Röststyrning för sjukanmälan/skolfunktioner

Innehållsförteckning

[1 Inledning 3](#_Toc490134647)

[2 Teknologier/termer 3](#_Toc490134648)

[2.1 Conversational UI 3](#_Toc490134649)

[2.2 Google now 4](#_Toc490134650)

[2.2.1 Utveckling till Google Now 4](#_Toc490134651)

[2.3 Siri 5](#_Toc490134652)

[2.3.1 Utveckling till Siri/iOS 5](#_Toc490134653)

[2.4 Google Assistant 5](#_Toc490134654)

[2.4.1 Utveckling till Google Assistant 5](#_Toc490134655)

[2.5 API.AI 6](#_Toc490134656)

[2.5.1 Utveckling mot api.ai 6](#_Toc490134657)

[2.6 Speech Synthesis Markup Language (SSML) 10](#_Toc490134658)

[2.7 Webhook 11](#_Toc490134659)

[2.7.1 Deploya lokalt 11](#_Toc490134660)

[2.7.2 Utveckla webhooken 12](#_Toc490134661)

[2.8 Övrigt 12](#_Toc490134662)

[2.8.1 Inloggning 12](#_Toc490134663)

[3 Sammanfattning 13](#_Toc490134664)

Röststyrning för sjukanmälan/skolfunktioner

# Inledning

Röststyrda assistenter till mobiltelefoner utvecklas ständigt och blir allt mer mänskliga och allmänt bättre, vilket i sin tur gör att fler börjar vilja använda sig av dem. Därför ökar fördelarna med att stödja det i sina egna appar. Röststyrning finns även i många bilar, hemassistenter, datorer, robotar och så vidare. Röststyrningen och dessa assistenter är till för att underlätta för användaren vid olika tillfällen. De kan göra allt från en vanlig sökning på nätet till att starta ett samtal åt dig, skapa påminnelser, berätta vädret, med mera. De hjälper även för de med synskador genom att ge ett gränssnitt som kan användas oavsett synförmåga. I framtiden, allt eftersom vanan att använda röst ökar, är det troligt att röststyrning kommer att finnas överallt där röst är smidigare än en skärm eller mus och tangentbord.

För att ta reda på mer om röststyrning och hur det skulle kunna utnyttjas av Tieto Education har vi använt oss av api.ai och Actions on Google för att skapa en proof-of-concept app som ska hantera sjukanmälan av barn, visa nyheter och ha möjlighet att utföra andra skolrelaterade funktioner med röststyrning.

# Teknologier/termer

## Conversational UI

Konversationsgränssnitt är en typ av gränssnitt som relativt nyligt fått popularitet. I och med framsteg inom naturlig språkbehandling (natural language processing, NLP) och artificiell intelligens har dessa gränssnitt faktiskt blivit möjliga att skapa. De försöker efterlikna mänskliga konversationer, något som är intuitivt för alla människor, för att skapa enkla sätt som användare kan uppnå sina mål. Om man ska kunna skapa ett gränssnitt som efterliknar en konversation måste man först förstå hur vi människor håller konversationer.

Först och främst är konversationer turbaserade; man talar en åt gången och väntar på sin tur att prata. När vi talar sänder vi ut mer eller mindre subtila signaler om att vi lämnar över turen och låter sedan den andra parten prata. I sitt gränssnitt är det därför viktigt att inte sända ut dessa när man inte vill lämna över turen. En stark sådan signal är att ställa en fråga. Om appen ställer en fråga bör den därför aldrig fortsätta prata utan att ge användaren en chans att svara.

En konversation följer också en röd tråd. Allt som person A säger måste flyta ihop med det person B säger efter det (ett så kallat *adjacency pair*) för att det ska kännas naturligt. Så även när person B vill byta ämne måste hen väva ihop sitt tal med det A sa just innan. Till exempel skulle ett adjacency pair kunna vara:

*“Hur är det med dig?”*

*“Bara bra! Vad skulle du vilja äta?”*

En av styrkorna med en konversation är den naturliga effektiviteten i hur vi människor brukar språket. Detta är också en av de större utmaningarna att utveckla kring, men när man på rätt sätt “läser mellan raderna” i en konversation kommer programmet framstå som mycket smartare och användarupplevelsen kommer bli mycket bättre. Till exempel om en app frågar hur många personer som ska dela på en taxi kan användaren svara *“Bara jag och min kompis”* istället för *“Två”*. Detta är något som appen måste kunna hantera.

Sist men inte minst är de viktigaste sakerna en utvecklare måste tänka på för att kunna föra en konversation med användaren. När det kommer till mänskliga konversationer så är det en konst vi människor utvecklat över tusentals år och är en av de naturligaste sakerna som finns för oss, men en som kan anta extremt många olika former för att fylla samma syfte. Därför måste appen, för att behålla en positiv användarupplevelse, utvecklas för att hantera alla de sätt användaren kan prata och inte försöka tvinga ett visst talmönster/vissa fraser. Det är enkelt att tänka på ett sätt användaren kan nå sitt mål, implementera det, och sedan hantera alla felsteg med att tvinga in användaren på den tänkta banan. Undvik fraser som: *“Jag förstod inte ‘Visa nyheter’, säg ‘Se nyheter’”*. Det finns nästintill oändligt många fler sätt att säga samma sak på, som alla bör stödjas. För att hitta så många som möjligt av dessa är det viktigt att genomföra många användartester.

Att ta hänsyn till allt detta kan kännas komplicerat, men Google har en mycket bra [guide till att designa en konversation](https://developers.google.com/actions/design/principles) som rekommenderas starkt. Där finns även mer läsning om do’s and don’ts och best practices. Till exempel hur man bör hantera “fel” för att reparera konversationen, när det passar mindre bra att använda en konversation i sin applikation, eller hur man designar en persona för att användaren ska känna att den kan ha en sammanhängande konversation med en trovärdig “person”.

## Google now

Google Now är en intelligent personlig assistent utvecklad av Google som går att styra med röststyrning eller via text. Google Now är en uppgradering av Google Voice Search som fanns redan 2002. Den fungerade genom att användaren ringde in till Google Voice Search-systemet med sin sökning.

Uppgraderingen till den nya Google Now assistenten släpptes 9e juli 2012 och finns tillgänglig i Google-appen för både Android och iOS. Den har bra stöd för många olika språk, däribland svenska. Google Now använder sig av ett röstgränssnitt för att svara på frågor, ge rekommendationer och utföra olika handlingar som hanteras genom att skicka förfrågningar till en webbservice. Rekommendationer görs genom ”Google kort”. De visar till exempel hur trafiken är i närheten, filmer som visas eller olika nyheter, med mera. Google Now aktiverades tidigare genom att säga ”Okay, Google Now” i Google-appen eller från vilken skärm som helst, men nu räcker det att endast säga ”Okay, Google”. Det går även att starta appen genom att hålla in hemknappen på de flesta telefoner. Det senaste året har Google Now ersatts av den smartare Google Assistant på nya enheter.

### Utveckling till Google Now

Google Now kan bara hantera ett röstkommando åt gången och saknar stöd för konversationer. Man måste också definiera exakt ordagrant vad användaren ska säga och den förstår inte naturliga variationer i talet. Eftersom den är så begränsad och håller på att ersättas av Google Assistant finns det inte riktigt någon anledning att utveckla något till Google Now.

## Siri

Siri är Apples intelligenta personliga assistent som bara finns till iOS. Siri lanserades första gången i oktober 2011. På samma sätt som Google Assistant använder Siri sig av ett konversationsgränssnitt för att svara på frågor, ge rekommendationer, och utföra handlingar genom att skicka förfrågningar till olika internettjänster. Siri anpassar sig till användarens språk, sökningar, och inställningar. Även svaren är individanpassade. Siri kan söka på internet, skapa events och påminnelser, ringa/smsa och kommunicera med andra iOS-integrerade appar, med mera. Siri har även stöd för svenska.

### Utveckling till Siri/iOS

Att utveckla till Siri är mer begränsat än vad det är till Google Assistant. Det finns ett begränsat antal “[intent domains](https://developer.apple.com/documentation/sirikit#2863953)” som har stöd i Siri. Dessa är bl.a. *Messaging*, *Payments* och *Ride booking*. (Alltså inget som skulle kunna användas för sjukanmälan).

Apple Developer har, för att utveckla till Siri, något som kallas SiriKit. SiriKit omfattar ramarna för intents och dess UI, som används för att implementera en “app extension” som integrerar tjänster med Siri. Appförläningen pratar sällan direkt med användaren, utan det är Siri som tar hand om all kommunikation med användaren och pratar bara med förlängningen när det är nödvändigt.

Det går alltså inte att utveckla egna kommandon till Siri, men att bygga en app och ansluta till api.ai direkt ska gå. Till Cordova finns det ett lättanvänt plugin för api.ai som ska vara kompatibelt med iOS och Android, men när vi försökte testa appen vi skapat på en iPhone kraschade den. Mer specifikt fungerade det att ansluta till agenten och få det förväntade svaret om man bara skickade en textsträng, men om man försökte låta användaren tala med telefonen gavs ett okänt fel. Exakt vad det beror på lyckades vi inte lista ut, men det finns en möjlighet att det är på grund av att mikrofon-koden i pluginet inte är kompatibelt med senare versioner av iOS. (iOS 9 var det som fanns när pluginet släpptes).

## Google Assistant

[Google Assistant](https://assistant.google.com/) lanserades i maj 2016. Google Assistant är baserat på Google Now, men kan till skillnad från Google Now delta i tvåvägskommunikation. Det gör att assistenten kan föra en bättre konversation och skapa ett mer personligt bruk genom att användaren kan göra olika inställningar som, till exempel, vad den ska kalla användaren. Tyvärr har Google Assistant inget stöd för svenska just nu, men det kommer förhoppningsvis att släppas snart. De språk som stöds är engelska, franska, tyska, hindi, portugisiska, japanska och spanska. Senare i år kommer även italienska och koreanska. För att kunna använda Google Assistant krävs det att språkinställningarna i mobilen är på något av de språken som stöds. Vi fick därför utveckla vår app på engelska. Google Assistant finns inte bara till de flesta Androidtelefoner från de senaste åren utan även till Google Home, en så kallad smart högtalare.

### Utveckling till Google Assistant

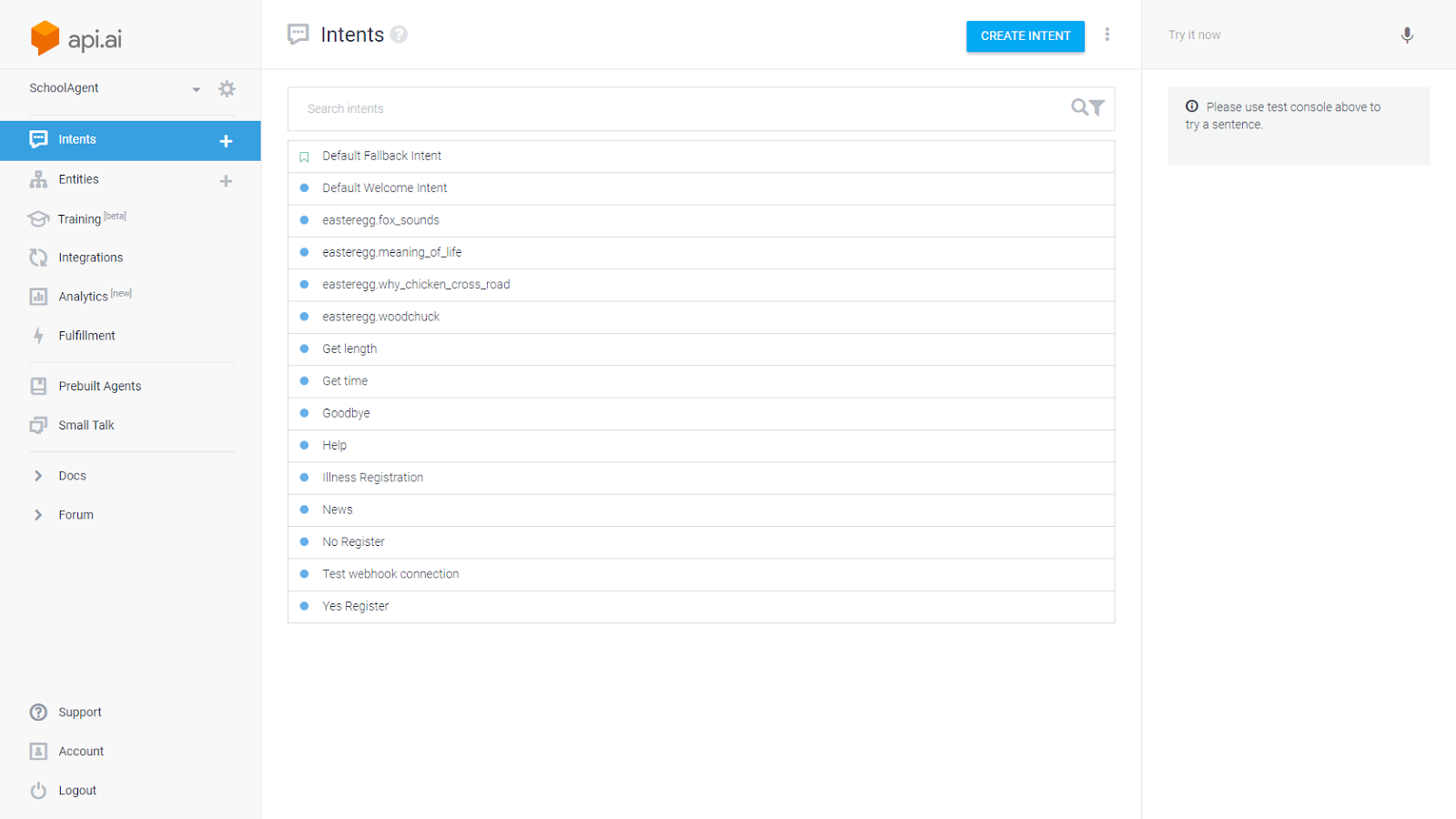
[Actions on Google](https://developers.google.com/actions/) gör det möjligt för vem som helst att utveckla appar till de enheter som stödjer Google Assistant. Med Actions on Google kan man bygga på funktionaliteten i Google Assistant genom så kallade actions. Dessa actions kan skapas med Googles programvara api.ai, som beskrivs senare i rapporten.

För att utveckla för Actions on Google måste man först skapa ett nytt projekt på [Actions on Google](https://console.actions.google.com/) och en [api.ai](https://console.api.ai/api-client/) agent. Actions-projektet låter en definiera metadata, som till exempel vad användaren säger för att starta appen, hur appens namn ska uttalas, m.m. Api.ai agenten definierar ”intents” som hanterar och mappar vad användaren säger mot motsvarande logik som utför förfrågningen och svarar användaren. För att testa hur appen fungerar på Google Assistantplattformen finns det i api.ai möjlighet att köra den i en emulator eller så går det att testa på en riktig enhet så länge den är inloggad på samma Googlekonto.

## API.AI

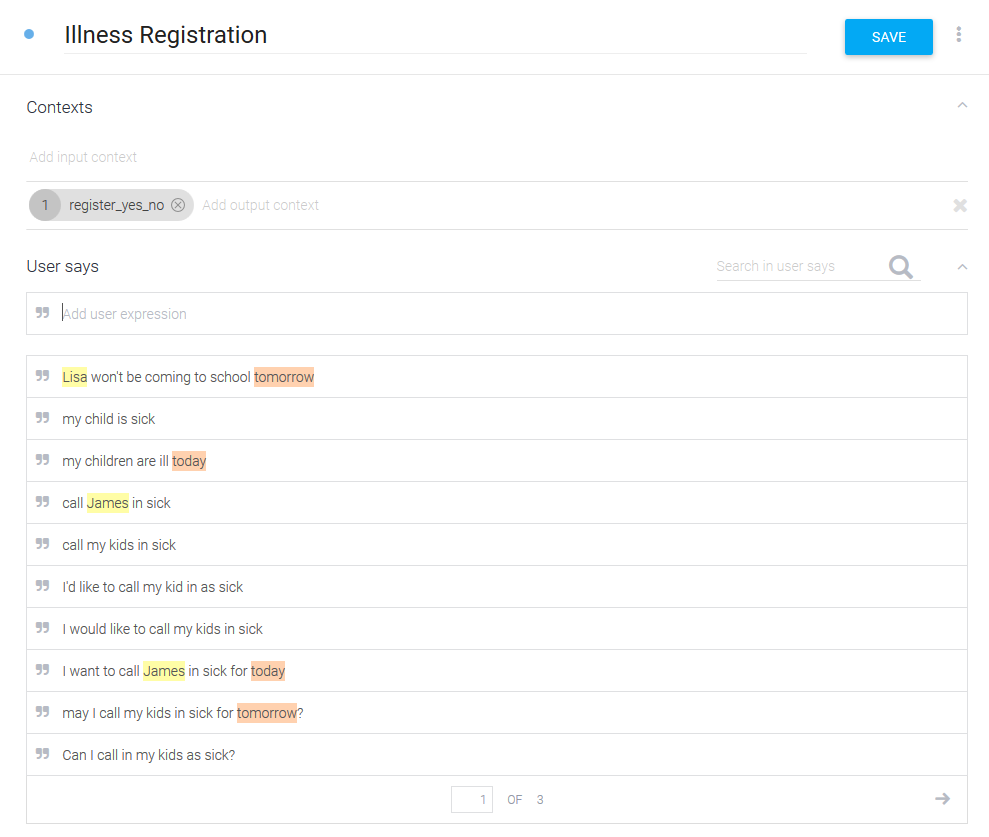
[Api.ai](https://api.ai/) är utvecklare av människa- till dator-interaktioner baserat på ”natural language conversations”. Företaget hette tidigare Speaktoit innan det köptes upp av Google i september 2016. De skapade ”Assistant”, som senare blev Google Assistant. Api.ai har även skapat en NLP-motor som, förutom att behandla naturligt tal, hanterar och kommer ihåg dialoghistorik, plats och användarinställningar. Dåvarande Speaktoit släppte api.ai i september 2014, så att tredjepartsutvecklare kunde använda sig av NLP-motorn som används i Assistant. Mjukvaruutvecklingsverktyget innehåller röstigenkänning, naturlig språkbehandling, och text-to-speech.

### Utveckling mot api.ai

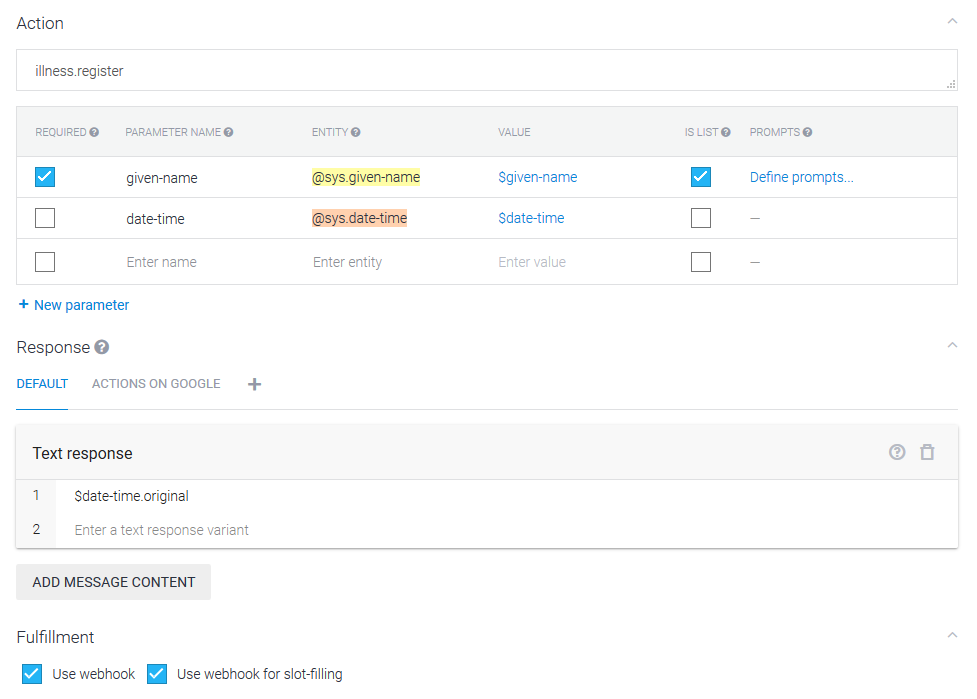
I api.ai-verktyget finns det många olika begrepp som används. *Intents*, som nämns tidigare, matchar användarens förfrågan med den tjänst som utför det användaren avsett. *Fulfillment* består av *webhooken*, externa API:er och databaser, det är de som sköter logiken. En *action* är det applikationen ska göra när ett intent har blivit triggat av en användares input. I intentet listar man fraser som en användare kan tänkas säga. I den agent vi skapade definierade vi flera olika intents för att hantera själva sjukregistreringsdelen, en hjälpfunktion, nyheter, m.m.

I bilden ovan syns en lista över alla intents som finns i vår agent, *SchoolAgent*. Många av dessa är bara till för test, men till exempel Illness Registration-intentet används ifall användaren säger något för att sjukanmäla sina barn. Fallback intent är ett viktigt specialfall eftersom där definieras vad appen ska säga när det inte gick att förstå användarens input.

För att använda en webhook till logiken bakom sina intents måste det aktiveras genom att bocka i *Use Webhook* under varje motsvarande intent. För mer information om att skapa en webhook se det avsnittet i denna rapport.



I denna bild syns även var man definierar vad användarna ska kunna säga för att trigga ett intent. Dessa bör täcka in så många olika variationer som möjligt av vad en användare skulle kunna tänkas säga, men måste inte alltid matcha ordagrant. Agenten förstår av sig själv, genom machine learning, små naturliga variationer i talet. Mer om det i ett kommande stycke.



I fraserna kan man använda sig av *entities*. Entities är till för att få ut parametervärden från användarens input. En entity är egentligen inget mer än en lista över saker som hör ihop. En entity som heter *frukt* skulle till exempel kunna innehålla listan *äpple, banan, päron, apelsin*. Om den entityn används i en fras skulle, när användaren säger någon av frukterna, värdet sparas för enkel åtkomst i koden som hanterar logiken. Api.ai har några färdiga inbyggda entities för att hantera de mest populära koncepten, som till exempel namn, datum, färger och så vidare. Den färdiga namnentityn (*@sys.given-name*) innehåller en lista på de 2500 vanligaste engelska namnen. Det går även att skapa egna entities för att anpassa till sin apps behov.

Api.ai har också några förbyggda agenter som man kan ladda ner och importera för att använda. Dessa innehåller då förslag på färdigskapade intents och entities. Det behövs dock en webhook för att få agenten att fungera och utföra något, annars är det bara att ändra och använda som man vill.

Det går även att aktivera *small talk.* Det är fördefinierade små fraser som man kan använda i sin app när användaren säger något “small talk”. Till exempel “What's your birth date?” besvaras då med “You know, I'm not really sure.” istället för ett fallback. Det går också att redigera alla fraserna.

Agenterna kan designas för att hantera konversationsflödet på ett speciellt sätt, detta kan göras med hjälp av *contexts*. Kontexter kan skapas för att applikationen ska komma ihåg att efterföljande förfrågningar hör till den föregående förfrågan. Kontexter kan även användas för att begränsa intents att inte matcha förrän vissa kontexter är satta. Till exempel skulle frasen “Help” kunna matcha mot ett generellt hjälpintent när ingen kontext är satt (inget speciellt pågår). Men när en kontext är satt, t.ex. *register\_yes\_or\_no* (något pågår), skulle den matcha mot ett intent där användaren får hjälp med att bekräfta sin registrering.

Api.ai-agenten lär sig även genom machine learning allt eftersom den används och uppdateras. Till exempel om en fras är ”I want to call James in sick today” så kan den matcha till intentet även om användaren skulle lägga till något extra utfyllnadsord, ändra namn eller ändra dag. Allting som sägs till agenten loggas i *Training*. Om en användare skulle säga något som inte matchas med en fras i ett intent, men borde göra det, går det att därifrån lägga till just den frasen användaren använde till rätt intent. Just nu är dock Training i beta och kan därför vara lite buggig i hur den lägger till rätt entities i fraserna. Tills vidare kan man därför manuellt lägga in frasen i rätt intent via den vanliga sidan.

En annan funktion som fortfarande är i beta, men som kan vara mycket användbar är *Analytics*. Där går det att se statistik över hur olika intents används, hur mycket de används, hur lång responstid de har, m.m.

Tidigare nämns det att man kan testa sin applikation i en Actions on Googles Assistant emulator eller på en enhet med Assistant. För att det ska fungera måste man under *Integrations* aktivera Actions on Google. Där går det även att aktivera integrering med många fler tjänster som till exempel Skype, Slack, Facebook Messenger, Alexa, eller något annat för att testa eller om det är mot någon av de tjänsterna agenten ska användas.

## Speech Synthesis Markup Language (SSML)

[SSML](https://www.w3.org/TR/speech-synthesis/) är ett XML-baserat markupspråk som används för att assistera talsyntesering. När man skickar text för att text-to-speech (tts) behandlas kan SSML markera hur texten ska betonas eller på vissa andra sätt uttalas. Till exempel kan man föra in pauser med viss längd i talet, markera att “10-9” ska uttalas som “tionde september” istället för det vanliga uttalet som skulle varit något i stil med “tio nio”, infoga mindre förinspelade ljudklipp i talet, m.m. För att förbättra användarupplevelsen genom att få sin app/persona att låta mer mänsklig är detta ett användbart verktyg som kan hjälpa till med att undvika vissa onaturliga/robotliknande uttal.

Google Assistant använder sig av en [variant av SSML](https://developers.google.com/actions/reference/ssml) som bland annat inte kräver någon XML DOCTYPE i början, utan bara en <speak> tag. I deras dokumentation kan man läsa fler exempel på vilka element som finns också. När det behövs är det mycket praktiskt och simpelt och när det inte behövs kan det helt ignoreras.

Vår fristående testapp som tar emot samma text från webhooken som Assistant och läser upp den via ett Cordova-plugin som anropar Androids inbyggda text-to-speech kunde inte hantera SSML. Hur mycket vi än provade att ändra olika taggar försökte den ändå bara läsa upp dem också. Det verkar som att Androids tts ska fungera med SSML så om vår app inte fungerade på grund av att pluginet förstör eller att vi inte hittade det korrekta sättet att skriva på vet vi inte.

Till Siri och/eller den inbyggda tts i iOS kunde vi inte heller hitta någon officiell eller inofficiell dokumentation som kunde ge ett klart svar om SSML stöds och i så fall om det är någon variant eller inte. Eftersom det i nuläget är för begränsat att utveckla till Siri är det kanske inte något problem till Siri, men eftersom det fortfarande ska gå att utveckla en fristående app kopplad mot api.ai skulle det varit bra att veta om den inbyggda tts stödjer det. Tyvärr kunde vi, som vi nämnt tidigare, inte testa vår app på iOS på grund av något okänt fel, möjligtvis med api.ai pluginet till Cordova. Det känns dock rimligtvis som att det borde finnas stöd för det.

## Webhook

För alla applikationer där det krävs mer logik (eller api-anrop eller liknande) än att bara svara en av ett par olika fraser då användaren säger något används något som kallas fulfillment/webhook. Google rekommenderar att man använder [deras molntjänst](https://api.ai/docs/getting-started/basic-fulfillment-conversation) för att host:a sin webhook, men det går lika bra att host:a den själv.

### Deploya lokalt

Det finns säkert fler sätt att göra det på, men det enkla sättet vi gjorde det på var genom att använda [node.js](https://nodejs.org/en/), [Google Cloud Functions Local Emulator](https://cloud.google.com/functions/docs/emulator) och [ngrok](https://ngrok.com/). Servern vi satte upp detta på var en Windows-server så vissa steg kommer att vara specifika för Windows, men det är inte svårt att göra på linux heller.

Börja med att ladda ner och installera node.js. Se till att mappen node.js installerats i har lagts till i PATH, samt att mappen *C:/Users/username/AppData/Roaming/npm* också finns i PATH. Installera sedan GCFLE genom att köra kommandot npm install -g @google-cloud/functions-emulator.Starta sedan genom att köra functions start i webhookmappen. Här bör man ange det projektnamn som användes när api.ai-agenten integrerades med Actions on Google för att vara konsekvent. (Eller ange där det som angavs här om det görs i motsatt ordning). Deploya webhooken med functions deploy agentName --trigger-http(agentNamemåste inte matcha api.ai-agentens namn utan det som används i webhooken, men det brukar vara agentens namn). Om deployen gick bra skriver programmet ut vilken URL som används. Den är normalt *http://localhost:8010/projektnamn/region/agentName.* *Region* är en av Googles [compute regions](https://cloud.google.com/compute/docs/regions-zones/regions-zones) och är, om den inte ändrats, *us-central1*. Den kan ändras genom kommandot functions config set region new-region-name.

Installera ngrok genom att ladda ner den körbara filen (programmet består av bara en .exe-fil) eller npm install -g ngrok. Se till att mappen där .exe-filen ligger lagts till i PATH. Därefter är det bara att starta ngrok med kommandot ngrok http port. (Porten som functions emulator använder är 8010). Ngrok skriver då ut den offentliga url man nu kan använda för att komma åt sin webhook. Den kommer vara på formatet *https://randomstring.ngrok.io* och ska användas istället för *localhost* i cloud functions URL för att agenten ska komma åt webhooken. Klistra sedan in den färdiga URL:en i agenten, under fulfillment så är den klar att användas. För mer teknisk information om ngrok och gratisversionens begränsningar läs deras [dokumentation](https://ngrok.com/docs) eller [produktinformation](https://ngrok.com/product).

### Utveckla webhooken

I webhookens *index.js-*fil skriver man den kod som hanterar vad som ska hända beroende på vad användaren säger. (Detta kan som vanligt delas upp i fler filer om man har mycket kod). Först rekommenderas att installera Actions on Google client library med kommandot npm install actions-on-google för att kunna använda deras [ApiAiApp klass](https://developers.google.com/actions/reference/nodejs/ApiAiApp) som förenklar jobbet att ta emot datan som skickats och att skicka tillbaka ett respons. Sedan skriver man koden. För enkla exempel se ett [exempel från Google](https://developers.google.com/actions/apiai/fulfillment#building_fulfillment_responses) eller vårt exempel:  
'use strict';

//importera

const ApiAiApp = require('actions-on-google').ApiAiApp;

//skapa konstanter för intents, context, prompts

const TEST\_INTENT = "test.helloworld";

//måste inte ha agentens namn, men det är bra

//ska matcha det som skrivs när man deployar i cloud functions

exports.agentName = function agentName (req, res) {

  console.log('Headers: ' + JSON.stringify(req.headers));

  console.log('Body: ' + JSON.stringify(req.body));

  //skapa en instans

  const app = new ApiAiApp({request: req, response: res});

  //skapa en map för vilket intent som ska köra vilken funktion

  let actionMap = new Map();

  actionMap.set(TEST\_INTENT, test);

  function test(){

      app.tell("Hello world");

  }

}

## Övrigt

### Inloggning

En praktisk begränsning att ta hänsyn till, om detta skulle implementeras på riktigt, som vi kom att tänka på under arbetets gång är problemet med inloggning. Eftersom användare loggar in med BankID (alla inloggningsmetoder har mer eller mindre samma problem) i Education appen och de måste vara inloggade för att ha rätt att sjukanmäla/göra annat blir det svårare att helt komma ifrån att användaren måste interagera med telefonens skärm. Beroende på säkerhetskrav går det kanske att lösa med att man sparar något token eller liknande på enheten, men om det är omöjligt/för osäkert kommer det att sänka användarupplevelsen då man måste använda skärmen för att logga in. (För man kan absolut inte tillåta användaren att säga lösenordet med rösten för att logga in).

# Sammanfattning

Vi tror att röststyrning kommer vara en stor grej i framtiden. Just nu är dock användningen av röststyrning och assistenter relativt låg i Sverige, men när Google Assistant får stöd för svenska lär användningen av röststyrning öka. Även om många är duktiga på engelska känns det mer naturligt för en användare att få prata på sitt modersmål. Vi själva har inte använt oss av, eller stött på röststyrning speciellt mycket tidigare, men nu efter arbetet har vi blivit mer intresserade och tycker att det är en bra funktion. Det har även varit lärorikt och roligt att testa utveckla med api.ai, webhook och Actions on Google.

Appen vi skapade för sjukanmälningen och skolnyheter går att bygga ut med fler funktioner, som till exempel något för läxor, skolaktiviteter eller dylikt. Det är bara att lägga till fler intents och hantera logiken i webhooken.