



EXAMENSARBETE INOM TEKNIK,  
GRUNDNIVÅ, 15 HP  
*STOCKHOLM, SVERIGE 2019*

# **Röststyrda applikationer och tillhörande arkitektur, design och utveckling**

Johan Andersson & Evan Saboo

## **Författare**

Johan Andersson, ande9@kth.se  
Information and Communication Technology  
KTH Royal Institute of Technology

Evan Saboo, saboo@kth.se  
Degree Programme in Computer Engineering  
KTH Royal Institute of Technology

## **Plats**

Stockholm, Sverige

## **Examinator**

Anders Sjögren  
Software and Computer Systems  
KTH Royal Institute of Technology

## **Handledare**

Fredrik Lundevall  
Software and Computer Systems  
KTH Royal Institute of Technology

## **Abstract**

Röststyrning och rösttolkning är ett gränssnitt mellan användare och dator som blivit vanligare i kommersiella produkter. Gränssnittet används i digitala assistenter, det vill säga en mjukvarubaserad tjänst utformad för att hjälpa användare att utföra digitala uppgifter. Dessa uppgifter inkluderar att svara på frågor, hantera scheman, hemkontroll, spela musik och mycket mer. Eftersom digitala assistenter är relativt nya så finns det ett behov av mer kunskap om hur man kan skapa applikationer för plattformen.

Rapporten ger kunskap om hur utvecklingsprocessen ser ut för en röststyrd applikation till Google Assistant. Detta görs med en fallstudie som ger en inblick i de olika design- och arkitekturval som ingår i mjukvaruutvecklingen. Resultatet beskriver lämpliga konversationsmönster för röstgränssnitt i röststyrda applikationer och ett lämpligt arkitekturmönster för kodbasen. Med hjälp av studien drogs slutsatser om vilka begränsningar som finns hos röststyrda applikationer.

## **Nyckelord**

Röstgränssnitt; Google Assistant; Programmering; Digital assistent; Arkitekturmönster; Konversationsmönster

## **Abstract**

Voice control and voice interpretation is an interface between users and computers that have become more common in commercial products. The interface is used in digital assistants, which is a software-based service designed to help users perform digital tasks. These tasks include answering questions, managing their schedules, home control, playing music and more. Because digital assistants are relatively new, there is a need for more knowledge about how to create applications for the platform.

The report provides information on how the development process looks like for a voice-controlled application for Google Assistant. This is done by a case study that provides insight into the various design and architecture choices that are included in the software development. The result describes a suitable conversation pattern for voice interfaces in voice-controlled applications and an appropriate architecture for the codebase. The study draws conclusions about the limitations of voice-controlled applications.

## **Keywords**

Voice-user Interface; Google Assistant; Programming; Digital Assistant; Architectural Pattern; Conversation Pattern

# Innehållsförteckning

<b>1</b>	<b>Introduktion</b>	<b>1</b>
1.1	Bakgrund . . . . .	1
1.2	Problem . . . . .	2
1.3	Syfte . . . . .	2
1.4	Mål . . . . .	3
1.5	Etik och samhällsnytta . . . . .	3
1.6	Metod . . . . .	4
1.7	Avgränsningar . . . . .	4
1.8	Disposition . . . . .	4
<b>2</b>	<b>Digitala assistenter och bakomliggande teknologi</b>	<b>5</b>
2.1	Artificiell intelligens . . . . .	5
2.2	Natural Language Processing . . . . .	7
2.3	Google Assistant . . . . .	9
2.4	Alexa, Siri och Cortana . . . . .	11
2.5	Google Cloud Platform . . . . .	12
2.6	Verktyg för utveckling av actions till Google Assistant . . . . .	13
2.7	Systemarkitektur av en röststyrd applikation . . . . .	15
2.8	Diverse tekniska verktyg . . . . .	17
2.9	Mönster inom mjukvaruutveckling . . . . .	20
2.10	Teori om projektmetoder . . . . .	24
2.11	Relaterat Arbete . . . . .	25
<b>3</b>	<b>Undersöknings- och projektmetoder</b>	<b>27</b>
3.1	Undersökningsmetod . . . . .	27
3.2	Hur frågeställningarna besvarades . . . . .	29
3.3	Fallstudiens utformning . . . . .	31
3.4	Val av teknologi . . . . .	32
3.5	Datahantering och Informationssökning . . . . .	33
3.6	Projektmetod . . . . .	34
3.7	Iterationbeskrivning och kravspecifikation . . . . .	35
<b>4</b>	<b>Utveckling av den röststyrda applikationen</b>	<b>40</b>

4.1	Konfiguration av Dialogflow . . . . .	40
4.2	Uppsättning av server, databas och Google Calendar API . . . . .	45
4.3	Hantering av fulfillments . . . . .	48
4.4	Testnings- och granskningsfas . . . . .	62
<b>5</b>	<b>Resultat</b>	<b>67</b>
5.1	Utvecklingsprocess . . . . .	67
5.2	Konversationsmönster för röstsgränssnitt . . . . .	69
5.3	Arkitekturmönster . . . . .	70
5.4	Konceptuell arkitektur . . . . .	71
5.5	Teknikvalens påverkan på programmeringsmodellen . . . . .	73
5.6	Begränsningar med den använda teknologin . . . . .	73
5.7	Applikationens funktionalitet . . . . .	74
<b>6</b>	<b>Slutsatser och diskussion</b>	<b>76</b>
6.1	Validitet och reliabilitet i metoden . . . . .	76
6.2	Validitet och reliabilitet i resultaten . . . . .	78
6.3	Besvarande av frågeställningar . . . . .	79
6.4	Hållbar utveckling . . . . .	83
6.5	Framtida arbete . . . . .	83
<b>7</b>	<b>Bilagor</b>	<b>95</b>

# 1 Introduktion

Taligenkänning (Speech recognition) är förmågan hos en maskin eller ett program att identifiera ord och fraser i talat språk och konvertera dem till ett maskinläsbart format. Teknologin används idag i flera områden såsom finans, marknadsföring och handel [1]. Taligenkänning används för att implementera röstanvändargränssnitt (Voice User Interface eller VUI). VUI låter användaren använda ett system genom röst- eller talkommandon. Detta möjliggör ett hand- och ögonfritt sätt att interagera med ett system medan man har sin uppmärksamhet åt annat håll. Eftersom användaren inte får något grafiskt gränssnitt till sitt förfogande måste en VUI tydligt uppge vilka möjliga interaktionsalternativ användaren kan utnyttja [2].

Ett användningsområde för VUI är i digitala assistenter. En digital assistent, även kallad virtuell assistent, är ett mjukvaruprogram som tar emot röst- eller textbaserade kommandon från användaren och försöker utföra kommandot. Den första röstigenkänningssystemet IBM Shoebox skapades av William C. Dersch och IBM i början av 1960-talet. Systemet kunde hantera 16 ord genom att ta emot röstkommandon via en mikrofon [3].

## 1.1 Bakgrund

En digital assistent går igenom tre steg för att kunna slutföra användarens röstkommando. I de två första stegen försöker assistenten tolka användarens kommando med hjälp av Natural Language Processing (NLP) [4]. Detta görs genom att först anropa en databas av inspelade ord och bäst matcha dem med användarens röstkommando. I andra steget försöker assistenten tolka exakt vad röstkommandot betyder. Systemet tar först fram olika relevanta svar och försöker sedan hitta det mest relevanta svaret genom att använda informationen den har lärt sig från tidigare röstkommandon. I sista steget utförs uppgiften som användaren förfrågade såsom att svara på enkla frågor eller utföra digitala kommandon. [5]

Det finns flera olika digitala assistenter på marknaden i dagsläget till olika

plattformar. Stora IT-företag som Google, Apple, Microsoft och Amazon har utvecklat egna assistenter som finns tillgängliga i dess produkter. Google Assistant finns tillgänglig i telefoner och på smarta högtalare [6]. Siri är Apples assistent som finns installerad Apples produkter [7]. Alexa är utvecklad av Amazon och finns tillgänglig bland annat i företagets smarta enheter [8]. Cortana är Microsofts assistent som finns tillgänglig i Windows 10, mobiltelefoner och smarta högtalare [9].

Antalet användningsområden för digitala assistenter är många och växer ständigt. Populära tjänster som tillhandahålls inkluderar uppspelning av strömmad media som exempelvis Spotify och hämtning av väderprognoser [10]. Digitala assistenter gör det även möjligt att kontrollera smarta enheter i hemmet som lampor och termostater via röststyrning [10]. De nämnda assistenterna erbjuder även möjligheter för tredjeparts mjukvara för att kunna tillföra ytterligare funktionalitet vid behov.

## **1.2 Problem**

Eftersom området för digitala assistenter är relativt nytt behövs det information om hur en röststyrd applikation kan utvecklas. Det behövs också information om vilka begränsningar man kan stöta på som utvecklare vid utveckling av en röststyrd applikation. För att fylla behovet av kunskap inom området behöver följande frågor besvaras:

- Hur ser programmeringsmodellen ut för applikationer som använder ett röstgränssnitt?
- Vilka begränsningar finns vid utveckling av en röststyrd applikation?
- Vilka resurser och ramverk finns att tillgå?

## **1.3 Syfte**

Undersökningen ska förbättra läsarens förmåga att utveckla röststyrda applikationer och samtidigt få kunskap om den bakomliggande teknologin för att kunna



göra välgrundade design- och arkitekturval. Intresset för röststyrda applikationer börjar bli stort och därför behövs kunskap om hur utveckling av dessa applikationer går till. Slutsatserna i rapporten ska hjälpa utvecklare och företag som vill ge sig in på marknaden för röststyrd mjukvara.

## **1.4 Mål**

Målet för projektet var att leverera en fallstudie som innefattar utvecklingsprocessen, design och arkitekturmönster bakom en röststyrd kalenderapplikation till Google Assistant. Fallstudien ger kunskap om hur utvecklingsprocessen kan se ut och föreslår vilka arkitektur- och designval som är lämpliga för en typisk röststyrd applikation. Denna information används för att besvara frågeställningarna som introducerades i sektion 1.2.

## **1.5 Etik och samhällsnytta**

Arbetet kan bidra till att fler röststyrda applikationer utvecklas till digitala assistenter vilket kan vara av samhällsnytta. Människor med synfel eller andra handikapp kan ha svårigheter med teknik som kräver ögon eller händer för att användas. Dessa personer kan istället ha nytta av röststyrda applikationer i sin vardag [11]. Röststyrda applikationer kan öppna nya möjligheter för personer som förlitar sig på hand- och ögonfri interaktion såsom programutveckling via röststyrning, vilket skulle innebära en förbättring i deras arbetsmiljö.

Personlig integritet är enligt FN en fundamental mänsklig rättighet [12]. Det är därför viktigt att den personliga integriteten respekteras, även på internet. Problem kan uppstå när en smart högtalare alltid måste lyssna på sin omgivning för att kunna uppfatta användarens röstkommandon. Om inte denna aspekt av assistenten är säker kan den personliga integriteten kränkas genom att personlig information sprids till tredje part [13]. Detta kan både vara ett brott mot mänskliga rättigheter och lagstiftning om olovlig avlyssning [14].

## **1.6 Metod**

Projektet använder en kvalitativ forskningsmetod i form av en fallstudie som strategi för att besvara frågeställningen. Syftet med en fallstudie är att ge djupgående kunskaper om ett specifikt ämne och passar därför bra tillsammans med en kvalitativ forskningsmetod. Slutsatser kommer att dras induktivt genom att granska data och erfarenheter som samlats under projektets gång [15].

I första steget genomfördes en litteraturstudie om hur digitala assistenter fungerar och vilka olika assistenter det finns. Utifrån litteraturstudien valdes den digitala assistenten som passade bäst till utvecklingen av den röststyrda applikationen som fokus för arbetet. Den valda assistenten användes sedan i fallstudien för att skapa en röststyrd applikation. Funktionen som applikationen uppfyller är att söka efter aktiviteter i en persons digitala kalender genom att ta emot ett röstkommando och ge tillbaka ett röstbaserat svar. Utifrån fallstudien producerades resultat som var underlaget för att besvara frågeställningarna som introducerades i sektion 1.2.

## **1.7 Avgränsningar**

Utvecklingsverktygen som används för att skapa röststyrda applikationer skiljer sig åt för de olika digitala assistenterna vilket medför att avgränsningar måste göras. Därför utvecklades applikationen bara till en digital assistent för att inte arbetet skulle avvika från projektets tidsplan.

## **1.8 Disposition**

I kapitel 2 ges bakgrundsinformation som är nödvändig för arbetet. I kapitel 3 förklaras vilka metoder som användes och hur arbetet utfördes. I kapitel 4 beskrivs utvecklingen av den röststyrda applikationen. I kapitel 5 redogörs projektets resultat. I kapitel 6 dras slutsatser och möjligheter för framtida arbete diskuteras.

## 2 Digitala assistenter och bakomliggande teknologi

Detta kapitel behandlar teknologin bakom digitala assistenter och vilka möjliga verktyg man kan använda för att bygga en röststyrd applikation till Google Assistant. Eftersom digitala assistenter använder sig av Natural Language Processing (NLP) är det viktigt att gå igenom vad det är och hur det fungerar. Men eftersom NLP är en gren av artificiell intelligens (AI) är det viktigt att först introducera AI och hur det är kopplat till digitala assistenter. Därför behandlar första sektionen (2.1) AI och sedan behandlas NLP i sektion 2.2. Sektion 2.3 handlar om Google Assistant och hur den fungerar. Sektion 2.4 ger information om Alexa, Siri och Cortana. Sektion 2.5 behandlar Google Cloud Platform. Sektion 2.6 handlar om vilka verktyg som finns för att utveckla funktionalitet till Google Assistant och beskriver även hur Natural Language Understanding används i assistenten. Sektion 2.7 ger en övergripande bild av systemarkitekturen för röststyrda applikationer. Sektion 2.8 beskriver olika verktyg som användes i detta projekt. Sektion 2.9 beskriver vad ett konversations- och arkitekturmönster är och hur dessa används. Sektion 2.10 beskriver agil projektmetodik och prioriteringstekniken MoSCoW. Sektion 2.11 tar upp relaterat arbete och hur detta arbete skiljer sig från dem.

### 2.1 Artificiell intelligens

Artificiell intelligens (AI) är en metod för att skapa en dator, en datorstyrd robot eller mjukvara som tänker intelligent på ett sätt som liknar det mänskliga sinnet [16][17, p. 1-5]. AI uppnås genom att studera mönstren i den mänskliga hjärnan och genom att analysera den kognitiva processen. Resultatet av dessa studier utvecklar intelligenta mjukvaror och system [17, p. 3]. Ett datorprogram måste kunna räkna, planera och lära sig för att uppnå artificiell intelligens.

Nedan följer en lista över viktiga ämnen inom artificiell intelligens:

- *Tankegång* (Reasoning) handlar om att använda fakta för att komma fram till logiska slutsatser [18, p. 102]. Ett enkelt exempel är om AI-systemet får två bitar av information där den första är "Elefanterna är gråa" och den andra

är ”Kalle är en elefant”, så kan systemet härleda att Kalle är grå.

- *Planering* (Planning) handlar om beslutsfattandet som utförs av intelligenta varelser som robotar, människor eller datorprogram när man försöker uppnå ett visst mål. Det handlar om att välja en sekvens av handlingar för att målet ska uppnås [18, p. 195-196].
- *Natural Language Processing* (NLP) hjälper datorn att tolka och manipulera mänskligt språk [19]. Naturliga språk som engelska, indiska, tyska, franska har ingen formulerad struktur och fortsätter att utvecklas. NLP är ett pågående försök att fånga alla detaljer från de naturliga språken. Mer information om NLP finns senare i kapitlet (sektion 2.2) eftersom det är en viktig del av digitala assistenter.
- *Maskininlärning* (Machine learning eller ML) ger mjukvarusystem möjlighet att automatiskt lära sig och förbättras utan att behöva programmera dem till det. ML fokuserar på utveckling av mjukvara som kan komma åt data och använda det för att lära upp sig själv. [20, p. 4-5]
- *Uppfattning* (Perception) är en term för tekniker som simulerar de sätt som människor uppfattar världen runt dem. Varje typ av teknik som simulerar någon mänsklig känsla är maskinuppfattning, oavsett om det är syn, hörsel, eller beröring. [17, p. 928-929]

Eftersom artificiell intelligens är ett stort område är det viktigt att förstå vilka system eller enheter som verkligen klassas som AI och vilka som påstås vara det. Därför har två kategorier definierats för att skilja mellan dem.

### 2.1.1 Svag AI

*Svag AI* (narrow AI) är en kategori där de flesta nuvarande AI-system ligger i. Detta betyder att ett AI-system endast är kunnigt inom ett specifikt område och kan inte hantera andra områden om den inte är programmerad till att göra det [21]. Alla kända digitala assistenter som nämndes i kapitel 1 anses vara artificiellt intelligenta men de faller ändå i den svaga kategorin eftersom de är programmerade till att utföra specifika uppgifter och kan inte tänka för

sig själva. Till exempel om man ber Google Assistant att slå på lamporna så förstår assistenten vissa nyckelord som "Lampor" och "På". Assistenten utför sedan kommandot genom att tända lamporna. Detta kan upplevas som mänskligt beteende men Google Assistant följer bara sin fördefinierade programmeringssekvens. I slutändan förstår Google Assistant inte nödvändigtvis betydelsen av vad användaren sa.

### 2.1.2 Stark AI

Den andra kategorin kallas för *stark AI* (True AI) och är en term som används för att beskriva en viss tankegång för utveckling av artificiell intelligens. Stark AI:s mål är att utveckla artificiell intelligens till den punkt där maskinens intellektuella förmåga är funktionellt lika med eller överstiger en människas [22]. För att en maskin ska uppnå stark AI måste den uppnå alla ovan nämnda ämnen och kunna kombinera dem för att uppnå ett mål.

Stark AI finns för närvarande inte. Vissa experter anser att det kan komma inom de närmsta 20 åren [23]. Andra förutspår mer försiktigt att tekniken kan utvecklas inom nästa århundrade eller att utvecklingen av stark AI kanske inte är möjlig alls [24]. I dagsläget ligger fokus mer på svag än stark AI då tekniken är mer användbar inom specifika områden.

## 2.2 Natural Language Processing

Natural Language Processing (NLP) hjälper datorer att tolka, förstå, och manipulera mänskligt språk. NLP använder sig av datorvetenskap och lingvistik för att skapa en översättning mellan mänsklig kommunikation och datorspråk. [19]

Digitala assistenter använder NLP för att tolka mänskligt språk men tekniken används även i enkla vardagliga applikationer. Enligt artikeln *8 natural language processing (NLP) examples you use every day* används NLP i chattjänster, sökmotorer och onlineformulär för att komplettera, förutse eller korrigera ord som användaren skriver [25]. Googles sökmotor använder olika NLP-algoritmer

för att snabbt ge användaren relevanta artiklar. Sökmotorn använder dubblett- och spamdetektion för att filtrera bort hemsidor eller artiklar som inte är relevanta till användarens sökning. En annan användning för NLP är spamdetektion som används i de flesta e-postklienter. Med tekniken kan inkommande e-post klassificeras i olika kategorier såsom viktiga-, sociala-, reklam- eller spammeddelanden. [25]

Det finns två olika delar av NLP som definierar hur tekniken fungerar, men de använder olika metoder för att tolka eller manipulera mänskligt språk. Den första är *Natural Language Understanding* och den andra är *Natural Language Generation*.

### **2.2.1 Natural Language Understanding**

Natural Language Understanding (NLU) är den delen som tolkar naturligt språk. Målet med NLU är att göra det möjligt för mjukvarusystem att förstå mänskligt språk i textform [26]. NLU är ett stort område som består av många delar såsom *relation extraction*, *semantic parsing* och *sentiment analysis* [27]. *Relation extraction* handlar om att extrahera objekt och dess relationer från en mening [28, p. 17]. Ett exempel kan vara att NLU-systemet tolkar meningen "Kalle är anställd på Google" och identifierar "Kalle" som subjekt, "anställd" som relation och "Google" som objekt. *Semantic parsing* innebär att översätta det naturliga språket till ett strukturerat format som en maskin kan använda [29, p. 7]. Till exempel kan en mening omvandlas till en databasfråga. *Sentiment analysis* undersöker en text för att hitta åsikter och känslor [30]. Till exempel kan en filmrecension analyseras för att hitta författarens åsikter om filmen och identifiera om recensionen var positiv eller negativ.

Taligenkänning är inte en del av NLU men är ett viktigt hjälpmedel för att översätta talat språk till läsbar text. Taligenkänning kan utföras med olika metoder som bland annat en Dold Markovmodell (HMM) och Artificiella Neurala Nätverk (ANN) [31]. I de flesta taligenkänningssystem används HMM men Google använder ANN [31][32].

### 2.2.2 Natural Language Generation

Natural Language Generation (NLG) används för att översätta datorspråk till naturligt språk [33]. Detta kan ses som motsatsen till NLU då NLG fattar beslut om hur en datastruktur ska översättas till läsbar text. Att utveckla NLG-system som kan översätta datorspråk till text i muntlig/skriftlig form är inte så lätt som man kan tro. Systemet behöver känna till flera domäner inom det naturliga språket såsom kunskap om domänen (få ut rätt information till användaren), kunskap om språket (lexikon, grammatik, semantik) och retorisk kunskap (anpassa text/tal efter miljön) [33].

Översättning av data till naturligt språk utförs genom att följa fyra steg [33]. De fyra stegen är bara en sammanfattning av hela översättningsprocessen och går inte in i de specifika detaljerna. I första steget identifierar NLG-systemet målet med översättningen [33]. Ett exempel kan vara att NLG-systemet ska ge dagens väderprognos till användaren, vilket är målet som systemet ska utföra. I andra steget samlar NLG-systemet ihop relevant data för att kunna bygga upp en mening [33]. Relevant data kan vara temperatur, vindhastighet och nederbörd. I tredje steget används lexikon och grammatiska regler för att strukturera upp meningen med den relevanta data. Med detta menas att data läggs på rätt plats i meningen, extra ord läggs till i meningen och vissa ord böjs för att meningen ska ha rätt syntaktisk struktur. I sista steget bestäms om översättningen/meningen ska vara i skriftlig eller muntlig form [33]. I detta steg läggs skiljetecken i meningen om det är i skriftlig form eller bestämmer hur meningen ska uttalas om det är i muntlig form.

## 2.3 Google Assistant

Google Assistant utvecklades som namnet antyder av Google. Lanseringen skedde år 2016 för mobila enheter och smarta högtalare (bild på en smart högtalare från Google hittas i bilaga C) [34]. Assistenten kan även köras på enheter som kallas för *Smart Display* [35]. Assistenten designades för att kunna hantera tvåvägskommunikation med användaren genom ett röstgränssnitt. Den första versionen hade enbart stöd för kommunikation på engelska, vilket sedan har

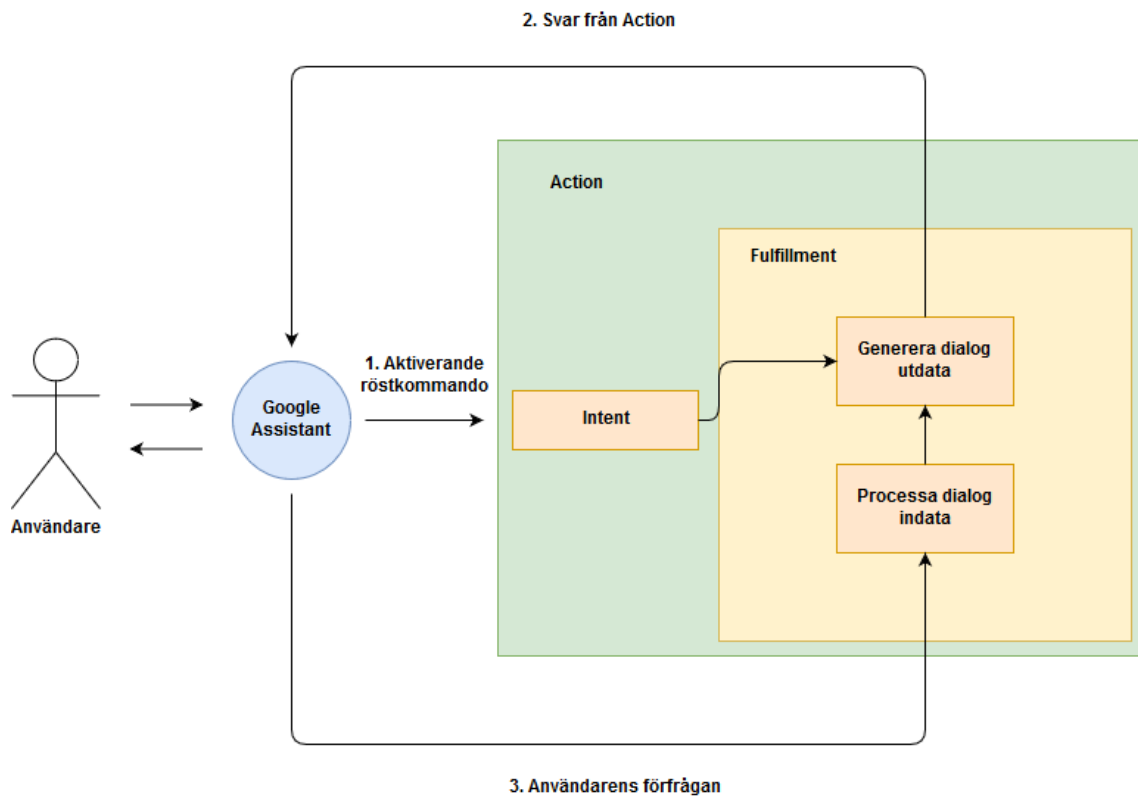
utökats till att i dagsläget stödja 19 olika språk, varav svenska är ett av dem [36].

Google Assistant Software Development Kit (Google Assistant SDK) är en uppsättning av verktyg som används för att köra röststyrd mjukvara på olika hårdvaruplattformar. Hotword Detection är en funktion som ingår i denna SDK. Funktionen gör att den smarta enheten kan lyssna efter specifika fraser och sedan ”vakna upp” när dessa fraser uttalas. Frasen ”OK Google” är standardfrasen för att väcka Google Assistant. Natural Language Understanding (NLU) ingår också i Google Assistant SDK för att tolka användarens kommando. [37]

*Actions on Google* är en utvecklarplattform som används för att skapa applikationer till Google Assistant genom att utvidga assistentens funktionalitet [38]. Tre konstruktioner används för att implementera funktionalitet i Google Assistant. Dessa är intents, fulfillments och actions. *Intents* är den uppgift som användaren vill att assistenten ska utföra, till exempel att spela musik. En intent representeras av en unik identifierare och röstkommando som används för att aktivera den specifika intenten. *Fulfillments* är den bakomliggande logiken som utför uppgiften som specificeras i en intent. *Actions* är den interaktion som byggs till assistenten så att den kan hantera en specifik intent och utföra uppgiften med hjälp av korresponderande fulfillment. [38]

Det finns två typer av actions, *smart home actions* och *conversational actions*. *Smart home actions* innebär att assistenten tar emot ett röstkommando och utför en handling som svar [38]. Detta kan innebära att spela musik, styra smarta lampor eller styrning av andra smarta enheter. *Conversational actions* är de funktioner som kräver tvåvägskommunikation för att utföras [38]. Figur 2.1 visar hur flödet kan se ut för en conversational action med tillhörande intent och fulfillment.





Figur 2.1: Actions On Google. Bilden är skapad av författarna.

Användaren i figur 2.1 inleder kommunikationen genom att "väcka" Google Assistant med frasen "Ok Google". Assistenten svarar med en hälsningfras och lyssnar sedan på användarens röstkommando som sedan skickas till Googles molntjänst. Röstkommandot tolkas i molnet och kopplas till ett specifikt intent som aktiverar en action. Ett fulfillment genererar sedan ett svar som skickas tillbaka till Google Assistant och spelas upp för användaren. Användarens nästa röstkommando skickas sedan direkt till samma fulfillment i molnet som hanterar indatan och generar ny utdata som skickas tillbaka till användaren. Det sista steget kan upprepas flera gånger om konversationen mellan användaren och Google Assistant fortsätter.

## 2.4 Alexa, Siri och Cortana

Alexa är Amazons digitala assistent som lanserades år 2014 [39][8]. Alexa finns tillgänglig i många olika typer av produkter som till exempel den smarta högtalaren Amazon Echo [40]. Genom att använda Alexa Voice Service Device

SDK kan Alexa integreras med produkter som har en mikrofon och högtalare [41]. Assistentens funktionalitet utgörs av så kallade Skills (Google actions motsvarighet). Det är möjligt att utöka funktionaliteten hos Alexa genom att utveckla nya Skills. I dagsläget stödjer Alexa Skills på engelska, franska, tyska, italienska, japanska och spanska [42].

Siri är Apples digitala assistent som lanserades år 2011 [43]. En skillnad mellan Siri och tidigare nämnda assistenter är att Siri endast finns tillgänglig på Apples produkter utan möjlighet för integration med hårdvara från andra företag. Utveckling av applikationer som använder Siri görs med hjälp av SiriKit. Genom att koppla en intent till en applikation kan man tillåta Siri att hantera någon funktion i applikationen med användarens röstkommando [44]. Vid utveckling av applikationer till den smarta högtalaren HomePod [45] så är möjligheterna sämre än för till exempel Alexa och Google Assistant. Högtalaren kan endast agera som en anslutningspunkt för applikationer som körs på en iPhone eller iPad [44]. Detta kräver då att det finns en iPhone eller iPad i närheten vilket kan anses vara en begränsande faktor i funktionaliteten. Siri stödjer kommunikation på 21 olika språk där svenska är inkluderat [46].

Cortana är Microsofts digitala assistent som lanserades år 2014 och finns tillgänglig till Windows, mobiltelefoner och smarta högtalare [47][48]. Cortana kan integreras med produkter genom Cortana Devices SDK [49]. Det är även möjligt att utveckla ny funktionalitet till assistenten. Detta görs med Cortana Skills Kit som används både för att utveckla funktionerna och sedan även distribuera dem till användare. Språkstödet för Cortana är i dagsläget 14 olika språk (ej svenska), men vid utveckling av nya funktioner kan endast engelska användas [50].

## **2.5 Google Cloud Platform**

Google Cloud Platform (GCP) är en samling av hårdvaruresurser som erbjuds av Google till allmänheten [51]. Dessa resurser är bland annat datorer, diskutrymme och nätverkslösningar. Användare kan komma åt resurserna genom olika typer av nätverksbaserade mjukvaruprodukter som tillhör Google.

Dessa typer av produkter kallas för molntjänster (cloud computing på engelska) [51]. Google erbjuder molntjänster inom kategorierna maskininlärning, säkerhet, mjukvaruutveckling, databashantering och Big data [52][53]. GCP erbjuder över 50 molntjänster som i dagsläget kan testas gratis i en tidsperiod och sedan fortsätta användas mot betalning [54][55]. För att kunna använda GCP måste först ett google konto<sup>1</sup> och sedan ett GCP projekt<sup>2</sup> skapas. Google Cloud Platform erbjuder även verktyg för utveckling av röststyrda applikationer till Google Assistant. Mer om verktygen förklaras i sektion 2.6.

GCP ger tre olika sätt att interagera med dess molntjänster. De tre sätten är *Google Cloud Platform Console* (webbgränssnitt), *Google Cloud SDK* (kommandoradsgränssnitt) och *Google Cloud Client Libraries* (kodbibliotek) [51].

## 2.6 Verktyg för utveckling av actions till Google Assistant

Utveckling av ny funktionalitet till Google Assistant sker genom att skapa nya actions [38]. Beroende på vilken typ av applikation som ska utvecklas finns olika verktyg tillgängliga för att skapa konversationen mellan användaren och den digitala assistenten. Dessa är mallar (templates) [56], Actions SDK [57] och Dialogflow [58]. Det förstnämnda kan användas för att skapa triviala applikationer som till exempel frågesport. Mallar kräver väldigt lite teknisk kunskap eftersom utvecklaren endast behöver fylla i ett kalkylark bestående av frågor och svar [56]. Utvecklingsmöjligheterna vid användandet av mallar är väldigt begränsade eftersom det bara går att skapa applikationer som det finns färdiga mallar för.

Actions SDK är en uppsättning av verktyg som ger utvecklare direkt åtkomst till den text som har genererats från användarens röstkommando [57]. Actions SDK innehåller ingen form av NLU vilket gör att det krävs att utvecklaren skriver kod som tolkar texten och genererar ett svar tillbaka till användaren. Om applikationen ska stödja conversational actions där dialogen kan följa många olika vägar blir det väldigt komplicerat att skriva kod för att tolka alla möjliga

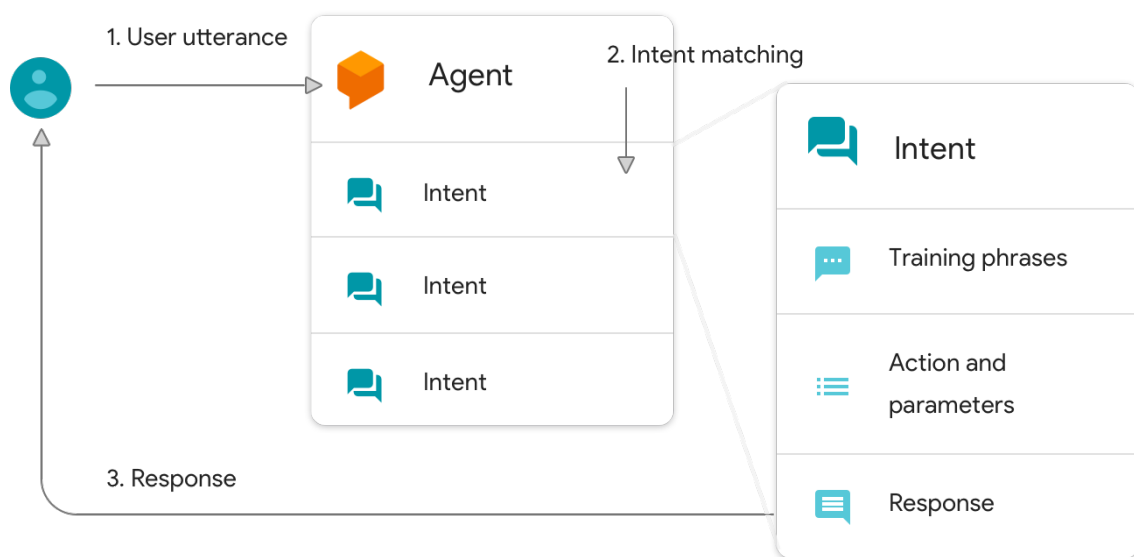
---

<sup>1</sup><https://support.google.com/accounts/answer/27441>

<sup>2</sup><https://cloud.google.com/resource-manager/docs/creating-managing-projects>

röstkommandon från användaren. På grund av detta lämpar sig Actions SDK bäst för enkla applikationer som till exempel en applikation som sätter ett alarm.

Dialogflow är ett företag som ägs av Google och som erbjuder en tjänst för att bygga chattbotar [58]. Tjänsten är en del av Google Cloud Platform [54]. Först och främst är Dialogflow en NLU-motor som är byggd ovanpå Actions SDK för att tolka vad en användare säger efter förutbestämda mönster [59]. Detta eliminerar utvecklarens behov att skapa sin egna NLU-mjukvara som var fallet om bara Actions SDK användes. Detta möjliggör skapandet av mer komplicerade applikationer även för utvecklare utan avancerad kunskap inom NLU. Dialogflow fungerar med hjälp av så kallade agenter. Dessa agenter använder NLU för att förstå talat språk och översätta det till användbar data [60]. Figur 2.2 visar hur Dialogflow hanterar ett röstkommando. Utvecklingen av actions i Dialogflow sker genom ett webbgränssnitt.



Figur 2.2: Dialogflow agent och matchning av intent. Bild av: Dialogflow. Creative Commons Attribution 3.0 [61]

Användarens röstkommando i figur 2.2 tolkas av agenten och jämförs med olika träningsfraser (training phrases i figuren) som utvecklaren har skapat för att kopplas ihop med rätt intent. Agenten extraherar sedan viktiga bitar av information ur röstkommandot och skickar det vidare till den action som är kopplad till det intent som aktiverades (action and parameters i figuren). Dessa bitar av data kallas för entities och kan till exempel vara en tidpunkt eller ett namn

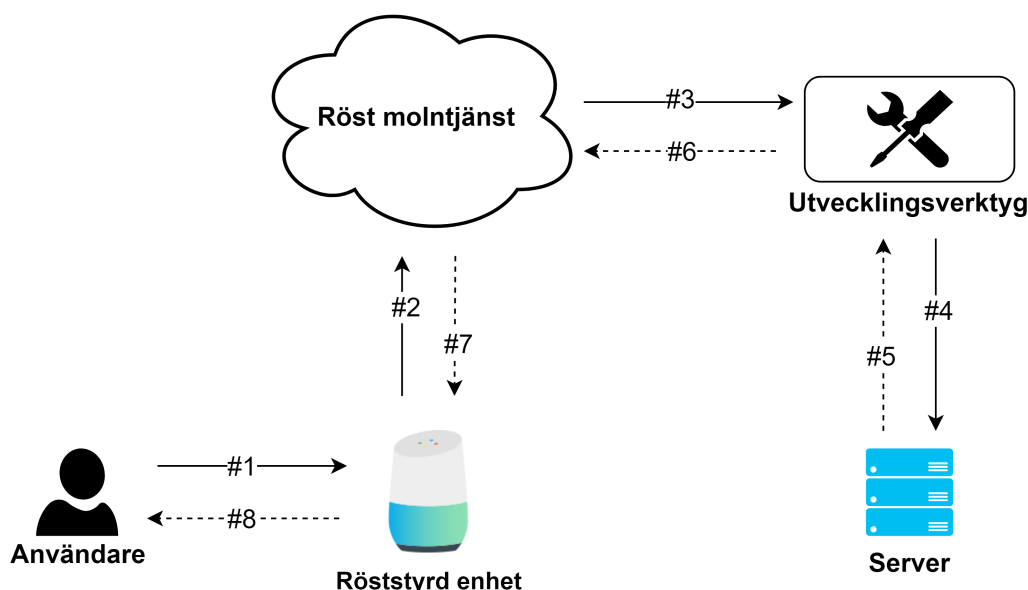
som en specifik action kräver som parameter. En entity består av en eller flera *entity types* och *entity entries*. En entity type är den typ av information som ska extraheras från användarens röstkommando och entity entries är ord eller fraser som är ekvivalenta med typen. Exempelvis om en entity type är frukt kan då olika entries vara äpple och apelsin. Om användaren då säger ordet äpple vet Dialogflow att samtalet handlar om frukt. Om agenten inte hittar den nödvändiga datan i användarens röstkommando skickar den ett svar till användaren och ber om mer information (Response i figuren).

Träningsfraserna används av Dialogflow för att träna upp agenten. Detta görs med hjälp av maskininlärning. Maskininlärning används för att utvidga träningsfraserna för att möjliggöra matchning med så många liknande fraser som möjligt. Exempelvis en träningsfras som ”vad har Johan i sin kalender?” används för att träna agenten att förstå olika variationer av denna fras som ”vad har Johan i sin almanacka?”. Dialogflow åstadkommer detta genom att skapa en dynamisk modell som tar beslut om hur användarens röstkommandon ska hanteras. Algoritmen som modellen använder för att ta dessa beslut är unik eftersom den är baserad på intentkonfigurationen. [62]

Varje intent kan generera svar till användaren men dessa svar är hårdkodade och ofta inte särskilt användbara. Därför används fulfillments för att bygga ut programlogiken vilket möjliggör mer intelligenta svar till användaren. Backend-koden för fulfillments körs på en extern webbserver som sedan anropas med hjälp av en webhook [63]. Utvecklingsprocessen redovisas i större detalj i sektion 4.

## 2.7 Systemarkitektur av en röststyrd applikation

Nedan visas en bild över systemarkitekturen för en röststyrd applikation. Syftet med denna bild är att ge en tydlig överblick över hela systemet för att skapa en förståelse för hur de olika delarna hänger ihop.



Figur 2.3: Övergripande bild av systemarkitekturen. Bilden är skapad av författarna med draw.io [64]

Figur 2.3 visar systemarkitekturen och hur de olika delarna av systemet kommunicerar med varandra. Pilarna i figuren visar händelseförloppet när en applikation körs. Förloppet inleds med att användaren ger ett röstkommando till röststyrda enheten som digitala assistenten körs i. Röstkommandot skickas vidare till digitala assistentens molntjänst där det körs mjukvara för taligenkänning som konverterar talet till text<sup>3</sup>. Denna text skickas sedan vidare till digitala assistentens utvecklingsverktyg som använder NLU för att koppla kommandot med applikationens intent. Utvecklingsverktyget kan vara Dialogflow, Amazon skills kit [65] som tillhör Alexa eller Microsoft Bot Framework [66] som tillhör Cortana. Om den röststyrda applikationen använder externa tjänster skickas kommandot vidare från utvecklingsverktyget till en server som använder de externa tjänsterna. En av de externa tjänsterna kan vara en kalendertjänst som servern kan hämta kalenderhändelser från eller en databas. Efter att servern är klar med de externa tjänsterna byggs ett textbaserat svar och skickas tillbaka till utvecklingsverktyget. Svaret skickas sedan tillbaka till digitala assistentens molntjänst där det konverteras från text till tal. Röstsvaret spelas sedan upp för användaren i röststyrda enheten. Om serverdelen inte fanns med i den

<sup>3</sup>Enheten måste alltid ha internetuppkoppling för att digitala assistenten ska bearbeta användarens kommando, annars ger enheten ett felmeddelande till användaren.

röststyrda applikationen skulle istället ett hårdkodat svar skickas tillbaka från utvecklingsverktyget.

## 2.8 Diverse tekniska verktyg

### Node.js

Node.js är en JavaScript exekveringsmiljö skapad av Ryan Dahl. Miljön är byggd på Chromes V8 JavaScript-motor [67] för att kunna köra JavaScript kod på servernivå [68, p. 3-5]. Node.js möjliggör I/O operationer med JavaScript såsom databasanrop och filsystemhantering. Dessa I/O operationer körs asynkront, vilket betyder att när en I/O operation körs går Node.js direkt till nästa operation istället för att vänta på att den första operationen blir klar.

Node.js erbjuder kodbibliotek som kallas för moduler. Det finns båda inbygga och externa moduler. Inbygga moduler kan man använda direkt utan någon form av installation medan externa moduler kan hämtas genom att använda bibliotekshanteraren Node Package Manager [69].

För att kunna hantera asynkroniska operationer finns det olika lösningar som kan användas i JavaScript, en av dem kallas för Promise. Promise är ett objekt i JavaScript som returnerar det resulterande värdet av en asynkron operation om operationen lyckas, eller ett annat värde (felmeddelande) om operationen misslyckas [70]. När ett Promise objekt skapas är den i ett ”väntande tillstånd”. Detta tillstånd kan antingen gå till ett uppfyllt eller avvisat tillstånd. Om den asynkrona operationen slutförs går Promise objektet till det uppfyllda tillståndet och ger ut värdet som operationen returnerar. Men om operationen misslyckas går objektet till det avvisade tillståndet och ger ut ett värde som beskriver misslyckandet (felmeddelande). I figur 2.4 visas ett enkelt kodexempel av ett Promise.

```
function foo(){
  return new Promise((resolve, reject) =>{
    // Utför någon asynkron operation som antingen kallar på:
    resolve(någotVärde); //Uppfylld
    // eller
    reject("Operationen misslyckades"); //Avvisad
  });
}

foo().then((någotVärde) =>{
  // Gör något med värdet
}).catch((felmeddelande) =>{
  // Hantera meddelandet
});
```

Figur 2.4: Kodexempel över hur Promises används. Bilden är skapad av författarna.

Figur 2.4 visar kodexempel på hur Promises fungerar. I funktionen `foo()` skapas och returneras ett nytt Promise objekt. I objektet definieras en anonym funktion med två parametrar, `resolve` och `reject`. Dessa parametrar används senare som funktioner för skicka tillbaka värdet som ska returneras av funktionen `foo()`. I den anonyma funktionen utförs någon typ av asynkron operation och kallar sedan antingen på `resolve` eller `reject`. Om `resolve` kallas betyder det att operationen lyckades och något värde ska returneras i funktionen `foo()`. Om operationen misslyckades kallas istället `reject` med ett felmeddelande som beskriver felet. För att hämta värdet som returneras från funktionen `foo()` används `then()` och `catch()`. Med funktionen `then()` kan man ta emot värdet som `resolve` hade som parameter. Funktionen `catch()` tar emot meddelandet som `reject` hade som parameter.

## Firestore

Firestore är en Backend-as-a-Service (BaaS) som erbjuder webbtjänster såsom webbhotell, autentisering och databasåtkomst [71]. Firestore ägs av Google och är en del av Google Cloud Platform. För att komma åt tjänsterna i Firestore används Firestore CLI, vilket administrerar Firestore-projekt för att tillhandahålla



en mängd olika verktyg såsom hantering, visning och distribuering [72].

Firebase erbjuder en molnbaserad lösning för databashantering kallad för Realtime Database. Realtime Database är en NoSQL molnbaserad databas som synkroniserar data i realtid [73]. Detta betyder att alla klienter som är kopplade med databasen automatiskt får uppdateringar med de senaste data. Med en NoSQL databas menas att databasen har en annan datastruktur än tabellformat. Realtime Database använder sig av JSON-format i form av en trädstruktur för att lagra data.

En annan tjänst som Firebase erbjuder heter Cloud Functions. Cloud Functions används för att exekvera JavaScript funktioner (i Node.js miljön) som kan skicka eller ta emot HTTPS anrop från en till flera klientenheter [74][75]. Med andra ord exekveras JavaScript funktioner i Firebase servern och skickar data till klienter när en händelse sker eller tar emot data när en klient skickar data till servern. Cloud Functions kan användas för att ta emot webhook anrop från Dialogflow.

## **Google Calendar och Google Calendar API**

Google har en webbaserad kalendertjänst som kallas för Google Calendar. I tjänsten kan en användare skapa kalendrar direkt via deras webbgränssnitt. I en kalender kan man skapa, redigera eller ta bort kalenderhändelser. Varje kalender har ett unikt kalender-ID för att identifiera kalendern. För att kunna hantera olika kalendrar utanför Googles webbgränssnitt kan man använda sig Google Calendar API. Google Calendar API är en RESTful webbtjänst [76] som kan nås via Hypertext Transfer Protocol (HTTP) anrop [77]. Med Calendar API:et kan man visa, skapa och ändra kalenderhändelser och utföra andra kalenderspecifika funktioner direkt från en extern applikation [77].

## **Webhook**

Webhook är en HTTP callback som skickar data från en nod till en annan nod när en händelse sker i den första noden [78]. HTTP callback betyder att en

avisering utförs med hjälp av HTTP POST när en händelse sker [78]. Om en webbapplikation vill veta om ny data har kommit till en server behöver den bara vänta tills servern skickar data med HTTP POST till en URL som applikationen använder för att ta emot datan. På sådant sätt behöver applikationen inte utföra en HTTP-förfrågan till servern för att kolla om något har ändrats.

Dialogflow använder webhooks på samma sätt för att skicka data till en server [63]. När en intent matchas med användarens kommando skickas kommandot vidare till servern genom att utföra en HTTP POST förfrågan till en URL som servern lyssnar på. Servern tar sedan emot kommandot och hanterar det.

## **Github**

Github är en webbtjänst som erbjuder versionskontroll med hjälp av versionshanteringsystemet Git. Github är också en samarbetsplattform där programutvecklare kan dela kod med varandra. Github ägs av Microsoft och är i dagsläget världens ledande webbhotell för källkod med över 31 miljoner registrerade användare [79][80].

## **Google Drive**

Google Drive är en molnlagringstjänst från Google. Med tjänsten blir det möjligt att skapa eller ladda upp dokument som lagras i molnet. Dessa dokument kan sedan delas och redigeras av flera användare samtidigt. Google Drive erbjuder verktyg för att redigera textdokument, kalkylark och presentationer. Tjänsten erbjuder en begränsad mängd gratis lagringsutrymme för filer som kan utökas genom betalning. [81]

## **2.9 Mönster inom mjukvaruutveckling**

Ett mönster inom mjukvaruutveckling är en generell och beprövad lösning som kan användas för att lösa vanligt uppstående problem. Arkitekturmönster beskriver strukturen av ett program genom att visa hur olika komponenter

av systemet hänger ihop och kommunicerar med varandra [82]. Dessa systemkomponenter kan vara en samling av flera klasser och moduler. Några exempel på arkitekturmönster är Model-View-Controller (MVC) och Layer Architectural Pattern [83]. Designmönster är en annan typ av mönster som används inom mjukvaruutveckling. Designmönster är starkare kopplat till objekt-orienterad programmering och beskriver systemet på en mer detaljerad nivå med hjälp av klassdiagram [84]. Exempel på designmönster är Observer [85, p. 326], Strategy [85, p. 349], Factory [85, p. 121] och Singleton [85, p. 144].

Ett mönster består ofta av ett problem, en lösning och konsekvenserna av att använda mönstret. Problemet består av en beskrivning av vilket problem som mönstret försöker lösa. Lösningen är en beskrivning av hur problemet ska lösas. Konsekvenserna är resultaten av att använda lösningen, både för- och nackdelar [85, p. 12-13].

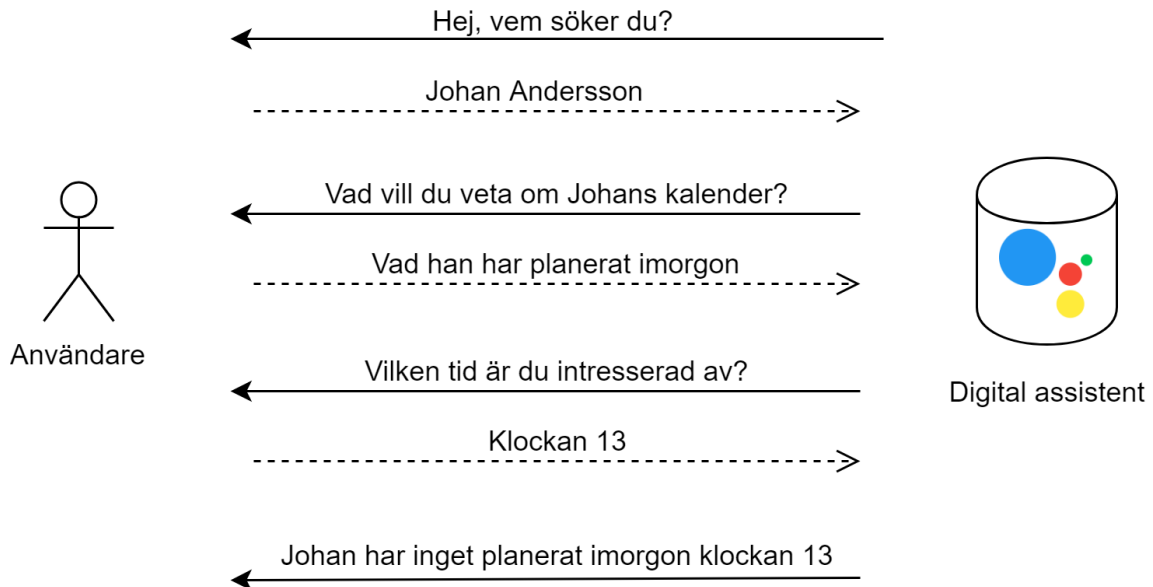
### **2.9.1 Konversationsmönster för röstgränssnitt**

När man utvecklar en röststyrd applikation är det viktigt att veta hur applikationens röstgränssnitt ska modelleras. I denna sektion presenteras 3 olika konversationsmönster som kan användas i en röststyrd applikation. Dessa konversationsmönster är tagna från artikeln *Uncovering Voice UI Design Patterns* [86].

#### **Konversationsmönster 1**

Det första konversationsmönstret går ut på att samla tillräckligt med information ett steg i taget från användaren tills användarbegäran kan utföras. Den digitala assistenten ställer en fråga i taget till användaren och sätter ihop alla svar till en komplett användarbegäran. I figur 2.5 visas ett scenario på hur konversationen skulle gå till mellan en användare och digital assistent. Konversationsmönstret skulle passa till röststyrda applikationer som behöver flera viktiga bitar av information från användaren utan att användaren behöver ge all denna information till assistenten på en gång. Användaren behöver inte heller klura ut vad hen ska svara eftersom assistenten ställer korta och tydliga frågor

som användaren enkelt kan svara på. Nackdelen med konversationsmönstret är att användaren inte kan uppge all information direkt med ett kommando utan måste vänta på frågorna från assistenten.



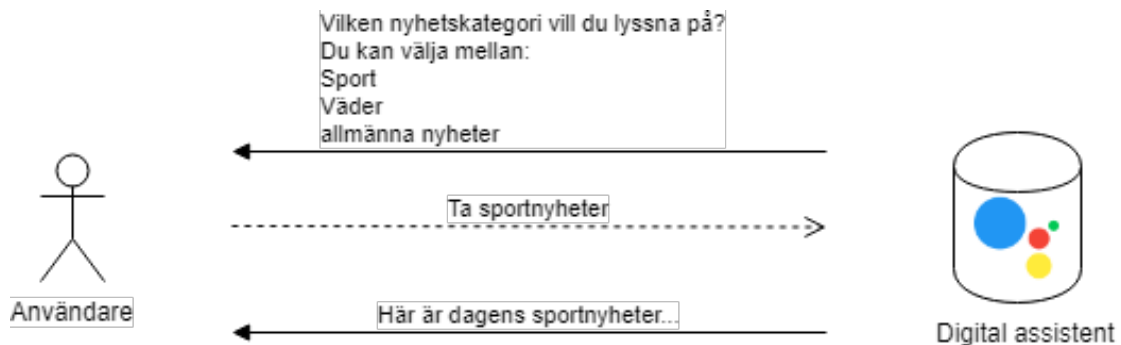
Figur 2.5: Konversation mellan användare och digital assistent om kalenderhändelser. Bilden är skapad av författarna med draw.io [64].

Figur 2.5 visar hur konversationsmönstret används i en konversation om kalenderhändelser. Assistenten börjar med att fråga vilken person som användaren söker och användaren uppger därefter ett namn. Assistenten fortsätter med att fråga vad användaren vill veta om den specifika personens kalender och vilken tidpunkt som är av intresse. Användaren svarar i tur och ordning på frågorna som till slut ger assistenten svaret som användaren sökte.

## Konversationsmönster 2

Det andra konversationsmönstret fokuserar på att assistenten presenterar alla alternativ för användaren. Man kan tänka sig att användaren får välja något från en navigeringsmeny. Assistenten börjar med att berätta vad applikationen kan göra och sedan presenterar ett antal val som användaren kan välja mellan. För användaren blir det då enkelt att veta vilka olika svar som kan ges. I figur 2.6 visas ett scenario där konversationsmönstret används.

En nackdel med konversationsmönstret är att det blir svårt för användaren att hålla koll på alla val som applikationen presenterar om det är många val. En lösning skulle vara att dela upp alla val i olika kategorier som användaren kan välja mellan.



Figur 2.6: Konversation mellan användare och digital assistent om nyheter. Bilden är skapad av författarna med draw.io [64].

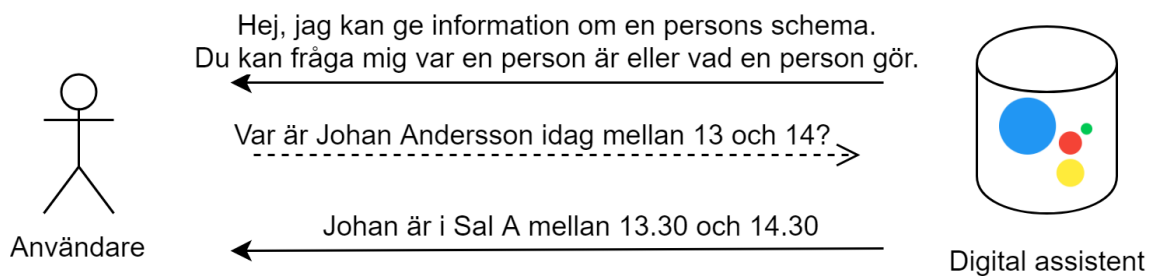
Figur 2.6 visar en röststyrd applikation som kan läsa upp nyheter för användaren. Först frågar assistenten vilken nyhetskategori användaren vill ha och presenterar sedan alla valbara kategorier. Användaren väljer en av kategorierna och sedan läser assistenten upp dagens nyheter i den kategorin.

### Konversationsmönster 3

I det tredje konversationsmönstret uppger den digitala assistenten vad applikationen kan göra men ger inga tydliga alternativ över vad användaren kan välja mellan. Applikationen ska därför kunna ta emot flera varianter av den fråga användaren ställer. Konversationsmönstret är lämpligt för applikationer som erbjuder många val. Detta kräver att användaren vet lite om hur applikationen fungerar och vilka frågor eller kommandon man kan ge till applikationen. Det är därför viktigt att applikationen kan ta emot olika variationer av kommandon som en användare kan tänkas ge.

För en ny användare kan det vara svårt att förstå hur en röststyrd applikation med konversationsmönstret fungerar. Det man kan göra är att ge små förslag över vad användaren kan fråga om, vilket visas i figur 2.7. Användaren behöver även testa

sig fram över vad applikationen kan ta emot om inga tydliga instruktioner ges till användaren.



Figur 2.7: Konversation mellan användare och digital assistent om flygbokning. Bilden är skapad av författarna med draw.io [64].

I figur 2.7 visas en röststyrd kalenderapplikation. Konversationen är utformad efter konversationsmönster 3. Först berättar assistenten för användaren vad som kan utföras och sedan tas ett kommando emot från användaren. I konversationen kan vi se att användaren utför sin förfrågan med bara ett kommando vilket assistenten sedan kan utföra eftersom all nödvändig information finns i kommandot.

## 2.10 Teori om projektmetoder

### Agila projektmetoden

Agil systemutveckling är en samling av iterativa metoder för mjukvaruutveckling [87]. Den Agila metoden hjälper utvecklare i ett projekt att leverera en produkt snabbare och sedan förbättra den med hjälp av feedback från kunden [88]. Istället för att satsa allt på en enda lansering av produkten levererar man istället små men fungerande funktioner. Krav, planering och resultat utvärderas kontinuerligt så att projektgruppen har en naturlig mekanism för att snabbt reagera på förändringar [89]. Genom att arbeta agilt kan en projektgrupp flexibelt reagera och ändra sina planer beroende på förändringar på marknaden eller feedback från kunden utan att behöva ändra ett års värde av planer.

## MoSCoW

*MoSCoW* är en prioriteringsteknik skapad av Dai Clegg år 1994 [90]. Metoden handlar om hur man organiserar alla krav i ett projekt genom att definiera vilka av de listade kraven som är viktigast och måste uppfyllas först för att ha en större chans till ett lyckat projekt.

Termen *MoSCoW* fokuserar på fyra kategorier som kraven kan falla under. Dessa är: *måste ha*, *bör ha*, *kan ha* och *kommer inte ha* [90]. I *måste ha* kategorin ska krav som måste uppfyllas vara i. Dessa krav är väsentliga och om de inte uppfylls är det en indikation av ett misslyckat projekt. I *bör ha* faller önskade krav som har hög prioritet men som inte är nödvändiga för en användbar slutprodukt. *Kan ha* kategorin tar emot krav som skulle kunna utföras om det finns tid kvar. I *kommer inte ha* kategorin finns krav som projektgruppen har kommit överens om att de inte kommer kunna leverera på grund av kunskaps-, tids- eller budgetbrist.

## 2.11 Relaterat Arbete

Det finns flera artiklar och guider som förklarar hur man kan skapa applikationer till digitala assistenter. En bra resurs som användes för att komma igång med skapandet av applikationen var en guide som Google har skapat. Denna guide är uppdelad i tre delar där den första delen är grundläggande kunskap om Dialogflow [91]. Den andra delen behandlar hantering av fulfillment [92] och den tredje delen går igenom extra funktionalitet som kan implementeras i en röststyrd applikation [93]. Denna guide användes i inledningen av utvecklingsfasen för att få kunskap om hur en applikation till Google Assistant kan utvecklas.

Två relaterade arbeten användes i projektet. Dessa är artikeln *Uncovering Voice UI Design Patterns* [86] skriven av Joe Kappes och Jiwon Paik, och forskningsarbetet *Voice User Interface Design Patterns* [94] skrivet av Dirk Schnelle och Fernando Lyardet. Artikeln går igenom fem olika konversationsmönster man kan använda i ett röstgränssnitt för digitala assistenter. Forskningsarbetet tar upp åtta konversationsmönster som kan användas i flera olika typer av röstgränssnitt (ej specifikt till digitala assistenter). Forskningsarbetet tar även upp styrkor och

svagheter för varje konversationsmönster.

Arbetet i denna rapport använder information från ovan nämnda resurser kombinerat med erfarenheten som fallstudien ger för att ge en helhetsbild av området. Denna helhetsbild inkluderar arkitektur- och konversationsmönster, information om bakomliggande teknologi och utvecklingsprocess.

Genom enkla googlesökningar kan man hitta andra artiklar och guider för hur man skapar en röststyrd applikation för Google Assistant. Dessa är oftast kortfattade och ger endast en överblick av ämnet. Det finns även onlinekurser som går igenom hur man skapar applikationer till digitala assistenter. Dessa resurser saknar ofta den detaljerade inblicken som en fallstudie genererar. Information om digitala assistenter och dess bakomliggande teknologi brukar inte heller finnas med i dessa översiktliga guider. Arbete som beskriver hur utvecklingen kan se ut för andra digitala assistenter än Google Assistant är inte relevanta för arbetet eftersom Google Assistant är fokus för rapporten. En tabell över några artiklar som hittades med googlesökning finns i bilaga B.



### **3 Undersöknings- och projektmetoder**

I detta kapitel beskrivs vilka metoder som användes under projektets gång och hur de användes. Det finns två typer av metoder som användes i projektet, undersökningsmetoder och projektmetoder. Undersökningsmetoderna fokuserar på hur data samlas in, tolkas och analyseras från ett specifikt fall. Projektmetoderna beskriver hur projektet utfördes. Sektion 3.1 beskriver vilka undersökningsmetoder som användes i detta projekt. Sektion 3.2 beskriver vilka resultat som krävs för att besvara frågeställningarna och hur dessa ska tas fram. Sektion 3.3 ger en beskrivning av fallstudiens utformning. Sektion 3.4 beskriver valen av teknologi som gjordes i projektet och vilka kriterier valen baserades på. Sektion 3.5 beskriver vilka verktyg som användes för datahantering och hur informationssökningen utfördes. Sektion 3.6 beskriver vilken projektmetod som användes i projektet och hur den användes. Sektion 3.7 går igenom kravspecifikationen och beskriver projektets iterationer.

#### **3.1 Undersökningsmetod**

I denna sektion presenteras undersökningsmetoder som är relevanta för arbetet.

##### **3.1.1 Kvantitativa och kvalitativa metoder**

Det finns två kategorier av forskningsmetoder, kvantitativa och kvalitativa. En kvantitativ undersökning använder experiment och olika former av mätningar för att testa hypoteser. Hypotesen måste därför vara formulerad på ett sådant sätt att den är mätbar. Kvantitativa metoder kräver en stor mängd data och statistisk analys för att undersökningen ska vara valid [15]. Kvalitativa metoder är utformade för att skapa förståelse för bakomliggande orsaker och beteenden för att forma hypoteser eller till exempel utveckla datorsystem. Kvalitativa metoder kräver en mindre mängd data än kvantitativa metoder [15].

### **3.1.2 Induktiva och deduktiva tillvägagångssätt**

Det finns olika tillvägagångssätt för att dra slutsatser och fastställa vad som är sant och falskt inom forskning [15]. Induktiva och deduktiva tillvägagångssätt är de vanligaste varianterna. Induktiva metoder innebär att slutsatser dras från insamlad data. Denna data är ofta insamlad med en kvalitativ forskningsmetod. Deduktiva metoder går ut på att verifiera eller falsifiera hypoteser. Dessa metoder använder ofta en stor mängd data och en kvantitativ undersökningsmetod.

### **3.1.3 Fallstudie**

En fallstudie är en forskningsstrategi som innebär att man studerar ett specifikt fall med tydliga avgränsningar [95, p. 56]. Detta görs med intentionen att skapa en detaljerad förståelse för processer, händelser, erfarenheter och relationer som är associerade med den specifika instansen av fallet [95, p. 52]. Användningen av en fallstudie som forskningsstrategi är vanlig vid småskalig forskning där målet är att få ut en detaljerad redogörelse av fallet.

Valet av fall som ligger som grund för studien är viktigt då det definierar studiens karaktär [95, p. 52]. Det är möjligt att inkludera flera fall i studien men den generella idén är att begränsa omfattningen för att möjliggöra en mer detaljerad inblick än vad som hade varit möjligt med en kvantitativ strategi. Målet med en fallstudie kan då formuleras som att ge kunskap om det generella fallet genom att studera ett specifikt fall [95, p. 53].

### **3.1.4 Den vetenskapliga arbetsmetoden**

För att ett arbete ska vara trovärdigt behöver det följa en välgrundad vetenskaplig arbetsmetod. En rapport skriven av Niclas Andersson och Anders Ekholm [96] presenterar en arbetsmetod som härstammar från en bok av Mario Bunge [97]. Arbetsmetoden är uppdelad i tio generella sekvenser:

1. Identifiera ett problem inom ett ämnesområde.
2. Beskriv problemet tydligt.

3. Kartlägg befintlig kunskap inom området, det vill säga identifiera information, metoder eller instrument som är relevanta för problemställningen.
4. Förklara och lös problemet med utgångspunkt från bakgrundskunskapen i steg 3. Om befintlig kunskap inom området inte är tillräckligt för att lösa problemet, gå vidare till steg 5, annars hoppa till det därpå följande steget.
5. Föreslå nya idéer, tekniker, teorier eller hypoteser och ta fram ny empirisk data för lösning av problemet.
6. Lägg fram lösningsförslag, antingen en exakt eller en approximativ lösning.
7. Härled konsekvenserna av den presenterade lösningen.
8. Testa lösningsförslaget.
9. Korrigera lösningsförslaget efter testresultatet.
10. Undersök lösningen i perspektiv av befintlig kunskap inom ämnesområdet (steg 3) och identifiera nya problemställningar.

### **3.1.5 Valda undersökningsmetoder**

Detta projekt använde en kvalitativ undersökningsmetod i form av en fallstudie. Studien bygger på utvecklingen av en röststyrd kalenderapplikation till Google Assistant. Projektet använder ett induktivt tillvägagångssätt för att dra slutsatser och forma teorier utifrån observationer och erfarenheter som samlats in under studien. Projektet följde Bunges vetenskapliga arbetsmetod eftersom den är passande till teknologiska undersökningar där mjukvara ska utvecklas.

## **3.2 Hur frågeställningarna besvarades**

Utifrån problemområdet som beskrevs i sektion 1.2 (punkt 1 och 2 i Bunges metod) togs tre frågeställningar fram. För att besvara frågeställningarna behöver arbetet leda till vissa resultat. Alla resultat som krävs för att besvara frågeställningarna och hur resultaten ska levereras beskrivs nedan.

Den första frågeställningen lyder: *Hur ser programmeringsmodellen ut för applikationer som använder ett röstgränssnitt?* För att besvara frågeställningen behövs en tydlig förståelse över vad en programmeringsmodell är. I grund och botten är programmeringsmodellen en abstrakt modell över hur en applikation fungerar. Modellen består av ett antal artefakter som beskriver olika delar av applikationen. För en röststyrd applikation behövs fyra artefakter för att förstå hur programmeringsmodellen ser ut. Nedan beskrivs varje artefakt och vad som behövs för att få fram dem:

1. *Utvecklingsprocessen* är stegen som ska utföras för att få fram en färdig applikation. För att ta fram utvecklingsprocessen för en röststyrd applikation krävs erfarenhetsresultaten som generas genom att utveckla en röststyrd applikation. Det krävs också teoretisk kunskap om olika verktyg och tjänster som ingår i utvecklingsprocessen. Dessa resultat ska tillsammans användas för att kunna formulera en utvecklingsprocess.
2. *Konversationsmönster* beskriver strukturen av en applikations röstgränssnitt. För att få fram ett konversationsmönster behövs först en litteraturstudie över vilka konversationsmönster som kan användas. Efter litteraturstudien implementeras och testas de möjliga konversationsmönster i en röststyrd applikation för att få fram ett lämpligt konversationsmönster.
3. *Arkitekturmönster* beskriver kodstrukturen i en applikation. För att få fram mönstret behövs en litteraturstudie över de möjliga arkitekturmönster som kan användas i en applikation. Av de möjliga mönster som finns implementeras och testas ett av dem i fallstudien.
4. *Konceptuell arkitektur* visar en överblick över en röststyrd applikations infrastruktur. Genom att utföra en litteraturstudie om de olika delarna som ingår i en röststyrd applikation tas en generell version av denna artefakt fram. Genom att utveckla en applikation under fallstudien kommer sedan denna generella arkitektur att anpassas för att passa den specifika applikationen.

Med dessa fyra artefakter kan den första frågeställningen besvaras.

Den andra frågeställningen lyder: *Vilka begränsningar finns vid utveckling*

*av en röststyrd applikation?* För att besvara denna frågeställning krävs en redogörelse för vilka begränsningar en utvecklare bör känna till vid utveckling av en röststyrd applikation till Google Assistant. Resultaten som behöver redovisas för att besvara frågeställningen är grundade i fallstudien. Fallstudien ger erfarenhetsresultat som kan användas för att beskriva olika begränsningar med den använda teknologin.

Den tredje frågeställningen lyder: *vilka resurser och ramverk finns att tillgå?* För att besvara denna frågeställning krävs en redogörelse av olika resurser och ramverk som är av intresse för applikationsutvecklare. För att leverera ett resultat som kan besvara denna frågeställning krävs en litteraturstudie för att samla relevant kunskap om tillgängliga utvecklingsverktyg för digitala assistenter.

### **3.3 Fallstudiens utformning**

Fallet som studien bygger på är ett mjukvaruprojekt där en röststyrd applikation till Google Assistant utvecklas. Applikationen hanterar frågor om användarens schemaläggning och kalenderhändelser. Användningsområdet kan till exempel vara frågor som "är person X tillgänglig klockan 13?" och "vad har person X schemalagt just nu?". Flera användare kan koppla sina digitala kalendrar till applikationen för att göra det möjligt att hämta informationen som behövs för att besvara användarens fråga. Applikationen använder svenska som språk och är utvecklad för att kunna hantera tvåvägskommunikation.

Valet av vilken specifik applikation som utvecklades i samband med studien gjordes genom att försöka uppfylla vad som kan kallas för en "typisk applikation". Alltså en applikation som har funktionalitet som ofta är inkluderad i röststyrd mjukvara. Detta innefattar en konversation där användaren kan välja mer än en "väg". Det inkluderar också behovet av programkod som körs på en webbserver som anropas med en webhook. Användningen av en databas och externa tjänster kan även anses vara typiskt för röststyrda applikationer. Alla ovan nämnda funktioner finns i applikationen som utvecklades vilket gjorde att den ansågs vara lämplig för studien.

### 3.4 Val av teknologi

Med tanke på de många olika digitala assistenterna och utvecklingsverktygen som finns tillgängliga krävs det att valen av teknologi är väl genomtänkta. De olika teknologiska val som gjordes inom projektet grundades på följande punkter.

- Tidigare personlig erfarenhet och kunskap.
- Teknologins relevans i dagsläget.
- Specifik funktionalitet som efterfrågas.

Personlig erfarenhet och kunskap kan påverka till exempel val av programmeringsspråk och kodbibliotek. Vid val av teknologi som är bekant sedan tidigare kan arbetet fortlöpa smidigare och kvaliteten höjas. Dock bör inte den personliga erfarenheten väga tyngre än de två andra punkterna eftersom detta kan påverka arbetets kvalitet negativt genom att mer lämplig teknologi förbises. Teknologins relevans innebär att valet av digital assistent och utvecklingsverktyg ska vara rimliga med avseende på produkternas popularitet bland användare. Specifik funktionalitet syftar på möjligheterna att utveckla viss funktionalitet som krävs för att uppfylla kravspecifikationen för applikationen.

Google Assistant är en av de mer populära digitala assistenterna och har den specifika funktionaliteten att den stödjer kommunikation på svenska och uppfyller därmed både punkt 2 och 3 ovan. Tidigare personlig erfarenhet fanns inte för någon av de digitala assistenterna och därför var det kriteriet inte relevant vid valet av Google Assistant för projektet. Google Assistant uppfyllde kriterierna bäst av alla digitala assistenter. Cortana och Alexa hade bra utvecklingsmöjligheter men saknade svenska. Siri har svenska men eftersom den är begränsad till Apples egna produkter får man begränsade utvecklingsmöjligheter.

Utvecklingen av applikationen skedde med hjälp av Dialogflow. Detta valet gjordes med avseende på punkt 2 och 3. Dialogflow är verktyget som har störst relevans i dagsläget vid utveckling av applikationer eftersom att mallar (templates) har begränsade utvecklingsmöjligheter och Actions SDK saknar NLU funktionaliteten som Dialogflow erbjuder.

Node.js användes som exekveringsmiljö på webbservern. Detta val gjordes på grund av att kodbibliotek finns tillgängliga till Dialogflow vilket underlättar utvecklingsprocessen. Node.js är också populärt bland webbutvecklare vilket överensstämmer med punkt 2 ovan.

Projektet använde en webbserver och en databas som hostades av Firebase. Firebase valdes för att en Firebase server kan ta emot webhook anrop från Dialogflow med hjälp av Cloud Functions. Dialogflow erbjuder en guide för att konfigurera en Firebase server för att kunna ta emot och hantera webhook anrop. Tjänsten är även gratis upp till en viss gräns vilket gör att den passar bra till mindre projekt, men kan även skalas upp för större projekt vid betalning. Det är också smidigt att använda samma tjänst för både server och databas.

För att hantera kalendrar i applikationen användes Googles egna kalendertjänst som heter Google Calendar. Valet av denna kalendertjänst gjordes på grund av tidigare erfarenhet med systemet. Åtkomst till tjänsten gjordes med Google Calendar API. Detta gjorde att konfigurationen av Calendar API:et blev smidigare eftersom ett Google konto redan hade skapats.

### **3.5 Datahantering och Informationssökning**

För att hantera applikationens programkod användes Github. Med Github lagrades källkoden för projektet och sparade hela historiken över alla ändringar i koden. Med tjänsten kunde samarbetet ske mer effektivt genom att använda tjänstens verktyg för att hantera kodändringar. För rapportskrivning användes det webbaserade LaTeX redigeringsverktyget Overleaf. Med verktyget kunde flera personer redigera rapporten samtidigt. Molnlagringstjänsten Google Drive användes för att hantera andra dokument såsom referenser och figurer.

För att hitta webbartiklar och forskningsrapporter användes Googles sökmotor, Google Scholar och IEEE Xplore. Sökmotorn användes även för att hitta enkla guider på hur man skapar en röststyrd applikation. Det vetenskapliga arkivet DiVA användes för att undersöka gamla examensarbeten som ansågs vara relevanta till projektet. För att hitta relevanta böcker som behövdes för att utföra litteraturstudien användes Library Genesis (LibGen) som är en sökmotor

för vetenskapliga artiklar och böcker.

### 3.6 Projektmetod

Alla projekt kan dra nytta av ett strukturerat tillvägagångsätt och tydliga projektmål. Ett agilt arbetssätt valdes för detta projekt på grund av den flexibilitet som det ger till arbetet. Tidigare positiva erfarenhet med agila arbetsmetoder påverkade också valet.

Arbetet följde en iterativ arbetsmetod som är en del av den agila metodologin. Projektet var uppdelat i två faser där varje fas hade ett antal iterationer. I den första fasen gjordes en undersökning i form av en litteraturstudie som var uppdelad i tre iterationer. Första iterationen av litteraturstudien fokuserade på digitala assistenters bakomliggande teknologi. Den andra iterationen fokuserade på kunskap om tjänster och verktyg för utveckling av röststyrda applikationer. Den tredje iterationen var inriktad på konversations- och arkitekturmönster som kan implementeras i röststyrda applikationer.

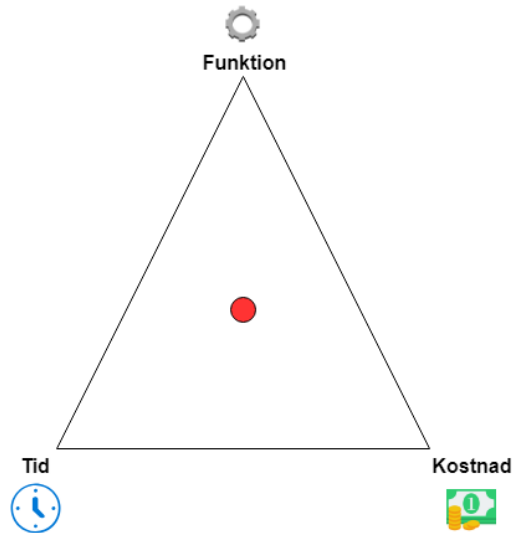
I den andra fasen behandlades utvecklingen av den röststyrda applikationen. Utvecklingen var uppdelad i fyra iterationer. I varje iteration lades det till ny funktionalitet som sedan testades för att kontrollera att funktionen fungerade. Efter alla iterationer testades hela applikationen för att säkerställa att allt fungerade som förväntat. Parallellt med utvecklingsfasen utfördes även undersökningen för att leverera resultaten som var nödvändiga för att besvara frågeställningarna. En utförlig beskrivning av varje fas och dess iterationer finns i sektion 3.7.

### Åtagandetriangeln

I projektet användes den åtagandetriangeln som beskrivs av Sven Eklund i boken *Arbeta i projekt* [98, p. 128-129]. I figur 3.1 visas triangeln som har hörnen funktion, tid och kostnad. Dessa tre hörn representerar tre viktiga egenskaper som man ska tänka på i ett projekt. Att uppfylla varje egenskap i triangeln är problematiskt eftersom varje projekt stöter på problem som kan påverka tiden,



kostnaden eller funktionaliteten. Därför ska man låta minst en av egenskaperna vara flexibel. I detta projekt var tiden begränsat till tio veckor och kostnaden var fixerad eftersom den inte hade diskuterats i början av arbetet. Detta gjorde att funktionen blev den flexibla delen genom att använda MoSCoW metoden.



Figur 3.1: Åtagandetriangeln. Bilden är skapad av författarna med draw.io[64].

I figur 3.1 visas en triangel med tre hörn som representerar projektets tre viktiga egenskaper, vilket är tid, funktion och kostnad.

### 3.7 Iterationbeskrivning och kravspecifikation

Under projektets gång användes prioriteringstekniken MoSCoW som beskrivs i sektion 2.10. Detta gjordes för att prioritera krav som är nödvändiga för att kunna besvara frågeställningarna. Nedan följer den kategoriserade kravspecifikationen:

#### **Måste ha:**

- Ta fram de fyra artefakterna som ingår i programmeringsmodellen
- Redogörelse av begränsningar vid utveckling av en röststyrd applikation
- Redogörelse av tillgängliga resurser och ramverk.

- Skapa en röststyrd applikation som kan hämta aktiviteter från en persons kalender.
- Applikationen ska kunna hantera olika kalenderfrågor till exempel “vad gör X idag?” eller “Var är X mellan 13:30 och 15:30?”.

**Bör ha:**

- Applikationen ska kunna hämta information från flera kalendrar.

**Kan ha:**

- Integrera applikationen med KTH:s lärarkalender.
- Applikationen kan ge ut kontaktinformation som till exempel telefonnummer eller e-postadress.

**Kommer inte ha:**

- Skicka mail eller SMS till personen användaren söker efter.

## **Fas 1: Litteraturstudie**

Litteraturstudien delades upp i 3 iterationer där varje iteration bidrog till kraven som finns i MoSCoW kategorierna. Nedan beskrivs varje iteration med dess mål och vilka krav de bidrog till. Denna fas motsvarar punkt 3 i Bunges vetenskapliga arbetsmetod (kartlägg befintlig kunskap inom området).

### **Iteration 1: Bakomliggande teknologi**

Den första iterationen i litteraturstudien riktade in sig på att studera digitala assistenters underliggande teknologi. Målet med iterationen var att förstå viktiga ämnen som Artificiell Intelligens och Natural Language Processing (som inkluderar NLU och NLG). Studien var första steget mot att förstå hur utvecklingsverktyg för digitala assistenter fungerar, vilket bidrog till att uppfylla kravet *resurser och ramverk* under *Måste ha* kategorin i MoSCoW.

## **Iteration 2: Tjänster och verktyg**

Den andra iterationen i litteraturstudien fokuserade på att studera olika tjänster och verktyg som finns tillgängliga för utveckling av röststyrda applikationer. Detta inkluderar tjänster som Google Cloud Platform, Dialogflow, Firebase och verktyg som olika kodbibliotek. Kunskapen som denna iteration gav gjorde det möjligt att ta beslut om vilken teknologi som skulle användas i projektet utifrån kriterierna som beskrivs i sektion 3.4. Tre av kraven under *Måste ha* kategorin i MoSCoW är kopplade till denna iteration. Utvecklingsprocessen är en del av programmeringsmodellen och för att skapa den behövdes kunskap om verktygen som användes i arbetet. För att skapa den konceptuella arkitekturen som också ingår i programmeringsmodellen krävdes kunskap om olika tjänster och verktyg och hur de kommunicerar med varandra. Denna iteration låg också till grund för att kunna svara på frågeställningen som rör resurser och ramverk.

## **Iteration 3: Konversations- och arkitekturmönster**

Den sista iterationen i litteraturstudien fokuserade på att hitta olika konversations- och arkitekturmönster som kunde implementeras i den röststyrda applikationen. Målet med iterationen var att jämföra olika mönster och välja ett konversations- och ett arkitekturmönster som skulle implementeras. Detta bidrog till att uppfylla kravet av att ta fram de fyra artefakterna som tillhör *Måste ha* kategorin genom att ta fram ett konversationsmönster och ett arkitekturmönster.

## **Fas 2: Fallstudie**

Utvecklingsfasen delades in i fyra iterationer som hade som mål att uppfylla olika punkter i MoSCoW. Nedan följer en beskrivning av dessa iterationer och vilka iterationsmål som sattes upp. Denna fas motsvarar steg 4-9 i Bunges vetenskapliga arbetsmetod. Stegen fokuserar på att fram ett lösningsförslag och testa det.

### **Iteration 1: Dialogflow**

Den första iterationen fokuserade på att konfigurera Dialogflow. Detta innebar att implementera det valda konversationsmönstret för applikationens röstgränssnitt i Dialogflow. Denna funktionalitet är kopplad till sista kravet i *Måste ha* kategorin. Det valda konversationsmönstret testades och förbättrades för att få ut den slutgiltiga artefakten. Erfarenhetsresultaten som konfigurationen av Dialogflow gav användes för att beskriva utvecklingsprocessen.

### **Iteration 2: Koppla Dialogflow med en server**

Den andra iterationen fokuserade på att koppla Dialogflow med en server. Detta gjordes genom att sätta upp en kommunikationsväg mellan Dialogflow och Firebase. Med hjälp av kopplingen blev det möjligt att skapa funktionalitet för att hämta kalenderhändelser, vilket tillhör den fjärde punkten i *Måste ha* kategorin. Iterationsmålet var att samla kunskap om hur Dialogflow kan kopplas till en server för att utvidga utvecklingsprocessen.

### **Iteration 3: Hämtning av kalenderhändelser**

I den tredje iterationen användes det valda arkitekturmönstret för att strukturera upp serverkoden. Sedan implementerades funktionalitet för att hämta kalenderhändelser från Googles kalendertjänst. Målet med iterationen var att bidra till att uppfylla två krav under *Måste ha* kategorin. Artefakten arkitekturmönster testades praktiskt och utvecklingsprocessen utvidgades med delen av arbetet som rör serverkoden. Kravet som säger att applikationen ska kunna hämta kalenderhändelser uppfylldes under denna iteration.

### **Iteration 4: Databasuppsättning**

För att uppfylla punkten i *Bör ha* kategorin behövdes en databas som lagrade kalenderinformation för att kunna komma åt olika kalendrar i Google Calendar.

Målet med denna iteration var att sätta upp databasen och koppla den med servern. Iterationen slutförde utvecklingsprocessen.

## 4 Utveckling av den röststyrda applikationen

I detta kapitel förklaras hur utvecklingen av den röststyrda applikationen gick till. Innan utvecklingen påbörjade behövdes först ett Google konto<sup>4</sup> och ett Google Cloud Platform projekt<sup>5</sup>. Kontot och projektet användes för att konfigurera Dialogflow, Firebase och Google Calendar API. Sektion 4.1 beskriver utvecklingen av en agent i Dialogflow. Sektion 4.2 förklarar konfiguration av servern, databasen och åtkomst till Google Calendar. Sektion 4.3 går igenom hur serverkoden ser ut för att hantera fulfillments. Sektion 4.4 går igenom verktygen som användes för att testa applikationen.

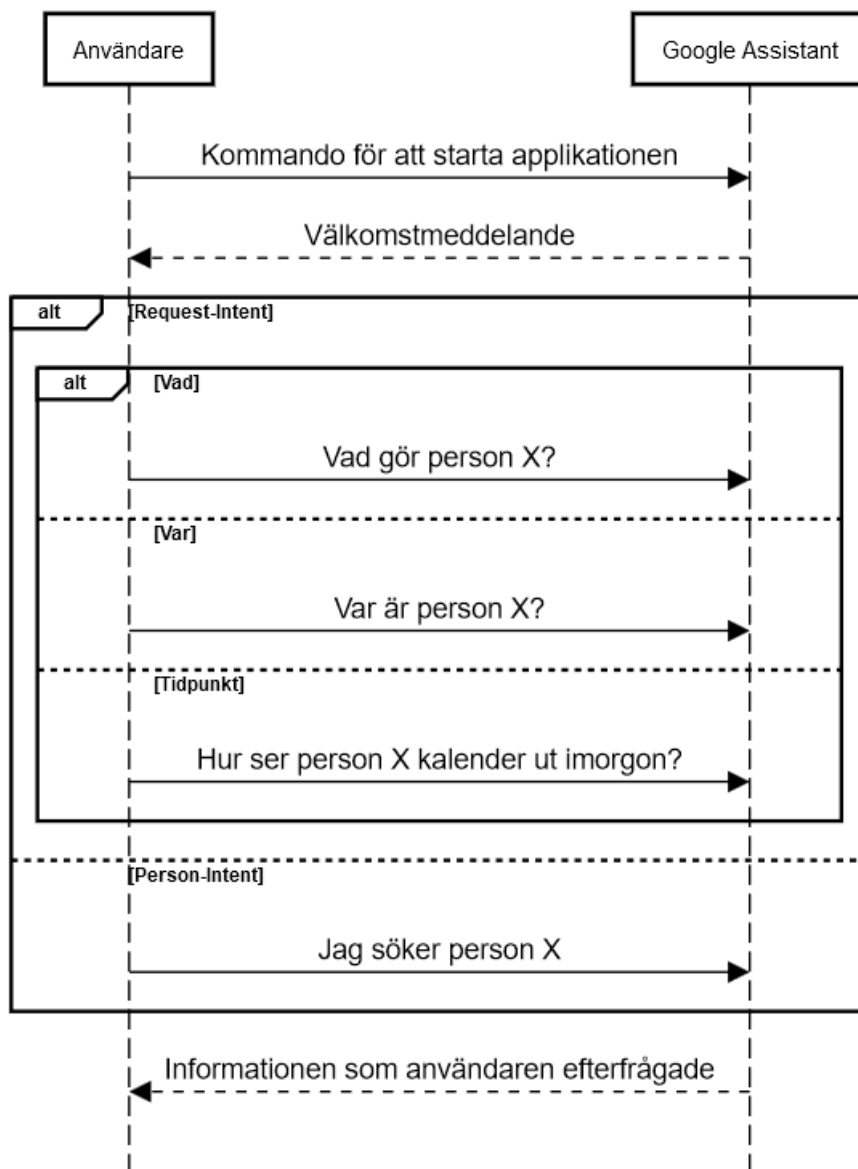
### 4.1 Konfiguration av Dialogflow

Första steget i utvecklingen av den röststyrda applikationen innebar att skapa och konfigurera en agent i Dialogflow. En agent kan skapas genom att följa Dialogflows egna guide [99]. Kalenderapplikationens grundläggande funktionalitet, som ligger under kategorin *måste ha* i MoSCoW, implementerades först. Ett diagram som visar hur det önskade konversationsflödet ska se ut skapades och användes som mall för att kunna identifiera behovet av intents. Diagrammet visas i figur 4.1.

---

<sup>4</sup><https://support.google.com/accounts/answer/27441>

<sup>5</sup><https://cloud.google.com/resource-manager/docs/creating-managing-projects>



Figur 4.1: Konversationsflöde. Bilden är skapad av författarna.

Figur 4.1 är ett sekvensdiagram som visar hur kommunikationen kan se ut mellan en användare och applikationen (Google Assistant i figuren). Tiden i figuren löper uppifrån och ner. Användaren inleder konversationen genom att be Google Assistant om att få prata med applikationen. Applikationen startar då och svarar med ett välkomstmeddelande. Användaren kan då antingen uppge att den söker en person eller ställa en mer specifik fråga angående någon persons schema. Detta visas i figuren i *alt*-rutan. Applikationen ger sedan ett svar till användaren som innehåller informationen som användaren efterfrågade. Med hjälp av figuren skapades följande intents:

- Default Welcome Intent
- Followup-Welcome-Intent
- Default Fallback Intent
- Followup-Fallback-Intent
- Person-Intent
- Request-Intent

*Default Welcome intent* anropas automatiskt av Google Assistant när applikationen startas och innehåller därför en välkomstfras som inleder konversationen. Intents som börjar med namnet *Default* finns i nyskapade agenter i Dialogflow och behöver därför inte tränas med hjälp av träningsfraser. Denna intent innehåller två olika fraser som används av applikationen, ”Hej, vem letar du efter?” och ”Hej, vem söker du?”. Dessa är skapade för att göra användargränssnittet mer intuitivt genom att ställa en direkt fråga. Eftersom att svaren som ska ges till användaren med denna intent är hårdkodade och ej kräver någon backend-logik så fanns inte heller något behov av att använda entities.

*Default Fallback Intent* används när användaren säger något som applikationen inte kan hantera. Detta kan ske till exempel om användaren talar otydligt eller efterfrågar funktioner som ligger utanför applikationens omfattning. Eftersom att det också är ett default intent krävdes inga träningsfraser. Endast ett svar har konfigurerats som är ”Jag förstod inte vad du sa. Vem söker du?”. Detta svar är designat så att om användaren har missförstått hur applikationen ska användas så ska den direkta frågan göra det lätt att ge ett korrekt svar tillbaka.

*Person-Intent* skapades för att hantera när en användare uttrycker att de söker en specifik person. Agenten tränades med fraserna som visas i figur 4.2. Denna Intent konfigurerades med två obligatoriska entities, förnamn och efternamn. Detta gjordes för att både för- och efternamn krävs för att hitta rätt kalender i databasen av kopplade kalendrar. Entities markeras som obligatoriska genom att kryssa i ”REQUIRED”-rutan i Dialogflow. Detta resulterar i att om användaren inte lämnar denna information kommer assistenten att explicit att efterfråga den.



Training phrases ?

Search training phrases

Q

^

” Add user expression

” Jag vill hitta Johan Andersson

” Jag söker Evan Saboo

” Jag söker Johan Andersson

Action and parameters ^

Enter action name

REQUIRED ?

PARAMETER NAME ?

ENTITY ?

VALUE

IS LIST ?

PROMPTS ?

☒

first-name

@first-name

\$first-name

☐

Define prompts...

☒

last-name

@last-name

\$last-name

☐

Define prompts...

☐

request-entity

@Request-entity

\$request-entity

☐

—

☐

Enter name

Enter entity

Enter value

☐

—

Figur 4.2: Person-Intent träningsfraser och entities. Bilden är skapad av författarna.

Figur 4.2 visar träningsfraserna som användes för att träna agenten för att kunna hantera Person-Intent. Färgerna över för- och efternamnen i träningsfraserna visar att agenten känner igen dessa ord som entities. I figuren finns även de tre entities som används i denna intent.

*Request-Intent* skapades för att hantera mer specifika frågor som ”Vad gör person-X imorgon klockan 12?”. Träningsfraserna som användes för att träna agenten visas i figur 4.3. Även denna intent har för- och efternamn som obligatoriska entities. Utöver det finns även ett flertal entities som berör datum och tidsperioder som visas i figur 4.4. Valet av entities som handlar om tid och datum gjordes för att fånga upp flera olika sätt som användaren kan uppge en tid eller tidsperiod på. Om datum utelämnas av användaren kommer nuvarande tidpunkt att användas för sökning i kalendern (mer om detta i sektion 4.3). En entity som kallas för request-entity skapades för denna intent. Tanken bakom det var att bättre kunna fånga användarens avsikt med sin fråga, och därmed även

ge ett mer passande svar. Med hjälp av denna entity kan applikationen särskilja mellan en fråga som berör vad en viss person gör, och var en person befinner sig.

Training phrases ?

” Add user expression

” Var är Anders Sjögren idag

” Var är Johan Andersson just nu

” Var är Kalle

” Var är Anders Sjögren mellan 13 och 14

” Var är Anders Sjögren mellan idag och imorgon

” Vad gör Anders Sjögren mellan 13 och 14

” Vad gör Anders Sjögren mellan idag och imorgon

” Vad gör Kalle idag

” Vad gör Kalle

” Var är Anders Sjögren

Figur 4.3: Request-Intent träningsfraser. Bilden är skapad av författarna.

Action and parameters

Enter action name

REQUIRED ?	PARAMETER NAME ?	ENTITY ?	VALUE	IS LIST ?	PROMPTS ?
<input type="checkbox"/>	date-time	@sys.date-time	\$date-time	<input type="checkbox"/>	—
<input checked="" type="checkbox"/>	first-name	@first-name	\$first-name	<input type="checkbox"/>	Define prompts...
<input checked="" type="checkbox"/>	last-name	@last-name	\$last-name	<input type="checkbox"/>	Vad är \$first-n...
<input type="checkbox"/>	date	@sys.date	\$date	<input type="checkbox"/>	—
<input type="checkbox"/>	date-period	@sys.date-period	\$date-period	<input type="checkbox"/>	—
<input type="checkbox"/>	request-entity	@Request-entity	\$request-entity	<input type="checkbox"/>	—
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>	—

+ New parameter

Figur 4.4: Request-Intent Entities. Bilden är skapad av författarna.

Figur 4.3 visar några av träningsfraserna som skapades i Request-Intent. Figur 4.4 visar alla entities som används i Request-Intent. I bilaga A hittas fler skärmdumpar som visar den fullständiga konfigurationen av Dialogflow.

Under testningen av applikationen upptäcktes att vissa namn inte kändes igen av agenten och därmed inte kopplades till rätt namn-entity. Det var namn som kan beskrivas som "ovanliga" som skapade problem. Detta visade sig vara svårt att lösa på ett bra sätt så vissa namn var tvungna att hårdkodas för att fungera (visas i figur A.4 och A.5). Med hjälp av två entities, first-name och last-name, kunde namn som annars inte hade identifierats av Dialogflow användas. Genom att kombinera entities som redan fanns tillgängliga i Dialogflow (entities med "sys" i namnet) med en lista av hårdkodade namn löstes problemet. Denna lösning har dock uppenbara brister som till exempel skalningsproblem eftersom det kräver manuell konfiguration av namnlistor.

## **4.2 Uppsättning av server, databas och Google Calendar API**

Innan förklaringen av hur serverkoden fungerar behövs en genomgång av hur servern, databasen och Google Calendar API sattes upp.

### **4.2.1 Serverkonfiguration**

I projektet användes en Firebase server för att ta emot och hantera webhook anrop från Dialogflow. För att installera Firebase CLI i en lokal maskin behövdes Node.js. Med Node.js kan man använda npm (Node Package Manager) för att installera Firebase CLI (kodbibliotek) genom att köra ett terminalkommando. Kommandot visas i kodblock 1 och betyder att Firebase CLI ska installeras globalt för att kunna användas över alla terminaler i datorn.

```
$ npm install -g firebase-tools
```

Kodblock 1: Terminalkommando för att installera Firebase CLI.

I nästa steg hämtades ett Node.js projekt från Github som har JavaScript-kod för att kunna ta emot webhook anrop från Dialogflow. Projektet är distribuerat av Dialogflow och kan hämtas genom att köra git-kommandot i kodblock 2.

```
$ git clone https://github.com/dialogflow/fulfillment-webhook-nodejs.git
```

Kodblock 2: Hämta mallkod för fulfillments.

Efter att projektet hämtades från Github behövde alla Node.js moduler hämtas till functions-mappen som existerar i projektets huvudkatalog. Detta gjordes för att kunna använda Node.js modulerna i Firebase servern. Kommandot för att hämta alla moduler visas i kodblock 3.

```
$ npm install
```

Kodblock 3: Hämta Node.js moduler

För kunna ladda upp kod till Firebase servern behövde man skapa ett Firebase projekt på deras hemsida, men eftersom ett Google Cloud Platform (GCP) projekt hade redan skapats kunde det användas som ett Firebase projekt. Efter projektskapandet användes kommandot i kodblock 4 för att logga in i Firebase med Google kontot som tillhörde Firebase projektet. Sedan valdes rätt Firebase projekt genom att skriva in kommandot i kodblock 5 på terminalen.

```
$ firebase login
```

Kodblock 4: Logga in i Firebase.

```
$ firebase use <projekt-ID>
```

Kodblock 5: Använd specifik Firebase projekt

Det projekt-ID som används i kodblock 5 hämtades genom att gå till Firebase Console<sup>6</sup> via en webbläsare, välja ett existerande projekt (eller skapa ett nytt projekt) och navigera till *inställningar (ikon) > Project settings*. I *General*-fliken finns projekt-ID:t.

---

<sup>6</sup><http://console.firebase.google.com>

Till sist behövde bara projektet laddas upp till servern genom att använda kommandot som visas i kodblock 6. Kommandot betyder att bara funktionen som har angivits ska laddas upp och användas i servern. Funktionen använder sig av Firebase Cloud Functions för att kunna ta emot webhook anrop. I vårt fall var funktionen redan definierad i `index.js` filen (finns i `functions` mappen).

```
$ firebase deploy --only functions:dialogflowFirebaseFulfillment
```

Kodblock 6: Ladda upp funktionen `dialogflowFirebaseFulfillment` till Firebase servern.

För att komma åt servern från Dialogflow behövdes det läggas in en HTTP URL i fulfillments sidan som finns i Dialogflow Console (figur A.6). URL:n hämtades från Firebase Console genom att navigera till *Functions* i navigationsmenyn och gå till *Dashboard*-fliken.

#### 4.2.2 Kalenderkonfiguration

Googles Calendar användes i projektet för att få tillgång till kalenderhändelser från olika kalendrar. För att kunna göra ett anrop till kalendertjänsten behövdes Google Calendar API. För att kunna använda API:et behövs en API-nyckel. Nyckeln hämtades genom att först gå till GCP:s webbgränssnitt<sup>7</sup> och sedan navigera till *APIs & Services > Library*. Därifrån ska man söka efter Google Calendar API och aktivera den. Efter att API:et har aktiverats ska man navigera till *APIs & Services > Credentials* och trycka på knappen "Create credentials" och välja "API key", då ska ett fönster öppnas som visar API-nyckeln.

I sektion 4.3.5 förklaras hur API-nyckeln användes för att hämta kalenderhändelser från en kalender.

#### 4.2.3 Databaskonfiguration

Firebase har sin egen implementation av databashantering, vilket kallas för Realtime Database. För att kunna använda databasen behövdes en privat nyckel

---

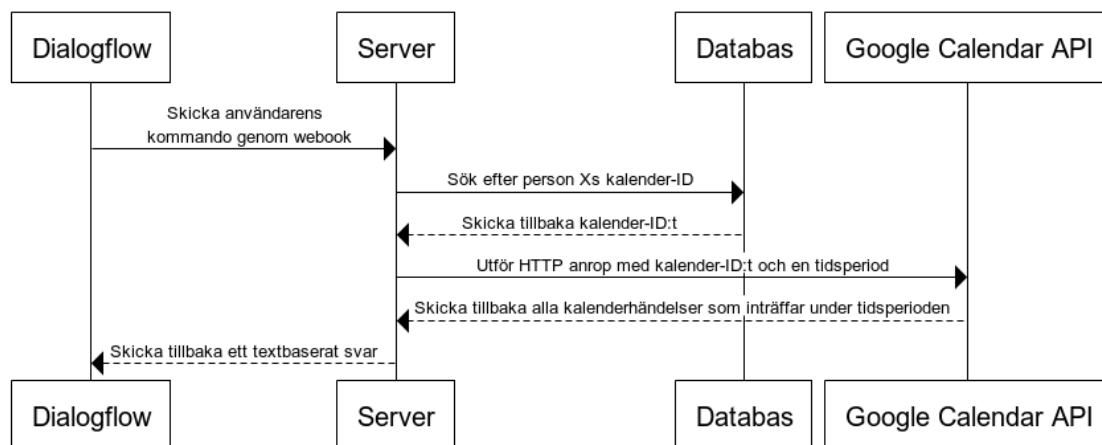
<sup>7</sup>[console.cloud.google.com](https://console.cloud.google.com)

i form av en JSON-fil. Filen hämtades genom att gå till Firebase Console och navigera till *inställningar (ikon)* > *Project settings*. På inställningssidan går man till *Service Accounts*-fliken och klickar på knappen "Generate new private key" för att hämta privata nyckeln. I samma flik visas även hur man konfigurerar databaskopplingen med nyckeln beroende på vilken programmeringsspråk man utvecklar i.

I sektion 4.3.2 förklaras hur servern hämtar data från databasen.

### 4.3 Hantering av fulfillments

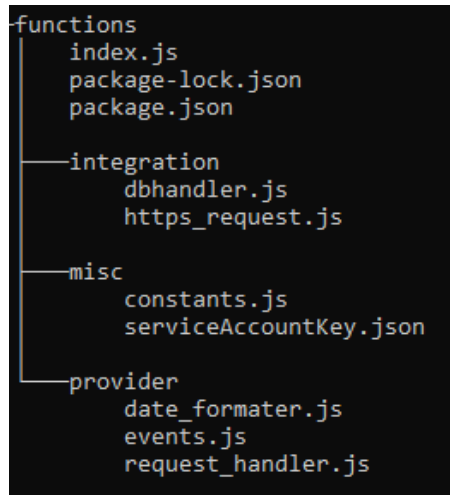
Alla fulfillments som servern hanterar skrevs i programmeringsspråket JavaScript. Node.js användes för att exekvera JavaScript-koden på servern. För att kunna få en överblick över hur servern hanterar webhook anrop och skickar svar tillbaka till Dialogflow skapades ett diagram som visar serverflödet (figur 4.5). Serverkodens filstruktur visas översiktligt i figur 4.6 och alla JavaScript filer och dess funktioner presenteras i figur 4.7.



Figur 4.5: Serverflöde. Bilden är skapad av författarna.

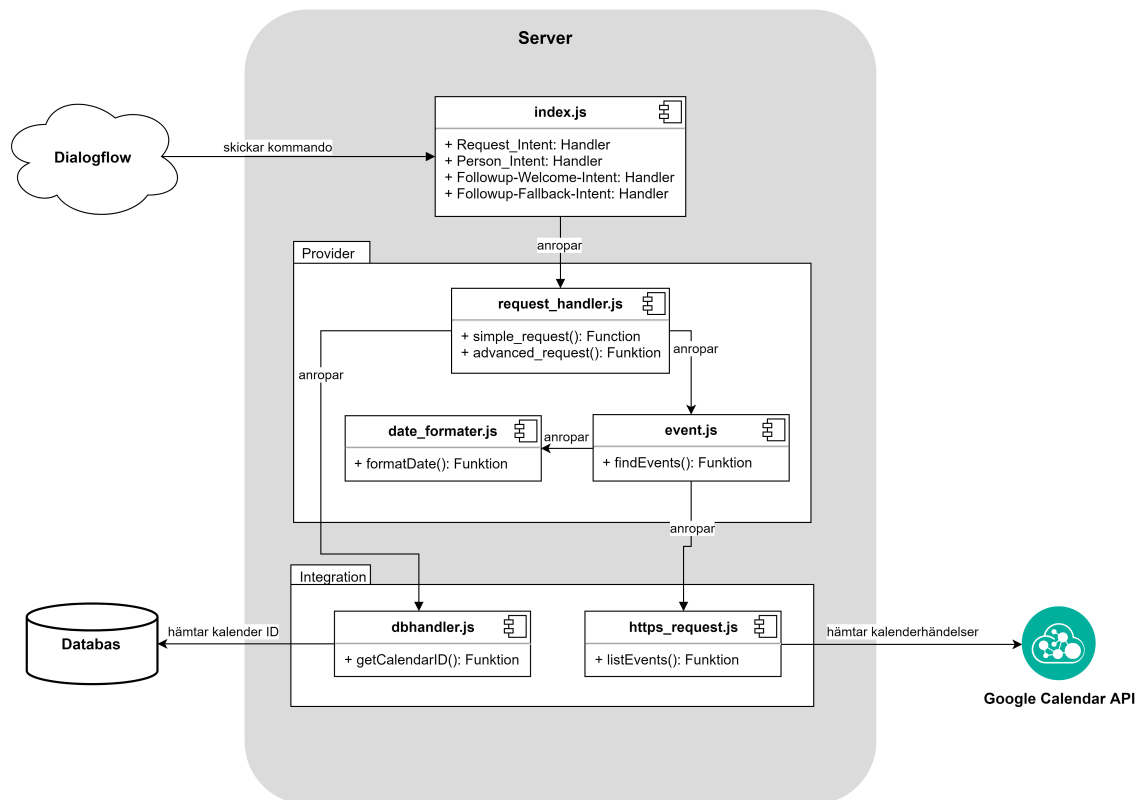
Figur 4.5 är ett sekvensdiagram som visar hur servern hanterar webhook-anrop och skickar svar tillbaka till Dialogflow. Först skickar Dialogflow användarens röstkommando till servern. Servern tar emot röstkommandot och söker i databasen efter personen som användaren vill ha kalenderinformation om. Efter att servern har fått kalender-ID:t från databasen utförs ett HTTPS anrop till

Google Calendar API:et med kalender-ID:t och tidpunkten (eller tidsperioden) taget från användarens röstkommando. API:et skickar då tillbaka alla hittade kalenderhändelser till servern och sedan bygger servern upp ett textbaserat svar som skickas tillbaka till Dialogflow.



Figur 4.6: Serverkodens filstruktur. Bilden är skapad av författarna.

I figur 4.6 visas filstrukturen för serverkoden, vilket kan kommas åt i Github [100].



Figur 4.7: Komponentdiagram över serverkoden. Bilden är skapad av författarna.

I figur 4.7 visas alla JavaScript filer och funktioner som serverkoden använder för att hantera användarens kommando. Serverkoden följer arkitekturmönstret *Layer Architectural Pattern*. Koden har delats upp i fyra lager som heter *Controller*, *Model*, *Integration* och *Data*. Controller-lagret tar emot data som kommer från Dialogflow och skickat det vidare till Model-lagret. Filen `index.js` representerar hela Controller-lagret. Model-lagret hanterar datan och bygger upp ett svar till användarens kommando. I figuren representeras Model-lagret av Provider-mappen. Integration-lagret hanterar anrop till databasen och Google Calendar API. Integration-lagret representeras av Integration-mappen. Data-lagret är Firebase databasen som lagrar alla kalender-ID.

#### 4.3.1 Mottagning av webhook anrop

För att kunna ta emot webhook anrop från Dialogflow användes Firebase Cloud Functions. För att hantera anropet användes något som kallas intent hanterare (handler på engelska). I figur 4.8 visas vilka moduler som behövs för att sätta



upp Cloud Functions och intent hanterare. I figur 4.9 visas kod för användning av intent hanterare och uppsättning av Cloud Functions.

```
1 //Import firebase functions
2 const functions = require('firebase-functions');
3 □ const {dialogflow} = require('actions-on-google');
4 const request_handler = require('./provider/request_handler');
5 const simple_request = request_handler.simple_request;
6 const advanced_request = request_handler.advanced_request;
7 const continue_text = " Vill du veta något mer?";
8
9 const REQUEST_INTENT = 'Request-Intent';
10 const PERSON_INTENT = 'Person-Intent';
11 const FF_INTENT = "Followup-Fallback-Intent";
12 const FW_INTENT = "Followup-Welcome-Intent";
13 const app = dialogflow();
```

Figur 4.8: Kodblock 1 taget från index.js [100]

I figur 4.8 visas kod för att hämta nödvändiga funktioner för att hantera intents. För att kunna använda intent hanterare behöver man först hämta in modulen *actions-on-google* med hjälp av funktionen `require()`. Denna modul har funktioner för att ta emot anrop och skicka svar tillbaka till Dialogflow. Modulen *firebase-functions* används för att låta Firebase servern ta emot webhook anrop från Dialogflow. Den sista modulen *request\_handler* har två funktioner som hanterar data som tas emot från Dialogflow. I figuren ser man även att fyra konstanter har skapats som innehåller namn på intents som finns i Dialogflow. Den sista konstanten `app` innehåller en instans av Dialogflow funktionen från kodbiblioteket som gör det möjligt att använda intent hanterare. Modulerna och konstanterna används senare i figur 4.9.

```

15  app.intent(PERSON_INTENT, (conv) => {
16      return simple_request(conv).
17      then((response) => {
18          conv.ask(response+ continue_text);
19      })
20      .catch((error) => {
21          conv.close(error);
22      });
23  });
24
25  app.intent(REQUEST_INTENT, (conv) => {
26      return advanced_request(conv).
27      then((response) => {
28          conv.ask(response+ continue_text);
29      })
30      .catch((error) => {
31          conv.close(error);
32      });
33  });
34
35  app.intent(FF_INTENT, (conv) => {...9 lines });
44
45  app.intent(FW_INTENT, (conv) => {...9 lines });
54
55  exports.dialogflowFirebaseFulfillment = functions.https.onRequest(app);

```

Figur 4.9: Kodblock 2 taget från index.js [100]

I Figur 4.9 visas fyra intent hanterare. För att sätta upp en hanterare används funktionen `intent()` som tar emot namnet på den intent den ska hantera och en anonym funktion som hanteraren ska exekvera när data tas emot från Dialogflow.

Den första intent hanteraren tar emot data från "Person-Intent". Den anonyma funktionen som är definierad i hanteraren tar emot en parameter som är ett objekt av klassen `DialogflowConversation` (Dialogflows kodbibliotek). Objektet innehåller användarens kommando och alla entities som tillhör "Person-Intent". I koden skickas objektet vidare till `simple_request()` (behandlas i sektion 4.3.2) funktionen för att hantera datan i objektet och få tillbaka ett textbaserat svar. Svaret skickas tillbaka till Dialogflow med funktionen `ask()` som tillhör `DialogflowConversation` klassen. Funktionen `ask()` meddelar även Dialogflow att användaren kan ge ett till röstkommando efter att det textbaserade svaret har lästs upp för användaren. Ett svar kan även skickas med funktionen `close()` och då kommer assistenten att läsa upp svaret och sedan avsluta applikationen.

I "Request-Intent" hanteraren ser man att funktionen `advanced_request()` (behandlas i sektion 4.3.2) används istället för `simple_request()` eftersom denna intent hanterar mer komplicerade röstkommandon. De två sista intent hanterarna använder samma funktion som den första intent hanteraren (Person-Intent) för att behandla data. I slutet av figuren exporteras en funktion från modulen *firebase-functions* för att låta konstanten `app` ta emot webhook anrop. Funktionen blir då en molnfunktion (Cloud Function) . Namnet till den exporterade funktionen är namnet som ska läggas in i Dialogflow fulfillments konfigurationen (visas i figur A.6).

### 4.3.2 Hantering av webhook anrop

För att hantera data från enkla Dialogflow intents används funktionen `simple_request()` som visas i figur 4.10 och figur 4.11. För att hantera avancerade intents används funktionen `advanced_request()` som visas i figur 4.12, figur 4.13 och figur 4.14.

```
12 function simple_request(conv) {  
13     var params = conv.parameters;  
14  
15     var fname = params['first-name']  
16         .hasOwnProperty('given-name')  
17         ? (params['first-name'])['given-name'] : params['first-name'];  
18     var lname = params['last-name']  
19         .hasOwnProperty('last-name')  
20         ? (params['last-name'])['last-name'] : params['last-name'];  
21  
22     var reqType = params['request-entity'];  
23 }
```

Figur 4.10: Kodblock 1 taget från funktionen `simple_request()` som finns i `request_handler.js` filen [100]

I figur 4.10 visas första delen av funktionen `simple_request()`. Funktionen hanterar intents som bara fångar tre entities från ett kommando, där "Person-Intent" är en av dessa. I objektet `conv` finns ett JSON objekt som innehåller parametrarna (entities) förnamn, efternamn och `request_entity`. Förnamn och efternamn används för att hitta personens kalender i databasen som användaren söker efter. `Request_entity` används för att veta hur användarens röstkommando är utformat för att kunna skicka tillbaka ett lämpligt svar. Om användaren till

exempel ställer frågan ”**Var är** Evan Saboo?” skickar då servern tillbaka svaret ”Evan är i Kista” eller om frågan är ”**Vad gör** Evan Saboo?” så skickar servern tillbaka ”Evan har föreläsning.”.



```
40 return new Promise((resolve, reject) => {
41     return getCalendarID(fname, lname)
42         .then((value) => {
43             if (value === null || value === 'null') {
44                 resolve("Jag kan tyvärr inte hitta någon information om personen.");
45                 return;
46             }
47             var calendarID = value[Object.keys(value)[0]].calendarID;
48             findEvents(calendarID, new Date(), null, fname, reqType)
49                 .then((resp) => {
50                     resolve(resp);
51                 })
52                 .catch((error) => {
53                     reject(error);
54                 });
55             });
56         .catch((error) => {
57             reject(error);
58         });
59     });
60 }
```

Figur 4.11: Kodblock 2 taget från funktionen `simple_request()` som finns i `request_handler.js` filen [100]

I figur 4.11 visas andra delen av funktionen `simple_request()`. I figuren kallas först funktionen `getCalendarID()` (behandlas i sektion 4.3.3) med parametrarna för- och efternamn för att hämta personens kalender-ID från databasen. Om personen som användaren söker efter inte hittas i databasen skickas ett textbaserat svar till Dialogflow om att personen inte kunde hittas.

Kalender-ID:t används senare i funktionen `findEvents()` (behandlas i sektion 4.3.4) för att hämta händelser från en persons kalender. Funktionen `findEvents()` tar också emot två tidpunktsobjekt för att söka efter kalenderhändelser som sker mellan tidpunkterna. I figuren skickas ett nytt tidpunktsobjekt som första tidpunkten och ett NULL-värde som andra tidpunkten. Detta betyder att användaren vill hitta en kalenderhändelse som sker just nu. Ett exempel på ett sådant kommando kan vara ”Var är Person X just nu”.

```

46 function advanced_request(conv) {
47     var params = conv.parameters;
48     var fname = params['first-name'].hasOwnProperty('given-name')
49         ? (params['first-name'])['given-name'] : params['first-name'];
50     var lname = params['last-name'].hasOwnProperty('last-name')
51         ? (params['last-name'])['last-name'] : params['last-name'];
52     var datetime = params['date-time'].hasOwnProperty('date_time')
53         ? (params['date-time'])['date_time'] : params['date-time'];
54     var reqType = params['request-entity'];
55     var date = params['date'];
56     var datePeriod = params['date-period'];

```

Figur 4.12: Kodblock 1 taget från funktionen `advanced_request()` som finns i `request_handler.js` filen [100]

I figur 4.12 visas första delen av funktionen `advanced_request()`. Funktionen har samma struktur som `simple_request()` men skillnaden mellan dem är att `advanced_request()` hanterar även specifika tidpunkter för att hitta en eller flera kalenderhändelser i en persons kalender. I figuren ser man att de nödvändiga parametrarna (entities) tas ut från `conv` objektet och tilldelas till nya variabler. Dessa parametrar behandlas i figur 4.13 och figur 4.14.

```

83 return new Promise((resolve, reject) => {
84     return getCalendarID(fname, lname)
85         .then((value) => {
86             if (value === null || value === 'null') {
87                 resolve("Jag kan tyvärr inte hitta någon information om personen.");
88                 return;
89             }
90
91             var calendarID = value[Object.keys(value)[0]].calendarID;
92             var startdate;
93             var enddate;
94             if (empty(datetime) && empty(date) && empty(datePeriod)) {
95                 startdate = new Date();
96                 enddate = null;
97             } else if (!empty(datetime)) {
98                 if (datetime.hasOwnProperty('startDateTime')) {
99                     startdate = datetime.startDateTime;
100                     enddate = datetime.endDateTime;
101                 } else {
102                     startdate = datetime;
103                     enddate = null;
104                 }

```

Figur 4.13: Kodblock 2 taget från funktionen `advanced_request()` som finns i `request_handler.js` filen [100]

Figur 4.13 visar den andra delen av funktionen `advanced_request()`. På samma sätt som i funktionen `simple_request()` används funktionen `getCalendarID()` för att hitta en persons kalender-ID genom att söka med personens för- och

efternamn. Efter att kalender-ID:t hittas definieras två variabler, startdate och enddate, som används för att hitta kalenderhändelser mellan två tidpunkter. Variablerna tilldelas med olika tidpunkter beroende på vad användaren har angett i sitt kommando. Om ingen tidpunkt anges i kommandot vill användaren bara veta vad person X har schemalagt just nu. Om en tidpunkt har angetts vill användaren veta vad person X har schemalagt vid en specifik tidpunkt. Om en tidsperiod har angetts vill användaren veta vad person X har schemalagt mellan två tidpunkter.

```
79 | } else if (!empty(date)) {  
80 |     datetime = new Date(date);  
81 |     startdate = datetime.setHours(0, 0, 0, 0);  
82 |     enddate = datetime.setHours(23, 59, 59, 999);  
83 |  
84 | } else if (!empty(datePeriod)) {  
85 |     startdate = (new Date(datePeriod.startDate)).setHours(0, 0, 0, 0);  
86 |     enddate = (new Date(datePeriod.endDate)).setHours(23, 59, 59, 999);  
87 |  
88 | } else {  
89 |     resolve(DEFAULT_RESPONSE);  
90 |     return;  
91 | }  
92 | findEvents(calendarID, startdate, enddate, fname, reqType)  
93 |     .then((resp) => {  
94 |         resolve(resp);  
95 |     })  
96 |     .catch((error) => {  
97 |         reject(error);  
98 |     });  
99 |  
100 | }, (error) => {  
101 |     reject(error);  
102 | });
```

Figur 4.14: Kodblock 3 taget från funktionen advanced\_request som finns i request\_handler.js filen [100]

I figur 4.14 visas tredje och sista delen av funktionen advanced\_request(). Koden fortsätter med att identifiera vilken tidpunkt användaren har angett. Efter att startdate och enddate har blivit tilldelade med tidpunktsobjekt anropas funktionen findEvents() för att hämta kalenderhändelser och bygga upp ett textbaserat svar till användaren.

### 4.3.3 Hämta kalender-ID från databasen

För att hämta kalender-ID:t behövdes först en koppling till Firebase databasen sättas upp. Detta görs i figur 4.15. I figur 4.16 visas funktionen getCalendarID()

för att hämta en persons kalender-ID från databasen.

```
1  const admin = require("firebase-admin");
2  const serviceAccount = require("../misc/serviceAccountKey.json");
3  const cts = require("../misc/constants");
4  const ERROR_MSG = cts.error_msg;
5
6  admin.initializeApp({
7    credential: admin.credential.cert(serviceAccount),
8    databaseURL: "https://newagent-44155.firebaseio.com"
9  });
```

Figur 4.15: Kodblock 1 taget från dbhandler.js. [100]

I figur 4.15 hämtas modulen *firebase-admin* som ger möjlighet att komma åt Firebase databasen. Med funktionen `initializeApp()` initieras en koppling till databasen. Funktionen tar emot två parametrar där ena är en privat nyckel som finns i `serviceAccountKey.json` filen och den andra är databasens URL. Parametrarna hämtades genom att följa instruktionerna i sektion 4.2.3.

```
11 function getCalendarID(fname, lname) {
12   var db = admin.database();
13   var key = 'calendars/';
14   var search = db.ref(key)
15     .orderByChild('name')
16     .equalTo(fname.toLowerCase() + " " + lname.toLowerCase());
17
18   return new Promise((resolve, reject) => {
19     search.once('value', (snapshot) => {
20       resolve(snapshot.val());
21     }, (error) => {
22       console.log("Database error: ", error);
23
24       reject(ERROR_MSG);
25     });
26   });
27 }
28 }
```

Figur 4.16: Kodblock 2 taget från dbhandler.js som visar funktionen `getCalendarID()`.

I figur 4.16 visas kod för att hämta en persons kalender-ID från databasen. För att förstå vad som hämtas från databasen visas en del av databasstrukturen i kodblock 7. Genom att först använda funktionen `ref()` skapas en referens till elementet

”calendars” för att kunna läsa eller skriva data till den databaspositionen. För att hitta barnelementet ”calendarID” används för- och efternamn som sökparametrar för att matcha med barnelementet ”name”. Med funktionen `once()` exekveras sökningen vilket antingen returnerar data tillbaka eller värdet `NULL` om inget hittas i databasen. Svaret från databasen skickas direkt tillbaka till funktionen som kallade på `getCalendarID()` med Promise parametern `resolve`.

```
{
  "calendars" : [ {
    "calendarID" : "1t2@group.calendar.google.com",
    "name" : "anders sjögren"
  }, {
    "calendarID" : "1t3@group.calendar.google.com",
    "name" : "evan saboo"
  } ]
}
```

Kodblock 7: Databas strukturen i JSON format

I kodblock 7 visas en liten del av databas strukturen. Strukturen är en JSON struktur där elementet ”calendars” är en array som innehåller två objekt.

#### 4.3.4 Bygga upp ett svar till användaren

För att kunna bygga upp ett svar till Dialogflow användes funktionen `findEvents()` som visas i figur 4.17, figur 4.18 och figur 4.19.



```

48 function findEvents(calendarId, startdate, enddate, fname, request_type) {
49     return new Promise((resolve, reject) => {
50         try {
51             startdate = new Date(startdate);
52             if (enddate === null) {
53                 dtMax = new Date(startdate);
54                 enddate = new Date(dtMax.setSeconds(dtMax.getSeconds() + 1));
55             } else {
56                 enddate = new Date(enddate);
57             }
58         } catch (error) {
59             console.log("Error on findEvents function: ", error);
60             reject(ERROR_MSG);
61             return;
62         }
63     });

```

Figur 4.17: Kodblock 1 taget från funktionen findEvents() som finns i events.js filen [100]

I figur 4.17 visas första delen av funktionen findEvents(). I början av funktionen hanteras parametrarna startdate och enddate. Om enddate är NULL kopieras tidpunkten från startdate och tilldelas till enddate för att sedan addera tidpunkten i enddate med en sekund. Anledningen till att detta görs är för att Calendar API:et kräver två tidpunkter för att kunna hitta kalenderhändelser.

```

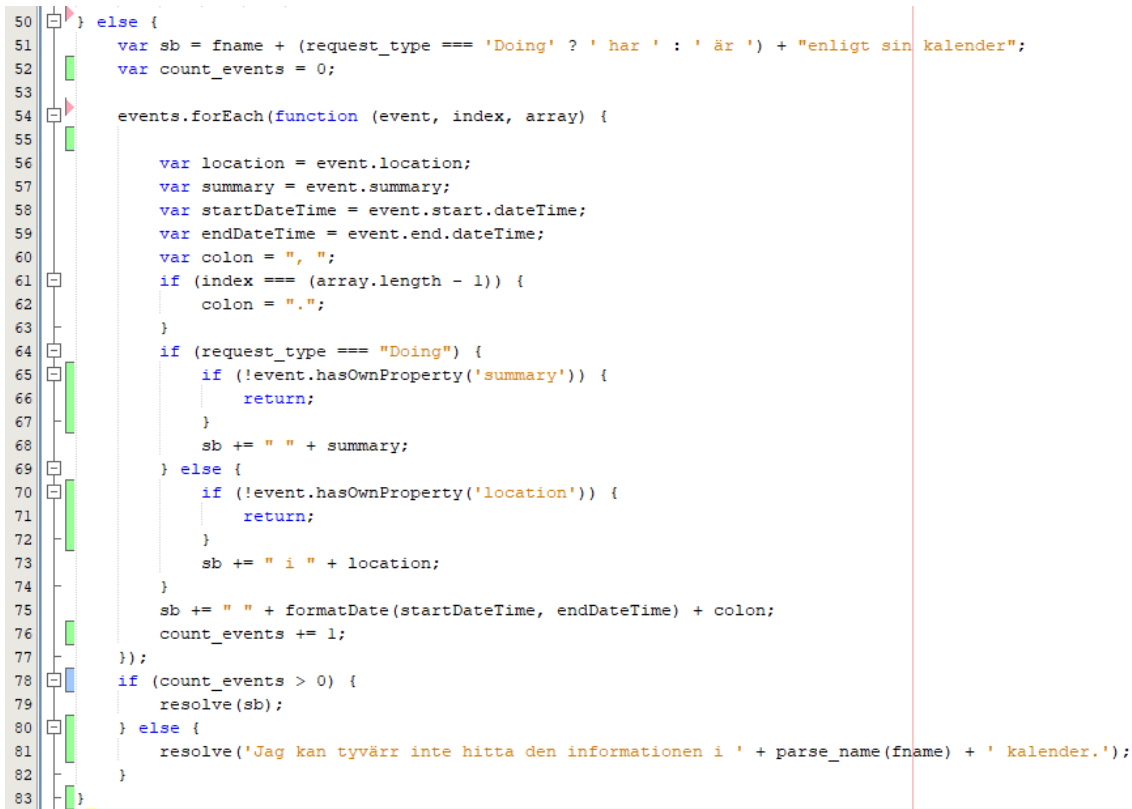
38 listEvents(calendarId,
39     encodeURIComponent(startdate.toISOString()),
40     encodeURIComponent(enddate.toISOString()))
41     .then((events) => {
42         if (events.length < 1) {
43             if (request_type === "Doing") {
44                 resolve(fname + ' har inget planerat.');

```

Figur 4.18: Kodblock 2 taget från funktionen findEvents() som finns i events.js filen [100]

Figur 4.18 visar andra delen av funktionen findEvents(). Efter att de två tidpunkterna har hanterats används dem i listEvent() (förklaras i sektion 4.3.5) som anropar Google Calendar API för att hämta alla kalenderhändelser som inträffar mellan de två tidpunkterna. Funktionen listEvent() returnerar sedan tillbaka ett JSON objekt som innehåller kalenderhändelser. Om JSON objektet inte innehåller någon kalenderhändelse returnerar funktionen findEvents() ett

textbaserat svar om att person X inte har något planerat under den angivna tidsperioden.



```
50 } else {
51   var sb = fname + (request_type === 'Doing' ? ' har ' : ' är ') + "enligt sin kalender";
52   var count_events = 0;
53
54   events.forEach(function (event, index, array) {
55
56     var location = event.location;
57     var summary = event.summary;
58     var startDateTime = event.start.dateTime;
59     var endDateTime = event.end.dateTime;
60     var colon = ", ";
61     if (index === (array.length - 1)) {
62       colon = ".";
63     }
64     if (request_type === "Doing") {
65       if (!event.hasOwnProperty('summary')) {
66         return;
67       }
68       sb += " " + summary;
69     } else {
70       if (!event.hasOwnProperty('location')) {
71         return;
72       }
73       sb += " i " + location;
74     }
75     sb += " " + formatDate(startDateTime, endDateTime) + colon;
76     count_events += 1;
77   });
78   if (count_events > 0) {
79     resolve(sb);
80   } else {
81     resolve('Jag kan tyvärr inte hitta den informationen i ' + parse_name(fname) + ' kalender.');
```

Figur 4.19: Kodblock 3 taget från funktionen findEvents() som finns i events.js filen [100]

Figur 4.19 visar tredje och sista kodstycket av funktionen findEvents(). Alla kalenderhändelser i JSON objektet och deras egenskaper sammanställs till ett komplett textbaserat svar. Varje kalenderhändelse har egenskaperna summary, location, start- och enddate. Summary anger vilken typ av händelse/aktivitet kalenderhändelsen är, till exempel "möte", "lektion" eller "redovisning". Egenskapen location anger på vilken plats händelsen sker. start- och enddate anger mellan vilka tidpunkter kalenderhändelsen sker.

Svaret som skapas i funktionen är beroende av vad användaren vill veta om någon persons kalender. Om användaren till exempel ställer frågan "Var är Person X?" returnerar då funktionen svaret "Person X är enligt sin kalender i Plats Y mellan Tid 1 och Tid 2", vilket innehåller egenskaperna location, start- och enddate taget från kalenderhändelsen. Om användaren istället frågar "Vad gör

Person X?” så skickar funktionen tillbaka svaret ”Person X har enligt sin kalender Aktivitet Y mellan Tid 1 och Tid 2”, där summary har tagit med i svaret istället för location. Om summary eller location inte har angetts i kalenderhändelsen blir den exkluderad från svaret.

#### 4.3.5 Hämta kalenderhändelser från Google Calendar

För att hämta kalenderhändelser från en persons publika kalender används funktionen `listEvents()` (figur 4.20). Funktionen gör ett HTTPS anrop till Google Calendar API:et för att sedan få tillbaka en lista av alla kalenderhändelser mellan två tidpunkter.

```
6 function listEvents(calendarId, dateFrom, dateTo) {
7     var apiKey = 'AIzaSyDXSrMo_Y0lV94nFcntnMbsID5wjleFLdg';
8     return new Promise((resolve, reject) => {
9         https.get('https://www.googleapis.com/calendar/v3/calendars/' +
10             calendarId +
11             '/events?key=' +
12             apiKey +
13             '&timeMin=' + dateFrom +
14             '&timeMax=' + dateTo, (resp) => {
15             var data = '';
16             // A chunk of data has been recieved.
17             resp.on('data', (chunk) => {
18                 data += chunk;
19             });
20             resp.on('end', function () {
21                 try {
22                     var events = JSON.parse(data);
23                     return resolve(events.items);
24                 } catch (e) {
25                     console.log("Error parsing json: ", e);
26                     reject(ERROR_MSG);
27                 }
28             });
29             }).on('error', function (e) {
30                 console.log("Error getting data from API: ", e);
31                 reject(ERROR_MSG);
32             });
33         });
34     });
35 }
```

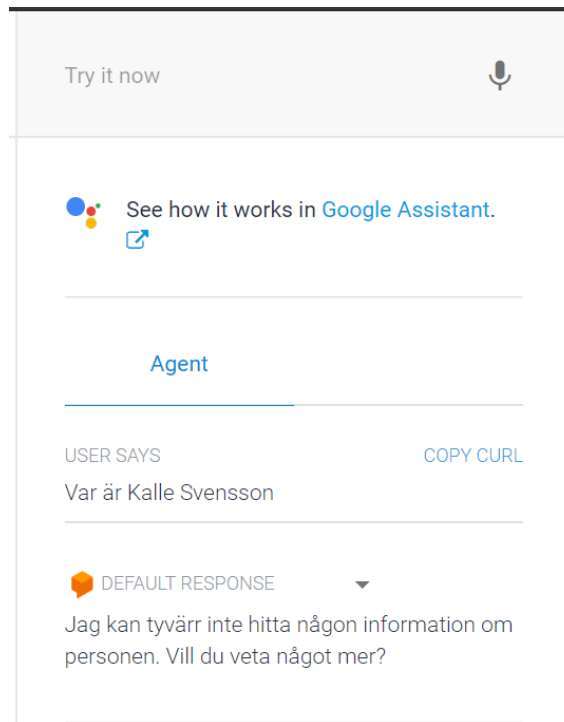
Figur 4.20: Funktion i `http_request.js` som anropar Google Calendar API för att hämta kalenderhändelser [100]. Bilden är skapad av författarna.

I figur 4.20 visas koden för funktionen `listEvents()`. Funktionen tar emot parametrarna kalender-ID, start- och slutpunkt. Med parametrarna körs ett

HTTPS anrop till Google Calendar API:et. För att kunna utföra HTTPS anrop behövdes först en modul hämtas in med funktionen `require('https')` och tilldelas till konstanten `https`. På kodrad 9 används konstanten för att göra ett HTTPS GET anrop till API:et med hjälp av en URL [101]. För att anropet ska hämta alla kalenderhändelser mellan två specifika tider behövs kalender-ID, API-nyckeln och tidsperioden. Dessa parametrar läggs i URL:n (visas i figuren). API:et skickar sedan tillbaka ett JSON objekt med alla kalenderhändelser. Om objektet som hämtas från API:et är för stort kommer det först delas upp i ett antal bitar och hämtas sekventiellt. Med funktionen `resp.on('data')` tas alla bitar emot och sätts ihop till ett komplett objekt. När alla bitar har hämtats exekveras funktionen `resp.on('end')` för att returnera JSON objektet.

#### 4.4 Testnings- och granskningsfas

Testningen av den röststyrda applikationen utfördes på tre olika sätt. Det första var genom att använda Dialogflows testkonsol som visas i figur 4.21 och figur 4.22. Konsolen kan kommas åt direkt från Dialogflow. Det andra sättet var att använda en Google Assistant simulator. Simulatoren visas i figur 4.23 och figur 4.24. Det tredje sättet var genom att använda en Google Home högtalare. För att kunna komma åt applikationen via Google Home behöver man först vara inloggad med Google kontot som är kopplat till den röststyrda applikationen och sedan uttala röstkommandot *"Hej Google, starta min testapp"* till Google Home (*testapp* är standardnamnet som varje Dialogflow applikation har från början). Högtalaren användes som testenheter för att kontrollera hur applikationen fungerar i en Google Assistant enhet.



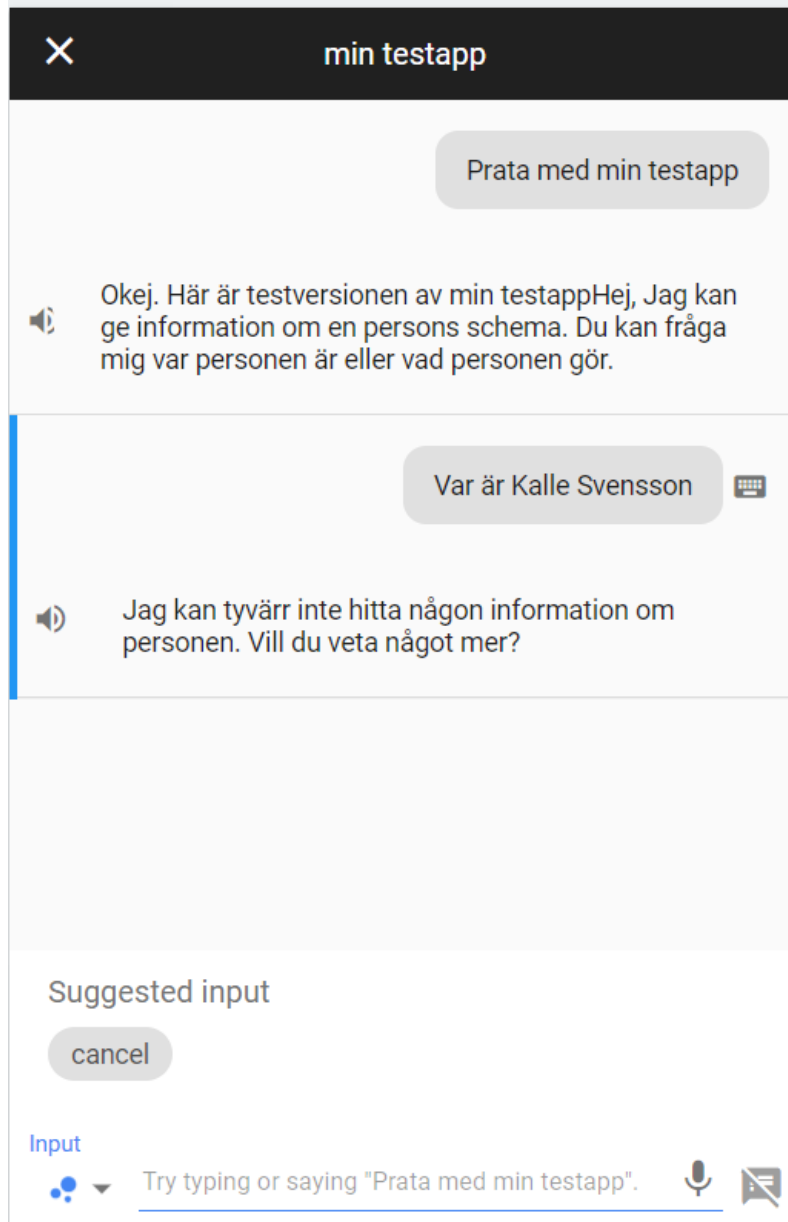
Figur 4.21: Första halvan av Dialogflows testkonsol. Bilden är skapad av författarna.

I figur 4.21 visas första halvan av Dialogflows testkonsol. Konsolen används för att testa ett kommando i någon intent. I figuren testas kommandot "Var är Kalle Svensson" och svaret blir "Jag kan tyvärr inte hitta någon information om personen" vilket betyder att applikationen inte kan hitta en person vid det namnet. Figur 4.22 visar den andra halvan av testkonsolen.

INTENT	
Request-Intent	
ACTION	
Not available	
PARAMETER	VALUE
date-period	
date-time	
first-name	{"given-name":"Kalle"}
request-entity	Where
date	
last-name	{"last-name":"Svensson"}
DIAGNOSTIC INFO	

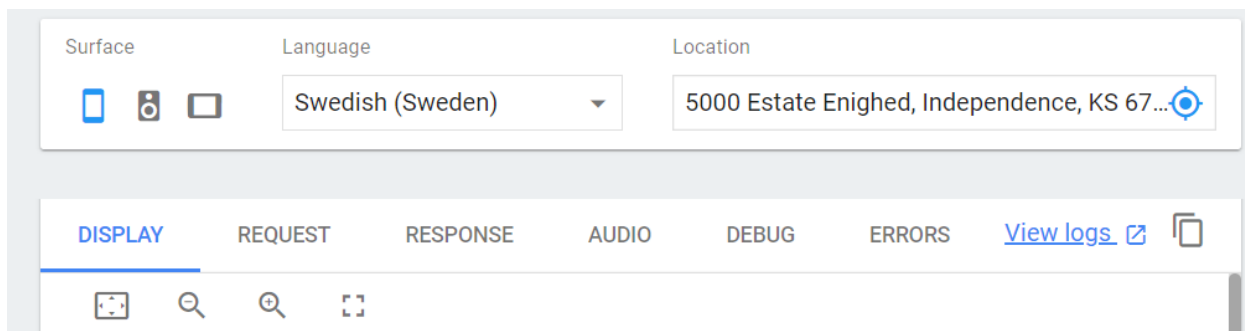
Figur 4.22: Andra halvan av Dialogflows testkonsol. Bilden är skapad av författarna.

Figur 4.22 visar den intent som kommandot från figur 4.21 matchar och vilka värden varje entity (parameter) fångar från kommandot. Längst ner i konsolen finns en knapp som heter "DIAGNOSTIC INFO". Denna knapp öppnar en ny ruta som visar vilken JSON-data som skickas till servern och vilken JSON-data Dialogflow tar emot från servern.



Figur 4.23: Skärmdump över simulatorns konversationsfönster. Bilden är skapad av författarna.

I figur 4.23 visas första delen av simulatorn. I konversationsfönstret kan man starta sin röststyrda applikation med kommandot *"prata med min testapp"* eller *"starta min testapp"*. För att testa olika kommandon kan man antingen skriva in dem i textfältet eller spela in dem med en mikrofon. Efter att ett kommando har matats in i konversationen svarar applikationen med ett lämpligt svar. I figur 4.23 kan vi se samma kommando som i figur 4.21. Direkt till höger om konversationsfönstret visas simulatorns inställningar (figur 4.24).



Figur 4.24: Skärmdump över simulatorns inställningar och diagnosinformation. Bilden är skapad av författarna.

I figur 4.24 visas inställningarna för simulatorn. I inställningarna kan man ändra platsinformation, språk och vilket hårdvarugränssnitt simulatorn ska använda. Simulatorn kan antingen köra mobil-, högtalar- eller ”smart display”-gränssnitt. I *Display*-fliken kan man se hur applikationen ser ut i den valda gränssnittet. I figur 4.24 finns också flikarna *Request*, *Response*, *Audio*, *Debug* och *Errors*. I *Request*-fliken visas data som skickas till servern när simulatorn tar emot en kommando. I *Response*-fliken visas data för svaret man fick från servern. I *Audio*-fliken kan man lyssna på svaret som presenteras för användaren. I *Debug*-fliken visas exakt vad som skickas mellan Google Assistant och Dialogflow och i *Errors*-fliken visas felmeddelande om något fel sker med hanteringen av användarens kommando.



## 5 Resultat

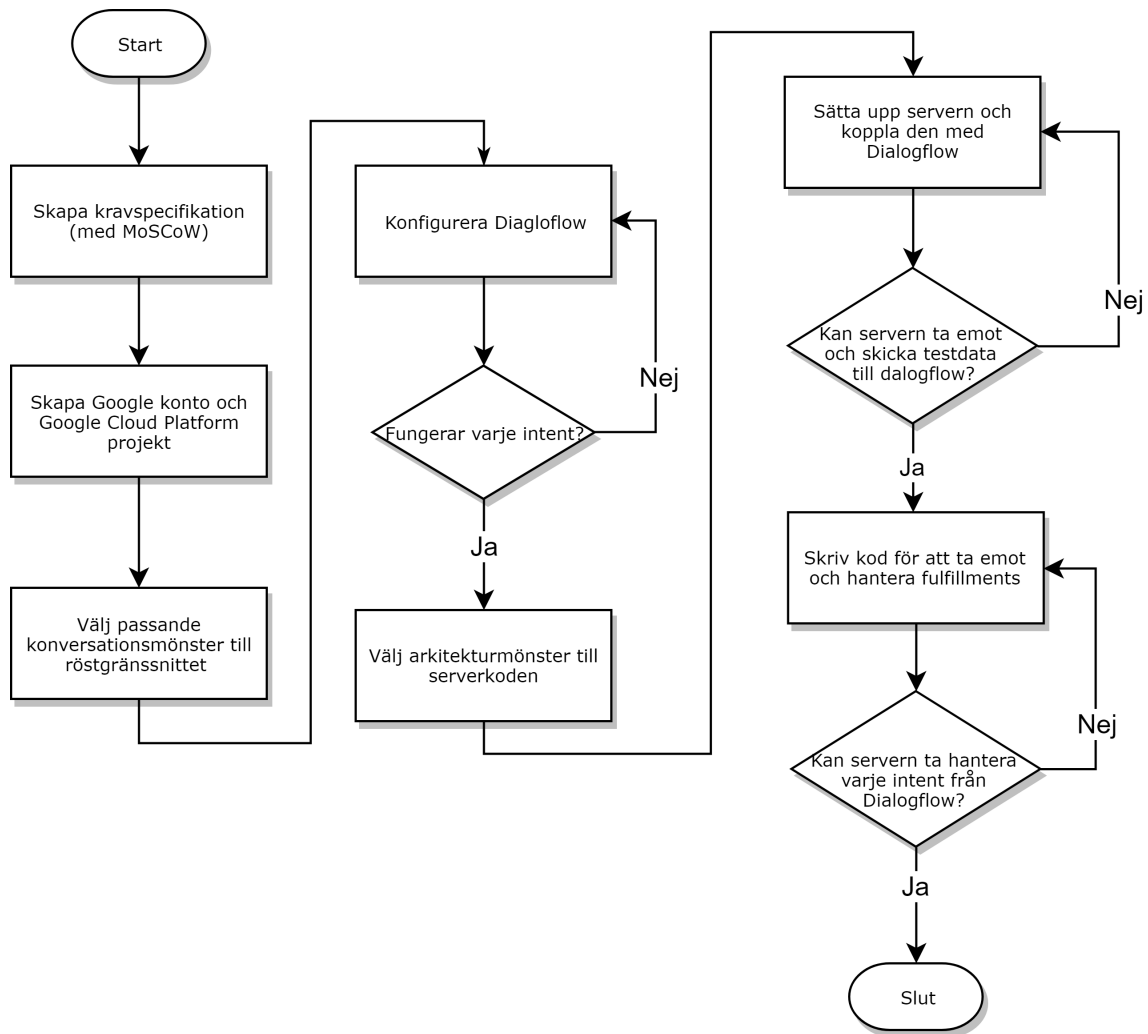
Detta kapitel redovisar resultaten av arbetet som är underlaget för att besvara följande frågeställningar:

- Hur ser programmeringsmodellen ut för applikationer som använder ett röstgränssnitt?
- Vilka begränsningar finns vid utveckling av en röststyrd applikation?
- Vilka resurser och ramverk finns att tillgå?

Sektion 5.1 presenterar en utvecklingsprocess som kan användas för att utveckla röststyrda applikationer med hjälp av Dialogflow. Sektion 5.2 redovisar ett konversationsmönster för ett röstgränssnitt. Ett arkitekturmönster presenteras i sektion 5.3 som beskriver hur kodbasen kan struktureras. Sektion 5.4 redovisar en detaljerad version av konceptuella arkitekturen. Sektion 5.5 beskriver hur olika teknikval påverkade programmeringsmodellen. Sektion 5.6 presenterar olika begränsningar som påträffades under utvecklingsfasen. Sektion 5.7 redovisar vad den utvecklade kalenderapplikationen klarar av.

### 5.1 Utvecklingsprocess

Utvecklingsprocessen består av olika steg i arbetet som leder till en fungerande applikation. Figur 5.1 visar en utvecklingsprocess som kan användas för utveckling av röststyrda applikationer till Google Assistant med hjälp av Dialogflow.



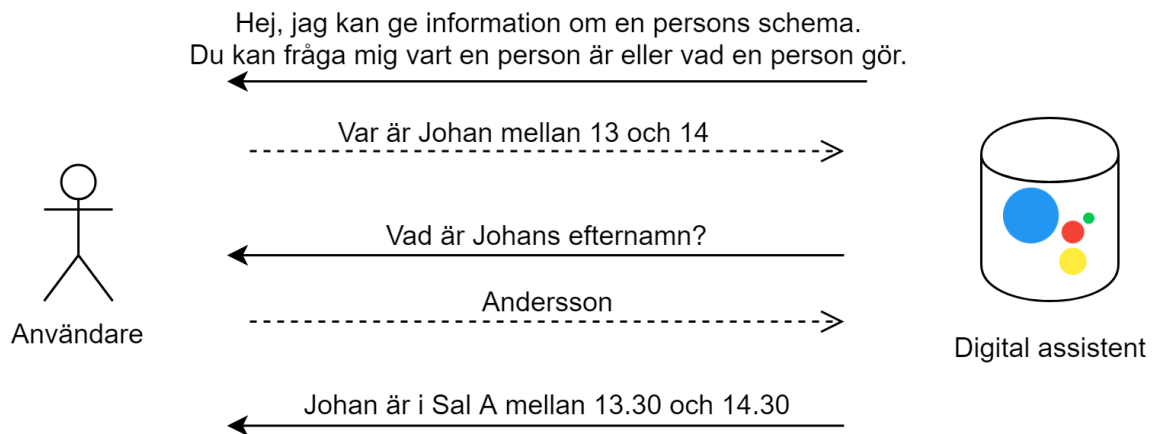
Figur 5.1: Flödesdiagram som visar utvecklingsprocessen. Bilden är skapad av författarna.

Figur 5.1 visar utvecklingsprocessen. Pilarna i figuren pekar på nästa steg i processen. Arbetet bör inledas med att skapa en kravspecifikation. Detta innebär att kraven för applikationen skrivs ner och eventuellt rangordnas (med till exempel MoSCoW). Steg två innebär att skapa ett Google konto och sedan ett Google Cloud Platform projekt. Dessa krävs för att kunna skapa applikationen med Googles utvecklingsverktyg. Steg tre innebär att ett konversationsmönster måste väljas för röstgränssnittet. Detta kan göras genom att välja konversationsmönstret som presenteras i sektion 5.2 eller genom att skapa en annan designlösning som passar till applikationen. Steg fyra innebär att Dialogflow konfigureras efter mönstret som bestämdes i föregående steg. Detta innefattar skapandet av intents, entities och fulfillments som tillsammans bildar

röstgränssnittet som användaren kommunicerar med. Detta steg i processen innehåller också testning för att säkerställa att allt fungerar innan nästa steg påbörjas. I steg fem bestäms ett arkitekturmönster för serverkoden. Detta görs för att få en bra struktur för koden som ska skrivas. Steg sex handlar om att sätta upp webbservern och möjliggöra kommunikationen mellan servern och Dialogflow. Även detta steg innehåller testning. Steg sju handlar om att skriva koden som hanterar alla fulfillment. Detta inkluderar också att konfigurera eventuella databaser och externa tjänster. Den slutgiltiga testningen av applikationen ingår också i detta steg för att säkerställa att applikationen uppfyller kravspecifikationen.

## **5.2 Konversationsmönster för röstgränssnitt**

I sektion 2.9.1 beskrivs tre olika konversationsmönster som kan användas i en röststyrd applikation. Konversationsmönster 1 samlar information från användaren ett steg i taget tills applikationen kan utföra begäran. Konversationsmönster 3 utför begäran på en gång genom att ta all information direkt från ett kommando. Den röststyrda applikationen som utvecklades i arbetet använder en kombination av konversationsmönster 1 och 3. Det nya konversationsmönstret förbättrar konversationsmönster 3 genom att använda en egenskap av konversationsmönster 1 för att hämta ytterligare information från användaren som saknas från användarens kommando. Detta görs genom att fråga användaren igen efter den saknade informationen. I figur 5.2 visas en dialog mellan en användare och applikationen som använder det nya konversationsmönstret.



Figur 5.2: Konversation mellan användare och digital assistent om en persons kalenderaktiviteter. Bilden är skapad av författarna med draw.io[64].

I figur 5.2 demonstreras hur kalenderapplikationen hanterar användarens kommando med hjälp av det nya konversationsmönstret. Applikationen som körs i assistenten kan söka och hämta kalenderhändelser från en persons schema och presentera dem till användaren. Applikationen följer konversationsmönster 3 genom att först ge en kort beskrivning till användaren om applikationens funktionalitet. Sedan uppger användaren sitt kommando till applikationen. Applikationen märker att någon information fattas från kommandot, vilket gör att applikationen behöver fråga efter informationen från användaren (som följer konversationsmönster 1). Efter att användaren uppger den saknade informationen kan applikationen slutföra kommandot.

### 5.3 Arkitekturmönster

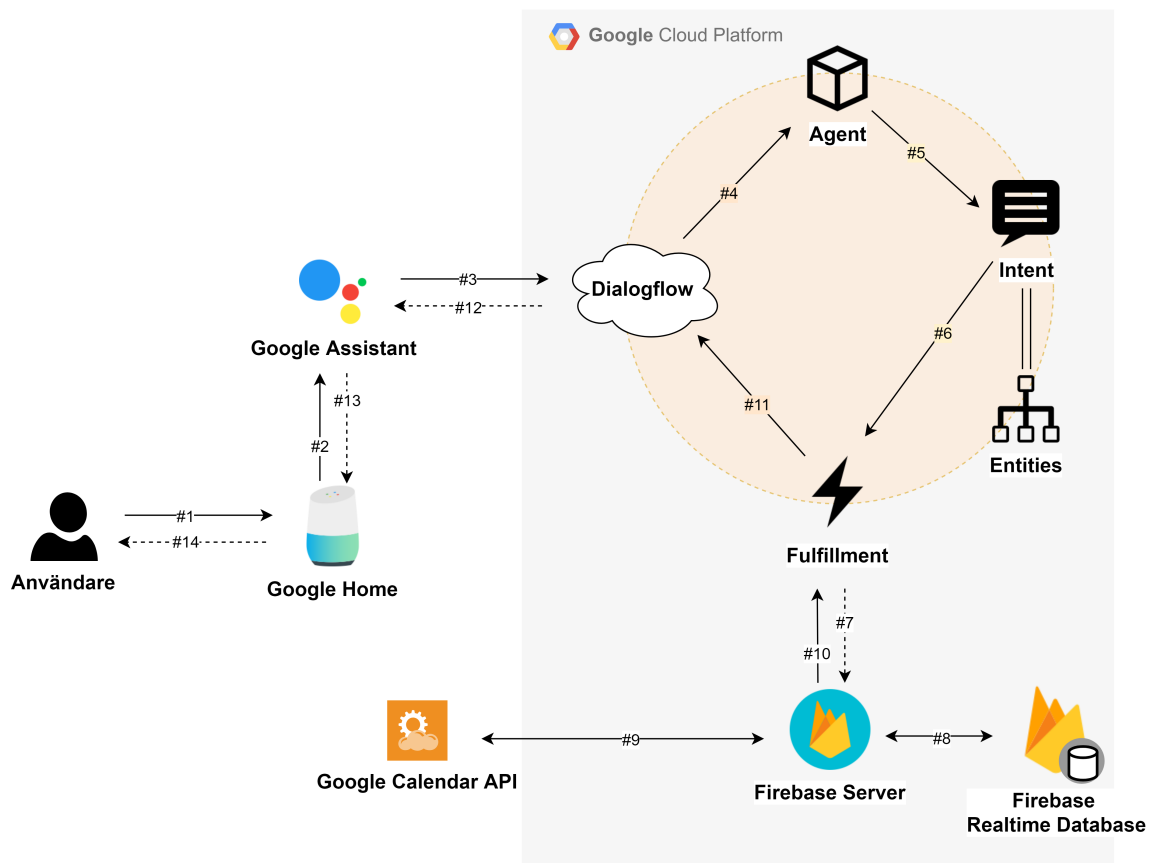
Projektets kalenderapplikation följde ett arkitekturmönster som bygger på att dela upp kodbasen i flera olika lager (Layer Architectural Pattern). Varje lager har en specifik uppgift vilket medför att koden blir strukturerad och lätt att hantera. Arkitekturmönstret är en mer generell variant av det välkända Model-View-Controller (MVC) mönstret. MVC bygger på att dela upp koden i lager som hanterar logiken (model), presentationen till användaren (view) och ett lager för att separera och kontrollera kommunikationen mellan dessa två lager (controller). Arkitekturmönstret som valdes till detta projekt var istället mer generellt med avseende på vilka lager som ska ingå. Koden är uppdelad i följande lager:

- Controller
- Model
- Integration
- Data

View-lagret som fanns med i MVC kan i detta fall abstraheras bort eftersom Dialogflow sköter all interaktion med användaren. Controllerns uppgift är att ta emot ett webhook anrop från Dialogflow och sedan anropa rätt funktioner i Model-lagret för att hantera anropet. Ansvar för att generera ett svar som kan skickas tillbaka till klienten ligger då hos Model. När anrop till en databas eller externt API behöver göras delegeras detta till Integration-lagret som är ansvarigt för kommunikation med yttre system. Data-lagret är själva databasen som används av Integration-lagret. Arkitekturmönstret som beskrivs ovan är flexibelt och fungerar för många olika applikationer. Vid behov kan lager tas bort, eller nya lager läggas till, beroende på hur applikationen som ska utvecklas ser ut.

## **5.4 Konceptuell arkitektur**

Den konceptuella arkitekturen togs fram genom en litteraturstudie av de olika systemen som ingår i en röststyrd applikation. En generell version av denna arkitektur presenterades i sektion 2.7 för att tidigt ge en översikt av hur helheten ser ut. Figur 5.3 visar en mer specifik version av denna arkitektur till Google Assistant.



Figur 5.3: Konceptuell arkitektur över hela kalenderapplikationen. Bilden är skapad av författarna.

Figur 5.3 visar systemets arkitektur. Det som skiljer denna arkitektur från den som presenterades i sektion 2.7 är att denna är specifik för kalenderapplikationen. Detta ger en mer detaljerad bild av systemarkitekturen än den generella versionen. Pilarna i figuren visar händelseförloppet när en applikation körs. Förloppet inleds med att användaren ger ett röstkommando till den röststyrda enheten (i detta fall en Google Home högtalare). Röstkommandot skickas vidare till Google Assistants molntjänst där det körs mjukvara för taligenkänning som konverterar talet till text. Denna text skickas sedan vidare till Dialogflow som representeras av den färgade cirkeln i figuren. Med hjälp av en agent, intents, entities och fulfillments tolkas kommandot. Med en webhook anropas servern för att hantera kommandot. I servern anropas Firebase databasen och Google Calendar API:et för att hämta nödvändig information. Sedan skickar servern ett textbaserat svar tillbaka till Dialogflow. Svaret skickas vidare till Google Assistants molntjänst där det konverteras från text till tal. Detta röstsvar skickas sedan till den smarta

enheten som spelar upp det för användaren.

## **5.5 Teknikvalens påverkan på programmeringsmodellen**

Valet av Dialogflow hade stor påverkan på programmeringsmodellen på grund av att utvecklingsprocessen förenklades. Behovet av att utveckla en egen NLU-motor finns inte om Dialogflow används. Med Dialogflow kunde intents skapas med ett webbgränssnitt istället för att koda med Actions SDK. Detta val gjorde att utvecklingsprocessen i programmeringsmodellen blev kortare och mer nybörjarvänlig. Valet av Node.js som exekveringsmiljö påverkade programmeringsmodellen på grund av kodbiblioteken som kan användas för att kommunicera med Dialogflow. Med kodbiblioteken behöver man bara sätta upp och använda en funktion för att ta emot webhook anrop från Dialogflow. Samma sak gäller det när man ska skicka ett svar tillbaka till Dialogflow då man bara behöver använda en funktion. Med detta val behövde man inte definiera funktioner för att hantera webhook anrop från Dialogflow vilket tog bort en extra kodningstid från utvecklingsprocessen.

## **5.6 Begränsningar med den använda teknologin**

Nedan presenteras en lista över alla begränsningar som stöttes på under utvecklingen av applikationen till Google Assistant som en utvecklare bör känna till. Begränsningarna diskuteras utförligare i sektion 6.3.

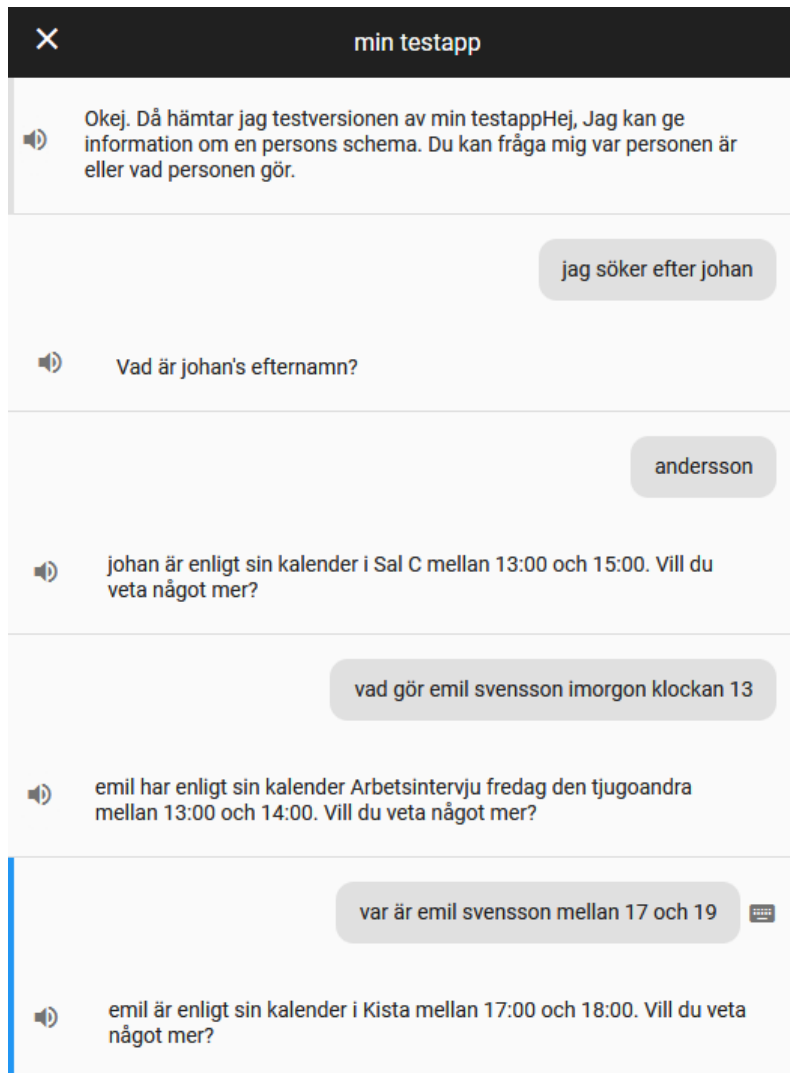
- Dialogflows NLU-motor kan ha problem med att känna igen svåruttalade personnamn i ett kommando.
- En applikation kan inte vara aktiv under en längre tidsperiod.
- Google Assistants taligenkänning kan ha problem med att känna igen vissa ord i ett röstkommando.
- Hantering av en fulfillment får maximalt ta fem sekunder.
- I nuläget finns det bara kodbibliotek till Node.js.

## 5.7 Applikationens funktionalitet

Eftersom att projektet använde åtagandetriangeln där funktionen var flexibel utvecklades inte all funktionalitet som fanns med i kravspecifikationen. Alla funktionella krav som hamnade under *måste ha* prioriterades högst i utvecklingsarbetet och uppfylldes. Den röststyrda applikationen kan hämta information från en persons kalender. Den kan också hantera olika frågor som handlar om var en person befinner sig, vad en person har planerat just nu eller vid någon annan tidpunkt. Denna funktionalitet uppfyller alla obligatoriska krav. Kravet som ligger under *bör ha* uppfylldes också. Detta innebär att applikationen kan hämta information från flera kalendrar. Genom att använda en databas kan informationen som krävs för att komma åt en kalender enkelt lagras vilket gör det möjligt att få åtkomst till olika kalendrar. Varje kalender-ID måste dock läggas till manuellt i databasen vilket medför skalningsproblem.

Applikationen uppfyller inte kraven som är listade under *kan ha* och *kommer inte ha*. Som förväntat låg denna funktionalitet tidsmässigt utanför projektets omfattning. Eftersom att dessa funktioner hade en låg prioritet ansågs avsaknaden av dem inte påverka projektets resultat. Figur 5.4 visar den färdiga applikationens funktionalitet.





Figur 5.4: Simulering av den färdiga applikationen. Bilden är skapad av författarna.

Figur 5.4 visar att kraven under *måste ha* och *bör ha* är uppfyllda genom en testkörning av applikationen i simulatorn.

## **6 Slutsatser och diskussion**

Detta kapitel diskuterar innehållet i rapporten. Sektion 6.1 diskuterar validiteten och reliabiliteten i metoderna som användes i arbetet. Sektion 6.2 behandlar validiteten och reliabiliteten i resultaten. Sektion 6.3 besvarar frågeställningarna utifrån resultaten som presenterades i kapitel 5. Sektion 6.4 går igenom hållbar utveckling och kopplar det till detta arbete. Sektion 6.5 diskuterar möjligheter för framtida arbete inom området.

### **6.1 Validitet och reliabilitet i metoden**

#### **Undersökningsmetoder**

Litteraturstudien användes för att samla teoretisk kunskap som var nödvändig för arbetet. Studien av digitala assistenter och dess bakomliggande teknologi samt relevanta tjänster och verktyg gav tillräcklig bakgrundsinformation för att svara på frågeställningarna. Utöver att användas för att besvara frågeställningarna gav litteraturstudien också nödvändig kunskap om olika tjänster och verktyg som användes för att kunna utföra fallstudien senare i arbetet. Litteraturstudien uppfyllde alltså sitt syfte eftersom att resultaten som krävdes för att besvara frågeställningarna levererades. Detta medför att valet av denna metod kan anses ha hög validitet. Gällande reliabilitet kan metoden bara anses vara pålitlig om källorna som används i studien är pålitliga. Kvalitetsgranskade forskningsrapporter och böcker användes som källor när det var möjligt för att öka reliabiliteten i metoden. Eftersom att digitala assistenter är relativt ny teknologi var detta dock inte alltid möjligt och webbkällor fick användas istället. Endast webbkällor som ansågs ha pålitliga författare användes för att inte negativt påverka studien reliabilitet. Vissa delar av informationen som samlades in från dessa källor stöttades sedan av kunskapen som fallstudien gav vilket stärker dess pålitlighet. Det är sannolikt att samma resultat skulle fås om någon annan använde samma metod förutsatt att teknologin som studeras inte genomgår stora förändringar i framtiden.

Fallstudien användes för att ge praktisk erfarenhet av att utveckla en röststyrd

applikation för att bättre kunna besvara frågeställningarna. En fallstudie ger en detaljerad bild av ett specifikt ämnesområde och var därför ett bra val för denna studie. Resultat som till exempel hur utvecklingsprocessen ser ut kan levereras med hjälp av den praktiska erfarenheten som fallstudien ger vilket gör att det finns validitet i metoden. Arbetet som utfördes i fallstudien dokumenterades grundligt i sektion 4 för att göra undersökningen transparent och därmed mer pålitlig. Denna tydliga beskrivning av varje steg gör det även möjligt för någon annan att använda arbetet som en referens vid skapande av ytterligare studier. Eftersom en fallstudie delvis bygger på personliga erfarenheter är det inte säkert att någon annan som använder samma metod får samma resultat. Detta är ur en reliabilitetssynpunkt en negativ aspekt av fallstudien. En möjlig förbättring till metoden skulle kunna vara att inkludera andra digitala assistenter och verktyg i studien för att få ett större underlag för att kunna besvara frågeställningarna.

Bunges vetenskapliga metod var en av undersökningsmetoderna som användes i projektet, vilket förklaras i sektion 3.1.4. Metoden användes i början av projektet för att veta vilka steg som ska tas i ett vetenskapligt arbete. Valet att använda metoden istället för någon annan vetenskaplig metod var beroende på att andra akademiska rapporter använde sig av den, vilket förstärkte metodens reliabilitet. Metoden var utformad för att först undersöka problemområdet och sedan ta fram och testa ett lösningsförslag. Detta var väldigt användbart i arbetet eftersom litteraturstudien gav bra kunskap om hur en röststyrd applikation skulle utvecklas. Med litteraturstudien blev utvecklandet av kalenderapplikationen i fallstudien smidigare och applikationen testades för att säkerställa att allting fungerade. Utifrån dessa steg producerades resultat som gav möjlighet för att svara på frågeställningarna som presenterades i sektion 1.2.

## **Projektmetoder**

Åtagandetriangeln som beskrivs i sektion 3.6 hade stor påverkan över kraven som sattes upp i projektet. Meningen med triangeln var att välja en av de tre hörnen som skulle vara flexibel för att ha större chans för lyckat projekt. Eftersom tiden och budgeten var begränsade blev funktion flexibel. Genom att ha funktionalitet som den flexibla hörnan blev det enklare att utföra projektet genom använda

MoSCoW metoden för att prioritera kraven. Från början fokuserade arbete på att uppfylla kraven som sattes upp i kategorin *måste ha* men till slut utfördes även kraven i *bör ha*. Med prioriteringstekniken lyckades arbetet få ut resultat för att besvara frågeställningarna.

Utifrån kraven blev det möjligt att arbeta agilt i form av iterationer, vilket beskrivs i sektion 3.7. Varje iteration som utfördes i arbetet hade som mål att antingen bidra till eller uppfylla ett krav. Om ett krav inte kunde uppfyllas under en iteration blev det lätt att gå tillbaka till förgående iterationer som var kopplade till kravet och utföra förändringar. Ett exempel var när mer information behövdes om ett verktyg som användes i utvecklingen av kalenderapplikationen, vilket resulterade i att gå tillbaka till litteraturstudien och samla mer information om verktyget.

## 6.2 Validitet och reliabilitet i resultaten

För att besvara hur programmeringsmodellen ser ut för röststyrda applikationer krävdes fyra artefakter. Dessa var utvecklingsprocessen, konversationsmönster, arkitekturmönster och en konceptuell arkitektur. Den utvecklingsprocess som redovisas i sektion 5.1 är utformad utifrån utvecklingsarbetet där Dialogflow används, men det är möjligt att generalisera den till att även passa andra projekt som använder andra verktyg. Därför kan den artefakten anses vara ett bra resultat som kan användas för att besvara frågeställningen. Både konversations- och arkitekturmönstren som redovisas är generella och inte kopplade till någon specifik hårdvara eller tjänst. Dessa resultat anses också därför också vara bra nog för att kunna besvara frågeställningen. Den konceptuella arkitekturen är starkt kopplad till Google Assistant och Dialogflow eftersom dessa användes i fallstudien. Detta resultat kan dock generaliseras för att passa andra plattformar som till exempel Alexa vilket medför att resultatet anses vara bra nog för att besvara frågeställningen. Eftersom att alla dessa artefakter är bra nog kan programmeringsmodellen redovisas och därmed besvara frågeställningen.

När det gäller pålitligheten i resultaten som utgör programmeringsmodellen så finns det en del svagheter. Trots att de konversations- och arkitekturmönster som

valdes för detta arbete var menade för att kunna passa olika typer av applikationer finns det inga mönster som passar alla möjliga applikationer. Därför kan det finnas situationer där de mönster som presenteras i denna rapport skulle fungera väldigt dåligt, eller inte fungera alls. En annan potentiell svaghet är att den konceptuella arkitekturen som presenteras kan bli inaktuell om teknologin den bygger på skulle förändras i framtiden.

Begränsningarna som presenterades i sektion 5.6 anses ha hög reliabilitet men är inte valida nog för att besvara en av frågeställningarna. Reliabiliteten är hög eftersom begränsningarna togs ut från testningsfaserna som utfördes under utvecklingen av kalenderapplikationen. Varje begränsning testades flera gånger för att försäkra om att de inte bara uppstod en gång. Vissa av begränsningarna kontrollerades även genom att söka i olika internetforum och undersöka om andra personer stötte på samma begränsningar. Validiteten av begränsningarna kan ses som bristande eftersom de bara har hittats i verktyg till Google Assistant och inte andra plattformar såsom Alexa och Cortana. Därför kan inte begränsningarna generaliseras för att kunna gälla för flera plattformar än Google Assistant.

Resultaten som presenterades i sektion 5.5 handlar om hur utvecklingsverktygen för Google Assistant påverkade programmeringsmodellen för röststyrda applikationer. Detta medför att bara verktyg för Google Assistant har använts i fallstudien, vilket inte uppfyller kravet av att svara på frågeställningen som handlar om resurser och ramverk. Med resultaten kan bara en del av frågeställningen besvaras eftersom resultaten för resurser och ramverk är bara specifik för Google Assistant och kan inte generaliseras för att täcka andra digitala assistenter.

### **6.3 Besvarande av frågeställningar**

Utifrån resultaten i sektion 5 kan slutsatser dras för att besvara frågeställningarna.

## **Hur ser programmeringsmodellen ut för applikationer som använder ett röst- och ljudinterface?**

Programmeringsmodellen består av fyra artefakter. Den första artefakten är utvecklingsprocessen, den andra är designen av applikationens röstgränssnitt, den tredje är kodens arkitekturmönster och den fjärde är en konceptuell arkitektur. För att besvara denna frågeställning behöver hela programmeringsmodellen som presenterades i sektion 5 generaliseras för att också fungera med röststyrda applikationer till andra digitala assistenter.

Utvecklingsprocessen som presenterades i sektion 5.1 är utformad för användning med Google Cloud Platform och Dialogflow som utvecklingsverktyg. Denna process kan generaliseras genom att byta ut alla moment som berör Google och Dialogflow med motsvarande verktyg för andra digitala assistenter. Detta kan till exempel vara Actions SDK eller någon annan plattform som används för att utveckla chattbotar. Om ingen server är nödvändig för applikationen som ska utvecklas kan serverdelen i utvecklingsprocessen utelämnas. Notera att terminologi som intents och fulfillments som används för att beskriva utvecklingsprocessen inte nödvändigtvis är korrekta vid användning av andra verktyg än Dialogflow.

Konversationsmönstret för röstgränssnittet som presenterades i sektion 5.2 är redan generellt eftersom ingen specifik utvecklingsplattform är direkt kopplat med det. Arkitekturmönstret som presenterades i 5.3 är också generellt och kan appliceras på kod som skrivs även för andra plattformar än Google Assistant.

Den konceptuella arkitekturen som presenterades i sektion 2.7 visar hur systemarkitekturen ser ut för applikationer som använder ett röstgränssnitt. Denna arkitektur är generell och passar därför in på flera olika digitala assistenter som Google Assistant, Alexa, Cortana och Siri. För Google Assistant gäller också resultatet som presenterades i sektion 5.4, vilket beskriver hur arkitekturen ser ut för en applikation som använder Dialogflow och Firebase.

Vi tycker att de ovannämnda artefakterna var de som krävdes för att beskriva programmeringsmodellen. Det kan argumenteras för att en designmodell i form av ett komponentdiagram av källkoden också bör vara en artefakt

som ingår i programmeringsmodellen. Vi gjorde bedömningen att en sådan designmodell skulle vara för applikationsspecifik för att tillföra något till en programmeringsmodellen som är skapad för att vara generell. Därför inkluderades inte designmodellen som en artefakt i programmeringsmodellen. Istället användes designmodellen endast i sektion 4.3 för att förtydliga hur serverkoden fungerar.

### **Vilka begränsningar finns vid utveckling av en röststyrd applikation?**

Dialogflow har system entities som en utvecklare kan använda för att känna igen ord som Google Assistant ska plocka upp. Några av Dialogflows entities är förnamn, efternamn, färg, datum, tid. Dessa entities är bra på att känna igen olika varianter av en sak. Ett exempel är att tid entity kan känna igen tidpunkt såsom "just nu", "imorgon", "om en vecka" och "förrgår". Men det finns andra entities som har svårt att plocka upp ord som tillhör den entityn. En av dem är förnamn och efternamn. Svenska namn kan enkelt plockas upp av assistenten men flera "ovanliga" namn identifieras inte som till exempel "Drishti Diem". En smidig lösning till problemet är att skapa egen entity som tar emot specifika namn. Eftersom Dialogflow har funktionen att inkludera en entity i en annan entity kan Dialogflows inbyggda entities inkluderas i nya entities. Då kan assistenten identifiera både vanliga och specifika namn och mappa dem till en entity.

Applikationen kan inte vara aktiv i en längre period eftersom applikationen måste avslutas när användaren har pratat klart med assistenten. Det går heller inte att programmatiskt stänga av mikrofonen i enheten som applikationen körs i för att den ska vara aktiv i bakgrunden. Användaren måste själv stänga av mikrofonen om man vill sluta prata med applikationen och ha den aktiv i enheten.

Det är viktigt att inte leda en användare till att säga svåruttalade ord när användaren pratar med applikationen. Vissa svenska ord är svåra för Google Assistant att plocka upp när användaren säger dem i en mening. Om till exempel användaren bara säger ordet "internalisera" till assistenten plockas det upp direkt men om användaren säger ordet i en mening plockas ordet upp som "inte analysera".

Dialogflow har en 5 sekunders timeout på hur långt tid ett webhook anrop kan ta. Om tiden överstigs kommer Dialogflow skicka tillbaka ett standardsvar till användaren om att röstkommandot tog för lång tid att hantera. För att hantera denna begränsning kan servern skicka tillbaka ett svar till användaren som säger hur långt tid kommandot kommer ta att bearbeta. Sedan kan användaren uppge ett annat kommando som hämtar svaret till det första kommandot. I vanliga fall är det bättre att optimera serverkoden för att den ska skicka ett svar i första försöket för att konversationen med användaren ska flyta på.

I nuläget har Dialogflow bara kodbibliotek till Node.js. Det blir svårare att hantera webhook anrop från Dialogflow om man utvecklar med ett annat programmeringsspråk än JavaScript. Det finns inofficiella kodbibliotek till andra språk men det är inte garanterat att de är lika pålitliga som Dialogflows officiella kodbibliotek.

### **Vilka resurser och ramverk finns att tillgå?**

För utveckling av röststyrda applikationer behövs en NLU-komponent i form av en chattbot. Det finns plattformar för att göra denna process smidig för utvecklare genom att eliminera behovet av att utveckla egen NLU-funktionalitet. Tjänster som Dialogflow, Alexa Skills Kit och Microsoft Bot Framework kan användas för att skapa en chattbot. Valet av tjänst kan bero på en mängd olika faktorer som kostnad, funktionalitet och med vilken digital assistent applikationen ska integreras med.

Om applikationen ska utvecklas till Google Assistant finns tre olika alternativa resurser som kan användas, Dialogflow, templates och Actions SDK. Templates är lämpligt för enkla ändamål och kräver väldigt lite teknisk kunskap. Motsatsen till templates skulle då kunna vara Actions SDK vilket ger utvecklaren väldigt mycket kontroll och väldigt lite hjälp. Actions SDK är lämplig om utvecklaren vill skapa en egen NLU-motor. För de allra flesta applikationer borde Dialogflow vara det mest lämpliga verktyget. Det grafiska gränssnittet är intuitivt och lätt att använda och ger samtidigt möjligheterna att skapa avancerade applikationer.

Dialogflow gör det väldigt lätt att skapa chattbotar men erbjuder även



kodbibliotek för att underlätta programmeringen av den bakomliggande logiken. Med Dialogflows kodbibliotek blir det enkelt att generera meddelanden som sedan skickas tillbaka till Dialogflow. Det finns endast officiella bibliotek för Node.js men det går att hitta bibliotek från tredje part till exempelvis Python och Java.

## 6.4 Hållbar utveckling

Brundtlandrapporten definierar hållbar utveckling som *”utveckling som tillgodoser dagens behov utan att äventyra kommande generationers möjligheter att tillgodose sina behov”* [102]. Detta innebär att samhället ska klara av att uppfylla mänskliga behov med de begränsade resurser som naturen ger utan att skada miljön och ekosystem. Förenta Nationerna har definierat 17 mål för att uppnå social, ekonomisk och ekologisk hållbar utveckling [103]. Mål nummer 9 handlar om att främja innovation och teknologiska framsteg [104]. Eftersom att syftet med arbetet är att förbättra läsarens förmåga att utveckla röststyrda applikationer så bidrar det med att öppna en väg för utveckling av innovativa röststyrda applikationer.

En annan positiv aspekt av röststyrda applikationer är att det möjliggör för personer med synnedsättning att utnyttja fler av nuvarande mjukvarutjänster, som beskrivet i sektion 1.5. Detta främjar social hållbarhet genom att underlätta deras vardag. En negativ aspekt av röststyrda applikationer är att de är beroende av uppkoppling till internet vilket bidrar till en större energiförbrukning. Materialet som krävs för att skapa röststyrda enheter kan vara svårt att återvinna vilket kan bidra negativt till den ekologiska hållbarheten.

## 6.5 Framtida arbete

Framtida arbete skulle kunna innefatta ytterligare studier av utveckling av applikationer till andra digitala assistenter. Detta skulle ge kunskap både om hur utvecklingsprocessen ser ut för andra plattformar och vilka skillnader det finns mellan olika digitala assistenter. Det finns även möjligheter för framtida arbete

som berör Google Assistant. Detta skulle kunna vara en fallstudie som använder Actions SDK istället för Dialogflow. Då skulle fokus istället ligga på hur NLU-funktionalitet kan utvecklas.

I arbetet undersöktes inte hur man sätter upp ett testramverk för en röststyrd applikation. Detta innefattar uppsättning av testfall som en applikation ska klara av och hur dessa fall kan automatiseras i en testmiljö. Att undersöka detta är ett möjligt framtida arbete.

## Referenser

- [1] Naomi van der Velde. *Innovative Uses of Speech Recognition Today*. URL: <https://www.globalme.net/blog/new-technology-in-speech-recognition> (besöktes 2019-01-04).
- [2] *Voice User Interfaces*. URL: <https://www.interaction-design.org/literature/topics/voice-user-interfaces> (besöktes 2019-01-04).
- [3] *IBM Shoebox*. URL: [http://www-03.ibm.com/ibm/history/exhibits/specialprod1/specialprod1\\_7.html](http://www-03.ibm.com/ibm/history/exhibits/specialprod1/specialprod1_7.html) (besöktes 2019-01-04).
- [4] Margaret Rouse. "Virtual assistant (AI assistant)". I: (2014). URL: <https://searchcrm.techtarget.com/definition/virtual-assistant> (besöktes 2019-01-04).
- [5] Vivek Sharma. "How do digital voice assistants (e.g. Alexa, Siri) work?" I: (okt. 2017). URL: <https://www.marshall.usc.edu/blog/how-do-digital-voice-assistants-eg-alexa-siri-work> (besöktes 2019-01-04).
- [6] *Google Assistant*. URL: <https://assistant.google.com/> (besöktes 2019-01-04).
- [7] *Use Siri on all your Apple devices*. URL: <https://support.apple.com/en-us/HT204389> (besöktes 2019-01-04).
- [8] *Amazon Alexa*. URL: <https://developer.amazon.com/alexa> (besöktes 2019-01-04).
- [9] *Use Cortana across your devices*. URL: <https://www.microsoft.com/en-us/cortana/devices/> (besöktes 2019-01-04).
- [10] *Here's what people actually use their Amazon Echo and other smart speakers for*. URL: [https://www.cnn.com/2018/09/10/adobe-analytics-what-people-use-amazon-echo-and-smart-speakers-for.html?fbclid=IwAR3sxfxEYMcfuUBVEpChDdg3\\_1r19QGvLQCn9F\\_dKmvHBQ\\_Q4McnzXGwqVk](https://www.cnn.com/2018/09/10/adobe-analytics-what-people-use-amazon-echo-and-smart-speakers-for.html?fbclid=IwAR3sxfxEYMcfuUBVEpChDdg3_1r19QGvLQCn9F_dKmvHBQ_Q4McnzXGwqVk) (besöktes 2019-01-04).

- [11] Jenny Medeiros. “Voice Assistants are Changing How Users with Disabilities Get Things Done”. I: (2018). URL: <https://www.modev.com/blog/voice-assistants-are-changing-how-users-with-disabilities-get-things-done> (besöktes 2019-01-04).
- [12] *Universal Declaration of Human Rights*. URL: <http://www.un.org/en/universal-declaration-human-rights/> (besöktes 2019-01-04).
- [13] Ben Dickson. *Beware the privacy and security risks of smart speakers*. 2018. URL: <https://bdtechtalks.com/2018/06/05/google-home-amazon-echo-privacy-security-risks/> (besöktes 2019-01-04).
- [14] *Olovlig avlyssning*. URL: [https://lagen.nu/begrepp/Olovlig\\_avlyssning](https://lagen.nu/begrepp/Olovlig_avlyssning) (besöktes 2019-01-04).
- [15] Anne Håkansson. “Portal of research methods and methodologies for research projects and degree projects”. I: *The 2013 World Congress in Computer Science, Computer Engineering, and Applied Computing WORLDCOMP 2013; Las Vegas, Nevada, USA, 22-25 July*. CSREA Press USA. 2013, s. 67–73.
- [16] Mike Moore. “What is AI? Everything you need to know”. I: (sept. 2018). URL: <https://www.techradar.com/news/what-is-ai-everything-you-need-to-know> (besöktes 2019-01-04).
- [17] Peter Norvig Stuart J. Russell. *Artificial Intelligence: A Modern Approach*. 3rd ed. Pearson, 2009. ISBN: 978-0-13-604259-4.
- [18] Dr Zafar. M. Alvi. “Artificial Intelligence”. I: (2014). URL: [http://vulms.vu.edu.pk/Courses/CS607/Downloads/AI\\_Complete\\_handouts\\_for\\_Printing.pdf](http://vulms.vu.edu.pk/Courses/CS607/Downloads/AI_Complete_handouts_for_Printing.pdf) (besöktes 2019-01-04).
- [19] *Natural Language Processing*. URL: [https://www.sas.com/en\\_us/insights/analytics/what-is-natural-language-processing-nlp.html](https://www.sas.com/en_us/insights/analytics/what-is-natural-language-processing-nlp.html) (besöktes 2019-01-04).
- [20] Judith Hurwitz och Daniel Kirsch. *Machine Learning for dummies*. IBM Limited ed. John Wiley & Sons, Inc., 2018. ISBN: 978-1-119-45494-6 (electronic bk.)

- [21] *Narrow Artificial Intelligence (Narrow AI)*. URL: <https://www.techopedia.com/definition/32874/narrow-artificial-intelligence-narrow-ai> (besöktes 2019-01-04).
- [22] *Strong Artificial Intelligence (Strong AI)*. URL: <https://www.techopedia.com/definition/31622/strong-artificial-intelligence-strong-ai> (besöktes 2019-01-04).
- [23] Jack Krupansky. “How Close Is AI to Human-level Intelligence Here in April 2018?” I: (april 2018). URL: <https://medium.com/@jackkrupansky/how-close-is-ai-to-human-level-intelligence-here-in-april-2018-9a6ceaff2f9d> (besöktes 2019-01-05).
- [24] J. M. Korhonen. “A “strong” AI might be impossible – otherwise we’d seen one by now”. I: (maj 2017). URL: <https://jmkorhonen.net/2017/05/05/a-strong-ai-might-be-impossible-otherwise-wed-seen-one-by-now/> (besöktes 2019-01-04).
- [25] Alyona Medelyan. “8 examples of Natural Language Processing you use every day without noticing”. I: (dec. 2016). URL: <https://www.linkedin.com/pulse/8-examples-natural-language-processing-you-use-every-day-medelyan> (besöktes 2019-01-04).
- [26] Roberto Navigli. *Natural Language Understanding: Instructions for (Present and Future) Use*. URL: <https://www.ijcai.org/proceedings/2018/0812.pdf> (besöktes 2019-01-04).
- [27] Bill MacCartney. *Understanding Natural Language Understanding*. Juli 2014. URL: <https://nlp.stanford.edu/~wcmac/papers/20140716-UNLU.pdf> (besöktes 2019-02-04).
- [28] Natalia Konstantinova. “Review of Relation Extraction Methods: What Is New Out There?” I: vol. 436. April 2014. DOI: 10.1007/978-3-319-12580-0\_2.
- [29] Bill MacCartney. *Introduction to semantic parsing*. Maj 2016. URL: <https://web.stanford.edu/class/cs224u/materials/cs224u-2016-intro-semparse.pdf> (besöktes 2019-02-03).

- [30] Bing Liu. “Sentiment analysis and opinion mining”. I: *Synthesis lectures on human language technologies* 5.1 (2012), s. 7.
- [31] Wouter Gevaert, Georgi Tsenov och Valeri Mladenov. “Neural networks used for speech recognition”. I: *Journal of Automatic Control* 20 (jan. 2010). DOI: 10.2298/JAC1001001G.
- [32] Françoise Beaufays. “The neural networks behind Google Voice transcription”. I: (aug. 2015). URL: <https://ai.googleblog.com/2015/08/the-neural-networks-behind-google-voice.html> (besöktes 2019-02-05).
- [33] John Bateman och Michael Zock. “Natural Language Generation”. I: *The Oxford Handbook of Computational Linguistics* (jan. 2012). DOI: 10.1093/oxfordhb/9780199276349.013.0015. (besöktes 2019-01-19).
- [34] Dan Grabham Maggie Tillman. “What is Google Assistant and what can it do?” I: (okt. 2018). URL: <https://www.pocket-lint.com/apps/news/google/137722-what-is-google-assistant-how-does-it-work-and-which-devices-offer-it> (besöktes 2019-01-04).
- [35] *Smart Displays with the Google Assistant*. URL: <https://assistant.google.com/platforms/displays> (besöktes 2019-02-07).
- [36] *Talk to the Google Assistant in multiple languages*. URL: <https://support.google.com/googlehome/answer/7550584?hl=en> (besöktes 2019-01-04).
- [37] *Google Assistant SDK*. URL: <https://developers.google.com/assistant/sdk/overview> (besöktes 2019-01-04).
- [38] *Actions for the Google Assistant*. URL: <https://developers.google.com/actions/extending-the-assistant> (besöktes 2019-01-04).
- [39] “Amazon Echo: Everything you need to know about the listening speaker”. I: (). URL: <https://www.wareable.com/features/amazon-echo-price-specs-release-date-667> (besöktes 2019-01-04).
- [40] *Echo, Echo Plus & Echo Dot*. URL: <https://developer.amazon.com/echo> (besöktes 2019-01-04).
- [41] *avs-device-sdk*. URL: <https://developer.amazon.com/alexa-voice-service/sdk> (besöktes 2019-01-04).

- [42] *Develop Skills in Multiple Languages*. URL: <https://developer.amazon.com/docs/custom-skills/develop-skills-in-multiple-languages.html> (besöktes 2019-01-04).
- [43] Luke Dormehl. “Today in Apple history: Siri debuts on iPhone 4s”. I: (okt. 2018). URL: <https://www.cultofmac.com/447783/today-in-apple-history-siri-makes-its-public-debut-on-iphone-4s/> (besöktes 2019-01-05).
- [44] *SiriKit*. URL: <https://developer.apple.com/documentation/sirikit> (besöktes 2019-01-05).
- [45] *HomePod*. URL: <https://www.apple.com/homepod/specs/> (besöktes 2019-01-05).
- [46] *Apple Siri*. URL: <https://www.apple.com/siri/> (besöktes 2019-01-05).
- [47] *Microsoft Cortana*. URL: <https://www.microsoft.com/en-us/cortana> (besöktes 2019-01-05).
- [48] Jez Corden. “A brief history of Cortana, Microsoft’s trusty digital assistant”. I: (april 2017). URL: <https://www.windowscentral.com/history-cortana-microsofts-digital-assistant> (besöktes 2019-01-05).
- [49] *Integrate Cortana Today*. URL: <https://developer.microsoft.com/en-us/cortana/devices> (besöktes 2019-01-05).
- [50] *Cortana Skills Kit FAQ*. URL: <https://docs.microsoft.com/en-us/cortana/skills/faq> (besöktes 2019-01-05).
- [51] *Google Cloud Platform Overview*. URL: <https://cloud.google.com/docs/overview/> (besöktes 2019-01-19).
- [52] Tom White. *Hadoop: The Definitive Guide*. O’Reilly Media, 2009, s. 3–4. ISBN: 978-0-596-52197-4.
- [53] *About the GCP Services*. URL: <https://cloud.google.com/docs/overview/cloud-platform-services> (besöktes 2019-01-19).
- [54] *Google Cloud Platform Services Summary*. URL: <https://cloud.google.com/terms/services> (besöktes 2019-01-19).

- [55] *Pricing*. URL: <https://cloud.google.com/pricing/> (besöktes 2019-01-19).
- [56] *Templates*. URL: <https://developers.google.com/actions/templates/> (besöktes 2019-01-04).
- [57] *Actions Basics*. URL: <https://developers.google.com/actions/sdk/> (besöktes 2019-01-04).
- [58] *Dialogflow*. URL: <https://dialogflow.com> (besöktes 2019-01-04).
- [59] *Actions Overview*. URL: <https://developers.google.com/actions/dialogflow/> (besöktes 2019-01-04).
- [60] *Agents Overview*. URL: <https://dialogflow.com/docs/agents> (besöktes 2019-01-04).
- [61] Dialogflow. *Image by: Dialogflow*. Creative Commons Attribution 3.0 License. URL: <https://dialogflow.com/docs/agents> (besöktes 2019-01-04).
- [62] *Training phrases*. URL: <https://dialogflow.com/docs/intents/training-phrases> (besöktes 2019-01-15).
- [63] *How fulfillment works*. URL: <https://dialogflow.com/docs/fulfillment/how-it-works> (besöktes 2019-01-04).
- [64] *draw.io*. URL: <https://about.draw.io>.
- [65] *Alexa Skills Kit*. URL: <https://developer.amazon.com/alexa-skills-kit> (besöktes 2019-02-06).
- [66] *Microsoft Bot Framework*. URL: <https://dev.botframework.com> (besöktes 2019-02-06).
- [67] *About V8*. URL: <https://v8.dev/docs> (besöktes 2019-01-04).
- [68] Mike Cantelon och Marc Harter och T.J. Holowaychuk och Nathan Rajlich. *Node.js in Action*. 2014. ISBN: 9781617290572.
- [69] *About NPM*. URL: <https://docs.npmjs.com/about-npm/> (besöktes 2019-01-05).



- [70] *Promise*. URL: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) (besöktes 2019-01-05).
- [71] Chris Esplin. “What is Firebase?” I: (2016). URL: <https://howtofirebase.com/what-is-firebase-fcb8614ba442> (besöktes 2019-01-05).
- [72] *Firebase CLI Reference*. URL: <https://firebase.google.com/docs/cli/> (besöktes 2019-01-05).
- [73] *Firebase Realtime Database*. URL: <https://firebase.google.com/docs/database/> (besöktes 2019-01-05).
- [74] *Cloud Functions for Firebase*. URL: <https://firebase.google.com/docs/functions> (besöktes 2019-02-03).
- [75] *Write and deploy your first functions*. URL: <https://firebase.google.com/docs/functions/get-started> (besöktes 2019-02-03).
- [76] *What Are RESTful Web Services?* Oracle. URL: <https://docs.oracle.com/javase/6/tutorial/doc/gijqy.html> (besöktes 2019-01-05).
- [77] *Get Started with the Calendar API*. Google. URL: <https://developers.google.com/calendar/overview> (besöktes 2019-01-05).
- [78] Troy Davis. “What is a WebHook?” I: (2011). URL: <https://webhooks.pbworks.com/w/page/13385124/FrontPage> (besöktes 2019-01-05).
- [79] *GitHub Facts*. 2018. URL: <https://github.com/about/facts> (besöktes 2019-01-05).
- [80] “Microsoft to acquire GitHub for \$7.5 billion”. I: (2018). URL: <https://news.microsoft.com/2018/06/04/microsoft-to-acquire-github-for-7-5-billion/> (besöktes 2019-01-05).
- [81] “Introducing Google Drive... yes, really”. I: (2012). URL: <https://drive.googleblog.com/2012/04/introducing-google-drive-yes-really.html> (besöktes 2019-01-05).

- [82] Jukka Paakki m. fl. "Software metrics by architectural pattern mining". I: *Proceedings of the International Conference on Software: Theory and Practice (16th IFIP World Computer Congress)*. Kluwer Beijing, China. 2000, s. 325–332.
- [83] Leif Lindbäck. *A First Course in Object Oriented Development*. Maj 2018, s. 51–56. URL: <http://leiflindback.se/iv1350/object-oriented-development.pdf> (besöktes 2019-02-06).
- [84] Neal Ford. "Contrasting architecture patterns with design patterns". I: (2015). URL: <https://www.oreilly.com/ideas/contrasting-architecture-patterns-with-design-patterns> (besöktes 2019-01-05).
- [85] Erich Gamma m. fl. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994. ISBN: 0201633612.
- [86] Jiwon Paik Joe Kappes. "Uncovering Voice UI Design Patterns". I: (aug. 2017). URL: <https://www.cooper.com/journal/2017/8/uncovering-voice-ui-design-patterns> (besöktes 2019-01-05).
- [87] Gaurav Kumar och Pradeep Kumar Bhatia. "Impact of agile methodology on software development process". I: *International Journal of Computer Technology and Electronics Engineering (IJCTEE)* 2.4 (2012), s. 46–50.
- [88] Mary Lotz. "Waterfall vs. Agile: Which is the Right Development Methodology for Your Project?" I: (maj 2013). URL: <https://www.seguetech.com/waterfall-vs-agile-methodology/> (besöktes 2019-01-05).
- [89] Emerson Taymor. *Agile Handbook*. Philosophie. URL: <http://agilehandbook.com/agile-handbook.pdf> (besöktes 2019-01-05).
- [90] Duncan Haughey. "MOSCOW METHOD". I: (2017). URL: <https://www.projectsmart.co.uk/moscow-method.php> (besöktes 2019-01-05).
- [91] *Build Actions for the Google Assistant (Level 1)*. URL: <https://codelabs.developers.google.com/codelabs/actions-1/#0> (besöktes 2019-01-05).

- [92] *Build Actions for the Google Assistant (Level 2)*. URL: <https://codelabs.developers.google.com/codelabs/actions-2/#0> (besöktes 2019-01-05).
- [93] *Build Actions for the Google Assistant (Level 3)*. URL: <https://codelabs.developers.google.com/codelabs/actions-3/#0> (besöktes 2019-01-05).
- [94] Dirk Schnelle och Fernando Lyardet. "Voice User Interface Design Patterns." I: *EuroPLoP*. 2006, s. 287–316.
- [95] Martyn. Denscombe. *The good research guide: for small-scale social research projects*. 4th ed. Maidenhead: Open University Press, 2010. ISBN: 978-0-335-24140-8 (electronic bk.)
- [96] Niclas Andersson och Anders Ekholm. *Vetenskaplighet - Utvärdering av tre implementeringsprojekt inom IT Bygg & Fastighet 2002*. swe. Tekn. rapport. Institutionen för Byggande och Arkitektur, Lunds Universitet, 2002. URL: [http://www.lth.se/fileadmin/projekteringsmetodik/research/Other\\_projects/utvarderingver1.pdf](http://www.lth.se/fileadmin/projekteringsmetodik/research/Other_projects/utvarderingver1.pdf) (besöktes 2019-01-22).
- [97] Mario Bunge. *Epistemology & Methodology I:: Exploring the World (Treatise on Basic Philosophy) (Volume 5)*. Springer, 1983. ISBN: 9027715238.
- [98] Sven Eklund. *Arbeta I Projekt*. 4th ed. 2011. ISBN: 9789144072753.
- [99] *Create and manage an agent*. URL: <https://dialogflow.com/docs/agents/create-manage> (besöktes 2019-02-07).
- [100] Johan Andersson Evan Saboo. *Röststyrd kalenderhämtnings applikation*. 2018. URL: <https://github.com/Joh4nAndersson/DA-Applikation>.
- [101] *Calendar API Reference*. URL: <https://developers.google.com/calendar/v3/reference/> (besöktes 2019-02-07).
- [102] "Hållbar utveckling". I: (dec. 2018). URL: <https://www.kth.se/om/miljo-hallbar-utveckling/utbildning-miljo-hallbar-utveckling/verktygslada/sustainable-development> (besöktes 2019-02-03).

- [103] *Globala Målen*. URL: <http://www.globalamalen.se/om-globala-malen> (besöktes 2019-01-02).
- [104] *Mål 9: Hållbar industri, innovationer och infrastruktur*. URL: <http://www.globalamalen.se/om-globala-malen/mal-9-hallbar-industri-innovationer-och-infrastruktur/> (besöktes 2019-01-02).

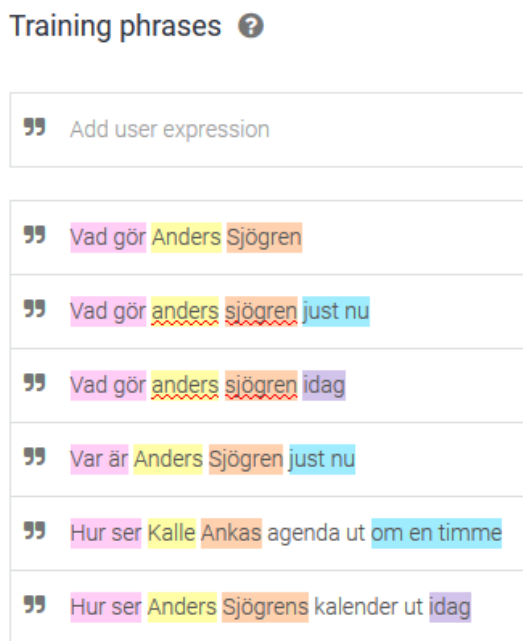
## **7 Bilagor**

## **Bilagor - Innehåll**

<b>A</b>	<b>Fullständig konfiguration av dialogflow</b>	<b>97</b>
<b>B</b>	<b>Guider</b>	<b>101</b>
<b>C</b>	<b>Google Home</b>	<b>102</b>

## A Fullständig konfiguration av dialogflow

I denna bilaga visas fortsättningen på konfigurationen av Dialogflow från sektion 4.1.



Figur A.1: Request-Intent träningsfraser del 2. Bilden är skapad av författarna.

Figur A.1 visar andra delen av träningsfraserna som användes för Request-Intent. Varje träningsfras är ett exempel på kommandon som användaren kan ge. Dialogflow använder dessa fraser för att träna sig själv till att kunna tolka fler möjliga kommandon från användaren.

The screenshot shows the 'Responses' and 'Fulfillment' sections of the Google Assistant configuration interface. The 'Responses' section is active, showing a 'Text response' configuration. It includes a list of responses with the first one being 'Något gick fel med förfrågan, försök igen.' and the second being 'Enter a text response variant'. There is an 'ADD RESPONSES' button and a toggle for 'Set this intent as end of conversation'. The 'Fulfillment' section is below, with a toggle for 'Enable webhook call for this intent' and a disabled toggle for 'Enable webhook call for slot filling'.

Figur A.2: Request-Intent responses. Bilden är skapad av författarna.

Figur A.2 visar de två sista delarna av Request-Intent konfigurationen. Den första heter Responses och är standardsvaret som Dialogflow skickar tillbaka om användarens kommando matchar till Request-Intent. Den andra delen heter "Fulfillments" vilket möjliggör användningen av webhooks för att skicka användarens kommando till en server. Om webhook anrop är aktiverat (visat i figuren) används inte standardsvaret i "Responses" utan istället skickas användarens kommando vidare till den kopplade servern. Men om servern inte lyckas skicka tillbaka ett svar till Dialogflow skickas istället standardsvaret till användaren.

The screenshot shows the 'Request-entity' configuration interface. It has a title 'Request-entity' and two checkboxes: 'Define synonyms' (checked) and 'Allow automated expansion' (unchecked). Below this is a table with two rows. The first row is for the entity 'Doing' with synonyms 'Vad gör, Va gör, Hur ser, Hu ser'. The second row is for the entity 'Where' with synonyms 'Vart är, Var är, vart finns, var finns, var hittas, var hittas'.

Entity	Synonyms
Doing	Vad gör, Va gör, Hur ser, Hu ser
Where	Vart är, Var är, vart finns, var finns, var hittas, var hittas

Figur A.3: Request-Entity. Bilden är skapad av författarna.

I figur A.3 visas konfigurationen av Request-Entity. Denna entity har två entity typer. Dessa är "Doing" och "Where". Entity typerna kontrollerar om användaren vill veta **var** eller **vad** en person gör för att skicka tillbaka ett passande svar.



first-name
<input type="checkbox"/> Define synonyms <input checked="" type="checkbox"/> Allow automated expansion
@sys.given-name:given-name
<u>Drishti</u>
Enter value


Figur A.4: first-name entity. Bilden är skapad av författarna.


Figur A.4 visas first-name entity där förnamn som applikationen bör känna igen läggs till. Denna entity innehåller en inbyggd entity (förnamn) och ett hårdkodat förnamn (Drishti) som inte ingår i den inbyggda entityn.

last-name
<input type="checkbox"/> Define synonyms <input checked="" type="checkbox"/> Allow automated expansion
@sys.last-name:last-name
Diem
Enter value
<a href="#">+ Add a row</a>


Figur A.5: last-name entity. Bilden är skapad av författarna.

Figur A.5 visas last-name entity där efternamn som applikationen bör känna igen läggs till. Denna entity innehåller en inbyggd entity (efternamn) och ett hårdkodat efternamn (Diem) som inte ingår i den inbyggda entityn.



 **Fulfillment**

## Webhook


ENABLED 

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#) specific to the API version enabled in this agent.

URL\*

BASIC AUTH

HEADERS

 Add header

DOMAINS

Figur A.6: Fulfillment webhook. Bilden är skapad av författarna.

Figur A.6 visar fliken för att konfigurera webhook anrop. URL:n pekar till en funktion som är en del av Cloud Functions (beskrivet i sektion 2.8) vilket kan ta emot webhook anrop. BASIC AUTH och HEADERS inmatningarna kan används för autentisering av webhook anrop för säker kommunikation mellan Dialogflow och server. Autentisering konfigurerades inte i kalenderapplikationen.

## B Guider

Denna bilaga innehåller en tabell av artiklar och guider om utveckling av röststyrda applikationer som hittades med en googlesökning.

Författare	Titel	Länk
Aditya Dehal	How to build apps for Google Assistant with no programming experience	<a href="https://medium.freecodecamp.org">medium.freecodecamp.org</a> (besöktes senast 2019-01-03)
Jessica Thornsby	Build your own Action for Google Assistant	<a href="https://androidauthority.com">androidauthority.com</a> (besöktes senast 2019-01-03)
Eric Kim	How To Develop A Google Assistant App: Getting Started With Dialogflow	<a href="https://clearbridgemobile.com">clearbridgemobile.com</a> (besöktes senast 2019-01-03)
Abhinav Tyagi	Google Assistant App in 5 min!	<a href="https://medium.com">medium.com</a> (besöktes senast 2019-01-03)
Ido Green	How to Build a Google Assistant App?	<a href="https://greenido.wordpress.com">greenido.wordpress.com</a> (besöktes senast 2019-01-03)

## C Google Home

Figur C.1 visar den röststyrda högtalaren Google Home som användes under projektet.



Figur C.1: Den smarta högtalaren Google Home. Bilden är skapad av författarna.

