Harri Filppula

TIETOTURVALLINEN JA YKSITYINEN TEXT JA VOICE CHAT - SOVELLUS PC:LLE

Tietojenkäsittelyn koulutusohjelma
2018

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

TIETOTURVALLINEN JA YKSITYINEN TEXT JA VOICE CHAT -SOVELLUS PC:LLE

Filppula, Harri
Satakunnan ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
helmikuu 2019
Sivumäärä: 56
Liitteitä: 0

Asiasanat: tietoturva, yksityisyys, ohjelmointi, verkkojuttelu

---

Opinnäytetyön aiheena oli tutkia tietoturvan ja yksityisyyden tilaa nykyisessä sähköisessä viestinnässä, ja kehittää sen pohjalta tietoturvallinen ja yksityinen text ja voice chat -sovellus pc:lle. Tavoitteena oli rakentaa yksinkertainen, mutta toimiva asiakas-palvelin -kokonaisuus, jonka pääprioriteetteina olisi käyttäjän hyvän yksityisyyden ja hyvän tietoturvan varmistaminen. Lisäksi työn tavoitteena oli olla erityisesti opettavainen ja haastava, mutta myös toteutettavissa oleva kokonaisuus, joka kartuttaa ohjelmointiosaamistani sekä asioiden todellista ymmärtämistä ohjelmoinnin eri osa-alueilla.

Opinnäytetyöni teoreettinen osuus keskittyi tutkimaan tietoturvan ja yksityisyyden tilaa nykyisessä sähköisessä viestinnässä. Osuus käsitteli tietoturvaa ja yksityisyyttä globaalisti yleisellä tasolla, sekä tarkemmin Suomen kohdalla. Lisäksi teoreettinen osuus käsitteli erikseen yksityisyyden ja tietoturvan tilaa text ja voice chat -sovellusten kohdalla yleisesti, sekä ottaen vertailukohteeksi muutaman esimerkkisovelluksen. Teoreettinen osuus nojautui hyvin pitkälti ajankohtaisiin web-artikkeleihin sekä Suomen ja EU:n lainsäädäntöihin aiheista.

Työn konkreettinen osuus oli Speech Bubble -niminen text ja voice chat -sovellus, jonka päätavoite oli tuoda ratkaisu teoreettisessa osuudessa esitettyihin chat-sovellusten epäkohtiin tai puutteisiin. Tavoitteisiinsa sovellus pyrki avoimella lähdekoodilla, sekä antamalla käyttäjälle itselleen päätäntävallan siitä, miten ja missä käyttäjä haluaa palvelinsovellusta hostata.

Työn lopputuloksena syntyi chat-sovellus nimeltä Speech Bubble, joka on hyvä ja toimiva kokonaisuus pienille käyttäjämäärille. Sovelluskokonaisuus tekee sen mihin se pyrki: se pitää huolen käyttäjiensä hyvästä yksityisyydestä ja hyvästä tietoturvasta toimimalla hyvien periaatteiden mukaan ja välttämällä epäloogisuudesta johtuvat virheet. Lopputuloksen yhteydessä selvisi myös hyviä parannusehdotuksia sekä jatkokehitysideoita. Parannusehdotuksissa ja jatkokehitysideoissa esitellyt seikat koskivat enimmäkseen käytettyjen menetelmien soveltuvuuden kyseenalaistamista ja arviointia. Nämä ehdotukset toimivat hyvinä esimerkkeinä tulevaisuuden projekteissa ja menetelmien käytössä yleisellä tasolla kertomalla siitä, miten jokin asia kannattaa toteuttaa ja miten ei. Tässä työssä esitelty sovellus onkin keskeneräinen projekti, jota on tarkoitus jatkokehittää tulevaisuudessa. Yksi työn epäsuorista tavoitteista olisi saada projektiin tulevaisuudessa innokkaita kehittäjiä mukaan jatkokehitysideoiden toteuttamiseen. Yhdessä oppimisen ja tekemisen kautta Speech Bubble voi saada hyvin tuulta siipiensä alle.

SECURE AND PRIVATE TEXT AND VOICE CHAT -APPLICATION FOR PC

Filppula, Harri
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in data processing
February 2019
Number of pages: 56
Appendices: 0

Keywords: security, privacy, programming, chat

The purpose of this thesis was to research the current state of security and privacy in electrical communications, and to develop a secure and private text and voice chat -application for pc, based on the results of the research. The goal was to build a simple, yet functional client-server -entity, whose main priorities were to ensure good privacy and good security of the end users. Additionally the project aimed at to be teaching and challenging but also doable entirety that accumulates my programming skills and the real understanding of different things in the programming.

The theoretical part of my thesis focused on studying the current state of security and privacy in the electrical communications. The part covered security and privacy in global scope as well as in the scope of Finland. In addition, the part separatedly addressed the state of privacy and security in text and voice chat applications in general, and by comparing different chat-applications. The theoretical part leaned mostly on topical web articles as well as on the laws of Finland and EU.

The concrete part of my thesis was a text and voice chat -application called Speech Bubble, of which the main objective was to create a solution for the drawbacks and shortcomings of the chat applications introduced in the theoretical part. The application pursued its objective by open sourcing itself and by giving the user the authority to decide how and where he would like to host the server.

As an end result, a chat application called Speech Bubble was created. The application is good and functional entity for small userbase. The application entirety does what it was aiming for: to take good care of users privacy and security by operating according to good principles, and by avoiding mistakes that originates from illogicality. Good improvement proposals and ideas also turned out over the end result. These things stated in those proposals and ideas considered mainly evaluation and questioning the suitability of some of the methods used in the application. These suggestions also work as good examples for future projects by telling in general about how some things should be executed and how they shouldn't be. The application explained in this thesis is in fact unfinished project, that is meant to be developed further in the future. One of the indirect goals of this project was to get some enthusiastic developers to contribute to this project. By doing and learning together Speech Bubble could get some breeze under the wings.

SISÄLLYS

1 JOHDANTO

Erilaiset sähköiset viestintäkanavat ovat nykyään yhä tiiviimmin integroituneet päivittäiseen elämäämme. Niistä on muodostunut niin luonnollinen osa arkeamme, että en esimerkiksi omalla kohdallani osaa sanoa milloin olen viimeksi ollut päivän aikana lähettämättä kenellekään minkäänlaista sähköistä viestiä missään muodossa. Nykyään näitä viestintäkanavia on enemmän kuin tarpeeksi, ja jaamme näiden kanavien kautta hyvinkin henkilökohtaisten asioiden lisäksi myös kuvia ja videoita enemmän kuin koskaan. Vaikka fyysisesti pitäisimme elämämme hyvinkin yksityisenä, voi yksityisyyden todellinen tila olla aivan jotain muuta virtuaalisessa maailmassa. Tuottamamme data kertoo meistä yleensä paljon enemmän kuin mitä haluamme myöntää, ja tämän takia jokaisen prioriteetteina nykyisessä sähköisessä viestinnässä tulisi olla yksityisyyden ja tietoturvan vaaliminen.

Radikaalit muutokset yksityisyyden ja tietoturvan kohdalla ovat harvinaisia, sillä se herättäisi liiakseen kuluttajien huomion. Käyttäjien data on nykyisin yhä arvokkaampaa, sillä datan avulla ihmisiin päästään myös vaikuttamaan. Tämän takia on huolestuttavaa, että nykyään erilaiset sähköiset viestintäsovellukset tuntuvat olevan yhä enenevässä määrin otsikoissa erilaisten tietoturvan tai yksityisyyden laiminlyöntien takia. Koska tietoturva ja yksityisyys kulkevat kuitenkin käsi kädessä, toisen laiminlyönti vaikuttaa myös toiseen.

Opinnäytetyöni alkuvaiheessa tutkin tietoturvan ja yksityisyyden nykyistä tilaa yhteiskunnassamme ja globaalilla tasolla. Myöhemmin työni keskittyy erityisesti eri text ja voice chat -sovellusten tietoturvan ja yksityisyyden tutkimiseen ja arvioimiseen. Lisäksi työni konkreettisessa osuudessa esittelen oman ratkaisuni työssäni esiteltyihin text ja voice chat -sovellusten haasteisiin. Tämä ratkaisu on Speech Bubble -niminen text ja voice chat -sovellus, jonka pääprioriteetteina on käyttäjän hyvän yksityisyyden ja hyvän tietoturvan varmistaminen. Pyrin työlläni tuomaan esiin asioita, joita kannattaa ottaa huomioon arkisessa elämässämme ja viestintäväyliemme käytössä. Kaikenkaikkiaan toivon, että työni voisi valaista epäselvyyksiä, ja tuoda uusia näkökulmia yksityisyyteen ja tietoturvaan liittyvissä asioissa.

2 TIETOTURVAN JA YKSITYISYYDEN MERKITYS SÄHKÖISESSÄ VIESTINNÄSSÄ

2.1 Tietoturva

Tietoturva on nykyään hyvin keskeinen aihealue käytännössä kaikessa sähköisessä viestinnässä. Internetin avoin rakenne ja luonne ovat syy siihen, miksi se on onnistunut kukoistamaan ja leviämään vuosien saatossa; niin hyvässä kuin pahassa. Sähköisen viestinnän monimuotoisuus ja moniulotteisuus, mikä mielestäni on hyvä asia, on kuitenkin osaltaan luonut hyvän pohjan erilaisille tietoturvahyökkäyksille ja riskeille. Monimutkaisuudeltaan ja kooltaan erilaiset hyökkäykset tähtäävät pääasiassa web-sovellusten alttiina oleviin ominaisuuksiin. Pelkästään jopa Viestintäviraston Tietoturva nyt! -artikkelikokoelman suuruus kertoo selkeästi hyvän tietoturvan kasvavasta merkityksestä, niin Suomessa kuin muuallakin maailmassa. Tietoturva on mielestäni asia, josta ei koskaan kuuluisi tinkiä rahan tähden. (Viestintäviraston www-sivut 2018, Cloudflare Inc. 2018)

Samalla, kun erilaiset palvelut nojautuvat yhä vahvemmin internetin tuomiin helpotuksiin, kasvaa tietoturvan merkitys entisestään. Esimerkiksi Internet Of Things -ajatus on saanut nykyään yhä enemmän huomiota ja samalla alamme hiljalleen jakaa internettiin datan lisäksi myös kaikenlaisia laitteita. Itseäni lähinnä karmii Internet of Things -ajattelu, sillä mielestäni ei ole viisasta keskittää kirjaimellisesti kaikkia elämämme osa-alueita yhteen verkon kautta, jonka historia on jo useaan otteeseen osoittanut epätäydelliseksi. Tässä kohtaa hyvä tietoturva on ehdoton elinehto, sillä esimerkiksi Viestintäviraston Suomen Kybersää 09/2018 kertoo, kuinka erilaiset palvelunestohyökkäykset, haavoittuvuudet, kiristysohjelmat, huijaukset ja vakoilut ovat käytännössä vain kasvaneet ajan saatossa. Vaikka intrernetin turvallisuuden täydellinen hallitseminen lienee mahdotonta, kaikki askeleet täydellisyyttä kohti ovat aina tärkeitä. Esimerkiksi Google alkoi kesäkuusta 2018 eteenpäin merkitä Chromessa vain https:ää käyttävät sivustot turvallisiksi. Tämä on yksi hyvä tapa herätellä sovelluskehittäjiä tietoturvakysymysten vakavasti ottamiseen. On kuitenkin hyvä huomata, että Emily Schecterin kirjoittamassa Googlen tietoturvallisuusblogissa ei mainittu sanallakaan yksityisyyttä. Tämä on mielestäni kovin outoa, sillä tietoturva ja

yksityisyys kulkevat kuitenkin aina käsi kädessä. (Viestintäviraston www-sivut 2018, Schecter 2018, Komulainen & Puopolo 2018, Russell 2017)

2.1.1 Tietoturva Suomen laissa

Suomen laki takaa jokaiselle Suomen kansalaiselle oikeuden luottamukselliseen sähköiseen viestintään. Tämä tarkoittaa sitä, että millään taholla ei ole oikeutta käsitellä kenenkään toisen sähköisiä viestejä tai viestintään liittyvää muuta dataa. Poikkeuksena ovat tilanteet, joissa Suomen laki erikseen antaa oikeuden viestinnän tutkimiseen, tai viestinnän osapuolet ovat antaneet käsittelylle suostumuksensa. Lain tavoite on taata jokaiselle osapuolelle hyvän ja turvallisen ympäristön sähköisten palveluiden tuottamiselle ja kuluttamiselle Suomessa, ja näin varmistaa, että kenenkään oikeudet, tietosuoja ja tietoturva eivät vaarannu. Näin siis jokaisen suomalaisen sähköisen viestintäpalvelun tuottajan tulee huolehtia palvelunsa tietoturvasta ja luottamuksellisuudesta. Puhuttaessa tietosuojasta puhutaan samalla myös tietoturvasta, sillä tietoturva on yksi tietosuojan toteuttamistapa. Laissa määritellyn tietoturvan ehdot pyrkivät suojaamaan tietojärjestelmät ja tietoaineisto. Näin ollen tietoturva tarkoittaa niitä teknisiä toimenpiteitä, joilla varmistetaan tiedonvälityksen ja käsittelyn eheys ja luottamuksellisuus, tietojärjestelmien käytettävyys, sekä tiedon omistajan eli rekisteröidyn oikeuksien täyttyminen ja toteutuminen. Nämä ovat mielestäni hyviä periaatteita, mikäli ne onnistutaan toteuttamaan käytännössä yllä esitetyllä tavalla. (Tietosuojavaltuutetun www-sivut 2018)

Luottamuksellisen viestinnän loukkaaminen on rangaistava teko, ja se käy ilmi Suomen perustuslaissa. Rajoituksena luottamukselliselle viestinnälle ovat kuitenkin lain oikeuttamat viranomaiset, jotka voivat tietyissä tilanteissa tutkia viestejä ja niiden välitystietoja joidenkin rikosten ehkäisyssä ja tutkinnassa. Viranomaisten oikeuksista luottamuksellisen viestinnän rajoittamiseen säädetään pakkokeinolaissa, poliisilaissa, henkilötietojen käsittelystä rajavartiolaitoksessa ja Tullissa annetuissa laissa sekä Rajavartiolaitoksen rikostorjunnasta annetussa laissa. Mielestäni toistaiseksi lait toimivat lähes tarkoituksenmukaisesti. Uudet tiedustelulainsäädännön muutokset kuitenkin maalaavat aika huolestuttavaa kuvaa siitä, millaisin perustein ja milloin virkavallalla on oikeus tiedustella kansalaisia salaa. Nämä lait eivät myöskään voi antaa takeita

ehtojen toteutumisesta Suomen rajojen ulkopuolella, sillä joidenkin maiden lainsäädäntö tietoturvaa koskien on huomattavasti löyhempi kuin Suomessa. (Viestintäviraston www-sivut 2018, Laki sähköisen viestinnän palveluista 917/2014, 243§ 1 mom. 1-16 k. Laki sähköisen viestinnän palveluista 917/2014, 322§, Suomen perustuslaki 731/1999, 10§)

2.1.2 Tietoturvan haasteet

Hyvän tietoturvan säilyttäminen ja ylläpitäminen ei ole ikinä helppoa. Koska tietoturva ja yksityisyys kulkevat käsi kädessä, täytyy löytää myös kultainen keskitie yksityisyyden ja turvallisuuden välillä. Lisäksi kilpajuoksu uusien tietoturva hyökkäysten ja -aukkojen ilmestymisten ja niiden estämisen ja paikkaamisen välillä tuntuu loputtomalta. Kyseinen tunne ei toisaalta ole kaukana totuudesta. Vuosittain miljoonat ihmiset joutuvat jonkinlaisen tietoturvaloukkauksen kohteeksi. Loukkausten vakavuus vaihtelee suuresti: Kyseessä voi olla esimerkiksi vahinko, jossa lääkäri katsoo väärän potilaan tietoja. Suuremman skaalan tapauksissa voi esimerkiksi omistautunut joukko hakkereita murtautua valtion järjestelmiin saadakseen tietoa vaikka armeijan salaisuuksista.

Teknologioiden jatkuva nopea kehitys on erittäin hyvä asia tietoturvan näkökulmasta, sillä uudet teknologiat, erityisesti sähköisessä viestinnässä, pyrkivät paikkaamaan valmiiksi tiedossa olevat vanhat ja ilmiselvät putteet. Toisaalta samalla voivat erilaiset haavoittuvuudet niin vanhemmissa kuin uudemmissakin teknologioissa helposti jäädä nopean kehityksen varjoon. Suurien kokonaisuuksien täydellien hallinta on käytännössä mahdotonta, mutta täydellisyyteen pyrkiminen on myös elinehto sähköisessä viestinnässä, sillä arkaluontoisen datan siirto internetin kautta on jatkuvaa.

Yksi suurimmista haasteista tietoturvan kanssa on teknologioiden moninaisuus ja monimuotoisuus, sillä tietoturvahyökkäykset ja tietoturva-aukkojen hyödyntäminen voi tapahtua niin monesta eri suunnasta ja niin monella eri tavalla, että jopa parhaiten suojatut järjestelmät eivät ikinä ole täydellisesti näiden uhkien saavuttamattomissa. Esimerkiksi loppuvuodesta 2017 löydettiin Wifi-verkkojen salaukseen suurimmassa osassa laitteista käytettävästä WPA2 protokollan käsittelystä vakava haavoittuvuus.

Kyseessä on todellisuudessa useamman haavoittuvuuden yhdistelmä, joka tunnetaan nimellä KRACKs (Key ReinstallationAttaCKs). Tämä haavoittuvuus mahdollistaa langattomien verkkojen salauksessa käytettyjen parametrien selvittämisen, ja näiden parametrien avulla on hyökkääjän mahdollista purkaa Wifi-verkossa kulkevan liikenteen salauksen. Tämä haavoittuvuus herättää kysymyksen siitä, milloin seuraavaksi kuulemme jonkin muun turvalliseksi ajatellun protokollan pettäneen. Yleisesti ottaen WPA2-salausta onkin pidetty erittäin turvallisena ja hyvänä protokollana, vaikka se ei täysin uusi keksintö olekaan. (Symantec Corporationin www-sivut n.d.)

Toinen hyvä esimerkki puolestaan uudemman teknologian haavoittuvuuksien hyödyntämisestä on PortSmash. Kyseessä on niin kutsuttu side-channel -haavoittuvuus, joka pyrkii varastamaan salauksiin käytettävät avaimet käyttämällä hyödyksi samanaikaista multi-threadingiä. Tämä koskee prosessoreita, jotka tukevat hyperthreadingiä. Hyperthreadingissä yhtä fyysistä ydintä kohtaan luodaan kaksi loogista ydintä, ja PortSmashiä on todistetusti hyödynnetty prosessoreissa, jotka ovat arkkitehtuuriltaan Skylake ja KabyLake. Haavoittuvuuden arvellaan kuitenkin toimivan myös AMD:n hyperthreadingiä tukevissa prosessoreissa. Kyseinen haavoittuvuus huolettaa myös sen tähden, että Intel johtaa markkinoita vielä tällä hetkellä, joten haavoittuvuuden vaikutuspiiri on hyvin suuri. (Mayersen 2018)

Edellä mainitut esimerkit kertovat mielestäni hyvin haavoittuvuuksien ja tietoturvauhkien luonteesta: Niitä on aina ollut, tulee aina olemaan ja ne odottavat vain löytäjäänsä. Suurimmat haasteet ovat siis jatkuva kilpajuoksu haavoittuvuuksia ja uhkia vastaan, sovellus- ja järjestelmäkokonaisuuksien tietoturvan kokonaisvaltainen hallinta ja tasapainoilu yksityisyyden ja tietoturvan välillä. (Cloudflare Inc. 2018, Viestintäviraston www-sivut 2017, Goodin 2017)

2.2 Yksityisyys

Yksityisyys on yhtä keskeinen osa sähköistä viestintää kuin tietoturva. Voidaan myös todeta, että ne kulkevat jatkuvasti käsi kädessä: Hyvää tietoturvaa ei voida saavuttaa, jos yksityisyyttä riisutaan liikaa. Samalla tavalla hyvää yksityisyyttä ei saavuteta ilman hyvää tietoturvaa. Tämä on tärkeä tiedostaa, sillä elämämme on yhä suuremmalta osin

digitalisoitua. Nykyään sähköisessä viestinnässä tuotamme enemmän dataa yhden päivän aikana kuin mitä olemme tuottaneet koko historiamme aikana. Samalla tavalla kuin täydellistä tietoturvaa on mahdoton saavuttaa, täydellinen anonyymiys internetissä on nykyään vain teoriaa. Data, jota luomme erilaisten web-palveluiden kautta muodostaa suuren ja tarkan kuvan meistä. Lisäksi linkittäessämme näitä eri palveluita toisiinsa käyttäjien ja käyttöoikeuksien kautta, herää kysymys siitä, miten yksityistä digitaalinen elämämme todella on. Tästä asiasta voisi käyttää termiä ”teknologiasokeus”, jossa kuluttajaa ei kiinnosta tarpeeksi se, miten hänen digitaalisesta elämästä on jäljellä enää yksityisyyden rippeet. Ennen puhelimia ja muita teknologian tuotteita ihmisillä oli konkreettisesti jäljellä muutakin kuin pelkät yksityisyyden rippeet, sillä esimerkiksi nykyinen anonyymius ei verkossa pidä enää paikkaansa. Adam Scott kertoo vuonna 2016 julkaistussa kirjassaan ”Building web applications that respect a users privacy and security” miten hakukoneet pystyvät eri mekanismein päättelemään persoonallisuutemme piirteitä ja terveydentilaa. Kirjassaan hän kertoo John Paparrizos MSc:n, Ryen W. White PhD:n ja Eric Horwitz MD PhD:n julkaisemasta, jossa he onnistuivat ennakoimaan haimasyövän diagnoosin käyttämällä tutkimukseen Bing-hakukoneen anonyymejä hakukyselyitä. Huolestuttavan tutkimuksesta tekee mielestäni se, että anonyymi data ei todellisuudessa ole riittävän anonyymiä. (Scott 2016)

Yksityisyys on kulttuurin luoma asia, ja yksityisyyden käsite on vuosien saatossa muuttunut hyvinkin paljon. Vanhemman sukupolven saattaa olla erittäin hankala ymmärtää nykyistä käsitystä yksityisyydestä jos esimerkiksi mietimme asiaa sen valossa, mitä ihmiset ovat valmiita jakamaan elämästään sosiaalisessa mediassa. Internetissä jakamamme kuvat ja videot, lähettämämme viestit, klikkailemamme linkit ja käyttämämme hakusanat antavat enemmän kuin tarpeeksi tietoa persoonallisuudestamme, vakaumuksistamme ja mieltymyksistämme itsemme profilointiin. Kohdennettujen mainosten ja linkkien avulla on mahdollista vaikuttaa niin mielipiteisiimme kuin käyttäytymiseemme. Vaikka luottokorttimme tiedot, sekä muut päällisin puolin arkaluontoiselta näyttävät asiat olisivat digitaalisessa maailmassa hyvin tallessa, sähköisessä viestinnässä jättämämme digitaalinen jalanjälki, eli luomamme data, on eri yrityksille paljon arvokkaampaa, kuin osaamme arvatakaan. Näiden asioiden takia digitaalisen yksityisyyden vaaliminen on aina vain tärkeämpää. Lisäksi mielestäni joissain tilanteissa on epäreilua se, että yritykset

rikastuvat minun datani avulla, kompensoimatta sitä minulle lähes mitenkään. Samalla en myöskään pysty määrittelemään tuottamalleni datalle arvoa, sillä sähköisen viestinnän abstraktiotaso on aina suhteellisen korkea. (Wibson 2018, Akred & Samani 2018, Paz 2014)

2.2.1 Yksityisyys Suomen laissa

Henkilötiedoilla laissa tarkoitetaan kaikkia niitä tietoja, joita on mahdollista liittää luonnolliseen, tunnistettavissa olevaan tai tunnistettuun henkilöön. Lisäksi henkilötietoja ovat myös luonnollisen henkilön elinolosuhteita tai ominaisuuksia kuvaavia asioita, jotka voidaa tunnistaa koskemaan hänen perhettään, kanssaan asuvia muita henkilöitä tai häntä itseään. Näitä tietoja voi olla talletettuna esimerkiksi sähköisessä muodossa, paperisena, äänitallenteena tai viedotallenteena. Henkilötietojen käsittelyllä laissa tarkoitetaan yllä mainittujen tietojen keräämistä, tallettamista, suojaamista, muuttamista, yhdistämistä, poistamista ja luovuttamista sekä muita näihin tietoihin kohdistuvaa käsittelyä.

Lait yksityisyydestä ja yksilön tietosuojasta on tehty varmistamaan, että jokaisen vapaudet ja oikeudet toteutuvat hänen henkilötietojensa käsittelyssä. Tietosuojan pääpyrkimys on osoittaa millä perustein ja milloin rekisteröidyn, eli asianomaisen henkilötietoja voidaan käsitellä. Suomen laissa yksityisyyden suojaan vaikuttavat henkilötietolaki, laki sähköisen viestinnän palveluista, työelämän tietosuojalaki, luottotietolaki, EU:n yleinen tietosuoja-asetus sekä EU:n tietosuojadirektiivi. Nämä lait yhdessä pyrkivät turvaamaan yksilön tietojen käsittelyn siten, että tietoja kerätään ja käsitellään tiettyjä ja laillisia tarkoitusperiä varten ja ettei niitä käsitellä yhteensopimattomalla tavalla näiden tarkoitusperien kanssa. Lait määrittävät myös sen, mitä tietoja ja milloin viranomainen on oikeutettu käsittelemään. Näitä tilanteita ovat esimerkiksi tiettyjen rikosten ennaltaehkäiseminen tai niiden selvittely. (Tietosuojavaltuutetun www-sivut 2018, Viestintäviraston www-sivut 2015, Euroopan parlamentin ja neuvoston direktiivi luonnollisten henkilöiden suojelusta henkilötietojen käsittelyssä sekä näiden tietojen vapaasta liikkuvuudesta ja direktiivin 95/46/EY kumoamisesta, 27.04.2016, (EU) 2016/679, EUVL L 119 4.5.2016, 1, Euroopan parlamentin ja neuvoston direktiivi luonnollisten henkilöiden suojelusta

toimivaltaisten viranomaisten suorittamassa henkilötietojen käsittelyssä rikosten ennalta estämistä, tutkimista, paljastamista tai rikoksiin liittyviä syytetoimia tai rikosoikeudellisten seuraamusten täytäntöönpanoa varten sekä näiden tietojen vapaasta liikkuvuudesta ja neuvoston puitepäätöksen 2008/977/YOS kumoamisesta, 27.4.2016, (EU) 2016/680, EUVL L 119 4.5.2016, 89 Henkilötietolaki 523/1999, 3§ 1 mom. 1-2 k., Suomen perustuslaki 731/1999, 10§ 1 mom.)

2.2.2 Yksityisyyden haasteet

Yksityisyyden käsite sähköisessä viestinnässä on huomattavasti monimutkaisempi kuin miten käsitämme yksityisyytemme sähköisen viestinnän ulkopuolella. Kun asiat siirtyvät virtuaalimaailmaan ja muuttuvat abstraktimmaksi kuin mitä ne ovat konkreettisesti, on hyvän yksityisyyden tavoittelu ja ymmärtäminen yhä hankalampaa. Yksi tietosuojan ja yksityisyyden suurimmista haasteista onkin aihealueen laajuus ja jatkuva laajentuminen. Päivittäin tuottamamme datamäärä sähköisessä viestinnässä on valtavaa, ja tämän datamäärän hallitseminen yksityisyyden näkökulmasta tuntuu lähes mahdottomalta. Suomen lain puitteissa tämä datan hallinnan kokonaisuus tuntuu vielä jokseenkin hallittavalta, mutta sähköisessä viestinnässä maiden väliset rajat ovat olemattomat. Tällöin käyttäessämme toisen maan lain alla toimivan sähköisen palveluntuottajan palvelua ei Suomen laki ole enää turvaamassa tietosuojaamme. Käyttäessämme esimerkiksi samaa käyttäjää useassa eri palvelussa, jaamme tahtomattamme myös muuta dataa näiden palveluiden kesken pelkän käyttäjädatan lisäksi. Mielestäni tällaisessa tilanteessa yksityisen datan hallitseminen hankaloittuu palveluiden mahdollisesti hyvinkin erilaisten tietosuojakäytäntöjen osalta. (Viestintäviraston www-sivut 2018)

Datan omistajuuden kysymys on eräs yksityisyyden haaste. Kuka todella omistaa sen datan, jonka olemme luoneet jonkun palveluntarjoajan palvelun kautta? Esimerkkinä voisi olla tilanne, jossa käyttäjä aloittaa uuden Reddit-keskustelun. Jokaista käyttäjää tulee palveluntarjoajan suojella tahalliselta häirinnältä mutta samaan aikaan emme halua, että jokin taho voi mielivaltaisesti poistaa aloittamamme keskustelut tai lähettämämme kommentit. Kysymys datan omistajuudesta nousee myös tilanteissa, joissa jokin sähköinen palvelu kerää jotain telemetriaa käyttäjistään, ja esimerkiksi

myy tätä dataa eteenpäin. Määritelmä siitä, mikä lasketaan yksityiseksi dataksi, tuntuu nykyään olevan kovin epämääräinen, eikä se ole mielestäni kovin hyvä asia. Tämä käy ilmi esimerkiksi Euroopan Komission selvityksestä siitä, mitkä tiedot ovat yksityistä dataa ja mitkä taas eivät. Kyseisessä selvityksessä todetaan, että henkilötietoihin lasketaan muiden muassa nimi ja sukunimi, kotiosoite, sähköpostiosoite, IP-osoite, evästetunnus ja paikannustiedot. Henkilötietoihin selvityksessä ei puolestaan lasketa anonymisoituja tietoja. Selvitys ei kuitenkaan ota kantaa siihen, mitä kaikkea tämä anonymisoitu data voi todella sisältää. (Euroopan Komission www-sivut 2018, Paz 2014)

Adam Scottin kirjassa oleva esimerkki tutkimuksesta, jossa anonyymistä hakukoneen datasta voitiin ennakoida haimasyövän diagnoosi, kertoo hyvin siitä, että myös se data, jota emme suoranaisesti miellä tärkeäksi, on todellisuudessa erittäin arvokasta tietyille tahoille. Nimimerkki Wibson kertoo artikkelissaan “How much is >Your< data worth? At least $240 per year. Likely much more” tutkimuksestaan, jossa hän arvioi Yhdysvaltojen digitaalisen mainosteollisuuden tuottavan vuositasolla keskimäärin $240 per henkilö käyttäen yksityistä dataa mainosteollisuuden hyväksi. Suurimmat osapuolet, jotka hyötyvät sekä anonymisoidusta että henkilökohtaisesta datasta ovat siis seuraavat tahot: Mainostajat, jotka pyrkivät luomaan datan avulla kohdennettuja mainoksia. Mainostajien lisäksi suurimpiin tahoihin kuuluvat datanvälittäjät, kuten Cambridge Analytica. Kyseinen yritys käytti saamaansa ja käsittelemäänsä dataa vaikuttaakseen vuonna 2016 amerikkalaisiin äänestäjiin, osittain myös mainostajien kautta. Dataa haluavat myös paikalliset varkaat ja hakkerit, pääsääntöisesti rahan takia. Lisäksi dataa käyttävät suuret mediat, pankki- ja muita maksupalveluita tarjoavat yritykset, suuret datan kerääjät kuten Google, sekä valtiot. Vaikka perusteena useasti näiden tahojen datan keräämiseen on esimerkiksi turvallisuuden varmistaminen kaikille osapuolille, on nämä asiat erittäin suuria haasteita yksityisyyden ja tietosuojan kannalta. Herää kysymys, miten voimme suojella kaikkialle hajautettua dataamme ja yrittää sen ohella säilyttää lähes olemattoman yksityisyytemme? (Wibson 2018, Lomas 2018, Kissel 2017)

3 TIETOTURVA JA YKSITYISYYS TEXT JA VOICE CHAT -SOVELLUKSISSA

3.1 Tietoturva ja yksityisyys

Nykyisin erilaisille tietokoneille ja mobiililaitteille on olemassa lukuisia erilaisia text ja voice chat -sovelluksia. Yksi suosituimmista sovelluksista PC:lle on esimerkiksi Skype, joka tarjoaa tavallisen chättäilyn lisäksi video ja äänipuheluita, sekä muita esimerkiksi yrityksille suunnattuja palveluita. Mobiililaitteilla ehkä suurimpana tekijänä tällä hetkellä on Whatsapp, jonka kautta voi lähettää tekstimuodossa olevia viestejä, äänipuheluita ja videopuheluita. Kumpaakin, sekä Skypeä että Whatsappia voi kuitenkin käyttää niin tietokoneelta kuin mobiililaitteesta. Tietoturvan merkitys tällaisissa sovelluksissa on yhä tärkeämmässä asemassa, sillä niiden kautta jaamme suuria määriä arkaluontoista dataa, joka voi olla esimerkiksi henkilökohtaisiin, poliittisiin tai työasioihin liittyvää. Uskoisin, että harva ihminen todella haluaa että hänen erilaiset yksityiselämäänsä koskevat asiat joutuvat vääriin käsiin. (Dickson 2017)

Tutkiessamme ja arvioidessamme erilaisten chat-sovellusten tietoturvan tasoa, on hyvä kiinnittää huomio seuraaviin asioihin: Salausmenetelmät, metadatan keräys, mahdollisuus viestien poistoon sekä onko kyseessä avoimen lähdekoodin sovellus. Salauksen arvioinnissa on hyvä tietää, käyttääkö sovellus jotain suosittua ja hyvin testattua protokollaa salaukseensa. Tästä hyvä esimerkki on Telegram, jota on kritisoitu siitä, että se käyttää omatekoista salausprotokollaa viestien salaukseen. Tämä protokolla on nimeltään MTProto, eikä sitä ole pystytty kunnolla testaamaan todellisen turvallisuuden mittaamiseksi. Mielestäni tämä tosiasia vaikuttaa jossain määrin huonolla tavalla kuluttajan ja palveluntarjoajan väliseen luottamukseen. Lisäksi salauksesta on hyvä tietää, onko yhteydet salattuja vain palvelimen ja clientin välissä, vai käyttääkö se end-to-end-encrytpionia, kuten Whatsapp. Tämä itsessään on hyvä asia, mutta ottaen huomioon, että Whatsapp on Facebookin omistuksessa, itse en pysty täysin uskoutumaan Whatsapille. Vaihtaisin sovellustani mielelläni esimerkiksi Signaliin, jos sillä olisi runsaammin käyttäjiä ystäväpiiristäni. (Techworld Staff 2019, Wen 2018)

E2E-salaus varmistaa, että viestit ovat salattuja clientiltä clientille, ja tällöin ei palvelin pysty viestien sisältöä lukemaan. Chat-sovelluksen tietoturvan kannalta on myös

tärkeää, että sovellus kerää minimaalisen vähän metadataa. Tietoturvan pääidea on huolehtia muun muassa yksityisyyden säilymisestä, jolloin minimaalinen metadatan keräys on tärkeää. Näiden lisäksi on hyvin tärkeää, että käyttäjät voivat halutessaan todella poistaa viestinsä sovelluksesta ja palvelimelta. Mielestäni luottamus on tärkeää kaiken sähköisen viestinnän kanssa, ja avoimen lähdekoodin sovellukset ovat esimerkkejä siitä, miten luottamusta voidaan kasvattaa läpinäkyvyyden kautta. Tällöin käyttäjä voi luottaa esimerkiksi viestiensä poistoon myös oman havainnointinsa kautta, eikä luottamus perustu pelkästään palveluntuottajan lupaukseen viestien poistosta. Pohtimalla näitä seikkoja sovelluksen tietoturvan arvioinnissa antaa suhteellisen kattavan kuvan siitä, mitä tietoturva todella merkitsee palveluntarjoajalle. (Dickson 2017)

Tietoturva yksinään ei takaa hyvää yksityisyyttä, ja tämä sama ajatus pätee kaikkiin text ja voice chat -sovelluksiin. Tuntuu että vasta viime aikoina erilaisten sovellusten yksityisyyteen liittyvien skandaalien seurauksena ihmiset ovat alkaneet heräilemään hiljalleen yksityisyyden merkityksen todelliseen ymmärtämiseen ja arvostamiseen. Esimerkiksi Facebookiin liittyvien skandaalien seurauksena satoja tuhansia suomalaisia on lähtenyt Facebookista. Konkreettinen liikehdintä esimerkiksi käyttäjämäärien laskussa on omanlaisensa viesti myös palveluntarjoajille, ja se on mielestäni hyvä asia. (Harmanen 2018)

Koska tietoturva ja yksityisyys kulkevat käsi kädessä, voimme sovelluksen yksityisyyden ja tietosuojan tasoa arvioidessamme tutkia pitkälti samoja pääkohtia kuin tietoturvan kanssa. Nämä kohdat olivat salausten käyttö, onko sovelluksen lähdekoodi avoin, millä tavalla viestien poisto onnistuu ja miten paljon sovellus kerää metadataa. Salauksen kohdalla yksityisyyttä voimme miettiä siltä kantilta, onko sovelluksessa salattu pelkkä palvelimen ja clientin välinen yhteys vai onko salaus toteutettu aina clientiltä clientille. Tämä on erittäin tärkeä kysymys, sillä jos käytämme sovellusta joka on omisteinen ja sen lähdekoodi ei ole avoin, emme voi koskaan olla täysin kartalla siitä, mitä dataa tämä sovellus meistä kerää ja mihin se sitä käyttää. Vaikka sovellus siis tarjoaisi hyvän tietoturvan vahvojen salauksien avulla, voi yksityisyytemme olla kuitenkin paljon huonommassa tilassa. Tähän samaan kastiin kuuluu myös mainittu metadata: Koska emme voi esimerkin sovelluksen lähdekoodia sen tarkemmin tutkia, on tietomme meistä kerätystä metadatasta pelkästään palveluntarjoajan sanojen varassa. Hyväeleisimmätkään pyrkimykset läpinäkyvyyteen eivät välttämättä anna

tyydyttävää vastausta yksityisyyttä koskeviin kysymyksiin. Hyvänä esimerkkinä toimii NSA:n antama raportti sen keräämästä metadatasta vuonna 2017: NSA taltioi vuonna 2016 noin 150 miljoonaa puhelua. Vuonna 2017 tallenteiden määrä oli 530 miljoonaa, vaikka NSA:lla oli kohteita edelliseen vuoteen verrattuna vähemmän. Mitä enemmän käyttäjän dataa on levinnyt eri osapuolille, olivatpa ne sitten valtion tai joitain mainontaa tai muita tutkimuksia tekeviä yrityksiä, sitä suurempi on riski sille, että data joutuu vääriin käsiin. (Coldewey 2018, Paz 2014)

Viestien poistamisen mahdollisuus ja myös muut omaan dataan liittyvät hallintaoikeudet ovat tärkeitä yksityisyyttä miettiessämme. Poistoon, salauksen tasoon ja metadatan keräämisen kartoittamiseen kaikkiin vaikuttaa se tosiasia, onko sovellus lähdekoodiltaan avoin. Mielestäni luottamus on yksi tärkeimpiä elementtejä text ja voice chat -sovelluksissa, ja avoin lähdekoodi on yksi tehokas tapa tuoda datan hallintaan liittyvät seikat myös hyvin kuluttajan saataville. Se, että konkreettisesti saamme tutkia sitä, mihin jokin sovellus käyttää dataamme ja millä tavalla, on elintärkeää, ja tämä väylä on mielestäni ainoa sellainen, jossa tietoturva ja yksityisyys voivat todella olla sillä tasolla kuin niiden kuuluisi joka paikassa ja tilanteessa olla.

3.1.1 Discord

Discord on avoimen lähdekoodin kirjastoihin perustuva omisteinen text ja voice chat -sovellus sekä , joka on täysin ilmainen käyttää. Discordin suosio on viime vuosina kasvanut hurjasti, ja tällä hetkellä Discordilla on yli 90 miljoonaa rekisteröitynyttä käyttäjää. Discord on räätälöity erityisesti pelaajille, sillä se on helppokäyttöinen ja viestit liikkuvat minimaalisella viiveellä. Nykyään Discord toimii myös julkaisualustana ja heiltä löytyy oma pelikauppa. (Melcon 2018, Discordin www-sivut 2018)

Discordin mukaan hyvä tietoturva ja viestinnän luottamuksellisuus ovat yksiä heidän pääprioriteettejaan. Discordin viestit kulkevatkin HTTPS:n avulla suojattuna, mutta heidän virallisilta sivuiltaan ei kuitenkaan löydy sen tarkempaa selvitystä heidän käyttämistään salausmekanismeista. Heidän sivuillaan ei kerrota esimerkiksi käyttääkö Discord end-to-end -salausta. Discord kuitenkin lupaa pitää käyttäjät turvassa väärinkäytöksiltä, häviöiltä, luvattomilta pääsyiltä, tiedonannolta, tuhoamisilta ja

muunnoksilta. Tämän lisäksi Discord pyrkii jatkuvasti parantamaan tietoturvaansa omien Discord Security Bug Bountien kautta. Tässä bug bounty -ohjelmassa Discord kertoo, ettei se ryhdy lakitoimiin sellaisten haavoittuvuuksien paljastamisesta, jotka on toteutettu heidän bug bounty -sääntöjensä mukaan. He sanovat myös aloittavansa toimet haavoittuvuuksien selvittämiseksi yleensä 24 tunnin sisällä raportin saamisesta. Tässä kohtaa ollaan siis paikalla, jossa loppukäyttäjän tulee pitkälti vain luottaa siihen mitä palveluntarjoaja kertoo sovelluksestaan ja sen tietoturvan takaamisesta. Mielestäni pelkät lupaukset eivät riitä, vaan tilalla pitäisi olla konkreettista näyttöä asiasta. (Discordin www-sivut 2018)

Yksityisyyden osalta Discordilla on mielestäni parantamisen varaa. Discordin Privacy Policy toteaa, ettei se myy käyttäjiensä dataa. Kuitenkin saman toteamuksen jälkeen Privacy Policy jatkaa, että tietyissä tilanteissa Discord voi jakaa dataamme “tiettyjen kolmansien osapuolien kanssa”. Discord kertoo myös kyseisessä toimintaperiaatteissaan erittäin ympäripyöreästi siitä, mitä dataa heidän sovellus todella käyttäjistä kerää. Lisäksi kolmansaien osapuolien määritelmät jäävät koko Privacy Policyssä kovin epämääräiseksi, ja näistä syistä Discord on myös ollut kritiikin kohteena käyttäjiensä datan jakamisen suhteen. Safyre Lyons esittää artikkelissaan “Discord might sell data, but for other reasons. Possibly.” laskelmia, joiden mukaan Discordilla ei ole mitään tarvetta datan myyntiin, sillä Nitro tuottaa tällä hetkellä enemmän kuin tarpeeksi tuloja yhtiölle. Nitron lisäksi Discord saa tuloja sijoittajilta, ja myös tätä seikkaa Discord käyttää puolustautuessaan näitä syytöksiä vastaan. Kohu ja epäselvyys Discordin tietosuojasta ja yksityisyydestä ovat herättäneet myös kilpailijoissa ivaamisen aihetta. TeamSpeak mainitsee myyntikampanjassaan nimeltä Skypen ja Discordin, syyttäessään näitä datan jakamisesta kolmansille osapuolille. TeamSpeak itse toteaa lähes kaikessa mainonnassaan, että he eivät myy eivätkä jaa käyttäjiensä dataa kenellekään. (Lyons 2018, Discordin www-sivut 2018, Honorof 2018, Bailey 2018)

Mielestäni pelkästään kritiikin määrä ja kaikki hämmennys Discordin datan keräämisen ja jakamisen suhteen pitäisi soittaa hälytyskelloja. Jos yrityksellä olisi puhtaat jauhot pussissaan, eikö silloin Discordin tietosuojakäytäntö kertoisi ilman epäselvyyksiä siitä, mitä dataa he keräävät ja miten he sitä jakavat eteenpäin? Omisteisuus ei itsessään ole huono asia, mutta se yhdistettynä ympäripyöreään tietosuojakäytäntöön herättää mielestäni liikaa hämmennystä. Luottamus on mielestäni

yksi ydinasia sähköisessä viestinnässä, ja sen puuttuminen tai rakoileminen kertoo mielestäni siitä, ettei kaikki ole kohdillaan. Lopulta kuitenkin täytyy todeta, että Discordin kohdalla ei ole toistaiseksi olemassa riittävän pitäviä todisteita datan myymisestä eikä sen myymättä olemisesta. Tämä on ehkä huonoin asetelma missä text ja voice chat -sovellus voi olla. (Honorof 2018, Discordin www-sivut 2018)

3.1.2 TeamSpeak

Teamspeak on kruununsa suosituimpana voip-sovelluksena Discordille luovuttanut, pelaajille tarkoitettu text ja voice chat -sovellus, joka on viime aikoina yrittänyt uudelleenbrändätä itseään Discordin kovan suosion tähden. Teamspeakin ominaisuuksien kirjo on suhteellisen laaja, ja sieltä löytyy esimerkiksi in-game-overlay, hyvä skaalautuvuus, mobiilisovellus sekä paljon muita hyödyllisiä ominaisuuksia. Teamspeak kuitenkin mainostaa itseään erityisesti turvalliseksi ja yksityiseksi vaihtoehdoksi, ja se on suurelta osin myös syy, miksi edelleen jotkut pelaajat nimenomaan haluavat pitäytyä Teamspeakissa Discordin sijaan. Vaikka itse luotankin TeamSpeakiin enemmän kuin Discordiin yksityisyyden ja tietoturvan kohdalla, TeamSpeak suljetun lähdekoodin sovelluksena ei ansaitse täyttä luottamustani. (Winkie 2018, TeamSpeakin www-sivut 2018)

Tietoturvan osalta Teamspeak on hoitanut asiat hyvin, sillä se salaa kaikki yhteydet oletuksena. Kyseessä on vahva salaus palvelimen ja clienttien välisessä liikenteessä. Lisäksi Teamspeak ei myy mainoksia ja vakuuttaa ettei se kerää käyttäjistään dataa eikä myy sitä eteenpäin kolmansille osapuolille. On hyvä kuitenkin ottaa huomioon, että koska kyseessä on omisteinen, suljetun lähdekoodin sovellus, ovat käyttäjät jälleen lähinnä palveluntarjoajan lupausten varassa. Koska Teamspeakilla ei ole mainosten ja muun myydyn datan kautta tuloja, on Teamspeak maksullinen. Se ei kuitenkaan pakota käyttäjiä heidän keskitetyille palvelimilleen, vaan palvelinta on myös mahdollista pitää hostattuna jossain muualla. Samalla se tarjoaa käyttäjilleen mahdollisuuden hostata palvelimia ei-kaupallisessa käytössä itsenäisesti omalla tietokoneella tietyin rajoituksin. Tämä antaa loppukäyttäjälle enemmän mahdollisuuksia vaikuttaa oman viestintänsä yksityisyyden ja turvallisuuden varmistamiseen. (Bailey 2018, TeamSpeakin www-sivut 2018)

Teamspeak toteaa tietosuojaselosteessaan hyvin selkeästi sen, mihin ja miten he käyttävät käyttäjiensä dataa. Tällä tavalla tulisi mielestäni tietosuojaselosteet aina luoda. TeamSpeak on myös eritellyt hyvin selkeästi selosteessaan ne kolmannet osapuolet, joille se joutuu lain tai palveluidensa ylläpitämisen tähden jakamaan käyttäjiensä dataa. Mielestäni tässä on hyvä esimerkki siitä, miten tietosuojaseloste rakentaa luottamusta sen hajottamisen sijaan. Toisaalta huolimatta siitä, miten joka käänteessä TeamSpeak kertoo ettei se jaa dataa muiden kolmansien osapuolien kanssa, on loppukäyttäjä kuitenkin vain palveluntarjoajan sanojen varassa. Mielestäni todellinen luottamus palveluntarjoajan ja käyttäjän välillä voidaan saavuttaa vain avoimen lähdekoodin avulla, sillä muut ratkaisut jättävät pelkkien sanojen varaan. (Bailey 2018, TeamSpeakin www-sivut 2018)

3.1.3 Mumble

Mumble puolestaan on avoimen lähdekoodin sovellus, joka Teamspeakin tapaan on salannut clientin ja palvelimen välisen tietoliikenteen. Se käyttää myös oletuksena public/private-key -autentikaatiota. Mumble on räätälöity ominaisuuksiltaan vastaamaan pelaajien tarvetta. Mumblella on ainoastaan vaihtoehtona käyttää heidän keskitettyjä palvelimiaan, minkä vuoksi Mumble on maksullinen. Palvelin on kuitenkin mahdollista ladata ja asentaa itselleen ilmaiseksi muutamaksi päiväksi. Mumble keskittyy tietoturvan takaamiseen sekä yksityisyyden ja luottamuksen rakentamiseen muiden muassa läpinäkyvyyden avulla. (Mumblen www-sivut N.d.)

Mumblen tietosuojakäytäntö on mielestäni yksiselitteinen ja selkeää luettavaa, sillä siinä kerrotaan selkeästi miten ja mitä dataa käyttäjistä kerätään, sekä mitä varten sitä kerätään. Suurin osa käytettävästä datasta menee Mumblen oman mainonnan kohdentamiseen, sekä palvelun ylläpitämisen kannalta välttämättömille kolmansille osapuolille. Kaikenkaikkiaan datan käyttö on hyvin rajattu ja selitetty selkeästi. Käyttäjän kannalta avoin lähdekoodi on erittäin hyvä, sillä kuka tahansa käyttäjä voi halutessaan tutkia koodia tarkemmin ja nähdä, seuraako Mumble todella omaa tietosuojakäytäntöään. Tämä on hyvä asia niin käyttäjälle kuin palveluntarjoajalle, sillä

mielestäni tätä kautta voidaan saavuttaa osapuolten todellinen keskinäinen luottamus. (LightSpeedGaming, LLC:n www-sivut 2018, Mumblen www-sivut N.d.)

3.1.4 Ventrilo

Ventrilo on myös tarkoitettu nimenomaan pelaajien käyttöön ja se on ikään kuin Teamspeakin ja Mumblen välimalli: Ventrilo on omisteinen, suljetun lähdekoodin sovellus, maksullinen ja palvelimet ovat keskitettyjä. Ventrilo kertoo sivuillaan heidän panostavansa datakeskustensa kaistanleveyden ja uptimen maksimointiin sekä tietoturvan varmistamiseen. Ventrilo pitää oletuksena clientin ja palvelimen välisen yhteyden salattuna. (Ventrilon www-sivut N.d.)

Ventrilon kohdalla kokonaiskuvaa yksityisyyden suojan todellisesta tilasta on hankala kartoittaa, sillä heidän palvelimiaan on hostattuna useassa eri maassa useissa eri yrityksissä. Lisäksi Ventrilon omilla sivuilla ei ole minkään näköistä tietosuojaselostetta. Myöskään esimerkiksi Suomen alueella palvelimia vuokraavan Gamelaunchin sivuilta ei löydy tietosuojaselostetta. Nämä seikat syövät luottamusta Ventrilon käytön turvallisuuden suhteen, koska kyseessä on kuitenkin mielestäni perusasioista, jotka tulisi ensimmäisenä selvittää kuluttajalle. (Ventrilon www-sivut N.d, Gamelaunchin www-sivut N.d.)

3.2 Haasteet

Tietoturvan kannalta katsoen voip-sovelluksilla on suhteellisen laaja kirjo erilaisia haasteita. Näitä ovat muiden muassa salakuuntelu, minkä takia yhteyksien salaus on erittäin tärkeää. Myös virukset ja erilaiset haittaohjelmat ovat asioita, joiden riskiä ei voi vähätellä. Text ja voice chat -sovellukset eivät ole yhtään sen paremmin turvassa näiltä uhilta kuin mikään muukaan internetin kanssa toimiva sovellus. Lisäksi tällaisten sovellusten uhkina ovat Denial Of Service -hyökkäykset, joissa hyökkääjä pyrkii kuluttamaan sovelluksen kaistanleveydestä niin paljon kuin mahdollista, tai muuten ylikuormittamalla palvelinta kuluttamalla sen resursseja. Myös man-in-the-middle -hyökkäykset ovat erittäin otollisia voip-sovelluksille. Myös tätä hyökkäystä vastaan on sovelluskehittäjän tärkeä implementoida hyvät salausmekaniikat. Edellä

mainittujen uhkien lisäksi haasteina ovat perinteiset identiteettivarkaudet ja erilaiset huijaukset, joiden ehkäisyssä tärkeintä on sovelluksen käyttäjien ohjeistaminen. Yllä mainittujen uhkien ehkäiseminen on mielestäni suhteellisen helppoa, sillä noudattamalla hyviä käytäntöjä ja periaatteita on useat itsestäänselvät uhat torjuttu. (Unuth 2018, Vatanen 2017, Goerzen & Rhodes 2014)

Yksityisyyden ja tietosuojan kannalta katsoen haasteina on datan jakamisen hallinnan hankaluus. Suurten datamäärien jakaminen eri osapuolille, olipa data sitten henkilötietoihin kuuluvaa tai anonyymiä metadataa, tuo aina omat riskinsä. Mitä suuremmalla joukolla dataa on hallussa, sitä suurempi on riski että data vuotaa joskus jotain kautta ulos. Lisäksi uskon, että monella text ja voice chat -sovelluksen omaavalla yrityksellä on haasteena pitäytyä erossa liiketoiminnasta, jossa käyttäjien dataa myydään tarkoituksenmukaisesti eteenpäin. Web-maailmassa data on valuuttaa, joten syy halulle myydä dataa eteenpäin on aika selkeä. Meidän ei kuitenkaan tule unohtaa niitä perusperiaatteita ja eettisiä toimintatapoja, jotka toimivat myös virtuaalisen elämämme ulkopuolella. Yksityisyys on asia, joka kuuluisi olla jokaisen sovelluskehittäjän yksi tärkeimmistä arvoista, ja yksityisyyden progressiivinen laiminlyönti on mielestäni huolestuttavaa. Erilaisten chat-sovellusten jokapäiväinen ja joka paikassa tapahtuva käyttö on luonut niistä erityislaatuisia “kultakaivoksia” datan tuotannon suhteen, ja pitääksemme kiinni jäljellä olevasta virtuaalisesta yksityisyydestämme, meidän tulee todella päästä eroon ajatuksesta “Ei minulla ole mitään salattavaa”. (Clark 2016, Scott 2016)

4 SPEECH BUBBLE YLEISESTI

4.1 Mikä on Speech Bubble?

Työni konkreettisena osuutena on kehittämäni avoimen lähdekoodin text ja voice chat -sovellus nimeltään Speech Bubble. Client ja server ovat vapaasti tutkittavissa ja käytettävissä repositoryissä: https://gitlab.com/Talkhunat/speech-bubble-client ja https://gitlab.com/Talkhunat/speech-bubble-server. Speech Bubblen kehitys lähti halusta luoda edellä mainittuihin ongelmiin jokin hyvä ratkaisu. SB pyrkii

yhdistämään työssä esitettyjen chat -sovellusten ominaisuuksista helppokäyttöisyyden, yksityisyyden ja tietoturvan. Speech Bubblen ydinajatus voidaankin tiivistää kirjaimilla SSP: Simplicity, Security & Privacy.

Speech Bubble on siis text ja voice chat -sovellus, jossa ei ole mitään ylimääräistä. Helppokäyttöisyys toteutetaan yksinkertaisella käyttöönotolla ja helppolukuisella käyttöliittymällä. Hyvä yksityisyys nimenomaan sovelluksen osalta saadaan toteutettua siten, että palvelin tallentaa käyttäjästä vain yhteyden muodostamiseen ja ylläpitoon tarvittavaa välttämätöntä dataa. Tällaista dataa on esimerkiksi käyttäjän IP-osoite, mutta sitäkään palvelin ei tallenna tietokantaan. On kuitenkin hyvä tiedostaa, että käyttäjä on itse vastuussa yksityisyydestään ja tietoturvastaan siinä määrin, missä hän päättää Speech Bubblen palvelinta hostata. Kaikki hostauspalveluita tarjoavat tahot eivät välttämättä kunnioita yksityisyyttä tai tietoturvaa toivotulla tavalla. Valinnanvapauden nimissä Speech Bubblen palvelinta on mahdollista hostata joko lokaalisti omalla tietokoneella tai etänä jonkin kolmannen osapuolen palvelimilla. Hyvä tietoturva SB:ssä saadaan käyttämällä vahvaa salausta ja Pythonin ssl-kirjaston tarjoamaa oletuskontekstia, itse-allekirjoitettuja sertifikaatteja tai vaihtoehtoisesti jonkin Certificate Authorityn, kuten Godaddyn tarjoamia sertifikaatteja. Näiden lisäksi sovelluksessa tehdyt ratkaisut pyrkivät minimoimaan logiikkavirheistä tai muista huonoista käytännöistä johtuvia tietoturvariskejä.

4.2 Rakenne yleisesti

Speech Bubble ei käytä mitään valmista frameworkkiä ratkaisuun, sillä halusin työtä tehdessäni päästä myös todella oppimaan asioita ruohonjuuritasolta alkaen. Yleisesti ottaen frameworkkien käyttö on viisasta ja jopa suositeltavaa esimerkiksi bugien ja muutoinkin ylläpidon takia, mutta työn tekeminen ilman kokonaisia frameworkkejä on minusta erittäin hyvä tapa oppia asioita tavallista syvällisemmin.

Speech Bubble on tehty kokonaan Pythonilla, tarkemmin Pythonin versiolla 3.6 sillä mielestäni Python on erittäin vahva ja selkeä kieli erityisesti sovelluksiin, joissa ollaan internetin kanssa tekemisissä. Lisäksi Pythonille on saatavilla kirjastoja ja paketteja, joita ei lisensoinnit rajoita työssäni olemalla esimerkiksi maksullisia käyttää. Salaus

on toteutettu Pythonin ssl-moduulilla, joka käyttää vapaasti käytettävissä olevaa OpenSSL-kirjastoa. Salasanojen hashaus palvelimella on toteutettu bcrypt-moduulilla, joka on vapaasti käytettävissä ja asennettavissa PyPi-repositorystä. Voice chat -osuus on tehty hyödyntäen PyPistä vapaasti käytettävää PyAudio-moduulia ja tietokantana palvelin käyttää SQLiteä avoimen lähdekoodin SQLAlchemy-ormin (Object-relational mapper) kautta. Kaikki projektissa käytetyt kirjastot ja moduulit estävät lisenssoinnissaan korkeintaan niiden käytön omisteisissa suljetun lähdekoodin sovelluksissa. (OpenSSL Software Foundationin www-sivut 2018, Python Software Foundationin www-sivut 2018)

4.2.1 Palvelin

Palvelin pohjautuu multithreadingiin, eli samanaikaiset tehtävät suoritetaan eri threadeissa. Jokaiselle palvelimelle asetetaan käynnistyksen yhteydessä jokin salasana. Clientit voivat kirjautua palvelimelle IP-osoitteen, porttinumeron ja palvelimelle asetetun salasanan avulla. Sovelluksessa ei siis ole keskitettyä autentikointia, kuten vaikkapa Discordilla, vaan autentikointi tapahtuu palvelinsovelluskohtaisesti. Koska multithreading tuo mukanaan omia riskejä datan käsittelyn synkronoinnissa ja muiden päällekkäisyyksien välttämisessä, pyrin pitämään eri threadien samojen resurssien käsittelyn mahdollisimman vähäisenä. (Washington 2015)

Palvelimen main-thread kuuntelee jatkuvasti saapuvia yhteyksiä. Saadessaan uuden yhteyden palvelin autentikoi käyttäjän luomassaan erillisessä threadissa. Jos autentikointi onnistuu, kyseiselle clientille luodaan kaksi erillistä threadia: toinen jatkuvasti vastaanottaa viestejä, toinen puolestaan huolehtii viestien taukoamattomasta lähettämisestä. Saadessaan viestin, vastaanottava thread tarkistaa onko viesti tarkoitettu palvelimelle, vai jollekin toiselle clientille. Jos viesti on tarkoitettu palvelimelle, se ohjataan erilliseen threadiin jossa hoidetaan esimerkiksi chat-huoneen vaihto. Lähettävä thread tarkistaa onko viesti palvelimen luoma, kuten esimerkiksi ilmoitus onnistuneessa chat-huoneen vaihdoksesta. Mikäli ei, viesti ohjataan chat-huoneessa oleville muille clienteille. (Zaccone 2015)

Koska kyseessä on multithreaded palvelin, tiedon välittäminen threadilta toiselle on välttämätöntä ja tulee tapahtua turvallisesti. Tähän Speech Bubblen palvelin käyttää Pythonin queue-moduulia. Tämän moduulin sisältämässä luokassa Queue on tarvittavat työkalut threadien lukituksiin ja lukitusten avaamiseen. Näin sovellus pystyy jakamaan threadien välillä dataa turvallisesti. Vaikka useiden threadien kanssa työskentely saattaa välillä tuntua erittäin sekavalta, minulle tämän työn parissa opettelu on ollut todella antoisaa ja opettavaista aikaa. Queueiden käyttö useiden threadien kanssa tuntui aluksi hankalalta, mutta loppua kohden kokonaisuus selkeytyi huomattavasti. Lisäksi koska Speech Bubble on tarkoitettu etupäässä suhteellisen pienelle joukolle clienttejä per palvelin, on threadien hallitseminen ja ylläpito vielä hyvin hoidettavissa. (Python Software Foundationin www-sivut 2018, Nguyen 2018, Agarwal & Baka 2018)

4.2.2 Client

Samalla tavalla kuin palvelin, client on multithreadingiin pohjautuva. Speech Bubblen graafinen käyttöliittymä on tehty Pythonin tkinter-moduulilla. Kyseisen moduulin käyttöä ei lisensointi rajoita kuin korkeintaan suljetun lähdekoodin sovelluksissa. Käytän työssäni tätä moduulia UI:n rakentamiseen sen tähden, että tkinter on cross-platform, joten sen käyttäminen Linuxilla, Windowsilla tai OSX:llä ei tuota ongelmaa. (Nguyen 2018, Moore 2018)

Jokainen client tarvitsee komentokehoteargumenttina uniikin id:n, jonka avulla palvelin identifioi eri clientit. Palvelimelle kirjautumiseen client tarvitsee palvelimen IP-osoitteen, porttinumeron, oman käyttäjätunnuksensa sekä palvelimen salasanan. Yksi client voi siis käyttää useita eri käyttäjänimiä, mutta palvelimelle client näyttäytyy yhtenä ja samana clienttinä tämän uniikin id:n takia. Jos Speech Bubbleen jossain vaiheessa kehitetään roolitukset eri clienteille, bannit sekä muut roolia koskevat rajoitukset asetetaan tämän uniikin id:n avulla.
Käynnistyksen jälkeen tkinterin instanssi pyörii sovelluksen main-threadissa, sillä tkinterin instanssi ei voi toimia muussa kuin main-threadissa. Sovellus avaa käyttäjälle kirjautumisikkunan, ja client jää odottamaan käyttäjän syötteitä. Käyttäjän painaessa kirjautumispainiketta, client lähettää erillisen threadin kautta kirjautumistiedot

palvelimelle, ja jää odottamaan palvelimen vastausta. Jos autentikointi onnistuu, käyttäjälle avautuu koti-ikkuna, jossa on esillä palvelimelta haetut ja saatavilla olevat chat-tihuoneet, sekä tieto siitä, missä huoneessa käyttäjä sillä hetkellä on. Tässä kohtaa UI jää jälleen odottamaan käyttäjän syötteitä. Käyttäjä pystyy graafisen käyttöliittymän avulla vaihtaa chat-huonetta klikkaamalla eri huoneita. Nämä eventit laukaisevat eri threadissa suoritettavia metodeita, jotka ilmoittavat palvelimelle tai hakevat palvelimelta halutut asiat. Samalla tavoin kuin palvelin, client välittää eri threadien välillä dataa queue-moduulin avulla. Koska clientin tulee myös jatkuvasti kuunnella palvelimelta saapuvia viestejä päivittääkseen UI:n elementtejä, main-threadissa on tasaisin väliajoin kutsuttava metodi, joka tarvittaessa päivittää UI:n elementit. Viestit siis lähtevät käyttäjän toimesta, mutta sovellus vastaanottaa viestejä taustalla omaan tahtiinsa. (Nguyen 2018)

5 SPEECH BUBBLE -PALVELIMEN RAKENNE JA TOIMINNALLISUUS SELITETTYNÄ

5.1 Palvelimen rakenne ja toiminnallisuus selitettynä

Tässä osuudessa käydään läpi palvelimen rakenne sekä toiminnallisuus metodien ja toimintalogiikan tasolla edellisiä kappaleita tarkemmin. Esittelen perusteet erilaisten ratkaisujen käytöille ja pyrin antamaan mahdollisimman kattavan ja selkeän kuvan tästä sovelluskokonaisuudesta. Työn loppuvaiheessa esittelen myös parannus- ja jatkokehitysideoita, sillä olen joutunut jättämään joitain erittäin mielenkiintoisia kehitysalueita pois Speech Bubblesta opinnäytetyön laajuuden hallinnoimissyistä.

5.1.1 Palvelimen käynnistys

Palvelin käynnistetään antamalla komentokehotteessa argumenttina palvelimen salasana ja suorittamalla projektikansio, jonka sisällä on __main__.py-tiedosto. Tämä tiedosto suorittuu automaattisesti Pythonin toimesta, jolloin sovellus hakee sertifikaattinsa kansiosta app/certs/ sekä luo ja alustaa app/server.py-moduulissa olevan Server-

luokan instanssin. Tämän instanssin sovellus luo antamalla sille argumenttina palvelimen IP-osoitteen, porttinumeron, samanaikaisesti kuunneltavien yhteyksien määrän, palvelimen salasanan ja sertifikaattitiedoston ssl:ää varten.

Konstruktorissaan Server-luokan instanssi alustaa yhteyksiin käytettävän socketin olemaan TCP-socket, sillä sertifikaattien kanssa toimiessamme haluamme varmistaa, että vastaanottaja vastaanottaa kaiken lähetetyn datan. Koska kyseessä on kuitenkin pelkkä text ja voice chat -sovellus ja minulla ei ole mitään hyvää tietoturvaa erityisempiä vaatimuksia sovelluksen tietoturvaa koskien, Server-luokan konstruktori alustaa context-ominaisuutensa ssl-moduulin create_default_context-metodilla, antaen sille argumenttina parametrina saamansa sertifikaattitiedoston polun, sekä clienttien autentikaation tarkoitetun Purpose-objektin. Kyseinen metodi lataa järjestelmän luotetut CA-sertifikaatit, ottaa käyttöön hostnamen tarkistuksen ja sertifikaattien validoinnin sekä valitsee kunnollisen ja turvallisen protokollan sekä salauksen asetukset. Tämän jälkeen konstruktorissa load_cert_chain-metodi lataa sille parametrinä annetun stringin avulla sertifikaattitiedoston .pem formaatissa, ja tämän tiedoston tulee sisältää sekä yksityinen avain että itse sertifikaatti. Kysenen tiedosto tulee säilyttää huolella ja varoa sen joutumista vääriin käsiin. Nämä alustukset ovat välttämättömimmät osat, jotta sovellus voi luoda salatun TCP-yhteyden. Alustuksen jälkeen __main__.py-moduuli kutsuu Server-luokan instanssin metodia run, joka alkaa odottaa saapuvia yhteyksiä. (Python Software Foundationin www-sivut 2018, Goerzen & Rhodes 2014)

5.1.2 Alustus, yhteyksien luonti ja autentikointi

Server-luokan metodi run (Kuva 1.) suorittuu main-threadissa odottaen jatkuvasti saapuvia yhteyksiä. Socket-moduulin metodi accept vastaanottaa ei-salatun yhteyden ja on niin sanotusti “blokkaava”. Main-threadin suoritus siis pysähtyy tuon metodin kohdalle, kunnes se vastaanottaa uuden yhteyden. Saadessaan uuden yhteyden sovellus “wrappaa” tämän tavallisen socketin palvelimen alustaman kontekstiobjektin avulla, luoden siitä ssl-socketin. Tämän wrapin jälkeen kaikki datan välitys tapahtuu juuri luodun ssl-socketin kautta, jolloin esimerkiksi kirjautumistietoja on turvallista välittää tämän salatun yhteyden avulla. Speech Bubblessa ei tällä hetkellä ole mahdollista käyttää kuin salattuja yhteyksiä. SSL-yhteyden luonnin jälkeen sovellus luo

models/models.py-moduulin Client-luokan instanssin, johon se tallentaa saadun yhteyden ja ssl-socketin. Tämä instanssi välitetään eteenpäin autentikaation hoitavalle threadille, jotta threadista pystytään aikanaan ilmoittamaan kyseiselle clientille, onnistuiko autentikaatio, vai ei. Tämän jälkeen run-metodi palaa odottamaan uusia yhteyksiä.

```
def run(self):
    loop = 0
    while True:
        try:
            if loop < 1:
                print("Waiting for connections..")
            connection, address = self.sock.accept()
            ssl_socket = self.context.wrap_socket(connection, server_side=True)
            cl = Client(ssl_socket, address)
            authenticate_user_thread = threading.Thread(target=self.receive_credentials, args=(cl,))
            authenticate_user_thread.start()
        except Exception as e:
            tb = traceback.format_exc()
            print("Exception in main thread!")
            print(str(e) + " " + str(tb))
        loop += 1
```

Kuva 1. Uusien yhteyksien vastaanottaminen run-metodissa.

Onnistuneen yhteyden vastaanoton jälkeen sovellus kutsuu receive_credentials-metodia, joka vastaanottaa salasanan autentikointia varten. Palvelin vastaanottaa clientiltä niin kauan dataa, kunnes se vastaanottaa neljän nullbyten listan. Tämä kertoo palvelimelle, että viesti on vastaanotettu kokonaisuudessaan, jonka jälkeen palvelin kutsuu serve_client-metodia, antaen sille argumenttina viestistä puretut tarvittavat tiedot. Tässä kohtaa palvelin voi tutkia viestin sisällön, sillä kyseessä ei voi olla clientiltä clientille tarkoitettu viesti. Kerron lisää kappaleessa 7 viestien rakenteesta ja kehittämästäni “protokollasta”, johon viestien lähetys Speech Bubblessa nojaa.

Serve_client-metodi ohjaa palvelimelle tarkoitetut viestit oikeisiin käsittelypolkuihin. Palvelimelle tarkoitetut viestit ovat ainoita, joiden sisällön palvelin tarkistaa. Tällaisia viestejä ovat esimerkiksi chat-huoneen vaihtopyyntö tai huoneen luontipyyntö. Autentikoinnin yhteydessä clientiltä tullut viestin sisältö on “authenticate”, jolloin serve_client kutsuu aluksi authenticate-metodia. Kyseinen metodi (Kuva 2) tarkistaa bcryptin avulla, että clientin antama salasana täsmää palvelimen hashattuun salasanaan. Salasanan ollessa oikein, tarkistetaan clientin uniikin id:n avulla, löytyykö clienttiä tietokannasta. Jos clienttiä ei löydy, sinne luodaan uusi, muussa tapauksessa do_after_pw_success ei tee tietokantaan muutoksia. Speech Bubble käyttää SQLiteä,

sillä se on helppokäyttöinen ja oivallinen pienikokkoisiin sovelluksiin. SQLiteä ei kutienkaan suositella käytettäväksi client/server applikaatioissa tai muuten sovelluksissa joissa on korkea samanaikaisuus. Tämä johtuu siitä, että yleensä erilaisten tiedostojärjestelmien performanssi ei ole kovin hyvä ja odottamattomien virheiden riski kasvaa, jos samanaikaisesti useampi thread pyrkii kirjoittamaan tietokantaan jotain. Tämän ongelman ratkaisemiseksi käytän tietokantaoperaatioiden turvana threading lockeja. Tällä tavoin varmistetaan, ettei kaksi threadia voi samaan aikaan tehdä muutoksia tietokantaan. (Zaccone 2015, SQLiten www-sivut N.d)

```
def authenticate(self, name, extra, client):
    pw_recvd = extra[0]
    if self.check_password(pw_recvd):
        self.auth_lock.acquire()
        session = Session()
        try:
            self.do_after_pw_success(name, extra, session)
            session.commit()
        except Exception as e:
            tb = traceback.format_exc()
            print("Errormsg: " + str(e) + ", " + str(tb))
            session.rollback()
        finally:
            session.close()
        self.auth_lock.release()
        return True
    else:
        return False
```

Kuva 2. Clientin autentikointi

Yllä olevassa kuvassa (Kuva 2) metodi palauttaa totuusarvon, riippuen siitä, täsmääkö clientin ja palvelimen salasanat. Paluuarvon ollessa tosi, Client-luokan instanssi lisätään palvelimen clientlistaan ja clientin chat-huoneeksi asetetaan mainlobby. Tämän jälkeen kyseiselle clientille käynnistetään kaksi erillistä threadia: toisessa suorittuu recv_messages-metodi, joka jatkuvasti vastaanottaa viestejä, ja toisessa suorittuu metodi send_messages, joka puolestaan lähettää niitä. Nämä threadit välittävät viestejä keskenään queuen kautta.

5.1.3 Viestien lähettäminen ja vastaanottaminen

Onnistuneen autentikoinnin jälkeen clientille käynnistetään kaksi threadia, joista toisessa suorittuu metodi recv_messages ja toisessa send_messages. Recv_messages odottaa jatkuvasti uutta dataa clientiltä. Metodi ohjaa viestit muille clienteille lähetettävien viestien queueen. Jos viesti on tarkoitettu palvelimelle, viesti välitetään metodille serve_client, jolle käynnistetään recv_messages-metodissa uusi thread.
Send_messages-metodi hakee toistuvasti lähetettävien viestien queuesta viestejä. Queue on blocking, eli suoritus jää odottamaan, että queueen lisätään jotain. Saadessaan viestin, se voidaan ohjata kolmeen eri paikkaan: Se voidaan ohjata clientille itselleen, jos kyseessä on palvelimen luoma vastausviesti johonkin clientin pyyntöön. Tavallisesti viestit välitetään siihen huoneeseen, joka on tallennettuna clientin room-ominaisuuteen. Viesti voidaan kuitenkin ohjata myös johonkin tiettyyn chathuoneeseen. Tätä tarvitaan tilanteissa, joissa client saapuu tai lähtee jostain chat-huoneesta, sillä client vaihtaa room-ominaisuutensa arvon ennen kuin se ilmoittaa kyseiselle lähtöhuoneelle poistumisestaan.

5.1.4 SSL-sertifikaatti

Tavallisesti webbisovelluksille luodaan sertifikaatti ja sertifikaatin allekirjoituspyyntö käyttäen esimerkiksi OpenSSL:n komentokehoteohjelmaa. Tämä allekirjoituspyyntö lähetetään sen jälkeen CA:lle, eli Certifcation Authoritylle, joita ovat muiden muassa GoDaddy, Digicert, Verisign ja Entrust. Allekirjoituksen jälkeen CA palauttaa allekirjoitetun sertifikaatin, jolloin esimerkiksi selaimissa webbisovelluksen url-kentässä näkyy vihreä lukko. Itse-allekirjoitettu sertifikaatti ei ole vähemmän turvallinen kuin CA:n allekirjoittama, mutta itse-allekirjoitetut sertifikaatit eivät ole yleisesti luotettuja selaimissa. Tällöin tällaiselle sivulle mentäessä selain kertoo, ettei sivuston sertifikaatti ole luotettu. (Krause 2018)

Kansiossa valmiina oleva esimerkkisertifikaatti on itse-allekirjoitettu, käyttäen OpenSSL-komentokehoteohjelmaa. Sertifikaatti luotiin käyttäen seuraavia komentoja: “openssl req -x509 -newkey rsa:4096 -nodes -keyout privatekey.key -out certificate.crt -days 365”. Kyseessä on siis useampi komento ketjutettu yhteen. Komento -x509 luo

itse-allekirjoitetun sertifikaatin sertifikaatin allekirjoituspyynnön sijaan. Komento -newkey argumenteilla rsa:4096 luo uuden 4096-bittisen yksityisen avaimen käyttäen RSA-algoritmiä. Komento -nodes määrittää, ettei yksityistä avainta salata millään salasanalla, sillä työssäni se on irrelevanttia. Seuraavaksi -keyout ottaa argumenttinaan yksityisen avaimen nimen, jonka tulee olla formaatissa .key tai .pem. Komento -out puolestaan määrittää tuotetun sertifikaatin nimen, ja tämän sertifikaatin tulee olla joko .crt tai .pem -formaatissa. Lopuksi komento -days määrittää sertifikaatin voimassaoloajan. Yleensä sertifikaattien voimassaoloaika tulisi olla suhteellisen lyhyt riskien välttämiseksi, mutta tässä työni vaiheessa sillä ei ole merkitystä. Certificate.crt annetaan client-sovellukselle, kun taas palvelimelle luodaan sertifikaatin ja yksityisen avaimen sisältävä yksi tiedosto .pem formaatissa. Tämä yhdistäminen onnistuu (Komento toimii ainakin Ubuntulla) esimerkiksi seuraavalla komennolla: cat certificate.crt privatekey.key > certrificate.pem. Tämän luodun certificate.pem-tiedoston palvelimen tulee säilyttää siten, ettei se missään kohden voi päätyä vääriin käsiin. Tästä Speech Bubble pitää toimintalogiikallaan huolen. Näillä komennoilla saadaan luotua helposti vahva sertifikaatti-yksityinen avain -yhdistelmä. (Aggarwal 2018, Davies 2011)

6 SPEECH BUBBLE -CLIENTIN RAKENNE JA TOIMINNALLISUUS SELITETTYNÄ

6.1 Clientin rakenne ja toiminnallisuus selitettynä

Tässä osiossa esittelen clientin rakennetta ja toiminnallisuutta metodien tasolla. Tarkoituksenani on tuoda mahdollisimman kattava kuva clientistä kokonaisuutena, poislukien käyttöliittymä, sekä perustella tekemiäni ratkaisuja. Osiossa käyn läpi sovelluksen aina käynnistyksestä ja alustuksista itse suoritukseen asti. Jatkokehitysideat sekä parannusehdotukset esittelen työssäni myöhemmin luvussa 8.2.

6.1.1 Clientin käynnistys

Client käynnistetään suorittamalla projektin juurikansio ja antamalla samalla komentokehotteessa clientille uniikki id. Tällöin suorittuu juurikansiossa oleva __main__.py-moduuli. Kyseinen moduuli luo client.py-moduulin Client-luokan instanssin ja antaa sille argumenttina komentokehotteesta saadun uniikin id:n, sekä palvelimen sertifikaatin. Tämän jälkeen sovellus luo gui/gui.py-moduulista MainWindow-luokan instanssin, joka on tkinter-moduulin Tk-luokasta periytetty luokka. Tkinter on graafisen käyttöliittymän luova moduuli, joka on cross-platform Windowsin, OS X:n ja Linuxin välillä. MainWindowille annetaan konstruktoriin argumentteina aiemmin luotu Client-luokasta tehty objekti, kyseisen objektin sisältämän window_update_queue-ominaisuus, sekä lähinnä käyttöliittymän tyylityksiä sisältävä settings.json -tiedosto. Tämän jälkeen MainWindow-luokan metodi update_window asetetaan suorittumaan tietyn ajan jälkeen siitä, kun MainWindow on käynnistynyt metodilla mainloop. Graafisen käyttöliittymän toiminnasta kerron tarkemmin kappaleessa 6.2. (Moore 2018)

6.1.2 Alustus, yhteyksien luonti ja autentikointi

Client alustaa konstruktorissaan queuet lähetettäville ja vastaanotettaville viesteille, audioviesteille ja käyttöliittymää päivittäville viesteille. Tällä hetkellä käytännössä kaikki vastaanotettavat tekstimuotoiset viestit päivittävät käyttöliittymää, esimerkiksi tulostaen viestin sisällön sille tarkoitettuun teksti-ikkunaan. Konstruktori alustaa myös Audiohandler-luokan instanssin, joka hoitaa audion valmistelun, nauhoittamisen ja soittamisen. Lisäksi konstruktorissa valmistellaan purpose ja context-ominaisuudet. Context-ominaisuus saa arvokseen palvelimen tavoin ssl-moduulin metodilla create_default_context tehdyn objektin. Kyseiselle oletuskontekstin luovalle metodille annetaan argumenttina ssl-moduulin Purpose-objekti, sekä palvelimen sertifikaattitiedosto. Tämä tiedosto sisältää pelkän sertifikaatin. Tässä vaiheessa Client-luokan instanssi jää odottamaan käyttäjän syötteitä hiirenklikkauksia.

Käyttöliittymässä käyttäjälle avautuu sovelluksen käynnistyessä ikkuna, johon käyttäjä syöttää palvelimen IP-osoitteen, porttinumeron, nimimerkkinsä ja palvelimen

salasanan. Tämän jälkeen käyttäjän painaessa kirjautumispainiketta, UI kutsuu Client-luokan instanssin metodia connect_to_server (Kuva 3). Tällöin client yrittää yhdistää tavallisen socketinsa parametrinä saamaansa IP-osoitteeseen ja porttiin. Yhdistymisen onnistuttua ilman ongelmia, tämä tavallinen socket “wrapataan” context-ominaisuuden metodilla wrap_socket, samalla tavoin kuin palvelimen kohdalla. Wrap-pauksen jälkeen client käyttää vain tätä juuri luotua ssl-sockettia.

Tämän jälkeen client lähettää authenticate-metodilla käyttäjän salasanan palvelimelle juuri salattua yhteyttä pitkin. Metodi jää odottamaan vastausta, ja palauttaa totuusarvon riippuen siitä, onnistuiko autentikointi vai ei. Autentikoinnin onnistuessa window_update_queueen laitetaan tästä tieto, jotta UI osaa päivittyä seuraavaan tilaan.

```
def connect_to_server(self, host, port, name, server_password):
    #FIXME: The print beneath should not print all the credentials.
    print(host + " " + port + " " + name + " " + server_password)
    self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        self.sock.settimeout(5)
        self.sock.connect((host, int(port)))
        self.sock.settimeout(None)
        self.ssl_sock = self.context.wrap_socket(self.sock, server_hostname=host)
        if self.authenticate(name, server_password):
            self.name = name
            self.window_update_queue.put({"auth_success":True})
            recv_messages_thread = threading.Thread(target=self.recv_messages, daemon=True)
            send_messages_thread = threading.Thread(target=self.send_messages, daemon=True)
            handle_recvd_messages_thread = threading.Thread(target=self.handle_recvd_messages, daemon=True)
            recv_messages_thread.start()
            send_messages_thread.start()
            handle_recvd_messages_thread.start()
            self.set_account_name(self.name)
            self.get_rooms()
        else:
            self.window_update_queue.put({"auth_success":False})
            self.ssl_sock.close()
    except TimeoutError:
        self.window_update_queue.put({"login_error":"Server connection timed out."})
    except OSError:
        self.window_update_queue.put({"login_error":"Unable to connect the server."})
    except ValueError:
        self.window_update_queue.put({"login_error":"Portnumber must be integer."})
    except Exception as e:
        tb = traceback.format_exc()
        print(str(tb))
        self.window_update_queue.put({"login_error":"Unhandled exception."})
        print(e)
        self.ssl_sock.close()
        print("Exiting app, cya!")
        sys.exit()
```

Kuva 3. Clientin yhdistäminen palvelimeen.

6.1.3 Viestien lähettäminen ja vastaanottaminen

Onnistuneen autentikoinnin jälkeen clientille käynnistetään kolme threadia (Kuva 3): yksi vastaanottaa viestejä, toinen puolestaan lähettää niitä taukoamatta. Viestien vastaanottamisen ja käsittelyn helpottamiseksi clientille luodaan vielä kolmas erillinen thread, jossa suorittuu metodi handle_recvd_messages. Tämä metodi huolehtii perustehtävien lisäksi niistä tehtävistä, joissa saattaa mennä tavallista enemmän aikaa. Tällä hetkellä myös audiodata kulkee saman socketin kautta kuin tekstimuodossa olevat viestit. Handle_recvd_messages-metodi vähentää turhia viiveitä viestien vastaanottamisessa, sillä recv_messages_thread vain vastaanottaa viestejä ja laittaa ne samantien message_in_queueen. Tällöin handle_recvd_messages voi rauhassa hakea uusia suoritettavia tehtäviä message_in_queuesta. Mielestäni tämä ratkaisu selkeyttää threadien työnjakoa ja metodien tehtäviä.

Kaikki lähetettävät viestit muunnetaan aluksi dictionary-kokoelmiksi, koska viestien sisältö rakentuu lennosta. Viestin ollessa lähetystä vaille valmis, send_text_messages-metodi (Kuva 4) serialisoi dictionarystä JSON-dokumentin metodilla prep_data, käyttäen serialisointiin Pythonin json-moduulia. (Agarwal & Baka 2018, Rischpater 2015)

```
def send_text_messages(self, message):
    data_dict = self.create_message_dict(self.name, message)
    msg = self.prep_data(data_dict)
    self.message_out_queue.put(msg)
```

Kuva 4. Teksimuodossa olevien viestien lähetys.

Jokainen clientin vastaanottama viesti puretaan osiin handle_recvd_messages-metodissa (Kuva 5). Varsinaisen purkamisen tekee metodi unwrap_data, joka deserialisoi JSON-formaatissa olevan vastaanotetun viestin heti sen jälkeen, kun viestiestä on poistettu ylimääräiset bittijonot, kuten kuvassa 4 näkyy. Kumpaankin, serialisointiin ja deserialisointiin käytetään Pythonin json-moduulia. Unwrap_data palautta viestin kentät tuplena, jonka jälkeen tästä tuplesta luodaan namedtuple, eli tässä tapauksessa Response-niminen tuple. Namedtuplen käyttö mahdollistaa tuple-objektin kenttiin viittaamisen niiden nimen avulla. Tämä selkeyttää tuplen käsittelyä huomattavasti, sillä tavallisen tuplen kohdalla kenttiin viittaminen tapahtuisi pelkän indeksin avulla. Purkamisen jälkeen Response-tuple laitetaan window_update_queueen, jotta clientin

käyttöliittymä saadaan päivitettyä. Audiodata ohjataan puolestaan audio_out_queueen, jonka sisällön käsittelyn hoitaa audiohandler.py-moduulin Audiohandler-luokan instanssi. Client-luokasta tehty objekti huolehtii audiohandlerin instanssin metodien kutsuista. (Rischpater 2015)

```
def handle_recvd_messages(self):
    while True:
        try:
            msg = self.message_in_queue.get()
            if b'[\x1A\x1A\x1A\x1A]' in msg:
                msg_stripped = msg.replace(b'[\x1A\x1A\x1A\x1A]', b'')
                self.audio_out_queue.put(msg_stripped)
            elif b'[\x0B\x0B\x0B\x0B]' in msg:
                msg_stripped = msg.replace(b'[\x0B\x0B\x0B\x0B]', b'')
                name, timestamp, message, extra = self.unwrap_data(msg_stripped.decode('utf-8'))
                self.window_update_queue.put(Response(name, timestamp, message, extra))
            elif b'[\x1B\x1B\x1B\x1B]' in msg:
                msg_stripped = msg.replace(b'[\x1B\x1B\x1B\x1B]', b'')
                name, timestamp, message, extra = self.unwrap_data(msg_stripped.decode('utf-8'))
                self.window_update_queue.put(Response(name, timestamp, message, extra))
            else:
                name, timestamp, message, extra = self.unwrap_data(msg.decode('utf-8'))
                self.window_update_queue.put(Response(name, timestamp, message, extra))
        except Exception:
            tb = traceback.format_exc()
            print(str(tb))
            print("Message received when error occurred: ",msg)
```

Kuva 5. Viestien käsittely vastaanottamisen jälkeen.

6.1.4 Audiodatan käsittely

Speech Bubble käyttää audiodatan käsittelyyn Pythonille tarjolla olevaa PyAudio-moduulia. Tämä moduuli tarjoaa sidokset cross-platform audio I/O kirjasto PortAudiolle. PyAudio mahdollistaa audiodatan nahoittamisen ja soittamisen Linuxilla, Windowsilla ja OS X:llä, ilman mitään lisensointiin liittyviä rajoituksia. Cross-platform, yksinkertaisuus ja selkeys olivat pääsyitä, miksi päädyin käyttämään juuri PyAudiota Speech Bubblen äänidatan käsittelyssä. (Python Software Foundationin www-sivut 2018, Everard & Bradbury 2014)

SB:n moduuli Audiohandler.py on rakennettu PyAudion päälle. Moduuli sisältää yhden luokan Audiohandler, joka alustaa konstruktorissaan audiostreamin widthin, channelit sekä raten. Luokan metodi start_stream luo Pyaudio-luokan instanssin avulla stream-objektin, joka saa Pyaudio-luokan metodissa open parametrinä Audiohandler-luokan metodin audio_callback. Tämän jälkeen stream käynnistetään, jolloin Pyaudio

kutsuu audio_callback-metodia erillisessä threadissa. Audiohandler-luokan metodi start_stream jää puolestaan odottamaan blokaten signal_audiohandler_queueen tulevaa arvoa. Tämän queuen avulla saadaan audiostream lopetettua toisen threadin kautta esimerkiksi silloin, kun käyttäjä haluaa lopettaa puhelun. (Hubert Phamin www-sivut N.d.)

Pyaudio kutsuu saamaansa audio_callback-metodia (Kuva 6) erillisessä threadissa aina silloin, kun se tarvitsee uutta soitettavaa dataa, ja/tai kun sillä on uutta nauhoitettua dataa saatavilla. Parametreinä saaduista muuttujista in_data on se, josta olemme kiinnostuneita. Kyseinen muuttuja sisältää nauhoitettua dataa, joka välitetään pass_audiodata-metodin bittijonolisäysten jälkeen message_out_queuelle, eli jonolle, johon toiselle clientille lähetettävä data laitetaan. Tämän jälkeen vastaanotetun audion jonosta audio_out_queue koetetaan ottaa seuraava soitettavan audiodatan pätkä. Jos dataa löytyy queuesta, se muunnetaan biteiksi ja välitetään returnille, jolloin audio saadaan soitettua. (Hubert Phamin www-sivut N.d.)

Välttääkseni tilanteen, jossa ääni katkeilee tai muuten rätisee, täytyy audio_out_queue olla ei-blokkaava. Tämä johtuu siitä, että Pyaudio kutsuu audio_callbackia, vaikka sillä olisi vain nauhoitettua dataa tarjolla. Tällöin jos audio_out_queue olisi blokkaava, sovellus jäisi tähän jumiin. Lisäksi on erittäin tärkeää pitää huoli siitä, että soitettavan datan “framecount”, eli sampleiden määrä, pysyy oikeassa suhteessa datan soittorateen nähden. Tämän takia audio_out_queuen ollessa tyhjä palautetaan returnille null-byteistä koostuva bittijono eli “hiljaisuutta”, joka varmistaa audiodatan eheyden. Jos hiljaisuutta ei lisättäisi, tämä aiheuttaisi myös katkeilevaa ääntä liian vähäisen datamäärän takia.

```
def audio_callback(self, in_data, frame_count, time_info, status):
    self.pass_audiodata(in_data)
    try:
        data_to_play = self.audio_out_queue.get(block=False)
        bytedata = bytes(data_to_play)
    except queue.Empty:
        bytedata = b'\x00' * (frame_count * 2 * 2)
    return(bytedata, pyaudio.paContinue)
```

Kuva 6. Audio_callback-metodin toiminta.

6.2 Käyttöliittymä

Käyttöliittymän luontiin Speech Bubble käyttää Pythonin tkinter-moduulia, koska se on cross-platform ja toimii Windowsilla, Linuxilla ja OS X:llä. Lisäksi moduulilla on suhteellisen nopeaa ja helppoa luoda graafisia käyttöliittymiä ilman erillistä designer-ohjelmaa. Toimintojensa puolesta tkinter pystyy luomaan kaiken tarvittavan tälle sovellukselle. Käyttöliittymän osalta olen pyrkinyt yksinkertaisuuteen ja helppokäyttöisyyteen. Tyylikkyys ja muutenkin esteettisyys on työssäni ihan toissijainen seikka, sillä tarkoitukseni oli luoda toimiva kokonaisuus ennen muuta. (Moore 2018)

6.2.1 Rakenne, alustus ja käynnistys

Tämä kappale jatkaa kappaleen 6.1.1 kerrontaa clientin käynnistyksestä, mutta keskittyen graafiseen käyttöliittymään. Clientin käynnistävä moduuli __main__.py luo kappaleen 6.1.1 mukaisesti gui/gui.py-moduulin MainWindow-luokan instanssin. Tälle instanssille annetaan argumenttina konstruktoriin client.py-moduulin Client-luokasta tehty objekti ja tämän objektin sisältämän window_update_queuen. MainWindow on periytetty tkinter-moduulin luokasta Tk, ja tämä toimii juurena koko graafiselle käyttöliittymälle. Olen pyrkinyt luomaan gui/gui.py-moduulin siten, että siellä olevat luokat toimivat ikäänkuin komponentteina, jotka sisältävät erilaisia widgettejä. Tk-luokan lisäksi gui/gui.py-moduulin luokista muutamat on periytetty tkinterin luokasta Frame, sillä Frame-luokka toimii widgettien containerina. Tämä komponenttiajattelu luo selkeyttä eri ikkunoiden hallintaan ja auttaa myös uudelleenkäyttämään jo luotuja komponentteja. (Moore 2018)

MainWindowin konstruktorissa luokan ominaisuuksiin alustetaan instanssit niistä luokista, jotka ovat periytetty tkinterin Frame-luokasta. Lisäksi konstruktorissa luodaan handlers/eventhandler.py-moduulin instanssi, jonka päätehtävänä on huolehtia eri containereiden näkyvyystilasta. Moduuli huolehtii esimerkiksi siitä, milloin virheistä kertova ikkuna on näkyvissä ja milloin ei. Tämän lisäksi moduuli hoitaa update_queueen tulevien viestien käsittelystä ja tämän kautta ikkunoiden datan asetuksista. Ikkunoiden näkyvyystiloista ja widgettien sijainneista sovellus huolehtii

tkinterin pack-geometrymanagerilla. Mielestäni kyseisen managerin avulla on helppoa ja yksinkertaista asettaa widgettejä kohdilleen. (Love 2018)

MainWindow-luokan instanssi voi suorittua ainoastaan sovelluksen mainthreadissa, sillä se on periytetty luokasta Tk. Instanssi aloittaa suorituksensa mainloop-metodikutsulla. Tämän jälkeen sovelluksen suoritus siirtyy kyseiseen metodiin, poistuen sieltä vasta silloin, kun ohjelman suoritus lopetetaan. Widgettien sisällön päivitys ei tästä syystä onnistuisi, ellemme asettaisi ennen mainlooppiin siirtymistä MainWindowia kutsumaan metodia after (Kuva 7). After-metodi suorittuu mainlooppiin siirtymisen jälkeen argumenttina saamiensa millisekuntien kuluttua, jolloin se suorittaa sille toisena argumenttina annetun metodin. Tässä tapauksessa se on MainWindow-luokan metodi update_window. Update_window tarkistaa (Kuva 8), löytyykö update_queuesta viestiä. Jos queuesta löytyy viesti, se välitetään eventhandler-moduulin instanssille, joka hoitaa widgettien datan päivityksen. Lopuksi update_window asettaa after-metodin suorittumaan 100 millisekunnin kuluttua, jolloin after kutsuu jälleen update_windowia. Näin update_window suorittuu loopissa. Tällä tavalla käyttöliittymä ei missään kohden “jäädy” ja pystymme muokkaamaan widgettien dataa mainlooppiin siirtymisen jälkeen. (Moore 2018)

```
app = gui.MainWindow(cl, cl.window_update_queue)
app.after(100, app.update_window)
app.mainloop()
```

Kuva 7. Tk-instanssin luonti ja mainlooppiin siirtyminen.

```
def update_window(self):
    try:
        item = self.update_queue.get_nowait()
        if item is not None:
            self.handler.handle_update(item)
    except queue.Empty:
        pass
    self.after(100, self.update_window)
```

Kuva 8. Update_window-metodin toiminta.

6.2.2 Ulkoasu ja toimintaketju

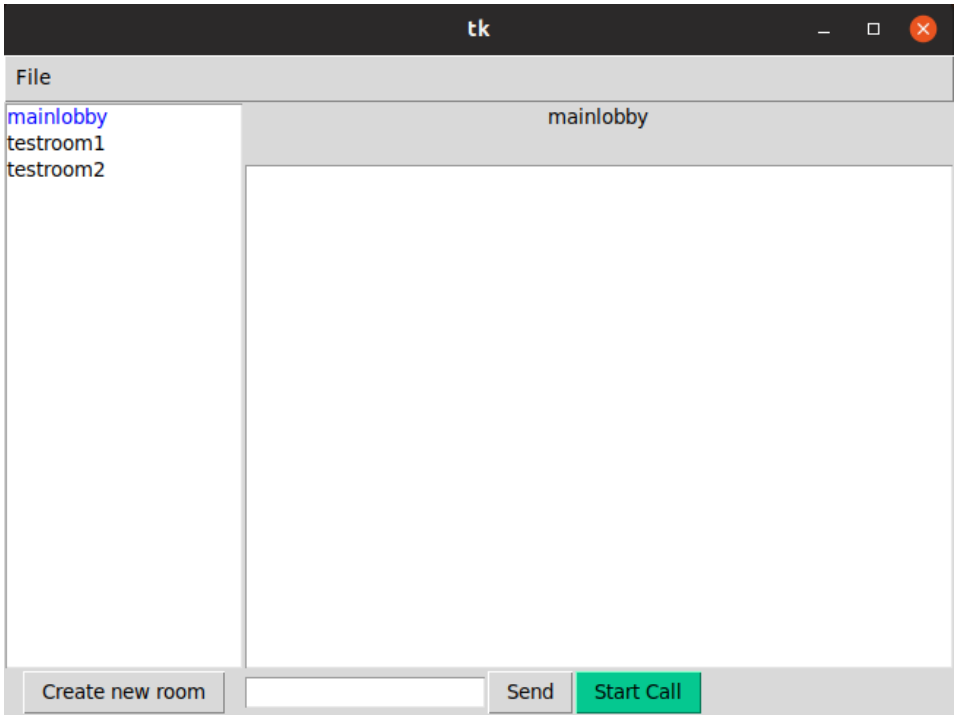
Tässä luvussa esittelen kuvien kera Speech Bubblen käyttöliittymää. Kuvat saattavat erota sovelluksen uusimman version ulkoasusta, sillä olen tehnyt jatkuvasti muutoksia sovellukseen tätä työtä kirjoittaessani. Sovelluksen ulkonäkö ei myöskään ole tämän työn pääprioriteetti, ja siksi lukijan ei tule liikaa kiinnittää huomiota osittain tylsän näköiseen ulkoasuun.

Käynnistyksen jälkeen käyttäjälle avautuu ikkuna (Kuva 9), jossa on syöttökentät palvelimen IP-osoitteelle, porttinumerolle, käyttäjätunnukselle ja palvelimen salasanalle, sekä kirjautumispainike. Kirjautumispainikkeen painalluksen jälkeen sovellus poistaa kirjautumisikkunan näkyvistä, ja näyttää odotustekstin widgettien tilalla, kunnes client saa palvelimelta autentikointiin vastauksen. Jos IP-osoite, porttinumero tai salasana on väärin, client avaa uuden ikkunan kirjautumisikkunan päälle, jossa on jonkinlainen virheviesti. Käyttäjän sulkiessa tämän virheikkunan, client asettaa kirjautumisikkunan näkyville.

Kuva 9. Speech Bubblen kirjautumisikkuna

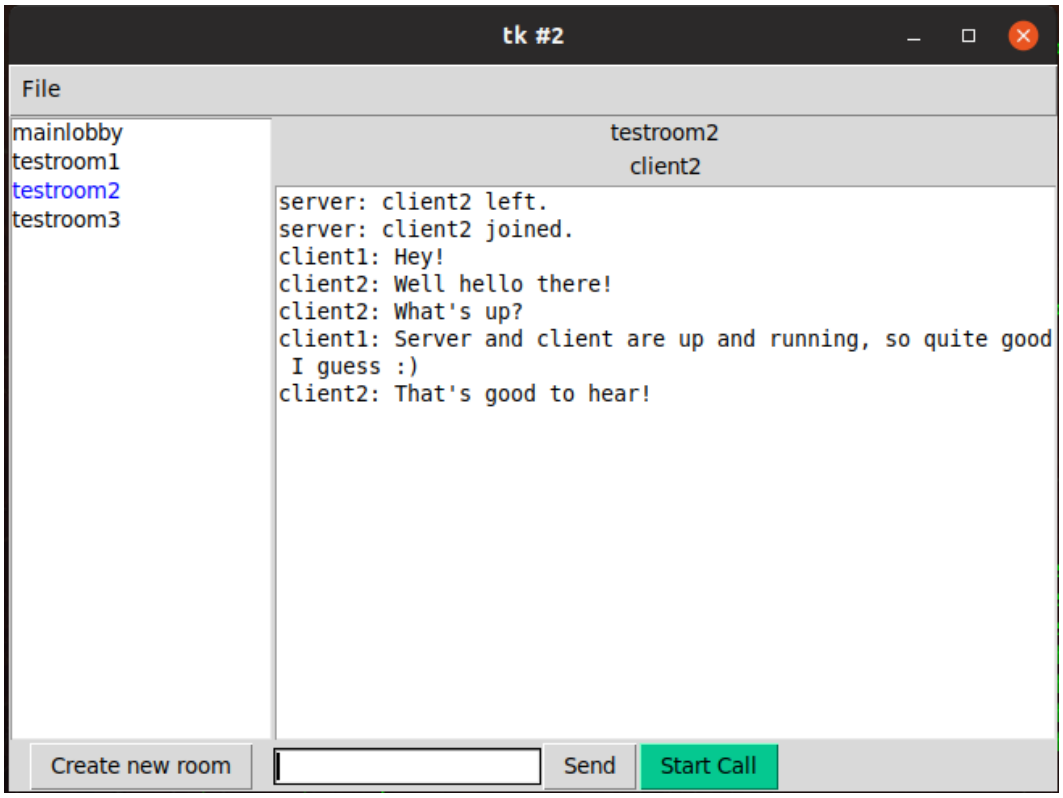
Autentikaation onnistuessa sovellus asettaa koti-ikkunan näkyviin (Kuva 10). Ikkunan vasemmassa reunassa on lista saatavilla olevista huoneista, joista clientin senhetkinen huoneen nimi on eri värinen kuin muiden huoneiden nimet. Käyttäjä voi vaihtaa huonetta kaksoisklikkaamalla toisen huoneen nimeä, jolloin sovellus lähettää palvelimelle huoneenvaihdospyynnön. Huonelistan alapuolella on huoneen luontiin

tarkoitettu painike, joka avaa käyttäjälle erillisen toplevel-ikkunan. Tämän ikkunan kautta sovellus pyytää palvelinta luomaan käyttäjän syötteen nimisen huoneen.



Kuva 10. Speech Bubblen koti-ikkuna.

Koti-ikkunan oikealla puolella on tekstille varattu ikkuna. Tämän ikkunan yläpuolella on clientin senhetkisen chat-huoneen nimi ja huonenimen sekä teksti-ikkunan välissä on lista huoneessa olevien clienttien käyttäjänimistä (Kuva 11), poislukien clientin oma käyttäjänimi. Teksti-ikkunaan client tulostaa kaikki saamansa viestit. Kyseisen ikkunan alapuolella on tekstin syöttökenttä. Kentän oikealla puolella on lähetyspainike, ja heti sen vieressä oikealla on voice chatin aloittamispainike. Tekstimuodossa olevat viestit lähetetään hiirenklikkauksen lisäksi myös käyttäjän painaessa Enteriä. Voice chattiin käyttäjä voi liittyä hiiren klikkauksella painikkeesta “Start call”. Tällöin sovellus pyytää palvelinta liittämään clientin text chat -huonetta vastaavaan voice chat -huoneeseen. Saadessaan tiedon huoneeseen liittämisestä client muuttaa soittopainikkeen värin punaiseksi ja siihen vaihtuu teksti “End call”. Käyttäjä liittyy ja lähtee voice chatista saman painikkeen avulla.



Kuva 11. Esimerkki text chatin toiminnasta.

7 VIESTIPROTOKOLLA

Tässä luvussa tutkin kehittämääni yksinkertaista viestiprotokollaa. Luvussa esittelen viestien rakennetta ja toimintalogiikkaa, sekä niihin liittyviä metodeita. Suurin osa metodeista on toiminnaltaan itsestään selviä, mutta erottelen tähän lukuun niitä metodeja, jotka tarvitsevat kuvia selityksen tueksi.

7.1 Viestien rakenne

Jokainen palvelimen ja clientin luoma viesti on rakenteeltaan samanlainen. Koska viestien sisältö rakentuu lennosta, ovat viestit aluksi dictionary-kokoelmia. Kaikki viestit serialisoidaan tästä dictionary-kokoelmasta JSON-dokumentiksi käyttäen Pythonin json-moduulia. Jokaisesta viestistä löytyy seuraavat ominaisuudet: name, timestamp, message sekä extra. Nimenä on aina lähettäjän nimi, ja tämän ominaisuuden avulla viedään lähettäjän nimitieto helposti eri clienttien käyttöliittymän viestiboksiin.

Jokainen viesti saa joko clientin tai palvelimen toimesta myös aikaleiman, joka on viestin lähetyshetki. Aikaleima on muodossa “pp.kk.vv, hh:mm”. Message-ominaisuus sisältää itse välitettävän viestin. Se voi olla tavallinen clientiltä clientille tarkoitettu viesti, tai palvelimelle osoitettu pyyntö. Extra-ominaisuus on lista, johon tilanteesta riippuen tietyssä järjestyksessä asetetaan arvoja. Esimerkiksi chat-huoneen vaihdoksessa message-ominaisuus sisältää viestin “switchroom” ja extra-listassa ensimmäisenä on kohdehuoneen nimi. Tämä ratkaisu on helppo ja mahdollistaa useiden erilaisten ja monimutkaisempienkin pyyntöjen luonnin rikkomatta tätä yksinkertaista viestirakennetta. (Agarwal & Baka 2018)

7.2 Pyynnöt palvelimelle

Jos clientin lähettämä viesti on tarkoitettu palvelimelle, viestin message-ominaisuuteen asetetaan haluttu pyyntö. Tällä hetkellä Speech Bubblen palvelin tukee seuraavia pyyntöjä: Viesti “getrooms” palauttaa clientille listan saatavilla olevista huoneista, sekä tiedon siitä, missä huoneessa client itse tällä hetkellä on. Viesti “setname” asettaa client-olion nimen haluttuun arvoon. Tämä siis vaikuttaa siihen, millä nimellä muut clientit näkevät toisensa chat-huoneessa. Viesti “switchroom” nimensä mukaisesti vaihtaa clientin huonetta siihen huoneeseen, joka täsmää extra-listan mukana tulleeseen huoneen nimeen. Viesti “createroom” luo extra-listan mukana tulleen stringin nimisen huoneen, tallentaen sen tietokantaan. Viestit “joinvoip” ja “leavevoip” asettavat ja poistavat clientin voice chat -huoneista. Palvelimella on voice chat -huoneita yhtä monta, kuin on tavallisia text chat -huoneita. Huoneet toimivat aina rinnakkain, eli mainloop text chat -huoneessa oleva client yhdistyy halutessaan vain ja ainoastaan mainloop voice chat -huoneeseen. Sama logiikka toimii jokaisessa muussakin tapauksessa. Viimeisimpänä olevaa viestiä “authenticate” käytetään clienttien autentikointiin. Nämä yllä mainitut pyynnöt ovat tällä hetkellä toiminnassa Speech Bubblen palvelimella.

7.3 Toimintalogiikka

Koska sovellus käsittää TCP-yhteyden loputtomana bittistriiminä, täytyy sovelluksella olla tiedossa millainen bittijono merkitsee viestin päättymistä. Viestit chat-

sovelluksissa on lähes poikkeuksetta eri mittaisia, joten sovellus ei voi nojata siihen, että yksi viesti on esimerkiksi 2048 bitin kokoinen. Speech Bubblen viestiprotokolla määrittää, että jokainen viesti päättyy hakasulkeista ja nullbyteistä koostuvaan jonoon: “[\x00\x00\x00\x00]”. Jonossa on hakasulkeiden sisällä useampi nullbyte sen tähden, että juuri tällaisen jonon esiintyminen viestin keskellä teksti- tai audiodatassa on erittäin pieni. Viestin katkeaminen kesken kaiken on ei-toivottu tilanne, ja siksi tuon jonon tulisi olla niin harvinainen kuin mahdollista. Oikeastaan bittijono voisi olla vieläkin huomattavasti pidempi. (Washington 2015)

Erottaakseen palvelimelle tarkoitetut viestit clientiltä clientille tarkoitetuista, client lisää bittijonon viestiinsä kertoakseen palvelimelle, että kyseessä on pyyntöviesti, jonka sisällön se voi tarkistaa. Tämä bittijono mudostuu seuraavista biteistä hakasulkujen sisällä: “[\x0B\x0B\x0B\x0B]”. Kyseinen jono tulee viestin loppuun, kuitenkin ennen nullbytejonoa, sillä nullbytejono on aina viimeisenä. Esimerkkinä on palvelimen recv_messages-metodi (Kuva 12), joka välittää saapuneen viestin joko clientin lähtevien viestien jonoon, tai serve_client-metodille, jolle käynnistetään erillinen thread. Ohjauksen palvelin tekee sen mukaan, löytyykö viestistä tiettyjä bittijonoja vai ei. Tämä on helppo ja yksinkertainen tapa varmistaa käyttäjän yksityisyyden kunnioittaminen.

```
def recv_messages(self, client):
    try:
        while True:
            data = bytearray()
            msg = ''
            while not msg:
                recvd = client.get_connection().recv(1024)
                if not recvd:
                    raise Exception()
                data = data + recvd
                if b'[\x00\x00\x00\x00]' in recvd:
                    msg = data
            if b'[\x0B\x0B\x0B\x0B]' in msg:
                msg_null_stripped = msg.replace(b'[\x00\x00\x00\x00]', b'')
                msg_stripped = msg_null_stripped.replace(b'[\x0B\x0B\x0B\x0B]', b'')
                name, timestamp, message, extra = self.unwrap_data(msg_stripped.decode('utf-8'))
                serve_client_thread = threading.Thread(target=self.serve_client, args=(name, timestamp, message, extra, client), daemon=False)
                serve_client_thread.start()
            else:
                client.get_message_queue().put((msg, None))
    except Exception:
        tb = traceback.format_exc()
        print(str(tb))
        print(recvd)
        self.handle_disconnect(client)
```

Kuva 12. Viestiprotokollan toiminta recv_messages-metodissa.

Samalla tavalla kuin recv_messages-metodi (Kuva 12) ohjaa viestejä spesiaalibittijonojen perusteella, toimii send_messages-metodi (Kuva 13). Jos se löytää lähtevien viestien queuesta saamastaan viestistä bittijonon “[\x0B\x0B\x0B\x0B]”, se ohjaa

viestin takaisin clientille itselleen. Kyseisestä jonosta client tietää, että viesti on palvelimen vastaus clientin tekemään pyyntöön. Muissa tapauksissa metodi ohjaa viestit johonkin huoneeseen, riippuen viestin luonteesta. Speech Bubblen protokollaan kuuluu myös pelkät ilmoitusluontoiset viestit, jotka ovat palvelimen generoimia ja jotka eivät ole vastauksia clientin tekemään suoraan pyyntöön. Näitä ovat esimerkiksi ilmoitukset huoneeseen liittyneistä tai lähtevistä clienteistä. Tällöin palvelin asettaa viestiin ennen nullbytejonoa bittijonon "[\x1B\x1B\x1B\x1B]", jolloin clientit tietävät kyseessä olevan pelkkä ilmoitusviesti.

Edellä mainittujen bittijonojen lisäksi SB:n protokolla sisältää bittijonon "[\x1A\x1A\x1A\x1A]", joka puolestaan tarkoittaa, että kyseessä on audiodataa. Palvelin ohjaa audiodatan erikseen voice chat -huoneissa oleville clienteille. SB:n aiemmassa kehitysvaiheessa palvelin ohjasi audiodatan tavallisten text chat-huoneiden perusteella, jolloin clientit vastaanottivat dataa vaikka heillä ei olisi voice chat ollutkaan päällä. Tämä oli käytäntönä huono, joten muokkasin palvelinta siten, että se lähettää audiodatan erikseen voice chat -huoneiden perusteella. Tällöin clientit eivät joudu vastaanottaa tarpeetonta dataa.

```
def send_messages(self, client):
    while True:
        message = client.get_message_queue().get()
        if message[0] is None:
            break
        elif b'[\x1A\x1A\x1A\x1A]' in message[0]:
            if client.get_voip_room() is not None:
                voip_client_list = client.get_voip_room().get_client_list()
                for cli in voip_client_list:
                    if cli is not client:
                        cli.get_connection().sendall(message[0])
        elif b'[\x0B\x0B\x0B\x0B]'in message[0]:
            client.get_connection().sendall(message[0])
        elif message[1] is None:
            room_client_list = client.get_current_room().get_client_list()
            for cli in room_client_list:
                if cli is not client:
                    cli.get_connection().sendall(message[0])
        else:
            room_client_list = message[1].get_client_list()
            for cli in room_client_list:
                if cli is not client:
                    cli.get_connection().sendall(message[0])
        client.get_message_queue().task_done()
```

Kuva 13. Send_messages-metodin toiminta.

8 LOPUKSI

8.1 Pohdintaa tehdystä työstä

Tämä työ on ollut minulle ymmärrystä avaava ja opettavainen kokemus. Olen oppinut paljon siitä, mitä ratkaisuja tällaiseen sovellukseen kannattaa tehdä ja mitä ei. Työn aloitusvaiheessa minulla ei ollut aiempaa kokemusta sockettien tasolla tapahtuvasta ohjelmoinnista, eikä näiden yhteyksien salaamisesta. Useiden eri threadien kanssa toimimisesta sain myös työssäni hyvää kokemusta, sillä en ole ollut aiemmin näin laajan multithreading-kokonaisuuden kehitysprojektissa mukana. Lisäksi olen päässyt erittäin hyvin kiinni Pythoniin kielenä ja ymmärrän nyt paljon paremmin sen toimintaa myös hieman pintaa syvemmältä.

Olen kohdannut työssäni niin ylä- kuin alamäkiä. Välillä ongelmien ratkaisu on tuntunut sujuvan kitkatta, välillä taas olen ihmetellyt jotain ongelmaa luvattoman kauan vain tajutakseni, että olen katsonut asiaa väärältä kantilta. Kuitenkin on ollut hienoa huomata, kuinka olen selvittänyt jokaisen ongelman. Prosessi on tuntunut etenevän oikeastaan portaittain: välillä tuntuu että en etene minnekään, kun taas yhtäkkiä saan ahaa-elämyksen ja tajuan aiemmin sekavilta tuntuneita asioita. Sockettien välisen kommunikaation virtamaisuuden ymmärtäminen auttoi todella SB:n protokollan kehittämisessä. Useamman threadin kanssa toimiminen tuntuu nykyään erittäin selkeältä ja olen päässyt aika hyvin perille tkinterin käytöstä. Käyttöliittymän tekemiseen en käyttänyt mitään designeria, sillä en tiedä yhtäkään tkinterin designeria. Tämä seikka oikeastaan auttoi paljon tkinterin ymmärtämisessä, sillä se pakotti minut oppimaan tkinterin geometrymanagerien toimintaa hieman syvällisemmin. Näiden lisäksi sertifikaatit ja niiden käyttö tuntuu nykyään huomattavasti selkeämmältä kuin alkuvaiheessa.

Olen myös oppinut työtäni tehdessä paljon audiodatan käsittelystä. PyAudio on oikeastaan todella yksinkertainen moduuli, mutta sen toimintaperiaatteiden ymmärtäminen otti hieman aikaa. Osaltaan tämä saattoi johtua siitä, että PyAudion dokumentaatio oli kovin vähäsananinen. Audiodatan kanssa toimimisen yhteydessä minun tuli myös hieman parannella luomaani viestiprotokollaa, sillä viestien

loppumista merkitsevät bittijonot saattoivat esiintyä audiodatan seassa. Työprosessissa osa asioista tuli korjattua yksinkertaisella yritys-erehdys-taktiikalla.

Olen erittäin tyytyväinen siihen, että työ ei paisunut liian suureksi, sillä siihen olisi ollut erittäin helppo langeta. Speech Bubble tekee sen mitä olin halunnutkin sen tekevän: Käyttö ja ulkoinen olemus on yksinkertainen, salaa yhteydet käyttäen sopivia menetelmiä ja pitää kiinni clienttien välisten viestien yksityisyydestä välttämällä turhia viestien sisällön tutkimisia. Olisin mielelläni halunnut viedä sovelluksen kuitenkin siihen pisteeseen, että se olisi jossain pilvipalvelussa hostattuna konkreettista käyttöä varten. Tällöin clientit kuitenkin vaatisivat jonkinlaisen järkevän mekaniikan uniikin id:n saamiseksi tämän komentokehotteessa annetun argumentin sijaan, sillä identifiointi palvelimella tapahtuu juuri tämän id:n perusteella, ja clientin tulisi aina riippua tässä samassa id:ssä. En kuitenkaan aio jäädä tähän opinnäytetyössä katettuun vaiheeseen Speech Bubble -projektin kanssa, sillä minulla on jo valmiina useampi hyvä jatkokehitysidea, joita haluaisin alkaa soveltaa sovellukseeni. Lisäksi koska Speech Bubble on avoimen lähdekoodin sovellus, kenties joku innokas ohjelmoija haluaa osallistua tämän projektin kehittämiseen.

8.2 Parannus- ja jatkokehitysideoita

Speech Bubble on ruohonjuuritasolta lähtien itse tehty sovellus, jossa oppiminen ja uuden opettelu ovat olleet vahvasti läsnä alusta alkaen. Tämän takia työn tekovaiheessa on tullut vastaan useita jatkokehitysideoita ja ajatuksia siitä, miten jonkin asian voisi tehdä paremmin. Olen opinnäytetyöni ohella ollut mukana oman alan projektissa, joka antoi myös näkökulmaa siitä, miten jokin asia kannattaisi tehdä. Tässä luvussa käyn lyhyesti läpi päällimmäiseksi jääneitä ajatuksia ja ideoita eri aiheista.

8.2.1 Sockettien tyyppi

Tällä hetkellä Speech Bubble käyttää yhtä TCP sockettia audio- ja tekstidatan välitykseen. Tekstityyppisen datan lähettämiseen ja vastaanottamiseen TCP-yhteys on juuri oikea, sillä haluamme varmistaa, että vastaanottaja on vastaanottanut kaiken lähettämämme datan. Lisäksi sovelluksessa käytetty ssl-moduulin wrap_socket-

metodi ei tue kuin TCP-yhteyksiä. Muitakin tapoja salaukselle on, mutta aikataulullisista syistä en ehtinyt perehtyä niihin.

Audiodatan kohdalla TCP-socket ei ole optimaalisin vaihtoehto. Koska audiodata on luonteeltaan jatkuva datan virta, TCP tekee käytännössä osittain turhaa työtä varmistaessaan jatkuvasti, että jokainen lähetetty bitti on saapunut myös perille. Audiodatalle parempi ratkaisu olisi ollut luoda erillinen UDP-socket, sillä UDP on kevyt ja nopea protokolla. Koska UDP on yhteydeton protokolla, se ei varmista, onko vastaanottaja saanut kaikki lähetetyt bitit. Audiodatan kanssa muutamien bittien hävikki ei kuitenkaan vaikuta siihen, pystyykö vastaanottaja ymmärtää viestiä. Testeissäni audioviestien viive TCP-yhteyden yli oli suunnilleen maksimissaan sekunnin luokkaa, mikä on suhteellisen paljon, mutta pääasia on kuitenkin se, että audioviestintä toimii. UDP:n avulla tämä viive olisi mahdollista minimoida. Speech Bubblessa ajan puutteen vuoksi en kutienkaan ehtinyt perehtyä siihen, miten UDP-socketin data olisi ollut mahdollista salata.

8.2.2 Käytetty kieli

Valitsin Pythonin projektini ainoaksi kieleksi, sillä olin ihastunut siihen aiemmissa projekteissani. Lisäksi Python on yleisesti ottaen erittäin hyvä kieli erityisesti palvelinpuolen sovelluksiin. Performanssinsa puolesta Python ei kuitenkaan ole tällaisessa multithreadingiin perustuvassa sovelluksessa välttämättä paras vaihtoehto. Pythonin ominaisuuksiin kuuluva GIL, eli Global Interpreter Lock, estää sen, että useat threadit voivat samanaikaisesti suorittaa natiivia bytecodea. GIL on välttämätön Pythonille Pythonin natiivin implementaation, CPythonin, muistinhallinnan takia. Käytännössä tämä tarkoittaa sitä, että Pythonin threading on rajoitettu yhteen CPU:hun, ja nämä threadit joutuvat vuorottelemaan keskenään GIL:stä.

Speech Bubble on kuitenkin suunniteltu pienille käyttäjämäärille palvelinta kohden eikä sovellus tee missään kohden älyttömästi CPU:ta rasittavaa laskennallista työtä. Suurin performanssin vaatija taitaakin olla PyAudio. Testeissäni huomasin audiosampleiden määrän muuttamisella olevan jonkin verran vaikutusta performanssiin. Tällä hetkellä samplerate on kuitenkin sen verran pieni, että äänenlaatu ei kärsi liian

paljoa, eikä se rasita CPU:ta liikaa. CPU-rasituksen ja skaalan pienuuden tähden tämänhetkinen ratkaisu “Python everywhere” toimii mielestäni kuitenkin erittäin hyvin. Speech Bubble on toteutettu pienille porukoille ja toistaiseksi performanssia ei vaadita niin paljon, että GIL:lä olisi siihen jotain huomattavaa vaikutusta. (Pillai 2017, Palach 2014)

8.2.3 Lisättäviä ominaisuuksia

Speech Bubble tekee sen mihin se pyrki. Sovelluksesta kuitenkin puuttuu tällä hetkellä käyttömukavuutta lisääviä ominaisuuksia, joita tulee mieleen koko ajan lisää. Ajatuksena esimerkiksi olisi kehittää kirjautumisikkunan File-menun Bookmarks-osioon toiminnallisuus, johon käyttäjän tallentamat palvelimen tiedot tallentuisivat. Tämä ominaisuus tulisi nimenomaan vähentämään IP-osoitteiden näppäilemisen määrää. Lisäksi jossain vaiheessa olisi myös hyvä luoda ominaisuus, jolla käyttäjä pystyy poistamaan chat-huoneita. Tällä hetkellä poisto pelkästään poistamalla koko tietokantatiedoston palvelimelta, jolloin kaikki huoneet katoavat. Seuraavan kerran kun palvelin käynnistetään, se luo automaattisesti mainlobby-huoneen.

Jotta huoneiden luonti ja poisto ei olisi yhtä sirkusta käyttäjien kesken, tulisi sovellukseen luoda käyttäjille roolitukset. Tällöin vältytään tilanteelta, jossa jokainen palvelimella oleva clientti voi halutessan luoda niin paljon huoneita kuin he haluavat. Tämän lisäksi roolituksien avulla hoidettaisiin bannaukset tai kickaukset, jos joku palvelimella oleva käyttäjä alkaa käydä häiriöksi. Roolitusten avulla voisi myös pystyä myöhemmin luomaan huoneita, joihin pääsisi vain tietyt clientit. Adminin roolin voisi myöntää vaikka sille clientille, joka ensimmäisenä kirjautuu palvelimelle siten, ettei palvelimen tietokannassa ole vielä muita clienttejä tallennettuna. Muiden clienttien käyttöoikeuksien jako jäisi tämän jälkeen admin-roolin omaavalle clientille.

Muita jatkokehitysideoita on muiden muassa ominaisuus, jolla äänenlaatua ja äänenvoimakkuutta voisi säätää helposti graafisen käyttöliittymän kautta. Sovelluksen nykyisessä versiossa ei ole vielä mahdollista sovelluksen kautta säätää näitä ominaisuuksia. Myöhemmässä vaiheessa Speech Bubblesta voisi tehdä myös web-version ilman erillisiä ladattavia clienttejä. Client-sovelluksen teeman ja muiden värien muuttaminen on

myös käyttömukavuutta lisäävä ominaisuus. Tässä oli muutama esimerkki hyvistä jatkokehitysideoista, jotka jossain vaiheessa toivottavasti ovat osa tätä sovellusta.

8.3 Loppusanat

Kaikenkaikkiaan olen erittäin tyytyväinen tekemiini sovelluksiin. Puutteita ja paranneltavaa on paljon, mutta en alunperinkään tavoitellut täydellisyyttä. Yksi työni tärkeimmistä elementeistä on oppiminen, ja sitä koen myös saaneeni runsaasti. On hienoa huomata kuinka olen täyttänyt ne vaatimukset, joita työlleni alunperin asetin. Oman kehityksen havaitseminen on ollut myös palkitsevaa. Jos nyt aloittaisin työni alusta, tekisin todennäköisesti monta asiaa hyvin eri tavalla. Näen tämän kuitenkin todella hyvänä asiana, sillä tämän perusteella kehitystä on tapahtunut. Toivon erityisesti, että työni voisi toimia opettavana esimerkkinä jollekulle ohjelmointia opiskelevalle, tai muuten innoittaa tätä käymään rohkeasti erilaisten vaikealta tuntuvien projekti-ideoiden kimppuun. Olisi myös todella hienoa, että sovellus saisi joskus jonkun innokkaan ohjelmoijan konkreettisen työpanoksen jonkin ominaisuuden tai jatkokehitysidean osalta.

LÄHTEET

Agarwal, B. & Baka B. 2018. Hands-On Data Structures and Algorithms with Python. Packt Publishing. https://learning.oreilly.com/library/view/hands-on-data-structures/9781788995573/

Aggarwal, A. 2018. Go Web Development Book. Packt Publishing. https://learning.oreilly.com/library/view/go-web-development/9781787286740/

Akred, J. & Samani, A. 2018 Your data is worth more than you think. Viitattu 19.11.2018. https://sloanreview.mit.edu/article/your-data-is-worth-more-than-you-think/

Bailey, D. 2018. TeamSpeak rebrands with a renewed promise to never sell your data (Unlike Discord, they say). Viitattu 27.11.2018. https://www.pcgamesn.com/teamspeak-5

Clark, B. 2016. ”I have nothing to hide” is killing the privacy argument. Viitattu 1.12.2018. https://thenextweb.com/opinion/2016/02/11/i-have-nothing-to-hide-is-killing-the-privacy-argument/#.tnw_7gvrIdG2

Cloudflare Inc. 2018. What is a data breach? Viitattu 19.11.2018. https://www.cloudflare.com/learning/security/what-is-a-data-breach/

Cloudflare Inc. 2018. What is web application security? Viitattu 16.11.2018. https://www.cloudflare.com/learning/security/what-is-web-application-security/

Coldewey, D. 2018. NSA triples metadata collection numbers, sucking up over 500 million call records in 2017. Viitattu 30.11.2018. https://techcrunch.com/2018/05/04/nsa-triples-metadata-collection-numbers-sucking-up-over-500-million-call-records-in-2017/

Davies, J. 2011. Implementing SSL/TLS Using Cryptography and PKI. John Wiley & Sons. https://learning.oreilly.com/library/view/implementing-ssltls-using/9780470920411/

Dickson, B. 2017. How secure is your favourite messaging app? Viitattu 26.11.2018. https://thenextweb.com/contributors/2017/06/19/secure-messaging-apps-signal-telegram-whatsapp/

Discordin www-sivut N.d. Discord privacy policy. Viitattu 27.11.2018. https://discordapp.com/privacy

Discordin www-sivut N.d. Discord security bug bounty. Viitattu 27.11.2018. https://discordapp.com/security

Discordin www-sivut 2018. Discord terms of service. Viitattu 27.11.2018. https://discordapp.com/terms

Euroopan Komission www-sivut 2018. Mitkä tiedot ovat henkilötietoja? Viitattu 21.11.2018. https://ec.europa.eu/info/law/law-topic/data-protection/reform/what-personal-data_fi

Euroopan parlamentin ja neuvoston direktiivi luonnollisten henkilöiden suojelusta henkilötietojen käsittelyssä sekä näiden tietojen vapaasta liikkuvuudesta ja direktiivin 95/46/EY kumoamisesta, 27.04.2016, (EU) 2016/679, EUVL L 119 4.5.2016, 1

Euroopan parlamentin ja neuvoston direktiivi luonnollisten henkilöiden suojelusta toimivaltaisten viranomaisten suorittamassa henkilötietojen käsittelyssä rikosten ennalta estämistä, tutkimista, paljastamista tai rikoksiin liittyviä syytetoimia tai rikosoikeudellisten seuraamusten täytäntöönpanoa varten sekä näiden tietojen vapaasta liikkuvuudesta ja neuvoston puitepäätöksen 2008/977/YOS kumoamisesta, 27.4.2016, (EU) 2016/680, EUVL L 119 4.5.2016, 89

Everard, B. & Bradbury, A. 2014. Learning Python with Raspberry Pi. John Wiley & Sons. https://learning.oreilly.com/library/view/learning-python-with/9781118717035/

Gamelaunchin www-sivut N.d. Viitattu 1.12.2018. https://www.gamelaunch.eu/

Goerzen, J. & Rhodes, B. 2014. Foundations of Python Network Programming, Third Edition. Apress. https://www.safaribooksonline.com/library/view/foundations-of-python/9781430258551/

Goodin, D. 2017. Serious flaw in WPA2 protocol lets attackers intercept passwords and much more. Viitattu 19.11.2018. https://arstechnica.com/information-technology/2017/10/severe-flaw-in-wpa2-protocol-leaves-wi-fi-traffic-open-to-eavesdropping/

Harmanen, S. 2018. IS: Suomalaisten Facebook-käyttäjien määrä romahti, sanoo asiantuntija – syynä yksityisyysskandaali? Viitattu 22.11.2018. https://yle.fi/uutiset/3-10375513

Henkilötietolaki 22.4.1999/523 muutoksineen.

Hubert Phamin www-sivut, N.d. PyAudio Documentation. Viitattu 12.1.2019. https://people.csail.mit.edu/hubert/pyaudio/docs/#example-callback-mode-audio-i-o

Hubert Phamin www-sivut, N.d. PyAudio. Viitattu 12.1.2019. https://people.csail.mit.edu/hubert/pyaudio/

Honorof, M. 2018. Help me, Tom's guide: Is Discord tracking me? Viitattu 1.12.2018. https://www.tomsguide.com/us/help-me-toms-guide-discord-permissions,review-5104.html

Kissel, J. 2017. Take control of your online privacy, 3rd edition. Take Control Books. https://www.safaribooksonline.com/library/view/take-control-of/9781492020400/

Komulainen, K. & Puopolo, I. 2018. Suurin tietoturvauhka on edelleen ihmisten varomattomuus. Viitattu 16.11.2018. http://uusityo.dna.fi/f-securen-hypponen-suurin-tietoturvauhka-edelleen-ihmisten-varomattomuus/

Krause, J. 2018. Windows Server 2016 Administration Cookbook. Packt Publishing. https://learning.oreilly.com/library/view/windows-server-2016/9781789135930/

Laki sähköisen viestinnän palveluista 7.11.2014/917 muutoksineen.

LightSpeedGaming, LLC:n www-sivut, 2018. GDPR Policy. Viitattu 1.12.2018. https://www.mumble.com/gdpr-policy.php

Lomas, N. 2018. Cambridge Analytica's Nix said it licenced "millions of data points" from Acxiom, Experian, Infogroup to target US voters. Viitattu 21.11.2018. https://techcrunch.com/2018/06/06/cambridge-analyticas-nix-said-it-licensed-millions-of-data-points-from-axciom-experian-infogroup-to-target-us-voters/

Love, D. 2018. Tkinter GUI Programming by Example. Viitattu 12.1.2019. https://learning.oreilly.com/library/view/tkinter-gui-programming/9781788627481/

Lyons, S. 2018. Discord might sell data, but for other reasons. Possibly. Viitattu 1.12.2018. https://medium.com/software-nerds/discord-might-sell-data-but-for-other-reasons-possibly-31aafbaaf684

Mayersen, I. 2018. Meet the latest vulnerability in your multi-threaded CPU: PortSmash. Viitattu 19.11.2018. https://www.techspot.com/news/77240-meet-latest-vulnerability-multi-threaded-cpu-portsmash.html

Melcon, A. 2018. Discord: Everything you need to know. Viitattu 27.11.2018. https://www.tomsguide.com/us/what-is-discord,review-5203.html

Moore, A. D. 2018. Python GUI Programming with Tkinter. Packt Publishing. https://learning.oreilly.com/library/view/python-gui-programming/9781788835886/

Mumblen www-sivut N.d. Viitattu 27.11.2018. https://wiki.mumble.info/wiki/Main_Page

Nguyen, Q. 2018. Mastering Concurrency in Python. Packt Publishing. https://learning.oreilly.com/library/view/mastering-concurrency-in/9781789343052/

OpenSSL Software Foundationin www-sivut, 2018. Welcome To OpenSSL!. Viitattu 2.1.2019. https://www.openssl.org/

Palach, J. 2014. Parallel Programming with Python. Packt Publishing. https://learning.oreilly.com/library/view/parallel-programming-with/9781783288397/

Paz. 2014. With great data comes great responsibility. Viitattu 19.11.2018. https://medium.com/@jazzpazz/with-great-data-comes-great-responsibility-72d3e1c94e27

Pillai, A. B. 2017. Software Architecture with Python. Packt Publishing. https://learning.oreilly.com/library/view/software-architecture-with/9781786468529/

Python Software Foundationin www-sivut 2018. 17.7. queue – A synchronized queue class. Viitattu 2.1.2019. https://docs.python.org/3.6/library/queue.html

Python Software Foundationin www-sivut 2018. 18.1. socket – Low-level networking interface. Viitattu 3.1.2019. https://docs.python.org/3.6/library/socket.html

Python Software Foundationin www-sivut 2018. 18.2. ssl – TLS/SSL wrapper for socket objects. Viitattu 3.1.2019. https://docs.python.org/3.6/library/ssl.html#ssl.create_default_context

Python Software Foundationin www-sivut 2018. PyAudio 0.2.11. Viitattu 2.1.2019. https://pypi.org/project/PyAudio/

Rischpater, R. 2015. JavaScript JSON Cookbook. Packt Publishing. https://learning.oreilly.com/library/view/javascript-json-cookbook/9781785286902/

Russel, B. 2017. We have abandoned every principle of the free and open internet. Viitattu 16.11.2018. https://www.theverge.com/2017/12/19/16792306/fcc-net-neutrality-open-internet-history-free-speech-anonymity

Schecter, E. 2018. A secure web is here to stay. Viitattu 16.11.2018. https://security.googleblog.com/2018/02/a-secure-web-is-here-to-stay.html

Scott, A. D. 2016. Building web apps that respect a user's privacy and security. O'Reilly Media, Inc. https://www.safaribooksonline.com/library/view/building-web-apps/9781492042921/

Scott, A. N.d. Principles of ethical web development. Viitattu 22.11.2018. https://ethicalweb.org/

Speech Bubble Client. 2019. https://gitlab.com/Talkhunat/speech-bubble-client

Speech Bubble Server. 2019. https://gitlab.com/Talkhunat/speech-bubble-server

SQLiten www-sivut, N.d. Appropriate Uses For SQLite. Viitattu 4.1.2019. https://www.sqlite.org/whentouse.html

Suomen perustuslaki 11.6.1999/731 muutoksineen.

Symantec Corporationin www-sivut. n.d. What you need to know about the WPA2 Wi-Fi network vulnerability. Viitattu 19.11.2018. https://us.norton.com/internetsecurity-emerging-threats-what-to-do-about-krack-vulnerability.html

TeamSpeakin www-sivut 2018. Viitattu 27.11.2018. https://teamspeak.com/en/

Techworld Staff. 2019. Best secure mobile messaging apps. Viitattu 22.1.2019. https://www.techworld.com/security/best-secure-mobile-messaging-apps-3629914/

Tietosuojavaltuutetun www-sivut. 2018. Tietosuoja. Viitattu 20.11.2018. https://tietosuoja.fi/tietosuoja

Tietosuojavaltuutetun www-sivut. 2018. Lainsäädäntöä. Viitattu 20.11.2018. https://tietosuoja.fi/lainsaadanto

Unuth, N. 2018. Security threats in VoIP. Viitattu 30.11.2018. https://www.lifewire.com/security-threats-in-voip-3426532

Vatanen, P. 2017. Näin tietomurtautuja huijaa sinua – hyökkäys voi olla kuin agenttielokuvasta. Viitattu 30.11.2018. https://yle.fi/uutiset/3-9488768

Ventrilon www-sivut N.d. Viitattu 27.11.2018. http://www.ventrilo.com/about.php

Viestintäviraston www-sivut. 2018. Kybersää 09/2018. Viitattu 16.11.2018. https://www.viestintavirasto.fi/attachments/tietoturva/Kybersaa_syyskuu_2018_verkkosivut.pdf

Viestintäviraston www-sivut. 2018. Luottamuksellisen viestinnän suoja. Viitattu 19.11.2018. https://www.viestintavirasto.fi/kyberturvallisuus/tietoturvaohjeet/palveluidenturvallinenkaytto/luottamuksellisenviestinnansuoja.html

Viestintäviraston www-sivut. 2018. Palveluiden turvallinen käyttö. Viitattu 16.11.2018. https://www.viestintavirasto.fi/kyberturvallisuus/tietoturvaohjeet/palveluidenturvallinenkaytto.html

Viestintäviraston www-sivut. 2015. [Teema] Yksityisyyden suoja ja sähköinen viestintä – Kuka valvoo ja mitä? Viitattu 20.11.2018. https://www.viestintavirasto.fi/kyberturvallisuus/tietoturvanyt/2015/05/ttn20150520132 2.html

Viestintäviraston www-sivut. 2017. WPA2 protokollan haavoittuvuudet mahdollistavat WiFi-verkkojen salauksen murtamisen. Viitattu 19.11.2018. https://www.viestintavirasto.fi/kyberturvallisuus/haavoittuvuudet/2017/haavoittuvuus-2017-033.html

Washington, S. & Sarker M. O. F. 2015. Learning Python network programming. Packt Publishing. https://learning.oreilly.com/library/view/learning-python-network/9781784396008/

Wen, H. 2018. What is Telegram and is it secure? Viitattu 27.11.2018. https://www.csoonline.com/article/3273344/privacy/what-is-telegram-and-is-it-secure.html

Wibson. 2018. How much is >Your< data worth? At least $240 per year. Likely much more. Viitattu 19.11.2018. https://medium.com/wibson/how-much-is-your-data-worth-at-least-240-per-year-likely-much-more-984e250c2ffa

Winkie, L. 2018. Ventrilo and TeamSpeak aren’t dead: Why diehards refuse to switch to Discord. Viitattu 27.11.2018. https://www.pcgamer.com/ventrilo-and-teamspeak-arent-dead-why-diehards-refuse-to-switch-to-discord/

Zaccone, G. 2015. Python Parallel Programming Cookbook. Packt Publishing. https://learning.oreilly.com/library/view/python-parallel-programming/9781785289583/