**Projektuppgift**

*DT071G*

**Programmering i C#.NET: Projekt**

AI Datorer AB:s Systemadministrativa Kommandotolk

**Max Karlstedt**

**MITTUNIVERSITETET  
Avdelningen för informationssystem och -teknologi**

**Författare:** Max Karlstedt, [maka2207@student.miun.se](mailto:xxxxx0000@student.miun.se)  
**Utbildningsprogram:** Webbutveckling, 120 hp  
**Huvudområde:** Datateknik  
**Termin, år:** HT, 2023

# Sammanfattning

**Syfte:** Webbprojektet avser AI Datorer AB:s systemadministrativa intranät där en systemadministratör via en kommandotolk i C#.NET ska kunna logga in och sedan administrera användare och deras behörigheter via anslutning till ett REST API vilket i sin tur kommunicerar med en icke-relationell databas av slaget MongoDB. **Teori:** Sammanfattningsvis är C# ett starkt typat objektorienterat programmeringsspråk medan .NET är en särskild utvecklingsplattform åt just det starkt typade objektorienterade programmeringsspråket. MongoDB är en icke-relationell databasform medan REST API är ett mellangränssnitt mellan mjukvaror att kommunicera via med varann - så att C#.NET kan göra ändringar i MongoDB indirekt via REST API:t. **Metod:** Utvecklingsmiljön har varit Visual Studio 2022 för C#.NET-delen medan MongoDB Compass har använts för att kontrollera att kommandotolkens kommandon faktiskt gjort ändringar i databasen. **Konstruktion:** Olika indelade .cs-klassfiler spelar olika roller i körningen av Konsolapplikationen - exempelvis finns först en för att etablera ett anslutningsobjekt vilket sedan skickas till en annan fil som sköter inloggning mot REST API, sedan finns en annan som validerar kommandon, en ytterligare som skickar iväg CRUD-anrop mot REST API, en som loggar ut den inloggade användaren, och en som agerar som datastrukturklass för att kunna utläsa särskilt erhållen JSON-data. Det finns även en .cs-klassfil som endast döljer inmatning av ett särskilt hemligt lösenord. **Resultat:** Det framgår i presenterade bilder med de olika kommandona körda och jämförelser mot MongoDB Compass att kundens alla överenskomna kravspecifikationer har uppfyllts. **Slutsatser:** C# är ett robust och typstarkt objektorienterat programmeringsspråk samtidigt som dess objektorienterande fokus är vad som samtidigt kan vara dess svaghet för mer funktionellinriktade webbutvecklare. C# skulle kunna bli nybörjarvänligare för blivande webbutvecklare genom att erbjuda tydligare och enklare hantering av JSON vilket är den primära datatypen att kommuniceras med hjälp av på webben. Källförteckning och eventuella bilagor tillkommer.

# Innehållsförteckning

[1 Introduktion 1](#_Toc153435292)

[1.1 Bakgrund och kunden 1](#_Toc153435293)

[1.2 Kundens målgrupp 1](#_Toc153435294)

[1.3 Kundens budget 1](#_Toc153435295)

[1.4 Juridiska överväganden 1](#_Toc153435296)

[1.5 Överenskommen kravspecifikation 2](#_Toc153435297)

[1.6 Kapitelöversikt 2](#_Toc153435298)

[2 Teori 4](#_Toc153435299)

[2.1 C#.NET 4](#_Toc153435300)

[2.1.1 .NET 4](#_Toc153435301)

[2.1.2 C# 5](#_Toc153435302)

[2.2 Kommandotolk 6](#_Toc153435303)

[2.3 REST API & MongoDB 6](#_Toc153435304)

[3 Metod 8](#_Toc153435305)

[3.1 Integrerad utvecklingsmiljö 8](#_Toc153435306)

[3.2 Publicering 8](#_Toc153435307)

[4 Konstruktion 9](#_Toc153435308)

[4.1 Inloggning i Konsolapplikationen och mot REST API 10](#_Toc153435309)

[4.2 Skriva och validera skrivna kommandon 11](#_Toc153435310)

[4.3 Skicka iväg validerat kommandon 12](#_Toc153435311)

[4.4 Bearbeta REST API-svar och visa eventuell JSON-data 13](#_Toc153435312)

[4.5 Utloggning efter färdig systemadministrering 13](#_Toc153435313)

[5 Resultat 14](#_Toc153435314)

[6 Slutsatser 20](#_Toc153435315)

[Källförteckning 21](#_Toc153435316)

# Introduktion

Projektrapporten börjar med att klargöra vem kunden är, varför kunden har beställt denna IT-lösning, såväl som deras budget (det vill säga, så kallade avgränsningar i vad som ska (och inte) levereras), eventuella juridiska överväganden, samt den överenskomna kravspecifikationen mellan kunden och uppdragstagaren. Till sist i detta inledande kapitel presenteras de resterande kapitlen i form av en kapitelöversikt.

## Bakgrund och kunden

AI Datorer har bedrivit försäljning av datorkomponenter sedan några år tillbaka och nu i samband med en separat beställning av att ta fram ett MERN-stacksbaserat Intranät (se *HT2023 DT162G - Projekt av Max Karlstedt*) för att underhålla deras inventerande av datorkomponenter de har till salu så önskar de även ett separat mjukvaruprogram som enbart en utnämnd systemadministratör har tillgång till.

Med andra ord önskar kunden en så kallad kommandotolk där systemadministratören kan logga in för att sedan kommunicera direkt med företagets databas där både datorkomponenter och användare (=anställda vid företaget AI Datorer AB) finns lagrade för att på så vis styra över användarnas olika behörigheter till databasen.

## Kundens målgrupp

I denna separata beställning av en systemadministrativ kommandotolk så är det endast den utvalda systemadministratören som är kundens målgrupp. Denna utvalda person kommer att få exklusiv tillgång att kunna utläsa, skapa, ändra och/eller radera befintliga användare via den direkta åtkomsten till databasen av slaget MongoDB - allt tack vare textbaserade kommandon som skrivs i en simpel kommandotolk.

## Kundens budget

I och med att kunden redan har en separat beställning om ett MERN-stacksbaserat Intranät där utvecklingen av REST API:t så kommer denna systemadministrativa kommandotolk vara begränsad till att kunna hantera/administrera användare samt deras behörigheter för det separata Intranätet. REST API:t och MongoDB kommer dessutom att ej redogöras här mer än dess grundläggande teori.

## Juridiska överväganden

Kunden har gett samtycke till uppdragstagaren att få använda REST API:t och dess anslutna MongoDB vilket har tagits fram i en separat beställning. Beträffande känslig information så lagras endast tillfälligt en så kallad *JSON Web Token* (JWT; 2) inuti kommandotolkprogrammet vilket används för att på ett autentiserande och auktoriserande vis kommunicera med REST API:t och dess anslutna MongoDB. När systemadministratören loggar ut inuti kommandotolkprogrammet så försvinner denna JWT och då systemadministratören måste logga in på nytt igen varje gång kommandotolkprogrammet startas så kommer JWT:n att nollställas.

## Överenskommen kravspecifikation

Kunden önskar kunna göra följande i den systemadministrativa kommandotolken:

* Ha ett särskilt lösenord för att ens kunna få ange inloggningsuppgifter som systemadministratör mot REST API:t och dess anslutna databas.
* Efter lyckad inloggning ska systemadministratören kunna:
  + Utläsa enskild användare eller alla användare i en lista.
  + Ändra en användaruppgift för en vald användare i taget.
  + (Av)blockera, (in)aktivera enskilda användarkonton.
  + Logga ut enskild användare (nollställas deras tokens).
  + Ändra en behörighet för en vald användare i taget.
  + Skapa en ny användare med användaruppgifter.
  + Radera en enskild användare i taget.
  + Logga ut ur kommandotolken via ett kommando.
* Kommandon ska kunna skrivas i stilen "adduser username useremail userpassword" där första ordet är kommandot och varje mellanslag avser parametrarna efter det huvudsakliga kommandot.
* Vissa ytterst känsliga kommandon ska kräva att systemadministratören skriver in sitt lösenord igen direkt som en av de parametrar som skickas med i kommandots körning.
* Felhantering som kräver att alla parametrar för varje kommando alltid matas in innan de skickas som REST API-anrop. Det sker även kontroller på REST API-sidan som säkerhetsimplementering ifall appen skulle hamna i orätta händer.
* Stöd direkt inuti kommandotolken om hur olika kommandon ska skrivas med dess parametrar genom att "help" skrivs så att systemadministratören aldrig behöver lämna kommandotolken vid användning för att läsa på om hur något kommando ska skrivas.

## Kapitelöversikt

I kommande kapitel kommer följande att avhandlas:

[Kapitel 2](#_Teori) går igenom grundläggande teori för C#.NET, kommandotolkar i allmänhet, samt mycket kortfattat om REST API och MongoDB trots att de två sistnämnda ej är det primära fokuset i denna beställning från kunden.

[Kapitel 3](#_Metod) visar metoderna för att ens kunna genomföra det som beskrivs i de nästkommande kapitlen.

[Kapitel 4](#_Konstruktion) berör konstruktionen av kommandotolkprogrammet i C#. NET samt hur olika saker och ting har lösts för att leva upp till kundens kravspecifikationer.

[Kapitel 5](#_Resultat) presenterar bilder med olika körda kommandon i kommandotolken, hur dessa då speglats sig i REST API:t och MongoDB, såväl som hur dessa då förhållit sig till kundens kravspecifikationer.

[Kapitel 6](#_Slutsatser) är slutsatser med reflektioner om arbetet samt knyter ihop säcken för projektrapporten. Efter detta följer vedertagen källförteckning samt eventuella bilagor.

# Teori

Detta kapitel redogör grundläggande teori som behövs för att förstå de valda implementerade lösningarna åt AI Datorer AB:s framtagna Systemadministrativa Kommandotolk vilket endast systemadministratörer har behörighet till.

## C#.NET

Trots att C#.NET skrivs hopsatt genomgående i denna projektrapport så är det värt att känna till att kunna särskilja på C# och .NET där det förstnämnda beskriver ett statiskt typat objektorienterat programmeringsspråk *(se* [*2.1.2 C#*](#_C#)*)* vars första version först släpptes i februari 2002(1, s.46). .NET *(se* [*2.1.1 .NET*](#_.NET)*)* är istället en av flera olika överlappande utvecklingsplattformar för utvecklingen av applikationer och (webb)tjänster(1, s.10).

### .NET

Det så kallade .NET öppen källkod och så kallad *managed code*-baserat mjukvaruramverk för Windows-, Linux- samt macOS-operativsystem, och det drivs av anställda hos Microsoft via .*NET Foundation*(3).

Då C# är ett .NET-kompatibelt språk så kan det först kompileras till ett intermediärt språk (eng. *Common Intermediate Language; CLI - observera att CLI också kan stå för Command-Line Interface: sv. kommandotolk*) vilket i sin tur sedan kompilerar det intermediära språket till körbar maskinkod(3, *Architecture*).

.NET ska inte förväxlas med .NET Framework vilket är skyddad mjukvara som körs på Windows-baserade operativsystem utvecklade av Microsoft(4) med start redan så tidigt som på sena 1990-talet och var då en del av den så kallade ".NET-strategin"(4, *History*).

När .NET 5.0 släpptes så kunde det egentligen ha hetat .NET Core 4.0 men ordet "Core" släpptes från varumärket och versionsnumret 4.0 undveks för att .NET Framework 4.x-versioner hade funnits redan sen 2010(3, *History*).

Sammanfattningsvis så används moderna .NET för utveckling på flera olika operativsystem inklusive mobila operativsystem såsom iOS och Android, medan .NET Framework är endast för att kunna underhålla befintliga applikationer på Windows-operativsystem enbart(1, s.19).

Till sist så kan Xamarin användas för att utveckla mobila appar bland annat(1,s.19) men värt att nämna är att det kommer att bytas ut och ersättas fullständigt av .NET MAUI någon gång i maj 2024, detta meddelat av Microsoft redan den 23:e maj 2022(5).

### C#

C# anammar statiskt typat, starkt typat, lexikalt skopat, imperativt, deklarativt, funktionellt, generiskt, objekt-/klassorienterad programmering såväl som komponentbaserad programmering. Språket utvecklades av danska Anders Hejlsberg vid Microsoft under februari 2000 - med första versionen C# 1 släppt i februari 2002(1, s.46) - och från och med i november 2023 så gäller C# versionen 12.0 i .NET 8.0(6).

C# är tänkt att vara ett simpelt, generaliserat, och samtidigt modernt objektorienterat programmeringsspråk vilket innebär att det ska kontrollera typstarkt, omfångsbredd av datatyper, erbjuda automatisk skräpsamling och möjliggöra för internationalisering så att även vi i Sverige kan ta del av det än att begränsas av våra extra alfabetiska bokstäver(6, *Design goals*).

Namnet C# kommer från musikvärlden där det är den upphöjda tonen av C - varav C# - vilket härmar lite det underförstådda namnbeteendet från C++ där ++ står för en ökning av 1(6, *History*). Syntaxen i C# liknar den från C, C++ och Java bland annat. Den skiljer sig däremot från JavaScript då i exempelvis sättet elementen i arrayer placeras då måsvingar används i C# medan hakparenteser används i JavaScript(6, *Syntax*).

En potentiell nackdel med C# är dess något möjligen sämre prestanda än optimerad C eller C++-kod i och med att det måste kompileras innan det körs så tillvida inte det kan förkompileras till processorarkitekturutvald maskinkod innan(7, *Nackdelar*).

Likt majoriteten av världens programmeringsspråk så innehåller C# variabler och samtidigt i och med starktypningen i C# så måste varje variabel få en fast typ vid deklarering. Exempelvis int heltal = 1; så måste rätt typ skrivas (int vilket är förkortning för integer=heltal) medan int heltal = 0.5; ej fungerar då det är fel slags data som försöker deklareras med den valda datatypen. Med hjälp av nyckelordet var så kan en låta kompilatorn få välja lämplig datatyp utifrån det tilldelade värdet till variabeln(8, *Types*).

C# är som tidigare sagt ett objektorienterat programmeringsspråk vilket syns främst genom att allt är så kallade *klasser* med *klassmedlemmar* i form av *fält(eng. fields)*, *egenskaper(eng. properties)* och *metoder(eng. methods)*. En klass är en mall för att beskriva något medan ett *objekt* från den klassen kan skapas med en objektinstans av den klassen (t.ex. new classObject();). Olika så kallade *åtkomstmodifierare(eng. access modifiers)* såsom public, protected internal, internal, private protected, private kan sättas för att styra över vilka klassmedlemmar som kan nås både utifrån såväl som inifrån klassen själv. Med hjälp av en punkt . så kan klassmedlemmar inuti en klass nås vanligtvis men beroende på övrig åtkomstmodifikation(8, *Object-oriented programming (OOP)*).

Majoriteten av klassmedlemmarna i majoriteten av tillgängliga klasser i C# via diverse klassbibliotek har medvetet valt att kalla metoder med verb vilket beskriver något som genomförs/görs/händer medan typer, variabler, fält och egenskaper vanligen har substantivnamn precis som klassnamn bör vara med(1, s.62-63). Exempelvis Console.WriteLine(); *skriver ut med radbrytning i konsolobjektet* medan Console.WindowHeight = 100; *sätter en fönsterhöjdegenskap för konsolobjektet*. Notera samtidigt att Console är klassnamnet i form av ett substantivt *(sv. konsol)*.

Till sist är det värt att känna till att i det objektorienterade programmeringsspråket C# så får klasser endast ärva från en klass(8, *Inheritance*) till skillnad från C++ och detta designval valdes medvetet(6, *Polymorphism*).

## Kommandotolk

En kommandotolk *(eng. command-line interface; CLI)* är ett textbaserat användargränssnitt där kommandon skrivs och körs radvis(9). Markören för att visa vart kommandot ska skrivas och i vilket sammanhang (exempelvis i vilken katalog körs kommandot och/eller med vilken behörighet körs kommandot, och så vidare) det körs kallas för *prompt* och brukar vara något blinkande likt skrivmarkören i ordbehandlingsprogram(9, *Anatomy of a shell CLI*).

Syntaxen i de flesta kommandotolkar är att först skriva kommando parameter1 parameter2 parameter3 osv och sedan trycka ENTER/RETUR på tangentbordet vilket betyder att varje mellanslag är avskiljaren mellan vilket kommando som önskas köras och sedan mellan varje dess möjliga parametrar vilket är en vanlig konvention att följa för kommandotolkar(9, *Anatomy of a shell CLI*).

Vissa kommandotolkars kommandon kan erbjuda valfria parametrar vilket indikeras med särskilt tecken (exempelvis /, -, med flera) vilket då kommandotolken kan kolla efter men inte obligatoriskt kräva(9, *Command-line option*).

I och med mellanslagets roll att särskilja på var den första parametern börjar efter ett kommando såväl som var nästa parameter börjar efter första parametern och så vidare så kan det bli problematiskt att få in önskade mellanslag som en del av en parameter i vissa kommandotolkar. En lösning kan då vara att använda understreck \_ eller vänster snedstreck \ där förstnämnda ersätter mellanslag inuti menade parametrar med understrecket och där sistnämnda indikerar mellanslag som en del av en parameter genom dess vänstra snedstreck innan mellanslaget(9, *The space character*).

## REST API & MongoDB

*REST API* består av orden *REST*(10) och *API*(11) där förstnämnda beskriver en mjukvaruarkitekturstil ämnad för att appliceras på internet(10) medan det sistnämnda beskriver ett gränssnitt för datorprogram att kommunicera med varandra utan direkt känna till varandras interna detaljer(11).

*REST* står för *Representational transfer* och handlar om att en webbtjänst ska kunna tjäna webbkonsumenter utan att behöva känna till deras "tillstånd" *(eng. state)*(10) där ett exempel på detta är att om webbkonsument frågar om "sida 10" på en webbplats så är det *REST* men frågar samma webbkonsument om "nästa sida" så betyder det att webbtjänsten måste hålla reda på "tillståndet" för denna webbkonsument och därmed alla webbkonsumenter som konsumerar samma webbtjänst.

En annan sak är ordet *representational (sv. representativt)* vilket handlar om att webbtjänsten kan skicka data tillbaka i annan form än dess lagringssätt(10, *Uniform interface*) där exempel på det är att data kanske lagras som XML men skickas utåt som JSON där fältnamn till och med skulle kunna vara helt annorlunda än fältnamn som lagras internt av webbtjänsten.

*API* står för *Application programming interface (sv. applikationsprogrammeringsgränssnitt)* och handlar om ett slags "mellangränssnitt" eller ett slags "intermediärt språk" vilket två eller fler datorprogram kommunicerar via(11). Tanken med API:er är att datorprogram som vill kunna kommunicera med varandra ska inte behöva veta allt om varandra för att kunna ta del av varandras tjänster och/eller information(11, *Purpose*). Exempel är att beställa mat vid en restaurang via serveringspersonal utan att nödvändigtvis behöva få reda på eller säga till exakt hur maten sedan ska tillagas innan servering.

MongoDB - vars namn kommer från det engelska ordet *hu****mongo****us* som betyder *jättestor* med tanken att det ska kunna hantera *jättestora* mängder med data(13, s.12) - är en icke-relationsdatabas av formen *dokumentorienterad databas* där varje dokument kan betraktas som ett JSON-aktigt dokument vars dataschema inte är obligatoriskt att implementera till skillnad från exempelvis MySQL. Den går under licensformen *source-available* vilket inte är samma sak som öppen källkod(12).

MongoDB lanserades först den 11:e februari 209 medan den senaste stabila versionen är 6.0.7 släppt den 28:e juni 2023(12). Likt varje införd tupel i en relationsdatabas valda tabell(er) så är det istället *dokument* som förs in i en *kollektion* i MongoDB(13, s.192). En viktig sak med dessa är att de får uppnå 16MB i maxstorlek per dokument(13, s.193).

Tack vare att data lagras i form av JSON-liknande dokumentobjekt så är fördelen att de inte kan drabbas av SQL-injektioner då några liknande SQL-strängar inte finns att tillgå att köras på samma vis med MongoDB (13, s.13).

# Metod

Följande kapitel redogör för vilka verktyg som har använts i projektet.

## Integrerad utvecklingsmiljö

I projektet - inuti Windows 10 - har *Visual Studio Community 2022* använts som integrerad utvecklingsmiljö för att koda C#-kod. Hämtat här: [https://visualstudio.microsoft.com/#vs-section](https://visualstudio.microsoft.com/%23vs-section) (*Visual Studio Community 2022* specifikt).

Versionsaspekterna är:

* Version 17.7.6 (senast uppdatering 2023-10-31) för själva *Visual Studio Community 2022 (Help -> Check for updates -> "Current version:")*.
* Version .NET 7.0 för .NET-ramverket *(Project -> Properties -> Application -> General -> "Target framework")*.
* Version 11.0 för C#-programmeringsspråket (*Project -> Properties -> Build -> Advanced -> "Language version")*.

Det grafiska användargränssnittet *MongoDB Compass* har använts för att kontrollera att data faktiskt har ändrats i databasen via REST API:t. Hämtat här: <https://www.mongodb.com/try/download/compass>

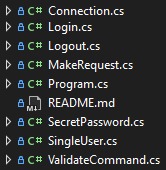
Här har MongoDB hämtats och installerats på Windows 10: <https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-windows/>

## Publicering

Projektet har ingen publicerad webbplats utan projektet finns att klona från följande GitHub repo och kan då köras lokalt så länge att NodeJS också är installerat: <https://github.com/WebbkodsLarlingen/dt071g-maka2207-projekt>

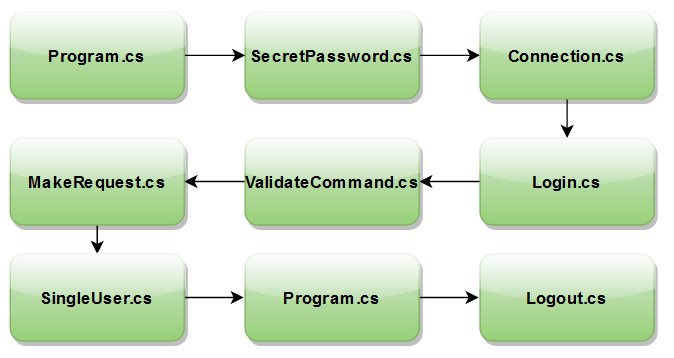
# Konstruktion

Detta kapitel avhandlar det faktiska praktiska arbetet i hur saker och ting har lösts samt varför de har lösts på de sätt de har lösts på. Det följer en hypotetisk logisk körsession av en systemadministratör av programmet som ett sätt att veta när, var, hur och varför olika kodlösningar har implementerats i den slutfärdiga Konsolapplikationen. Alla relevanta kodfiler för projektet återfinns i *Figur 1* nedan (ignorera README.md).



Figur : Filer i projektet

Den logiska körningen för .cs-filerna ovan är enligt *Figur 2* nedan där det syns att Program.cs dyker upp igen vilket är på grund av dess startpunkt såväl som att där ligger den huvudsakliga while-loopen.



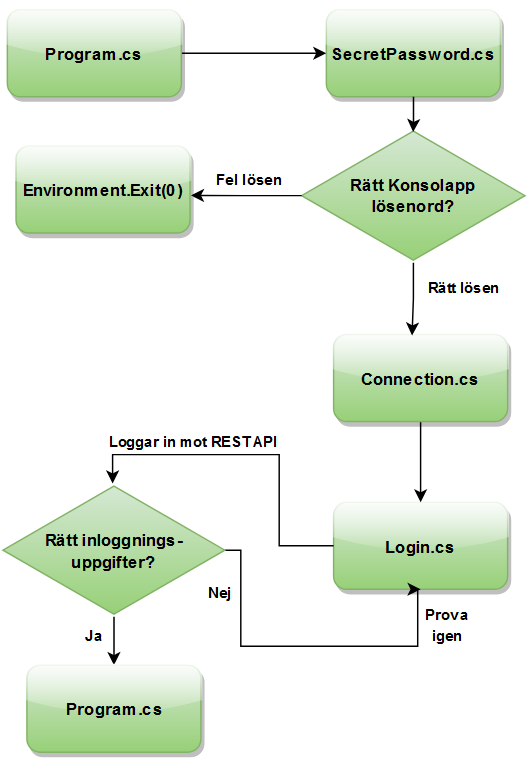
Figur : Hypotetiskt flödesschema

Nu redogörs - i respektive underrubrik i detta kapitel - kodlösningarna för de olika stegen för en typisk körning av Konsolapplikationen:

1. Logga in i Konsolapplikationen och sedan mot REST API:t.
2. Skriva kommandon som först valideras.
3. Validerat kommando skickas vidare för att skickas till REST API.
4. Mottaget svar från REST API och eventuell JSON-data bearbetas.
5. Utloggning ur Konsolapplikationen efter färdigt arbete.

## Inloggning i Konsolapplikationen och mot REST API

I *Figur 3* nedan visas flödesschemat för först inloggning i Konsolapplikationen (sköts av SecretPassword.cs) och sedan inloggning mot själva REST API:t (sköts av Connection.cs & Login.cs) för att kunna erhålla en JWT (JSON Web Token; 2) vilket används för att kunna göra övriga HTTP-anrop mot samma REST API. Inloggning sker genom att skriva användarnamn lösenord (separera med ett mellanslag) och sedan trycka på ENTER/RETUR.



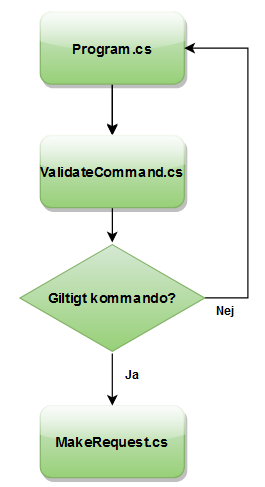
Figur : Logga in i Konsolapplikation och mot REST API

Program.cs börjar med att sätta titel för kommandotolkfönstret och sedan dess dimensioner vilket behövs för att bekvämt kunna se alla användare när kommandot showallusers körs efter lyckad inloggning mot REST API:t.

Tack vare SecretPassword.cs så kan tecken för lösenordet för att komma vidare i Konsolapplikationen döljas. Sedan skapas ett anslutningsobjekt med hjälp av Connection.cs vilket sedan används för att inuti en while-loop prova att logga in via Login.cs varav den sistnämnda kodfilen är där manuell inmatning av användarnamn och lösenord anges.

Om inloggning misslyckas så går det att prova igen eller bara stänga av konsolapplikationen manuellt. Om inloggningsuppgifterna är korrekta så kommer användaren att föras vidare i Program.cs där den förhoppningsvis inloggade systemadministratör nu kan påbörja sitt systemadministrativa arbete.

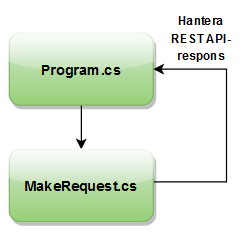
## Skriva och validera skrivna kommandon

Efter lyckad inloggning kan kommandon skrivas i Program.cs och dessa skickas vidare till ValidateCommand.cs (se Figur 4 till vänster här) vilket i sin tur - och med hjälp av fem strängarrayer - kontrollerar följande i ordningen det beskrivs: separera kommandot från eventuella parametrar och kontrollera om kommandot finns i strängarray. Lagra sedan index för där kommandot låg i strängarray så detta kan jämföras mot motsvarande index i övriga arrayer då detta säkerställer att rätt antal parametrar, rätt HTTP-metod, rätt URI mot REST API, och rätt JSON-fält skickas tillbaka om allt valideras korrekt här. Specialfall hanteras innan antalet parametrar kontrolleras mot samma index i en strängarray som index i strängarrayen där giltigt kommando ligger i. Om kommandot är giltigt så går vi tillbaka till Program.cs, gör i ordning ett JSON-objekt med rätt fältnamn och fältvärden som sedan hanteras av MakeRequest.cs. Det framgår ej i figuren till vänster men MakeRequest.cs anropas inuti Program.cs.

Figur : Skriva &   
validera kommandon

## Skicka iväg validerat kommandon

I *Figur 5* nedan visas hur Program.cs nu skickar iväg färdigskapat JSON-objekt till MakeRequest.cs vilket i sin tur då har allt som den behöver för att kunna skicka korrekt REST API-anrop: rätt JSON-data, rätt URI, och rätt HTTP-metod vald.



Figur : Skicka korrekt utformat   
HTTP-anrop till MongoDB REST API

MakeRequest.cs har en elegant kodlösning (se utdrag nedan, kodrader 22-32) där den kan välja rätt HTTP-metod och skicka iväg JSON-data (även om den är tom) vilket möjliggör att skicka GET-anrop till rätt URI, och med JSON-data vilket annars inte går med inbyggd HttpClient.GetAsync()-metod.

var reqAll = new HttpRequestMessage

{

// Choose correct HttpMethod by provided variable

Method = methodType == "get" ? HttpMethod.Get :

methodType == "post" ? HttpMethod.Post :

methodType == "put" ? HttpMethod.Put : HttpMethod.Delete,

// Set correct REST API Endpoint by provided string variable

RequestUri = new Uri("http://localhost:5000/api" + apiStr),

// Finally set any possible JSON Body data

Content = new StringContent(jsonString, Encoding.UTF8, "application/json")

};

När den skickat iväg det byggda HttpRequestMessage-objektet vilket i sin tur ligger inuti ett try-catch-block så kan Program.cs bearbeta REST API-svaret. Då kommer även eventuell JSON-data att bearbetas. Den kommer alltid att visa antingen "FEL" eller "OK" som resultat från HTTP-anropet (även om den misslyckades kommunicera med REST API:t på grund av avstängd server). Sedan börjar

## Bearbeta REST API-svar och visa eventuell JSON-data



Figur : Bearbeta REST API & JSON

I *Figur 6* ovan visas hur först Program.cs efter mottaget resultat från MakeRequest.cs (se [*steg 4.3*](#_Skicka_iväg_validerat) på föregående sida) nu använder sig av en datastruktur lagrad i klassen UserData som återfinns i SingleUser.cs. Trots att filnamnet antyder om att bara kunna visa en enskild användare så kan det ändå återanvändas genom List<UserData> då varje användare har exakt samma data från REST API:t och kan således itereras över med samma klassbaserade datastruktur. Detta sker med hjälp av kodraderna (från Program.cs, rader 230-236):

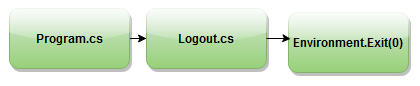
List<UserData> userList = JsonConvert.DeserializeObject<List<UserData>>(dataVal.ToString());  
foreach (UserData userData in userList)

{ // Loopa igenom här...}

Det är endast kommandona showallusers och showuser <username||useremail> som använder sig av SingleUser.cs då övriga kommandon inte hämtar något om användarna att visa i Konsolapplikationen utan endast redigera i databasen och sedan informera om resultatet.

Beträffande REST API-svaren så visas detta med färgade [OK]/[OK - FRÅN API] eller [FEL]/[FEL - FRÅN API] vilket finns i början av Program.cs på raderna 42-46. Två olika slags indikationer: att Konsolprogrammet gjorde sitt [OK/FEL] och att REST API:t gjorde sitt [OK - FRÅN API/FEL - FRÅN API].

## Utloggning efter färdig systemadministrering



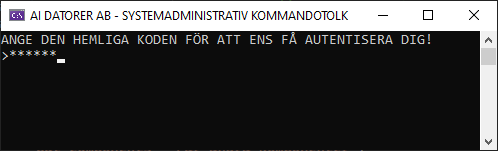
Figur : Utloggning efter färdigt arbete

I *Figur 7* ovan visas till sist hur körflödet fortfarande sker inuti Program.cs och hur Logout.cs anropas när logout skrivits som enbart kommando. Då försöker den logga ut mot REST API:t men den kommer alltid att avsluta programmet oavsett resultat från REST API:t.

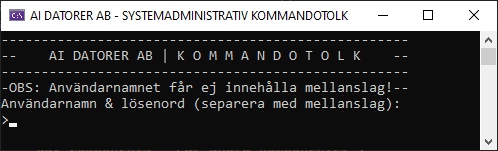
# Resultat

I detta näst sista kapitel innan slutsatser redogörs Konsolapplikationen implementerade utseende och funktionalitet samt hur användningen av den i praktiken återspeglas i MongoDB Compass-mjukvaran vilket kan visa MongoDB-data på ett mer överskådligt vis.

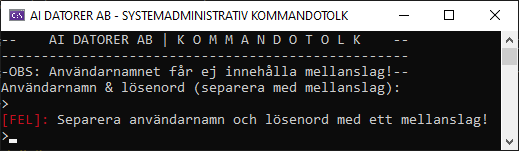
Se *Figur 8* nedan: som önskat så visas först krav på lösenord vars tecken döljs vid inmatning. Detta måste göras innan det ens går att försöka ansluta mot REST API:t. Nästa steg syns sedan i *Figur 9* nedan där användaren nu ska mata in användarnamn och lösenord med ett mellanslag som separerar dem. Därefter visar *Figur10* nedan hur fel tillhandahålls av själva Konsolapplikationen. När det står [OK/FEL - FRÅN API] så är det istället JSON-data som tolkats och omvandlats till en sträng att skriva ut.



Figur : Logga in i Konsolapplikationen

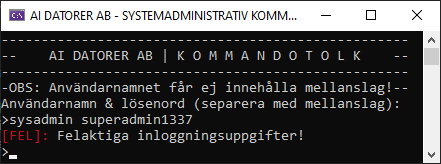


Figur : Logga in mot REST API



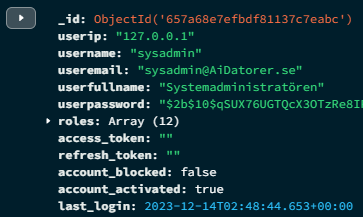
Figur : Inkorrekt inloggning #1

På nästa sida i *Figur 11* visas kommandot sysadmin superadmin1337 vilket med flit är felaktiga uppgifter (lösenordet är superAdmin1337) så endast ett litet a ska bli ett stort A så är det korrekt. Notera hur det inte står [FEL - FRÅN API] och det är för att det är Konsolapplikationen som har inbakat felmeddelande som visas då den bara inväntar på relevant HTTP-statuskod.

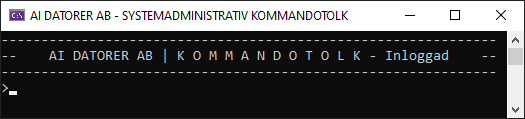


Figur : Inkorrekt inloggning #2

I *Figur 12* nedan så går det att se att sysadmin är den med flest roller (roles: Array (12)) vilket betyder att endast denna användare kan göra allt i databasen vilket inkluderar all form av systemadministrering av övriga användare. I *Figur 13* visas till sist hur det ser ut vid lyckad inloggning. Observera att bredden för Konsolapplikationen som standard egentligen är Console.WindowWidth = 230; för att kunna fullständig information för kommandona showallusers och showuser <username||useremail>.

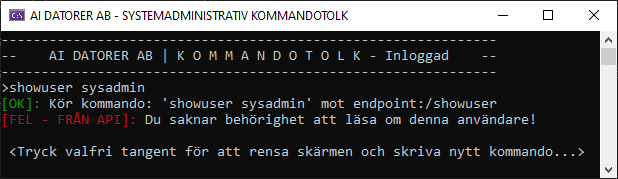


Figur : Systemadministratören ("sysadmin") i MongoDB



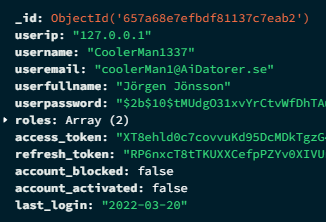
Figur : Lyckad inloggning mot REST API

Nu är det bara att skriva help som första kommando för att få en lista över alla tillgängliga kommandon, exempel på dem och vad de gör för något.

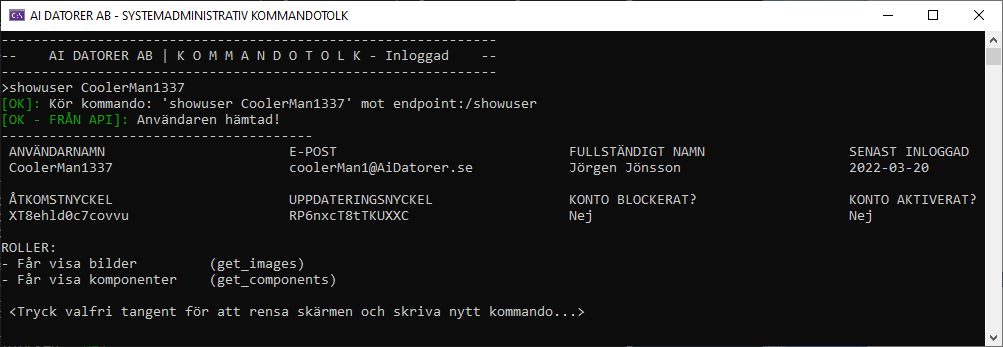


Figur : Vid otillåtna HTTP-anrop

I Figur 14 ovan visas dock först ett exempel på två saker: 1) när du får felmeddelande som kommer från REST API:t vilket visas som sagt var i [FEL - FRÅN API] och, 2) när du försöker göra CRUD som involverar själva sysadmin själv vilket aldrig är tillåtet. Det syns även [OK] som alltid visas när kommandot har validerats och vad som körs och mot vilket URI.



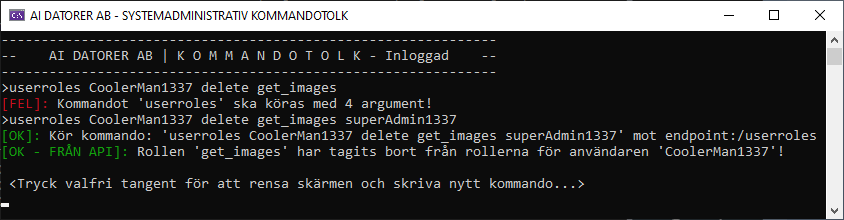
Figur : Användare i MongoDB



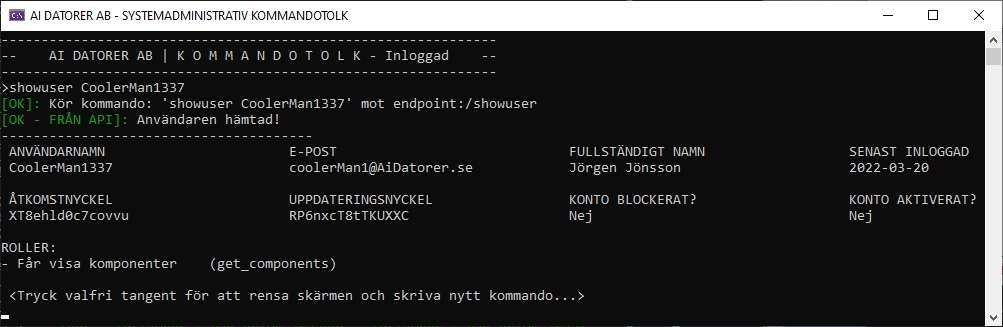
Figur : Samma användare i Konsolapplikationen

I Figur 15 & Figur 16 ovanför här så visas först en användare som sedan visas med hjälp av kommandot showuser username. Se nu även "ROLLER:".

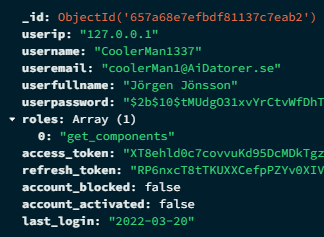
Under "ROLLER:" i *Figur 16* på föregående sida framgår det att användaren endast har tillstånd att visa bilder och visa komponenter och inom deras parenteser är då vad som ska anges för att kunna radera en av dessa roller/behörigheter.



Figur : Radera en roll/behörighet från samma användare



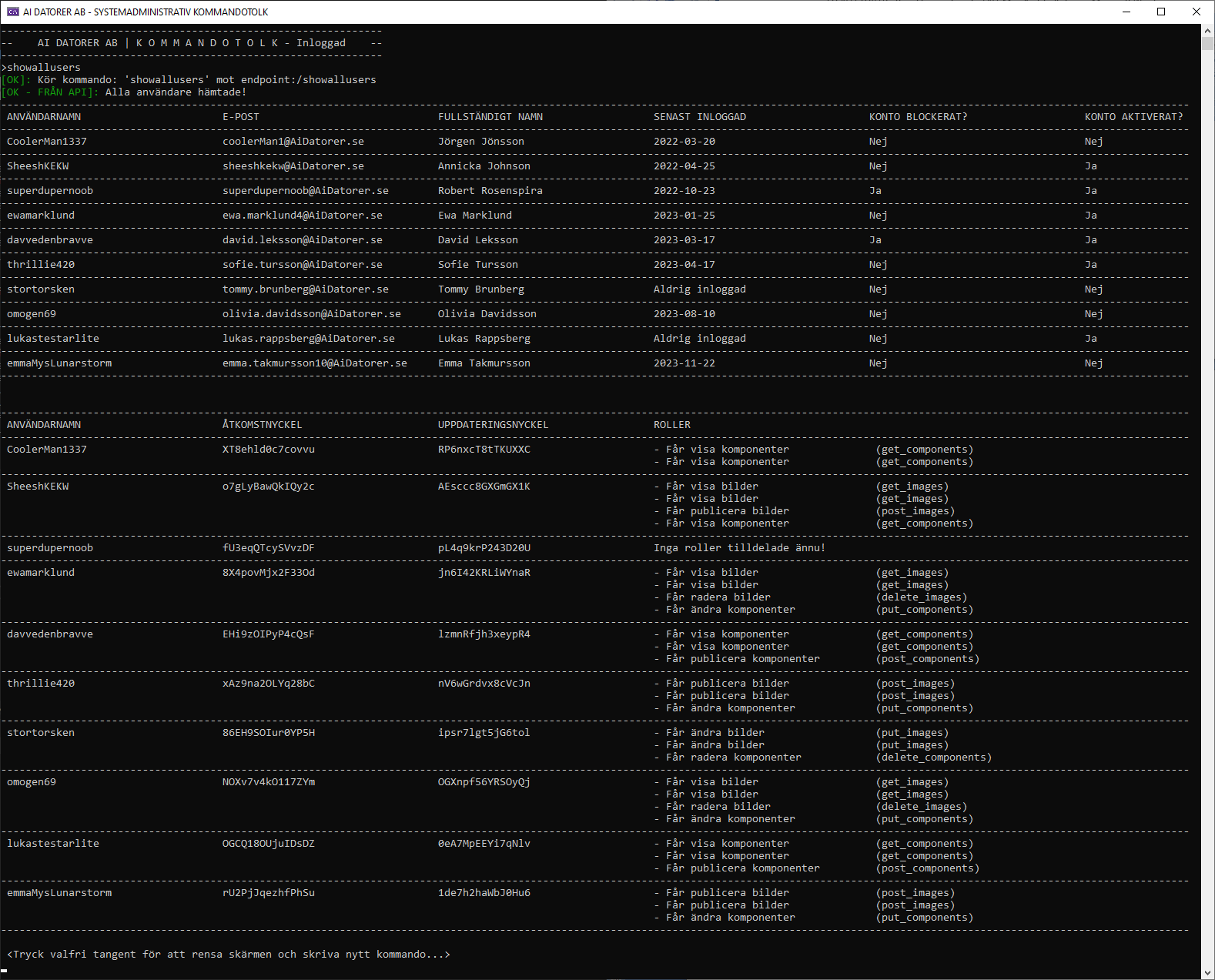
Figur : Samma användare efter borttagen roll/behörighet



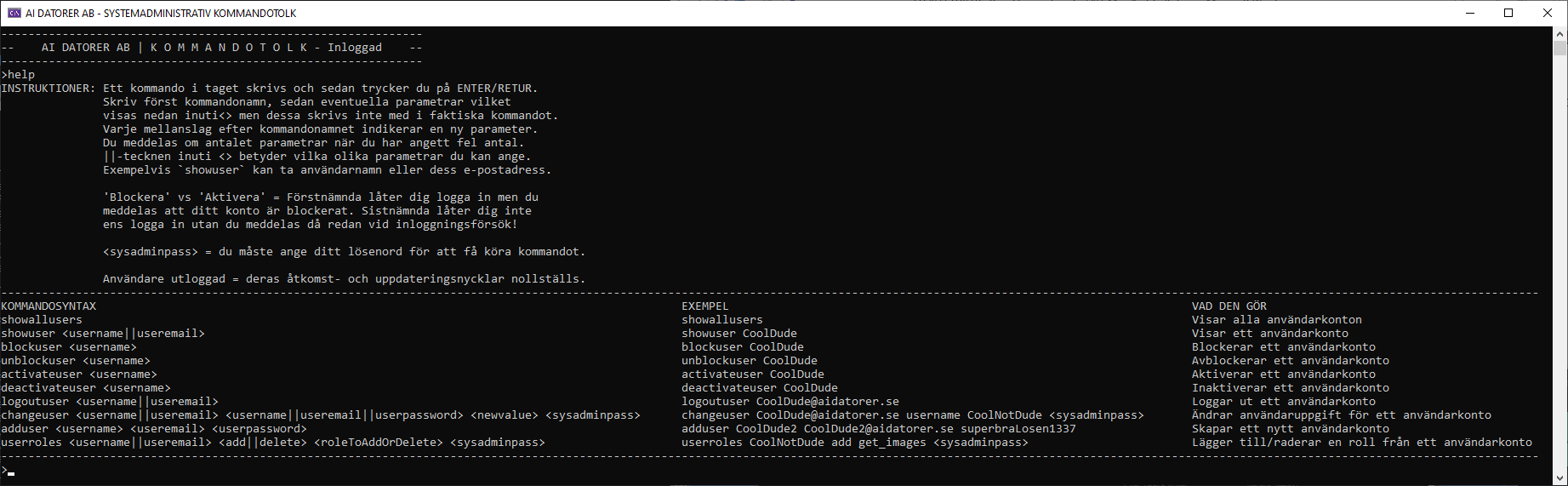
Figur : Samma användare i MongoDB efter raderad roll/behörighet

I *Figur 17, Figur 18* & *Figur 19* på föregående sida visas först hur en behörighet för användaren CoolerMan1337 försöks tas bort med hjälp av kommandot userroles CoolerMan1337 delete get\_images men det misslyckas för att systemadministratörens lösenord måste anges då av säkerhetsskäl. Detta fångas upp av valideringsklassen utan problem så kommandot skickas aldrig vidare för att kunna skickas till REST API:t.

Det korrekta kommandot userroles CoolerMan1337 delete get\_images superAdmin1337 skickas dock vidare och svar från REST API:t är att det gick att radera rollen/behörigheten get\_images från användaren CoolerMan1337. Detta återspeglas sedan i Konsolapplikationen (se *Figur 18* på föregående sida) och i MongoDB efteråt (se *Figur 19* på föregående sida).



Figur : Alla användare från MongoDB

I *Figur 20* på föregående sida visas alla användare efter att kommandot showallusers körts. Notera att i slutet på varje kommando så uppmanas valfri tangent att tryckas på för att komma tillbaka med rensad skärm.

Figur : Lista över alla kommandon

Avslutningsvis i *Figur 21* ovan här så visas alla tillgängliga kommandon när kommandot help skrivs vilket underlättar för att se vad som kan skrivas i Konsolapplikationen. Exakt samma omfattande kommandodokumentation återfinns i [GitHub repot för Projektet](https://github.com/WebbkodsLarlingen/dt071g-maka2207-projekt).

Detta avslutar detta kapitel och nu återstår endast slutsatserna för projektet och sedan tillkommer vedertagen källförteckning och noll bilagor.

# Slutsatser

a.

Källförteckning

1. Price, Mark J. *C# 11 and .NET 7 - Modern Cross-Platform Development Fundamentals - Start building websites and services with ASP.NET Core 7, Blazor, and EF Core 7 (Seventh Edition)*. UK: Packt Publishing Ltd. 2022
2. Wikipedia, "JSON Web Token", <https://en.wikipedia.org/wiki/JSON_Web_Token> Ändrad senast: 2023-12-01 Hämtad: 2023-12-05
3. Wikipedia, ".NET", <https://en.wikipedia.org/wiki/.NET> Ändrad senast: 2023-11-21 Hämtad: 2023-12-05
4. Wikipedia, ".NET Framework", <https://en.wikipedia.org/wiki/.NET_Framework> Ändrad senast: 2023-11-22 Hämtad: 2023-12-05
5. Wikipedia, "Xamarin", [https://en.wikipedia.org/wiki/Xamarin#.NET\_MAUI](https://en.wikipedia.org/wiki/Xamarin%23.NET_MAUI) Ändrad senast: 2023-11-25 Hämtad: 2023-12-05
6. Wikipedia, "C Sharp (programming language)", <https://en.wikipedia.org/wiki/C_Sharp_(programming_language)> Ändrad senast: 2023-12-12 Hämtad: 2023-12-14
7. Wikipedia, "C-sharp", <https://sv.wikipedia.org/wiki/C-sharp> Ändrad senast: 2022-11-12 Hämtad: 2023-12-14
8. Wikipedia, "C Sharp syntax", <https://en.wikipedia.org/wiki/C_Sharp_syntax> Ändrad senast: 2023-12-08 Hämtad: 2023-12-14
9. Wikipedia, "Command-line interface", <https://en.wikipedia.org/wiki/Command-line_interface> Ändrad senast: 2023-12-09 Hämtad: 2023-12-14
10. Wikipedia, "REST", <https://en.wikipedia.org/wiki/REST> Ändrad senast: 2023-12-13 Hämtad: 2023-12-14
11. Wikipedia, "API", <https://en.wikipedia.org/wiki/API> Ändrad senast: 2023-12-11 Hämtad: 2023-12-14
12. Wikipedia, "MongoDB", <https://en.wikipedia.org/wiki/MongoDB> Ändrad senast: 2023-12-06 Hämtad: 2023-12-14
13. B. Dayley. *Node.js, MongoDB and Angular Web Development (Second Edition)*. USA: Pearson Education Inc. 2018