LAB University of Applied Sciences

KOGNITIIVINEN CHATTIBOTTI

Kognitiivisen chatbot-sovelluksen toteutus Bot Framework-ohjelmistokehyksellä

LAB-AMMATTIKORKEAKOULU
Insinööri (AMK), Tieto- ja Viestintätekniikka
Ohjelmistotekniikka
2020
Sampo Sahla

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Sahla, Sampo	Opinnäytetyö, AMK	2020
	Sivumäärä	
	31	

Työn nimi

Kognitiivinen Chattibotti
Kognitiivisen chatbot-sovelluksen toteutus Bot Framework-ohjelmistokehyksellä

Tutkinto

Tieto- ja viestintätekniikka, Ohjelmistotekniikka (AMK)

Tiivistelmä

Tämän opinnäytetyön tavoitteena oli toteuttaa kognitiivinen chattibotti. Opinnäytetyön toimeksiantajana toimi IT-konsultointiyritys Enfo Oyj.

Työssä tarkastellaan Microsoftin Bot Framework-ohjelmistokehyksellä botin toteutusta ja älykkäiden eli kognitiivisten palveluiden liittämistä bottiin ja kokonaisuuden tuomista käyttäjälle.

Termiä ”kognitiiviset palvelut” käytetään kuvaamaan älykkäitä ohjelmistoratkaisuja, jotka käsittelevät dataa ilman täydellisiä ohjeita. Esimerkiksi kielen ymmärryksessä tekstistä voidaan löytää tarkoitus ilman, että kyseistä tekstiä olisi ennen syötetty palvelulle.

Opinnäytetyössä käydään läpi toteutuksen teknologioita ja järjestelmiä. Käytännön osuudessa käydään läpi ympäristö, jossa sovellusta ylläpidetään sekä sovelluksen arkkitehtuuria. Lopuksi kuvataan toteutusta näyttäen esimerkkejä toteutuksen koodista. Työ on toteutettu täysin JavaScript-ohjelmointikieltä käyttäen.

Asiasanat

JavaScript, chattibotti, kognitiiviset palvelut

Abstract

Author(s)	Type of publication	Published
Sahla, Sampo	Bachelor's thesis	2020
	Number of pages	
	31	

Title of publication

Cognitive Chatbot
Making a cognitive chatbot application with Bot Framework a software development framework

Name of Degree

Bachelor of Information Technology

Abstract

The goal of this thesis was to make a cognitive chatbot. The thesis was made for an IT consulting firm called Enfo Oyj.

The thesis explores making a chatbot with cognitive services, or intelligent services, using Bot Framework a software framework made by Microsoft.

The term "cognitive services" is used to describe intelligent software solutions, that can handle data without having seen the same data before or knowing specially what this data means. For example, in language understanding the software can deduce what a piece of text is saying, without ever actually having seen that text or its structure before.

Technologies and systems used in the project are first described as the technical part of this thesis. The environment where software is maintained, and the software architecture of the project is explored, in the practical section of this thesis. The implementation of the project is described using examples from the project code. Software has been made entirely using JavaScript programming language.

Keywords

JavaScript, chatbot, cognitive services

SISÄLLYS

1 JOHDANTO

Työn tavoitteena on toteuttaa bottipalvelun luonti sekä kognitiivisten palveluiden liittäminen chattibottiin. Työssä keskitytään tarkemmin bottipalveluiden valmiisiin runkoihin ja niiden implementointiin www-ratkaisuun sekä kognitiivisten palveluiden käyttämiseen valmiiden runkojen kanssa.

Monet asiakaspalvelutyöt vaativat paljon henkilöstöä. Kognitiivisilla bottipalveluilla voidaan esimerkiksi vähentää turhaa ihmisresurssien käyttöä ja toteuttaa automatisoituja tehtäviä. Ihmisresurssit voidaan sijoittaa tehtäviin, jotka palvelevat yritystä paremmin. Myös turvallisuusjärjestelmiä voitaisiin potentiaalisesti automatisoida. Kameravalvonnan tarkasteluun ja vastaaviin prosesseihin ei tarvittaisi ihmisiä, sillä bottipalvelu voisi ilmoittaa poikkeavista tapahtumista esimerkiksi viestillä.

Työ toteutetaan Enfo Oyj nimiselle yritykselle. Enfo on IT-alan yritys, joka tuottaa erilaisia ohjelmistoratkaisuja asiakkailleen. Enfo toimii Suomessa, Ruotsissa ja Norjassa, joissa yrityksellä on 11 toimistoa eri kaupungeissa. Enfo perustettiin 1964 ja sen tämänhetkinen toimitusjohtaja on Seppo Kuula (tammikuu 2018 -). (Enfo Oyj 2020a.) Enfolla on yhteistyökumppaneita yritysten kuten Microsoft, SAP, Informatica ja IBM kanssa. Enfolla on myös osaajia yhteistyökumppaneiden tarjoamien palveluiden kanssa. (Enfo Oyj 2020b.) Enfon liiketoimialana on ilmoitettu tietojekäsittely, palvelintilan vuokraus ja niihin liittyvät palvelut. Vuonna 2019 Enfo työllisti 360 henkilöä ja liikevaihto oli 53,5 miljoonaa euroa. (Asiakastieto 2020.)

Työ tilattiin POC (Proof of Concept) -projektina. Työn tarkoituksena on esittää chattibottien toiminnallisuutta ja kannattavuutta nykyisille ja uusille asiakkaille. Tutkimuksessa käydään läpi bottipalvelun luontia tarkemmin, käyttäen esimerkkinä Azure Bot Service työkaluja ja pääkomponenttina Microsoftin Bot Framework-ohjelmistokehystä runkona itse chattibotille. Kognitiivisten palveluiden osalta käytetään Language Understanding (LUIS) palvelua, OpenCV avointa konenäkökirjastoa ja Azure Face palvelua, jotka integroidaan bottipalveluun.

Tavoitteena on luoda chattibotti, jolle voidaan keskustella käyttäen valintoja tai itse kirjoittamalla. Botin tulee ymmärtää käyttäjän tarkoitus lauseista ja ohjata keskustelua eteenpäin oikeaan paikkaan. Keskustelun on tarkoitus vaikuttaa luonnolliselta käyttäjälle. Palvelun tulee tunnistaa käyttäjän läsnäolo ja aloittaa keskustelu ilman käyttäjän toimia.

2 PALVELUN TARVE JA KÄYTTÖ

Palvelun tarkoituksena on automatisoida vastaanottopalveluita. Palvelu voisi esimerkiksi korvata rakennuksen aulan info- tai vastaanottopalvelut. Tavallinen käyttö tapahtuisi päätelaitteella, jossa avoimena on palvelun käyttöliittymä. Palvelu tunnistaisi, kun käyttäjä saapuu päätelaitteelle ja osaisi keskustelun perusteella ohjata käyttäjän oikeaan paikkaan. Käyttöliittymän sekä chattibotin haluttiin toimivan pilvessä. Päätelaitteena haluttiin toimivan mikä tahansa laite, jolla voidaan kirjoittaa ja käyttää kameraa. Päätelaitteena voisi toimia esimerkiksi tabletti tai tietokone.

Tyypillisenä käyttötapauksena voisi olla esimerkiksi käyttäjä, joka on tulossa tapaamiseen, mutta ei tiedä missä huoneessa tapaaminen järjestetään. Käyttäjä voisi kysyä palvelulta tapaamisen paikkaa antamalla palvelulle tietoja itsestään. Käyttäjän antamat tiedot voitaisiin liittää tietokantaan tallennettuun tietoon ja ohjata käyttäjä oikeaan paikkaan.

Tuote haluttiin toteuttaa Proof of Concept -projektina (POC) ja näyttömallina olemassa oleville asiakkaille sekä uusille potentiaalisille asiakkaille. Tuotteella haluttiin myös osoittaa yrityksen taitoja uudenlaisissa ratkaisuissa. Tuotteen haluttiin käyttävän Microsoft LUIS kognitiivista kielen ymmärrystä keskustelujen navigoinnin avustukseen.

Käyttöympäristönä toimivat demotilat kuten messut, mutta palvelu tehtiin yritysten aulat mielessä. Käyttäjä voisi tulla chattibotin päätepisteelle ja chattibotti aloittaisi keskustelun tunnistaen käyttäjän päätepisteen kameraa käyttäen. Jos käyttäjä on jo aikaisemmin käyttänyt chattibottia ja antanut luvan tallentaa käyttäjän tunnistetiedot, voisi chattibotti ohittaa osan tarvittavasta keskustelusta aiemman tiedon perusteella. Chattibotti antaa myös muuta tietoa kuten sään ja näyttää tilanteen mukaan erilaisia emojikasvoja luodakseen luonnollisempaa keskustelua käyttäjän kanssa. Emojikasvojen toisena ideana oli tehdä chattibotista hieman uniikimpi sekä hauskempi käyttää.

Tarkoitetut käyttäjät ovat yritysasiakkaita, jotka ovat tulossa tapaamaan yrityksen henkilökuntaa. Nämä käyttäjät chattibotti ohjaa kysymysten perusteella oikealle henkilölle. Tarvittaessa chattibotti osaisi muodostaa Skype-yhteyden ihmiseen.

3 SOVELLUKSESSA KÄYTETYTTYJÄ TEKNOLOGIOITA

3.1 Chattibotti

Chattibotilla tai botilla tarkoitetaan keskustelua käyvää sovellusta. Chattibottia tehdessä hyödynnetään valmiita ohjelmistokehyksiä. Ohjelmistokehysten käyttäminen vapauttaa kehittäjän tekemään vain niitä osia, jotka ovat sovelluskohtaisia ja se poistaa lähes aina tarpeen ohjelmoida tiedonvälitykseen tarvittavat osat, jotka voivat viedä paljon resursseja projektilta.

Chattibotilla tarkoitetaan esimerkiksi nettisivulle implementoitua käyttöliittymää, joka voi simuloida keskustelua käyttäjän kanssa. Usein chattibotin simuloiman keskustelun on tarkoitus vaikuttaa edes lähes luonnolliselta. Chattibotti ymmärtää jollakin tasolla käyttäjän sanomasta tarkoituksen. (Expert System 2019.)

Chattibotteja voidaan käyttää esimerkiksi poistamaan turhaa manuaalista työtä. Botteja voidaan tehdä monenlaisia, kuten asiakaspalvelubotti verkkosivulla, tilausjärjestelmäbotti tai vastaanottopalvelubotti.

Chattibotti koostuu useasta komponentista. Komponentteja voi olla niin paljon kuin käyttötarkoitus vaatii. Yleensä rakenne pysyy hyvin samantyyppisenä riippumatta käyttötarkoituksesta tai valmiin rungon käytöstä. Rakenne koostuu yleensä:

- Chattibotin rungosta, joka sisältää toiminnallisuuksia kuten viestien kulkemisen logiikka.
- Käytettävistä palveluista (esim. kognitiiviset palvelut), jotka voi jakautua useampaan komponenttiin.
- Käyttöliittymästä, josta otetaan yhteys chattibottiin.

Koska monet runkopalveluita tuottavat toimijat haluavat mahdollistaa kehitystyön mahdollisimman monelle, on eri valmistajien runkopalvelut mahdollistettu monelle eri kielelle. Microsoft tuottaa Azure Bot Service -palveluaan C#, JavaScript, Java sekä Python -kielille (Microsoft Azure 2020a). IBM tuottaa palveluaan omalla käyttöliittymällä, josta palvelu voidaan integroida esimerkiksi Node.JS applikaatioon tai suoraan www-sivulle.

3.2 Teknologiat ja käsitteet

Projektin teknologiat voidaan jakaa kognitiivisiin palveluihin ja muihin ohjelmistoteknologioihin. Ohjelmistomielessä kategorioilla ei ole eroa, mutta kognitiivisten palveluiden tarkoitus on ymmärtää dataa ilman, että niille on annettava tarkkoja ohjeita.

Kognitiiviset palvelut ovat esimerkiksi SDK- ja API-työkaluja, jotka auttavat kehittäjiä älykkäiden applikaatioiden tuottamisessa ilman tekoäly tai datatieteiden osaamista. Kognitiivisilla palveluilla voidaan lisätä esimerkiksi tunteiden, puheen ja kielen ymmärrystä applikaatioihin. (Microsoft Azure 2019b.)

3.2.1 Back-end

Back-end on ohjelmistokehityksessä käsite, joka tulee ohjelmiston erottamisesta kahteen osaan, toista osaa kutsutaan termillä front-end. Kun puhutaan back-end kerroksesta yleensä tarkoitetaan palvelinta ja sillä toimivaa koodia. Tämä sisältää siis datan säilytyksen sekä sen käsittelyn. MySQL ja muut SQL-teknologiat ovat järjestelmiä, joilla voidaan hallita tietokantoja. Tietokantajärjestelmät toimivat back-end kerroksessa. Muita back-end teknologioita ovat esimerkiksi palvelinkielet Ruby, Python ja Node.js.

Tavallisesti back-end kerroksen sisältönä pidettyjä ominaisuuksia on tehty front-end ratkaisussa ja siksi tiettyjä ominaisuuksia voi olla hieman vaikea jakaa näihin kategorioihin tai sanoa, että jokin tietty ominaisuus on vain back-end tai front-end ominaisuus. Datan käsittelyssä tämä raja on joskus häilyvä.

Ennen lähes kaikki logiikka sovelluksessa suoritettiin palvelimella. Tähän sisältyi dynaamisten www-sivujen tuominen käyttäjäpäätteeseen, tietokannan kanssa kommunikointi, identiteetin autentikointi ja ilmoitusten tuottaminen. Tässä toimintamallissa jokaisen prosessin kutsu on lähetettävä käyttäjäpäästä palvelimelle. Tämä aiheuttaa paljon viivettä sekä rasitusta palvelimelle. Nykypäivän päätelaitteissa on enemmän suorituskykyä, joka mahdollistaa koodin suorittamisen päätelaitteilla. Tällä voidaan myös vähentää käyttäjän kokemaa viivettä. Käyttäjäpäässä suoritettava koodi mahdollistaa esimerkiksi dynaamisten sivujen reaaliaikaisen lataamisen suorittamalla koodia selaimessa, joka tekee muutoksia sisältöön, jonka käyttäjä näkee. (Cloudflare 2020.)

3.2.2 Front-end

Front-end on käsite käyttäjäpäässä suoritettavasta koodista tai kaikesta koodista, joka vaikuttaa käyttöliittymään. Modernit tietokoneet ovat suorituskykyisempiä, joka mahdollistaa päätelaitteessa entistä raskaamman koodin suorittaminen. Palvelimien ei tarvitse suorittaa

kaikkea logiikkaa, joten ne voivat vaatia vähemmän resursseja ja monessa tapauksessa ovat myös halvempia ylläpitää.

Isomorfinen renderöinti on uudempi tapa tuottaa websovelluksia. Ideana tekniikassa on renderöidä websovellus JavaScript ohjelmistokehyksellä kuten Angular, React tai Vue palvelinpäässä ensimmäisellä latauskerralla ja käyttäjäpäässä tämän jälkeen. (Pluralsight 2020.)

3.2.3 JavaScript

JavaScript on ajonaikana käännettävä korkean tason ohjelmointikieli, jota käytetään www-sivuilla. Www-sivuilla ajettava JavaScript-koodi tuo sivulle toiminnallisuutta. JavaScript noudattaa ECMAScript standardia. Monet palvelut selainympäristöjen ulkopuolella, kuten Node.js (ks. 3.2.5) ja Deno, käyttävät JavaScript-kieltä. JavaScript on yksisäikeinen kieli, joka voi toimia sekä proseduraalisena, että olio-ohjelmointikielenä. (Mozilla 2020.)

ECMAScript 2017 mukainen JavaScript mahdollistaa esimerkiksi asynkronisen koodin kirjoittamisen avainsanoja käyttäen. Vuosittain julkaistava uusi standardi tuo mukanaan kieleen uusia ominaisuuksia. (ECMA International 2020.)

3.2.4 JSON

JSON (JavaScript Object Notation) on tiedonsiirtomuoto, joka on helppolukuista ja helposti kirjoitettavaa. Se on riippumaton kielistä, mutta käyttää konventioita C-perheen kielistä. JSON rakenne koostuu järjestämättömistä avain-arvo-pareista, joita voidaan toteuttaa mm. objektina, listana tai hash-taulukkona. (JSON 2020.) Avaimet annetaan merkkijonoina, mutta arvot voivat olla JSON-objekteja, listoja, null arvoja, numeroita, merkkijonoja tai boolean-arvoja. Objekti voi olla yksinkertainen tai sisältää monimutkikkaita sisäkkäisiä rakenteita. JSON-objektin rakenteen suunnittelua rajoittaa ainoastaan arvojen tyypit. JSON ei tue kommentteja. JSON-objektia ympäröi aaltosulkeet ja jos objekteja halutaan useampi samalle tasolle, käytetään tähän lista rakennetta ja sisällytetään objektit vielä hakasulkeisiin (KUVA 1).

Monet palvelut kuten npm ja yarn käyttävät JSON-notaatiota määrittelyissä (package.json). Se on helppo implementoida ja useat kielet tukevat suoraan JSON notaatiota. Muita määrittelyihin käytettyjä notaatioita on esimerkiksi YAML ja XML.

KUVA 1. JSON-rakenne

3.2.5 Node.js

Node.js JavaScript-moottori mahdollistaa JavaScript koodin suorittamisen alustariippumattomasti. Tämä tarkoittaa, että JavaScript ei ole sidottuna selaimeen niin kuin tyypillisesti muuten olisi. Node.js tuo paljon ominaisuuksia jo sellaisenaan, mutta monet kirjastot, joita moottoria varten on tehty, tuovat lisää ominaisuuksia ja hyötyjä sovelluksille. Näitä kirjastoja hallitaan NPM (Node Package Manager) ohjelmistorekisterillä. NPM on maailman laajin ohjelmistorekisteri (NPM 2020).

Node.js soveltuu skaalautuville verkkosovelluksille. Node.js tukee yhteyksien käsittelyä samanaikaisesti. Node.js käsittelee tulevia kutsuja silmukassa, jota kutsutaan tapahtumasilmukaksi.

3.2.6 Blob Storage

Blob Storage on Azuren objektivarasto-ratkaisu pilveä varten, joka on optimoitu säilömään massiivisia määriä jäsentämätöntä dataa (Microsoft Azure 2020b). Blob Storage ei rajoita millaista dataa siellä säilytetään, joten se sopii hyvin esimerkiksi kuvien tai jäsentämättömän JSON-datan säilömiseen.

Blob Storage sisältää kontteja, jotka ovat kokoelmia ”blobeja”. Blobit ovat dataa, joita käsitellään kolmessa eri muodossa:

- Block blobit ovat lohkoista muodostuvaa teksti- tai binäärimuotoista dataa.
- Append blobit koostuvat lohkoista, kuten block blobit, joita on optimoitu liitäntätoiminnoille.
- Page blobit ovat tiedostoista koostuvaa dataa.

3.2.7 Restify

Restify on skaalautuva Node.js REST palvelin ratkaisu. Restify auttaa verkkopalveluiden tekemisessä oikein ja helposti. Esimerkiksi staattisen palvelimen luonti on erittäin helppoa, käyttäen Restify node-kirjastoa. Se ei tarvitse suurta määrää koodia (KUVA 2).

```
// Setup Restify Server
var server = restify.createServer({
  socketio: true
});

server.listen(process.env.port || process.env.PORT || 3977, function () {
  console.log("\n\n\n %s listening to %s", server.name, server.url);
});

// Static serving
server.get(
  "/",
  restify.plugins.serveStatic({ directory: "./public/", default: "index.html" })
); //Main page
```

KUVA 2. Restify palvelimen toteutus

REST on arkkitehtuurityyli rajapintojen toteuttamiseen. REST palvelin (Representational State Transfer) on palvelin, joka vastaa standardin mukaisiin create, read, update ja delete kutsuihin. Käyttäjäpäätteen tekemän REST kutsun tulee sisältää kaikki tieto, jonka palvelin tarvitsee ymmärtääkseen kutsun ilman tilatietoa palvelimella.

3.2.8 Azure Bot Service

Azure Bot Service on joukko työkaluja botin luontiin. Työkalujen avulla voi rakentaa botteja suoraan mukana tulevista malleista. (Microsoft Azure 2019d.)

Azure Bot Servicen tarjoamat työkalut ovat:

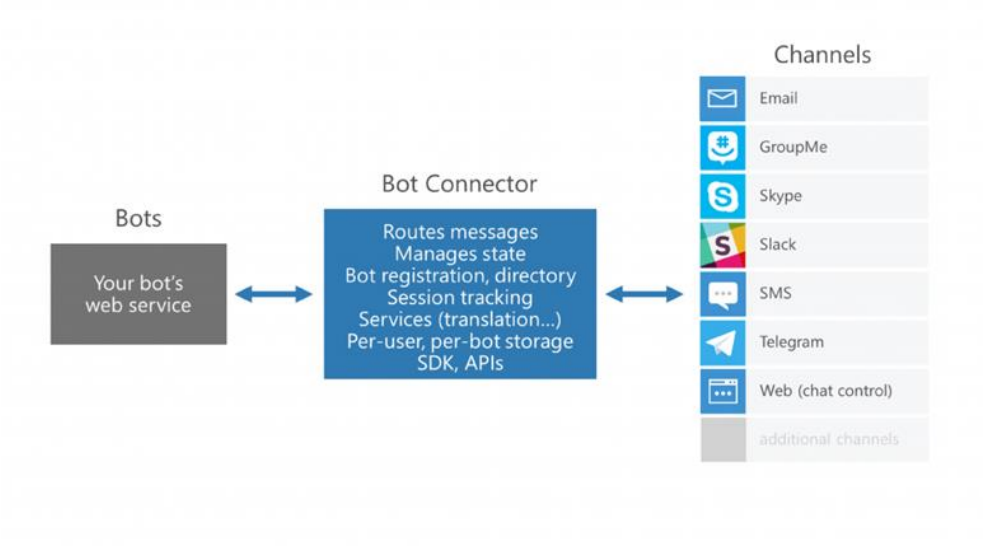
- Bot Framework SDK
- Bot Framework Tools
- Bot Framework Service (BFS)
- Bot deployment and channel configuration.

(Microsoft Azure 2019e.).

3.2.9 Microsoft Bot Framework

Microsoft Bot Framework on osa Azure Bot Service työkaluja ja on yksi pääkomponenteista chattibotin luomisessa Azureen. Bot Framework on isompi komponentti, joka koostuu pienemmistä komponenteista. Bot Frameworkin keskeiset komponentit ovat muun muassa Bot Connector, Bot Builder SDK ja Bot Directory.

Bot Connector muodostaa yhteyden toivottuihin kanaviin kuten sähköpostiin, SMS:ään taikka Slackkiin. Yhteyttä muodostaessa on botti rekisteröitävä, jonka jälkeen voidaan muodostaa yhteys ja julkaista botti Bot Directoryssa. (MSDN Blogi 2016a.)



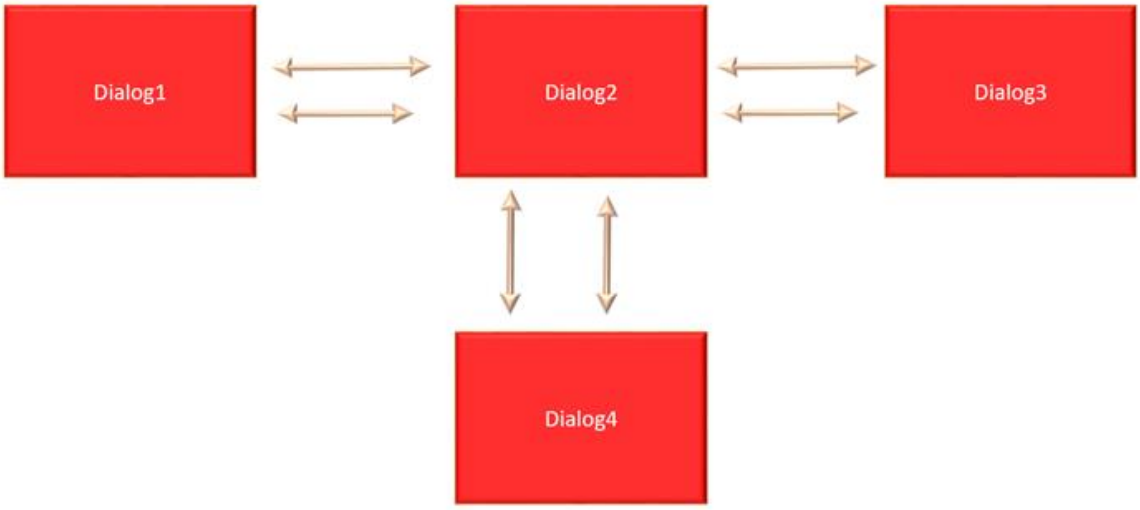
KUVA 3. Bot Connector toimintatapa (MSDN Blogi 2016b.)

Kuvasta (KUVA 3) nähdään Bot Connector komponentin toimintapa:

- vasemmalla kuvataan luotu botin www-palvelu
- keskellä on Bot Connector komponentti
- oikealla näkyy kanavia, joihin botti on mahdollista yhdistää.

Bot Connector välittää viestejä kanavien ja botin välillä. Komponentti suorittaa kaiken datan välityksen webbipalvelun ja halutun kanavan välillä.

Bot Builder SDK on avoimen lähdekoodin ohjelmistopaketti, jota ylläpidetään GitHubissa. Bot Builder SDK paketin avulla voidaan rakentaa dialogeja Node.js- tai C#-pohjaisiin botteihin. (MSDN Blogi 2016a.) Dialogirakennetta voidaan tarkastella kuvasta (KUVA 4).



KUVA 4. Dialogirakenne (MSDN Blogi 2016c.)

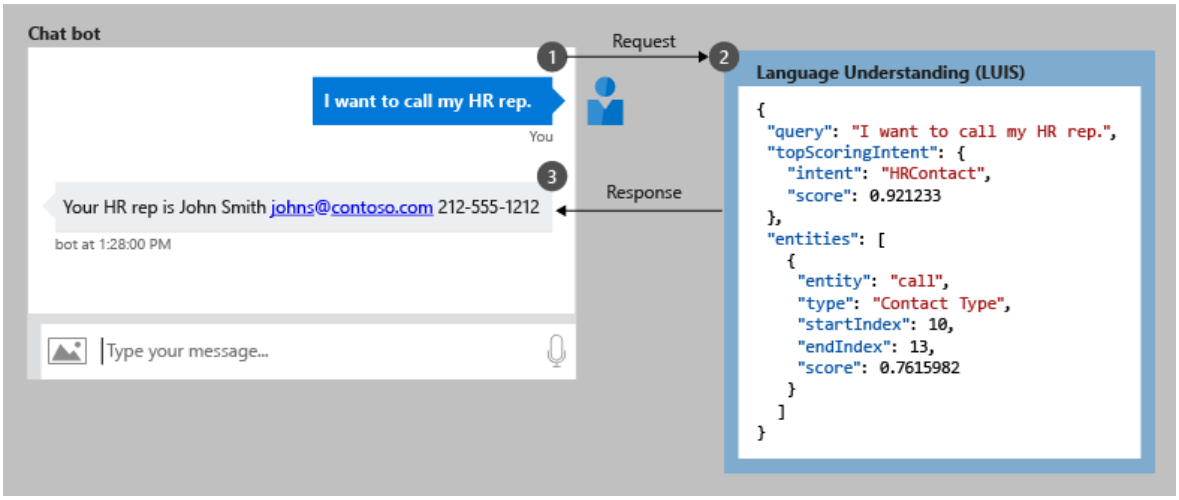
Kuvassa (KUVA 4) jokainen dialogi voi aloittaa keskustelun. Dialogista voidaan hypätä seuraavaan dialogiin, josta voidaan hypätä taikaisin edelliseen dialogiin tai jatkaa seuraavaan. Mistä dialogista tahansa voidaan palauttaa käyttäjälle viesti tai lopettaa prosessi.

Bot Framework -kirjaston keskeinen käsite on session-luokka. Session-luokka sisältää ominaisuuksia ja kutsuja, jotka auttavat luomaan keskustelun botin kanssa. Session mahdollistaa esimerkiksi viestien lähettämisen, dialogin aloittamisen ja dialogin lopettamisen. Session mahdollistaa myös tiedon välityksen dialogista toiselle. Session hallitsee keskustelua käyttäjän kanssa. (Microsoft 2020.)

3.2.10 LUIS

LUIS on kielen ymmärryspalvelu. LUIS auttaa applikaatioita ymmärtämään, mitä henkilö tarkoittaa heidän omilla sanoillaan. LUIS käyttää koneoppimista auttaakseen kehittäjiä rakentamaan applikaatioita, jotka voivat vastaanottaa käyttäjän syöttämää luonnollista kieltä ja poimimaan siitä tarkoituksen. (Microsoft Azure 2019a.)

LUIS toimii lähettämällä tekstiä LUIS natural language processing endpoint API rajapintaan, joka palauttaa vastauksen JSON muodossa kuten kuvassa (KUVA 5) on esitetty.



KUVA 5. LUIS toimintamalli (Microsoft Azure 2019c.)

Esimerkissä nähtävät vaiheet ovat seuraavat:

- Sovellus lähettää tekstin rajapintaan (1).
- LUIS käyttää oppimismallia tekstille ja palauttaa JSON muodossa vastauksen sovellukselle (2).
- Sovellus käyttää JSON vastausta jatkaakseen keskustelua (3).

(Microsoft Azure 2019c.)

LUIS voi käyttää valmiita malleja kielen ymmärtämiseen tai sille voidaan kehittää oma malli. Valmiit mallit ovat myös muokattavissa haluttuun käyttötarkoitukseen. Opetusmateriaalia lisätessä on kielenymmärrysmalli koulutettava uudestaan. Kouluttaessa luo LUIS uuden version kielenymmärrysmallista, jota voidaan käyttää.

3.2.11 Google Cloud Translation API

Google Cloud Translation API on Googlen tarjoama rajapinta käännöksiin, joka toimii dynaamisesti ja sillä voidaan kääntää tuhansia kielipareja. Palvelu on mahdollista integroida mihin tahansa, sillä se toimii pilvessä ja sen käyttämiseen hyödynnetään rajapintaa.

Node.js ympäristössä voidaan palvelu ottaa käyttöön asentamalla yksi node-kirjasto. Muilla ohjelmointikielillä kuten C# tai PYTHON käyttöönotto on vastaavanlaista.

3.2.12 OpenCV

OpenCV (Open Source Computer Vision Library) on avoimen lähdekoodin konenäkö- sekä koneoppikirjasto, joka on tarkoitettu nopeuttamaan konenäön integrointia kaupallisiin tuotteisiin (OpenCV 2020). Kirjastoa voidaan käyttää esimerkiksi ihmisten tunnistamiseen

kuvasta tai videosta. Avoin lähdekoodi tarkoittaa, että kirjasto on myös muokattavissa omiin tarpeisiin.

OpenCV on kirjoitettu C++ ohjelmointikielellä. Useille ohjelmointikielille on tehty versioita kirjastosta, jotka mahdollistavat käytön useammassa ympäristössä. OpenCV voidaan esimerkiksi ottaa selainkäyttöön hyödyntämällä kirjaston WebAssembly versiota. Eri alustoilla toimivat versiot kirjastosta eivät välttämättä sisällä kaikkea toiminnallisuutta.

3.2.13 Azure Face

Azure Face on Microsoftin tuottama kognitiivinen palvelu, jolla voidaan mm. vertailla ja tunnistaa kasvoja, saaden näistä informaatiota, kuten minkä ikäinen henkilö kuvassa on. Face palvelulle lähetetään kuva, jonka tämä analysoi ja palauttaa tietoja kuvassa olevasta henkilöstä tai henkilöistä. Face-palvelun palauttama data kertoo esimerkiksi mistä kohdasta kuvaa kasvot löytyvät, henkilön hiusten värin, sukupuolen, iän ja se ilmoittaa esimerkiksi mahdollisista laseista, kuulokkeista tai muista somisteista.

Azure Face rajapinnan käyttö tapahtuu lähettämällä kuva binäärimuodossa rajapintaan. Riippuen rajapinnasta palvelu palauttaa esimerkiksi kasvojen sijainnin kuvassa, tietoa henkilön piirteistä (esim. hiusten väri ja ikä) tai kasvojen tunnetilan.

4 YLEISTASON TOTEUTUS

4.1 Vaatimukset

Työ tehtiin Enfo Oyj:n toimesta yrityksen sisäisenä projektina. Asiakas halusi tuotteen näytteenä firman osaamisesta sekä mallikappaleena potentiaalisille sekä nykyisille asiakkaille.

Chattibotin oli toimittava aulapalveluna eli vastaanottajana. Botin oli tarkoitus ymmärtää normaalia kieltä. Botille haluttiin myös ympäristö, josta tulisi botin persoona esiin muussakin muodossa kuin tekstinä. Vaatimuksia alettiin kartoittamaan projektin aloituspalavereissa.

4.2 Kartoitus

Kartoitus aloitettiin listaamalla tarpeet ja tutkimalla niihin sopivia ratkaisuja hyödyntäen hakukoneita sekä aiempaa kokemusta mahdollisista hyödynnettävistä teknologioista. Kartoitettavia tarpeita olivat seuraavat:

- missä ympäristössä palvelu toteutettaisiin
- millä teknologioilla palvelu toteutettaisiin
- miten palvelu tuotaisiin käyttäjälle.

Ympäristöksi valittiin Azure, sillä aikaisempaa kokemusta löytyi kollegoilta jo ennestään. Azure oli myös helppo ympäristö toteuttaa monenlaisia ratkaisuja eikä se aseta monia rajoitteita. Azure mahdollisti lähes vapaat kädet teknologioiden valinnassa, vaikka valmistajan omat palvelut toimivat ympäristössä varmemmin.

Botin luontiin teknologiaksi valittiin Microsoft Bot Framework, sillä kehitystiimistä löytyi jo aikaisempaa kokemusta Microsoftin palveluiden osalta. Bot Framework oli helppo testata, sillä sille löytyy oma emulaattori, jolla voidaan testata bottia lokaalisti. Testauksen onnistuminen lokaalisti nopeuttaa työtä, sillä palvelua ei tarvitse toimittaa erikseen mihinkään ympäristöön, jossa sitä pitäisi testata.

Kielen ymmärrykseen valittiin LUIS, jonka tuottaja on Azure. LUIS valittiin muun muassa helpon integraation takia. Kielen ymmärrykseen tarvittiin ohelle myös kääntäjä, sillä LUIS ei ymmärrä Suomea suoraan. Kääntämiseen valittiin Googlen Cloud Translation rajapinta. Googlen rajapinta valittiin, koska muut vaihtoehdot kehitystiimin testauksessa antoivat useammin huonoja käännöksiä.

Palvelun kasvojen tunnistukseen valittiin käytettäväksi OpenCV kirjasto. OpenCV valittiin koska se on aikaisemmin todettu toimivaksi ja sen käyttö on ilmaista. Azure Face palvelua käytettiin alustansa takia kasvojen analysointiin tunnistuksen jälkeen.

4.3 Ympäristö

Toteutus kokonaisuudessaan toimii Azuren pilvipalvelussa ja ei näin vaadi omaa palvelinta toimiakseen. Ympäristön pääkomponentteina toimii Azuren App Service alusta. Kummallekin sovelluksen pääkerrokselle on tehty oma App Service. App Servicet ovat automaattisesti saatavissa Azuren palveluille luoduista linkeistä. Halutessaan palveluille voisi määrittää omat DNS asetukset ja toimittaa sovellukset omista osoitteista. DNS määrityksiä ei katsottu tarpeelliseksi palvelulle.

Pääkerrosten lisäksi palvelu hyödyntää Azuren Blob storage palvelua, jossa säilytetään jäsentämätöntä JSON-dataa sekä kuvia BASE64-koodattuna kasvojen tunnistusta varten. Blob storagea hyödynnetään Azuren Cognitive Search palvelua käyttäen, jonka avulla data voidaan löytää nopeasti ja tehokkaasti jäsentämättömyydestä huolimatta.

4.4 Arkkitehtuuri

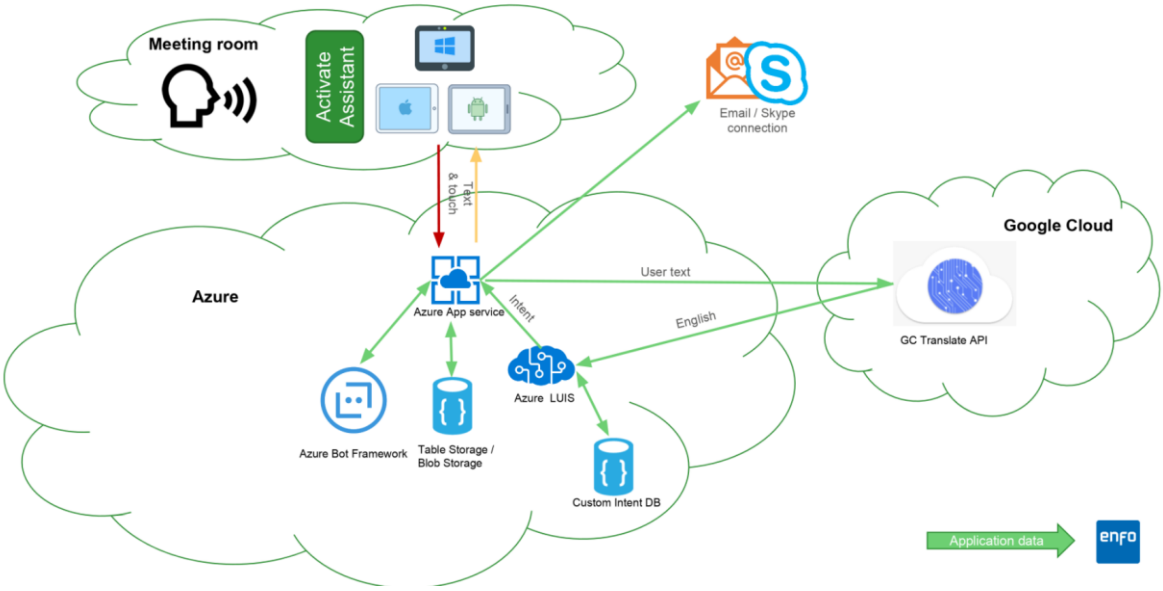
Projektissa on sovellettu hajautetun tietojenkäsittelyn periaatteita, jossa on jaettu projekti kahteen osaan tai kerrokseen. Kerrokset toimivat omina kokonaisuuksinaan, jotka kommunikoivat keskenään. Nämä kerrokset ovat toisistaan riippumattomia ja halutessa voitaisiin korvata jollakin muulla toteutuksella sillä rajoitteella, että kommunikaatio sovellusten välillä tapahtuu noudattaen määritettyä rakennetta. Kerrokset ovat käynnissä omissa Azure App Service ympäristöissään, mutta saman Azure-asiakkuuden sisällä.

Chattibottikerros on projektin osa, joka sisältää botin toiminnan ja ”business-logiikan”. Chattibottikerros käsittelee keskusteludialogirakennetta, sekä datakyselyitä Blob storagesta. Tämä kerros käsittelee chattibotin tekstin käännön ja sen tulkitsemisen sekä yhdistämisen tarkoitukseen, jotta voidaan ohjata käyttäjä oikealle keskustelupolulle. Käännöksiä sekä tarkoitukseen linkittämistä hoitavat Googlen Cloud translation API-palvelu, sekä LUIS kielenymmärryspalvelu. Keskustelupoluissa logiikka perustuu käyttäjän vastauksiin dialogia käydessä.

Blob storageen on tallennettu kasvojentunnistus dataa ja JSON-dataa erilaisista yrityksistä sekä heidän yhteyshenkilöistään. Blob storage voi pitää sisällään lähes minkälaista dataa tahansa, mutta tämä data on jäsentämätöntä. Jäsentämätöntä dataa käsitellään Azure Cognitive Search palvelulla, jossa on sisäänrakennetut ominaisuudet etsiä datasta haluttua osaa tehokkaasti. Kaikki kutsut Blob storageen tehdään siis Cognitive Search

palvelun läpi, joka palauttaa hakumallilla löydetyn tuloksen, joka on samankaltaisin hakutermin kanssa.

Front-end kerros sisältää sen mitä käyttäjälle halutaan näyttää, eli www-sivun ja integroidun chattibottikehyksen. Kerros sisältää myös kasvojentunnistuslogiikan, joka kertoo chattibottikerrokselle, kun käyttäjä saapuu päätepisteelle ja onko käyttäjä liitetty johonkin asiakastietoon. Kasvojentunnistus on myös hajautettu kahteen osaan, selaimen sisäiseen kasvojen tunnistukseen, joka on toteutettu hyödyntäen OpenCV kirjastoa, ja tämän dataa hyödyntävään toiseen osaan, joka hakee tietoa kasvoista käyttäen Azure Face kognitiivista palvelua. Kasvojentunnistus oli toteutettava front-end toteutuksessa, sillä videon välittäminen back-end toteutukselle loisi turhaa datan välitystä kerrokselta toiselle ja sitä haluttiin välttää.

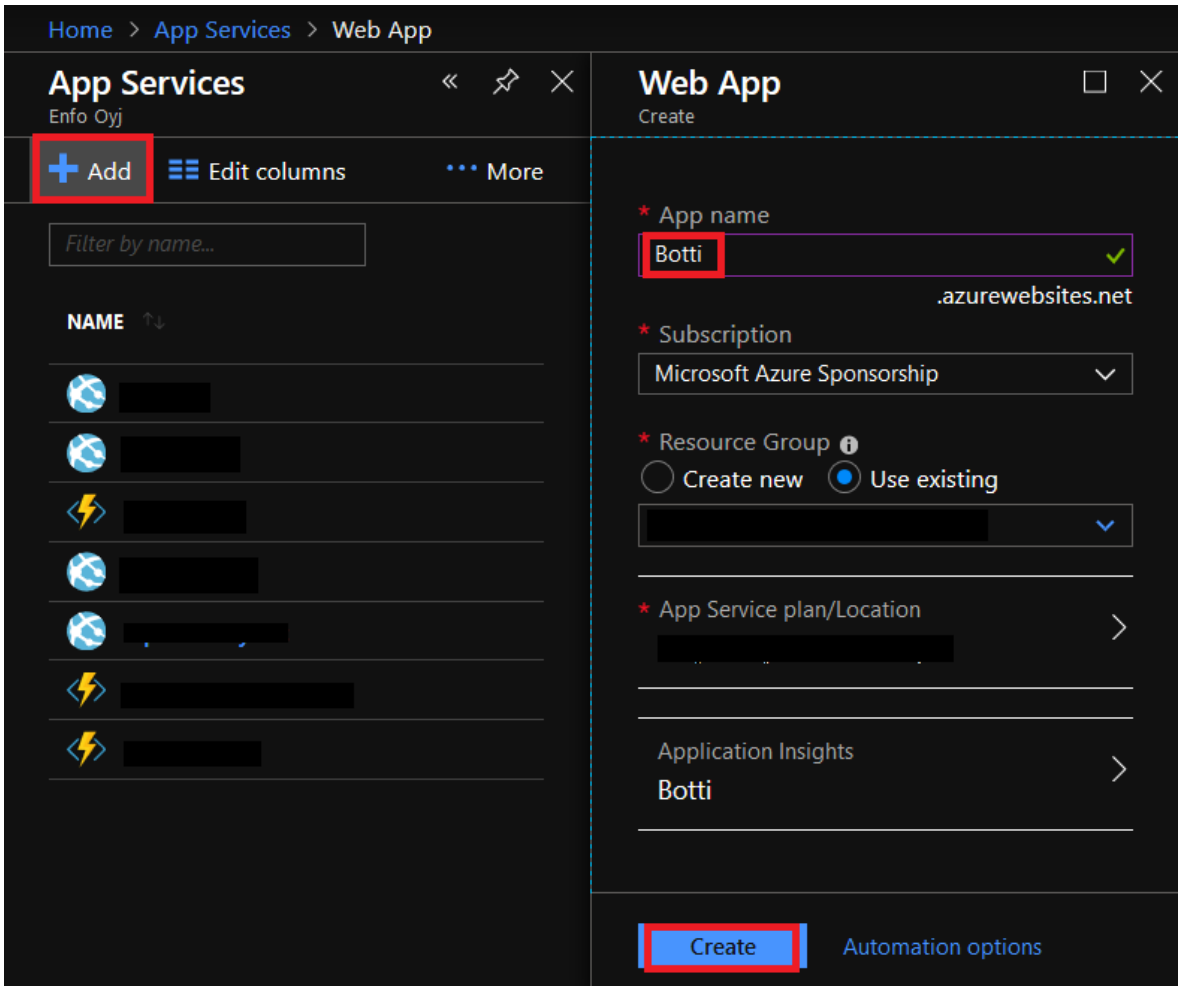


KUVA 6. Chattibotin arkkitehtuurin

5 CHATTIBOTIN TOTEUTUS

5.1 Ympäristön asennus

Ympäristön asennukset ja projektien aloittamiset ovat suoraviivaisia prosesseja. Botille on ensin tehtävä App Service, Azuren omassa portaalissa, kuvan (KUVA 7) näyttämällä tavalla. Luotava applikaatio on tyyppiä ”web app”, sillä sen on pystyttävä toimimaan verkossa. Kun App Service on luotu, voidaan siihen tuoda toteutus ja se suoritetaan palveluna Azuren portaalissa. Tämä prosessi on tehtävä uudestaan myös front-end kerrokselle.



KUVA 7. Web App luonti Azure portaalissa

5.2 Back-end

Back-end kerros, joka sisältää botin toiminnan, toteutettiin Node.js projektina, joka tuotiin Azure portaalin Web App projektiin. Node.js päällä käytetään Restify Node-moduulia eli pakettia, jonka avulla tuodaan REST ominaisuuksia palvelimeen. Node -projektissa Bot Framework palveluun luodaan yhteys käyttämällä Node-moduulia, jonka voi ottaa

käyttöön, kuvan (KUVA 8) esittämällä tavalla. Ennen pakettien käyttöä on paketit asennettava projektin riippuvuuksiin, joka tapahtuu Node-projektissa lisäämällä riippuvuudet package.json tiedostoon kuvan (KUVA 9) osoittamalla tavalla.

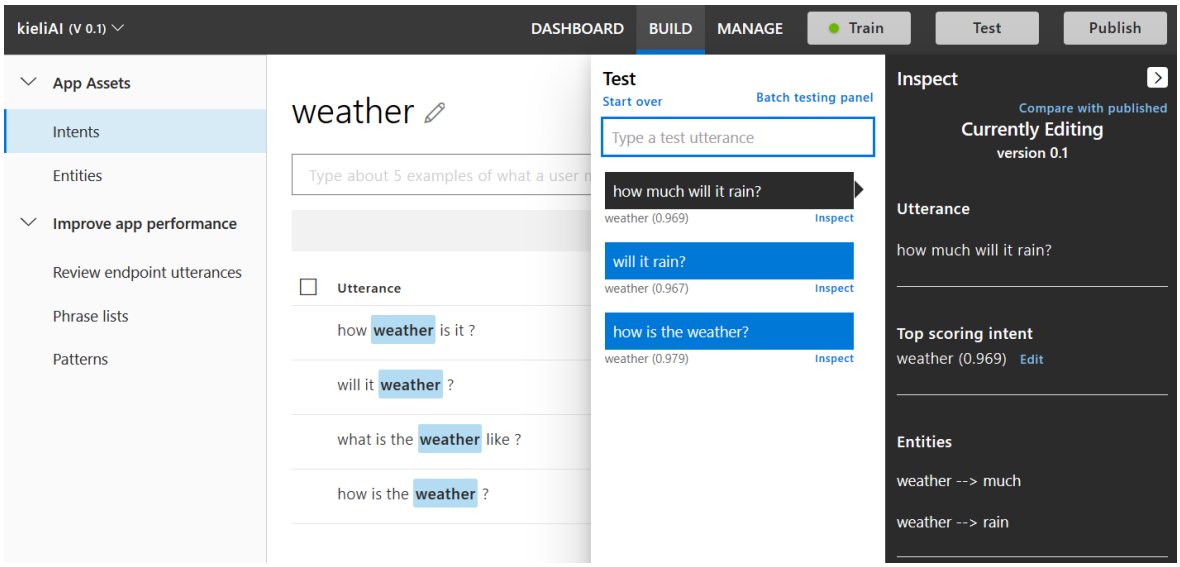
```
var builder = require("botbuilder");

var connector = new builder.ChatConnector({
    appId: process.env.MicrosoftAppId,
    appPassword: process.env.MicrosoftAppPassword,
    openIdMetadata: process.env.BotOpenIdMetadata
});
```

KUVA 8. Botbuilder-moduulin tuonti projektiin

```
{
  "name": "enfo-meeting-room-chatbot",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
  },
  "author": "Enfo Oyj",
  "license": "ISC",
  "dependencies": {
    "botbuilder": "^3.13.1",
    "botbuilder-ai": "^4.0.0-preview1.2",
    "botbuilder-azure": "^3.0.4"
  },
  "devDependencies": {
  }
}
```

KUVA 9. Suppea package.json

Bottia varten oli luotava Web App Bot päätepiste -palvelu Azuren portaalissa, johon Node-palvelu ottaa yhteyden. Tämän jälkeen tapahtumia botissa pystytään hallinnoimaan käyttäen botbuilder-kirjaston omia toiminnallisuuksia. Botbuilder-kirjasto hoitaa kaiken käyttäjän ja botin välisen datan siirtämisen, kuten esimerkiksi tekstin ja mahdollisten kuvien välittämisen.



KUVA 10. LUIS applikaation käyttöliittymä

Kielen ymmärrys bottiin tuotiin LUIS palvelulla. Kielenymmärrystä varten on otettava käyttöön LUIS sovellus, jota voidaan opettaa ymmärtämään erilaisia lausahduksia. Erilaisista lausahduksista LUIS oppii ymmärtämään vastaavanlaisia lausahduksia ja arvioi aina kuinka varma on löytämästä tarkoituksestaan, kuten kuvasta (KUVA 10) voidaan todeta. Voidaan myös todeta, että LUIS osaa päätellä erilaiseen lauseen liittyvän samaan tarkoitukseen, tässä tapauksessa halutaan tietää säästä.

Toiminta LUIS-palvelussa perustuu ”utterances”, ”intents” ja ”entities” määritteisiin. Intents eli tarkoitukset ovat erilaisia lauseita, joihin voidaan linkittää samaa tarkoittavia lauseita, tarkoituksia voidaan miettiä kategorioina. Entiteetit ovat osia lausahduksista, jotka kertovat tarkoituksen kohteen. Lausahdukset eli ”utterances” ovat käyttäjän syöttämää tekstiä. Käyttäjän kysyessä esimerkiksi ”Millainen sää on Espoossa?” voidaan lausahdus liittää sää tarkoitukseen, ja tästä voitaisiin ottaa entiteettinä Espoo, joka olisi tarkoituksen kohde, näin LUIS tietäisi käyttäjän pyytävän säätä Espoosta.

Koska LUIS ei ymmärrä Suomea, oli se myös otettava huomioon työssä ja tämä tarve hoidettiin käyttämällä Googlen Cloud Translation -rajapintaa. Botbuilder-kirjastossa on valmiit puitteet kielen ymmärrystä tukemaan ja tämä tapahtui hyödyntäen kirjaston omia metodeja. Normaalista kielen ymmärryksestä poiketen pyyntö LUIS-palveluun tehdään pyytäen ensin käännöstä Googlen rajapinnasta, jonka jälkeen voidaan jatkaa pyyntöä LUIS-palveluun.

Keskustelujen ohjaukseen käytetään Bot Framework -kirjaston dialogi ominaisuutta. Dialogeilla voidaan keskustelua ohjata haluttuun suuntaan ja aktivoida eri keskustelupolkuja

perustuen käyttäjän viestiin. Kun käyttäjä lähettää viestin botille, tämä ohjataan käännettynä LUIS-kirjastolle, joka on asetettu Node-moduulista tehdyn botin ”recognizer” luokalle. LUIS toimii myös omana Node-moduulinaan, joka voidaan ottaa käyttöön syöttämälle kirjastolle käytettävän tilin tunnukset. LUIS otetaan käyttöön Bot Framework -kirjastoa hyödyntäen ja tämä tapahtuu yhdellä rivillä koodia tunnusten syötön jälkeen kuvan (KUVA 11) mukaisesti. Kuvassa ”recognizer” on LUIS luokasta tehty objekti, johon on syötetty käyttäjätunnisteet.

```
bot.recognizer(recognizer);
```

KUVA 11. Botin recognizer-objekti

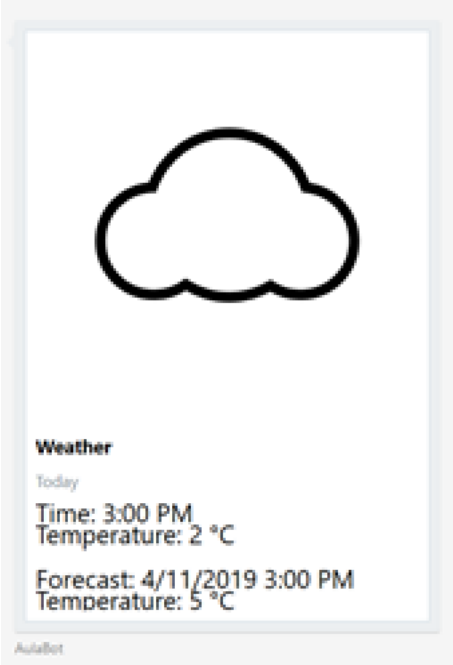
Bottia käyttäessä käyttäjällä on kaksi mahdollista tilaa, käyttäjä voi olla dialogissa tai dialogin ulkopuolella. Dialogin ulkopuolella käyttäjän syöte botille ohjataan LUIS-kirjastolle, mutta dialogissa käyttäjän syöte tarkastetaan suoraan dialogin sisällä ilman tarkoituksen ymmärrystä, koska vaihtoehtoja on rajatusti, ei tarkoituksen ymmärrykselle ole tarvetta dialogirakenteen sisällä.

```
weather.dialog('/', function (session, next) {
  let options = { method: 'GET', uri: weatherURI(), resolveWithFullResponse: true };
  request(options)
  .then( function (response) {
    parseXML(response.body, function (err, result) {
        weatherData = result;
    });
    parsedWeatherData = parseData(weatherData);
    return "weaS " + parsedWeatherData;
  })
  .then( (result) => {
    let imageData = result.image.toString();
    let imageData2 = Buffer.from(result.tomorrow.image).toString('base64');
    let resultCard = new builder
    .HeroCard(session).title("weatherTitle").subtitle("weatherTitleSub").text(weatherDataAsHTML(result, imageData2))
    .images([
      builder.CardImage.create(session, "data:image/svg+xml,"+ imageData)
    ]);

    session.send(new builder.Message(session).addAttachment(resultCard));
    session.send("default_ending");
  });

  session.endDialog();
});
```

KUVA 12. ”HeroCard” palaute

Dialogin aktivoituessa noudatetaan rakennetta, joka dialogille on määritetty koodissa. Toteutuksessa esimerkiksi sään kysely aktivoi ”weather” dialogin, joka tekee käyttäjälle

kortin, näyttäen sen hetkisen sään ja ikonin kyseiselle säätilalle kuvan (KUVA 12) esittämällä tavalla. Dialogeissa botin on mahdollista palauttaa esimerkiksi pelkkää tekstiä, kortteja, painikkeita ja kuvia. Kuvasta (KUVA 13) voidaan nähdä Bot Framework -kirjastolla luotu ”HeroCard”, joka voi sisältää esimerkiksi tekstiä, painikkeita, kuvia ja joillakin laitteilla kosketustoimintoja. Sisältö kortille tuodaan syöttämällä tekstiä HTML muodossa ja käyttämällä HeroCard-luokan sisäänrakennettua toiminnallisuutta.

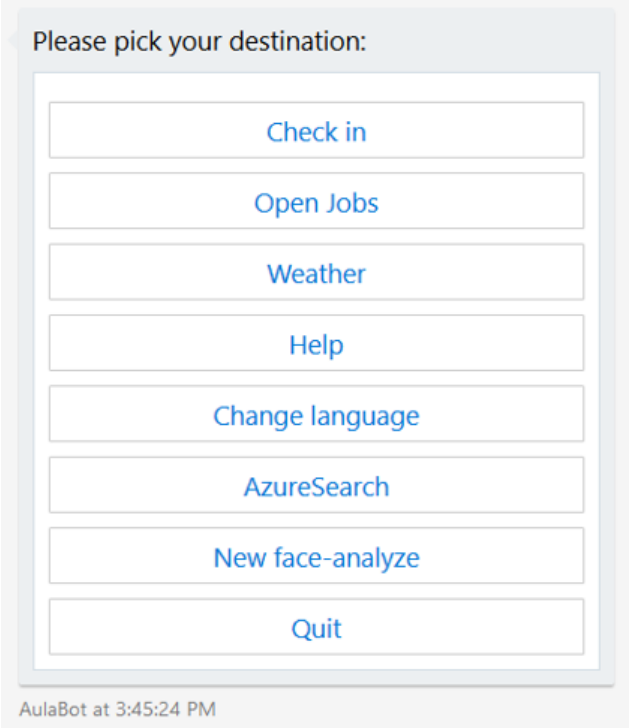


KUVA 13. ”Hero Card” käyttöliittymässä

Dialogeissa käytetään usein session muuttujaa, joka on aina läsnä botin ollessa yhdistettynä keskusteluun. Session voi sisältää koodissa siihen tallennettua dataa esimerkiksi keskustelusta käyttäjän antamia vastauksia kysymyksiin, mutta session sisältää myös funktionaalisuutta. Session sisään rakennettuja funktioita käyttämällä voidaan esimerkiksi lähettää viestejä käyttäjälle, aloittaa dialogi ja lopettaa dialogi. Monet toteutuksen dialogit käyttävät yksinkertaista ”prompts and responses” rakennetta (KUVA 14), jossa kysellään käyttäjältä kysymyksiä ja voidaan vastata käyttäen käyttäjäsyötteitä hyödyksi. Toteutuksessa käytetään usein painikkeita (KUVA 15) pyytäessä käyttäjäsyötettä. Tätä rakennetta hyödyntäen on helppoa kerätä käyttäjältä kysymyksiin vastauksia.

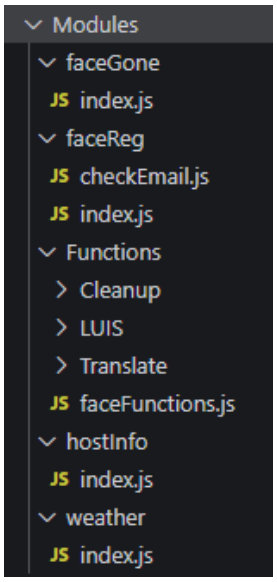
```
var builder = require('botbuilder');
const basicDialog = new builder.Library('basicDialog')

basicDialog.dialog('/', [
    function (session, args, results, next) {
        session.dialogData.firstPrompt = args;
        builder.Prompts.text(prompts(1));
    },
    function (session, results, next) {
        session.dialogData.secondPrompt = results.response;
        builder.Prompts.text(session, prompts(2))
    },
    function (session, results, next) {
        session.send(searchResult());
        session.endDialog();
    }
]);

module.exports.createLibrary = function () {
    return basicDialog.clone();
};
```

KUVA 14. Yksinkertainen dialogi



KUVA 15. ”Prompti” käyttöliittymässä

Kokonaisuudessaan back-end kerros koostuu monesta osasta ja funktiosta, joita on eriytetty omiin moduuleihinsa (KUVA 16). Eriytettyjä moduuleja ja funktioita yhdistää Node-projektissa tyypillisesti app.js tiedosto, jossa sijaitsee palvelimen toiminta ja kutsut erilaisille funktioille ja moduuleille. Dialogien eriytys tapahtuu tekemällä niistä moduuleja (KUVA 17). Dialogi-moduulia voidaan kutsua palvelussa sen relatiivista kansiopolkua käyttäen (KUVA 18). Moduloinnilla tehdään koodista helppolukuisempaa ja helpotetaan myöhempää kehitystä järjestämällä koodikanta selkeisiin osiin.



KUVA 16. Työn modulointi

```
module.exports.createLibrary = function () {
    return weather.clone();
};
```

KUVA 17. Komponentin modulointi

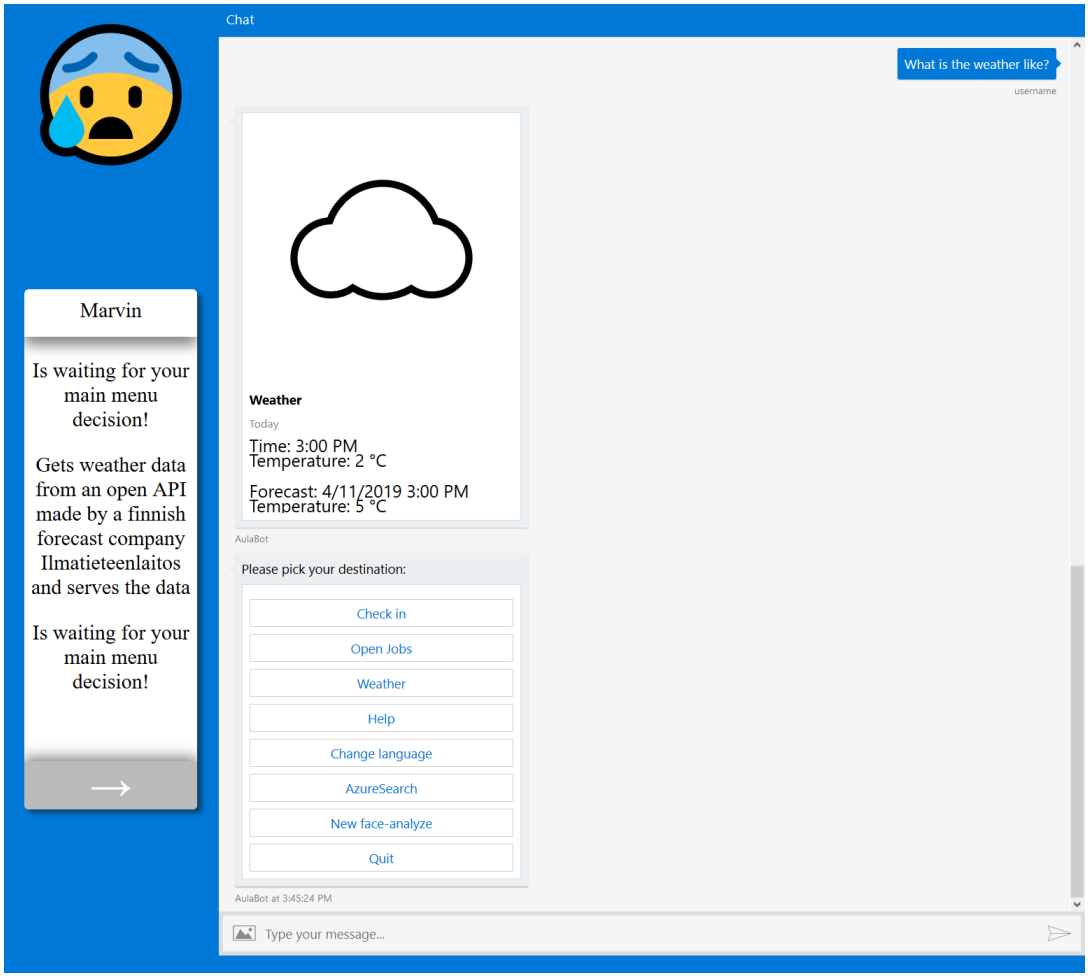
```
bot.library(require("./Modules/weather").createLibrary());
```

KUVA 18. Komponentin käyttö palvelussa

5.3 Front-end

Käyttöliittymä (KUVA 19) koko työlle luotiin erillisenä Node-projektina, jossa upotettiin botti sivulle käyttäen HTML iframe elementtiä. Botin ”persoonaa” haluttiin tuoda esille, joka toteutettiin vaihtuvilla hymiöillä sivun yläkulmassa. Hymiö sivulla vaihtuisi esimerkiksi kuvan näyttämäksi, jos lämpötila säätä tarkastellessa olisi liian pieni. Sivu kertoo taustalla tapahtuvasta toiminnasta laatikossa keskustelun vieressä tekstimuodossa. Keskustelu tapahtuu sivun päälaatikossa, jonka alareunassa on tekstisyötekenttä käyttäjälle. Käyttäjän viestit näkyvät keskusteluikkunan oikeassa reunassa ja botin viestit vasemmassa reunassa.

Keskusteluikkuna käyttää pääasiassa Bot Framework -kirjaston ennalta määriteltyä tyylittelyä, mutta sen vieressä näkyvät elementit ovat itse tyyliteltyjä. Työssä on käytetty pelkkiä CSS-määrittelyjä tyylittelyihin, sillä projektia varten ei katsottu tarpeelliseksi käyttää mitään tyylikirjastoa tai muuta CSS-kieleen kääntyvää, kuten SCSS. Tyylimäärittelyt on pidetty yksinkertaisina (KUVA 20) ja helppolukuisina.

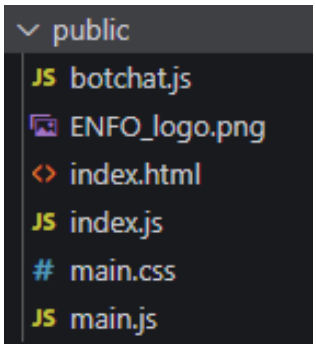


KUVA 19. Käyttöliittymä

```css
.avatars {
    font-size: 7em;
}

#botDiv {
    position: static;
}

#botThinking > ul {
    list-style-type: none;
    margin: 0 auto;
    padding: 2%;
    top: 0; left: 0; bottom: 0; right: 0;
    overflow: hidden;
}

#botThinking > ul > li {
    margin: 5% 0 5% 0;
}

.bubbleContainer {
    z-index: 999;
    position: relative;
    margin: 10% auto;
}

.bubbleHeader {
    position: relative;
}
```

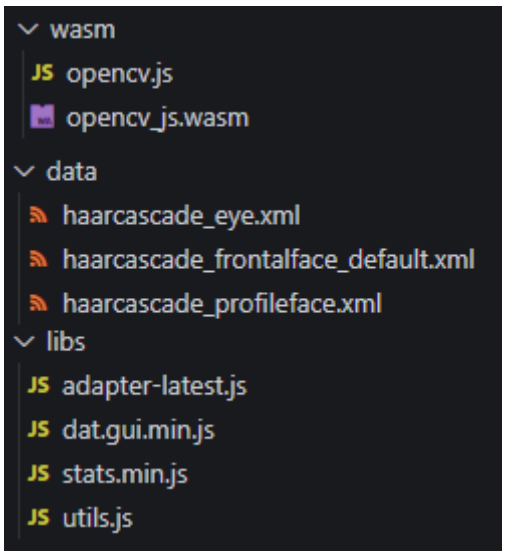
KUVA 20. CSS-määrittelyt.

Käyttöliittymän rakenteesta on tehty myös yksinkertainen. Rakenne koostuu HTML, JS ja CSS tiedostoista sekä yhdestä PNG tiedostosta, joka sisältää yrityksen logon (KUVA 21), nämä tiedostot ovat Node-projektin public kansiossa.



KUVA 21. Projektin sisäinen tiedostorakenne

Käyttöliittymässä on paljon JavaScript toiminnallisuutta, jolla toteutetaan kasvojentunnistuksen prosessi ja yhteydenotto bottiin käyttäjän saapuessa päätepisteelle. Front-end kerrokseen kasvojen tunnistusta varten on tuotu OpenCV-kirjaston WebAssembly-versio (KUVA 22), joka on koottu projektia varten. Normaalissa muodossaan OpenCV tarvitsee esimerkiksi Windows-laitteella Python kielen ja C-kielien ominaisuuksia, sekä muita erilaisia kirjastoja toimiakseen. Kootussa muodossaan saadaan OpenCV toimimaan JavaScript-moottoria hyödyntäen ja tämä riittää toimintaan esimerkiksi selaimessa.



KUVA 22. OpenCV-kirjaston tiedostot

Kasvojentunnistukseen OpenCV kirjastolla tarvitaan päätelaitteessa kamera. Videon tarvitsee olla tarpeeksi korkearesoluutioinen, jotta kasvoja voidaan tunnistaa. Koodia kasvojentunnistukseen suoritetaan front-end kerroksessa käyttäen selaimen JavaScript-moottoria.

```
navigator.mediaDevices
  .getUserMedia({ video: resolution, audio: false })
  .then(function(s) {
    stream = s;
    video.srcObject = s;
    video.play();
  });
```

KUVA 23. Kameran käynnistys selaimessa

```
canvasInput = document.createElement("canvas");
canvasInputCtx = canvasInput.getContext("2d");
canvasBuffer = document.createElement("canvas");
canvasBufferCtx = canvasBuffer.getContext("2d");

srcMat = new cv.Mat(videoHeight, videoWidth, cv.CV_8UC4);
grayMat = new cv.Mat(videoHeight, videoWidth, cv.CV_8UC1);

faceClassifier = new cv.CascadeClassifier();
faceClassifier.load("haarcascade_frontalface_default.xml");

requestAnimationFrame(processVideo);
```

KUVA 24. OpenCV videon tarkastelu canvas-elementtillä

Selainpäässä käynnistetään ensin videon toisto päätelaitteen kameraa hyödyntäen (KUVA 23). Jos videon toistossa onnistutaan, syötetään video canvas-elementille (KUVA 24) ja otetaan käyttöön OpenCV-kirjasto, joka käyttää avointa dataa kasvojen tunnistamiseen.

```
if (faceVect.size() ≠ 1) {
  noFace += 1;
  oneFace = noFace ⩾ 20 ? 0 : oneFace;
}
else {
  noFace = 0
  oneFace += 1;
}

// check whether faces have been detected for long enough
faceFound = oneFace == 150 ? true : false;

if (faceFound) {
  takePicture();
  faceIsPresent = true;
}
```

KUVA 25. Kasvojen tunnistuslogiikkaa

Sovellus alkaa tunnistamaan kasvoja ja tunnistaessaan kasvot noin 1.5 sekuntia yhtäjaksoisesti otetaan niistä kuva. Tunnistamalla kasvoja yhtäjaksoisesti pidempään vältytään keskustelun aktivoitumiselta, kun kameran edessä näkyvä henkilö ei ole jäämässä päätelaitteelle. OpenCV kutsuu processVideo-funktiota jokaisella kuvan päivityksellä. Videon kuva päivittyy noin 100 kertaa sekunnissa. Funktiossa tarkastellaan kasvojen yhtäjaksoista esiintymistä videossa kasvattamalla numeroa joka päivityksellä ja ehtojen täyttyessä otetaan kuva kasvoista (KUVA 25). Funktiossa on huomioitu noin 0.2 sekunnin virhemarginaali kasvojen hetkelliselle katoamiselle esimerkiksi pään liikkeen tai huonon kameran resoluution takia.

```
$.ajax({
  type: "POST",
  url: window.location.href + 'image',
  data: pictureObj,
  success: function (resdata) {
    if (typeof (resdata) == 'string') {
      newUser(resdata);
      sendResponse(resdata);
    } else {
      recognizedUser(resdata);
    }
  }
});
```

KUVA 26. Kasvojen analysoinnin kutsu

Kun kasvoista on saatu kuva, lähetetään tämä kuva palvelimelle (KUVA 26) ja pyydetään palvelinta analysoimaan kuva ja tunnistamaan onko käyttäjästä jo olemassa olevaa kuvaa tallennettuna tietokannassa. Jos aikaisempi kuva löydetään, osaa palvelin etsiä sitä kuvaa vastaavat tiedot käyttäjästä ja chattibotti voi ilmoittaa tunnistavansa käyttäjän.

5.4 Jatkokehitys ja parannukset

Toteutus on toimiva ja sopii hyvin demokäyttöön, mutta arkkitehtuurin ja rakenteen osalta parantaminen olisi suotavaa. Hajauttamalla arkkitehtuurissa kokonaisuuksia selvempiin osiin saataisiin jatkokehityksestä helpompaa. Kaikki autentikointi voitaisiin toteuttaa back-end kerroksessa ja näin toteutuksessa olisi vähemmän haavoittuvaisia kohtia. Tarkempi dokumentointi helpottaisi jatkokehitystä.

Työtä toteuttaessa tuli ilmi, että back-end toteutusta olisi mahdollista muuttaa modulaarisemmaksi, kuin nykyisessä toteutuksessa. Monissa komponenteissa on samantyyppistä logiikkaa uudelleen kirjoitettuna ja näitä voisi yhdistää komponentteihin. Komponenttimaista rakennetta ei hyödynnetty toteutuksessa tarpeeksi.

Nykyinen ratkaisu ei huomio turvallisuutta kovinkaan hyvin ja tätä voisi kehittää, tuomalla turvallisuuteen keskittyviä moduuleita ja käyttää näitä hyödyksi. Yhtenä esimerkkinä voitaisiin hyödyntää Node-moduulia Retire.js. Retire.js skannaa Node.js sovelluksen riippuvuuksia ja tarkistaa onko käytössä haavoittuvaisia versioita kirjastoista. Azure Face palvelun käyttö voitaisiin myös siirtää front-end kerroksesta back-end kerrokseen, sillä vain OpenCV-kirjaston on toimittava front-end kerroksessa ja palvelun pitäminen front-end

kerroksessa luo mahdollisen haavoittuvuuden, jolla voitaisiin saada palvelun autentikointitunnukset.

Palveluun voitaisiin myös lisätä käyttäjän todennustaso, jotta vain halutut käyttäjä voisivat käyttää palvelua. Käyttäjän todennus toisi sovellukselle myös lisää turvallisuutta, sillä vain käyttäjätunnuksen omaava käyttäjä pääsee käsiksi palveluun. Hyvällä todennuksella estetään myös sovelluksen käyttö muilta kuin ihmisiltä.

Front-end kerrosta olisi helpompi jatkokehittää, jos se olisi tehty alusta saakka jollakin ohjelmistokehyksellä. Jos nyt halutaan tuoda monimutkikkaampaa logiikkaa tai toimintaa käyttöliittymään, vaati se joko paljon ominaisuuksien käsin tekoa tai koko kerroksen migraatiota ohjelmistokehykseen. Ohjelmistokehyksen tarkoitus on nopeuttaa ja helpottaa kehitysprosessia.

Yksikkötestaustason lisäämisellä huomattaisiin mahdolliset virheet koodissa jo kehitysvaiheessa. Testien lisääminen olisi iso prosessi, mutta tuotannollisessa versiossa tämä olisi kannattavaa käyttäjäkokemuksen parantamisen tähden. Yksikkötestaus voitaisiin toteuttaa esimerkiksi omana osuutena julkaisuputkea, jolloin päivityksiä ei voitaisi toteuttaa ennen testien suorittamista onnistuneesti.

6 YHTEENVETO

Työn tavoitteena oli tehdä kognitiivinen chattibotti, joka voi tunnistaa kasvoja ja ymmärtää kieltä käyttäen Bot Framework -ohjelmistokehystä sekä muita Microsoftin tuottamia palveluita. Työ toteutettiin asiakkaana toimineen Enfo Oyj:n toimesta. Työ tehtiin esittämään yrityksen kykyä toteuttaa uudenlaisia ohjelmistoratkaisuja.

Valmis toteutus kykenee tunnistamaan käyttäjän saapumisen päätepisteelle, aloittamaan keskustelun tämän kanssa tunnistaen onko käyttäjästä aikaisempaa tallennettua tietoa ja simuloimaan luonnollista keskustelun kulkua hyödyntäen kielenymmärrys kirjastoa ja Bot Framework -ohjelmistokehyksen toiminnallisuutta. Kasvot tunnistetaan käyttäjän saapuessa päätepisteelle ja analysoidaan tämän jälkeen. Tavoitteiden täyttämisessä onnistuttiin.

Työn kehitysvaiheessa huomattiin back-end ja front-end kerroksien jatkokehitystarpeita. Back-end kerroksen modulaarisuutta voitaisiin lisätä ja toistettua logiikkaa sovelluksessa yhdistää uudelleen käytettäviin komponentteihin. Front-end kerroksen toteutuksessa voitaisiin lisätä turvallisuutta lisäämällä käyttäjien autentikointitaso ja siirtämällä kuvan analysointi back-end kerrokseen. Myös erilaisia haavoittuvuuksia tutkivia ohjelmistokirjastoja voitaisiin käyttää palvelun turvallisuuden varmistamiseen. Yksikkötestaustason lisäämisellä luotaisiin varmuus palvelun toiminnan jatkumisesta halutulla tavalla. Työtä tehdessä etenkin ongelmia tuotti kasvojentunnistus. Kesken kehitysvaiheen huomattiin, että toiminto oli suoritettava täysin selaimessa päätelaitteella, sillä videodatan siirto palvelimelle olisi tuonut turhaa datan siirtoa kerrokselta toiselle. Toisena ongelmana kasvojentunnistuksessa ilmeni kasvojen tunnistamisen nopeus. Palvelu aktivoitui tunnistaessaan myös kameran ohi kävelleistä henkilöitä. Kasvojentunnistus saatiin toteutettua onnistuneesti päätelaitetta käyttäen, ja ratkaisu nopeuteen löydettiin rajoittamalla aktivoituminen kasvojen tunnistukseen yhtäjaksoisella ajalla.

Lopputulos on toimiva ja täyttää suunnitteluvaiheessa asetetut vaatimukset. Palvelun tarjonta esittää teknologioiden toiminnallisuutta, mutta sisällöltään on palvelu hyvin suppea. Ohjelmistokehykset mahdollistavat laajankin palvelun kehityksen, ja esimerkiksi kalenterivarauksien teko palvelun kautta olisi käyttäjää auttava toiminto.

LÄHTEET

Asiakastieto 2020. Enfo Oyj [viitattu 5.10.2020]. Saatavissa: https://www.asiakastieto.fi/yritykset/fi/enfo-oyj/20812129/yleiskuva

Cloudflare 2020. What does server side mean? [viitattu 14.09.2020]. Saatavissa: https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/

ECMA International 2020. Introduction [viitattu 4.10.2020]. Saatavissa: https://www.ecma-international.org/ecma-262/11.0/index.html

Enfo Oyj 2020a. Meistä [viitattu 2.2.2020]. Saatavissa: https://www.enfo.fi/meista

Enfo Oyj 2020b. Kumppanit [viitattu 2.2.2020]. Saatavissa: https://www.enfo.fi/kumppanit

Expert System 2019. Chatbot: What is Chatbot? Why are Chatbots Important? [viitattu 17.3.2019]. Saatavissa: https://www.expertsystem.com/chatbot/

JSON 2020 [viitattu 30.8.2020]. Saatavissa: https://www.json.org/json-en.html

Microsoft 2020. Session class [viitattu 14.09.2020]. Saatavissa: https://docs.microsoft.com/en-us/javascript/api/botbuilder/session?view=botbuilder-ts-3.0

Microsoft Azure 2019a. What is Language Understanding (LUIS)? [viitattu 9.4.2019]. Saatavissa: https://docs.microsoft.com/en-us/azure/cognitive-services/luis/what-is-luis

Microsoft Azure 2019b. What are Azure Cognitive Services? [viitattu 17.3.2019]. Saatavissa: https://docs.microsoft.com/en-us/azure/cognitive-services/welcome

Microsoft Azure 2019c. Use LUIS in a chat bot [viitattu 18.3.2019]. Saatavissa: https://docs.microsoft.com/en-us/azure/cognitive-services/luis/what-is-luis

Microsoft Azure 2019d. About Azure Bot Service [viitattu 17.3.2019]. Saatavissa: https://docs.microsoft.com/en-us/azure/bot-service/bot-service-overview-introduction?view=azure-bot-service-3.0

Microsoft Azure 2019e. What is the Bot Framework SDK? [viitattu 17.3.2019]. Saatavissa: https://docs.microsoft.com/en-us/azure/bot-service/bot-service-overview-introduction?view=azure-bot-service-4.0

Microsoft Azure 2020a. What's new [viitattu 24.7.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/bot-service/what-is-new?view=azure-bot-service-4.0

Microsoft Azure 2020b. Introduction to Azure Blob storage [viitattu 4.11.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction

Mozilla 2020. JavaScript [viitattu 3.10.2020]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript

MSDN Blogi 2016a. What is Microsoft Bot Framework Overview [viitattu 17.3.2019]. Saatavissa: https://blogs.msdn.microsoft.com/uk_faculty_connection/2016/04/05/what-is-microsoft-bot-framework-overview/

MSDN Blogi 2016b. Bot Connector [viitattu 17.3.2019]. Saatavissa: https://docs.microsoft.com/en-us/archive/blogs/uk_faculty_connection/what-is-microsoft-bot-framework-overview#bot-connector

MSDN Blogi 2016c. Simple Dialog Model [viitattu 17.3.2019]. Saatavissa: https://docs.microsoft.com/en-us/archive/blogs/uk_faculty_connection/what-is-microsoft-bot-framework-overview#bot-builder-sdk

NPM 2020, About npm [viitattu 23.8.2020]. Saatavissa: https://docs.npmjs.com/about-npm/

OpenCV 2020. About [viitattu 16.8.2020]. Saatavissa: https://opencv.org/about/

Pluralsight 2020. FRONTEND VS. BACKEND: WHAT'S THE DIFFERENCE? [viitattu 14.09.2020]. Saatavissa: https://www.pluralsight.com/blog/software-development/front-end-vs-back-end