Undersökning av Workflow Core och Elasticsearch för förbättrad hantering av arbetsflöden

LUNDS UNIVERSITET
Campus Helsingborg

LTH Ingenjörshögskolan vid Campus Helsingborg
Institutionen för datavetenskap

Examensarbete:
Najdat Jarkas
Muhamad Tatari

© Copyright Najdat Jarkas, Muhamad Tatari

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Lunds universitet
Lund 2023

Sammanfattning

Under den senaste tiden har det skett en betydande ökning av intresse inom forskningen och utvecklingen av arbetsflödesmotorer som är tekniska verktyg för att styra en sekvens av steg och processer. Detta har bidragit till bättre automatisering av komplexa och tidskrävande arbetsprocesser. I en del av de verksamhetssystem som Advania levererar finns det ofta ett behov av att kunna sätta upp och konfigurera olika typer av långvariga processer. Detta examensarbete undersöker Workflow Core och dess användning tillsammans med MongoDB och Elasticsearch i samarbete med Advania.

Workflow Core är en arbetsflödesmotor byggd på öppen källkod som erbjuder många kraftfulla funktioner. I det presenterade arbetet har en responsiv webbapplikation utvecklats för att underlätta användningen av Workflow Core. Webbapplikationen möjliggör visuellt skapande av ett arbetsflöde som sedan kan exporteras i JSON-format och direkt utnyttjas av ett Workflow Core-program. Dessutom möjliggör webbapplikationen spårning av framsteg för olika arbetsflödesinstanser, vilket ger bättre överblick över exekveringsprocessen.

Den andra delen av examensarbetet innebar att en undersökning gjordes kring anslutningen av Workflow Core-programet till MongoDB och Elasticsearch-servern. När uppdatering eller modifiering av Workflow Core-program och datasbasstruktur sker kan det uppstå indexeringsfel i Elasticsearch-servern. Workflow Core har ett tillägg som heter Elasticsearch-plugin som sköter indexeringen och sökningen i Elasticsearch. Felen inträffar enligt testerna som utfördes inom ramen för examensarbetet när Elasticsearch-servern är nere i flera minuter samtidigt som en arbetsflöde-datauppdatering sker. Det betyder att dataintegritetsfel uppstår mellan Elasticsearch-server och MongoDB då de inte är synkroniserade. I examensarbetet presenteras olika metoder för att synkronisera MongoDB med Elasticsearch. Att skapa en fungerande synkronisering ligger utanför examensarbetets ram.

Nyckelord:

Workflow Core, MongoDB, Elasticsearch, React, React-Flow, Elsa Workflow, Elasticsearch-indexering, Arbetsflödesmotor, NoSQL, arbetsflödesdesignverktyg, Webbapplikation, Arbetsflödesspårning, Workflow Core JSON-fil Exportering,

Abstract

In recent times, there has been a significant increase in interest regarding the research and development of workflow engines, which are technical tools to control a sequence of steps and processes. This has enhanced the automation process of complex and time-consuming work processes. In some of the operational systems that Advania delivers, there is often a need to set up and configure different types of long-term processes. This thesis explores Workflow Core and its use together with MongoDB and Elasticsearch, in collaboration with Advania.

Workflow Core is a powerful workflow engine built on open source code. A responsive web application has been developed in this project in order to simplify the use of Workflow Core. This web application enables a visual creation of a workflow, which can be exported in JSON format and directly utilized by a Workflow Core program. In addition, the web application enables progress tracking for different workflow instances, which provides a better overview of the execution process.

The second part of this thesis involves investigating the integration of the Workflow Core application with MongoDB and Elasticsearch servers. Workflow Core offers an extension called Elasticsearch plugin that handles the indexing and searching in Elasticsearch. Some indexing errors may appear in the Elasticsearch server when updating or modifying the workflow or database structure. According to the tests performed in this thesis, the errors occur when the Elasticsearch server experiences downtime for several minutes while a data update is sent from the workflow engine. This leads to data integrity problems between Elasticsearch server and MongoDB as they become out of sync. This thesis presents different methods to synchronize MongoDB with Elasticsearch. Creating or achieving a working synchronization is outside the scope of this thesis.

Key words:

Workflow Core, MongoDB, Elasticsearch, React, React-Flow, Elsa Workflow, Elasticsearch Indexing, Workflow Engine, NoSQL, Workflow Design tool, Web Application, Workflow Tracking. Workflow Core JSON File Export.

Förord

Med stor glädje och tacksamhet presenterar vi detta examensarbete som markerar slutet på vår trevliga resa inom datateknik på högskoleingenjörsutbildningen vid Lunds tekniska högskola.

Vi vill uttrycka vår tacksamhet till Advania för möjligheten att genomföra examensarbetet hos dem och särskilt tacka Henry Kjönsberg för det stöd han har gett oss under samarbetet. Dessutom vill vi tacka vår handledare, Roger Henriksson, och vår examinator, Christin Lindholm, för all hjälp och fantastisk vägledning under resan. Tack till våra vänner och familjer för all kärlek och stöttning!

Nu har vi avslutat en framgångsrik fas och är redo att stå inför en ny och spännande utmaning.

Innehållsförteckning

1. Inledning

Det här examensarbetet utförs i samarbete med Advania och handlar om att undersöka Workflow Core och hitta en lösning som möjliggör enkel användning av funktioner som Workflow Core erbjuder.

1.1 Bakgrund

Advania är ett företag som levererar IT-lösningar, som till exempel molntjänster, till andra företag, offentliga verksamheter och myndigheter [1]. Företaget finns i Sverige och fem andra länder från norra Europa. Genom långsiktiga kundrelationer samt ledande teknologier hjälper Advania sina kunder att växa med IT.

När det gäller exekvering av komplexa uppgifter kan det vara fördelaktigt att betrakta dem som arbetsflöden. På så sätt kan uppgifterna delas upp i mindre delar och hanteras i en bestämd ordning. För att hantera arbetsflöden är det bra att ha i åtanke att det finns kraftfulla licensfria arbetsflödesmotorer som är tekniska verktyg för att underlätta processen. Trots det är det utmanande att hitta en arbetsflödesmotor som passar ens behov på bästa sätt.

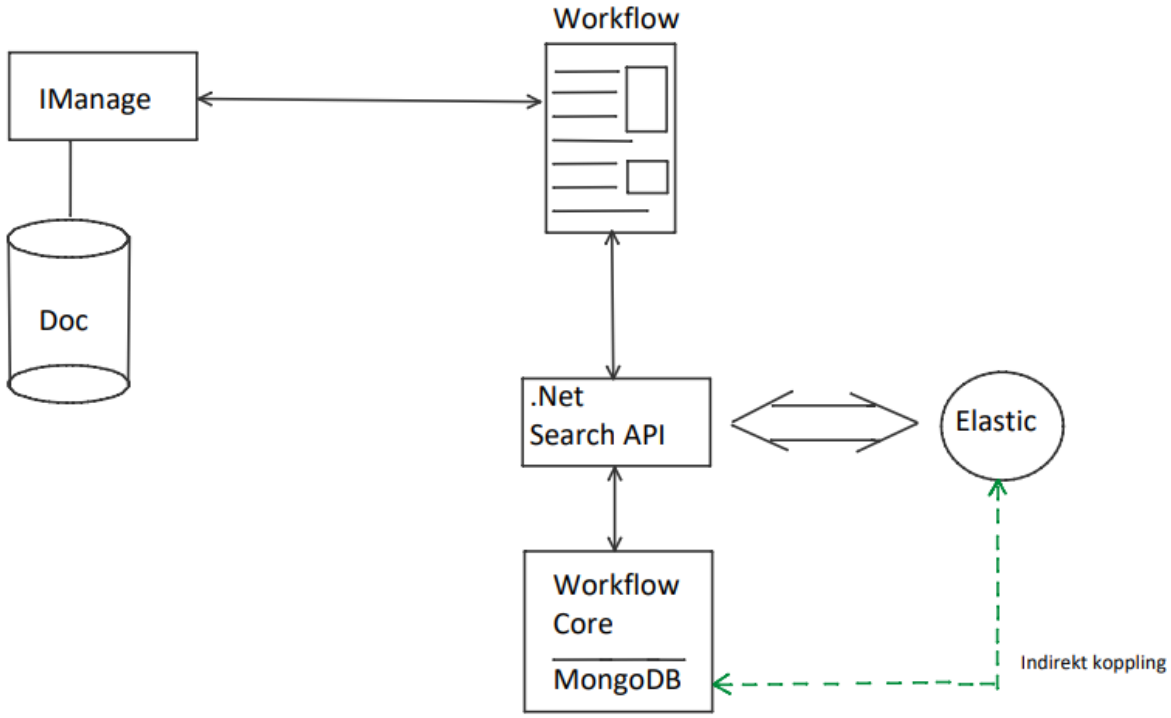
Workflow Core är enligt [2]-[3] en lätt arbetsflödemotor byggd på öppen källkod. Tanken med den är att den ska användas för att hålla ordning och spåra framsteg i en långvarig process. För övrigt riktar Workflow Core sig mot .NET 5 standarder och har MIT-licens som tillåter bland annat användning, kopiering och modifiering. Ett konkret exempel på hur Workflow Core kan användas i praktiken är att automatisera en rekryteringsprocess för ett rekryteringsföretag. Genom att skapa ett arbetsflöde för jobbansökningar och definiera varje steg i processen tillsammans med dess beroenden kan arbetssökande skicka in sina ansökningar via företagets webbplats. Arbetsflödet tilldelar en unik ärendekod till varje jobbansökan och låter rekryteraren granska ansökningen och uppdatera dess status i varje steg. Genom att använda Workflow Core kan arbetssökande få automatiska statusuppdateringar via e-post eller annan kommunikationskanal tills ärendet är avslutat.

Advania har använt Workflow Core, men modifierat och begränsat det utifrån behov. För närvarande har de skapat ett arbetsflöde som hanterar en specifik dokumenthanteringsprocess. Deras lösning är således specialbyggd och hårdkodad för just det här fallet. Nu vill Advania undersöka Workflow Core djupare, gärna hitta ett sätt att få mer kontroll över arbetsflöden och skapa en generaliserad lösning för att bygga olika arbetsflöden. Detta kan underlätta och förbättra deras verksamhet i framtiden. Därför presenteras i denna rapport en utveckling av sättet att använda Workflow Core samt hur ett gränssnitt skapats för att göra hanteringen av arbetsflöden lättare.

Dessutom har Workflow Core en Elasticsearch-plugin som gör det möjligt att indexera arbetflöden i Elasticsearch. Elasticsearch är en distribuerad sökningsserver som låter användare lagra, söka och analysera enorma mängder data [4]. Genom Elasticsearch-plugin sparas arbetflödesdata i ett ElasticSearch-index, vilket är en samling av dokument av typen JSON. Detta i sin tur möjliggör för användaren att söka upp arbetsflödes tillstånd och data på ett effektivt sätt.

Workflow Core kan använda sig av Elasticsearch som sökningsserver och MongoDB som lagringsleverantör. MongoDB är en icke-relationsdatabas (NoSQL databas) som lagrar datan i form av dokument [5]. Ett exempel på ett MongoDB dokument kan vara en JSON-fil.

I samarbete med Advania undersöktes hur Elasticsearch indexering skulle kunna hanteras i drift tillsammans med MongoDB. Detta gjordes eftersom det uppstår indexeringsfel vid driftstörningar på grund av till exempel mjukvaruuppdateringar, ändringar i MongoDB:s databasstruktur, nya workflow-release eller andra okända orsaker (se figur 1). Därför är det viktigt att övervaka indexeringen och ha åtgärdsplaner på plats för att hantera problemen.



Figur 1: Översikt över systemet och den indirekta (streckade) kopplingen som kan tappas och orsaka indexeringsfel.

1.2 Syfte

Syftet med examensarbetet var att utveckla sättet att använda Workflow Core för att bättre möta Advanias behov inom hantering av arbetsflöden. Dessutom undersökte projektet hur Elasticsearch-indexering kan hanteras i drift tillsammans med MongoDB för att undvika och hantera indexeringsfel.

1.3 Problemformulering

Examensarbetet gick ut på att undersöka och besvara följande frågor:

1. Vad är Workflow Core och varför ansåg Advania att deras användning av den är begränsad?
2. Vilka fördelar och nackdelar har Workflow Core jämfört med andra arbetsflödesmotorer baserade på öppen källkod, om sådana finns?
3. Varför valde Advania just Workflow Core?
4. Hur kan Workflow Core vidareutvecklas och implementeras för att utnyttja fler funktioner och möjligheter?
5. Vilka typer av fel kan uppstå vid anslutningen mellan Elasticsearch och MongoDB? Kan de hanteras och hur?

1.4 Motivering av examensarbetet

Detta examensarbete har som målsättning att presentera en utveckling och förbättring av ett redan existerande system. Advania påpekar att deras användning av Workflow Core är begränsad, men samtidigt har en stor potential för utveckling och förbättring. De föreslår att om användning av Workflow Core förbättras och uppdateras så skulle det bli mer användbart och öppna möjligheter för mer kontroll över arbetsflöden. En annan brist är att användning av Elasticsearch-plugin tillsammans med MongoDB drabbas av indexeringsfel på grund av olika anledningar. Därför är det viktigt att undersöka metoder för att hantera indexeringen i drift. Att hitta en lösning som förenklar användning av Workflow Core-funktioner och att kunna hantera indexeringen i drift skulle ge inspiration till programmerare och även till andra företag som använder eller tänker använda Workflow Core.

Examensarbetet är dessutom valt med tanke på hur intressant och lärorikt arbetet kommer vara för oss. Genom det här examensarbetet öppnas möjligheten för oss att jobba med varierande uppgifter som är relaterade till vår utbildning och våra erfarenheter. Företaget som vi samarbetar med (Advania) erbjuder en arbetsplats och relevanta resurser som behövs under examensarbetet. Att vi har möjlighet att jobba nära Advania ger oss stor överblick om hur arbetslivet ser ut. Dessutom har Advania möjligheten att erbjuda vidareanställning efter examen vid nöje av båda parter.

1.5 Avgränsningar

- Workflow Core är utvecklat för att följa standarderna i .NET 5.
- Projektet genomfördes med åtkomst till en MongoDB-server med egengenererade testdata och en lokal Elasticsearch-server.
- Om Workflow Core: Arbetsflödetsmotor studerades enligt Advanias önskemål och täcker därmed inte alla områden.

1.6 Arbetsfördelning

Författarna tillbringade större delen av arbetet sida vid sida och samarbetade aktivt. Najdat hade mer fokus på designbeslut, medan Muhamad fokuserade mer på kodning.

	Najdat Jarkas	Muhamad Tatari
Analys	50	50
Intervjuer	50	50
Design	60	40
Implementation/kodning	30	70
Testning/utvärdering	50	50
Rapport och presentation	50	50
Poster	60	40

2. Metod

Examensarbetet består av frågor som besvaras med hjälp av teoretiska och praktiska moment. Det teoretiska gav fördjupad förståelse för problemen samt var ett underlag för lösningar genom intervjuer och litteraturstudier/dokumentstudier. Medan det praktiska handlade om att koda, testa och att ta fram möjliga lösningar.

Först genomfördes litteraturstudier och påbörjades intervjuerna med representanter från Advania gällande Workflow Core och andra arbetsflödesmotorer. Efter att kraftfulla funktioner och begränsningar hos Workflow Core hade identifierats, påbörjades även utvecklingen av prototyper. Varje prototyp visades upp för Advania under en intervju för att säkerställa att arbetet var på rätt väg. Innan en slutlig prototyp visades upp för Advania genomfördes scenariotester för kvalitetssäkring. Efter arbetet med intervjuerna och prototyperna återupptogs litteraturstudier och tester genomfördes gällande Workflow Core och arbetflödesindexering i Elasticsearch. Till sist genomfördes försök att synkronisera en Elasticsearch-server med en MongoDB-databas.

2.1 Litteraturstudier

Genom att genomföra litteraturstudier kunde examensarbetarna studera vad Workflow Core är samt identifiera andra tillgängliga licensfria alternativ. Därutöver gav dessa studier en mer ingående förståelse för vad MongoDB och ElasticSearch är och hur de kan vara relaterade till Workflow Core. För att hitta litteraturen letade vi i “IEEE Xplore”, “ACM Digital Library”, “Google Scholar” och “Google” databaser. Google användes för att hitta ett antal officiella dokumentationer för bland annat Workflow Core-biblioteket. De officiella dokumentationer är skrivna av verktygets utvecklare och förklarar i detalj vad som kan göras med deras verktyg och hur de kan implementeras. I de andra databaserna sökte vi litteratur genom att använda listor och kombinationer av följande nyckelord:

- Workflow Core
- Open-source
- ElasticSearch
- MongoDB
- Non-relational
- Workflow
- Engine
- Scheduling
- Saga-transaction
- .NET
- Integrated
- Workflow-builder
- Elsa

- BPM
- NoSQL
- SQL
- Workflow
- Database
- Engine

Ett konkret exempel på en sökning är (Workflow) AND (Open-source) AND (engine)

2.2 Intervjuer

Syftet med intervjuerna var att få en övergripande uppfattning om vad representanter från Advania generellt tycker om olika prototyper som presenterades under arbetet och vilka förbättringar skulle behövas. Det genomfördes totalt tre intervjuer, varvid alla viktiga detaljer och insikter noggrant noterades och dokumenterades i Google Docs. Intervjutekniken som användes var en ostrukturerad intervju, där möten var informella och inleddes med öppna frågor [6].

Intervjuerna utfördes på Advanias kontor med examensarbetarnas handledare och utvecklingsteam som består av tre personer från Advania. Den första intervjun handlade bland annat om varför Advania har använt sig av Workflow Core. Vad de tyckte var viktigt gällande sättet att använda och utveckla Workflow Core. Den andra och tredje intervjun inleddes med frågor om prototyperna som till exempel:

- Vad tycker Advania generellt om lösningen som prototypen visar?
- Vilka krav saknas?
- Hur kan prototypen förbättras mer?

2.3 Prototyper

Efter litteraturstudierna och instudering av Workflow Core-dokumentation valde examensarbetarna att börja utveckla prototyper genom att skapa ett användargränssnitt i två iterationer. Tanken bakom användargränssnittet var att underlätta arbetet med Workflow Core genom att bygga nya arbetsflöden eller modifiera ett befintligt arbetsflöde på ett visuellt sätt, vilket gör att programmerare slipper skriva en del kod. I användargränssnittet bör en användare kunna bygga ett visuellt arbetsflöde och sedan exportera det till Workflow Core i JSON-format. Den exporterade JSON-filen kan i sin tur registreras som ett arbetsflöde i Workflow Core. För att få en grundläggande förståelse för hur den här möjligheten ser ut, började arbetet med att se vilka licensfria bibliotek som kan vara till nytta i det här arbetet. Därefter skapades två prototyper som byggde vidare på varandra och som också visades upp för Advanias representanter.

2.4 Testning

För att leverera ett system/prototyp med hög kvalitet är det särskilt viktigt att testa systemet och säkerställa att dess funktioner fungerar som de ska. I det här arbetet, och under utvecklingen av prototyperna, utfördes ett antal tester kontinuerligt. Kritiska funktioner som är avgörande för systemets huvudsyfte, som till exempel JSON-fil-exporteringen, prioriterades över icke-kritiska funktioner. Dessutom designades också ett antal tester som utfördes och dokumenterades innan en slutlig prototyp levererades till Advania. Den primära testmetoden som användes var manuell testning utförd av utvecklarna i samarbete med användarna. Med manuell testning menas att prototypens funktionalitet granskades av testarna själva utan hjälp av tekniska automationsverktyg. Sist men inte minst genomfördes också varierande tester av icke-funktionella krav för de olika versionerna av prototypen som presenteras i det examensarbete.

2.4.1 Testning av icke-funktionella krav

Testning av icke-funktionella krav syftar till att kontrollera om det finns något fel i designen av prototypen(webbapplikationen). Dessutom granskas tillgängligheten i olika webbläsare och enheter samt svarstiden för knappar och omladdning av webbsidan för att säkerställa korta svarstider (mindre än en sekund). Testprotokollet som användes var en checklista för att underlätta testningen och verifiera att webbapplikationen håller rätt kvalitet. (Checklistan presenteras först i avsnitt 4.3.1)

2.4.2 Funktionell testning

För att kunna testa funktionalitet så användes scenariotestning. Den är en metod som används för att kunna täcka ett antal tester av olika funktioner samtidigt [7]. Vid scenariotestning spelar utvecklarna rollen som slutanvändare och tar reda på de verkliga scenarion och användningsfallen som kan finnas. Ett exempel på ett scenario är användarregistrering. I detta scenario spelar testarna rollen som användare och verifierar att de kan skapa ett nytt konto. Samtidigt testas det också att det inte är möjligt att skapa ett konto med ogiltiga eller duplicerade uppgifter. (Scenarierna presenteras först i avsnitt 4.3.2)

2.4.3 Testning gällande Elasticsearch indexering

För att utföra tester gällande Elasticsearch indexering användes ett arbetsflöde som är byggt av examensarbetarna under examensarbetet. Koden modifierades så att den använde sig av testupplagor av MongoDB och Elasticsearch som kördes på en lokal dator. Följande tester i tabell 1 syftade till att köra arbetsflödet flera gånger för att utforska och analysera hur indexering och datan sparas i Elasticsearch-servern vid arbetsflödesexekvering. I varje test exekverades arbetsflödet flera gånger med olika in-parametrar för att undvika upprepning av samma process. Dessutom simulerades typiska driftstörningar, som att stänga av Elasticsearch-servern eller att ändra på datastrukturen för ett arbetsflöde för att undersöka om Elasticsearch-indexering skulle påverkas om olika datastrukturer sparades i MongoDB.

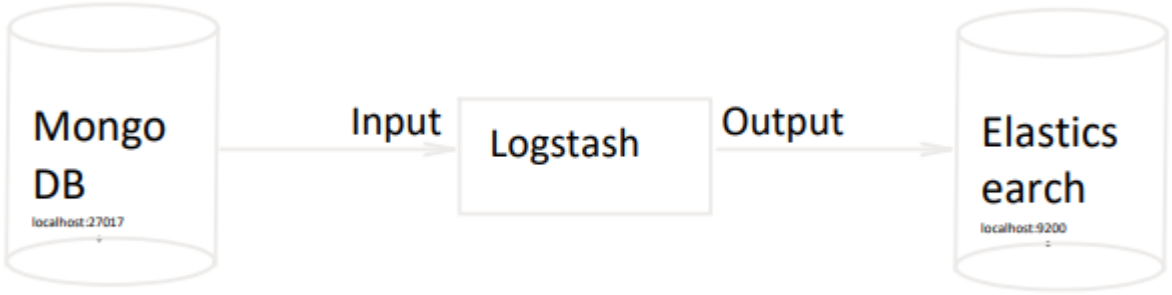
Tabell 1: Tester gällande Elasticsearch indexering.

Test	Beskrivning
1	Testet kördes under ideala omständigheter, utan att utsätta exekveringen för någon driftstörning.
2	Testet kördes under villkoret att Elasticsearch-servern startades om vid olika tidpunkter, både innan exekveringen påbörjades och under exekveringen av olika steg.
3	Testet kördes under villkoret att elasticsearch-servern stängdes ner för en längre period, både innan exekveringen påbörjades och under exekveringen av olika steg.
4	Testet kördes under ideala omständigheter (med primär och lokal databas istället för mongoDB), utan att utsätta exekveringen för någon driftstörning.

Dessutom undersöktes kopplingen mellan MongoDB och Elasticsearch-sökningsservern genom att stänga ner och starta om båda tjänsterna, antingen samtidigt eller någon av dem i taget, upprepade gånger. Slutligen förändrades datan som var sparad i MongoDB för att se om förändringarna återspeglades i Elasticsearch-servern eller inte.

2.4.4 Försök att synkronisera MongoDB och Elasticsearch

Det genomfördes totalt fyra försök att synkronisera datan från MongoDB till Elasticsearch, vilket skulle hjälpa vid eventuell omindexering av Elasticsearch. Försöken inleddes med ett initieringsstadium för att se om Elasticsearch-pluginen skulle lyckas skapa indexer i Elasticsearch-servern. När initieringsstadiet misslyckades avbröts försöket omedelbart. Som verktyg användes Logstash som är en modul (en s.k. processing engine) som kan användas för att ta input från MongoDB och indexerar datan till Elasticsearch-servern (se figur 2)**.** I alla försök användes MongoDB version 6.0.5, medan versionen för det inbyggda Workflow Core-biblioteket för Elasticsearch var 3.8.3. Versionerna för Logstash och Elasticsearch varierades däremot och valdes på så sätt att de var kompatibla med varandra. Dessa versioner presenteras i tabell 2.



Figur 2: En förenklad modell av hur data kan synkroniseras i realtid mellan MongoDB och ElasticSearch.

Tabell 2: Elasticsearch och Logstash versionerna som användes i försöken.

Miljö	Elasticsearch version	Logstash version
1	8.7.1	8.7.1
2	7.0.0	7.0.0
3	6.8.23	6.8.23
4	5.5.0	5.5.0

2.5 Källkritik

Källorna som användes i detta examensarbete kan klassificeras i tre kategorier. Till den första kategorin hör officiella webbplatser och GitHub-dokumentationer där utvecklare, ägare eller deras representanter står bakom källorna och är ansvariga för innehållet. Ett exempel på en sådan källa är [2] som presenterar en produkt, ger beskrivningar och förklarar hur vissa funktioner kan användas.

Den andra kategorin innefattar konferensartiklar som publicerades efter 2010. Dessa artiklar nämner tydligt vem författarna är och har referenser som stödjer deras innehåll. Dessutom vänder sig dessa källor till forskare eller tekniska experter. Ett exempel på en sådan källa är [8].

Den tredje kategorin omfattar övriga källor som antingen hänvisar till officiella webbplatser och GitHub-dokumentationer eller har författare som är experter inom sina områden. Ett exempel på en sådan källa är [9] som listar olika arbetsflödesmotorer tillsammans med korta förklaringar och hänvisningar till officiella GitHub-dokumentationer eller webbplatser.

För att säkerställa trovärdigheten jämfördes även informationen från källorna med examensarbetarnas förkunskaper och andra källor.

3. Teoretisk bakgrund

För att ge en överblick över arbetet introduceras först allmänt arbetsflödesmotorer. Därefter presenteras Workflow Core och lite om Elsa. Efter det sammanfattas vad React Flow-biblioteket är allt eftersom den används som huvudkomponent för användargränssnittet i prototyperna. Till sist tas MongoDB och Elasticsearch upp eftersom MongoDB används både i prototyperna och tillsammans med Elasticsearch i det här arbetet.

3.1 Allmänt om arbetsflödesmotorer

Under senaste tiden har mycket intresse och utveckling kring arbetsflödesmotorer ägt rum, påpekar [8]. Det beskrivs att en arbetsflödesmotor är en typ av mjukvara som används inom programvaruutveckling både för att hantera och exekvera processer. Den ger möjlighet att automatisera processerna och att följa upp dem från start till slut. Dessutom nämns det att ett arbetsflöde kan ha användarinteraktion såväl som datorinteraktion. Livscykeln för ett arbetsflöde består av bland annat tre viktiga faser. Den första fasen är att bygga och strukturera arbetsflödet med dess data. Den andra fasen är att studera och bestämma vilka resurser varje aktivitet i arbetsflödet kommer att behöva vid exekvering. Den tredje är själva exekveringen av arbetsflödet.

Det finns flera tillgängliga licensfria arbetsflödesmotorer och system som hanterar arbetsflöden på nätet. Till exempel presenteras en lista av sådana system och arbetsflödesmotorer på [9]. Enligt listan är Workflow Core ett exempel på bibliotek som används inbäddat och Elsa Workflow ett exempel på en fullt utvecklad produkt. Dessutom tar [10] upp att det finns något som kallas för vetenskapliga arbetsflödesmotorer. Det konstateras att det finns ett antal skillnader mellan vetenskapliga och icke vetenskapliga arbetsflödesmotorer, och några av dem är att:

- Vetenskapliga arbetsflöden använder arbetsflöden för att utföra vetenskapliga experiment. Det kräver att arbetsflödesmotorn och systemet är tillräckligt enkla för att göra det möjligt att analysera och hantera datan.
- Vetenskapliga arbetsflöden kan ha extrafunktioner för till exempel dataanalys.

Några licensfria och vetenskapliga arbetsflödesmotorer som Kepler och Triana presenteras [10]. Kepler är en användarvänlig arbetsflödesmotor och som sparar arbetsflöden i form av xml och Triana som har stöd för bland annat signal, text och bilbehandling.

Det här arbetet behandlar icke-vetenskapliga arbetsflödesmotorer och fokuserar på Workflow Core men tar även upp lite om Elsa Workflow som är enligt [9] ett .Net-bibliotek liksom Workflow Core .

3.2 Workflow Core

Enligt [2] definieras strukturen på arbetsflöden som en kedja av steg som är sammankopplade. För att få kontroll över arbetsflöden och dess steg finns det stöd med inbyggda kontrollstrukturer och Saga-transaktioner (se 3.2.5 och 3.2.7). Workflow Core stöder, för övrigt, både JSON och YAML-formateringar för att definiera och bygga arbetsflöden. Externa händelser och aktiviteter hanteras via en särskild programmodul. Denna brukar kallas för en värd (host). Dessutom kan ett Workflow Core-program utökas med mellanprogramvara (Middleware), som till exempel kan användas för att spara loggar efter varje steg.

I [2] nämns att Workflow Core kan kopplas ihop med många olika databaser som till exempel MongoDB. Dessutom kan en plugin för Elasticsearch användas för indexering och sökning. Workflow Core stöder flernodskluster-körning och kan dessutom ha tillägg för att hantera användarinteraktion med arbetsflöden. För kvalitetssäkring av arbetsflöden har Workflow Core felhanteringssystem och inbyggt stöd för testning av arbetsflöden. Mer om Workflow Core beskrivs enligt [2] i avsnitt 3.2.1-3.2.11.

3.2.1 Steg

Ett steg i Workflow Core kan implementeras och specificeras i programmeringsspråket C# genom arv med hjälp av abstrakta klasser. Stegen kan dessutom definieras enkelt med lambdauttryck istället för att skapa en stegklass, om så önskas. I och med att ett arbetsflöde består av flera sammankopplade steg och processer, kan strukturen i ett arbetsflöde definieras genom att referera till ett redan skapat steg (klass) via C# kod, JSON- eller YAML-format. Sist men inte minst kan ett steg ta indata (inputs) och returnera utdata (outputs) från respektive till övriga steg.

3.2.2 Värd

I en miljö med flera registrerade arbetsflöden och dess instanser, är värden ansvarig för att köra de arbetsflöden som är redo att köras. Den ansvarar dessutom för att lägga till händelser till arbetsflöden som väntar på att en händelse ska inträffa.

3.2.3 Externa händelser (Event)

En extern händelse i Workflow Core kommer från en extern källa och implementeras liknande ett vanligt steg i ett arbetsflöde. Ett exempel på en extern händelse är när ett digitalt dokument skapas och vissa uppgifter fylls i. Därefter inträffar en extern händelse där en annan part behöver granska och signera dokumentet. När en extern händelse i Workflow Core förväntas inträffa, kommer arbetsflödet inte att fortsätta processen innan händelsen är triggad. En händelse triggas externt och arbetsflödet väntar på det så att arbetsflödesexekvering kan fortsätta. Det är möjligt att sätta villkor för att avbryta händelsen så att arbtetsflödesmotor slutar vänta på händelsen. Ett annat val är att ange ett brytdatum (EffectiveDate) som styr huruvida ett arbetsflöde ska svara på en extern händelse eller inte.

3.2.4 Aktivitet

En aktivitet i Workflow Core behandlas likt en extern händelse. Ett arbetsflöde kan vänta på att en aktivitet, som är en extern kö av uppgifter, ska köras klart innan arbetsflödet fortsätter att exekvera sina steg. Aktiviteten kan ha ett avbrottsvillkor och ett brytdatum på samma sätt som en extern händelse.

3.2.5 Kontrollstrukturer

Ett arbetsflöde och dess kedjor av steg manipuleras av flera olika kontrollstrukturer. Det kan skapas flera grenar, och arbetsflödesmotorn bestämmer vilken eller vilka grenar som arbetsflödet ska fortsätta med, beroende på om ett visst villkor är uppfyllt. Olika Workflow Core-inbyggda satser såsom while-sats, if-sats och forEach-sats kan användas för att styra arbetsflödet på önskat sätt. Det bör observeras att forEach-satser exekveras parallellt. Förutom det kan en schemaläggningsfunktion användas för att starta exekvering av steg asynkront i bakgrunden. Dessutom finns det en fördröjningsfunktion för att pausa exekveringen under en viss tid. Slutligen kan en upprepningsfunktion (Recur) användas för att några specifika steg ska köras repetitivt i bakgrunden tills ett visst villkor är uppfyllt.

3.2.6 Felhantering

Felhanteringen konfigureras antingen på stegnivå eller global nivå. Åtgärderna vid fel är antingen att försöka igen (retry), skjuta upp (suspend), avbryta (terminate) eller ersätta (compensate). Det bör dock observeras att Workflow Core också har felhanteringfunktioner för bland annat exekvering av Saga-transaktioner och mellanprogramvara. Lite om felhantering för dessa funktioner tas upp senare i avsnitt 3.2.7 “Saga-transaktion” och 3.2.8 “Mellanprogramvara (Middleware)”.

3.2.7 Saga-transaktion

En saga är en mall som används för att hantera flera steg och transaktioner. I en saga utförs varje steg för sig själv som en transaktion. Om ett steg misslyckas så hanteras det av Saga som initierar kompensationsåtgärder för att eventuellt ångra alla tidigare steg i Saga. En användare kan skicka in parametrar (inputs) för att ange vilken kompensationsåtgärd som ska vidtas om det behövs.

3.2.8 Mellanprogramvara (Middleware)

Ett arbetsflöde i Workflow Core kan utökas med mellanprogramvara. En mellanprogramvara kan köras innan exekvering av varje arbetsflöde eller efter att varje arbetsflöde har exekverat klart. Dessutom kan en mellanprogramvara implementeras på stegnivå inom ett visst arbetsflöde. Genom att använda ett eller flera mellanprogramvaror kan beteendet av arbetsflöden och dess steg ändras. Mellanprogramvara körs i den ordning de blir registrerade i, vilket bör observeras vid implementering och registrering av mellanprogramvara.

Å ena sidan, i den officiella dokumentationen för Workflow Core, tas det inte upp hur felhantering på stegnivå kan gå till. Å andra sidan lyfts det upp att felhantering för mellanprogramvara som körs innan exekveringen av ett arbetsflöde hanteras på ett annat sätt än mellanprogramvara som körs efter ett arbetsflöde. Fel i mellanprogramvara som körs innan exekvering av arbetsflöden hanteras av en metod som kallar på ett arbetsflöde och som får arbetsflödet att börja köra. Fel i mellanprogramvara som körs efter exekvering av ett arbetsflöde har däremot alltid en inbyggd hanteringsteknik som registrerar loggar och låter exekvering av arbetsflöden opåverkat.

3.2.9 Lagringsleverantör (Persistence Provider)

I och med att arbetsflöden kan vara långvariga processer så behöver dess data sparas i ett persistent minne eller databas. Med det menas att innehållet i minnet eller databasen ska finnas kvar även efter att servern har startats om eller varit avstängd. Ett sådant här minne eller databas brukar kallas för en lagringsleverantör. I fall en lagringsleverantör inte har specificerat i Workflow Core-programmet så sparas arbetsflöden lokalt i en så kallat minneslagringsleverantör. Workflow Core har stöd för användning av flera databaser som MongoDB, SQL Server, PostgreSQL, SQLite, Amazon DynamoDB, Cosmos DB och Redis. I det här examensarbetet används MongoDB för att lagra data av Workflow Core.

3.2.10 Elasticsearch-plugin

Elasticsearch kan användas som sökningsserver för Workflow Core. Således ger den möjlighet att indexera ett flertal arbetsflöden och söka efter dess tillstånd och data. Elasticsearch-pluginen matchar användarens textsökning med de textfält som sparas i Elasticsearch-servern och som definieras av arbetsflödesprogrammet. Textfälten som Elasticsearch-pluginen söker i är arbetsflödesreferenser, beskrivningar, status och definitioner. Utöver det kan man söka bland egna skapade dataobjekt om de implementerar ISearchable. Sökfunktionen hos Elasticsearch-pluginen ger möjlighet för användaren att ta kontroll över sökresultatet. Bland annat kan man begränsa antalet sökresultat som returneras, hoppa över ett antal sökresultat och till och med applicera en lista av filter på sökningen. ScalarFilter, DateRangeFilter, NumericRangeFilter och StatusFilter är typer av filter som kan appliceras på sökning.

3.2.11 Övrigt i Workflow Core

Workflow Core stöder för övrigt flernodskluster, men som standard körs den på en enda nod. För att köra med flernodskluster behöver ett antal saker, som till exempel köhanterare, konfigureras. Dessutom har Workflow Core stödmetoder för testautomation av arbetsflöden skapade med Workflow Core. Med ramverken xUnit eller nUnit kan dessa stödmetoder vara till nytta så att en användare kan simulera en arbetsflödesexekvering. En användare kan till exempel mata in initialdata med de stödmetoderna och automatiskt kontrollera om utdatan är som förväntat eller inte. Sist men inte minst är det även möjligt att ha användarinteraktion med arbetsflöden i Workflow Core genom att installera ett tilläggspaket. Ett enkelt exempel på hur ett sådant tillägg kan användas är att låta användare godkänna eller avbryta en process.

3.3 Elsa Workflow

Elsa Workflow är en arbetsflödesmotor baserad på öppen källkod med MIT licensen och .Net standarder [11]. I Elsa finns det enligt [12] flexibilitet när det gäller att bygga arbetsflöden, eftersom olika sätt är tillgängliga för användning. Det första är att arbetsflöden kan byggas med kod skriven i C#. Den andra är att bygga det med hjälp av dataformat i JSON-syntax. Det tredje är genom att använda något som kallas “Workflow Designer” som gör det enkelt att designa ett arbetsflöde visuellt och ger möjlighet att återanvända HTML5 komponenter som kan bäddas in i egna webbsidor. De designade arbetsflöden kan senare exporteras i JSON-format som senare utnyttjas för att bygga arbetsflöden.

Dessutom nämns det i [13] att Elsa har stöd för ett virtuellt Dashboard som gör hanteringen och spårning av arbetsflöden enkelt. Det finns även versionshanteringsstöd tillsammans med det virtuella Dashboard:et. När en ny version av ett arbetsflöde publiceras, ökar dess versionsnummer automatiskt och nya instanser av arbetsflöden ska därmed fungera enligt uppdateringen. Förutom det, är Elsas arbetsflöden inbyggda så att äldre instanser av ett arbetsflöde som har uppdaterats inte automatiskt påverkas i sin tur av en uppdatering. Elsas officiella dokumentation nämner att den kan användas för både långvariga- och kortvariga processer. Med en kortvarig process, till skillnad från en långvariga process, menas att ett

arbetsflöde körs från början till slutet på en gång. Arbetsflödesmotorn kan hantera komplexa arbetsflöden som även inkluderar användarinteraktioner. Arbetetsflödesmotorn kan kopplas ihop med många databaser som bland annat InMemory (Lokal databas), MongoDB, YesSQL och Entity Framework Core. När det gäller utvärdering av uttryck vid körning har Elsa inbyggda utvärderingssätt som bland annat använder Javascript och Liquid. Dessutom kan Elsa köras i flernodskluster och erbjuder redundans om någon nod går ner. Sist men inte minst, har Elsa mycket utökningsmöjligheter, vilket presenteras i Elsas dokumentation som en av de viktigaste funktionerna. En användare kan utöka inbyggda arbetsflödes aktiviteter med egna aktiviteter. Sådana utökningar ska även vara synliga i Workflow Designer. Elsa kan också utökas för att stödja andra lagringsleverantörer än till exempel MongoDB, men då behöver det implementeras på egen hand. Inbyggda utvärderingssätt kan även utökas till att använda uttryck från andra programmeringsspråk såsom Python. Inför framtiden planeras enligt [11] ytterligare funktioner att läggas till. Till exempel planeras möjligheten att testa och debugga arbetsflöden från “Workflow Designer” att läggas till. Dessutom planeras det också att införa sökning och indexeringsstöd med Lucene.

3.4 React-flow

React-flow är ett biblioteket som är licensierat under MIT fri licens [14]. Därmed betraktas React-flow som en React-komponent. Komponenten ger möjlighet för programmerare att på ett enkelt sätt bygga visuella noder, flöden och interaktiva diagram och anpassa dem till det individuella behovet. React-flow kommer med inbyggda tillägg som bland annat minimap- och kontrollkomponenter som visualiserar en liten version av till exempel ett arbetsflöde respektive hjälper användaren att orientera noderna och kontrollera vyport.

3.5 MongoDB

Enligt [15] har den kontinuerliga utvecklingen inom olika forskningsområden och sektorer, exempelvis industrin, lett till en ökning av stora datamängder. Att hantera dessa stora datamängder utgjorde en utmaning och ledde till utvecklingen av NoSQL-databaser som MongoDB för att ge hög skalbarhet, tillgänglighet och prestanda för moderna applikationer. MongoDB är dokumentorienterad och använder ett flexibelt schema som tillåter lagring av strukturerad, semistrukturerad och ostrukturerad data. Utvecklarna kan enkelt lägga till eller ta bort fält från dokument utan att behöva ändra i hela schemat. Det påpekas i [16] att MongoDB lagrar data i ett JSON-liknande dokumentformat som kallas BSON, vilket står för Binary JSON.

3.5.1 Jämförelse mellan SQL och NoSQL Databaser

SQL-databaser använder SQL som språk för att hantera och manipulera data [16]. SQL-databaser är relationsdatabaser som lagrar data i tabeller med rader och kolumner och använder ett schema för att definiera datastrukturen. Den är designad för att hantera strukturerad data, med fasta datatyper och strikta relationer mellan tabeller. Enligt [15] är NoSQL-databaser, som tidigare nämnts, icke-relationella databaser som lagrar data i flexibla format som dokument, nyckel-värdepar och grafer. NoSQL-databaser är designade för att hantera ostrukturerad, semistrukturerad och snabbt föränderliga data, vilket gör dem väl lämpade för hantering av stora data. NoSQL-databaser använder inte ett schema för att definiera datastruktur, vilket möjliggör mer flexibel datamodellering. NoSQL-databaser använder en mängd olika frågespråk och API:er för att komma åt och manipulera data. Dessutom kan de utformas för att skalas horisontellt över flera servrar.

3.6 ElasticSearch

ElasticSearch är en dokumentorienterad, icke-relationsdatabas med öppen källkod som är avsedd för fulltextsökning och dataanalys i nästan realtid. Enligt [17] har den blivit ett populärt verktyg som används av ingenjörsteam runt om i världen de senaste åren på grund av dess egenskaper. Den erbjuder skalbarhet och enkla REST API:er att använda [18].

Enligt [18], är Elasticsearch byggt på Apache Lucene och släpptes först 2010 av Elasticsearch N.V. (nu känt som Elastic). Elasticserach kan användas för många olika fall som till exempel app-sökning, webbplatssökning, företagssökning, logganalys, affärsanalys och andra användningsområden också.

3.6.1 Sökning i ElasticSearch

Enligt [19] så använder Elasticsearch ett frågespråk (Query DSL) som förlitar sig på JSON-objekt. Sökningar i ElasticSearch har två sammanhang för query-satser, query-kontext och filterkontext. [20] beskriver att query-kontexten beräknar en relevanspoäng och svarar på hur väl ett dokument matchar en query-sats, medan filterkontexten svarar ja eller nej på om ett dokument matchar en query-sats, utan att beräkna en poäng. En filterkontext används vanligtvis för att filtrera strukturerad data och är snabbare tack vare att filterna som används oftast i Elasticsearch blir automatiskt placerade i cacheminnet, vilket gör att prestandan ökar.

3.6.2 Indexering och omindexering i Elasticsearch

Källa [18] beskriver att ElasticSearch-index är en samling av dokument som är relaterade till varandra. Dessa dokument lagras i JSON-format och innehåller en uppsättning nycklar som representerar fält eller egenskaper med sina tillhörande värden. Elasticsearch använder en datastruktur som kallas inverterat index som möjliggör effektiva fulltextsökningar. Den listar varje unikt ord som finns i ett dokument och identifierar alla dokument som innehåller varje ord. Vid indexering lagras dokumenten och ett inverterat index byggs upp för att göra dokumentdatan sökbar i nästan realtid. Indexering sker genom Elasticsearch Index API, som gör det möjligt att lägga till eller uppdatera ett JSON-dokument i ett specifikt index.

I Elasticsearch förekommer begreppen kluster (cluster), nod (node), skärva (shard) och replik (replica). Enligt [21] är Elasticsearch en distribuerad plattform som alltid är tillgänglig och skalbar som gör det möjligt att lägga till fler noder i klustret för att öka kapaciteten. Elasticsearch fördelar data och sökbelastning automatiskt över alla noder, så att applikationen kan växa utan behov av att göra ändringar i koden.Ett Elasticsearch-index består av en eller flera fysiska skärvor som är logiskt grupperade. Genom att fördela dokumenten i ett index över flera skärvor och noder, säkerställer Elasticsearch hög tillgänglighet. Vid behov flyttar Elasticsearch automatiskt skärvor över befintliga noder för att balansera belastningen.

Det finns två typer av skärvor som används, primära skärvor och repliker. Varje dokument i ett index tillhör en primär skärva. Replikskärvor är kopior av primära skärvor som används för redundans och ökad sökkapacitet vid läsoperationer, t.ex. för att söka eller hämta ett dokument. [22] påpekar att för att använda nya fält och dokumenttyper i Elasticsearch-index efter en uppdatering av schemat måste indexet omindexeras. Detta görs genom att låsa indexet för att förhindra skrivningar medan data kopieras, skapa ett nytt index med det uppdaterade schemat, kopiera data från det gamla indexet till det nya, jämföra gamla och nya data för att kontrollera eventuella inkonsekvenser, och byta alias för att peka på det nya indexet.

Efter omindexeringen kan de nya fälten och dokumenttyperna användas för sökningar och analyser. Det är viktigt att planera för omindexeringen eftersom det kan ta tid beroende på storleken på indexet och mängden data som behöver kopieras. I detta arbete genomfördes en undersökning om hur omindexering kan ske.

3.6.3 Elasticsearch och MongoDB

Elasticsearch och MongoDB är båda databaser och har några likheter men ändå väljer många användare som [19] att använda MongoDB eller andra databaser som primär datalagring och för att upprätthålla dataintegritet, medan de använder ElasticSearch för att hantera läsning och sökning över data. Det vill säga att de använder en kombination av till exempel Mongodb och ElasticSearch.

Enligt [19] så används MongoDB mest i kombinationen med ElasticSearch eftersom det är lätt att använda, flexibel och stöder lås och kan enkelt säkerhetskopieras. Några experiment utfördes för att testa kombinationen mellan MongoDB och Elasticsearch, där MongoDB är huvuddatabas och ElasticSearch är “sekundär” databas för sökningen över data. Resultatet visar både positiva och negativa aspekter. För det första så är kombinationen lätt att implementera och använda. För det andra är båda systemen väldokumenterade. Slutligen har kombinationen bra dataintegritet och hög läshastighet. Däremot så var högre minnesanvändning och längre skrivtider bland det negativa.

Det finns flera sätt att synkronisera eller transportera data från MongoDB till Elasticsearch-servern, och några av dem presenteras i [23] respektive [24]. För det första kan synkroniseringen ske via verktyget Logstash. Logstash är ett licensfritt verktyg från Elastic och ELK-stack. Logstash kan ta input-data från en källa, som MongoDB, och sedan indexera den datan i Elasticsearch-servern. Dessutom kan filter användas, om så önskas, för att hantera input-datan innan den skickas för indexering i Elasticsearch-servern. Logstash senaste version 8.7 kräver enligt [25] en Java (JVM), antingen Java 17 eller Java 11.

För det andra kan även synkronisering ske via Mongo-Connector. Det är ett verktyg från MongoDB som är byggd i Python. Det kan kopiera data från MongoDB till bland annat Elasticsearch-servrar. Verktyget kräver bland annat att MongoDB körs med "replica-set"-inställning, för att den ska kunna skriva data till Elasticsearch-server och synkronisera datan tillsammans med MongoDB CRUD operationer. För det tredje kan en sådan synkronisering ske på liknande sätt via Elasticsearch-River-MongoDB som också kräver att MongoDB körs med “replica-set”-inställning. Enligt [26] är den senaste versionen av Elasticsearch-River-MongoDB 2.0.11 kompatibel med MongoDB version 3.0.0 och ElasticSearch 1.7.3. Slutligen kan transportering också ske via verktyget Transporter. Verktyget är licensfritt och utvecklat av Compose som erbjuder en molnplattform för databaser.

4. Resultat

Först presenteras en sammanfattning av vad Workflow Core är och en jämförelse mellan Workflow Core och andra arbetsflödesmotorer. Därefter presenteras de viktigaste punkterna från de intervjuer som genomfördes i det här arbetet i syfte att förstå varför Advania valde Workflow Core och varför de upplevde att deras användning av det var begränsad. Direkt efter detta presenteras de båda versionerna av en prototyp av en React-webbapplikation. Sedan beskrivs utvärderings- och kvalitetssäkringsarbete som utfördes på den slutliga prototypen. Till sist presenteras resultaten från testerna kring Elasticsearch-indexering och försöken att synkronisera en Elasticsearch-server med MongoDB.

4.1 Workflow Core

Workflow Core är arbetsflödesmotor som kan användas för att definiera, hantera och köra arbetsflöden i .Net applikationer. Workflow Core biblioteket är särskilt bra och kraftfullt för att hantera långvariga och komplexa arbetsflöden som kan innehålla flera steg och beroende. En av de mest tilltalande aspekterna av Workflow Core är dess flexibilitet och förmåga att integreras med olika system som MongoDB och Elasticsearch i det här arbetet. Därför är det möjligt att skapa skräddarsydda arbetsflöden som passar de specifika behoven för en applikation eller organisation. Dessutom erbjuder denna arbetsflödesmotor en mängd olika funktioner som gör det enkelt att definiera och hantera önskade arbetsflöden på ett smidigt sätt.

4.1.1 Workflow Core och andra arbetsflödesmotorer

Vid jämförande mellan Workflow Core och andra arbetsflödemotorer baserade på öppen källkod är det viktigt att tänka på att det finns en uppsjö med arbetsflödesmotorer ute på nätet, som har samma kärnfunktionalitet som Workflow Core. För att avgöra vilka fördelar och nackdelar som finns med Workflow Core jämfört med andra motorer, bör man ta hänsyn till de individuella behoven och inte bara sammanföra funktioner som erbjuds av varje arbetsflödesmotor. Trots det kan det nämnas lite fördelar och nackdelar med Workflow Core jämfört med Elsa och även med vetenskapliga arbetsflödesmotorer.

För ett av Advanias projekt, till exempel, kan man nämna att tillgången till en Elasticsearch-plugin utgör en fördel jämfört med Elsa. Dessutom erbjuder Workflow Core stöd för testautomation och skapande av tester med xUnit eller NUnit, vilket inte nämns i Elsas officiella dokumentation om funktioner. Workflow Core stöder byggande av arbetsflöde med Yaml-format vilket inte erbjuds av Elsa. Å andra sidan kan Elsa anses ha fler möjligheter för utökning av olika funktioner. Dessutom kan Elsa vara fördelaktigt om man behöver ett dashboard eller en visuell möjlighet att skapa och hantera arbetsflöden. Elsa har också möjlighet till versionhantering på ett visuellt sätt.

Forskare inom olika områden kan föredra att använda vetenskapliga arbetsflöden istället för att använda Workflow Core, då vetenskapliga arbetsflödemotorer kan erbjuda enkelhet och

extrafunktioner som dataanalys. Workflow Core är mer inriktat mot personer med programmeringskunskaper, särskilt inom C#-programmeringsspråket, vilket kan vara fördelaktigt för många programmeringsprojekt. Men för forskare och personer utanför programmeringsbranschen kan detta innebära en nackdel eftersom det kräver en viss teknisk kompetens för att kunna använda verktyget. Varje arbetsflödesmotor har sina fördelar och nackdelar beroende på användaren och de specifika behoven för användningen. Därför är det viktigt att utvärdera noggrant vilken arbetsflödesmotor som passar bäst för projektet innan man bestämmer sig.

4.2 Intervjuhöjdpunkter och prototyputveckling

Det genomfördes sammanlagt tre intervjuer med representanter från Advania, och två prototyper visades upp för dem. Efter den första och andra intervjun utvecklades prototyp 1 respektive den slutliga prototypen. I den tredje intervjun presenterades den slutliga prototypen för att säkerställa att den uppfyllde sitt syfte och för att diskutera framtida arbete.

4.2.1 Intervju 1

För att få en överblick över Workflow Core och förstå vikten av dess användning hos Advania, gjordes en första intervju. Dessutom siktade intervjun på att svara på några frågor gällande varför just Workflow Core valts av Advania, varför deras användning av den anses begränsad och vilka deras förväntningar var gällande examensarbetet.

Representanterna från Advania påpekade att det ofta finns behov av att kunna sätta upp och konfigurera olika typer av processer och arbetsflöden i en del av de verksamhetssystem som Advania levererar. Dessa arbetsflöden kan ta lång tid att exekvera och behöver ibland vänta på olika typer av signeringar. Advania hade undersökt marknaden för tillgängliga workflow-tjänster som har vissa krav och som passar deras kunders behov. Några av de grundläggande krav som sattes i urvalet var att:

- “Workflow” tjänsten inte fick vara molnbaserad utan den skulle kunna hanteras helt lokalt i egen miljö.
- Att Advania ska ha tillgång till källkod för att säkerställa funktionalitet och de tekniska förutsättningarna från ett säkerhetsperspektiv.
- Att den “Workflow” tjänsten ska vara en egen funktion och inte ingå i ett större ramverk som inte tillför något.

Advania ansåg att deras användning av Workflow Core var begränsad och att de inte kunde skapa en komplett plattform på grund av resursbegränsning och tidsbrist inom deras projekt. Därför utvecklade de endast den absolut nödvändiga funktionalitet och vidareutvecklade inte funktioner med Workflow Core som plattform. Företagets syfte med detta examensarbete är att skulle kunna förbättra Advanias möjligheter att vidareutveckla arbetsflödestjänster.

4.2.2 Intervju 2

En design om hur den första prototypen skulle se ut och dess funktioner presenterades till utvecklingsteamet i Advania(se 4.2.4 Första prototyp).

Den primära tanken och prototypen gav ett bra intryck på handledaren och utvecklingsteamet i Advania. De ville gärna att prototypen skulle kopplas mot en MongoDB-databas. De önskade att en undersökning skulle genomföras angående möjligheten att binda arbetsflödesinstanser till arbetsflödesmallar och att spåra varje instans och hur långt i ett flöde ett visst ärende har kommit.

Vad gäller designen så tyckte Advania att navigationsmenyn behövde modifieras och att den borde ha mindre antal knappar. Till exempel skulle de funktioner som finns i prototypen kunna samlas i en dropdown-meny. Sist men inte minst ansåg Advania att en vidareutvecklad produkt, baserad på den primära idén, är något de absolut kan testa i framtiden. Slutligen påpekades det att det kan föreligga vissa begränsningar, som till exempel tidsramen för examensarbetet. Således kunde webbapplikationen omfatta grundläggande funktionalitet och inte allt som önskades eller inkluderades av Workflow Core.

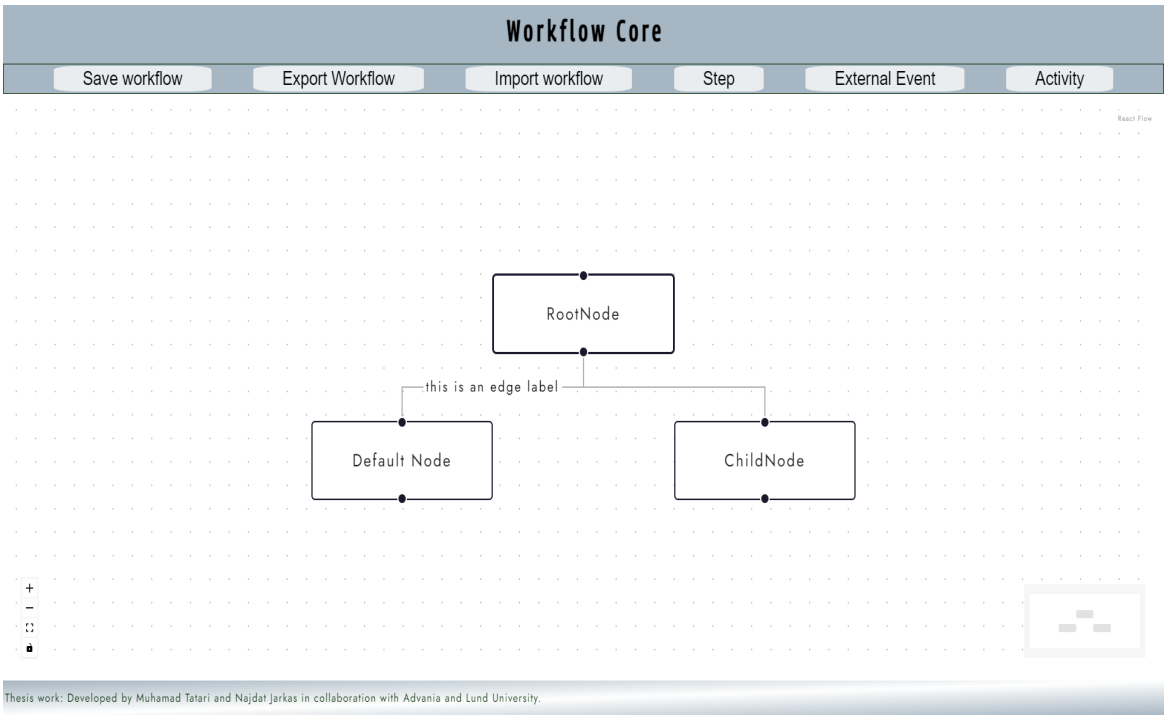
4.2.3 Intervju 3

Den slutgiltiga prototypen presenterades för Advanias representanter på plats via en lokal dator (se 4.2.5 Den slutliga prototypen). Prototypen fick positiv feedback och representanterna var nöjda med resultatet och tyckte att den hade fyllt sitt syfte. Efter presentationen uppstod dessutom, en diskussion gällande framtida möjligheter, där representanterna påpekade att vår huvudkomponent i prototypen, React-flow komponenten, möjligen skulle kunna användas och integreras med deras befintliga system. Det vill säga att prototypen inte bara skulle kunna användas som en fristående produkt, som den såg ut i det här arbetet, utan även som en grund till integrationslösning för att effektivisera nuvarande processer för spårning av arbetsflöden. Just nu kan arbetsflöden spåras av Advania genom att visa ett dokument, och det tycktes att det skulle vara bra att spåra arbetsflöden visuellt istället genom att integrera vår React-flow komponent med systemet och strukturen på det spårningsdokumentet som Advania använder.

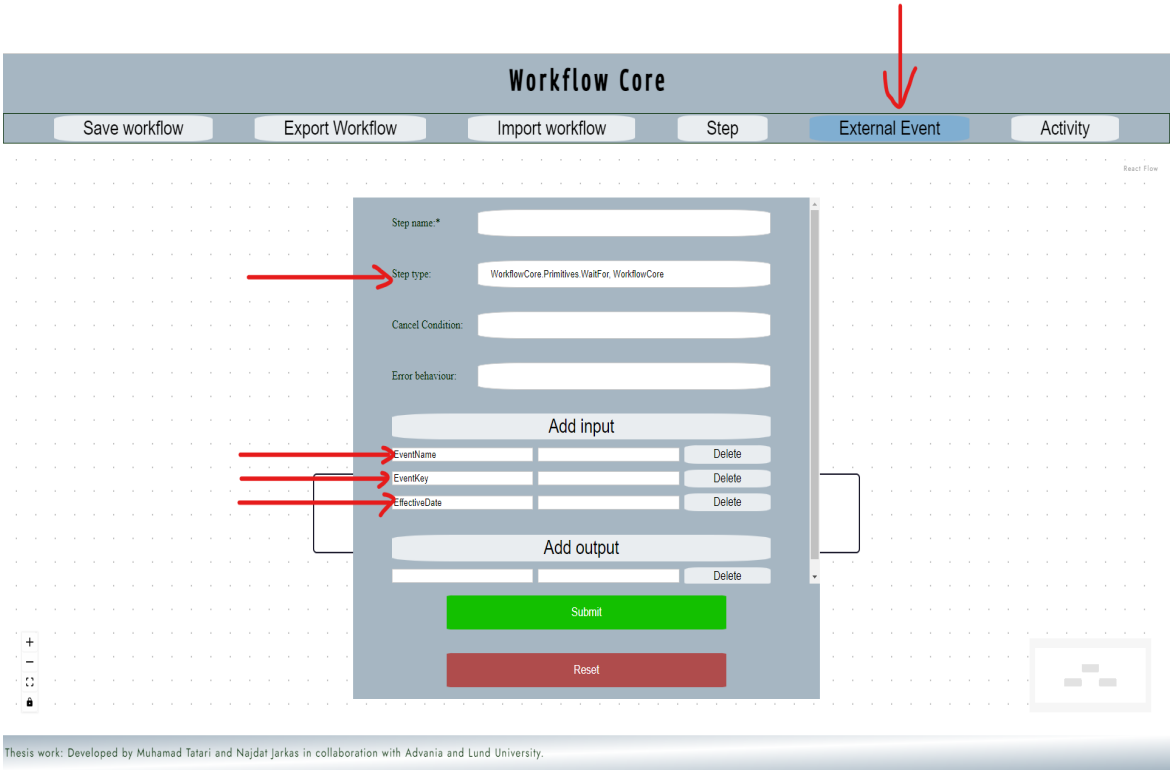
4.2.4 Första prototypen

I den första prototypen skapades en webbapplikation som nästan saknar all funktionalitet, men som kan demonstrera tanken (Se figur 3 och 4). React-ramverket, HTML, CSS, JSX och React Flow-komponenter användes för implementering. Målet med sidan är inte att ha så bra design som möjligt, utan att ge en prototyp som fyller Advanias önskemål och behov.

Prototypen visade att användaren kan spara, importera och även exportera arbetsflödet till ett Workflow Core-program genom en knapp i navigationsmenyn. Dessutom kan användaren skapa steg av olika typer. Prototypen av webbapplikationen kan också hjälpa till med att skapa stegen genom att automatiskt ge förslag på att fylla i fält som finns i ett formulär (se figur 4). Genom att använda React Flow-komponenten kan användaren tydligt se var noderna är placerade med hjälp av en minimap, då en minimap visar en miniatyrbild av noderna och deras placering i förhållande till det större området. En användare ska också kunna zooma in och ut, låsa noderna från att modifieras och automatisk anpassa vyn till noderna.



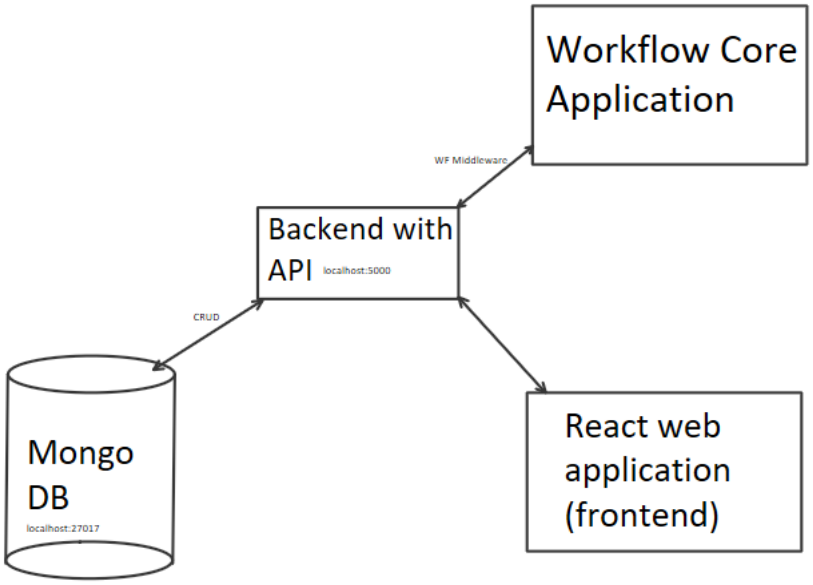
Figur 3: Ett arbetsflöde med React Flow-komponent och första prototypdesign.



Figur 4: Ett formulär som dyker upp när ett steg skapas. Formuläret är anpassat till (External Event). Pilarna som pekar på fälten visar förslag på vad som kan fyllas i.

4.2.5 Den slutliga prototypen

Utvecklingen av den andra prototypen var baserad på den feedback som har fåtts under intervju 2. En väldigt enkel MongoDB som körs på localhost:27107 har byggts. Databasen innehåller tre samlingar: Nodes, Edges och WFollowingData. Nodes består av ett antal noder och varje nod representerar ett steg i arbetsflödet tillsammans med dess data, som beskriver indata (inputs) och utdata (outputs). Edges består av ett antal kantlinjer där varje kantlinje kopplas mellan exakt två noder. Kantlinjerna kan ha olika typer och varje typ presenteras i webbapplikation med ett unikt utseende. Sist men inte minst är WFollowingData skapad för att undersöka möjligheten att spåra arbetsflöden i Workflow Core.



Figur 5: Översikt över hur webbapplikationen och Workflow Core-applikationen kommunicerar med databasen i prototyp 2.

I figur 5 visas hur olika system interagerar med varandra. Ett arbetsflöde byggt av Workflow Core kommunicerar med backend genom att en eller flera HTTP-webbförfrågningar skickas från Workflow Core "mellanprogramvara". Workflow Core-applikationen som använts i prototyp 2 har två olika mellanprogramvaror, där det ena körs mellan stegen och det andra körs efter att arbetsflöden har exekverats färdigt.

Mellanprogramvaran som körs på stegnivå extraherar data från den inbyggda Workflow Core IStepExecutionContext. Den extraherade datan pekar på vilken typ av arbetsflöde som körs, den unika identitet som en aktiv instans har och det steg som håller på att exekvera.

Därefter skickas datan till backend som i sin tur sparar data i MongoDB under samlingen WFollowingData. Mellanprogammet som körs efter att ett arbetsflöde har exekverat klart filtrerar bort icke relevanta typer av arbetsflöden och skickar en unik identitet på den instans som har kört klart till WFollowingData. Detta underlättar testningen och kommunikationen från och till React-webbapplikationen.

MongoDB CRUD har använts för att skapa API:er som Workflow Core applikationen respektive React-webbapplikation kan använda sig av. Nedan följer de API:er som är skapade för prototyp 2.

Get-operationerna för att hämta data för noder, kantlinjer och spårningsdata:

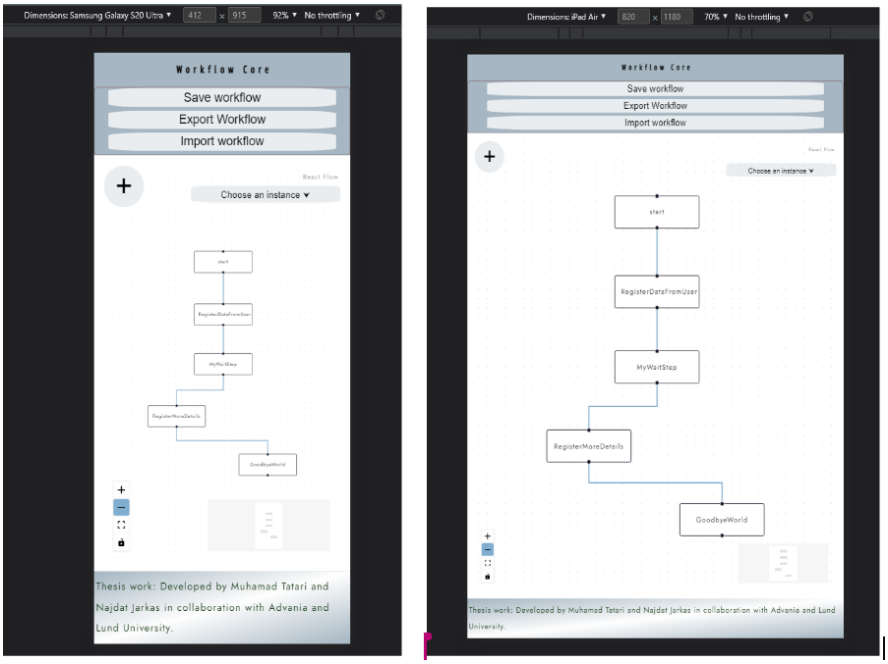
- http://localhost:5000/getNodes
- http://localhost:5000/getEdges
- http://localhost:5000/getFollowingData

Post-operationerna används för att skicka data till servern för att spara, uppdatera och ta bort noder, kantlinjer och spårningsdata:

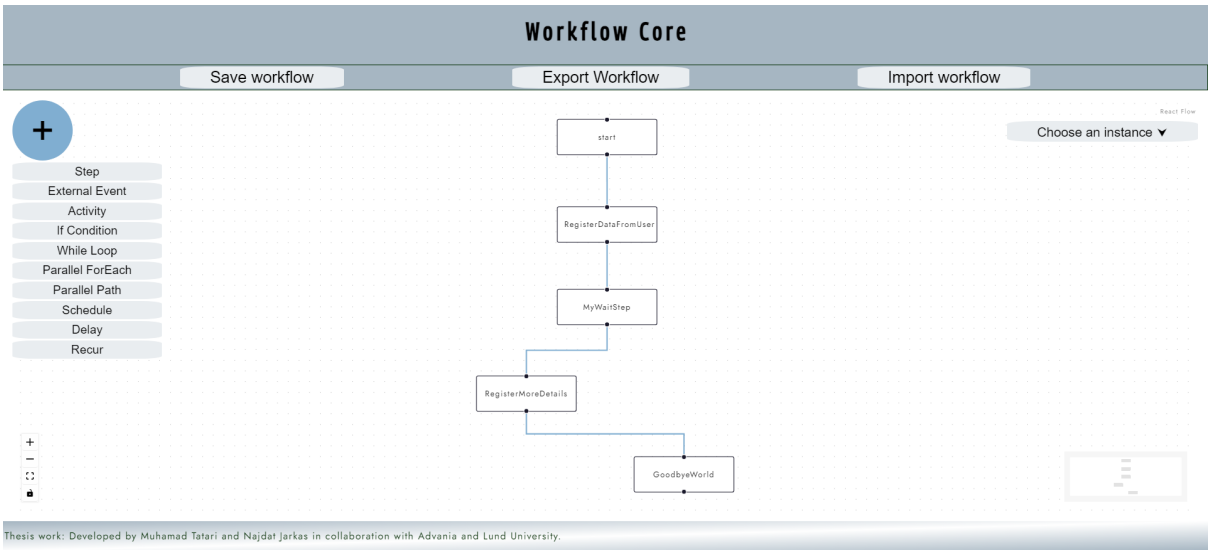
- http://localhost:5000/saveNodes
- http://localhost:5000/saveEdges
- http://localhost:5000/insertFollowingData
- http://localhost:5000/deleteFollowingData

4.2.5.1 Design och funktioner

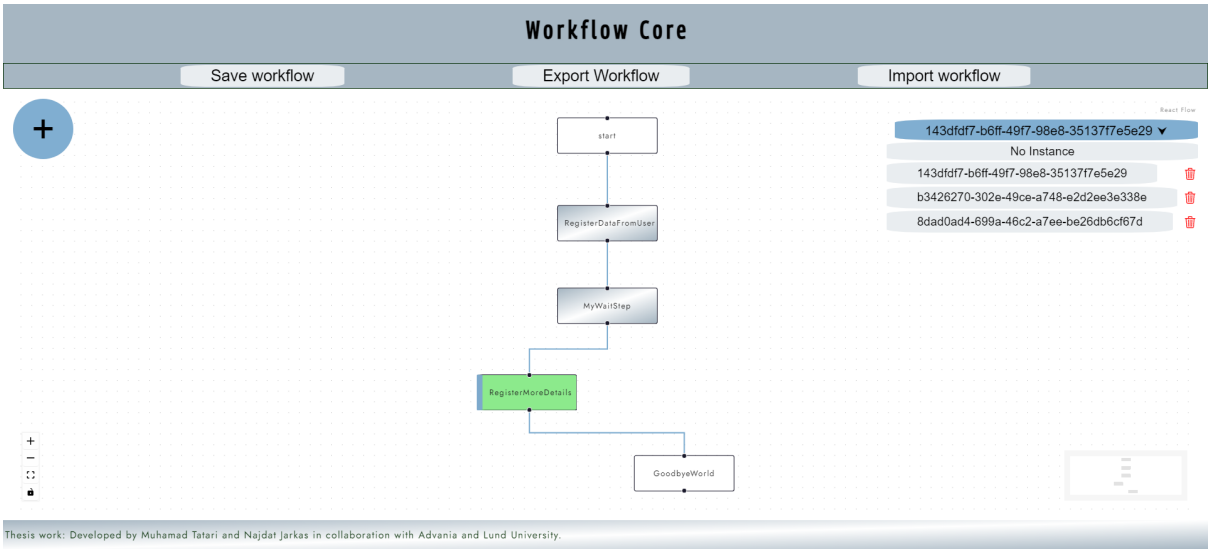
Prototyp 2 är nu responsiv och kan användas på olika enheter som datorer, mobiltelefoner och surfplattor (se figur 6). Funktionerna som tidigare fanns på navigationsmenyn i första prototypen har flyttats till en dropdown-knapp som finns på vänstra sidan med en plus-symbol (se figur 7). En ny funktion har också implementerats för att kunna välja en instans av ett arbetsflöde och spåra processen för den valda instansen. Dessutom kan en vald instans tas bort. Om ett steg är klart ändras bakgrunden för det steget till grå och om den är pågående ändras bakgrunden till grön (se figur 8 och 9). Enkla animationer har också lagts till för alla dropdown-menyer, dropdown-knappar och för det steg som är pågående.



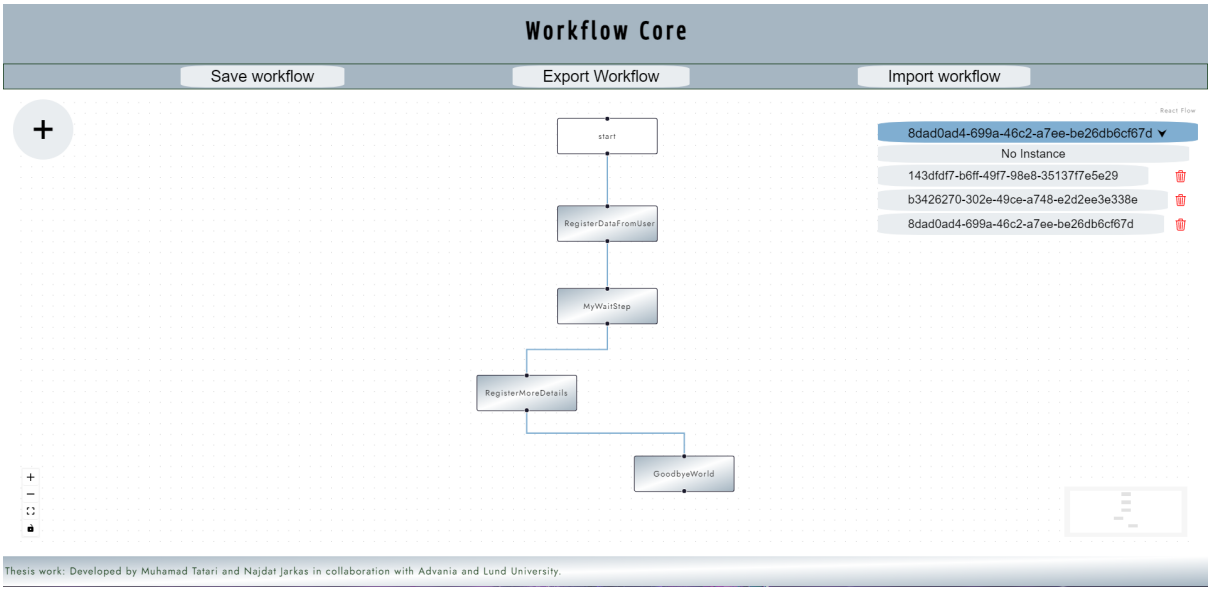
Figur 6: Översikt över hur prototyp 2 ser ut på olika enheter.



Figur 7: Översikt över hur prototyp 2 ser ut på en dator.



Figur 8: Spårning av ett arbetsflöde och framsteg för en vald instans.



Figur 9: Spårning av ett arbetsflöde och framsteg för en instans som har kört färdigt.

4.3 Kvalitetssäkring för den slutliga prototypen

För att leverera en fungerande prototyp genomfördes två typer av tester. Den första var testning av icke-funktionella krav, och den andra var funktionell testning genom testscenarier.

4.3.1 Testning av icke-funktionella krav

De kategorier av icke-funktionella krav som testades var: Tillgänglighet, prestanda, portabilitet, användbarhet och pålitlighet. De är presenterade som en checklista av frågor.

Checklista	
Kan man komma in i webbapplikationen på olika enheter?	**X**
Laddas webbsidan in fullständigt på mindre än 1 sekund?	**X**
Är alla knappar på plats?	**X**
Är knapparna så väl tilltagna att det går att klicka på dem på en liten skärm?	**X**
Fungerar drop-down-knapparna?	**X**
Kan man smidigt flytta blocken?	**X**
Kan man ansluta två block till varandra?	**X**
Ändras knappens färg när man för musen över den?	**X**
Kan man zooma in och ut?	**X**
Är språket på webbplatsen skriven på begriplig svenska??	**X**
Kan man fylla i formulären ?	**X**
Skulle en användare kunna skapa ett arbetsflöde utan problem ?	**X**

4.3.2 Testscenarier

Ett arbetsflöde ska skapas, sparas, spåras och exporteras till JSON-fil för varje scenario. I Appendix 8.1 github presenteras hur Workflow Core har använts för att köra alla testscenarierna. Koden för Testscenario 1 finns i mappen Test1 och så vidare. Dessutom återfinns det också data om alla steg, kopplingar mellan stegen och spårningsdata som genomfördes med hjälp av React-webbapplikationen i mappen CollectionInDatabase och i undermapparna Nodes(1), Edges(1) respektive WFollowingData(1) och så vidare.

4.3.2.1 Testscenario 1

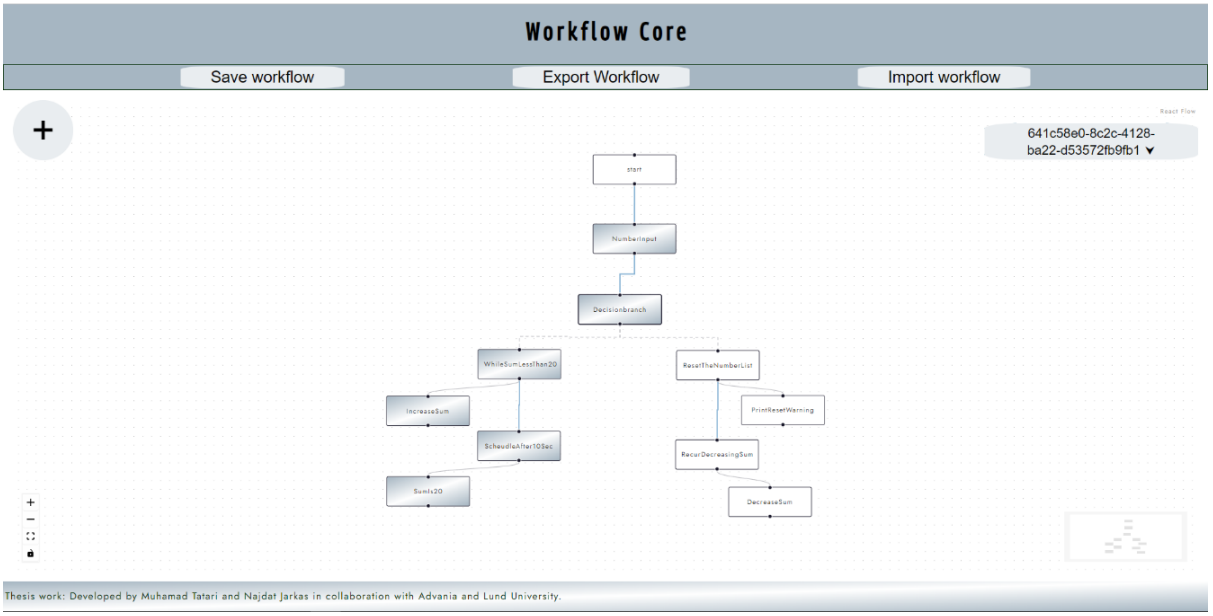
Arbetsflödet i detta scenario presenteras i figur 10 och består av följande steg:

1) Första steg: Användaren av arbetsflödet ska mata in två siffror. Sedan ska steget skicka vidare summan av dessa två siffrorna till övriga steg.

2) Andra steg: Detta steg är av typen “Decision Branch” och kontrollerar om summan är större eller mindre än 20. Beroende på det här villkoret väljer arbetsflödesmotorn vilken gren som arbetsflödet ska fortsätta i. Om summan är mindre än 20 så körs branch 1, medan om den är större så körs branch 2.

- Branch 1: Den här grenen består av tre steg:
  - a- While-loop steg: Den ökar summan med 1 tills summan blir 20.
  - b- Schema: schemalägger en minut och kör sen nästa steg.
  - c- Steg: Visa meddelandet “Summan är nu 20”.
- Branch 2: Den här grenen består också av tre steg:
  - a- Foreach-steg: i detta steg skrivs “summan är större än 20” ut tre gånger.
  - b- Recur-steg: Här minskas summan med 1 och det upprepas tills att summan blir noll.
  - c- Steg: Visa meddelandet “Summan är nu 0”

Scenariot testade följande funktioner: Att skapa vanligt steg, Decision branch-steg, Foreach-steg, Schedule-steg, Recur-steg och While-loop-steg. Dessutom testades spara-, exporterings-, importerings- och spårningsfunktionerna. Det uppstod ett kritiskt fel i Decision branch-steget. Det blev fel i textformatet av JSON-filen vid exportering. Dessutom återspeglades ändringar av Decision-branschen inte när JSON-filen exporterades. Efter korrigering i koden kördes samma test igen och försöket har lyckats utan problem.



Figur 10: Spårning av arbetsflödet i testscenario 1 och framstegen för en instans som har kört färdigt.

4.3.2.2 Testscenario 2

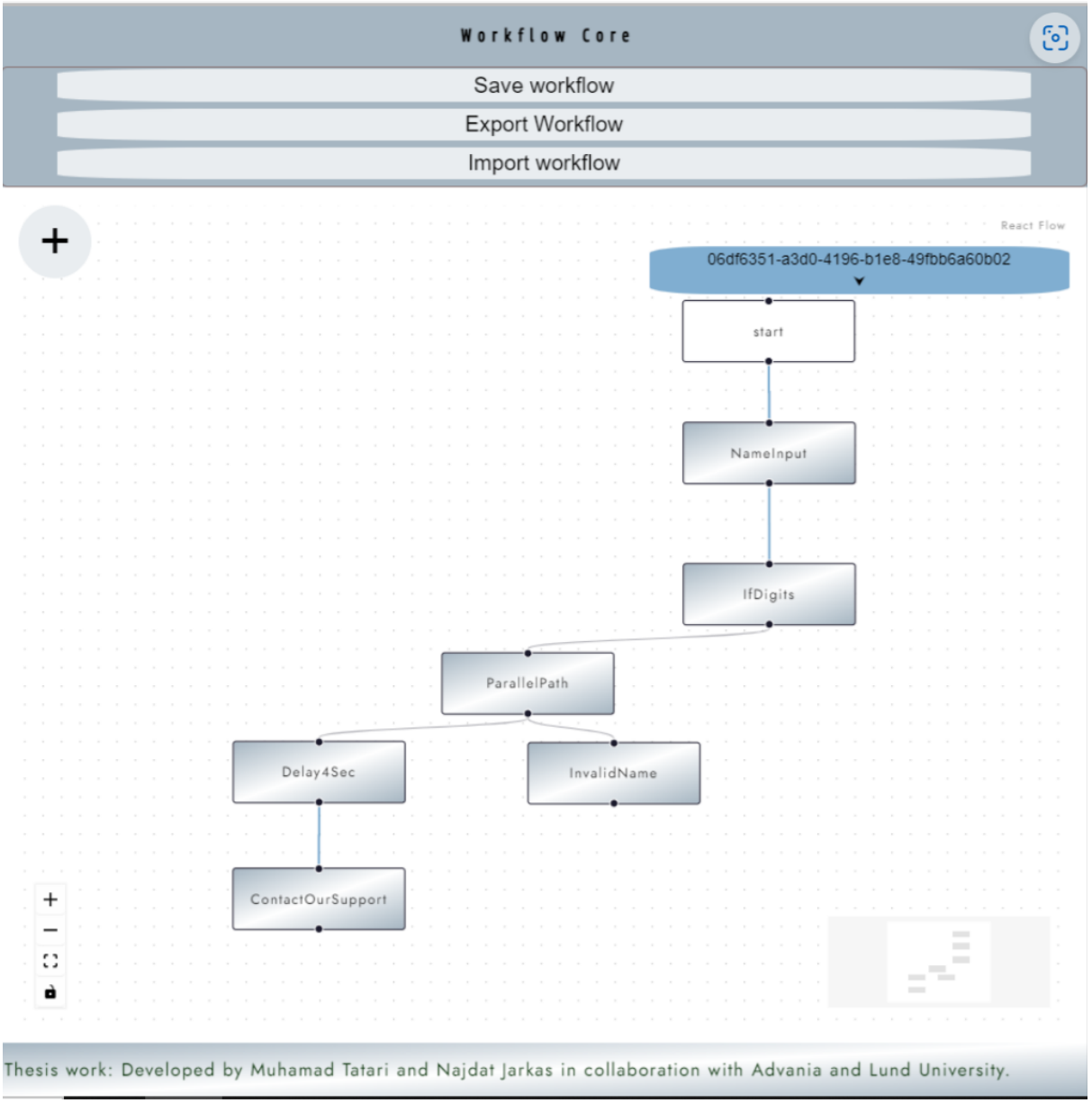
Arbetsflödet i detta scenario presenteras i figur 11 och består av följande steg:

1) Första steg: Användaren av arbetsflödet ska mata sitt namn. Sedan ska steget skicka vidare namnet till nästa steg.

2) Andra steg: Detta steg är av typen “If-sats”: Detta steg kontrollerar om uppgivet namn är giltigt eller inte. Ifall det är rätt namn så fortsätter det till nästa steg annars stoppas arbetsflödet.

3) Tredje steg: Detta steg är av typen “Parallel paths”: Den ska ha två parallella steg som skall göras. Den första är att skriva ”Ditt namn är (namn)” och den andra steget medför en fördröjning på fem sekunder. Sen skickas meddelandet “kontakta vår kundservice !”

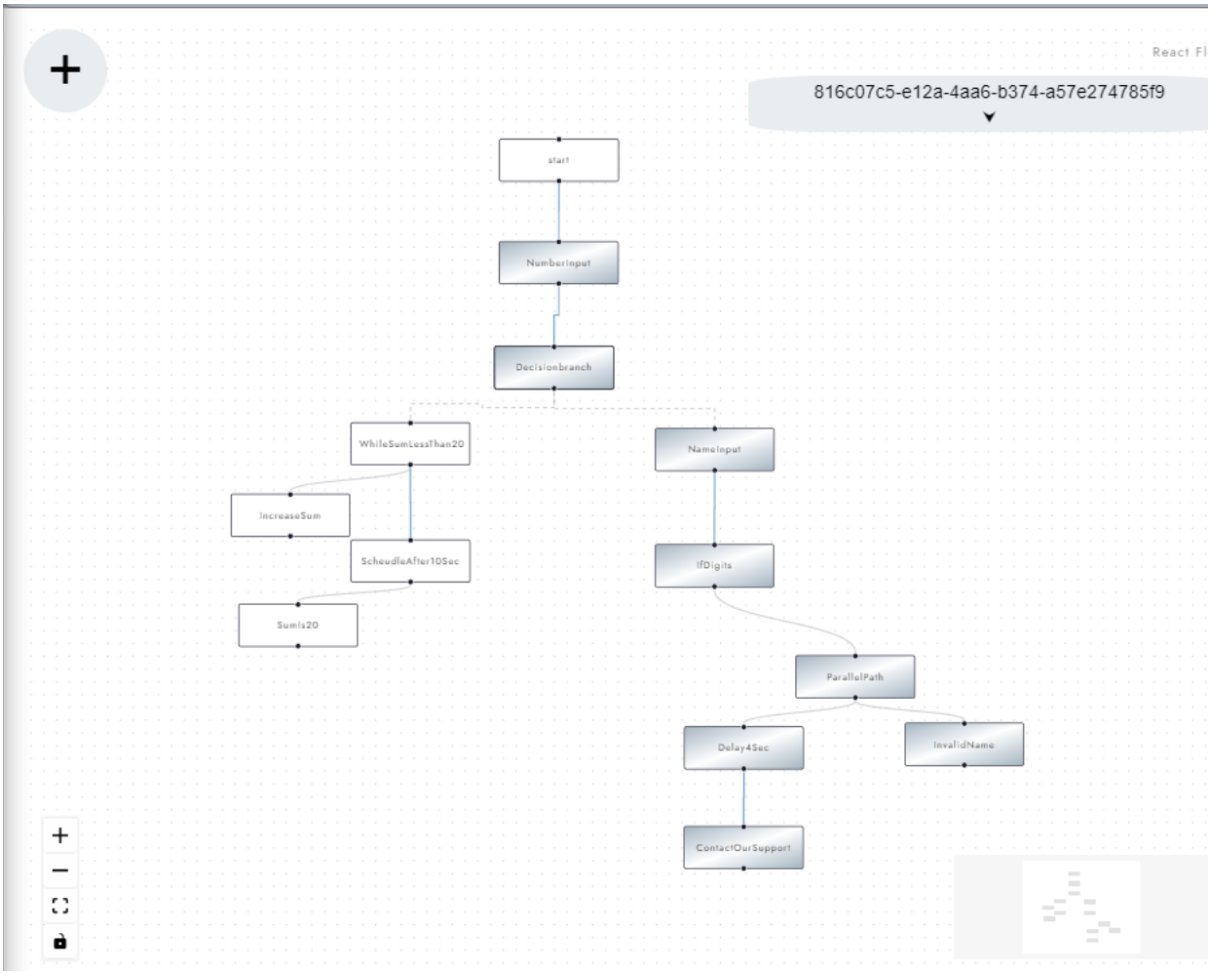
Detta scenario testade följande funktioner: Att skapa vanligt steg, “If-sats”, “Parallel paths” och “delay”. Exekveringen av testscenariot lyckades utan att kritiska fel hittades. JSON-filen exporterades och kördes i Workflow Core-programmet utan problem.



Figur 11: Spårning av arbetsflödet i testscenario 2 och framstegen för en instans som har kört färdigt.

4.3.2.3 Testscenario 3

I den här scenarion kombinerades arbetsflöden från scenario 1 och 2 för att kunna testa skapandet av ett stort arbetsflöde och testa alla funktionerna tillsammans (se figur 12). Exekveringen lyckades. JSON-filen exporterades och kördes i Workflow Core-programmet utan problem.



Figur 12: Spårning av arbetsflödet i testscenario 3 och framstegen för en instans som har kört färdigt.

4.4 Om Elasticsearch-indexering och MongoDB

Flera tester utfördes för att identifiera när indexeringsfel uppstår och för att undersöka kopplingen mellan en Elasticsearch-server och en MongoDB-databas genom ett Workflow Core-program. Dessutom genomfördes flera försök med avsikt att synkronisera en Elasticsearch-server med MongoDB-databas.

4.4.1 Elasticsearch-indexering

De fyra utförda testerna i tabell 1 i avsnitt 2.4.3 “Testning gällande Elasticsearch indexering” visade att indexeringen i Elasticsearch fungerar som förväntat i de flesta fallen. Under test 1, 2 och 4 hittades ingen avvikande beteende eller fel i indexeringen. Däremot observerades det i test 3 att dataintegriteten påverkades och att datan i MongoDB inte var synkroniserad med Elasticsearch-servern. Ett sådant dataintegritetsfel kan återskapas med följande scenario: Om ett steg i ett arbetsflödet ska uppdatera datan (som delas av hela arbetsflödet och som sparas i MongoDB) och Elaticsearch-servern samtidigt är nere för en längre period, så kommer uppdateringen inte att synkroniseras med Elasticsearch när den blir tillgänglig igen. Den perioden uppskattas vara mer än 5 minuter lång.

Något som bör observeras är att datan återigen blir korrekt synkroniserad om ett senare steg uppdaterar datan. Ett konkret exempel på detta, som observerades vid testningen, presenteras i tabell 3.

Tabell 3: Ett konkret exempel på när problem med dataintegriteten uppstår

	Datan i MongoDB	Datan i Elasticsearch-server
Elasticsearch-servern är uppe	sum = 30	sum = 30
Elasticsearch-servern är nere för en längre period, samtidigt som en datauppdatering skett	sum = 20	Ej tillgänglig
Elasticsearch-servern är uppe igen	sum = 20	sum = 30
En datauppdatering sker igen	sum = 20	sum = 20

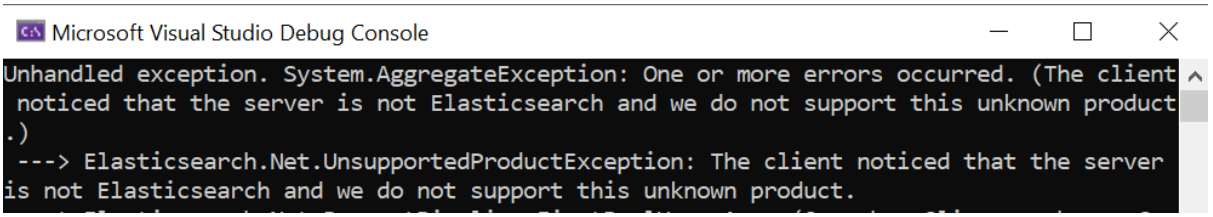
De återstående testerna som utfördes gällande Elasticsearch-pluginen visade att när data direkt manipulerades i MongoDB, reflekterades inte ändringar i Elasticsearch-servern. Dessutom påverkades inte indexeringen och datan i Elasticsearch-servern när både MongoDB och Elasticsearch-servern stängdes ner och startades om, samtidigt eller någon av dem i taget. Vid tester där datastrukturen för ett arbetsflöde ändrades upptäcktes inga fel i indexeringen. Tidigare data som hade indexerats före uppdateringen förblev oförändrad, vilket är förväntat, medan data som indexerades efter uppdatering av datastrukturen indexerades korrekt, och följde uppdateringen.

4.4.2 Försök att synkronisera Elasticsearch-server med MongoDB

Det genomfördes totalt fyra försök med fyra olika miljöer som presenterades i tabell 2 som finns i avsnitt 2.4.4 “Försök att synkronisera MongoDB och Elasticsearch”. Försöket i miljö 1 misslyckades redan i initieringsstadiet, då “.Net” biblioteket som gör indexering i Workflow Core inte lyckades skapa index i Elasticsearch-servern. Detta trots att inga felmeddelande visades i loggarna.

Vid försök i miljö 2 och 3 lyckades “.Net” bibliotek skapa index i Elasticsearch-servern, vilket visade att Elasticsearch-plugin i Workflow Core inte var kompatibel med Elasticsearch-server version 8.7.1 (se miljö 1 i tabell 2). Försöken att synkronisera MongoDB och Elasticsearch via Logstash (se figur 2) misslyckades, då inga index skapades i Elasticsearch-servern. Detta trots att loggarna inte visade några felmeddelande och dessutom bekräftade att kopplingen mellan MongoDB till Logstash och Elasticsearch hade lyckats. Vårt debuggingarbete visade även att Logstash fick rätt data från MongoDB, men trots det inte lyckades skapa de önskade indexen i Elasticsearch-servern. Relevanta delar av loggarna som visar att kopplingen via logstash var framgångsrik och att indata (inputs) från MongoDB är hämtad, finns i Appendix 8.3. Dessutom bifogas också konfigurationsfilen som användes för Logstash i Appendix 8.2.

Sist men inte minst misslyckades försöket i miljö 4 redan i initieringsstadiet, men denna gång visades ett felmeddelande gällande inkompatibilitet (se figur 13).



Figur 13: En del av felmeddelandet som dyker upp vid initieringsstadiet för miljö 4

5. Diskussion

Först presenteras en utvärdering av den slutliga prototypen och de observationer som gjordes under arbetet och testningen. Därefter analyseras Elasticsearch och MongoDB i relation till Workflow Core och de utförda försöken att synkronisera Elasticsearch med MongoDB. Efter det diskuteras möjliga områden för framtida arbete och en etisk reflektion tas upp. Slutligen presenteras en sammanfattning av detta examensarbete.

5.1 Workflow Core och den slutliga prototypen

För att kunna förbättra sättet att använda Workflow Core så skapades en prototyp i form av en webbapplikation. Prototypen syftar på att bygga det önskade arbetsflöden visuellt och använda flera funktioner som erbjuds av Workflow Core på ett smidigt sätt. Därefter kan man exportera arbetsflödet till en JSON-fil som i sin tur används i Workflow Core-programmet.

Vid prototyptestning fungerade alla kritiska funktioner enligt design. De utförda testerna och testscenarierna täckte en stor del av funktionaliteten, men ändå var tidsramen och antalet tester begränsade. Därför är det viktigt att se prototypen som en början till en färdig produkt och utföra ytterligare tester ifall den ska användas i praktiken. Under scenariotestningen hittades ett antal buggar som inte påverkade huvudfunktionaliteten i prototypen. Exempel på dessa buggar inkluderade fel som är relaterade till webbsidans animationer och att sidan inte reagerade omedelbart på ändringar vid rendering. Sådana buggar fixades så fort de upptäcktes, men samtidigt tyder det på att det kan finnas andra “dolda” buggar som kan eller inte kan påverka webbapplikationens syfte och mål.

Något som också observerades vid testscenario 2 är att de parallella stegen inte kördes parallellt utan de kördes sekventiellt. Enligt Daniel på [27] så är detta förväntat beteende då Workflow Core arbetsflöden förväntas vara långvariga bakgrundsprocesser. Användarna, som vi i det här arbetet, kan få en annan uppfattning om vad parallell exekvering är vid läsning av bibliotekets officiella dokumentation. Daniel belyser i svar på flera frågor på Github att parallellt i det här sammanhanget pekar på att ingen parallel bransch kan blocka någon annan bransch. Å ena sidan, enligt våra tester, blockas branscherna ändå om bland annat await Task.Delay(<<x sec>>) eller Thread.Sleep(<<x sec>>) används. Å andra sidan om något inbyggt steg i Workflow Core används, som till exempel Delay, så blockas inte de andra parallella branscherna från den fördröjningen. Därför är det viktigt att förstå vad som verkligen menas med parallell exekvering i det här sammanhanget. I och med att examensarbetarna, som andra användare, hade haft fel uppfattning om vad som menas med parallell exekvering så behövde vår algoritm som exporterar JSON-stegen från React-applikationen modifieras. Detta beror på att algoritmen förutsätter att parallella branscher körs parallellt i en icke sorterad ordning.

Sortering och ordning av de parallella stegen spelar dock en roll i det här fallet. Ett konkret exempel är följande:

- Steg x av typen Delay körs “parallellt” med steg y som är ett egenskapat steg.
    - Om x kommer först i JSON filen och sedan y, så ska både x och y köras “parallellt”, därmed ska y inte påverkas av x. Ifall x ska köra en process efter fördröjning och y fortfarande är upptagen så kommer x blockas även i det här fallet.
    - Om y kommer först i JSON filen och sedan x, då betyder det att x ska blockas tills y har exekverat klart.

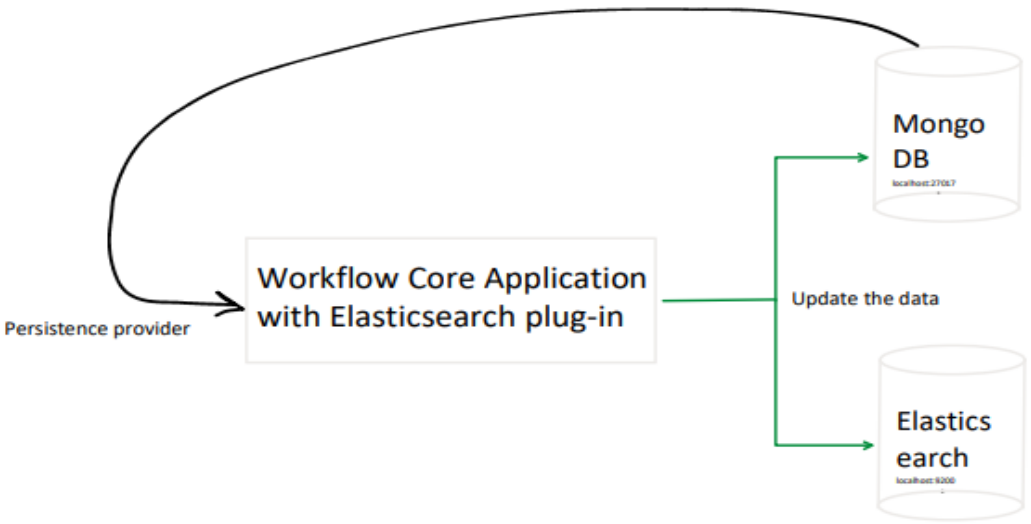
På grund av det modifierades algoritmen som exporterar JSON-filen så att den sorterar de parallella stegen i samma tur och ordning de kommer i React-webbapplikationen (från vänster till höger).

5.2 Elasticsearch

Efter att ha analyserat testerna gällande Elasticsearch-pluginen anser vi att MongoDB inte är synkroniserat eller direkt kopplad till Elasticsearch-servern. Detta baseras på att inga indexeringsfel observerades vid simulerade driftstörningar. Att direkta dataändringar gjorda i MongoDB inte återspeglades i Elasticsearch-servern, tyder på det också.

Det som sker är förmodligen att Workflow Core-applikationen skickar datauppdateringar via Elasticsearch-plugin till både MongoDB och Elasticsearch-Servern (se figur 14). Det visades också i testerna att dataintegritetsproblem uppstod när Elasticsearch-servern var nere under en längre period, men inte vid kortare perioder. Vilket kan förklaras med att datauppdateringar försöks skickas ett antal gånger eller alternativt buffras de en viss tid i hopp att få framgångsrik anslutning till Elasticsearch-servern.

För övrigt, baserat på tabell 3 som finns i avsnitt 4.4.1 “Elasticsearch-indexering”, kommer datan i Elasticsearch-servern att bli synkroniserad igen så fort en uppdatering sker, under förutsättningen att Elasticsearch-servern är tillgänglig. Detta innebär att det kan vara möjligt att, som förslag, schemalägga "fejkade" datauppdateringar samtidigt som ett arbetsflöde körs, tills det har slutförts. Detta förslag skulle vara enkelt att implementera med hjälp av React-webbapplikationen, som är byggd i det här arbetet, om det funnits stöd för fullt parallell exekvering i Workflow Core. Även utan stöd skulle detta ändå vara möjligt att implementera en sådan lösning genom att lägga ned mer arbete på koden och undersöka alla aspekter noggrant. En nackdel med detta förslag är att arbetsflödetsstrukturen blir mer komplex, samtidigt som minnes- och processoranvändning ökar då en massa med datauppdateringar skickas i onödan. Dessutom skulle det grundläggande problemet kvarstå, vilket är att MongoDB och Elasticsearch inte är direkt synkroniserade. Detta bortser dessutom från att omindexeringar fortfarande kan behövas om användaren önskar ändra i redan indexerade data eller om strukturen på arbetsflödet respektive MongoDB skulle ändras på ett genomgripande sätt.



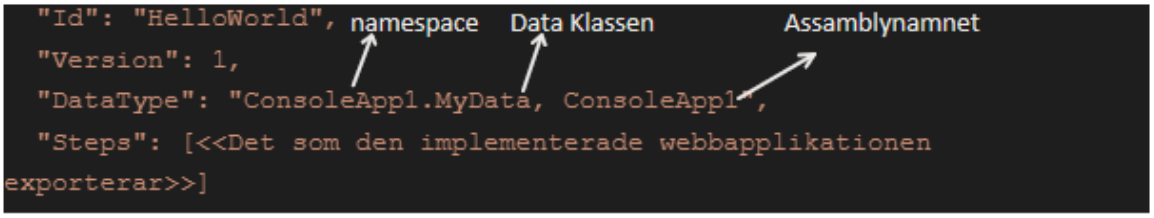
Figur 14: En modell för hur vi tror att Workflow Core fungerar med MongoDB och Elasticsearch-plugin.

När det gäller försöken att synkronisera MongoDB med Elasticsearch valdes i det här arbetet Logstash bland de andra metoderna som presenteras i den teoretiska bakgrunden (se avsnitt 3.6.3). Logstash valdes med tanken att både Elasticsearch och Logstash är officiella produkter från Elastic. Det ansågs att det fanns hög chans att uppnå en välfungerande synkronisering med denna kombination. Dessvärre misslyckades trots det de försöken i de olika miljöerna (som presenteras i tabell 2) med att få en fungerande synkronisering. Att få en fungerande synkronisering skulle hjälpa vid omindexering eller vid eventuellt flytt av MongoDB- och Elasticsearch-servrar.

Å ena sidan tycks det att de misslyckade Logstash-försöken i miljö 2 och 3, som presenteras i tabell 2, borde lyckas om mer ansträngning läggs ned för att hitta orsaken till att indexen inte skapades i Elasticsearch-servern. Detta trots att loggarna inte visade några felmeddelande och att debuggingarbetet skulle vara tidskrävande. Å andra sidan var det intressant att studera vidare och testa om en annan metod bland metoderna som presenteras i avsnitt 3.6.3 skulle uppnå en fungerande och effektiv synkronisering. Även i de andra metoderna förutses flera hinder, inklusive att säkerställa kompatibilitet mellan Elasticsearch-versionen som stöds av Workflow Core Elasticsearch-plugin och de Elasticsearch-versioner som stöds av verktygen i de andra metoderna. Dessvärre och på grund av komplexiteten hos de miljöerna vi arbetade med och tidsramen för detta examensarbete, var det inte möjligt att vidare utreda orsakerna till varför försöken misslyckades eller att utforska hur de andra metoderna skulle fungera.

5.3 Framtida arbete

Vid JSON-exportering i den slutliga prototypen sparas bara stegen av ett arbetsflöde I JSON-format. Det betyder att ett arbetsflödes ID, version och datatyp behöver fyllas på egen hand (se figur 15). I prototypen hanteras och spåras bara ett enda arbetsflöde med dess olika instanser i taget. Som framtida arbete skulle webapplikationen utvecklas för att även kunna hantera flera arbetsflöden åt gången. För detta ändamål behöver ändringar i MongoDB och användargränsnittet införas. En möjlig lösning är att bygga ett dashboard för webbapplikationen och en annan är att helt enkelt göra "import workflow" knappen till en dropdown-meny, vilket möjliggör hantering av flera arbetsflöden förutsatt att datastrukturen i MongoDB anpassas därefter. Efter sådan utveckling skulle arbetsflödets ID och version automatiskt, och på ett enkelt sätt, skrivas in i den exporterade JSON-filen. När det gäller datatypen för arbetsflöden är den hårdkodad till projektets assemblynamn, dataklassen och dess namespace i C#-projektet (se figur 15). Det hade också varit bra om prototypen kan anpassas så att datatypen antingen kan matas in eller automatiskt hämtas från projektet som använder Workflow Core.



Figur 15: Skelettet för ett Workflow Core-arbetsflöde i JSON-format. Datatypen är hårdkodad till C# projektet och pilarna visar hur datatypen bör skrivas.

Dessutom ska det vara möjligt att ha kontoregistrering och inloggning för användare. Syftet med det är att användare ska kunna skapa sina arbetsflöde och spara dem samtidigt som säkerheten förbättras. I ett sådant fall ska alla ha tillgång till sina egna arbetsflöden men ingen ska kunna ha obehörig åtkomst till någon annans data. Det arbetet skulle kräva utveckling och implementering för en avancerad och säker registreringsprocess.

Om den prototypen ska användas i praktiken bör fler tester att utföras och användarvänligheten behöver ses över. En del av detta arbete skulle innebära att studera vem användarna är. Därefter kan användartester utföras för att analysera hur användarna reagerar och för att kunna förbättra prototypen utifrån det.

5.4 Slutsats

Examensarbetet undersökte och besvarade frågorna som presenterades i avsnitt 1.3 “Problemformulering”.

1. **Vad är Workflow Core och varför ansåg Advania att deras användning av den är begränsad?**

- I detta examensarbete utforskades Workflow Core, som är en kraftfull arbetsflödesmotor designad för att hantera långvariga processer. Advania utvecklade bara den absolut nödvändiga funktionaliteten med Workflow Core på grund av resursbegränsningar och tidsbrist. De insåg dock att det fanns stor potential för vidareutveckling med Workflow Core.

2. **Vilka fördelar och nackdelar har Workflow Core jämfört med andra arbetsflödesmotorer baserade på öppen källkod, om sådana finns?**

- Workflow Core jämfördes med andra arbetflödestjänster baserade på öppen källkod och det påpekades att valet av arbetflödestjänst bör ta hänsyn till de individuella behoven och inte bara sammanföra funktioner som erbjuds av varje arbetsflödesmotor. Detta är eftersom det finns en uppsjö med kraftfulla arbetsflödesmotorer på marknaden.

3. **Varför valde Advania just Workflow Core?**

- Grundläggande krav för urvalet av arbetflödestjänst sattes av Advania och det visade sig att Workflow Core var en bra match utifrån ett säkerhetsperspektiv, eftersom Workflow Core bland annat inte är molnbaserad.

4. **Hur kan Workflow Core vidareutvecklas och implementeras för att utnyttja fler funktioner och möjligheter?**

- Det utvecklades en webbapplikation i detta examensarbete som skulle underlätta skapandet av olika arbetsflöden i framtiden. Webbapplikationen har en responsiv design och tillåter användare att visuellt bygga arbetsflöden för att sedan exportera en JSON-fil som direkt kan utnyttjas av ett Workflow Core-program. Förutom det har webbapplikationen möjlighet att spåra framsteg för olika instanser av ett arbetsflöde.

5. **Vilka typer av fel kan uppstå vid anslutningen mellan Elasticsearch och MongoDB? Kan de hanteras och hur?**

- Genomförda tester i det här arbetet tyder på att det inte finns en direkt koppling mellan Elasticsearch-server och MongoDB när Elasticsearch-plugin i Workflow Core används. Om Elasticsearch-servern är nere för en längre period samtidigt som en uppdatering av arbetsflödet sker så uppstår det dataintegritetsfel då Elasticsearch-servern och MongoDB inte är synkroniserade. För att hantera synkroniseringen presenterades fyra metoder via Logstash, Mongo-Connector, Elasticsearch-River-MongoDB och Transporter. De utförda försöken för att synkronisera Elasticsearch-servern med MongoDB via Logstash misslyckades. En sådan synkronisering skulle hjälpa om en omindexering av Elasticsearch-server skulle behövas.

5.5 Etiska reflektioner

Examensarbetet kan vara användbart inte bara för företaget Advania utan även till andra företag eller programmerare som använder Workflow Core eller planerar att använda Workflow Core. De kan dra inspiration från användningen av React-Flow för att skapa en anpassad användargränssnitt som förenklar skapandet och hanteringen av arbetsflöden i Workflow Core. Dessutom kan de titta på den analys som gjordes om hur Elasticsearch-pluginen fungerar tillsammans med MongoDB som lagringsleverantör i Workflow Core. Sist men inte minst kan det också vara intressant att studera våra misslyckade försök att synkronisera en Elasticsearch-server och en MongoDB och se om någon av metoderna som presenteras i det här arbetet kan vara till nytta.

5.5.1 Sekretess

I och med att arbetsflödemotorer som används av företag kan involvera insamling och lagring av data om användare så är sekretess en viktig etisk aspekt att beakta. Det är viktigt att läsa vilka sekretess- och GDPR-regler som gäller en arbetsflödes- motor för att säkerställa att datainsamlingen och användningen av personlig information hanteras och skyddas på rätt sätt. Arbetsflödesmotorer måste ha lämpliga åtgärder för att skydda och anonymisera sina användare mot obehöriga åtkomst och missbruk.

6. Terminologi

- .NET - En ramverk utvecklat av Microsoft som stöder utveckling och körning av appar och webbtjänster i Windows [28]

- C# - En objektorientad programmeringsspråk utvecklat av Microsoft och är en del av :NET-ramverket. [29]

- Öppen-källkod - En programvara med källkod som alla kan inspektera, modifiera och förbättra. [30]

- MIT-license - MIT-licensen ger tillstånd för användare att återanvända kod för alla ändamål. En originalkopia av MIT-licensen måste vara med på den modifierade programmet innan distributionen. [31]

- JSON - JavaScript Object Notation, ett öppet datautbytesformat som är både mänskligt och maskinläsbart. JSON är oberoende av alla programmeringsspråk och är en vanlig API-utgång i en mängd olika applikationer.[32]

- NOSQL-databas - En dokumentorienterad databas som lagrar data annorlunda än relationäl databaser. [33]

- React - En ramverk som används för att bygga användargränssnitt. [34]

7. Referenser

[1] “Om oss | Vi gör det enkelt att växa med IT | Advania,” www.advania.se. https://www.advania.se/om-advania (accessed April 02, 2023).

[2] “Workflow Core.” workflow-core.readthedocs.io. https://workflow-core.readthedocs.io/en/latest (accessed April 03, 2023).

[3] D. Gerlag. “Workflow Core.” GitHub.com. https://github.com/danielgerlag/workflow-core/blob/master/LICENSE.md (accessed April 03, 2023).

[4] R. Abueg. “What is Elasticsearch and what is it used for?.” Knowi.com. https://www.knowi.com/blog/what-is-elastic-search (accessed April 04, 2023).

[5] “What is a Non-Relational Database?.” MongoDB. https://www.mongodb.com/databases/non-relational (accessed April 05, 2023).

[6] J. Herrity. “How to Succeed in Unstructured Interviews.” Indeed Career Guide. https://www.indeed.com/career-advice/interviewing/unstructured-interviews (accessed May 24, 2023).

[7] S. Bose. “How to create Test Scenarios with Examples.” BrowserStack. https://www.browserstack.com/guide/how-to-create-test-scenarios (accessed May 24, 2023).

[8] L. L, S. Sriya, B. B and N. S. Kumar, "Flexible Workflow Engine With Load Balancing," in *2022 International Conference for Advancement in Technology (ICONAT)*, Goa, India, 2022, pp. 1-5, doi: 10.1109/ICONAT53423.2022.9725954.

[9] M. Wahnon. “awesome-workflow-engines.” GitHub.com. https://github.com/meirwah/awesome-workflow-engines#readme (accessed April 15, 2023).

[10] L. Hongbiao, L. Feng and Y. Wanjun, "The research of scientific workflow engine," in *2010 IEEE International Conference on Software Engineering and Service Sciences*, Beijing, China, 2010, pp. 412-414, doi: 10.1109/ICSESS.2010.5552352.

[11] “elsa-workflows/elsa-core.” GitHub.com. https://github.com/elsa-workflows/elsa-core (accessed April 16, 2023).

[12] “Elsa Workflows - Add workflow capabilities to any .NET project.” v2.elsaworkflows.io. https://v2.elsaworkflows.io (accessed April 15, 2023).

[13] “Features · ELSA.” elsa-workflows.github.io. https://elsa-workflows.github.io/elsa-core/docs/features/features (accessed April 15, 2023).

[14] “Introduction - React Flow.” reactflow.dev. https://reactflow.dev/docs/concepts/introduction (accessed April 06, 2023).

[15] A. K. Samanta and N. Chaki, "Performance Monitoring of MongoDB on Varied Cluster Configuration: An Experimental Approach," in *2021 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT),* Zallaq, Bahrain, 2021, pp. 525-530, doi: 10.1109/3ICT53449.2021.9581673.

[16] P. W. Jordaan and J. E. W. Holm, "Reflection on MongoDB Database Logical and Physical Modeling," in *2019 IEEE AFRICON*, Accra, Ghana, 2019, pp. 1-8, doi: 10.1109/AFRICON46755.2019.9133972.

[17] B. Wei, J. Dai, L. Deng and H. Huang, "An Optimization Method for Elasticsearch Index Shard Number," in *2020 16th International Conference on Computational Intelligence and Security (CIS)*, Guangxi, China, 2020, pp. 191-195, doi: 10.1109/CIS52066.2020.00048.

[18] “What is Elasticsearch | Elastic.” Elastic.co. https://www.elastic.co/what-is/elasticsearch (accessed May 11, 2023).

[19] L. Vokorokos, M. Uchnár and L. Leščišin, "Performance optimization of applications based on non-relational databases," in *2016 International Conference on Emerging eLearning Technologies and Applications (ICETA)*, Stary Smokovec, Slovakia, 2016, pp. 371-376, doi: 10.1109/ICETA.2016.7802079.

[20] “Query and filter context | Elasticsearch Guide [8.7] | Elastic.” Elastic.co. https://www.elastic.co/guide/en/elasticsearch/reference/current/query-filter-context.html#query-context (accessed May 13, 2023).

[21] “Scalability and resilience: clusters, nodes, and shards | Elasticsearch Guide [8.6] | Elastic.” Elastic.co. https://www.elastic.co/guide/en/elasticsearch/reference/current/scalability.html (accessed May 13, 2023).

[22] “Reindex an Elasticsearch index.” *GOV.UK Developer Documentation*. https://docs.publishing.service.gov.uk/manual/reindex-elasticsearch.html (accessed May 13, 2023).

[23] V. Agrawal “Integrating Elasticsearch and MongoDB: Made Easy - Learn | Hevo.” https://hevodata.com/learn/integrating-elasticsearch-and-mongodb/#MongoDB_River_Plugin (accessed May 14, 2023).

[24] A. Gupta. “5 Different ways to synchronize data from MongoDB to ElasticSearch.” Medium. https://code.likeagirl.io/5-different-ways-to-synchronize-data-from-mongodb-to-elasticsearch-d8456b83d44f (accessed May 14, 2023).

[25] “Getting Started with Logstash | Logstash Reference [8.7] | Elastic.” Elastic.co. https://www.elastic.co/guide/en/logstash/current/getting-started-with-logstash.html (accessed May 14, 2023).

[26] R. Willy. “elasticsearch-river-mongoDB.” GitHub.com. https://github.com/richardwilly98/elasticsearch-river-mongodb (accessed May 14, 2023).

[27] “Parallel execution of steps is not async · Issue #793 · danielgerlag/workflow-core,” *GitHub*. https://github.com/danielgerlag/workflow-core/issues/793 (accessed May 15, 2023).

[28] “Översikt över .NET Framework” learn.microsoft.com. https://learn.microsoft.com/sv-se/dotnet/framework/get-started/overview (accessed May 24, 2023).

[29] “Object-Oriented Programming (C#),” learn.microsoft.com. https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/tutorials/oop (accessed May 24, 2023).

[30] “What is open source?,” Opensource.com. https://opensource.com/resources/what-open-source (accessed May 24, 2023).

[31] “What is the MIT License? Top 10 questions answered | Snyk,” https://snyk.io/learn/what-is-mit-license/ (accessed May 24, 2023).

[32] “What is JSON? ” Amazon Web Services, Inc. https://aws.amazon.com/documentdb/what-is-json/ (accessed May 24, 2023).

[33] “NoSQL Databases Explained,” MongoDB. https://www.mongodb.com/nosql-explained (accessed May 24, 2023).

[34] “React,” react.dev. https://react.dev/ (accessed May 24, 2023).

8. Appendix

8.1 Github

https://github.com/MTatari/ExjobbAppendix

8.2 Konfigurationsfilen till Logstash

```
input {
        mongodb {
            uri => 'mongodb://localhost:27017/ReactwebbApplication'
                        placeholder_db_dir =>
'C:\Users\semutat001\Downloads\logstash-7.0.0\logstash-7.0.0\bin\opt\logstash-mongodb'
                        placeholder_db_name => 'logstash_sqlite.db'
                        collection => 'sampleWF'
                        unpack_mongo_id => true
        }
}

filter {

}

output {
      stdout {
            codec => rubydebug
      }
      elasticsearch {
            action => "index"
```

```
        index => "indextest1"
        hosts => ["localhost:9200"]
    }
}
```

8.3 logstash loggar

-[2023-05-13T11:13:00,175][INFO][logstash.outputs.elasticsearch] New Elasticsearch output {:class=>"LogStash::Outputs::ElasticSearch", :hosts=>["//localhost:9200"]}

-[2023-05-13T11:13:00,183][INFO][logstash.outputs.elasticsearch] Using default mapping template

-[2023-05-13T11:13:00,191][INFO][logstash.javapipeline] Starting pipeline {:pipeline_id=>"main", "pipeline.workers"=>12, "pipeline.batch.size"=>125, "pipeline.batch.delay"=>50, "pipeline.max_inflight"=>1500, :thread=>"#<Thread:0x49c2e459 run>"}

-[2023-05-13T11:13:00,276][INFO][logstash.outputs.elasticsearch] Attempting to install template {:manage_template=>{"index_patterns"=>"logstash-*", "version"=>60001, "settings"=>{"index.refresh_interval"=>"5s", "number_of_shards"=>1, "index.lifecycle.name"=>"logstash-policy", "index.lifecycle.rollover_alias"=>"logstash"}, "mappings"=>{"dynamic_templates"=>[{"message_field"=>{"path_match"=>"message", "match_mapping_type"=>"string", "mapping"=>{"type"=>"text", "norms"=>false}}}, {"string_fields"=>{"match"=>"*", "match_mapping_type"=>"string", "mapping"=>{"type"=>"text", "norms"=>false, "fields"=>{"keyword"=>{"type"=>"keyword", "ignore_above"=>256}}}}}], "properties"=>{"@timestamp"=>{"type"=>"date"}, "@version"=>{"type"=>"keyword"}, "geoip"=>{"dynamic"=>true, "properties"=>{"ip"=>{"type"=>"ip"}, "location"=>{"type"=>"geo_point"}, "latitude"=>{"type"=>"half_float"}, "longitude"=>{"type"=>"half_float"}}}}}}}

-[2023-05-13T11:13:01,186][INFO][logstash.inputs.mongodb] Registering MongoDB input

-[2023-05-13T11:13:02,084][INFO][logstash.inputs.mongodb] init placeholder for logstash_since_sampleWF: {"_id"=>BSON::ObjectId('645f546038d6326c4fa7e5ea'), "Data"=>{"_t"=>"ConsoleApp1.MyData, ConsoleApp1, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null", "firstNum"=>13, "secondNum"=>8, "sum"=>21, "numberList"=>[13, 8, 21], "name"=>"abc"}, "Description"=>nil, "Reference"=>nil, "WorkflowDefinitionId"=>"HelloWorld", "Version"=>1, "NextExecution"=>nil, "Status"=>"Complete", "CreateTime"=>2023-05-11 13:42:44 UTC, "CompleteTime"=>2023-05-11 13:42:55 UTC, "ExecutionPointers"=>[{"_id"=>"eef5b783-6aaf-49dc-a2ee-bb21d63277c9", "StepId"=>0,

"Active"=>false, "SleepUntil"=>nil, "PersistenceData"=>nil, "StartTime"=>2023-05-11
13:42:44 UTC, "EndTime"=>2023-05-11 13:42:48 UTC, "EventName"=>nil, "EventKey"=>nil,
"EventPublished"=>false, "EventData"=>nil, "ExtensionAttributes"=>{}, "StepName"=>nil,
"RetryCount"=>0, "Children"=>[], "ContextItem"=>nil, "PredecessorId"=>nil, "Outcome"=>nil,
"Status"=>"Complete", "Scope"=>[]}, {"_id"=>"b406a5ba-bc29-48be-83bc-0f5dc79dfe44",
"StepId"=>1, "Active"=>false, "SleepUntil"=>nil, "PersistenceData"=>nil,
"StartTime"=>2023-05-11 13:42:48 UTC, "EndTime"=>2023-05-11 13:42:48 UTC,
"EventName"=>nil, "EventKey"=>nil, "EventPublished"=>false, "EventData"=>nil,
"ExtensionAttributes"=>{}, "StepName"=>nil, "RetryCount"=>0, "Children"=>[],
"ContextItem"=>nil, "PredecessorId"=>"eef5b783-6aaf-49dc-a2ee-bb21d63277c9",
"Outcome"=>nil, "Status"=>"Complete", "Scope"=>[]},
{"_id"=>"5281d9e3-434e-41d6-9315-5c886472828c", "StepId"=>6, "Active"=>false,
"SleepUntil"=>nil, "PersistenceData"=>nil, "StartTime"=>2023-05-11 13:42:48 UTC,
"EndTime"=>2023-05-11 13:42:50 UTC, "EventName"=>nil, "EventKey"=>nil,
"EventPublished"=>false, "EventData"=>nil, "ExtensionAttributes"=>{}, "StepName"=>nil,
"RetryCount"=>0, "Children"=>[], "ContextItem"=>nil,
"PredecessorId"=>"b406a5ba-bc29-48be-83bc-0f5dc79dfe44", "Outcome"=>nil,
"Status"=>"Complete", "Scope"=>[]}, {"_id"=>"4ef3cdb7-3265-4f77-8ff4-da988fa3cf1b",
"StepId"=>7, "Active"=>false, "SleepUntil"=>nil, "PersistenceData"=>nil,
"StartTime"=>2023-05-11 13:42:50 UTC, "EndTime"=>2023-05-11 13:42:55 UTC,
"EventName"=>nil, "EventKey"=>nil, "EventPublished"=>false, "EventData"=>nil,
"ExtensionAttributes"=>{}, "StepName"=>nil, "RetryCount"=>0,
"Children"=>["d89857b1-1321-470f-8ab1-53f65bb901de"], "ContextItem"=>nil,
"PredecessorId"=>"5281d9e3-434e-41d6-9315-5c886472828c", "Outcome"=>nil,
"Status"=>"Complete", "Scope"=>[]}, {"_id"=>"d89857b1-1321-470f-8ab1-53f65bb901de",
"StepId"=>8, "Active"=>false, "SleepUntil"=>nil, "PersistenceData"=>nil,
"StartTime"=>2023-05-11 13:42:50 UTC, "EndTime"=>2023-05-11 13:42:55 UTC,
"EventName"=>nil, "EventKey"=>nil, "EventPublished"=>false, "EventData"=>nil,
"ExtensionAttributes"=>{}, "StepName"=>nil, "RetryCount"=>0,
"Children"=>["8fec2b64-a575-4991-9556-1a32ed256c23",
"79120a04-ddd5-43b3-ad5c-cf17ac1478e7"], "ContextItem"=>nil,
"PredecessorId"=>"4ef3cdb7-3265-4f77-8ff4-da988fa3cf1b", "Outcome"=>nil,
"Status"=>"Complete", "Scope"=>["4ef3cdb7-3265-4f77-8ff4-da988fa3cf1b"]},
{"_id"=>"8fec2b64-a575-4991-9556-1a32ed256c23", "StepId"=>10, "Active"=>false,
"SleepUntil"=>2023-05-11 13:42:55 UTC, "PersistenceData"=>nil, "StartTime"=>2023-05-11
13:42:50 UTC, "EndTime"=>2023-05-11 13:42:55 UTC, "EventName"=>nil, "EventKey"=>nil,
"EventPublished"=>false, "EventData"=>nil, "ExtensionAttributes"=>{}, "StepName"=>nil,
"RetryCount"=>0, "Children"=>[], "ContextItem"=>nil,
"PredecessorId"=>"d89857b1-1321-470f-8ab1-53f65bb901de", "Outcome"=>nil,
"Status"=>"Complete", "Scope"=>["d89857b1-1321-470f-8ab1-53f65bb901de",
"4ef3cdb7-3265-4f77-8ff4-da988fa3cf1b"]},
{"_id"=>"79120a04-ddd5-43b3-ad5c-cf17ac1478e7", "StepId"=>9, "Active"=>false,
"SleepUntil"=>nil, "PersistenceData"=>nil, "StartTime"=>2023-05-11 13:42:50 UTC,
"EndTime"=>2023-05-11 13:42:50 UTC, "EventName"=>nil, "EventKey"=>nil,
"EventPublished"=>false, "EventData"=>nil, "ExtensionAttributes"=>{}, "StepName"=>nil,
"RetryCount"=>0, "Children"=>[], "ContextItem"=>nil,
"PredecessorId"=>"d89857b1-1321-470f-8ab1-53f65bb901de", "Outcome"=>nil,
"Status"=>"Complete", "Scope"=>["d89857b1-1321-470f-8ab1-53f65bb901de",

"4ef3cdb7-3265-4f77-8ff4-da988fa3cf1b"]},
{"_id"=>"f7eeffd0-e06b-4293-abfc-46af7010d45a", "StepId"=>11, "Active"=>false,
"SleepUntil"=>nil, "PersistenceData"=>nil, "StartTime"=>2023-05-11 13:42:55 UTC,
"EndTime"=>2023-05-11 13:42:55 UTC, "EventName"=>nil, "EventKey"=>nil,
"EventPublished"=>false, "EventData"=>nil, "ExtensionAttributes"=>{}, "StepName"=>nil,
"RetryCount"=>0, "Children"=>[], "ContextItem"=>nil,
"PredecessorId"=>"8fec2b64-a575-4991-9556-1a32ed256c23", "Outcome"=>nil,
"Status"=>"Complete", "Scope"=>["d89857b1-1321-470f-8ab1-53f65bb901de",
"4ef3cdb7-3265-4f77-8ff4-da988fa3cf1b"]}]}

-[2023-05-13T11:13:02,090][INFO][logstash.javapipeline] Pipeline started
{"pipeline.id"=>"main"}

-[2023-05-13T11:13:02,116][INFO][logstash.agent] Pipelines running {:count=>1,
:running_pipelines=>[:main], :non_running_pipelines=>[]}

-[2023-05-13T11:13:02,286][INFO][logstash.agent] Successfully started Logstash
API endpoint {:port=>9600}