Karelia-ammattikorkeakoulu
Tradenomi (AMK), tietojenkäsittely

Twilio-alustan hyödyntäminen pikatulkkauksen tilaamisen automatisoinnissa

Mira Järnfors

Opinnäytetyö, Joulukuu 2024

www.karelia.fi

Karelia AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Joulukuu 2024
Tietojenkäsittelyn koulutus

Tikkarinne 9
80200 JOENSUU
+358 13 260 600

Tekijä(t)
Mira Järnfors

Nimeke
Twilio-alustan hyödyntäminen pikatulkkauksen automatisoinnissa

Toimeksiantaja
Youpret oy

Tiivistelmä

Tässä toiminnallisessa opinnäytetyössä on tarkoituksena luoda toimeksiantotyönä tulkkauspalveluita tarjoavalle yritykselle palvelu. Palvelun tarkoituksena on tarjota yrityksen jo olemassa oleville asiakkaille mahdollisuus tilata pikatulkki automaattisesti soittamalla tiettyyn puhelinnumeroon; tämän automatisaation avulla asiakkaan pitäisi pystyä soiton jälkeen yhdistämään puhelu suoraan vapaana olevaan tulkkiin.

Työ toteutettiin hyödyntämällä enimmäkseen Twilio-ohjelmaa; Twilio tarjoaa tekstiviesti- ja puhelupalveluiden automatisointia varten oman kehitysympäristönsä sekä koodin kirjoittamista, että vuokaavioiden luomista varten. Lisäksi työssä hyödynnettiin toimeksiantoyrityksen omia työkaluja, jotka koostuivat muun muassa backendista, servlet tiedostoista, erilaisista testaustyökaluista ja yrityksen omasta hallintanäkymästä.

Työn tuloksena saatiin yritykselle luotua palvelu, joka täyttää yrityksen alussa asettamat tavoitteet palvelulle; käyttäjä pystyy soittamaan ja hänet yhdistetään suoraan tulkkiin, mikäli käyttäjä todetaan alussa yrityksen asiakkaaksi ja kyseiselle kieliparille löytyy vapaana oleva tulkki.

Kieli
suomi

Sivuja 36
Liitteet 2
Liitesivumäärä 2

Asiasanat
automaatio, Twilio, rajapinta, pikatulkkaus

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
January 2024
Degree Programme in Business Information Technology

Tikkarinne 9
80200 JOENSUU
FINLAND
+ 358 13 260 600

Author (s)
Mira Järnfors

Title
The Automatisation of Ordering an Instant Interpretation through Twilio

Commissioned by
Youpret oy

Abstract

The objective of this thesis is to create a service for a company that provides interpretation services as a commissioned project. The purpose of the service is to offer the company's existing clients the ability to automatically request an instant interpretation. This is done by calling a specific phone number; through this automation, clients should be able to connect directly to an available interpreter after the call.

The thesis was created by utilising largely the services offered by Twilio, which is a platform that provides a development environment for the processes of automating SMS and phone services. These environments include a platform for coding as well as creating flowcharts. In addition to these the company's own tools, such as their backend systems, servlet files, multitude of testing tools, as well as the company's own administrative management interface.

As a result of the aforementioned work, a service was successfully created that meets the company's objectives and needs. Users can make a call and be connected to an interpreter directly, provided they are firstly identified as existing customers of the company, and a suitable interpreter is available.

Language	Pages 36 Appendices 2 Pages of Appendices 2
Finnish	

Keywords
automatisation, Twilio, application programming interface, instant interpretation

Sisältö

Kuviot

1 Johdanto

1.1 Opinnäytetyön sisältö ja laajuus

Tämän toiminnallisen opinnäytetyön tavoitteena on tuottaa pikatulkkauksen tilaamisen automatisoiva palvelu Twilio-alustan avulla, mikä suoritetaan toimeksiantona tulkkauspalveluita tarjoavalle yritykselle Youpret oy. Opinnäytetyö on jäsennelty lukuihin; ensimmäisenä niistä on johdanto, jossa esitellään aihe lukijalle, sekä pyritän perustelemaan aihevalintaa, kertomaan työn lähtökohdista, aiheen rajauksesta ja pintapuolisesti työn tavoitteesta. Toinen luku kattaa työn kirjallisuuskatsauksen, eli tarkemmin sen keskeisimpiä käsitteitä ja niiden relevanttiutta työn kannalta. Kolmannessa luvussa määritellään tarkasti opinnäytetyön tehtävät ja avataan enemmän aiemmin mainittuja tavoitteita, sekä työn aikana varauduttaviin ja kohdattaviin riskeihin ja ongelmiin. Seuraavassa luvussa kuvataan opinnäytetyön aikana sovellettavia menetelmiä sekä työskentelytapoja, ja viidennessä luvussa käsitellään huomioon otettavaa luotettavuutta ja eettisyyttä. Seuraavassa luvussa avataan opinnäytetyön aikataulua sekä sen haasteita sekä opinnäytetyöhön liittyvää rahoitusta ja rahallista korvausta. Seitsemännessä luvussa käsitellään varsinaista toiminnallista toteutusta ensiksi vuokaavion osalta, sitten Twilion Function-kehitysympäristön puolen tuotoksia ja viimeisenä Eclipsessä backend puolelle luotuja metodeja. Viimeinen luku kattaa opinnäytetyön yhteenvedon sekä pohdintaa liittyen sen hyödyllisyyteen ja mahdollisiin jatko- ja kehitysmahdollisuuksiin tulevaisuudessa. Viimeisenä työstä löytyvät työssä käytetyt lähteet.

1.2 Aihevalinta ja sen rajaus

Lopullinen aihevalinta muotoutui toimeksiannon perusteella, kuten yllä mainittiin; yrityksen tarve pienentää asiakaspalvelun kuormaa toistuvilla sekä samankaltaisilla pyynnöillä loi tarpeen prosessin automatisoinnille, mikä on tämän opinnäytetyön tavoite. Koska Youpret tarjoaa pikatulkkauksia sadoilla kielillä, ei kaikkien kielten automatisointi ole järkevää eikä loppujen lopuksi edes mahdollista – koska lopullinen palveluprosessi on tarkoitus suorittaa

puhelimella, rajaa se myös ylipäätään valittavien kielien määrää, sillä ei olisi kovin kattavaa sisällyttää kymmenkuntaa kieltä mahdollisiksi valinnoiksi. Siispä tavoiteltava lopputyö on rajattu toimeksiantajan pyynnöstä yhteen pyydetyimmistä kielistä eli venäjän kieleen. Opinnäytetyön jälkeen on mahdollista laajentaa tätä koskemaan kieliä, joille on yrityksen tietojen mukaan suuri kysyntä.

Opinnäytetyön tavoitteena on tarjota yritykselle sekä sen asiakkaille valmis palvelu, johon asiakas voi soittaa, numeronäppäinten avulla valita pikatulkkauksen venäjän kielellä ja saada mahdollinen vapaana oleva tulkki käyttöönsä. Automatisoinnin tavoitteena on myös vähentää asiakaspalvelun kuormitusta, jotta toistuvien ja samankaltaisten pyyntöjen määrä vähenee.

1.3 Lähtökohdat opinnäytetyölle

Opinnäytetyötä ei aloiteta täysin olemattomista lähtökohdista, sillä yrityksessä oli aikaisemmin yritetty implementoida samanlaista palvelua – mikä ei erinäisistä syistä lopulta toteutunut – joten esimerkiksi vuokaaviota varten pystyi ottamaan huomioon edellisellä kerralla olevat relevantit tarpeet, sekä mitä niitä varten pitäisi lisätä. Lisäksi palvelua luodessa tarvittavat Java-luokat, sekä niitä varten luodut metodit ja muuttujat ovat valmiina yrityksellä. Kuitenkin esimerkiksi funktiota, jonka avulla varsinainen backend-puolelle otettava yhteys, asiakkaan tunnistaminen puhelinnumeron avulla ja vapaan tulkin löytäminen tapahtuvat, ovat tekijän vastuulla luoda.

1.4 Toimeksiantajayritys Youpret

Youpret on tulkkauspalveluita – pääasiassa puhelin- sekä asiantuntijatulkkauksia – tarjoava joensuulainen yritys. Yrityksen päätavoitteena on varmistaa, että asiakkaat saavat mutkatonta tulkkauspalvelua, joka edistää sujuvaa kommunikaatiota eri kielten ja ihmisten välillä. Puhelintulkkaukset varmistavat sen, että ihmiset voiva kommunikoida kustannustehokkaasti ja helposti sekä asiat etenevät sujuvasti. (Youpret 2024a.)

Youpretin henkilöstö koostuu monipuolisista ammattilaisista eri aloilta; yrityksen pääosassa ovat sekä in-house-, että freelance-tulkit, mutta varsinainen henkilöstö koostuu tulkkaus- ja kielialan sekä rekrytoinnin, ja opetusalan ammattilaisista, finanssi- ja laskutuksen asiantuntijoista, ohjelmistokehittäjistä sekä markkinoinnin- ja asiakaspalvelun tiimeistä. (Youpret 2024b.)

1.5 Twilio

Jo otsikossa mainittu Twilio on erilaisia automatisointipalveluita tarjoava yritys – esimerkkinä toiminnastaan he tarjoavat muun muassa automaattisen tekstiviestimuistutuksen saamisen tai videopuhelun soittaminen. Yritys tarjoaa näitä palveluita varten oman kehitysympäristönsä, jossa voi muun muassa luoda erilaisia palveluita varten tehtäviä funktioita, joiden avulla palvelu voi ottaa yhteyden tarvittavalle backend-palvelimelle sekä luoda erilaisia vuokaavioita palvelun prosessin etenemisestä Twilion tarjoamassa Studio ympäristössä. (Twilio 2024a.)

2 Opinnäytetyön tietoperusta

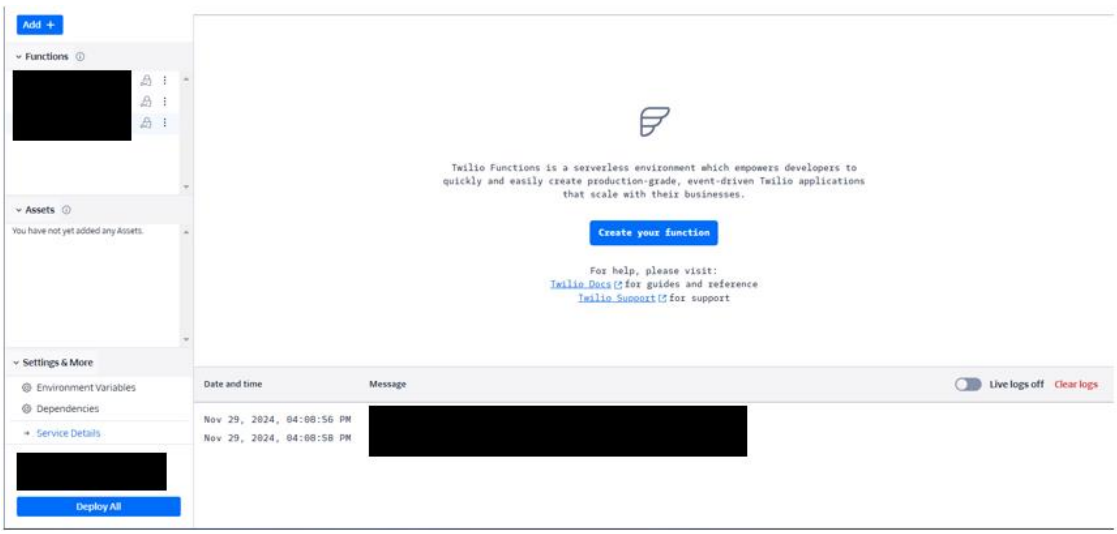
2.1 Koodauskieli ja ohjelmointiympäristö

Palvelun backend puolta suunniteltaessa käytetään pääasiassa kahta kieltä: Javaa sekä JavaScriptiä. Käsitteellä backend viitataan siihen osaan kehitysprosessista, joka tapahtuu ns. piilossa käyttäjältä: eli kaikenlainen tietojen käsittely, tietokantojen hallinnoiminen ja muu vastaavanlainen toiminnallisuus viittaavat kaikki backend-käsitteeseen (Merriam-Webster 2024). Tätä avataan lisää alaluvussa 2.2.1. Ensimmäisellä näistä kielistä ohjelmoidaan pääsääntöisesti pelkästään yrityksen käyttämässä Eclipse-ohjelmointiympäristössä tarpeellisten metodien luomiseen, kun taas jälkimmäistä käytetään Twilion omassa kehitysympäristössä palvelun kehittämiseen.

Ensimmäinen opinnäytetyön toiminnallista puolta luodessa oleva käytettävä ohjelmointiympäristö on Eclipse. Tällä on tarkoitus suorittaa kaikki sellaiset toiminnallisuudet, kuten käyttäjän todentaminen heidän soittavan puhelinnumeron avulla, käyttäjän yksilöllisen tunnuksen hakeminen, tulkkauksen luominen ja lopulta kieliparille sopivan tulkin etsiminen.

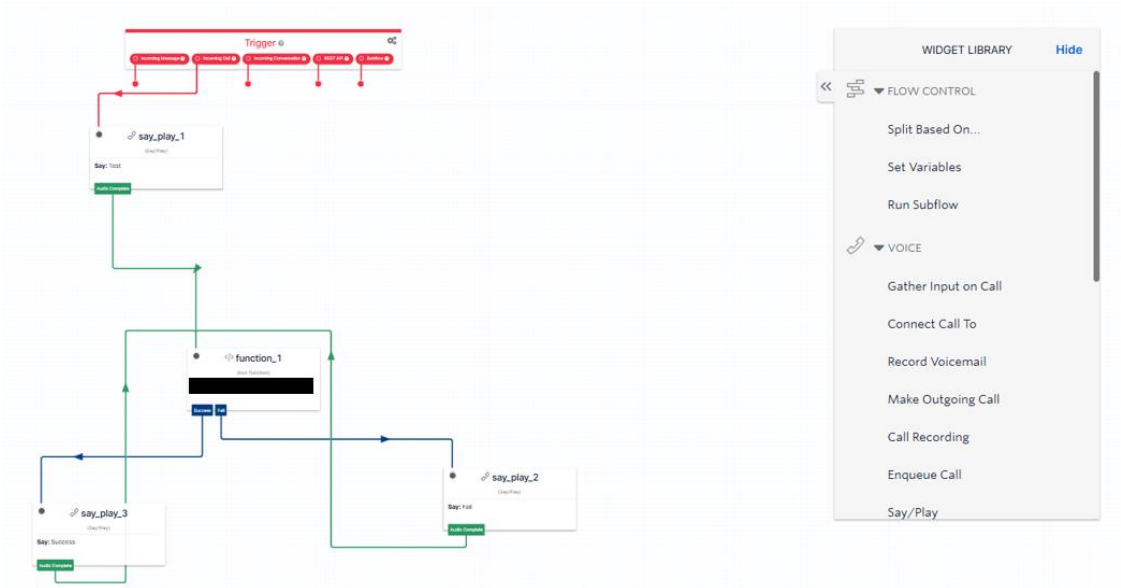
Palvelu itsessään toimii täysin puhelimen välityksellä, eli käyttäjä tarvitsee vain mobiililaitteen käyttääkseen palvelua. Samalla tavalla tulkki tarvitsee mobiililaitteen, mutta toisin kuin käyttäjällä, tulkkien laitteille on asetettu toimeksiantajayrityksen puolesta rajoite, että laitteen pitää olla älypuhelin.

Ohjelmaa luodessa ympäristönä käytetään pääsääntöisesti Twilion omaa kehitysympäristöä. Valinta perustuu sen tarjoamaan helppouteen – on aikaa säästävämpää kehittää funktiota Twilion omassa ympäristössä, joka on räätälöity funktioiden luomiseen, joita käytetään Twilion vuokaavioiden kanssa, eikä siirtyä erilliselle alustalle. Lisäksi Twilio Functions-ominaisuus tekee web-infrastruktuurin skaalauksesta yritykselle helpompaa, sillä se eliminoi tarpeen oman palvelimen ylläpitoon erikseen Twiliota varten. Näihin ominaisuuksiin luetaan muun muassa jo mainitut funktioiden luomiset, sekä ympäristömuuttujien ja muiden resurssien asettamiset, sekä tarvittavan debuggauksen suorituksen. Ohjelmointia varten ympäristöstä ei puutu mitään ja se toimii kaikin puolin samalla tavalla kuin tavallinen ohjelmointiympäristö. (Twilio 2024b.) Alla olevassa kuvassa on havainnoitu Twilion kehitysympäristöä.



Kuva 1 Twilion Functions-kehitysympäristö.

Yhtenä Twilion tarjoamista resursseista on yllä mainittua automatisaation suunnittelua ja toteutusta varten luotu alusta vuokaavioiden kehitystä varten; näitä voidaan rakentaa Twilion kehittelemien pienoistoimintojen, tunnetummin englanninkielisellä termillään widgettien, avulla. (Twilio 2024c.) Toimintojen avulla kokonainen puhelinsoittoprosessi voidaan automatisoida, jonka toimintaa on havainnoitu alla olevalla kaaviolla.



Kuva 2 Twilio Studion toiminta kuvattuna.

2.2 Keskeisimmät käsitteet

2.2.1 Backend

Kuten jo aikaisemmin käsitettä backend hieman avattiin, niin termi toimii yläkäsitteenä prosesseille, sekä konsepteille, jotka toimivat käyttäjältä piilossa (Merriam-Webster 2024). Backendin toiminnan kulmakiviin lasketaan erilaiset käytettävät teknologiat, joita avataan seuraavassa kappaleessa, sekä backendin kehittämiseen tarkoitetut kielet ja tietokantojen toiminta ja ohjelmointirajapintojen kanssa yhteistyön tekeminen. Tässä luvussa käsitellään kahta ensimmäistä konseptia, kun taas ohjelmointirajapintoja avataan enemmän alaluvussa 2.2.2.

Kun puhutaan backendista konseptina, on mainittava ne kehityspilarit, eli teknologiat, ja kielet, joista backendin rakentaminen koostuu; jälkimmäisestä on mainittu ylempänä jo Java-kieli, mutta muihin useimmiten käytettyihin backendin ohjelmointikieliin luetaan sen lisäksi muun muassa Python, C#, JavaScript pohjainen Node.js, PHP, ja Ruby. Teknologioita backendin kanssa on useampia, mutta tärkeimpinä mainittakoon frameworkit eli viitekehykset, tietokannat ja web serverit eli verkkopalvelimet. (Built In 2023.)

Viitekehykset tuovat keskeistä apua backendin luomisessa, kehittämisessä ja ylläpidossa, sillä erilaiset viitekehykset – jotka useimmiten ovat kielikohtaisia kuten Spring Java-kielessä tai Django Python-kielessä – tarjoavat kehittäjille mahdollisuuksia abstrahoimalla monia pienien prioriteettien yksityiskohtia, kuten esimerkiksi Django-viitekehyksessä pyyntöjen käsittelyä. Ne voivat tarjota myös suurempaakin apua, kuten erilaisia kirjastoja, todentamista ja auktorisointiratkaisuja sekä muita turvallisuusratkaisuja.

Tietokannat ovat olennainen osa mitä tahansa kehitysprojektia, sillä ilman tietokantoja suurien datamäärien tallennuksesta, hakemisesta ja käsittelystä tule hyvinkin hankalaa, ellei jopa mahdotonta. Näin ollen projektin backendia kehittäessä sen yhdistäminen tietokantaan on keskeistä sen toiminnan kannalta.

Viimeisenä tärkeänä komponenttina backendin kehityksessä mainittakoon **verkkopalvelimet**, joiden tehtävänä on vastaanottaa clientiltä – eli laiteelta, josta pyyntö lähtee esimerkiksi käyttäjän toimesta – saatuja pyyntöjä, käsitellä ne ja vastata clientille (pyynnöistä ja vastauksista lisää seuraavassa alaluvussa). Kaikki nämä käsitteet yhdessä auttavat luomaan suoraviivaisesti, sulavasti ja tehokkaasti toimivan backendin käyttäjän eduksi. (Built In 2023.)

Kun käsitteellä backend viitataan käyttäjältä piilossa oleviin prosesseihin, niin käyttäjän näkyvissä olevat prosessit ja siihen liittyvät konseptit kulkevat ylätermin frontend alla. Tässä ei frontend-käsitteen konseptia avata samalla tavalla kuin backend-käsitettä, mutta niiden yhteistyö on mainittava puhuttaessa backendistä, sillä ne kulkevat käsi kädessä kaikissa kehityksen vaiheissa.

2.2.2 Ohjelmointirajapinta

Ohjelmointirajapinta – johon viitataan usein myös lyhenteellä API (eng. application programming interface), joka tulee sen englanninkielisestä termistä – on kaikessa yksinkertaisuudessaan työkalu, jota tarjotaan niin kehittäjille kuin muillekin harrastajille ja maalikoille applikaation kanssa toimimiseen.

Rajapinnat toimivat lähettämällä pyynnön yhdestä laitteesta toiselle. Prosessi voidaan myös automatisoida siten, että sen sijaan, että pyyntö lähtee käyttäjän klikattua esimerkiksi hakunappulaa, tämä tapahtuu automaattisesti: voi jokapäiväinen sääapplikaatio toimia esimerkkinä automatisoidusta rajapintapyynnöstä – esimerkiksi näyttämällä 10 päivän säätilan applikaatio voi lähettää automaattisen pyynnön rajapinnalle hakea 10. päivän sään keskiyöllä. Näin voidaan esimerkiksi rajapintapyyntöprosessi automatisoida, mutta se ei tietenkään toimi lähes kaikissa tapauksissa. Näin siis joko käyttäjän pyynnöstä tai automaatiolla laite lähettää pyynnön esimerkiksi datan hakemisesta, jolloin rajapinta toimii tiedon hakijana serveriltä. Laite voi myös lähettää pyynnön datan julkaisemisesta, päivityksestä tai poistosta. Tämän jälkeen serveri lähettää statuskoodin, vastausotsikon ja -tekstin alkuperäiselle pyynnön tehneelle laitteelle. (Postman 2024.)

Rajapinnat voivat olla kolmea erilaista tyyppiä; yksityisiä-, julkisia ja partnerirajapintoja. Yksityiset ja julkiset rajapinnat ovat melko yksiselitteisiä; yksityiset, kuten esimerkiksi tässä toimeksiannossa sovellettava rajapinta on organisaation sisäinen, mikä sisältää toimeksiannon kannalta tarpeellisia tietoja, kuten työntekijöiden -ja kielten saatavuuksia. Julkiset rajapinnat, kuten esimerkiksi Forecan tarjoama corporate.foreca.com rajapinta sisältää julkista tietoa muun muassa säästä ja esimerkiksi kehittäjät voivat hyödyntää rajapintaa integroimalla sen omille nettisivuilleen näyttämään käyttäjälle säätä (Foreca 2024). Partnerirajapinnat ovat tavallaan sekoitus näitä molempia tarjoamalla alustansa vain niiden kanssa toimiville tietyille asiakkaille. Yksi esimerkki partnerirajapinnasta on Shutterstock, joka toimii useiden suurten yritysten kanssa tarjoamalla heille pääsyn omaan mediakatalogiinsa. (Postman 2024.)

Pystyäkseen tekemään pyynnön serverille, rajapinta tarvitsee viisi elementtiä: pyynnön **metodi**, sen **parametrit**, **päätepiste** sekä **pyyntöotsikko** ja – **runko**. **Metodi** on kaikessa yksinkertaisuudessaan toiminta, joka halutaan suorittaa: niitä ovat tiedon hakeminen eli GET, tiedon lähetystä tai julkaisua varten oleva POST, jo olemassa olevan tiedon päivitystä varten oleva PUT ja tiedon poistamista varten tarvittava DELETE. **Parametrit** viittaavat niihin tarvittaviin tietoihin, joita pyyntöä tehtäessä lähetetään. Tiedot voivat olla mitä vain, kuten esimerkiksi haettavan käyttäjän id-tieto. Parametrit voidaan lähettää joko osana URL-osoitetta, tai pyyntötekstin mukana. **Päätepiste** (eng. endpoint), tunnetummin URL-osoite, on se osoite, johon tiedon lähetys suunnataan – se ei kuitenkaan ole sama kuin käyttäjän käyttämä sivusto, vaan erillinen tietojenkäsittelyä varten luotu sivusto. Esimerkiksi jos käyttäjä käyttää Foreca.fi, niin tietoa käsitellään Forecan omalla API-sivustolla. **Pyyntöotsikko** ja -**runko** (eng. request header ja request body) voidaan ajatella samanlaisena kokonaisuutena kuin kirjekuori; kirjekuoren ulkopuolelle kirjoitettavat tiedot vastaavat otsikossa olevia tietoja, kuten sisältötiedon (eng. content-type) tyypin ja auktorisointi varmistaa, että käyttäjä on valtuutettu suorittamaan toiminnallisuutta. Kirjekuoren sisältö taas vastaa vastausrunkoa – runko käsittää sen tiedon, joka pyynnön mukana lähetetään, esimerkiksi tietoa luodessa se voi olla käyttäjän koko nimi, sekä uuden käyttäjäprofiilin sähköposti ja salasana. (Postman 2024.)

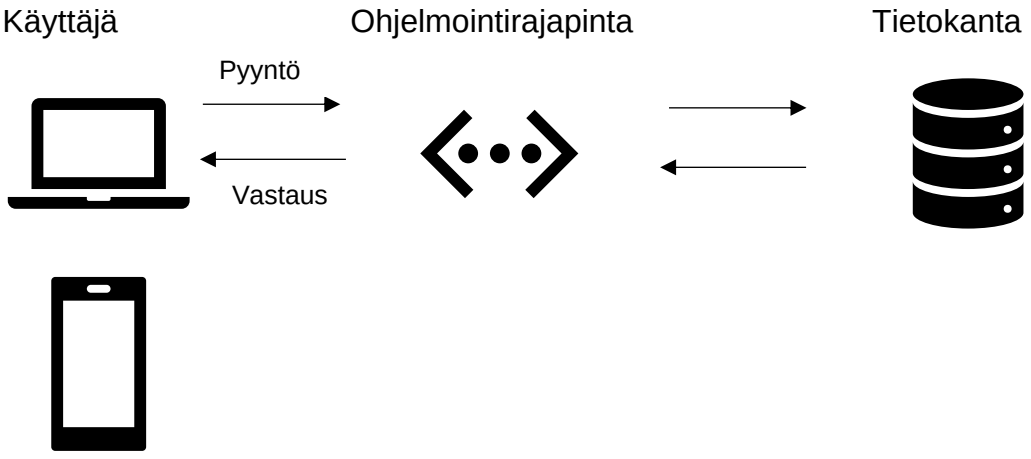
Serveri tarvitsee vastaamiseen verrattuna pyyntöön vain kolme asiaa: **statuskoodin**, sekä **vastausotsikon** ja – **rungon**. **Vastausotsikko** ja –**runko** (eng. response header ja response body) ovat hyvin samanlaisia, kuin yllä mainitut pyyntövastikkeensa. Vastausrunko saattaa pyynnöstä riippuen sisältää haetun tiedon tai ilmoituksen siitä, että uuden tiedon luominen tai poistaminen, tai tiedon päivitys onnistui. Vastausotsikko sisältää pyyntöotsikon lailla sisällön tyypin, sekä lisätietoina myös serverin vastauksesta riippuvan statuskoodin, vastauksen koon (eng. content-length) ja mahdolliset evästetiedot.

Erilaisia **statuskoodeja** on lukuisia ja Internet Assigned Numbers Authority (lyh. IANA) organisaatio omaa kattavan listan erilaisista vastauskoodeista. Näistä numerolla yksi (1) alkavat ovat informatiivisia statuskoodeja, eli pyyntö on vastaanotettu ja prosessi jatkuu, numerolla kaksi (2) alkavat ovat onnistuneille

pyynnöille, ja numerolla kolme (3) alkavat ovat uudelleenohjaukseen liittyviä statuskoodeja – eli jotta pyyntö voidaan onnistuneesti suorittaa loppuun, on tehtävä vielä jotakin. Luvuilla neljä (4) ja viisi (5) alkavat statuskoodit viittaavat joillain tavoilla virheisiin – ensimmäisessä ne ovat clientin eli pyynnön lähettävän laitteen puolesta tapahtuvia virheitä (joko pyyntöä ei voida suorittaa tai se sisältää syntaksivirheitä), kun taas jälkimmäisessä ne ovat serverin puolella tapahtuvia virheitä. (IANA 2022.)

Seuraavaksi on listattu muutama tarkempi esimerkki, joita todennäköisemmin kehitysprosessin aikana kohdataan. Tämän tarkoituksena on havainnoida lukijalle, että millaisia yksityiskohtaisempia statuskoodeja on olemassa: epäonnistuneista datan lähetys- tai vastaanottostatuskoodeista yleisimpiä ovat 400 Bad Request (epäonnistunut pyyntö käyttäjän puolesta, esimerkiksi jos kieltä valitessa puhelun aikana käyttäjä painaa sellaista numeroa, jolle ei ole vastaavaa kieltä), 403 Unauthorised, eli käyttäjällä ei ole tarvittavia autentikointimetodeita tarjottavana (kuten esimerkiksi käyttäjä ei olisi toimeksiantoyrityksen asiakas, jolla on oikeudet tilata pikatulkkaus puhelun kautta) ja 500 Internal Server Error (joka saattaisi johtua esimerkiksi toimeksiantoyrityksen serverin toimimattomuudesta). (IANA 2022.)

Alla olevassa kuvassa on havainnoitu koko ohjelmointirajapinnan pyynnön lähetyksen ja vastaanottamisen prosessia.



Kuvio 1 Ohjelmointirajapinnan toiminta kuvattuna

2.2.3 Servlet

Yksi keskeisimmistä, opinnäytetyössä hyödynnettävistä käsitteistä tulee yrityksen oman backend koodin tiedostoista, nimeltä servlet; niiden tarkoitus on toimia lisäkätenä laajentamaan yleensä – mutta ei aina – web-pohjaisten sovelluksien toimintoja pyyntö-vastausmenetelmällä, jonka konseptia avataan myöhemmin tässä luvussa. Näihin toimintoihin voidaan laskea esimerkiksi clientiltä tehtyjä HTTP-pyyntöjä. Servlettien etuna onkin juuri niiden helppo mukauttaminen eri järjestelmiin web-pohjaisissa sovelluksissa, sillä ne toimivat vain serverin domainissa, eivätkä vaadi verkkoselaimen tukevan Javaa. Lisäksi ne ovat helposti skaalattavissa ja tehokkaita, koska servlettejä pystytään käsittelemään joko yhdessä tai useammassa säikeessä – eli pienemmässä suorituksessa yhden prosessin sisällä – hajautetusti eri backend servereillä. (Crawford & Hunter 2001.)

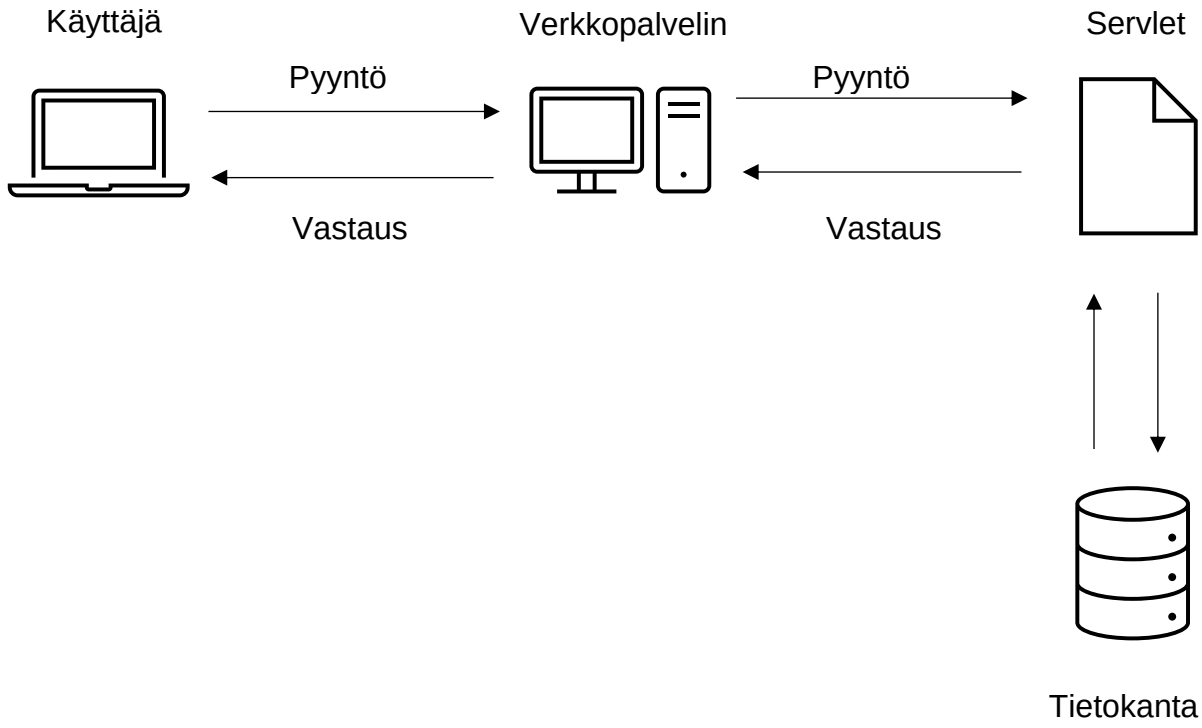
Seuraavissa kappaleissa kerrotaan tarkemmin servlettien elämänkaaresta. Saatuaan pyynnön servletin elämäkaari alkaa; se on kolmivaiheinen prosessi, jota hallinnoi sen aloitus, pyyntöjen käsittely – joskin on mahdollista, ettei servlet käsittele yhtäkään pyyntöä elämänkaarensa aikana – ja tuho (Crawford & Hunter 2001).

Aloitusprosessin käynnistämisen kutsulla ei ole niinkään merkitystä, sillä se voidaan aloittaa esimerkiksi käynnistäessä serveriä tai kun servlettiä ensimmäisen kerran kutsutaan, mutta tavasta riippumatta aloitusprosessia kutsutaan vain kerran servletin elämänkaaren aikana. Aloitusprosessin eli init()-metodin aikana servletille voidaan syöttää tiettyjä objekteja, joita servlet käyttää seuraavassa vaiheessa eli pyyntöjen käsittelyssä. Nämä eivät kuitenkaan ole suorananisesti vielä pyyntöjä, vaan ennemminkin parametreja tai oletusarvoja. (Crawford & Hunter 2001.) init()-metodi saa parametrikseen config, joka sisältää yllä mainitut konfiguraatio- ja alustustiedot.

Kun aloitusprosessi on saatu onnistuneesti suoritettua, aloittaa servlet pyyntöjen käsittelyn. Kuten yllä mainittiin, niin servlet ei välttämättä käsittele elämänkaarensa aikana yhtäkään pyyntöä, mutta oletusarvo on, että sille syötetään pyyntöjä. Pyyntöjen käsittelyyn servletit käyttävät service()-metodia,

joka kutsutaan sen saatua clientiltä pyyntö, jonka se sitten voi delegoida vielä eteenpäin sen mukaan, mitä pyyntöjä käsitellään; esimerkiksi tiedonhaku pyyntöjä käsitellään doGet()-metodeilla, kun taas doPost()-metodit käsittelevät tietojen luomista. Parametreinaan nämä metodit saavat req- ja res-parametrit, eli clientin pyynnön ja servlet vastauksen sisältävät objektit. Pyynnön tiedot sisältävätkin juuri tämän pyynnön tyypin tiedot, jonka perusteella delegointi tapahtuu. Vastaus taasen on usein JSON, HTML, tai XML-muodossa olevassa objektissa palautettavaa tietoa. Tätä kutsutaan yllä mainituksi pyyntö-vastausmalliksi (eng. request-response model). (Oracle 2024.)
Tuho eli destroy()-metodi, samoin kuten init()-metodi, kutsutaan vain kerran servletin elämänkaaren aikana – aivan sen lopussa. destroy()-metodin kutsuminen antaa servletille tilaisuuden siivota pois servletin käytössä olevat resurssit, eli tuhota käynnissä olevat säikeet, ja vapauttaa käytössä ollut muisti. Servletin kutsuttua tätä metodia, se ei enää pysty käsittelemään pyyntöjä ja lopettaa metodin kutsun seurauksena toimintansa. (Oracle 2024.)

Alla olevassa kuviossa on kuvattu servlettien toimintaprosessia.



Kuvio 2 Servletin toiminta kuvattuna

2.2.4 Pikatulkkaus

Kolmas monesti opinnäytetyössä mainittu käsite, jota ei usein kohtaa tulkkausalan ulkopuolella *pikatulkkaus.* Kyseiselle termille on hankala löytää tarkkaa määritelmää ottaen huomioon, kuinka usein kyseinen termi on päivittäisessä käytössä tulkkausalalla. Pro gradu -tutkielmassaan Holopainen määrittelee pikatulkkauksen olevan aikaan tai paikkaan sitomaton tulkkauksen muoto; yksinkertaisuudessaan tulkkausta tarvitseva henkilö ottaa palveluntarjoajaan yhteyttä tarvitsemalleen kieliparille ja tarjouspyyntö on voimassa tietyn ajan, ja hyväksyttyään tarjouksen tulkkaus alkaa välittömästi. (Holopainen 2022, 6–7.)

2.3 Aiempi tutkimus

Koska opinnäytetyö tehdään toiminnallisena eikä sen lähtökohtina tai perustana ole muita tutkimuksia, ei tässä oteta aikaisempia tutkimuksia huomioon. Ottaen myös huomioon sen, että työ tehdään toimeksiantona yritykselle projektityönä, ei yrityksellekään ole alussa mainittua keskeneräistä vuokaaviota lukuun ottamatta omaa materiaalia, johon verrata työtä.

3 Opinnäytetyön tavoite ja tutkimustehtävä

3.1 Tehtävät ja tavoite

Opinnäytetyön tavoitteena on saada toimiva palvelu yrityksen käyttöön. Toiminnallisiin tehtäviin kuuluvat muun muassa käyttäjille räätälöidyn prosessin luominen soittaessa asiakaspalveluun ja sen automatisointi Twilion vuokaavion avulla. Tähän prosessiin sisältyy muun muassa prosessin kielen valinta, sen pikatulkkauksen tilaamisen vahvistaminen ja tulkkauskielen valinta kaikki numeronäppäimien avulla. Tämän jälkeen järjestelmä lähettää backend-puolelle pyynnön selvittää, onko kyseisen kielen tulkkia saatavilla pikatulkkausta varten. Kielteisen vastauksen sattuessa prosessi ilmoittaa, ettei tulkkeja ole valitettavasti saatavilla ja kysyy, että haluaako asiakas yhdistää

asiakaspalveluun. Mikäli vastaus on myöntävä, lähetetään tulkille pyyntö tulkkaustarjouksesta ja jos tulkki kieltäytyy, seuraa prosessia sama vaihe, kuin jos saatavilla tulkkia ei olisi löytynyt. Mikäli tulkki löytyy ja hän hyväksyy tulkkauksen, yhdistetään asiakas tulkin puhelinnumeroon ja tulkkaus voi alkaa normaaliin tapaan.

3.2 Ongelmat

Kaikissa opinnäytetöissä voi tulla vastaan odotettuja ja odottamattomia ongelmia, ja niin tässäkin. Suurin etu näiden ongelmien ehkäisylle on niiden tunnistaminen ja niiden vaikutuksen tunnistaminen työhön etukäteen. Ilmeisin ongelma projektissa on tietenkin motivaation loppuminen ja uupuminen; ottaen huomioon kirjoittajan 40-tuntisen työviikon opintojen lisäksi se vaikuttaa hyvin vahvasti mahdollisen loppuun palamisen. Tämän ehkäisyksi kirjoittaja sopi toimeksiantajan kanssa mahdollisuudesta kirjoittaa opinnäytetyötä töissä, painolla 70 % töitä ja loput ajasta kirjoittamista.

Toinen keskeisimmistä ongelmista opinnäytetyöhön liittyen on tekijän rajoittunut taitotaso joissakin koodaamiseen liittyvissä aspekteissa; useimmiten ongelmaan törmätessä on mahdollista pyytää apua toimeksiantajan yrityksen teknologiatiimin Senior System Developer -jäseneltä, mutta koska kyseisellä työntekijällä on omat työt, ei aina voida aikatauluttaa avun saamista nopeasti. Hieman liittyen myös edelliseen ongelmaan yksi suuri ongelma on opinnäytetyöneuvojan puuttuminen firman sisällä – ongelmien sattuessa opinnäytetyön tekijällä ei ole mahdollisuutta kääntyä heti jonkun puoleen ja saada apua, vaan saattaa olla, että ongelmien ratkaisemisemisen pitkittyminen hidastaa työn etenemistä.

Viimeisin on aikataulun tiukkuus. Opinnäytetyön toiminnallista osuutta aloitettiin kesän mittaan, mutta koko prosessiin (suunnittelu, toteutus, kirjoitus ja muut byrokraattiset seikat) menevälle ajalle on varattu noin viisi (5) kuukautta.

4 Opinnäytetyön menetelmälliset valinnat

4.1 Menetelmät

Salminen-Tuomaalan, Hautamäen ja Sarvikkaan mukaan toiminnallisessa opinnäytetyössä ei välttämättä sovelleta mitään tiettyä työskentelymenetelmää, mikä työn laadun huomioon ottaminen on järkevin tapa lähestyä opinnäytetyötä. Työn laadun takia aineistoa ei tähän kerätä juuri ollenkaan, joten pakottamalla tietynlainen aineistokeruumenetelmä työlle saattaisi tekemis- ja kirjoittamisprosessi hankaloitua. Kuitenkin se pieni määrä aineistoa, joka työtä varten kerätään – muun muassa autentikointimetodeihin funktioihin tutustuminen tapahtuu Twilion oman dokumentaation kautta, joten parhaimmaksi menetelmätavaksi voisi tässä kutsua laadullista, eli kvalitatiivista menettelytapaa. (Salminen-Tuomaala, Hautamäki, Sarvikas 2023, 643.)

4.2 Työskentelytavat

Opinnäytetyön kehitysprosessi eteni hyvin suoraviivaisesti; ensin toimeksiantaja ilmoitti projektin vaatimukset ja rajaukset, minkä jälkeen luotiin vuokaavio hahmottamaan prosessia. Vuokaavion ensiversion luomisen jälkeen aloitettiin Twilion kehitysympäristössä funktion luominen – ensin yhteys palvelimelle, sitten vapaan tulkin kysely ja yhdistys tulkkiin – ja viimeisenä viimeisteltiin vuokaavio ja testattiin erilaisia virhetilanteita varten.

Opinnäytetyön toimeksiannon luonteen vuoksi työskentelytapana pyrittiin pitämään toimistotyöaikoja viikonpäivinä; tämä sen takia, että projektin toimeksiantaja yritys kehotti tekemään työtä työajalla, mutta myös sen takia, että vapaa-aikaan jää tarpeeksi aikaa, mikä näin ollen myös säästää henkisiä resursseja. Tällä oli myös helpottava vaikutus opinnäytetyön melko tiukkaan aikatauluun, sillä 8-tuntisen työpäivän aikana opinnäytetyötä saatiin helposti edistettyä ja pitäydyttyä aikataulussa. Aikaa projektiin käytetään noin seitsemän (7) kuukautta alusta loppuun; aika-arvio tulee siitä, kun projektin toimeksianto annettiin aina suunniteltuun valmistumispäivään asti.

5 Luotettavuus ja eettisyys

Tämän luvun tarkoituksena on avata opinnäytetyötä kirjoittaessa huomioon otettavia luotettavuuden ja eettisyyden kysymyksiä ja aspekteja, joita kirjoittaessa sekä toiminnallisuutta osuutta toteutettaessa nousi esille. Nämä tehtiin hyödyntämällä apuna laadullisen tutkimuksen luotettavuuden arvioinnin kriteereitä.

Koska työssä ei varsinaisesti käsitelty henkilötietoja, tai millään muullakaan tavalla arkaluontoista tietoa, ei niiden eettisiä kysymyksiä tai luotettavuuden kannalta relevantteja kysymyksiä huomioida tässä työssä. Tietenkin esimerkiksi tilanteessa, jossa verrataan soittavan käyttäjän puhelinnumeroa olemassa oleviin asiakkaisiin, otetaan huomioon näiden tietojen käsittelyssä kaikki mahdolliset varovaisuudet. Lisäksi työssä otetaan huomioon muun muassa lähteiden luotettavuus, ja niiden relevanttius. Eettisyyteen kuuluu myös lähteiden oikein merkitseminen ja sekä kirjoittajien kunnioittaminen. Jotta lukija voi olla varma työn tiedon luotettavuudesta, pyritään työssä minimoimaan tekoälyn käyttö kirjoitusapuna, sillä tekoälyllä on kiisteltävä vaikutus muun muassa salassa pidettävään tietoon ja kokonaisuudessaan oikean tiedon tarjontaan. Lisäksi opinnäytetyössä ja suunnitelmassa tullaan noudattamaan yrityksen salassapitosopimusta.

6 Aikataulu ja rahoitus

6.1 Aikataulu

Aikataulullisesti opinnäytetyö alkoi 2024 kesäkuussa ja jatkui heinäkuussa toimeksiannon saannista. Kesäkuun aikana projektin toimeksiannon aloituksen lisäksi määriteltiin myös parametrit ja rajattiin projektin aihe ja sen tavoite. Lisäksi Twilio-alustaan tutustumien – tarkemmin sen sisäisiin ominaisuuksiin, kuten tarvittavan vuokaavion hahmotukseen ja Twilion sisäänrakennetun kehitysympäristöön tutustuminen tapahtuivat tässä kuussa. Lisäksi toimeksiantaja yrityksen sisäisen opetusasiantuntijan kanssa aiheesta keskustelu tapahtui tässä vaiheessa. Opetusasiantuntijan näkökulma otettiin

mukaan sen takia, että saatiin virtaviivaistettua ja räätälöityä tulkkauksen tilaamisen prosessi firman asiakkaille mahdollisimman yksinkertaiseksi – opetusasiantuntija on myös entinen asiakaspalvelutyöntekijä firman sisällä, joten hänen oletetaan tietävän asiakkaiden tarpeet. Elo- ja syyskuun aikana pyrittiin viimeistelemään toimintaa varten tarvittavien funktioiden toiminta sekä aloittamaan alkeellista testausprosessia, johon kuuluu muun muassa puhelimen kautta yhteyden otto Twilion tarjoamaan puhelinnumeroon ja varmistaminen, että prosessi ottaa yhteyden yrityksen backend puolelle. Lisäksi syyskuussa aloitettiin aloittaa ja saada valmiiksi kirjallisuuskatsaus, opinnäytetyösuunnitelma ja osallistua tarvittaville suunnitteluseminaareille kuuntelijaksi. Lokakuussa osallistuttiin itse esittäjänä ja opponoitiin suunnitteluseminaareissa.

Alustavan aikataulun mukaan marraskuun puoleen väliin mennessä opinnäytetyö olisi kirjoitettu ja valmiina arvioitavaksi, sekä toiminnallinen puoli valmis kaikkien kriteerien mukaisesti.

6.2 Rahoitus

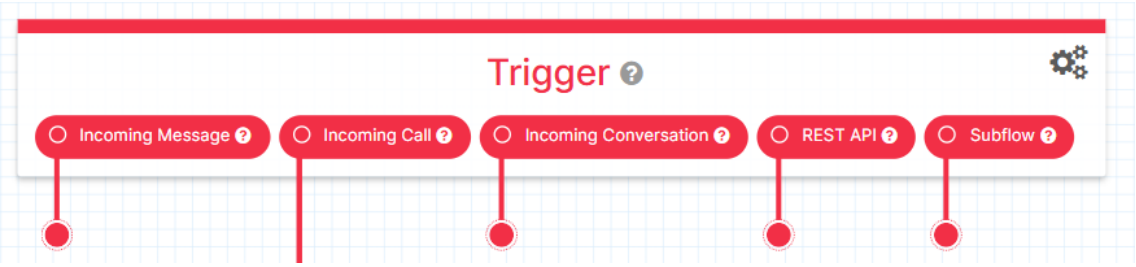
Opinnäytetyössä ei haeta ulkopuolista rahoitusta, mutta koska työ suoritetaan toimeksiantona yritykselle, saa kirjoittaja palkkaa työskentelyajalta. Kuluja opinnäytetyöhön liittyy vain yllä mainitun palkan maksaminen yritykselle, sekä pieni, arvoltaan muutaman euron korvaus Twilio alustalle maksettavasta puhelinnumerosta, joka otetaan käyttöön projektia varten.

7 Toteutus

7.1 Studio -vuokaavio

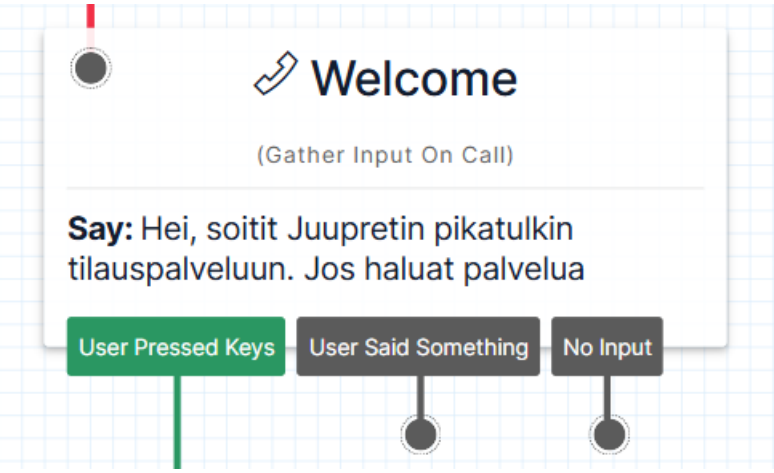
Tässä alaluvussa käsitellään opinnäytetyön vuokaaviota, joka toimii yhdessä Twilion oman kehitysympäristön funktioiden kanssa. Funktioita avataan enemmän alaluvussa 7.2.

1. **Trigger**: Vuokaavion toiminta käynnistetään alla olevan pienoisohjelman avulla – käyttäjä suunnitellessaan vuokaaviota on vapaa valitsemaan vuokaavion aloitustavan omiin tarpeisiinsa sopivaksi – tuleva viesti, keskustelu, API-kutsu tai alussa voidaan myös käynnistää eri alivuokaavio –, mutta tätä opinnäytetyötä varten valittiin tulevan puhelun triggeri. Kyseinen pienoisohjelma on automaattisesti olemassa jokaisessa vuokaaviossa, eikä sen valitseminen vuokaavioon ole valinnaista.



Kuva 3 vuokaavion aloitus

2. **Welcome:** Tämän jälkeen käyttäjä tervehditään ja sanomalla ”Hei, soitit Juupretin pikatulkin tilauspalveluun. Jos haluat palvelua suomeksi, paina 1. Jos haluat yhdistää asiakaspalveluun, paina 2.” Käyttäjän painettua numeronäppäintä siirrytään seuraavaan kohtaan.



Kuva 4 puhelun aloittava "Welcome" widget

3. **Check_If_Existing_Customer:** Tämän jälkeen tarkistetaan, että soittajan puhelinnumero on osa asiakkaiden puhelinnumeroita. Riippuen siitä kumpi se on, tulee lopputulos olemaan erilainen – jos soittaja ei ole asiakas, seuraa vuokaavio Fail-polkua. Mikäli kyseessä on asiakas, seuraa Success-polku ja seuraava vaihe.

Kuva 5 asiakkaan tarkistusta kutsuva funktio "Check_If_Existing_Customer" widget

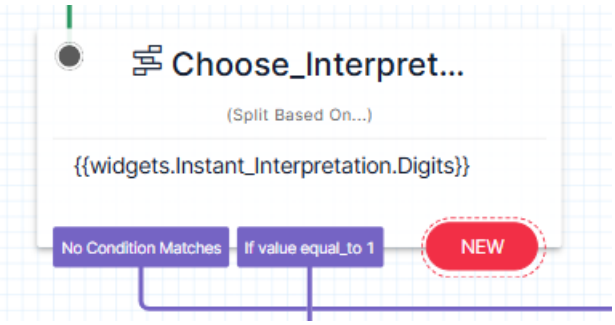
4. **set_customerKeyString**: Jahka asiakas on tunnistettu asiakkaaksi, asetetaan heidän Eclipsen backend puolelta haettu uniikki avainmerkkisarja keyString pienoisohjelman avulla vuokaavioon muistiin, jotta kyseistä merkkisarjaa pystytään köyttämään myöhemmin vuokaaviossa ja funktioissa. Tästä vaiheesta ei tarvitse päästä onnistuneesti läpi, vaan vuokaavio jatkaa automaattisesti toimintaansa ilman ulkoisia laukaisimia.

Kuva 6 asiakkaan keyString-tiedon muuttujaan tallentava "set_customerKeyString" widget

5. **Instant_Interpretation;** Tämän jälkeen asiakas suorittaa kielenvalinnan, joita tällä hetkellä on vain se yksi mahdollinen valinta, eli venäjän kieli. Koko viesti, joka asiakkaalle luetaan, kuuluu ”Jos haluat pikatulkkauksen venäjäksi paina 1 ja jos haluat yhdistää asiakaspalveluun, paina 2.”

Kuva 7 kielivalinnan tekevä "Instant_Interpretation" widget

6. **Choose_Interpretation_Language:** Tämän jälkeen käyttäjän vuokaavio jakautuu jälleen kerran eri suuntiin käyttäjän valinnan perusteella, mutta useamman valinnan puutteen vuoksi pienoisohjelma jakautuu kahteen eri vaihtoehtoon tällä hetkellä. Pienoisohjelma tallentaa edellisessä vaiheessa valitun arvon.

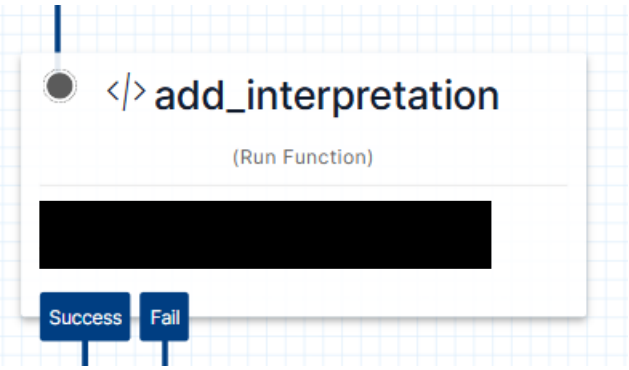


Kuva 8 käyttäjän syötteen tallentava "Choose_Interpretation_Language" widget

7. **Set_Lang:** Alemmassa pienoisohjelmassa asetetaan tarvittavat kielet muuttujiin toLanguage ja fromLanguage, jotka ovat lähdekieli ja kohdekieli, vastaavasti eli suomen ja venäjän kielet. Kuten yllä olevassa vastaavanlaisessa pienoisohjelmassa, tämäkään ei vaadi käyttäjältä erityistä syötettä, jotta prosessia voidaan jatkaa.

Kuva 9 lähde- ja kohdekielet muuttujiin tallentava "Choose_Interpretation_Language" widget

8. **Add_interpretation:** Kun kielet on asetettu ja käyttäjä tunnistettu yrityksen asiakkaaksi, lähetetään nämä tiedot eteenpäin alla olevalle funktiolle. Alla oleva funktio luo yrityksen järjestelmään olemassa olevan tulkkauksen. Jos tuloksena on onnistunut tulkkauksen luominen, annetaan käyttäjän jatkaa seuraavaan vaiheeseen, kun taas epäonnistunut tulkkauksen luominen johtaa prosessin lopetukseen.



Kuva 10 tulkkausta tallentavaa funktiota kutsuva "Add_interpretation" widget

9. **setInterpretationKeyString:** seuraava pienoisohjelma asettaa tulkkaukselle luodun keyString muuttujan talteen, jotta sitä voidaan käyttää tulkkia etsivässä funktiossa, jolle se seuraavaksi välitetään.

Kuva 11 tulkkauksen keyStringin tallentava "setInterpretationKeyString" widget

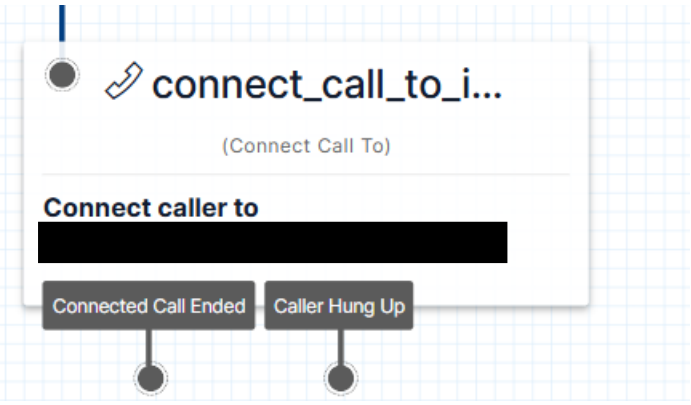
10. **interpretation_check_loop:** tässä kutsutaan viimeisen kerran vuokaaviota varten luotua funktiota, jonka tarkoitus tässä on tarkistaa tietyin väliajoin, onko sopivaa tulkkia löytynyt.

Kuva 12 tulkkistatuksen tarkistava "interpretation_check_loop" widget

11. **set_Interpreter_Phone:** Seuraavaksi, kun tulkkaus on onnistuneesti luotu ja on luonnos-tilassa, tulkille lähetetään heidän applikaatioonsa ilmoitus saatavilla olevasta tulkkauksesta. Jos tulkki hyväksyy tulkkauksen, alla oleva vaihe vuokaaviossa tallentaa kyseisen tulkin puhelinnumeron muuttujaan setInterpreterPhone käytettäväksi myöhemmin vuokaaviossa.

Kuva 13 tulkin puhelinnumeron muuttujaan tallentava "set_Interpreter_Phone" widget

12. **connect_call_to_interpreter:** vuokaavion viimeisessä vaiheessa soittava asiakas yhdistetään suoraan tulkin puhelinnumeroon.



Kuva 14 asiakkaan tulkkiin yhdistävä "connect_call_to_interpreter" widget

7.2 Function -kehitysympäristö

Ylempänä esiteltiin vuokaaviota, jonka kohdissa 3., 8., ja 10. viitataan funktioihin, jotka suorittavat eri pyyntöjä Eclipsen backend-puolelle riippuen niiden tarpeesta. Tässä alaluvussa käsitellään näitä funktioita ja miten ne toimivat vuokaavion kanssa. Seuraavassa alaluvussa 7.3 esitellään backend-puolella olevat koodit, jotka suorittavat varsinaisen käyttäjän asiakasstatuksen tarkistamisen, uuden tulkkauksen luomisen ja sille sopivan tulkin etsimisen.

1. **Check_if_existing_customer**: Ensimmäisenä käsitellään käyttäjän asiakasstatusta tarkistavaa funktiota. Tämä tapahtuu vuokaaviossa numeron 3. pienoisohjelman kohdalla. Funktio alkaa käyttämällä event-nimistä objektia, jonka avulla päästään käsiksi vuokaaviossa lähetettävään arvoon, joka lähetetään otsikolla phone. Kun tieto arvon sisältävästä puhelinnumerosta saadaan, otetaan se talteen ja muokataan vaatimusten mukaan maakoodi pois ja vaihdetaan numero 0 sen tilalle. Tämän jälkeen muokattu arvo lähetetään eteenpäin POST-pyynnöllä tiettyyn API:n osoitteeseen, joka tarkistaa varsinaisen asiakasstatuksen. Mikäli asiakasstatus tunnistettiin onnistuneesti, saadaan se tieto takaisin backend puolelta ja lähetetään tieto vuokaaviolle, että käyttäjä on yrityksen asiakas ja tilausprosessi voidaan jatkaa seuraavaan osioon.

```
const axios = require('axios');

//Suoritettava funktio
exports.handler = async function (event, callback) {

  //Event-objektilla haettu asiakkaan puhelinnumero
  let phoneNumber = event.phone;

  // Muokataan asiakkaan puhelinnumeroa backend-puolen vaatimusten vuoksi
  if (phoneNumber.startsWith('+358')) {
    phoneNumber = phoneNumber.replace('+358', '0');
  }

  //Käytettävä endpoint-osoite. Testausvaiheessa käytetään ngrok-ohjelman avulla luotua automaattista osoitetta
paikallisena ympäristönä. Osoitteen paramterina lähetetään vuokaaviosta otettu soittavan asiakkaan puhelinnumero.
  const API_URL = [redacted]
                                {phoneNumber}`;


  //Asiakkaan tarkistamista varten suoritetaan POST HTTP-pyyntö yllä olevaan osoitteeseen. Mikäli käyttäjä on
tunnistettava asiakas, niin funktiosta saatu tulos otetaan talteen, muussa tappauksessa palautetaan virheilmoitus,
jolloin vuokaaio toimii Fail-polun mukaan
  const post = new Promise((resolve, reject) => {
    axios.post(API_URL).then((res) => {
      resolve(res.data)
    }).catch((err)=>{
      console.log(err);
    })
  })

  //Onnistuneen funktion suorituksen jälkeen tulos lähetetän takaisin vuokaaviolle.
  post.then((result) => {
    callback(null, result);
  });

};
```

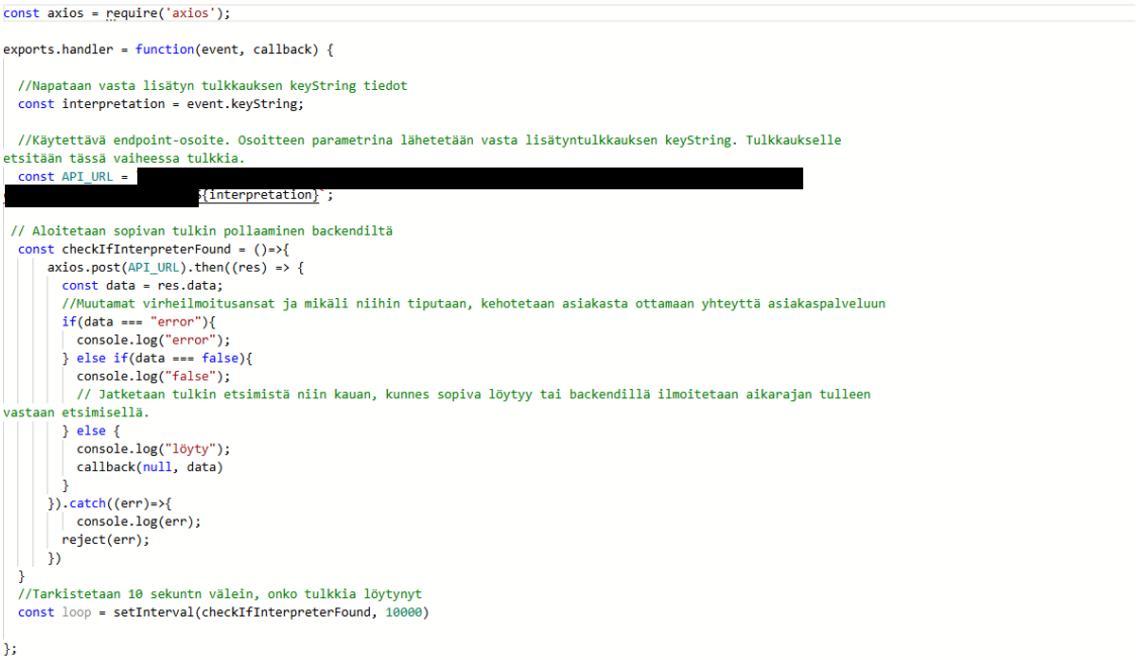
Kuva 15 käyttäjän asiakasstatuksen tarkistuksen pyynnön lähettävä "Check_if_existing_customer" funktio

2. **Find_customer_and_add_interpretation**: vuokaavion seuraava funktio tapahtuu kohdassa numero 8. Asiakasstatuksen tarkistamisen jälkeen luodaan uusi tulkkaus. Tässä funktiossa käytetään samaa event-objektia nappaamaan edellisestä käyttäjätiedot palauttaneesta *Check_if_existing_customer* funktiosta asiakkaan keyString, jonka avulla luodaan uusi tulkkaus kyseiselle asiakkaalle. Pyyntö suoritetaan jälleen kerran POST-muodossa.

```
const axios = require('axios');

exports.handler = function (context, event, callback) {

  // Lisää käyttäjälle uusi pikatulkkaus tunnistetun käyttäjän id:llä. Haetaan id Event-objektilla
  let customerKeyString = event.customerKeyString;

  //Käytettävä endpoint-osoite. Osoitteen parametrina lähetetään tunnistetun asiakkaan keyString, joka haettiin ja
palautettiin vuokaaviosta saadun puhelinnumeron avulla backendin puolelta
  const API_URL = [redacted]
[redacted]{customerKeyString}`;

  //Asiakkaan tarkistamista varten suoritetaan POST HTTP-pyyntö yllä olevaan osoitteeseen. Tällä on tarkoitus luoda
uusi tulkkaus järjestelmään
    const post = new Promise((resolve, reject) => {
      axios.post(API_URL).then((res) => {
        resolve(res.data)
      }).catch((err)=>{
        reject(err);
      })
    })

    //Onnistuneen funktion suorituksen jälkeen tulos lähetetän takaisin vuokaaviolle.
    post.then((result) => {
      const id = result;
      callback(null, result);

    }).catch((err)=>{
      console.log(err)
    })

}
```

Kuva 16 asiakkaan tiedoilla uuden tulkkauksen lisäävä "Find_customer_and_add_interpretation" funktio

3. **Check_interpreter_found**: -> viimeinen vuokaaviossa kutsuttava funktio tapahtuu kohdassa 10. Tässä vaiheessa asiakkaan status on tarkistettu ja tulkkaus on luotu, joten voidaan aloittaa sopivan – eli kieliparin omaavan vapaana olevan tulkin – etsiminen. Tämä tapahtuu jälleen siten, että kun edellinen funktio palauttaa uuden luodun tulkkauksen tiedot, napataan sen keyString talteen ja aloitetaan kysely keyStringiä parametrina käyttäen. Kysely tehdään POST-pyynnön muodossa ja se tapahtuu joka 10. sekunti ja jos tulkkia ei löydy, kyselyä jatketaan niin kauan, kunnes jompikumpi prosesseista aikakatkaisee itsensä – backend tai Function.



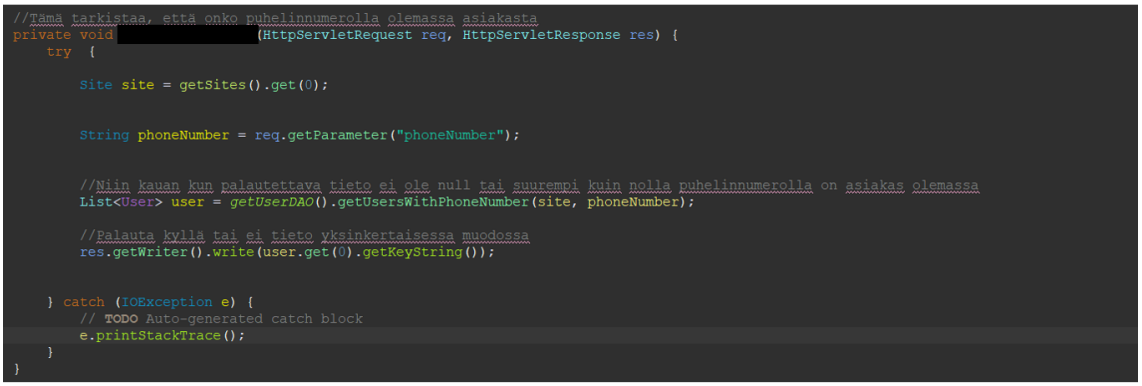
```
const axios = require('axios');

exports.handler = function(event, callback) {

  //Napataan vasta lisätyn tulkkauksen keyString tiedot
  const interpretation = event.keyString;

  //Käytettävä endpoint-osoite. Osoitteen parametrina lähetetään vasta lisätyntulkkauksen keyString. Tulkkaukselle
etsitään tässä vaiheessa tulkkia.
  const API_URL = [redacted]
[redacted]{interpretation}`;

 // Aloitetaan sopivan tulkin pollaaminen backendiltä
  const checkIfInterpreterFound = ()=>{
      axios.post(API_URL).then((res) => {
        const data = res.data;
        //Muutamat virheilmoitusansat ja mikäli niihin tiputaan, kehotetaan asiakasta ottamaan yhteyttä asiakaspalveluun
        if(data === "error"){
          console.log("error");
        } else if(data === false){
          console.log("false");
          // Jatketaan tulkin etsimistä niin kauan, kunnes sopiva löytyy tai backendillä ilmoitetaan aikarajan tulleen
vastaan etsimisellä.
        } else {
          console.log("löyty");
          callback(null, data)
        }
      }).catch((err)=>{
          console.log(err);
        reject(err);
      })
  }
  //Tarkistetaan 10 sekuntn välein, onko tulkkia löytynyt
  const loop = setInterval(checkIfInterpreterFound, 10000)

};
```

Kuva 17 vapaana olevan tulkin statuksen tarkistava "Check_interpreter_found" funktio

7.3 Eclipse

Toteutuksen viimeisenä osana tässä alaluvussa kuvataan sitä, mitä tapahtuu silloin, kun Function-kehitysympäristöstä lähetetään pyynnöt eri aiheissa Eclipsen puoleiselle backendille. Kaikki tehdyt pyynnöt suoritetaan doPost()-metodeina, eli ne saavat parametreikseen servlettien määrittelemät HttpServletRequest ja HttpServletResponse pyyntö– ja vastausobjektit. Pyyntöjä ei tehdä julkisesti avoimen yhteyden kautta, vaan erikseen Twiliolle hyväksytyllä yhteydellä, joka ei vaatinut monimutkaisten autentikointimetodien implementointia.

1. **twilioPhoneOrder**: tämä prosessi tapahtuu edellisen alaluvun kohdan 1. tehdyn pyynnön aikana. Pyynnön parametrina lähetetty puhelinnumero tallennetaan muuttujaan, minkä jälkeen etsitään puhelinnumeroita sisältävästä listasta kyseistä puhelinnumeroa. Jos tulos ei ole null tai 0, niin puhelinnumerolle löytyy asiakastiedot ja kyseisistä tiedoista palautetaan asiakkaan keyString. Mikäli käyttäjä ei ole asiakas, tässä kohtaa voisi tulla vastaan statuskoodi 403 eli Unauthorised.

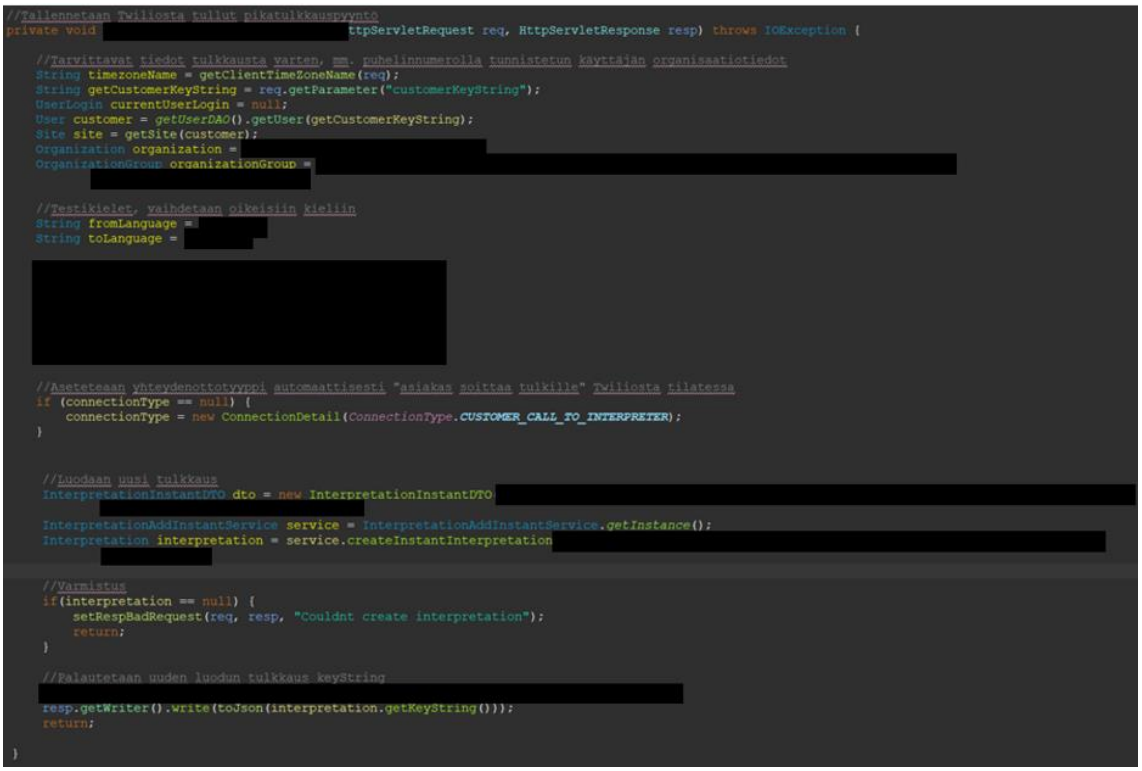


```
//Tämä tarkistaa, että onko puhelinnumerolla olemassa asiakasta
private void [redacted](HttpServletRequest req, HttpServletResponse res) {
    try  {

        Site site = getSites().get(0);


        String phoneNumber = req.getParameter("phoneNumber");


        //Niin kauan kun palautettava tieto ei ole null tai suurempi kuin nolla puhelinnumerolla on asiakas olemassa
        List<User> user = getUserDAO().getUsersWithPhoneNumber(site, phoneNumber);

        //Palauta kyllä tai ei tieto yksinkertaisessa muodossa
        res.getWriter().write(user.get(0).getKeyString());


    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Kuva 18 käyttäjän asiakkaan puhelinnumeron palauttava "twilioPhoneOrder" metodi

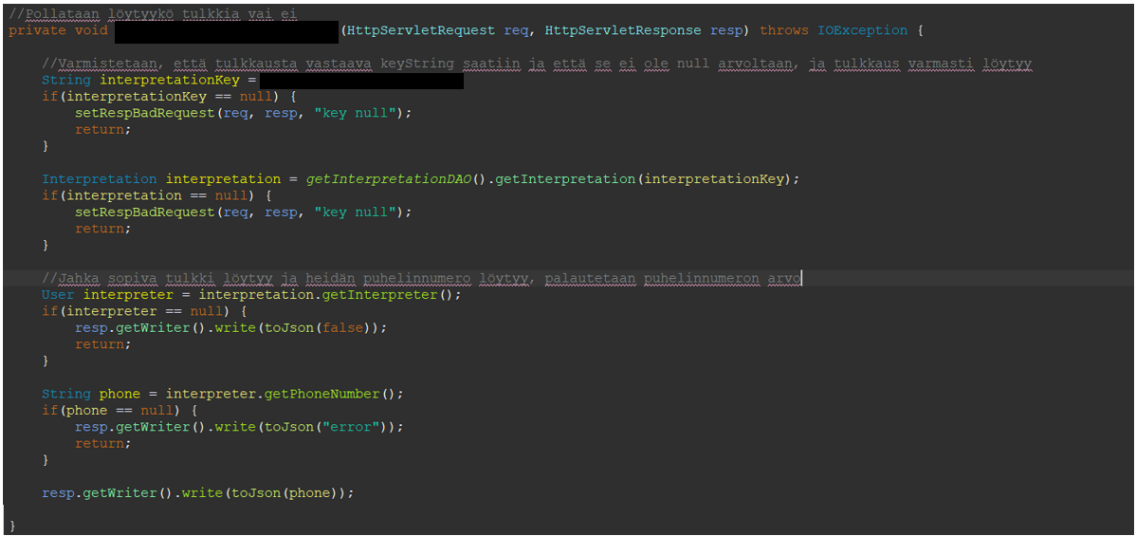
2. **saveInstantInterpretationTwilio**: Kun asiakkaan statuksesta ollaan varmistuttu, voidaan luoda uusi tulkkaus heidän tiedoillaan. Backend puolella voidaan parametrina lähetetyn uniikin avainmerkkisarjan keyString muuttujan avulla etsiä kaikki tarvittavat asiakkaan tiedot uuden tulkkauksen luomista varten. Näitä ovat muun muassa asiakkaan organisaation tiedot. Joitain tietoja asetetaan tässä kohta amyös nulliksi, sillä siinä, missä esimerkiksi applikaation tai asiakaspalvelun kautta tilattavassa tulkkauksessa asiakas saa itse määrätä esimerkiksi haluamansa yhteydenottotavan, tässä ne on asetettava manuaalisti – suurin syy tälle on tilauksen helpottaminen, sillä jos asiakas soittaa itse palveluun tilatakseen tulkin tilannetta tuskin sujuvoittaa, jos asiakkaan pitäisi lopettaa tuo puhelu ja odottaa uutta, tulkin aloittamaa puhelua. Lisäksi muut tiedot, joita ei voi puhelun aikana asettaa, kuten asiakkaan viitetiedot laitetaan tässä kohtaan tyhjiksi, ja täydennetään tulkkauksen jälkeen asiakastietojen avulla. Jahka tarvittavat tiedot on joko noudettu tai asetettu, voidaan niiden avulla luoda uusi tulkkaus. Kun varmistutaan

siitä, että tulkkauksen luominen todellakin onnistui, voidaan kyseisen tulkkauksen keyString muuttuja palauttaa.



Kuva 19 uuden tulkkauksen tallentava "saveInstantInterpretationTwilio" metodi

3. **checkInterpreterFoundTwilio**: viimeisenä kuvailtavana vaiheena tässä tapahtuu sopivan tulkin etsiminen. Tätä metodia kutsutaan funktiossa 3 pyynnön parametrina tallennetun tulkkauksen tiedot. Aluksi varmistetaan, että tulkkaus on ja sille luotu keyString muuttuja ovat todellakin olemassa, minkä jälkeen voidaan aloittaa sopivan tulkin etsiminen. Tässä vaiheessa vapaana oleville, suomi-venäjä kieliparin omaaville tulkeille lähtee pyyntöjä avoimesta pikatulkkauksesta. Heti kun joku näistä tulkeista on hyväksynyt tarjouspyynnön käyttämällään laitteella, tallennetaan heidän puhelinnumeronsa muuttujaan ja palautetaan kyseinen tieto. Näin ollen se voidaan palauttaa funktiossa vuokaaviolle ja siirtää puhelu jatkumaan kyseiseen puhelinnumeroon.



```
//Pollataan löytyykö tulkkia vai ei
private void [redacted](HttpServletRequest req, HttpServletResponse resp) throws IOException {

    //Varmistetaan, että tulkkausta vastaava keyString saatiin ja että se ei ole null arvoltaan, ja tulkkaus varmasti löytyy
    String interpretationKey = [redacted]
    if(interpretationKey == null) {
        setRespBadRequest(req, resp, "key null");
        return;
    }

    Interpretation interpretation = getInterpretationDAO().getInterpretation(interpretationKey);
    if(interpretation == null) {
        setRespBadRequest(req, resp, "key null");
        return;
    }

    //Jahka sopiva tulkki löytyy ja heidän puhelinnumero löytyy, palautetaan puhelinnumeron arvo
    User interpreter = interpretation.getInterpreter();
    if(interpreter == null) {
        resp.getWriter().write(toJson(false));
        return;
    }

    String phone = interpreter.getPhoneNumber();
    if(phone == null) {
        resp.getWriter().write(toJson("error"));
        return;
    }

    resp.getWriter().write(toJson(phone));

}
```

Kuva 20 vapaana olevaa tulkin puhelinnumeron palautta "checkInterpreterFoundTwilio" metodi

8 Tulokset ja yhteenveto

Opinnäytetyön alussa asetetut tavoitteet koskivat asiakaspalvelun ylikuormituksen helpottamista, sillä he joutuvat usein käsittelemään samanlaisia yksinkertaisia ja toistuvia pyyntöjä pikatulkkauksille samalle kieliparille, mille tämä automatisointi toimii ongelman ratkaisuna. Lopullinen palvelu saatiin tehtyä tavoitteiden ja toimeksiantoyrityksen toiveiden mukaisesti, eli palvelun käytössä olevaan puhelinnumeroon soittava käyttäjä tunnistetaan yrityksen asiakkaaksi – tarkistus sen takia, ettei ketä tahansa pystyisi tilaamaan pikatulkkausta, mutta myös olemassa olevien laskutustietojen tarkistamista varten. Käyttäjä saa myös valita kielen puhelun aikana, vaikka kielipareja voi valita vain yhden tällä hetkellä – tähän päätökseen päästiin sen takia, että asiakas on tietoinen tilaamastaan kieliparista, mutta myös sen takia, että jos kieliä päätetään lisätä myöhemmin palveluun, tarvitsee Twilion avulla lisätä vain uusi pienoisohjelma käsittelemään eri syötettä sen sijaan, että lisättäisiin kokonaan kielivalintaprosessi. Kun Twiliolta lähetetty HTTP-pyyntö on päässyt perille backendille ja vapaana oleva tulkki on toivottavasti löydetty sitä kautta, yhdistetään asiakas automaattisesti uuteen puheluun, johon tulkki vastaa, näin aloittaen uuden tulkkauksen.

Aikataulullisesti opinnäytetyössä tuli pidettyä kiinni alkuperäisestä suunnitelmasta, jonka mukaan kesä-heinäkuun aikana toimeksianto saatiin yritykseltä, sekä sidoshenkilöiden kanssa tehtiin tarvittavat rajaukset projektille, joita olivat muun muassa kieliparien määrien rajaukset sekä mitä ne ovat – yksi kielipari, jossa lähtökieli on suomi ja kohdekieli on venäjä. Lisäksi tässä ajassa asetettiin tarvittavat tavoitteet työlle ja aloitettiin varsinaista toiminnallista osuutta tutustumalla uuden ohjelmiston, eli Twilion, dokumentointiin sekä muihin työkaluihin, joita käytettäisiin esimerkiksi testausprosessin aikana. Elokuun aikana toiminnallista osuutta oli alun perin tarkoitus saada lopeteltua, mutta jo ennakoitujen riskien mukaisesti sairastapausten ja kesälomien vuoksi toiminnallinen tekeminen hidastui. Testausta aloiteltiin kuitenkin syyskuun vaiheessa ja lopullisia muutoksia työhön viimeisteltiin syyskuun aikana. Loka-marraskuu välillä keskityttiin suunnittelu- ja toteutusseminaarien pitoihin, sekä kirjoituspajojen suorituksiin. Näin kirjoitusprosessi sai alkunsa ja marraskuun loppu- sekä joulukuun alkupuolella lopullinen työ saatiin valmiiksi.

9 Pohdinta

Kuten Johdanto-osuudessa mainittiin, yhtenä mahdollisena jatkokehitysmahdollisuutena työlle on muiden, yritykselle suuressa kysynnässä olevien tulkattavien kielten lisääminen – näitä kieliä voisivat mahdollisesti olla esimerkiksi ukrainan ja eri arabian kielten lisääminen. Tämä johtuu edelleen tarpeesta automatisoida toistuvia puhelinpyyntöjä pikatulkkauksesta näille kielille.

Opinnäytetyötä on myös helppo hyödyntää yrityksen sisällä, sillä kaikki työkalut ovat yrityksen kustantamia tai jo heillä olemassa olevia, sekä työn tekijän esimiehet ovat seuranneet viikoittain työn edistymistä viikoittaisissa suunnittelu- ja teknologiatiimin palavereissa, joten he ovat tietoisia mahdollisista kehitystarpeista projektia varten.

Opinnäytetyön alkuvaiheessa raameja rajatessa, tavoitteita asettaessa ja toimintatapoja miettiessä pyrittiin mahdollisimman laajasti etsimään erilaisia työkaluja tekijän käyttöön, sekä etsiä sopiva työntekomenetelmiä. Laajalti nämä

toteutuivatkin, eli kovin paljoa – jos mitään – ei tehty eri tavalla kuin mitä alussa haluttiin.

Omasta näkökulmasta opinnäytetyöni tarjosi minulle hyvän mahdollisuuden syventyä jo aikaisemmin oppimaani – koulussa opittu on kuitenkin aina vain teoriaa. Sen soveltaminen ensimmäistä kertaa harjoittelussa oli mielekästä puuhaa, mutta silloin tämän alan työelämään niin ummikkona tulleena ihmisenä oli hienoa huomata, että pääsin soveltamaan uudestaan samoja, jo oppimiani taitoja. Pidän itseäni myös melko asiakaspalveluhenkisenä ihmisenä ja olen nauttinut työtä tehdessä siitä, että pääsen auttamaan ihmisiä tekemällä jotain konkreettista heidän elämäänsä; alun perin humanistiselta alalta ammattikorkeakouluun vaihtaneena opiskelijana kaipasin silloin jotain konkreettista annettavaa ihmisille ja halusin jättää jonkinmoisen pysyvämmän jäljen maailmaan ja koen onnistuneeni siinä tavoitteessa tämän opinnäytetyön myötä.

Viimeisenä tämän opinnäytetyön kannalta halutaan kiittää eritoten toimeksiantoyrityksen opetusasiantuntija Asser Kakkoa, sekä Senior System Developeria Joonas Krühnia. Kyseiset henkilöt auttoivat suuresti opinnäytetyön aikana minua, tarjoten ammatillisia taitojaan prosessin tekemiseksi mahdollisimman käyttäjäystävälliseksi.

Lähteet

Crawford, W., Hunter, J. 2001. Java Servlet Programming, Second Edition. O'Reilly Media, Inc.https://books.google.fi/books?id=u9WbAgAAQBAJ&lpg=PR5&ots=lr3auiSprf&dq=servlet&lr&hl=fi&pg=PA6#v=onepage&q&f=false. 21.10.2024.

Foreca. 2024. https://corporate.foreca.com/en/. 01.12.2024

García Gallardo, E. 2023. What is back-end development?. Built In. https://builtin.com/software-engineering-perspectives/back-end-development. 21.10.2024

Holopainen, O. 2022. Pikatulkkauksen laatu tulkkien näkökulmasta. Pro gradu -tutkielma, Itä-Suomen yliopisto. https://erepo.uef.fi/bitstream/handle/123456789/27593/urn_nbn_fi_uef-20220496.pdf?sequence=1&isAllowed=y. 10.09.2024

Internet Assigned Numbers Authority. 2024. Hypertext transfer protocol (HTTP) status code registry. https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml. 10.09.2024

Merriam-Webster. 2024. back end. Merriam-Webster. https://www.merriam-webster.com/dictionary/back%20end. 28.11.2024

Oracle. 2010. *Java EE 5 tutorial: The persistence lifecycle*. https://docs.oracle.com/javaee/5/tutorial/doc/bnafe.html. 07.09.2024

Oracle. 2024. Interface Servlet. https://docs.oracle.com/javaee/7/api/javax/servlet/Servlet.html. 22.10.2024.

Salminen-Tuomaala, M., Hautamäki, T., & Sarvikas, H. 2023. Kohti laadukkaita toiminnallisia opinnäytetöitä. Seinäjoen ammattikorkeakoulu. https://urn.fi/URN:NBN:fi-fe20231211153209. 07.09.2024.

Twilio. 2024a. What is Twilio? An introduction to the leading customer engagement platform. Twilio Inc. https://twilio.com/en-us/resource-center/what-is-twilio-an-introduction-to-the-leading-customer-engagement-platform. 07.09.2024

Twilio. 2024b. Function Docs. Twilio Inc. https://www.twilio.com/docs/serverless/functions-assets/functions. 23.10.2024

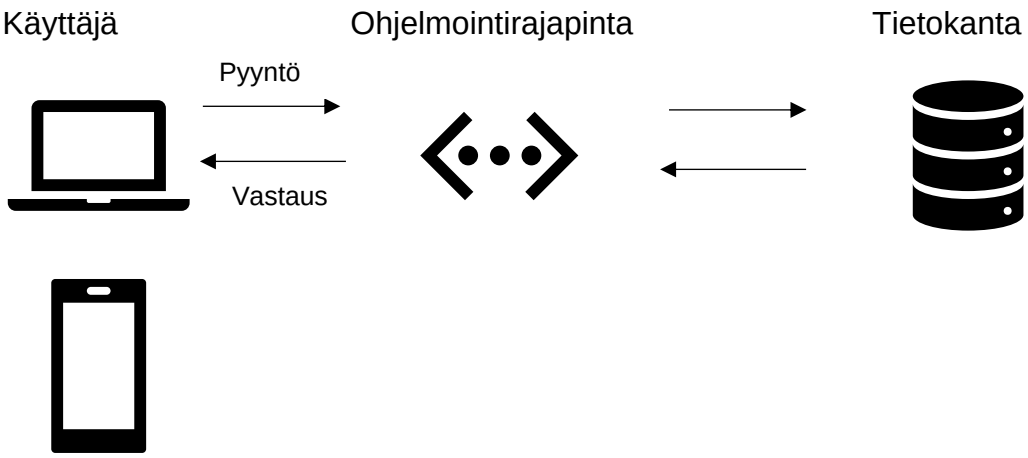
Twilio. 2024c. Studio Docs. Twilio Inc. https://www.twilio.com/docs/studio. 23.10.2024

Youpret. 2024a. Youpretin tarina. https://www.youpret.com/fi/youpret-lyhyesti/. 01.12.2024

Youpret. 2024b. Youpret työpaikkana. https://www.youpret.com/fi/youpret-tyopaikkana/. 01.12.2024

Kuviot

Kuvio 1 Ohjelmointirajapinnan toiminta kuvattuna



Kuvio 2 Servletin toiminta kuvattuna