoleHelper. Resultatet är en stabil och användarvänlig applikation som uppfyller de funktioner som definierades i problemformuleringen och fungerar som avsett vid testning.

1 Innehållsförteckning

2 Introduktion / Problemdefinition

Syftet med det här projektet är att tillämpa och fördjupa de kunskaper jag har fått under kursen *Programmering i C# .NET* vid Mittuniversitetet. Projektet består i att utveckla en konsolbaserad To-do-applikation där användaren kan skapa, visa, uppdatera, markera och ta bort uppgifter. Alla uppgifter lagras i en lokal SQLite-databas, vilket gör att informationen bevaras mellan körningarna av programmet.

Bakgrunden till projektet är behovet av ett enkelt och tydligt exempel på hur man kan bygga en strukturerad applikation i C#. Det huvudsakliga problemet som projektet adresserar är hur man kan skapa en uppgiftshanterare med en hållbar kodstruktur och databaskoppling, samtidigt som applikationen förblir lätt att använda.

Målet är dubbelt:

1. Att skapa en fungerande applikation som löser problemet med att organisera och hantera uppgifter på ett effektivt sätt.
2. Att visa förståelse för centrala programmeringskoncept såsom objektorienterad programmering, databashantering, felhantering och interaktion via konsolen.

Applikationen riktar sig både till kursens lärare och examinatorer, som ska kunna bedöma programmets uppbyggnad och funktionalitet, samt
till studenter och nybörjare i C#, som vill se ett konkret exempel på hur man kan kombinera C#-klasser, databashantering och CRUD-operationer (Create, Read, Update, Delete) i ett fungerande projekt.

3 Teori

3.1 C# och .NET

C# är ett modernt, populärt och objektorienterat programmeringsspråk som skapades av Microsoft. Det används för många typer av applikationer, bland annat mobilappar, webbapplikationer, spel och databasanvändande program. Eftersom språket är objektorienterat möjliggör det strukturerad kod och återanvändbara komponenter. [1] I detta projekt används C# för att implementera en enkel konsolbaserad TODO-applikation.

.NET är en plattform som möjliggör körning av C#-program på olika operativsystem. Plattformen innehåller kompilatorer, runtime, klassbibliotek och ramverk för olika typer av applikationer, såsom ASP.NET för webbappar och MAUI för mobil- och desktop-appar. Projektet använder .NET 9 som bas, vilket ger tillgång till moderna funktioner samt stöd för SQLite-databas. [2]

.NET CLI (Command-Line Interface) är ett verktyg som används för att skapa, bygga och köra .NET-projekt direkt via terminalen. Kommandon som dotnet new, dotnet build och dotnet run används för att hantera projektets livscykel. CLI:n ingår i .NET SDK, som även innehåller kompilatorn och de bibliotek som krävs för att köra applikationer [3]. I projektet används Visual Studio Code som utvecklingsmiljö, där .NET CLI hanterar projektets kompilering och körning i bakgrunden.

3.2 Konsolapplikation

Projektet är en konsolapplikation, vilket innebär att användaren interagerar via en textbaserad meny och anger input direkt i konsolen. Programmet använder Console.ReadLine() för inmatning och Console.WriteLine() för output. Kontrollstrukturer som if och switch samt loopar används för att bearbeta användarens val och uppdatera programmet.

3.3 Databashantering med SQLite

SQLite används som lokal relationsdatabas för att lagra uppgifter. Den kräver ingen separat server och är lätt att integrera i projektet.

- Tabellstruktur: Varje uppgift representeras som en rad med kolumner för Id, Title, Description, IsCompleted, DueDate och CreatedDate.

- SQL-kommandon: Programmet använder INSERT, SELECT, UPDATE och DELETE för att hantera uppgifter i databasen.

I projektet använder jag Microsoft.Data.Sqlite som tilläggsbibliotek för att kommunicera med SQLite. Detta är ett exempel på ett bibliotek/tillägg som ger specifik funktionalitet utan att utgöra ett ramverk. Ramverk, så som ASP.NET Core eller MAUI, styr istället hela applikationens struktur, vilket inte används i detta projekt.

3.4 LINQ (Language Integrated Query)

LINQ (Language Integrated Query) är en inbyggd del av C# som gör det möjligt att bearbeta och filtrera data på ett deklarativt och lättläst sätt, både från databaser och från datalistor i minnet. Istället för att använda traditionella loopar och if-satser för att söka eller sortera data, kan man med LINQ skriva korta och tydliga uttryck direkt mot listor, arrayer eller resultat från databasanrop. [4]

I denna applikation används LINQ främst för att filtrera och sortera listor av uppgifter som hämtats från databasen. När TodoDatabase-klassen returnerar en lista med uppgifter (List<TodoItem>), kan dessa bearbetas vidare i programmet med LINQ. Till exempel kan man filtrera ut uppgifter som ännu inte är klara.

4 Metod

För att implementera den konsolbaserade TODO-applikation har arbetet delats upp i flera logiska steg och metoder.

4.1 Planering och design

Inledningsvis identifierades de funktioner som behövdes i applikationen:

- Visa uppgifter
- Lägga till uppgift
- Markera uppgift som klar
- Ta bort uppgift
- Visa arkiv över klara uppgifter

Applikationen delades upp i klasser med tydliga ansvarsområden:

- TodoItem – representerar en uppgift med egenskaper som titel, beskrivning, status och datum.
- TodoDatabase – hanterar all kommunikation med SQLite.
- TodoApp – innehåller logik för användarinteraktion och menyflöde.
- ConsoleHelper – ansvarar för konsolens utseende, färgmarkeringar och hjälpfunktioner som paus och rubriker.

För att visualisera applikationens struktur och relationer skapades flödesscheman och UML-diagram med Violet UML Editor (se Figur 1 under Konstruktions-delen)**.**

4.2 Databashantering

Alla uppgifter lagras i en lokal SQLite-databas (todos.db).

TodoDatabase-klassen innehåller följande metoder:

- AddTodo(string title, string description, DateTime? dueDate) – lägger till en ny uppgift.
- GetTodos() – hämtar ej klara uppgifter.
- GetCompletedTodos() – hämtar klara uppgifter (arkiv).
- RemoveTodo(int id) – tar bort en uppgift.
- markComplete(int id, bool isCompleted = true) – markerar en uppgift som klar.

Databasen bearbetas med SQL-kommandon
(INSERT, SELECT, UPDATE, DELETE). Denna lösning gör det enkelt att spara och hämta uppgifter samt att hålla applikationen uppdaterad i realtid.

4.3 Användarinteraktion

Användaren navigerar via en textbaserad meny som visas kontinuerligt. Alla val bearbetas med switch-satsern och listan över uppgifter hämtas från databasen. Uppgifter kan markeras som klara, vilket uppdaterar databasen och flyttar dem till arkivet.

Färgmarkeringar används i konsolen för att visa uppgifter som är förfallna (röd), nära förfallna (gul) eller normala (vit).

4.4 Objektorienterad struktur

Projektet använder objektorienterad programmering för att skapa modulär och återanvändbar kod.

- TodoItem representerar varje uppgift som ett objekt med egenskaper som titel, beskrivning, status och förfallodatum.
- TodoDatabase kapslar in all databashantering, vilket separerar datalager från gränssnittet.
- TodoApp – logik och användarflöde, inklusive filterfunktioner som "denna vecka".
- ConsoleHelper – separation av design och presentation från logik.

Genom denna uppdelning blir koden mer strukturerad, lättare att underhålla och utöka.

4.5 Verktyg och programvaror

Följande verktyg och programvaror har använts för att skapa applikationen:

- Visual Studio Code – kodredigering och projektstruktur.
- .NET 9 / C# – programmeringsspråk och runtime.
- .NET CLI – hantering av projektets livscykel (skapa, bygga, köra).
- SQLite – lokal databas för lagring av uppgifter.
- Microsoft.Data.Sqlite – NuGet-paket för att kommunicera med SQLite i C#.
- GitHub – versionshantering och publicering av koden.

- Violet UML Editor – skapande av UML-diagram för projektets struktur.

Denna metodik har möjliggjort en stegvis utveckling av applikationen, från planering och design till implementation, testning och presentation av funktioner.

5 Konstruktion

Arbetet med applikationen planerades och byggdes i flera steg för att skapa en tydlig och modulär struktur som uppfyller alla funktioner. Målet var att utveckla en konsolbaserad applikation där användaren kan lägga till, visa, markera, filtrera och ta bort uppgifter (todos), med permanent lagring i en lokal SQLite-databas.

5.1 Planering och struktur

Applikationen är uppdelad i fyra huvudkomponenter med tydliga ansvarsområden:

- TodoItem – en klass som representerar en uppgift med egenskaper som Id, Title, Description, DueDate, IsCompleted och CreatedAt. Klassen ansvarar enbart för datamodellen (single responsibility).
- TodoDatabase – hanterar all kommunikation med SQLite. Skapar tabellen om den inte finns och innehåller metoder för att lägga till, hämta, uppdatera och ta bort uppgifter.
- TodoApp – innehåller applikationens huvudlogik och menyflöde. Hämtar uppgifter från databasen, uppdaterar listor, hanterar filterfunktioner som “denna vecka” och anropar metoder i TodoDatabase vid användarval.
- ConsoleHelper – ansvarar för presentation och design i konsolen, t.ex. färgmarkering av uppgifter beroende på förfallodatum, rubriker, pausfunktioner och annat som förbättrar användarupplevelsen.

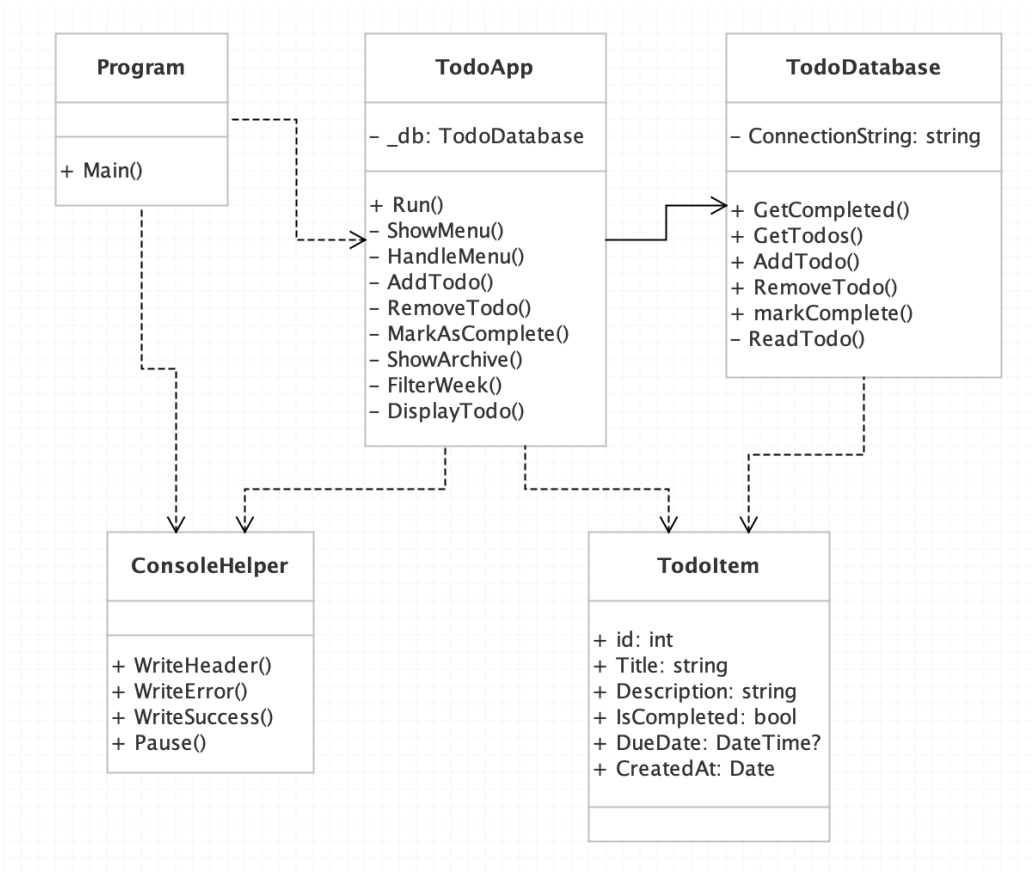
Denna uppdelning gör koden modulär, lätt att underhålla och enkel att utöka med nya funktioner.

5.2 Klassdiagram

Ett UML-diagram (skapad med Violet UML Editor) visar relationerna mellan de fyra klasserna: TodoItem, TodoDatabase, TodoApp och ConsoleHelper. Diagrammet visar alla publika metoder och attribut samt deras synlighet (+ för public, - för private).

- TodoApp använder TodoDatabase för datalagring och ConsoleHelper för presentation.
- Program innehåller endast anropet TodoApp.Run() som startpunkt.

Figur 1. UML-klassdiagram över applikationens klasser.



5.3 Databashantering

För lagring av uppgifter används SQLite, vilket innebär att all data sparas lokalt i en fil (todos.db).

- Installation av paket: Microsoft.Data.Sqlite installerades via kommandot *dotnet add package Microsoft.Data.Sqlite*
- Vid applikationens start kontrolleras om tabellen Todos finns. Om den inte finns skapas den vid behov.
- Alla SQL-kommandon (INSERT, SELECT, UPDATE, DELETE) skrivs och körs i TodoDatabase-klassen med hjälp av Microsoft.Data.Sqlite.
- Vid tillägg av en uppgift hanteras tomma fält, exempelvis DueDate, genom att spara värdet som NULL i databasen om användaren inte anger något datum.
- Vid visning av uppgifter används SELECT för att hämta antingen alla uppgifter, bara de ej klara, eller de som är markerade som klara (arkivet).
- Uppdateringar sker med UPDATE för att ändra status på en uppgift till klar.

- Borttagning sker med DELETE FROM Todos WHERE Id = ...

Denna struktur gör det enkelt att både lagra och hämta information, samt att säkerställa att alla ändringar bevaras mellan körningarna av programmet.

5.4 Bearbetning av data

Användaren interagerar med konsolen för att:

- Lägga till titel, beskrivning och valfritt förfallodatum.
- Visa uppgifter med färgmarkering: röd = förfallen, gul = nära förfallodatum, vit = övriga.
- Markera uppgifter som klara, ta bort uppgifter eller filtrera för veckan.

Användarens menyval hanteras i TodoApp genom en loop som kontinuerligt visar menyn. Valen bearbetas med en switch-sats, som anropar metoder som i sin tur kommunicerar med TodoDatabase för datahantering och ConsoleHelper för utskrift och färgmarkeringar. Resultaten presenteras direkt i konsolen.

6 Resultat

Resultatet av arbetet med TODO-applikationen blev en fullt fungerande konsolapplikation som uppfyller de mål som satts upp i problemdefinitionen. Applikationen tillåter användaren att lägga till, visa, markera och ta bort uppgifter, med permanent lagring i en lokal SQLite-databas.

6.1 Funktioner och användargränssnitt

Applikationens funktioner presenterar resultat på följande sätt:

- Ej klara uppgifter visas vid start med titel, beskrivning och förfallodatum.
- Färgmarkeringar: röd = förfallen, gul = inom 3 dagar, vit = övriga.
- Arkiv visar klara uppgifter separat.
- Filtreringsfunktion visar uppgifter för den kommande veckan.
- Förfallodatum – kan anges, uppdateras och visas för varje uppgift.

Användaren navigerar via en enkel textbaserad meny som visas efter varje åtgärd, vilket ger ett tydligt och repeterbart flöde. Alla ändringar sparas direkt i databasen, vilket säkerställer att uppgifter bevaras mellan körningarna av programmet.

6.2 Testning

För att säkerställa att applikationen fungerar som tänkt genomfördes följande tester:

- Lägga till uppgifter med/utan datum, visa uppgifter, markera som klara, ta bort uppgifter och filtrera denna vecka.
- Alla funktioner fungerar som avsett, med korrekt färgmarkering och databaskommunikation.

Alla testade funktioner fungerade som avsett, och inga kritiska fel upptäcktes.

6.3 Sammanfattning av resultat

Applikationen uppfyller alla mål, fungerar stabilt, använder SQLite korrekt och ger användaren tydlig visuell feedback via färger i konsolen.

7 Slutsatser

Under utvecklingen av TODO-applikationen stötte jag främst på utmaningar kring hantering av användarinmatning och databaskommunikation, till exempel att korrekt hantera tomma fält för förfallodatum och säkerställa att alla ändringar uppdaterades i SQLite-databasen.

Applikationen fungerar överlag bra. Jag tycker att användargränssnittet i konsolen är enkelt och tydligt, alla planerade funktioner är implementerade och databasen uppdateras korrekt. Det som kan förbättras är bland annat användarvänligheten vid felaktiga inmatningar, samt att göra gränssnittet mer dynamiskt och intuitivt.

Om jag skulle göra om projektet idag, eller om jag haft mer tid, skulle jag överväga att lägga till fler funktioner, som sortering och filtrering av uppgifter, eller ett webgränssnitt för att förbättra användarupplevelsen. Applikationen skulle även kunna utvecklas vidare genom stöd för kategorier, prioriteringar och notifikationer.

Genom arbetet med projektet har jag lärt mig hur man strukturerar en applikation med objektorienterad programmering, hur man använder SQLite tillsammans med C# och .NET, samt hur man planerar, implementerar och testar en komplett konsolapplikation från start till slut.

8 Källförteckning

[1] W3Schools, ”C# Introduction”, https://www.w3schools.com/cs/cs_intro.php
Hämtad 2025-10-27.

[2] Microsoft Learn, ” https://learn.microsoft.com/en-us/dotnet/fundamentals/”,
Hämtad 2025-10-27.

[3] Microsoft Learn, ”.NET CLI overview”, https://learn.microsoft.com/en-us/dotnet/core/tools/
Hämtad 2025-10-27.

[4] Mark J. Price, *C# 13 and .NET 9*. 9 uppl. Birmingham: Packt Publishing Ltd., 2024