Juha Hämäläinen

TWITCH CHATBOTIN OHJELMOINTI

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma

Toukokuu 2016

MAMK
University of Applied Sciences

KUVAILULEHTI

MAMK University of Applied Sciences	**Opinnäytetyön päivämäärä** 26.5.2016
Tekijä(t) Juha Hämäläinen	**Koulutusohjelma ja suuntautuminen** **Tietojenkäsittelyn koulutusohjelma**
Nimeke Twitch chatbotin ohjelmointi	

Tiivistelmä

Opinnäytetyön tarkoituksena oli luoda botti, joka kykenee valvomaan Twitchin tarjoaman suoratoistopalvelun yhteydessä tapahtuvaa keskustelua ja antamaan streamaajalle chatin hallintaa helpottavat työkalut. Samalla oli tarkoitus tutustua siihen, millaisia tekniikoita ja toimintoja vaaditaan IRC-chattia valvovan botin luomiseksi.

Opinnäytetyön teoriaosuus käsitteli IRCn historiaa ja esitteli botin toteutuksessa vaadittuja tekniikoita. Opinnäytetyön aikana toteutettiin botti, joka piti kyetä suoriutumaan yksinkertaisista IRC-chatin valvomiseen liittyvistä toiminnoista, kuten tunnistamaan chatissa esiintyviä komentoja ja sanoja. Botti toteutettiin Javalla ja sen asetukset tallennettiin XML-tiedostoihin. Jotta botti kykenisi suoriutumaan sille asetetuista tehtävistä Twitchin toimintaympäristöstä, sen täytyi kyetä kommunikoimaan Twitchin IRC-palvelimen kanssa sekä Twitchin tarjoamien verkkopalveluiden kanssa.

Tämä edellytti tutustumista Javan verkkoyhteystekniikoihin sisältyviin HTTP- ja socket-luokkiin. Twitchin verkkopalvelut eli web servicet on toteutettu noudattaen REST-arkkitehtuurityylin asettamia rajoitteita ja käyttää käyttäjän tunnistamiseen OAuth-sovelluskehystä, joten opinnäytetyössä perehdytään myös siihen, kuinka Javalla luodaan yhteys RESTful-verkkopalveluun. Opinnäytetyössä selvitettiin myös APIn ja Web APIn käsitteellisiä eroja ja yhtenäisyyksiä. Käytännön osuuden alustuksessa avattiin hieman bottia käsitteenä ja selvitettiin millaisia käyttötarkoituksia boteilla on ollut aikojen saatossa. Bottia testattiin Twitchin omassa toimintaympäristössä testikäyttäjän avustuksella. Botti suoriutui sille asetetuista vaatimuksista oletetulla tavalla.

Asiasanat (avainsanat)
verkko-ohjelmointi, verkkojuttelu, suoratoisto, protokollat, Java, XML, ohjelmointi, olio-ohjelmointi

Sivumäärä 46+2	**Kieli** suomi
Huomautus (huomautukset liitteistä)	
Ohjaavan opettajan nimi Janne Turunen	**Opinnäytetyön toimeksiantaja** -

DESCRIPTION

MAMK University of Applied Sciences	**Date of the bachelor's thesis** 26 May 2016
Author(s) Juha Hämäläinen	**Degree programme and option** Business Information Technology
Name of the bachelor's thesis Programming a Twitch chatbot	

Abstract

The subject of this bachelor's thesis was to program a bot that would be capable of moderating a Twitch livestream platform. Moderation consists of following conversation on the Twitch chat. The other purpose of the bot is to provide tools to make it easier for streamers to edit channel information. During the process of creating the bot I studied technologies that are required to complete the bot.

The theory part of this bachelor's thesis covered the history of IRC and explained the technologies behind the bot. The requirements for the bot were as follows: The bot must be capable of handling simple tasks in IRC like recognizing simple commands and words that may appear in the chat. The Bot was programmed with Java and its settings were saved in XML files. For the bot to be able to handle these tasks in Twitch's chat environment it had to be capable of communicating with Twitch's IRC server and with other web services provided by Twitch.

This required that I had to get familiar with HTTP and socket classes included in Java's web communication technologies. The bachelor's thesis also explained the process of connecting to the RESTful web services with Java, because Twitch had designed its web services to be RESTful. As Twitch uses OAuth framework to recognize user authorization process, also this framework was explained in this bachelor's thesis. Since web services are strongly tied with the term web API I introduced it alongside its differences and similarities between the ordinary API. The practical part of this bachelor's thesis began with an introduction to bots as a term and the purposes bots have had during their existence. The bot created was tested in Twitch's environment with the help of a test user and it managed to complete the expectations that were set to it.

Subject headings, (keywords) web programming, chatting, livestreaming, protocols, Java, XML, programming, object-oriented programming	
Pages 46+2	**Language** **Finnish**
Remarks, notes on appendices	
Tutor Janne Turunen	**Bachelor's thesis assigned by** -

SISÄLTÖ

1 JOHDANTO

Tämän opinnäytetyön aiheena on toteuttaa botti, joka valvoo Twitchin streamchattia. Twitch on internetissä sijaitseva videoalusta ja pelaajayhteisö, joka keskittyy pääasiassa streamattuun eli suoratoistettuun videomateriaaliin. Tämä videomateriaali keskittyy pääasiassa erilaisiin peleihin, mutta myös poikkeuksia löytyy, kuten nahan työstöä tai 3D-mallinnusta esittäviä kanavia. Twitch tarjoaa videoalustan lisäksi mahdollisuuden käydä reaaliaikaista keskustelua muiden katsojien kanssa streamin vieressä näkyvän IRC-chatin välityksellä. Chatissa tapahtuvan keskustelun seuraamiseen ja ylläpitämiseen käytetään pääasiassa katsojien seasta valittuja ylläpitäjiä sekä botteja. Keskustelua seurataan pääasiassa siksi, että keskustelun aiheet pysyisivät asiallisina ja että streamin viihdyttävyys ei kärsisi chatissa esiintyvien epäsopivien kommenttien takia. Chattia on myös vaikea seurata, jos siellä esiintyy suuria määriä turhanpäiväisiä kommentteja.

Tässä opinnäytetyössä toteutettavan botin toiminnalle määriteltyjä vaatimuksia ovat ne, että botin täytyy toimia ilman erillistä palvelinta ja botin toimintaa pitää pystyä ohjaamaan ja mukauttamaan graafisen käyttöliittymän kautta. Botin toiminta keskittyy lähestulkoon täysin verkon ylitse kuljetettavaan ja luettavaan dataan, joten tästä syystä opinnäytetyön luku kaksi keskittyy esittelemään botin toteutukseen tarvittavien tekniikoiden teoriaa. Luku alkaa Javan historian ja sen tarjoamien HTTP- ja socket-verkkoyhteystekniikoiden esittelyllä. Javan jälkeen käsitellään XML-merkintäkieltä ja sen manipuloimiseen käytettävää Javalle tarkoitettu JDOM-arkistoa. Tämän jälkeen luvussa esitellään verkkopalvelujen (web service) toteutuksessa yleisesti käytettävään termiin Web API, josta esitellään sen yhtäläisyyksiä ja eroavaisuuksia käsitteenä tavalliseen APIin. Verkkopalveluiden yhteydessä esitellään Twitchin kanavien muokkaamisessa olennaisiin OAuth-todennuskehykseen ja REST-arkkitehtuurityyliin. Luvun lopuksi esitellään IRCin historiaa ja toimintaperiaatetta.

Opinnäytetyön kolmas luku esittelee käsitteen bottien historiaa ja bottien käyttötapoja nykyaikana. Tämän lisäksi luvussa käydään tarkemmin lävitse botille asetetut toiminnalliset vaatimukset, sekä toteutuksessa tarvitut toimenpiteet. Luku siis käsittelee pääasiassa botin toteutusta ja siinä esiintyneitä ongelmia. Samalla se myös laajentaa ja tarjoaa käytännön esimerkkejä aiemmassa luvuissa esiteltyihin tekniikoihin. Luvun

lopussa esitellään aikaan saadut tulokset. Samalla näytetään myös botin toimintaa oikeassa käyttöympäristössä.

Opinnäytetyön neljäs luku toimii opinnäytetyön päätäntönä. Siinä esitellään botin toteutukseen liittyviä omia kokemuksia ja arvioidaan toteutukseen käytettyjen tekniikoiden toimivuutta tehtävään nähden. Luku arvioi lyhyesti omaa mielipidettä botin onnistumisesta ja käy lävitse toimenpiteitä, joiden avulla bottia voisi kehittää tulevaisuudessa..

2 KÄYTETYT TEKNIIKAT

Opinnäytetyössä käytettävät keskeisimmät tekniikat ovat Java, XML, REST, OAuth ja IRC. XML-luvussa perustellaan syitä siihen, miksi XMLää päädyttiin käyttämään datan tallennusmuotona. XML-luku esittelee myös XMLn käsittelyssä käytettävän JDOMiin ja vertailee sen etuja muihin tarjolla oleviin APIhin. REST-luvussa esitellään lyhyesti mikä REST on ja mihin sitä käytetään, sekä tutustutaan RESTiin vahvasti sidoksissa olevaan termiin API. OAuth-luvussa esitellään Twitchin APIssa käyttäjäoikeuksien valtuuttamiseen käytettävä OAuth-sovelluskehys ja sen toimintaperiaate. IRC-luvussa keskitytään IRCn historiaan, sen toimintaideaan ja siihen miten Twitchin chatti poikkeaa normaalista IRC-chatistä.

Opinnäytetyötä lukiessa tulee huomioida, että RESTin ja APIn yhteydessä viitataan usein käsitteeseen verkkopalvelu (Web Service). Tällä käsitteellä tarkoitetaan sovellusta, joka tarjoaa internetin välityksellä toiselle sovellukselle pääsyn palvelun sisällään pitämiin resursseihin (What are Web Services? 2016). Verkkopalvelu-käsitettä ei siis tule sekoittaa jonkin yrityksen tarjoamana verkossa tapahtuvana palveluna (kuten verkkokauppa) vaan kyseessä on sovellukselta sovellukselle tarjottu pääsy palvelun resursseihin.

2.1 Javan verkkoyhteystekniikat

Java on 1990-luvun puolivälissä kehitetty ohjelmointikieli, joka pohjautuu vahvasti C- ja C++-ohjelmointikieliin (Gosling 2013). Javan kehityksen alussa sille asetettiin viisi tavoitetta: sen tulee käyttää oliopohjaisia metodeja, sama ohjelma täytyy pystyä ajamaan usealla eri käyttöjärjestelmällä, sen täytyy sisältää sisäänrakennettu tuki tietoko-

neverkoille, sillä täytyy pystyä suorittamaan ohjelmia turvallisesti etäyhteyden välityksellä ja sen täytyy olla helppokäyttöinen käyttäen niitä toimintoja, jotka oli havaittu hyviksi muiden oliopohjaisten ohjelmointikielten kanssa (History of Java programming language 2013). Nämä edellä mainitut ominaisuudet ovat mahdollistaneet Javan laajan levinneisyyden. Nykyään sitä käytetään useissa eri toimintaympäristöissä, kuten erilaisissa verkkosovelluksissa, client- ja palvelinpohjaisissa sovelluksissa, roboteissa, peleissä ja mobiilisovelluksissa (Kumar 2015).

Javassa on ensimmäisestä vakaasta versiosta alkaen ollut palvelinten välisiä yhteyksiä varten tarvittavat tekniikat (Kramer 1996). Javan tarjoamat verkkoyhteystekniikoissa käytettävät APIt (Application Programming Interface) toimivat matalalla tasolla (Elkstein 2008b). Matalan tason APIen avulla päästään käsiksi syvemmällä esiintyvään dataan ja käsittelemään niitä suoraan. Tällöin käsiteltävästä datasta saadaan enemmän irti, koska siihen pääsee vapaammin käsiksi ja tietoa ei häviä matkan varrella. Matalan tason APIen haittapuoli on se, että niiden avulla on yleensä hitaampaa ja vaikeampaa toteuttaa haluttu sovellus kuin erillisen sovelluskehyksen avulla.

KUVA 1. Lähetettävän URIn osat

Tässä opinnäytetyössä tullaan käyttämään pääasiassa Javan HTTP- ja socket-luokkia. Javan HTTP-luokka on matalan tason API, jonka avulla pystytään ottamaan yhteys toiselle palvelimelle URIn (Uniform Resource Identifier) avulla ja lukemaan sen palauttama vastaus (Class HttpUrlConnection 2016). HTTP-luokan toiminta perustuu URIn mukana lähetettäviin parametreihin, jotka toimivat samalla periaatteella kuin verkkosivulla olevan lomakkeen (form) sisältämien tietojen lähetys (Oracle 2015). HTTP-luokan perusideana on se, että kaikki sen kautta muodostetut yhteydet ovat yksittäisiä ja ne katkaistaan välittömästi sen jälkeen, kun palvelin antaa sille vastauk-

sen (Class HttpUrlConnection 2016). HTTP-luokan avulla lähetettävä URI sisältää perusosan, joka osoittaa sen palvelimen ja verkkopalvelun, josta tiedot haetaan. Tämän verkko-osoitteen perään liitetään halutut parametrit ”?”-merkin avulla. Parametrien tarkoitus on antaa palvelulle lisätietoja sen suhteen mitä sen halutaan tekevän. Esimerkkinä kuvassa 1 käytetään Twitchin APIn tarjoamaa palvelua, joka listaa kaikki Twitchissä olevat streamit. Parametrien avulla näistä voidaan ottaa 25 ensimmäistä streamia, jotka streamaavat Starcraft-peliä. (Kuva 1.)

Toinen opinnäytetyössä käytettävä verkkoyhteystekniikka, socket-luokka, poikkeaa jo perusidealtaan HTTP-luokasta. Yleensä HTTP-protokollalla muodostetuissa yhteyksissä asiakassovellus muodostaa yhteyden palvelimeen lähettäen sille samalla erillisen pyynnön tai komennon, jonka jälkeen pyyntö käsitellään ja yhteys katkaistaan. Botin ja IRC-palvelimen välinen yhteys kuitenkin poikkeaa tässä kohtaa tavanomaisista yhteyksistä, koska botin toimintaperiaatteen takia yhteys täytyy jättää auki. Tässä tapauksessa botin ja IRC-palvelimen välillä on aktiivinen yhteys, jonka kautta ne viestivät toisillensa. Socket-luokka ei ole myöskään rajoitettu käyttämään ainoastaan HTTP-protokollaa, vaan sitä voi käyttää myös muiden protokollien, kuten tiedostojen siirtoon tarkoitetun FTPn (File Transfer Protocol) kanssa tai sähköpostipalvelinten kanssa kommunikointiin tarkoitetun SMTPn (Simple Mail Transfer Protocol) kanssa (What is a Socket? 2016).

Tätä varten yhteys muodostetaan erillisen Socket-luokan kautta, joka tarvitsee parametreikseen palvelimen osoitteen (ip-osoite tai URI) ja portin, johon yhteys muodostetaan (Vesterholm & Kyppö 2008, 484). Tämän jälkeen botti pystyy lukemaan palvelimen lähettämiä viestejä ja kirjoittamaan omia viestejä palvelimelle. Vesterholm ja Kyppö (2008, 485) käyttävät näistä luku- ja kirjoitusväylistä termiä tietovirta. Botin ja palvelimen välisiä tietovirtoja on kaksi, joista toinen virta on tarkoitettu palvelimen lähettämille viesteille (inputStream) ja toinen botin lähettämille viesteille (outputStream). Tällä tavalla palvelin ja botti kykenevät erottelemaan vastaanotetut ja lähetetyt viestit toisistaan.

Ennen kuin botti kykenee käymään keskustelua Twitchin IRC-palvelimen kanssa, sen täytyy ensin liittyä palvelimella sijaitsevalle kanavalle. Kanavalle liittymistä varten tarvitaan kanavan nimi, käyttäjätunnus ja mahdollisesti salasana, mikäli kanavalle on asetettu sellainen (Davis 1999). Nämä tiedot lähetetään palvelimelle kirjoittamalla ne

lähteville viesteille tarkoitettuun tietovirtaan. Normaalisti IRC-palvelimen käyttäjätunnus olisi vapaavalintainen (kunhan käyttäjätunnus on vapaana), mutta koska botti muodostaa yhteyden Twitchin palvelimelle, käyttäjätunnuksena toimii Twitchiin rekisteröidyn tilin nimi pienellä kirjoitettuna. Salasanana toimii käyttäjätiliin liitetty oAuth-token. Kun nämä tiedot on välitetty, botti on liittynyt onnistuneesti kanavalle. Kanavalle liittymisen jälkeen palvelin lähettää botille listan kanavalla olevista muista käyttäjistä. Palvelin lähettää myös 10 sekunnin välein päivityslistan, joka sisältää listan niistä käyttäjistä, jotka ovat tuona aikana liittyneet tai poistuneet kanavalta. (Twitch IRC 2013.)

Kun botti on muodostanut yhteyden palvelimeen, jäljellä on enää yhteyden ylläpito. Ylläpidolla tarkoitetaan tässä yhteydessä sitä, että botti varmistaa, ettei yhteys palvelimeen pääse katkeamaan. Tämä tapahtuu PING ja PONG -viestien avulla. Twitchin IRC-palvelin lähettää noin viiden minuutin välein viestin, joka on muodossa ”PING tmi.twitch.tv”. Tähän viestiin botin täytyy vastata lähettämällä palvelimelle viesti ”PONG tmi.twitch.tv”. Näillä viesteillä palvelin varmistaa sen, että botti on edelleen paikalla, eikä katkaise niiden välistä yhteyttä. (Twitch IRC 2013.)

2.2 XML

XML (Extensible Markup Language) on tekstipohjainen formaatti, jota käytetään datan kuvaamiseen. Tässä opinnäytetyössä XMLää käytetään botin asetusten tallentamiseen sekä muihin botin kannalta olennaisten tietojen säilyttämiseen. Rakenteeltaan XML muistuttaa HTMLää, jolloin data esitetään erilaisten elementtien eli tagien avulla (Rouse 2014). Sen lisäksi, että data voidaan laittaa elementin aloitus- ja lopetustagin väliin, voidaan se asettaa aloitustagin sisälle attribuuttina (esim. title=”kirja”). Attribuutin arvon tulee olla lainausmerkkien sisällä. (Myer 2005.)

```
<?xml version="1.0" encoding="UTF-8"?>
<kayttajaesim>
    <kayttaja nimi="Mikko" salasana="Mikko123"/>
    <kayttaja>
        <nimi>Mikko</nimi>
        <salasana>Mikko123</salasana>
    </kayttaja>
</kayttajaesim>
```

KUVA 2. Esimerkki eri tavoista esittää tietoa XML-tiedostossa

Koska datan muotoilutyyli XMLssä on vapaa, kaikki data mikä voidaan esittää elementtien avulla, voidaan esittää myös attribuuttien kautta (kuva 2). Sille, milloin käytetään attribuuttia ja milloin elementtiä, ei ole yleistä standardia, mutta nyrkkisääntö on olemassa: elementtejä käytetään moniarvoisen datan kanssa ja attribuuttia yksiarvoisen datan kanssa. Jos datalle voidaan asettaa useampi kuin yksi arvo tai dataa on paljon, se on todennäköisesti parempi laittaa elementtiin, jonka jälkeen sitä voidaan käsitellä helpommin tekstidatana. (McLaughling 2001, 35.)

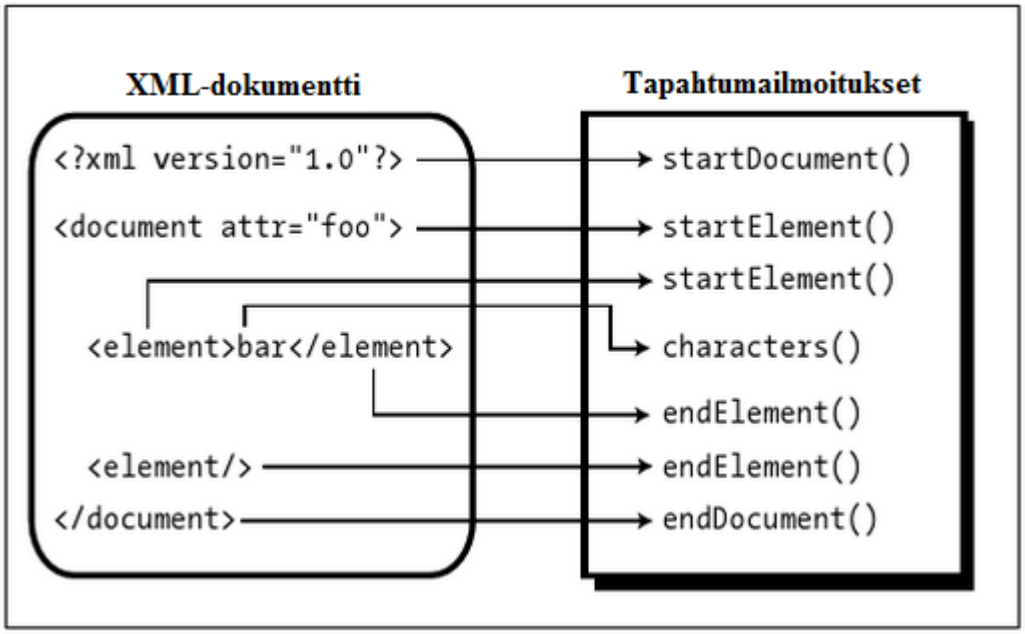
XMLn etuina datan kuvauksessa on sen siirrettävyys, yhteiskäytettävyys ja sen vapaamuotoisuus (Obasanjo 2003). Siirrettävyydellä tarkoitetaan sitä, että koska XML on loppujen lopuksi pelkkää tekstiä, se toimii kaikilla käyttöjärjestelmillä ja on täten alustariippumaton. XMLn yhteiskäytettävyydellä tarkoitetaan sitä, että XML on jokaisen itse vapaasti muotoiltavissa. Sillä ei ole ennalta määriteltyjä tageja, vaan jokainen voi itse kehitellä haluamansa tagit (Myer 2005). Tämä mahdollistaa sen, että organisaatiot voivat itse luoda dokumenttimuodon, sisällön ja datan esittämisessä käytettävät määritelmät (Rouse 2014). Tämän ansiosta eri organisaatiot voivat luoda erilaisia standardeja eri ohjelmien kanssa kommunikointiin ja tiedon siirtoon, mikä mahdollistaa niiden yhteiskäytettävyyden. XMLn vapaamuotoisuudella viitataan pääasiassa siihen, että koska XML on yksinkertaista toteuttaa ja toteutustavat voivat silti olla monimutkaisia, soveltuu XML useisiin erilaisiin projekteihin, mikä säästää aikaa. (McLaughling 2001, 20–21.)

Ennen kuin XMLään tallennettua dataa voidaan käsitellä, tarvitaan erillinen XML parseri (XML Parser 2016). Parserin tehtävä on ottaa XML-dokumentti vastaan raakamuodossa ja muodostaa siitä datarakenne, jota voidaan manipuloida muilla XML-työkaluilla ja Java-rajapinnoilla (Silmaril Consultants 2014). Parserin tehtäviin kuuluu myös varmistaa se, että XML-dokumentti on oikeanmuotoinen ja tarvittaessa myös se, onko XML-dokumentti validi. XML-dokumentin oikeanmuotoisuus on edellytys sen toimivuudelle ja sillä tarkoitetaan sitä, että jokaisella XML-dokumentin alkavalla tagilla on oltava päättävä tagi. XML-dokumentin validoinnilla varmistetaan dokumentin toimivuus. Validointi ei ole välttämätöntä XMLn toiminnan kannalta, mutta sillä varmistetaan se, että XML-dokumentti toimii myös muilla parsereilla ja ohjelmilla, jotka mahdollisesti lukevat XML-dokumenttia. XMLn validoinnissa käytetään erillisiä

DTD- tai skeemadokumentteja, jotka määrittelevät XMLn sisällä olevat elementit, attribuutit ja niiden sisällään pitämien arvojen tyypit. (McLaughling 2001, 25.)

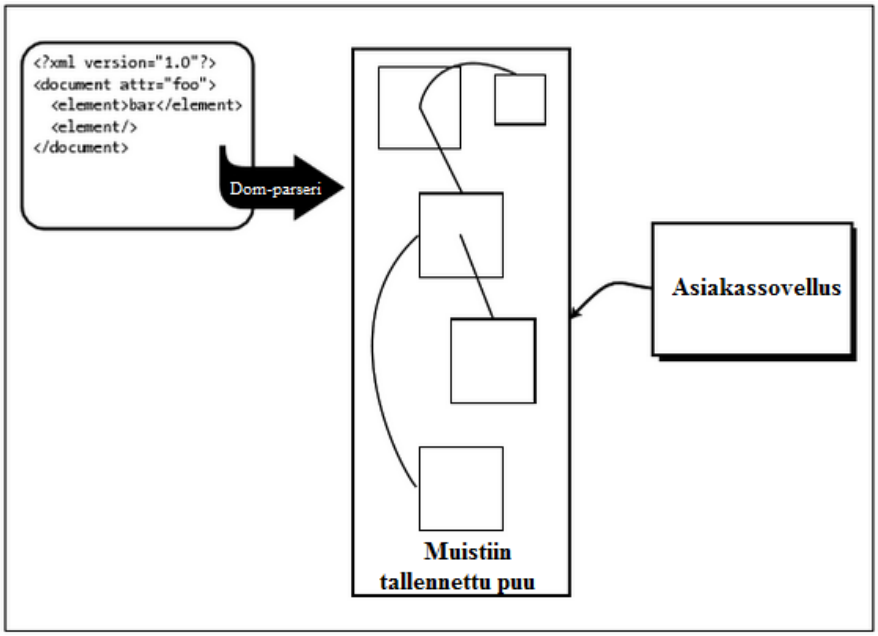
2.3 JDOM

JDOM on avoimeen lähdekoodiin perustuva matalan tason API, jonka avulla voidaan käsitellä suoraan XML-dokumentin sisältöä. Se on yksi kolmesta McLaughlingin (2001, 22–23) mainitsemasta yleisesti käytetyistä matalan tason APIsta, jotka on standardoitu. Kaksi muuta yleisesti käytettyä APIa ovat SAX (Simple API for XML) ja DOM (Document Object Model). Jokainen näistä kolmesta APIsta käsittelee XML-dokumentteja hieman eri tavalla.



KUVA 3. XML-tiedoston lukeminen SAXilla (Means & Bodie 2002, 3)

SAXin suurin poikkeavuus JDOMiin löytyy sen tavasta käsitellä ja esittää XML-dokumentti. SAX näyttää XML-dokumentin pala palalta parsinnan edistyessä (McLaughling 2001, 131). SAX luo parsinnan aikana takaisinkutsuluokkia, jotka mahdollistavat ohjelmakoodin liittämisen SAXin parsimistapahtumiin (events) (kuva 3). Näitä tapahtumia voidaan käsitellä suoraan parsimisen aikana, mikä mahdollistaa dokumentin käsittelyn ilman, että se täytyy ladata kokonaan muistiin. SAXin etuna onkin sen keveys ja se, että se vaatii vähemmän muistia kuin DOM (Events vs. Trees 2004).



KUVA 4. XML-tiedosto DOM-puuna (Means & Bodie 2002, 2)

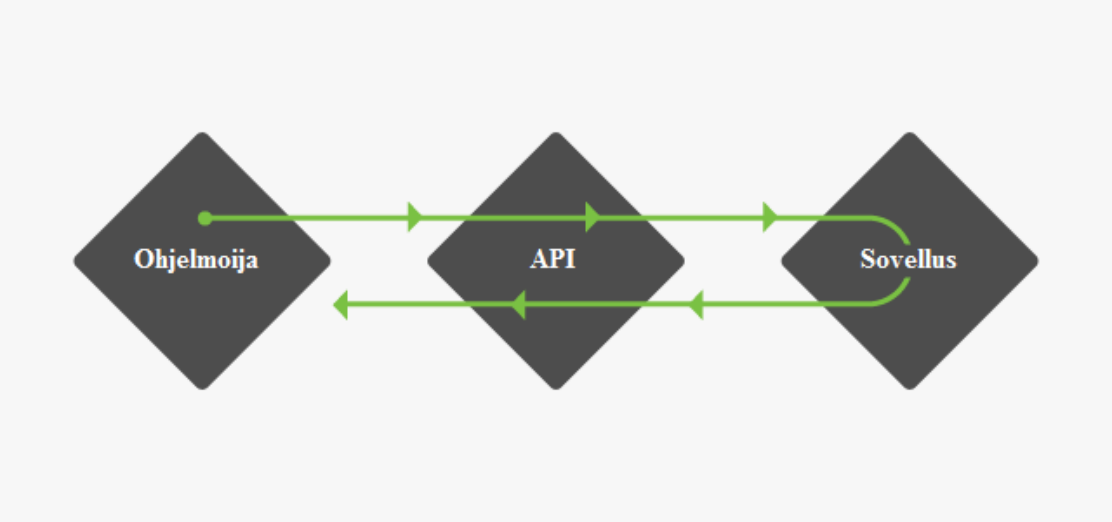
DOM käsittelee XML-dokumentin samalla tavalla puumuodossa kuten JDOM. DOM lukee ensin XML-dokumentin kokonaan muistiin, minkä jälkeen XML-dokumentin sisältöön pääsee käsiksi erinäisten elementtikohtaisten rajapintojen kautta (kuva 4). Eroavaisuus näiden kahden rajapinnan välillä löytyy käyttöympäristöstä. JDOM on kehitetty erityisesti Java-ohjelmointiin, toisin kuin DOM, joka on suunniteltu edustamaan dokumenttien sisältöä ja malleja riippumatta käytettävästä ohjelmointikielestä tai työkaluista (McLaughling 2001, 129).

JDOM valikoitui käytettäväksi rajapinnaksi sen takia, että se käsittelee XML-dokumentin itselleni tutussa puumuodossa ja se on suunniteltu Java-ympäristöön, mikä mahdollistaa sen, että XML-dokumentin käsittelyssä voidaan käyttää Javan Collections-luokkia, toisin kuin DOMissa. JDOM myös hyväksyy syötteenä SAXin tapahtumia, JDBCn ResultSet-olioita ja DOM-puita, jonka ansiosta se pystyy käsittelemään eri muodoissa olevia XML-dokumentteja. (Mission 2012.)

2.4 Web API

APIlla tarkoitetaan metodien sarjaa, jonka avulla päästään käsiksi dataan, johon pääsy olisi muussa tapauksessa suljettu. APIt tarjoavat siis eräänlaisen reitin kahden tai useamman ohjelman välille, jonka kautta ne kykenevät liikuttamaan dataa toisilleen (Proffit 2014). APIen toimintaan liittyy usein joukko sääntöjä, joita ohjelmoijien tulee

noudattaa. Ne määrittelevät muun muassa sen minkälaista tietoa API vaatii toimiakseen ja sen minkälaisen vastauksen API antaa sitä käyttäneelle ohjelmalle (Patterson 2015).



KUVA 5. APIa käyttävän sovelluksen toiminta (Patterson 2015)

APIen pääasiallinen käyttötarkoitus on tarjota ohjelmoijille työkalut, joiden avulla he voivat rakentaa sovelluksen tai palvelun, joka hakee käytettävän datan ulkoisesta lähteestä (Michel 2013, 12). APIen hyöty on se, että ne muotoilevat palautetun datan helpommin käsiteltävässä muodossa (Beach 2013). API siis toimii eräänlaisena välikätenä ohjelmoijan ja palvelun välillä, tarjoten helpon reitin päästä käsiksi dataan, mutta samalla rajoittaen saatavilla olevan datan määrää (kuva 5).

Web APIn toimintaperiaate on pääosin sama kuin APIn, mutta poikkeuksiakin löytyy. Suurin ero Web APIn ja APIn välillä on, että siinä missä API koostuu joukosta metodeja, Web APIt koostuvat joukosta verkkopalveluja, jotka tarjoavat palautettavan datan. Web APIn toinen olennainen tunnuspiirre on se, että ne ovat täysin riippumattomia niin käyttöjärjestelmästä kuin ohjelmointikielestä, joiden avulla niihin otetaan yhteyttä. Tavalliset APIt sen sijaan ovat yleensä rajoitettu toimimaan tietyllä ohjelmointikielellä tai ainakin jokaiselle tuetulle ohjelmointikielelle on jouduttu tekemään oma versionsa APIsta. Toisaalta tavalliset APIt ovat yleensä myös rajattu tietyn tyyppisen datan käsittelyyn, kun taas Web APIt tarjoavat ainoastaan rajapinnan, jonka kautta pääsee käsiksi kyseisen palveluntarjoajan tuottamaan dataan (Michel 2013, 12). (Gosnell 2005, 1–2.)

Riippumatta siis siitä, onko kyseessä perinteinen API vai Web API, APIt mahdollistavat sovellusten toteuttamisen ilman, että ohjelmointia täytyisi aloittaa aina alusta. Tällöin saman APIn avulla toteutetut sovellukset käyttävät APIlla toteutetun osuuden kohdalla samaa liiketoimintalogiikkaa (Kearn 2015). Ohjelmoinnissa liiketoimintalogiikalla tarkoitetaan niitä sääntöjä, jotka määrittelevät kuinka dataa luodaan, näytetään, varastoidaan tai muutetaan (Harvey 2014). Ohjelman liiketoimintalogiikka pitää sisällään kaikki algoritmit ja koodin, joka tarvitaan siihen, että ohjelma toimii yrityksen asiakkailla ja palvelimilla. Liiketoimintalogiikka ei tosin pidä sisällään ohjelman käyttöliittymän ulkoasua tai sitä dataa, joka kulkee sovelluksen ja palvelimen välillä eri verkkoprotokollien välityksellä. (Business Logic 2016.)

API-pohjainen liiketoimintalogiikka mahdollistaa sovellusten nopeamman kehityksen, mikä vähentää sovelluksen toteutusvaiheen kustannuksia (Kearn 2015). Erityisesti Web APIen avulla toteutetut sovellukset ovat myös kevyempiä, mikä on olennaista esimerkiksi mobiililaitteille tarkoitettujen sovellusten kohdalla (Riddel 2016). Web APIen avulla toteutetut sovellukset toimivat käytettävän datan päällä erillisenä tasona, joka tarjoaa ainoastaan käyttöliittymän datan esittämistä ja käsittelyä varten (Kearn 2015).

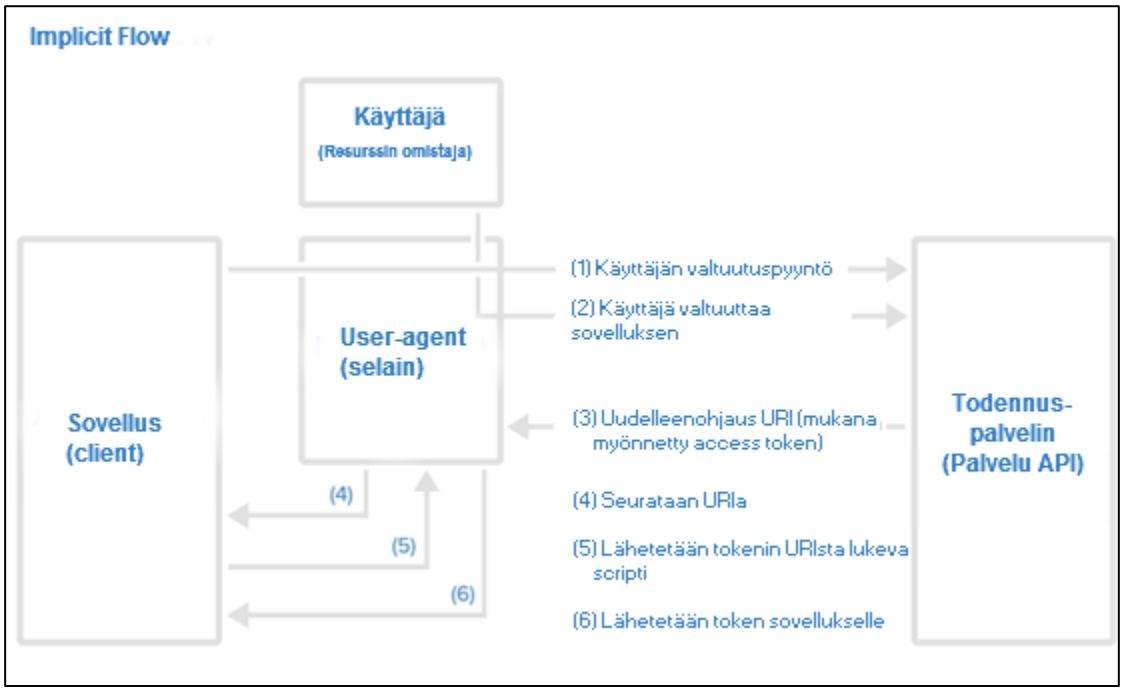
2.5 OAuth

OAuth on sovelluskehys (framework), jonka tarkoitus oli alun perin olla protokolla (Hammer 2012). Sen toimintaperiaatteen ymmärtämiseen liittyy hyvin usein virheitä (Anicas 2014). Hyvin usein nimittäin ymmärretään, että OAuth olisi käyttäjän tunnistusprotokolla, mitä se ei ole. Todellisuudessa OAuth on välityskehys, jota käytetään tunnistamaan käyttäjän sovellukselle antama lupa. OAuthin avulla käyttäjä pystyy valtuuttamaan sovellukselle rajoitetun oikeuden suorittaa toimintoja verkkopalvelussa ja käyttää hänen tietojaan ilman, että käyttäjän täytyisi antaa sovellukselle käyttäjätunnuksensa ja salasanansa (Twitter 2016). OAuthia käytetään näiden ominaisuuksien vuoksi yleensä erilaisissa mobiili-, verkko- ja työpöytäsovelluksissa (Anicas 2014). Näissä toimintaympäristöissä OAuthin käyttötarkoitus on antaa sovelluksille lupa suorittaa toimintoja käyttäjän nimissä, ilman että käyttäjä olisi välttämättä edes paikalla. (Richer 2016.)

OAuthin toiminta perustuu siis käyttäjän myöntämään lupaan, josta käytetään nimitystä token, sekä OAuthin eri rooleihin. Ensimmäinen rooli on käyttäjä, joka myöntää OAuthia käyttävälle luvan, eli tokenin. Toinen rooli on itse sovellus, joka käyttää tokenia. Kolmantena roolina toimii tunnistuspalvelin, joka joko sallii tai estää sovelluksen toiminnan verkkoympäristössä riippuen siitä, onko token kelvollinen vai ei. Viimeinen rooli on resurssipalvelin, jossa suoritetaan sovelluksen haluama toiminto. (Anicas 2014.)

Kaikki sovellukset, jotka käyttävät OAuthia täytyy rekisteröidä palveluntarjoajan sivustolle. Tällöin niille myönnetään client id- ja client secret -tunnukset, joita käytetään API-kutsuissa. Client id on sovelluksen julkinen tunnus, jota voi jakaa vapaasti käytettäväksi, mutta client secret sen sijaan toimii sovelluksen salasanana ja se täytyy pitää salassa. (Authentication 2012.)

OAuthilla on useita erilaisia tapoja pyytää itselleen tokenia verkkopalvelulta. Käytettävä pyyntötapa riippuu pääasiassa sovelluksen toimintaympäristöstä. Ensimmäinen tapa, Authorization Code, on tarkoitettu palvelimella toimivien sovellusten käyttöön. Toinen tapa, Implicit, on tarkoitettu palvelimettomille sovelluksille, kuten esimerkiksi mobiilisovelluksille. Twitch tukee ainoastaan näitä kahta pyyntötapaa omassa APIssaan (Authentication 2012). Kolmas tapa, Client Credentials, on tarkoitettu siihen tapaukseen, jos sovellus haluaisi muokata omaa palveluntarjoajan tiliä ja tämän käyttäminen vaatii client secretin lisäämisen REST-pyyntöön. Viimeinen tapa, Resource Owner Password Credentials, on tarkoitettu ainoastaan verkkopalvelun omistajan käyttöön, joten sitä ei tarjota yleiseen käyttöön lainkaan. (Anicas 2014.)



KUVA 6. Epäsuora (implicit) tokenin hankintatapa (Anicas 2014)

Koska jokainen hankintaprosessi on hieman erilainen, on myös niissä käytävät vaiheet erilaiset. Omassa opinnäytetyössäni tulen käyttämään implicit-tyyppistä hankintatapaa, joka ei vaadi sovellukselle omaa palvelinta. Implicit eli epäsuora hankintatapa perustuu siihen, että token palautetaan sovellukselle URIn mukana, josta sen voi poimia suoraan (kuva 6).

2.6 REST

REST eli Representional State Transfer on arkkitehtuurityyli, jonka tarkoituksena on säilyttää useiden itsenäisesti kehittyvien järjestelmien välinen toimivuus (Ruokonen 2010). Se ei siis itsessään ole palvelujen kehittämiseen käytettävä järjestelmä, vaan sarja erilaisia vaatimuksia ja rajoituksia, joita käytetään sovellusten ja palvelujen kehittämiseen (Sandoval 2009, 7). Verkkopalveluista (Web Service), jotka on toteutettu RESTin periaatteiden mukaisesti, käytetään yleisesti termiä RESTful (Kalali & Mehta 2013, 8).

RESTiä käytetään yleensä aina HTTP-protokollaan pohjautuvien verkkopalveluiden kanssa sen keveyden takia (Rouse ym. 2014). Keveyden lisäksi RESTin etuja ovat alustariippumattomuus ja kyky kommunikoida eri ohjelmointikielillä toteutettujen palveluiden kanssa (Elkstein 2008a).

RESTin arkkitehtuurityyli perustuu erilaisiin pyyntöihin ja niitä seuraaviin vastauksiin (Kalali & Mehta 2013, 8–9). Pyynnöt lähetetään verkkopalveluun URIn avulla (Sandoval 2009, 10). URI on osoitin, jonka avulla osoitetaan palvelun sijainti internetissä (Rouse & Norman 2005). Yleisemmin tunnettu URL (Uniform Resource Location) sen sijaan on URIn alalaji, jonka ero URIin verrattuna on se kuinka tarkasti resurssin sijainti ja tyyppi ilmaistaan (Miessler 2016). URL kertoo sen verkkoprotokollan, jonka avulla resurssiin pääsee käsiksi (esimerkiksi HTTP tai FTP), resurssin tiedostotyypin ja mahdollisesti myös portin, johon yhteys muodostetaan (What is a URL? 1997). URIn ja URLin välinen ero on siis pieni ja Miesslerin (2016) mukaan URL on vanhentunut termi, joka on yhdistymässä URIksi.

URIlla lähetettyjen pyyntöjen yhteydessä välitetään erillinen käskyverbi (POST, PUT, DELETE ja GET), joilla kerrotaan verkkopalvelulle mitä sen tulee tehdä saamallaan datalla. POST-pyyntöä käytetään uuden tiedon luomiseen palvelimelle, PUT-pyyntöä jo olemassa olevan tiedon päivittämiseen, DELETE-pyyntöä tiedon poistamiseen ja GET-pyyntöä tiedon noutamiseen (Elkstein 2008a).

Ensimmäinen ja tärkein RESTful-verkkopalveluille asetettu vaatimus on se, että palvelun täytyy olla suunniteltu clientin ja palvelimen väliseen toimintaan perustuen (client-server system). Tämän vaatimuksen tarkoituksena on mahdollistaa se, että palvelimella sijaitsevaa verkkopalvelua ja verkkopalvelua käyttävää asiakassovellusta voidaan molempia kehittää ja ylläpitää toisistaan riippumattomasti. (iMasters Expert 2015.)

Seuraava RESTful-verkkopalvelun vaatimus on se, että palvelu on tilaton (stateless). Tilattomalla palvelulla tarkoitetaan sitä, että jokainen sille lähetetty HTTP-pyyntö on yksilöllinen ja kaikki palvelun tarvitsema tieto löytyy sille lähetetystä HTTP-pyynnöstä. Toisin sanoen joka kerta kun HTTP-pyyntö lähetetään verkkopalvelulle, sille täytyy antaa uudestaan palvelun vaatimat parametrit. (RESTful Web Services - Statelessness 2016.)

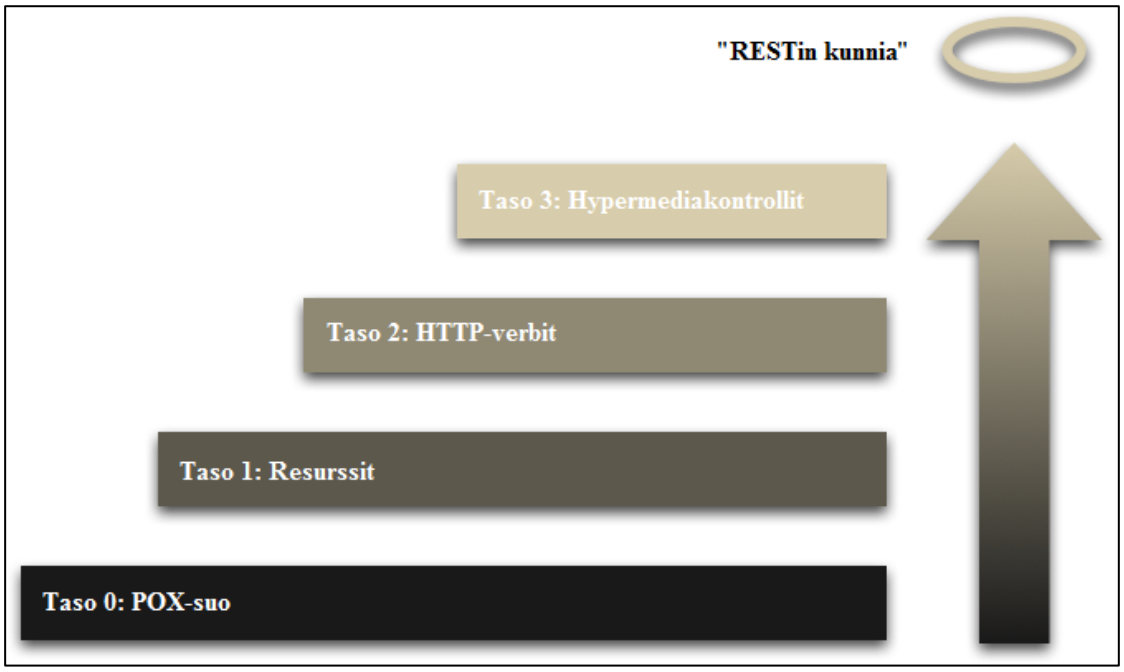
Koska samaa verkkopalvelua saattaa käyttää useampi asiakassovellus, verkkopalvelun täytyy pystyä varastoimaan HTTP-pyynnölle palautettava data välimuistiin. Tällöin palvelimella pidetään väliaikaisesti muistissa HTTP-pyynnölle palautettu data, jota

käytetään uudestaan heti, kun vastaava pyyntö saapuu palvelimelle. Tämän vaatimuksen tarkoituksena on pääasiassa vähentää palvelimelta asiakassovelluksille lähtevää tiedonsiirron määrää ja keventää palvelimelle kohdistuvaa kuormitusta. Tämän vaatimuksen toteutustavoissa on kuitenkin eroavaisuuksia, jotka riippuvat verkkopalvelun tarjoamien resurssien päivitystiheydestä. (Kirby 2013.)

Neljäs RESTful-palvelun vaatimuskriteeri on se, että jokaisella Web APIn palvelulla täytyy olla yksilöllinen URI. Tämän vaatimuksen tarkoitus on helpottaa verkkopalvelun tarjoamien resurssien tunnistamista (iMasters Expert 2015). Se myös mahdollistaa palvelun tarjoamien resurssien manipuloinnin lisäämällä URIin parametrejä, tehden samalla verkkopalvelusta optimaalisemman suurten datamäärien lähettämiseen (Louvel 2013).

Jotta RESTin ensimmäisenä mainittu kriteeri täyttyisi, RESTful-palveluilta edellytetään sitä, että ne on toteutettu tasoista muodostetuille järjestelmälle (layered system). Tällä tarkoitetaan sitä, että asiakassovellus ei ikinä lähetä HTTP-pyyntöä suoraan sille palvelimelle, jossa tavoiteltu resurssi on, vaan palvelimen ja internetin välissä täytyy olla välikäsi (iMasters Expert 2015). Tämä mahdollistaa sen, että palveluntarjoaja voi vapaasti laajentaa palvelua ja muuttaa palvelinrakennetta ilman, että se vaikuttaa asiakassovellusten toimintaan. Ainoa haittapuoli tässä vaatimuksessa on se, että se saattaa joissain tapauksissa aiheuttaa viivettä HTTP-pyynnön käsittelyyn ja vastaukseen. (Fielding 2002.)

Viimeinen RESTin asettama vaatimus on tarjota valmista koodia asiakassovellukselle (Sandoval 2009, 8). Tämä vaatimus on valinnainen ja sen tarkoitus on pääasiassa antaa asiakassovellukselle valmiita työkaluja verkkopalvelun palauttaman datan käsittelyyn ja keventää samalla asiakassovellusten kehittämiseen vaadittua työtä (Fielding 2002).



KUVA 7. Richardsonin kypsyysmalli (Fowler 2010)

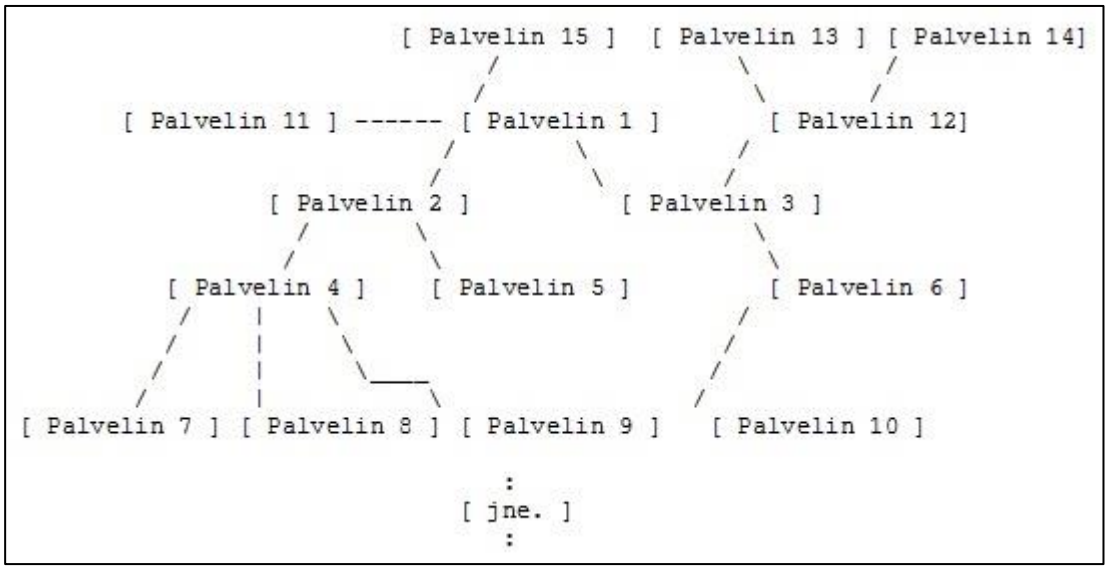
Jotta verkkopalvelua voidaan kutsua RESTful-palveluksi, sen täytyy seurata tarkasti edellä mainittuja kriteerejä (paitsi viimeistä, joka on valinnainen). Näiden kriteerien täyttymistä ja verkkopalvelun kypsyyttä voidaan mitata Leonard Richardsonin kehittämää kypsyysmallia (kuva 7). Mallin alimmalla tasolla on sellaiset verkkopalvelut, jotka tarjoavat kaikki palvelut saman URIn kautta ja käyttävät ainoastaan yhtä metodia HTTP-pyynnön yhteydessä (HTTPssä yleensä POST-metodia). Esimerkkinä tällaisesta järjestelmästä ovat SOAPilla toteutetut verkkopalvelut. (What is the Richardson Maturity Model? 2016.)

Jotta verkkopalvelu saavuttaisi tason yksi, sen täytyy olla hajautettu useampaan päätepisteeseen eli resurssiin. Sen lisäksi asiakassovelluksen ja palvelun välisessä kommunikaatiossa manipuloidaan haluttua resurssia sen sijaan, että kutsuttaisiin jotain tiettyä metodia (Palmer 2015). Seuraavan tason saavuttamiseksi vaaditaan HTTP-verbien monipuolisempi käyttö. Sen sijaan, että tyydyttäisiin ainoastaan POSTiin, käytetään haluttua toimintoa oikeasti kuvaavaa verbiä (iMasters Expert 2015). Tämä vaaditaan pääasiassa siksi, että HTTP-protokolla määrittelee GETin täysin turvalliseksi operaatioksi, jota voidaan käyttää rajoittamattomasti ilman, että tuloksissa tapahtuu muutoksia (Fowler 2010). Richardsonin mallin viimeisen tason saavuttamiseen verkkopalveluun täytyy käyttää HATEOASia (Hypertext as The Engine Of Application State) (What is the Richardson Maturity Model? 2016). Tällöin verkkopalvelu osaa itse automaattisesti kertoa sen, mitä sen palauttamalla tiedolla voi tai täytyy tehdä seuraavak-

si ja missä palvelun tarjoamassa resurssissa kyseinen toiminta voidaan suorittaa (Lublinsky 2010). Tämä auttaa asiakassovellusten kehittäjiä tutustumaan verkkopalvelun toimintaan ja sen tarjoamiin resursseihin (Fowler 2010).

2.7 IRC

IRC on avoin protokolla, jonka Jarkko Oikarinen kehitti 1988 (Stenberg 2011). Se on suunniteltu toimimaan TCP/IP-protokollaa käyttävissä järjestelmissä (Oikarinen & Reed 1993). IRCissä keskustelu tapahtuu erillisellä palvelimella, jolle muodostetaan yhteys joko erillisen IRC-clientin kautta tai verkossa olevan käyttöliittymän kautta (IRC 2016). IRCin toiminta perustuu client/palvelin -malliin, jossa client-sovellus pyytää palvelimelta resursseja tai palveluja (Rouse & Sullivan 2008).



KUVA 8. IRC-verkko (Oikarinen & Reed 1993)

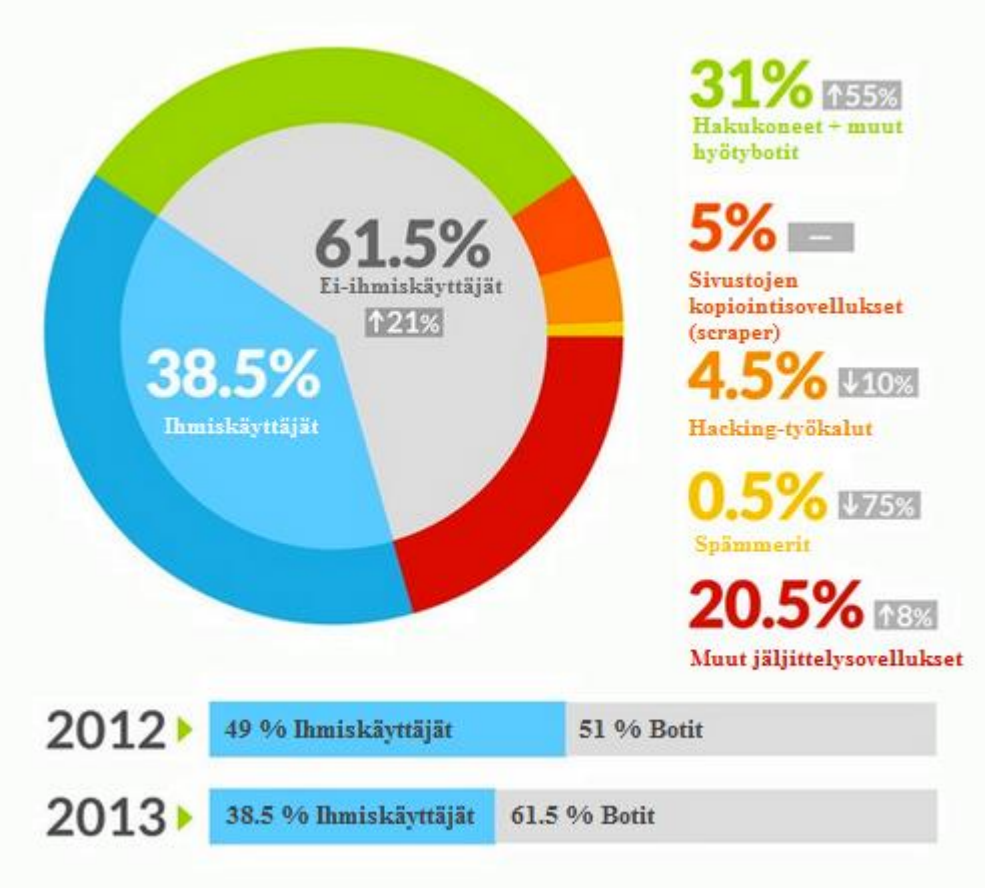
IRC-palvelimen toiminta perustuu STP-protokollaan (Spanning Tree Protocol), jossa eri palvelinten välillä on useita eri reittejä (Oikarinen & Reed 1993). STP-protokollan päätarkoituksena on estää silmukoiden syntyminen ja varmistaa, että palvelinten välillä on aina toimiva yhteys (Heikkiniemi 2000). Näitä toisiinsa yhdistettyjä palvelimia kutsutaan myös IRC-verkoksi (kuva 8). IRC-verkon sisällä olevat kanavat ja käyttäjätunnukset (nick) ovat uniikkeja ja niitä voi olla ainoastaan yksi. Tämä tarkoittaa myös sitä, että kanavat ja käyttäjätunnukset näkyvät palvelimelta toiselle. Poikkeuksen kuitenkin muodostavat ne kanavat, jotka alkavat &-merkillä (normaalisti kanavan nimi alkaa #-merkillä). Nämä kanavat eivät näy muille IRC-verkon palvelimille, vaan ne

ovat niin sanottuja paikallisia kanavia, jotka näkyvät vain sille palvelimelle, jossa ne sijaitsevat. (Caraballo & Lo 2014.)

Kanaville liittymiseen ja siellä keskustelemiseen käytetään yleensä erillistä IRC-clienttia, joita on tarjolla hyvin pitkälti jokaiselle käyttöjärjestelmälle (Caraballo & Lo 2014). Teoriassa kuitenkin IRC-kanavalla keskusteluun riittää mikä tahansa ohjelma, mikä vain kykenee muodostamaan yhteyden palvelimelle. IRC-chatissa poikkeuksellista on, että vaikka kanavat ja käyttäjätunnukset ovat uniikkeja, käyttäjillä ei yleensä ole omistusoikeutta niihin. Tämä johtuu pääasiassa siitä, että kanavia luodaan sitä mukaa kun ensimmäinen henkilö liittyy kyseiselle kanavalle. Käyttäjätunnukset toimivat samalla periaatteella. Mikäli käyttäjätunnus on varattu, kuka tahansa voi ottaa sen käyttöönsä. Tosin nykyään palvelut tarjoavat myös mahdollisuuden rekisteröidä kanavan tai käyttäjänimen omaan käyttöönsä (IRC 2016). (van Loon & Lo 2004.)

3 BOTIN TOTEUTUS

Botilla, eli tarkemmin sanottuna internet botilla, tarkoitetaan sovellusta, joka suorittaa automaattisesti sille osoitettuja tehtäviä internetissä (Internet Bot 2016). Termi botti on lyhennetty sanasta robotti (What’s a Bot? 2016). Botteja käytetään pääasiassa suorittamaan yksinkertaisia ja helposti toistettavia tehtäviä, jotka olisivat ihmisen suorittamana liian hitaita, turhia tai mahdottomia toteuttaa (Internet Bot 2016). Sen lisäksi että niitä käytetään hyötykäyttöön soveltuvina työkaluina, botteja käytetään internetissä usein myös haittaohjelmina, jolloin niistä käytetään myös termiä zombi (Bradley 2014).



KUVA 9. Bottien osuus internetissä tapahtuvasta liikenteestä (Kerr 2013; Gaffan 2012)

Vaikka botteja käytetään usein haittaohjelmina, niitä käytetään myös lähes yhtä paljon hyödyllisiin käyttötarkoituksiin (kuva 9). Bottien aiheuttaman tietoliikenteen kasvun uskotaan johtuvan muun muassa erilaisten hakukoneiden yleistymisestä ja internetin käyttäjien määrän lisääntymisessä (Kerr 2013). Hyötykäytössä botteja käytetään hakukoneina, hintavertailun tukena ja internetsivuilla olevan median päivittämisessä (Different Types of Internet Bots and How They Are Used 2016). Haittaohjelmina niitä käytettiin aiemmin spämmäykseen, mutta nykyään niitä käytetään enemmänkin palvelunestohyökkäyksissä ja identiteettivarkauksiin tai -huijauksiin (Kerr 2013).

Hyvän ja pahan botin ero ei nykyään ole myöskään enää niin mustavalkoinen kuin ennen. Esimerkiksi käyttäjän toimintaa ja tietoja seuraava botti kategorisoidaan hyväksi tai pahaksi pääosin sillä perusteella, mihin sen keräämää tietoa käytetään. Jos botin avulla kalasteltuja tietoja käytetään esimerkiksi identiteettivarkauden toteuttamiseen tai käyttäjän tietoihin murtautumiseen, kyseessä on paha botti. Mutta
jos samoja tietoja käytetään asiakaskäyttäytymisen tutkimiseen ja markkinointiin, kyseessä onkin hyvä botti. (Ellis 2015.)

Tällä hetkellä bottien luvattu maa löytyy Twitteristä, jossa niitä käytetään muun muassa ihmisten huvittamiseen, markkinointiin sekä erilaisiin tutkimuksiin. Hupikäytöstä esimerkkinä toimii botti, joka huomauttaa twiittaajaa kirjoitusvirheistä sekä Big Benin Twitter-tili, joka kertoo tasatunnein kellonajan toistamalla BONG-sanaa (BrainJet 2014). Markkinoinnin apuna botteja käyttävät esimerkiksi museot, joiden Twitterissä toimivat botit twiittaavat kuvia museoiden sisältämistä taideteoksista (Voon 2016). Tutkimuskäytöstä esimerkkinä toimii muun muassa Microsoftin toteuttama Tay-botti, jonka käyttötarkoituksena oli toteuttaa tekoäly, joka oppii Twitterin kautta keskustelemaan muiden käyttäjien kanssa kuin ihminen (Binkowski 2016).

Twitchissä bottien käyttötarkoitus on Twitterissä toimiviin botteihin verrattuna paljon yksinkertaisempi ja kapeampi. Suurin osa Twitchissä olevista boteista ovat chatbotteja, jotka toimivat Twitchin IRC-kanavien valvojina, pitäen chatin puhtaana ja poistaen sieltä ne käyttäjät, jotka eivät kykene asialliseen keskusteluun. Chatissä liikkuu myös sellaisia botteja, joiden tarkoituksena on edistää streamiä seuraavien katsojien yhteisöllisyyttä järjestämällä erilaisia pelejä tai kirjoittamalla hauskoja kommentteja. (Wiesehan 2015.)

3.1 Tavoitteet

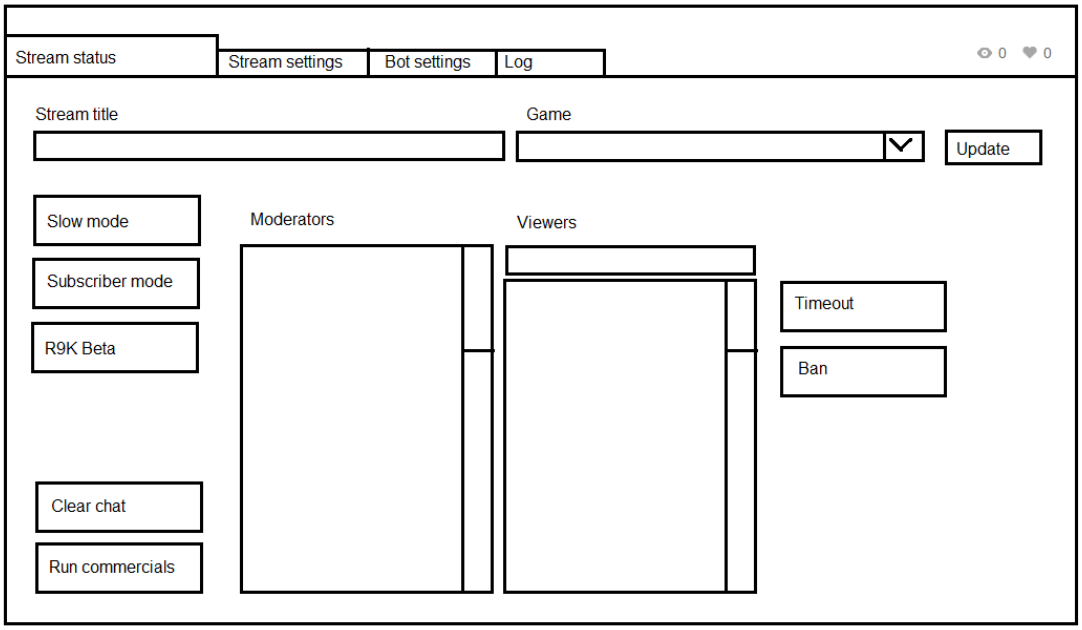
Opinnäytetyön tavoitteena on siis ohjelmoida Javalla botti, joka valvoo Twitchin IRC-palvelimella tapahtuvaa keskustelua. Botilla tulee olla oma hallintapaneeli erilaisia asetuksia ja chatin hallintaan liittyviä toimintoja varten. Botin ja chattipalvelimen välinen viestintä on tarkoitus ohjelmoida alusta asti itse, mikäli vain mahdollista. Botin pitää myös kyetä seuraamaan chatissa käytyä keskustelua ja tunnistamaan siellä mahdollisesti esiintyvät ei-halutut sanat ja botille osoitetut komennot ja toimia niiden mukaan.

Palvelin lähettää botille tietyin väliajoin erilaisia viestejä, joiden mukaan sen tulee toimia tietyllä tavalla. Näistä olennaisin on PING-viesti, jonka tarkoitus on säilyttää aktiivinen yhteys botin ja palvelimen välillä. Palvelin lähettää botille myös säännöllisin väliajoin listan chattiin liittyneistä ja poistuneista käyttäjistä. Koska botin on tarkoitus toimia ilman erillistä palvelinta, botin käyttämät tiedot ja asetukset tallennetaan

erillisiin XML-tiedostoihin. XML-tiedoston rakenne ja tietojen muokkaus on tarkoitus toteuttaa JDOMilla.

Hallintapaneelin kautta tulee pystyä nopeasti ottamaan käyttöön tai poistaa käytöstä chatin eri tiloja, kuten slow mode ja subscriber mode. Hallintapaneelin kautta tulee myös tapahtua botin asennus eli käyttöönotossa vaadittujen tietojen keräys. Hallintapaneelin kautta tulee pystyä muokkaamaan helposti streamin otsikkoa ja streamattavan pelin nimeä. Twitch tarjoaa yhteistyökumppaneilleen mahdollisuuden näyttää mainoksia streamin aikana. Mainosten ajamista varten botin hallintapaneeliin on tarkoitus oma painike mainosten käynnistämistä varten.

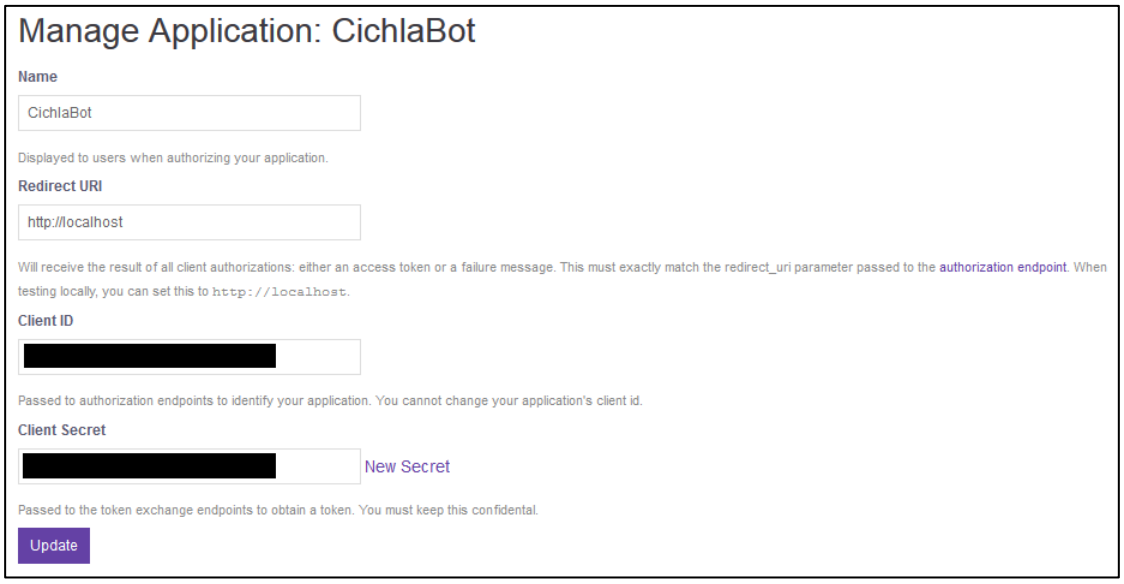
Hallintapaneelin kautta toteutetaan myös botin ja streamin asetusten hoitaminen. Tähän sisältyy niin mainosten, timeouttien ja kirjautumistietojen muokkaus. Asetuksiin sisältyy myös vaihtoehdot vaikuttaa siihen, minkälaisia tietoja botti kirjoittaa tapahtumalokiin.



KUVA 10. Rautalankamalli toteutettavan botin hallintapaneelista

Aloitin botin työstämisen suunnittelemalla sen käyttöliittymästä rautalankamallin (kuva 10; liite 1). Rautalankamallin tarkoitus on antaa parempi käsitys toteutettavan sovelluksen toiminnallisista ja teknisistä vaatimuksista (Wulf 2012). Oman työni osalta sen tarkoitus oli antaa havainnollisempi käsitys siitä, minkälaisia toimintoja ja asetuk-

sia tulen työn aikana tarvitsemaan toimivan kokonaisuuden aikaan saamiseksi. Se myös tulee auttamaan myöhemmin tarvittavan XML-tiedoston suunnittelemisessa.



KUVA 11. Sovelluksen rekisteröinti Twitchiin

Seuraava vaihe botin toteutuksessa oli botin rekisteröinti Twitchiin ja erillisen käyttäjätunnuksen luominen sitä varten (kuva 11). Botin rekisteröinti Twitchiin on pakollinen toimenpide, joka vaaditaan ennen kuin ohjelmoijalle myönnetään client ID ja client secret, joita käytetään myöhemmin API-kutsuissa (Authentication 2012). Oman käyttäjätunnuksen luominen bottia varten ei käytännössä ole pakollinen toimenpide, mikäli ohjelmoijalla on jo olemassa oleva käyttäjätunnus Twitchiin. Käyttäjätunnusta tullaan käyttämään pääasiassa ainoastaan IRC-chattiin kirjautumiseen ja sinne kirjoittamiseen. Kuitenkin, mikäli muut chatissä olevat käyttäjät halutaan ottaa huomioon, on oman käyttäjätunnuksen luominen botille kannattavaa ja jopa suositeltavaa. Näin muiden käyttäjien on helpompi ymmärtää, että chattiin kirjoittaa botti, jonka viestit on automatisoitu, eikä itse käyttäjä.

3.2 IRC

IRC-yhteyden muodostaminen Twitchin IRC-palvelimelle noudattaa suurimmaksi osaksi samoja sääntöjä kuin tavalliselle IRC-palvelulle tapahtuvan yhteyden muodostaminen. Yhteys muodostetaan luomalla socketti kanavalla tapahtuvan keskustelun lukemista varten. Socketin luomista varten tarvitaan palvelimen URI ja portin numero. Botin ohjelmoinnin aikana (maaliskuu 2016) Twitch alkoi siirtää IRC-palvelimiaan

Amazonin tarjoamaan AWS-pilvipalveluun (Amazon Web Services), mikä aiheutti sen, että osalla kanavista muodostettiin yhteys käyttäen vanhaa URIa ja osalle uutta URIa (Bashore 2016).

```
public void createSocket() throws IOException{
    // Muodostetaan yhteys IRC-palvelimelle
    this.socket = new Socket(this.server, this.port);

    writer();
} // createSocket
public void writer() throws IOException{
    // Muodostetaan yhteys palvelimelle kirjoittavaan tietovirtaan
    OutputStreamWriter osw = new OutputStreamWriter(this.socket.getOutputStream(), StandardCharsets.UTF_8);
    this.writer = new BufferedWriter(osw);
} // writer

public BufferedReader reader() throws IOException{
    // Muodostetaan yhteys palvelimelta tulevaan tietovirtaan
    InputStreamReader isr = new InputStreamReader(this.socket.getInputStream(), StandardCharsets.UTF_8);
    this.reader = new BufferedReader(isr);
    return this.reader;
} // reader

public void connect() throws IOException{
    // Liitytään kanavalle
    write("CAP REQ :twitch.tv/membership");
    write("CAP REQ :twitch.tv/commands");
    write("CAP REQ :twitch.tv/tags");
    write("PASS oauth:" + this.token);
    write("NICK " + this.username);
    write("JOIN " + this.channel);
} // connect
```

KUVA 12. IRC-yhteyden muodostaminen Javalla

Suurimmat eroavaisuudet Twitchin IRC-palvelimen toiminnassa näkyy kanavalle liittymisen yhteydessä. Vaikka IRC-kanavat yleensä sallivat käyttäjän valita vapaasti oman käyttäjätunnuksen (nick), niin Twitch on päättänyt rajoittaa käyttäjätunnusta siten, että käyttäjätunnuksen täytyy olla sama kuin Twitch-käyttäjätunnus. Tämän lisäksi Twitch on luonut kolme erilaista kerrosta kanavalle liittymiseen, jotka vaikuttavat palvelimen lähettämiin viestiin, joko muokaten niiden sisältöä tai suodattaen osan viesteistä jättäen ne kokonaan lähettämättä. Twitchin kanavalle liittyessä täytyy myös olla tarkkana sen suhteen, missä järjestyksessä liittymiseen tarvittavat viestit tulee lähettää (kuva 12). Ensimmäisenä täytyy lähettää pyyntö, joka määrittelee millä tasolla viestejä halutaan lukea (CAP REQ -viestit). Tämän jälkeen tulee lähettää käyttäjän myöntämä OAuth-token, joka toimii salasanana. Ilman toimivaa OAuth-tokenia botti ei pysty liittymään kanavalle. Tokenin lähettämisen jälkeen lähetetään käyttäjätunnus pienaakkosilla kirjoitettuna. Vasta näiden viestien jälkeen palvelimelle voi lähettää JOIN-komennon, joka saa botin liittymään kanavalle ja mahdollistaa kanavalla tapahtuvan keskustelun seuraamisen.

```
// Tämän metodin avulla kirjoitetaan viestit IRC-palvelimelle ja huolehditaan,
// että Twitchin asettamat rajoitukset lähetettäville viesteille ei ylity missään vaiheessa.
private void write(String message) throws IOException{
    // Lisätään lähetettävä viesti lähetettävien viestien jonoon
    this.messageQueue.add(message);

    // Jos sallittu lähetettyjen viestien raja on ylittynyt, puhdistetaan aikaleimalista
    if(this.messageTimestamps.size() >= this.messageLimit){
        checkTimestampList();
    }

    // Käydään läpi lähetettävien viestien jono
    while((this.messageTimestamps.size() < this.messageLimit) && (this.messageQueue.size() > 0)){
        // Kirjoitetaan viesti
        this.writer.write(this.messageQueue.get(0) + "\r\n");
        this.writer.flush();

        // Poistetaan viesti jonosta
        this.messageQueue.remove(0);

        // Merkitään lähetetystä viestistä unix-timestamp listaan, jotta vältytään ylittämästä
        // lähetettävien viestien rajaa
        this.messageTimestamps.add(System.currentTimeMillis() / 1000L);

        // Tyhjennetään vanhentuneet aikaleimat
        checkTimestampList();
    }
} // write

// Metodi, jota käytetään tyhjentämään messageTimestamp-listasta ne
// aikaleimat, jotka ovat luotu myöhemmin kuin 30 sekuntia sitten.
private void checkTimestampList(){
    for(int i = 0; i < this.messageTimestamps.size(); i++){
        if(((System.currentTimeMillis() / 1000L) - this.messageTimestamps.get(0)) > 30){
            this.messageTimestamps.remove(0);
        }
    }
} // checkTimestampList
```

KUVA 13. Palvelimelle lähetettävien viestien käsittely

Botin toiminnan sydämenä toimii viestien lähettämisestä vastaavat metodit (kuva 13). Sen tarkoituksena on seurata viimeisten 30 sekunnin aikana lähetettyjen viestien määrää ja pitää se Twitchin määrittelemien rajoitusten alapuolella. Tämän lisäksi sen tulee ottaa huomioon se, että botin täytyy kyetä vastaamaan välittömästi IRC-palvelimelle, mikäli palvelin on lähettänyt yhteyttä ylläpitävän PING-viestin. Nämä vaatimukset täytetään muodostamalla lähetettävistä viesteistä kaksi jonoa, joista toiseen tallennetaan lähetettävä viesti ja toiseen aikaleima, jolloin viesti on lähetetty. Poikkeuksena tähän toimintaan on palvelimen PING-viestiin lähetettävä viesti, joka lähetetään suoraan jonon ohitse, mutta sen lähetysajankohta lisätään aikaleimajonoon, jottei lähetettävien viestien raja ylity. Lähetettävien viestien jonoa tyhjennetään samaa tahtia viestien lähettämisen kanssa eli kun viesti on lähetetty, poistetaan kyseinen viesti jonosta. Aikaleimajonoa sen sijaan tyhjennetään vasta siinä vaiheessa, kun viestin lähetyksestä kulunut aika ylittää kolmenkymmenen sekunnin rajan. Näiden kahden jonon avulla pystytään varmistamaan, että kaikki tarvittavat viestit lähetetään varmasti ja se, että botti kykenee vastaamaan aina palvelimen PING-viestiin.

3.3 XML

Jotta käyttäjä kykenisi mukauttamaan bottia ja sen toimintaa omia tarpeitaan vastaavaksi, täytyy botille luoda sen toimintaan vaikuttavia asetuksia. Asetukset täytyy myös tallentaa erilliseen tiedostoon, jotta ne pysyisivät tallessa myös seuraavan käynnistyskerran yhteydessä. Tässä opinnäytetyössä asetustiedostoina käytetään XML-tiedostoja, joiden luomiseen ja muokkaamiseen käytetään Javalle saatavilla olevaa JDOMia.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>hamalainen</groupId>
    <artifactId>Oppari</artifactId>
    <version>1.0</version>
    <packaging>jar</packaging>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.7</maven.compiler.source>
        <maven.compiler.target>1.7</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.jdom</groupId>
            <artifactId>jdom</artifactId>
            <version>2.0.2</version>
        </dependency>
    </dependencies>
</project>
```

KUVA 14. Opinnäytetyöprojektin POM-tiedosto

Koska JDOM ei sisälly Javan oletuksena, se täytyy ensin liittää mukaan projektiin. JDOMin tai minkä tahansa muun Java-arkiston (JAR) voi liittää projektiin lataamalla se ensin internetistä ja sen jälkeen siirtämällä se projektiin. Toinen vaihtoehto arkiston liittämiseen on käyttää Mavenia tai jotain muuta vastaavaa kääntämisen automatisointiin tarkoitettua työkalua (build automation tool). Mavenia käytettäessä JDOMin JAR-pakettia ei tarvitse ladata itse, vaan sen sijaan luodaan Mavenin POM-tiedostoon (Project Object Model) riippuvuus, jonka pohjalta Maven noutaa kääntämisen yhteydessä JDOMin internetistä ja lisää sen käännettävään ohjelmaan (Apache 2016). POM on XML-tiedosto, joka pitää sisällään projektiin liittyvät tiedot, kuten ohjelman version, projektin nimen, kehittäjäorganisaation sekä projektin riippuvuudet (dependancies) eli projektin kääntämisen yhteydessä tarvittavat ulkopuoliset Java-arkistot (kuva 14). Mavenin pääasiallinen käyttötarkoitus on mahdollistaa projektin sisältämän ohjelman kääntäminen millä tahansa koneella, joka on yhteydessä internetiin ja käyttää samaa

kehitysympäristöä, ilman että koneelle täytyy käydä erikseen asentamassa projektin tarvitsemat Java-arkistot (Apache 2016).

```
public ArrayList<String> readXML(String filename, String attribute) throws JDOMException, IOException{
    ArrayList<String> content = new ArrayList();

    File xml = new File("xmlFiles/" + filename + ".xml");
    SAXBuilder builder = new SAXBuilder();
    Document saxDoc = builder.build(xml); // Luetaan XML-tiedosto SAXin avulla ohjelman muistiin
    Element root = saxDoc.getRootElement();

    List<Element> children = root.getChildren();
    Iterator itr = children.iterator();

    while(itr.hasNext()){
        // XMLään sijoitettua nimeä käytetään key-arvona.
        Element child = (Element)itr.next();
        String value = child.getAttributeValue(attribute);

        content.add(value);
    }

    return content;
} // readXML
```

KUVA 15. Kuinka JDOM lukee XML-tiedoston

Kuten luvussa 2.3 todettiin, JDOM on erillinen Java-arkisto, joka antaa ohjelmoijalle työkalut helpottamaan XML-tiedoston manipulointia. JDOMin toiminnassa olennaista on ottaa huomioon se, että JDOM ei kykene lukemaan XML-tiedostoa ilman SAXin tai DOMin apua. XML-tiedosto täytyy ensin lukea SAXilla tai DOMilla muistiin, josta se käännetään JDOMin puumuotoon. JDOM kuitenkin tarjoaa ohjelmoijalle työkalut, joiden avulla XML-tiedoston luetuttaminen on helppoa, eikä edellytä ymmärrystä SAXista tai DOMista (kuva 15). XML-tiedoston lukemiseen JDOMin kanssa suositellaan SAXia. Tämä johtuu siitä, että JDOM ja DOM esittävät molemmat XML-tiedoston puumuodossa, joka voi tehdä ohjelmasta raskaan, mikäli luettava XML-tiedosto on suuri (Rosen 2002). XML-tiedoston sisällään pitämät tiedot palautetaan joko ArrayList- tai Map-luokan muuttujana, riippuen XML-tiedostosta. Yksinkertaisemmat asetukset, kuten estetyt sanat ja käyttäjät palautetaan ArrayList-luokan avulla, sillä käyttäjätunnusta voidaan käyttää kyseisen elementin etsimiseen ja muokkaamiseen. Monimutkaisemmat asetukset, joiden attribuutit voivat muuttua, täytyy palauttaa Map-luokan avulla, sillä tällöin palautettavat arvot voidaan liittää eli indeksoida elementin nimeen. Tällöin asetuksen arvoa hakiessa tai muokatessa ei tarvitse tietää asetuksen nykyistä arvoa, vaan sen sijaan arvoa voidaan manipuloida elementin nimen perusteella.

```
public void editElement(String filename, String tag, String tagvalue, String target, String value) throws IOException, JDOMException{
    File xml = new File("xmlFiles/" + filename + ".xml");
    SAXBuilder builder = new SAXBuilder();
    Document saxDoc = builder.build(xml);
    Element root = saxDoc.getRootElement();

    // Haetaan kaikki juurielementin sisällä olevat elementit
    List<Element> children = root.getChildren();
    Iterator itr = children.iterator();

    // Etsitään tarvittava elementti ja päivitetään se
    while(itr.hasNext()){
        Element child = (Element)itr.next();

        if(child.getAttributeValue(tag).equals(tagvalue)){
            child.getAttribute(target).setValue(value);
        }
    }

    saveXML(filename, saxDoc);
} // editElement

private void saveXML(String filename, Document doc) throws FileNotFoundException, IOException{
    String location = "xmlFiles/" + filename + ".xml";

    // Avataan tallennettava tiedosto OutputStreamWriteriin ja tallennetaan tiedosto
    try (OutputStreamWriter output = new OutputStreamWriter(new FileOutputStream(location), "UTF-8")) {
        XMLOutputter xmlOutput = new XMLOutputter();
        xmlOutput.setFormat(Format.getPrettyFormat()); // Tehdään XML-tiedostosta luettavampi lisäämällä whitespacet
        xmlOutput.output(doc, output);
        output.close();
    }
} // saveXML
```

KUVA 16. XML-tiedoston muokkaus

Kun XML-tiedosto on saatu ohjelman muistiin, on tiedoston muokkaus varsin yksinkertaista. Tiedostossa olevia elementtejä voi etsiä usealla eri tavalla, mutta itse käytin pääasiassa elementin sisällään pitämän attribuutin nimen ja arvon. XML-tiedoston elementtejä läpi käytäessä tulee ottaa huomioon, että elementtien läpi käyntiin ei voi käyttää for-silmukkaa, mikäli tiedoston elementtejä aiotaan muokata tai poistaa. Tämä johtuu siitä, että elementin muokkaus tai poistaminen sotkee for-silmukan sen hetkisen sijainnin, jolloin se ei osaa enää siirtyä seuraavaan elementtiin. Tämän takia elementtien läpi käynnin apuna käytetään erillistä iteraattoria (iterator). Iteraattorin etu silmukkaan nähden on se, että se ei tarvitse tarkkaa tietoa siitä missä kohtaa listaa se sijaitsee. Iteraattori määrittelee sijaintinsa suhteessa edelliseen palauttamaansa elementtiin nähden (What is difference between Iterator and for loop 2016).

Seuraava ongelma XML-tiedoston käsittelyssä tulee vastaan, kun tiedostoon halutaan sijoittaa tietoa, joka sisältää ääkkösiä. Tästä syystä XML-tiedostojen elementit ovat englanniksi, koska tällöin vältytään ainakin XML-tiedoston parsinta- ja lukuvaiheissa mahdollisilta virheiltä. Mikäli XML-tiedostossa käytetään ääkkösiä, ne saattavat aiheuttaa ongelmia eri merkistöjen kanssa, sillä osa Javan sisällään pitämistä tiedoston avaamiseen ja tallentamiseen tarkoitetuista luokista käyttää oletuksena tietokoneen oletusmerkistöä, joka on riippuvainen käyttöjärjestelmästä. Koska on mahdollista, että käyttäjä antaa joihinkin asetustietoihin (kuten botin lähettämiin viesteihin) ääkkösiä sisältäviä arvoja, täytyy kaikki kirjoitettava data kääntää UTF-8 -merkistöön. Tämä

voidaan toteuttaa avaamalla tiedosto ensin FileOutputStream-luokan avulla, josta tiedostoon kirjoitetaan OutputStreamWriter-luokan avulla, joka mahdollistaa tallennuksessa käytettävän merkistön määrittelyn (kuva 16).

3.4 REST

RESTin päätarkoitus botin toteutuksessa on tarjota rajapinta kanavan tietojen lukemiseen ja päivittämiseen. RESTin toteutus onnistuu kohtuullisen helposti Javan HttpURLConnection-luokan avulla, mutta kuten matalan tason APIt yleensä, se tarjoaa varsin yksinkertaiset työkalut REST-sovelluksen toteuttamiseen. Javalle on olemassa myös RESTin käyttöön suunniteltuja sovelluskehyksiä, jotka isommissa järjestelmissä helpottaisivat RESTin parissa työskentelyä tarjoamalla valmiita työkaluja rajapinnan toteutukseen ja datan käsittelemiseen useilla eri tiedostoformaateilla. Usealla niistä kuitenkin on huonona puolena se, että ne on joko dokumentoitu epäselvästi tai se, että niiden opetteluun täytyy varata huomattavasti aikaa (Gaic 2015). Koska omassa työssäni RESTiä käytettiin varsin yksinkertaisissa toiminnoissa, päätin toteuttaa REST-ominaisuudet suoraan Javan omilla työkaluilla, ilman raskaan sovelluskehyksen käyttöönottoa. Twitchin REST-rajapinnan palauttaman JSON-datan käsittelyä varten kuitenkin jouduin lisäämään uuden riippuvuuden Maveniin JDOM-arkistoa varten, sillä Javan oletuspaketit eivät sisällä kovin hyviä työkaluja JSONin käsittelyyn.

```
try{
    URL uri = new URL(serviceuri);
    HttpURLConnection conn = (HttpURLConnection) uri.openConnection();

    // Lisätään PUT-parametrit
    conn.setRequestProperty("Accept", "application/vnd.twitchtv.v3.json");
    conn.setRequestProperty("Authorization", "OAuth " + token);
    conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded;charset=UTF-8");
    conn.setRequestProperty("Content-Length", "" + event.getBytes().length);
    conn.setRequestMethod("PUT");
    conn.setDoOutput(true);

    try{
        OutputStream output = conn.getOutputStream();
        Writer writer = new OutputStreamWriter(output, "UTF-8");
        writer.write(event);
        writer.close();
        output.close();
    }catch(IOException error){
        System.out.println(error);
    }
    // Vastaus tallennetaan tähän
    StringBuilder response;

    try{
        BufferedReader rd = new BufferedReader(new InputStreamReader(conn.getInputStream(), "UTF8"));
        response = new StringBuilder();
        String line;

        // Luetaan niin kauan kuin vastaus jatkuu
        while ((line = rd.readLine()) != null) {
            response.append(line);
        }

        json = new JSONObject(response.toString());

        // Suljetaan yhteys
        conn.disconnect();
    }catch(MalformedURLException uriError){
        System.out.println("Error in URI: " + uriError);
    }finally{
        // Varmistetaan, että yhteys katkaistaan
        conn.disconnect();
    }
}catch(IOException | JSONException e){
    System.out.println(e);
}
```

KUVA 17. REST-pyynnön lähetys ja käsittely

REST-pyynnön luonti alkaa yhteyden luomisella kohdeverkkopalveluun. Yhteyden luonnin jälkeen palvelulle lähetetään tunnistukseen ja dataan liittyvät parametrit HTTPn headerin eli otsikkotietojen mukana. Accept-parametrin tarkoitus Twitchin käyttöympäristössä on kohdentaa käytettävän REST APIn versio, sillä Twitch tukee edelleen myös vanhempia API-versioita. Authorization-parametrin tarkoitus on välittää palvelimelle REST-palveluissa usein käytössä oleva API-avain, joka Twitchillä on käyttäjän myöntämä OAuth-token. API-avaimia käytetään pääasiassa turvallisuussyistä. Niiden avulla kyetään tunnistamaan palvelua käyttävä sovellus tai käyttäjä, jonka nimissä toimitaan ilman, että pyynnön yhteydessä täytyy lähettää erillistä käyttäjätunnus-salasana -yhdistelmää, mikä on mahdollinen tietoturvariski (Hazlewood 2014). Content-Type- ja Content-Length -parametrien avulla verkkopalvelulle kerrotaan se, minkälaista dataa se on ottamassa vastaan ja kuinka paljon dataa on tulossa. Tässä tapauksessa pyynnön yhteydessä varmistetaan se, että merkistökoodauksena on UTF-8, jotta ääkköset toimisivat oikein. SetRequestMethod-parametriin annetaan arvoksi API-dokumentaation määrittelemä HTTP-verbi, joka riippuu siitä, mitä datalla halu-

taan tehdä. SetDoOutput-parametri on yksi olennaisimmista parametreista, mikä REST-pyyntöön liitetään. Siihen annettu Boolean-tyyppinen arvo kertoo verkkopalvelulle sen, onko pyynnön yhteyteen liitetty erillisiä parametreja. GET-tyyppisissä REST-pyynnöissä, joiden avulla haetaan tietoa palvelimelta, tämän parametrin arvo voi käytettävästä APIsta riippuen olla false. Kaikki muut HTTP-verbit edellyttävät, että tämä arvo on true eli tosi, sillä muuten palvelu ei käsittele pyynnön yhteydessä lähetettäviä tietoja ei lähetetä. (Kuva 17.)

TAULUKKO 1. HTTP-vastauskoodit ja niiden aiheuttajat

Vastauskoodi	Tapahtuma
1xx	Pyyntö saatu, mutta sitä ei ole vielä käsitelty.
2xx	Pyyntö saatu ja se on suoritettu onnistuneesti.
3xx	Uudelleenohjaus. Palvelu tarvitsee lisää tietoja ennen kuin pyyntö voidaan suorittaa.
4xx	Client-virhe. Pyyntö on joko virheellinen tai sitä ei voida toteuttaa.
5xx	Palvelinpään virhe. Palvelin ei kyennyt toteuttamaan pyyntöä johtuen palvelimen virheestä, vaikka pyyntö olikin oikein muodostettu.

Kun pyynnön otsikkotiedot on määritelty, pyyntöön on aika liittää palvelimelle lähetettävä data, mikäli sitä on olemassa. Tämän jälkeen aiemmin muodostettu yhteys suljetaan ja odotetaan palvelimen vastausta. Palvelin vastaa pyyntöön kolmenumeroisella HTTP-vastauskoodilla sekä REST-pyynnöstä riippuen dataa palauttamalla. HTTP-vastauskoodeja on useita, mutta ne voidaan rajata karkeasti niiden ensimmäisen numeron perusteella (taulukko 1). Mikäli pyyntö on onnistunut täydellisesti, verkkopalvelun vastauskoodin pitäisi olla 200. Muussa tapauksessa joko pyydetyllä resurssilla ei ole ollut dataa, jota tarjota tai sitten pyynnössä itsessään on ollut jotain vikaa. Mikäli kaikki on kuitenkin mennyt oikein, yhteys verkkopalveluun katkaistaan ja sen palauttama data otetaan talteen ja sitä jatkokäsitellään tarpeen mukaan.

3.5 OAuth -tokenien hankinta

Koska botilla on oma Twitch-käyttäjätunnus, tulen tarvitsemaan bottia varten kaksi erillistä OAuth-tokenia. Näistä ensimmäinen hankitaan botin omalta käyttäjätunnukselta ja sitä tarvitaan Twitchin IRC-palvelimelle kirjautumiseen ja siellä suoritettavien

toimintojen toteuttamiseen. Toinen hankitaan siltä käyttäjältä, jonka streamchatissä botin tulee toimia. Tämä token tulee sisältämään enemmän oikeuksia kuin ensimmäinen token, koska botilla pitää olla oikeudet päästä käsiksi kyseisen käyttäjän omiin tietoihin, sekä streamin tietoihin.

TAULUKKO 2. Twitchin OAuth scopet

Scope	**Selite**
channel_check_subscriptions	Lukuoikeus tarkistaa, onko kanava tilattu tietyn käyttäjän toimesta
channel_commercial	Oikeus mainosten käynnistämiseksi
channel_editor	Oikeus muokata kanavan metadataa (streamattava peli, kanavan otsikko)
channel_read	Lukuoikeus kanavan yksityisiin tietoihin, kuten stream-avaimeen ja sähköpostiosoitteeseen
channel_stream	Oikeus nollata käyttäjän stream-avain ja luoda uusi avain vanhan tilalle
channel_subscriptions	Lukuoikeus kanavan tilanneiden käyttäjien listaan
chat_login	Oikeus kirjautua IRC-palvelimelle ja lähettää sinne viestejä käyttäjän puolesta
user_blocks_edit	Lupa estää toinen käyttäjä ja poistaa esto käyttäjän puolesta
user_blocks_read	Lukuoikeus estettyjen käyttäjien listaan
user_follows_edit	Lupa hallinnoida käyttäjän seuraamia kanavia
user_read	Lupa lukea käyttäjän yksityisiä tietoja, kuten sähköpostiosoite
user_subscriptions	Lukuoikeus käyttäjien tilaamien kanavien listaan

Kuten taulukossa 1 näkyy, Twitchillä on useita erilaisia scopeja eli tässä tapauksessa käyttöoikeuden rajoituksia, joiden avulla käyttäjä kykenee rajaamaan niitä oikeuksia, joihin Twitchin APIa käyttävät palvelut pääsevät käsiksi. Ilman näitä erikseen pyydettäviä oikeuksia botilla on oikeus ainoastaan päästä käyttäjän perustietoihin, mutta se ei todellakaan kykene suorittamaan lähes mitään sille tavoitelluista toiminnoista. Mitään rajoituksia pyydettävien oikeuksien määrän suhteen ei ole, vaan botti voisi teoriassa pyytää luvan kaikkia näitä oikeuksia varten. On kuitenkin suositeltavaa pyytää ainoastaan oikeudet niihin scopeihin, joita toteutettavassa palvelussa tullaan tarvitsemaan,

sillä käyttäjä jolta lupa pyydetään, näkee listan häneltä pyydetyistä oikeuksista. (Authentication 2012.)

```
https://api.twitch.tv/kraken/oauth2/authorize
    ?response_type=token
    &client_id=[your client ID]
    &redirect_uri=[your registered redirect URI]
    &scope=[space separated list of scopes]
```

KUVA 18. Scope-oikeuksien URIn rakenne (Authentication 2012)

Koska botin toiminta perustuu siihen, että botti toimii ilman palvelinta, tulen käyttämään tokenin hankinnassa aiemmin mainittua epäsuoraa (Implicit) hankintatapaa. Tämä perustuu siihen, että käyttäjä ohjataan Twitchin API-sivulle, jossa pyydetään oikeuksia haluttuihin scopeihin. Samalla URIn mukana lähetetään tiedot oikeuksia pyytävästä sovelluksesta ja sen tarvitsemista oikeuksista (kuva 18). Tämän jälkeen, mikäli käyttäjä on antanut sovellukselle luvan päästä käsiksi sen pyytämiin tietoihin, käyttäjä ohjataan sille sivustolle, joka annettiin Twitchille sovelluksen rekisteröinnin yhteydessä. Tokenin hankinnasta tulee tämän takia hieman epäkäytännöllinen, koska token palautetaan URIn fragmenttina, eli se on liitetty URIin ”#”-merkillä erotettuna (Authentication 2012). Tämä siis tarkoittaa sitä, että token täytyy poimia URIsta joko Javascriptillä tai poimimalla se selaimen osoitepalkista, jonka jälkeen se syötetään botille erillisen lomakkeen kautta. Itse aion noudattaa jälkimmäistä tapaa ja luoda botille ensimmäisen käynnistyksen yhteydessä näytettävän lomakkeen, jossa pyydetään tällä tavalla noudettua tokenia (liite 1).

TAULUKKO 3. Botin toiminnan kannalta olennaiset scopet

Käyttäjätunnus	**Tarvittavat scopet**
Botin käyttäjätunnus	chat_login
Streamaajan käyttäjätunnus	channel_commercial channel_editor channel_read

Taulukossa 2 näkyy botin toiminnan kannalta olennaiset scopet. Näitä scopeja edellytetään botin toiminnan varmistamiseksi. Scopeja valittaessa on päädytty pitäytymään

kanavan toimintojen suorittamiseen vaadituissa scopeissa, sillä botin toiminnan kannalta käyttäjän tiedoissa ei ole mitään tarpeellista.

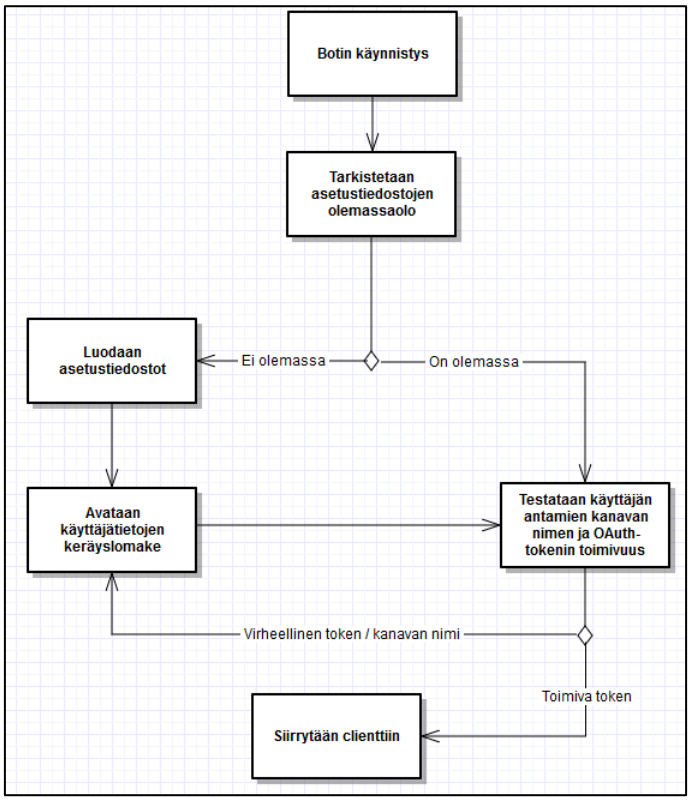
```
// Muuttujat
private final String clientid = "[redacted]";
private final String redirecturi = "http://localhost";
private final String scopes = "chat_login";
private final String authURI = "https://api.twitch.tv/kraken/oauth2/authorize?response_type=token&client_id=" +
                               clientid + "&redirect_uri=" + redirecturi + "&scope=" + scopes + "&force_verify=true";

private void GetTokenButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // Hankitaan OAuth-token ohjaamalla käyttäjä Twitchin API-sivuille

    if(Desktop.isDesktopSupported()){
        try{
            // Aukaistaan URI selaimeen
            Desktop.getDesktop().browse(new URI(authURI));
        }catch(IOException ioe){
            ErrorLabel.setText("Error in input: " + ioe);
        }catch(URISyntaxException urie){
            ErrorLabel.setText("Error in URI. Web site couldn't be opened. " + urie);
        }
    }
}
```

KUVA 19. OAuth-tokenin hankkimiseen tarvittava koodi

Käytännössä OAuth-tokenin hankinta on helppoa ja vaatii ainoastaan vähän koodia (kuva 19). Koodissa rakennetaan ensin kuvan 18 kaltainen URI, johon korvataan oikeat tiedot. Kuvassa esitetty koodi toimii napin painalluksen pohjalta, jolloin ohjelma avaa oletusselaimeen authURI-muuttujasta muodostuvan linkin. Virheiden välttymiseksi koodiin on lisätty vielä tapahtumakäsittelijät virhetilanteiden varalle. Kun käyttäjä hyväksyy Twitchin sivustolla esitetyn pyynnön, Twitch ohjaa käyttäjän automaattisesti redirect_uri-muuttujan osoittamalle sivustolle, josta myönnetty token voidaan poimia joko Javascriptillä tai kopioimalla se selaimen osoiteriviltä.

3.6 Client

Botin ohjaukseen toteutettu client eli graafinen käyttöliittymä on toteutettu Oraclen tarjoamassa Netbeans IDE -ympäristössä. Netbeans tarjoaa erilliset kehitystyökalut, joiden avulla elementtejä pystytään luomaan ja muokkaamaan IDE-ympäristössä ilman, että niitä täytyisi koodata alusta asti itse. IDE-ympäristössä toimiessa täytyy ottaa huomioon se, että käyttöliittymän elementtien rakentamisen nopeus maksetaan myöhemmin käyttöliittymän muokkaamisen kohdalla. Netbeans nimittäin lukitsee kaiken sen koodin, minkä se generoi käyttöliittymän luonnin yhteydessä, joten kaikki käyttöliittymän elementteihin kohdistuva kustomointi täytyy tehdä kehitystyökalujen asetusten kautta.



KUVA 20. Botin käynnistysprosessi

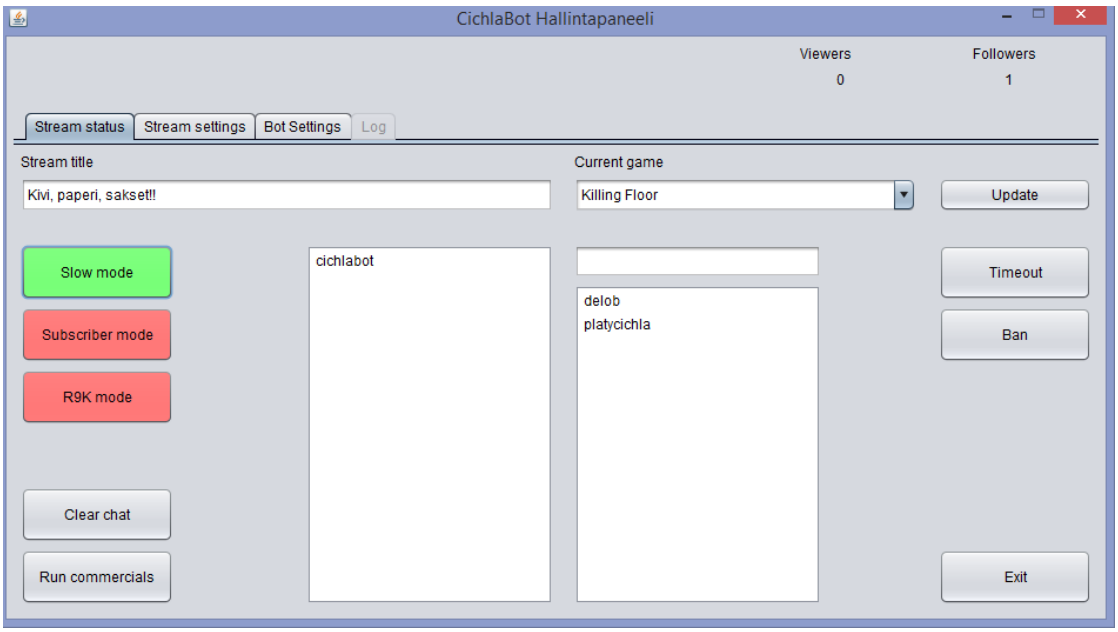
Kuvassa 20 esitellään botin käynnistysprosessissa läpi käytävät vaiheet. Botin käynnistys on toteutettu siten, että asetustiedostot ja niiden sisältämät tiedot käydään ensin lävitse siten, että niiden olemassaolo ja tietojen oikeellisuus varmistetaan. Tällä tavalla vältytään käynnistyksen yhteydessä mahdollisesti tapahtuvilta virheiltä. Clientin toteutuksessa tuli ensimmäisen kerran vastaan Javan kankeus käyttöliittymän mukauttamisessa. Client pitää sisällään useita erilaisia listatyyppisiä elementtejä, johon luetteloidaan muun muassa kanavan chatissä olevat käyttäjät. Tätä listaa on tarkoitus päivittää sitä mukaan kun IRC-palvelin ilmoittaa käyttäjien poistumisesta ja saapumisesta. Listaelementit sisältävät niiden päivittämiseen tarvittavat funktiot, mutta ne eivät ole erityisen vakaita. Kokeilin ensin listojen päivittämistä suoraan funktioiden avulla, jolloin elementin päivittäminen toimi erinomaisesti, kunhan testauksessa toimi kanava, jolla oli vain 100–300 katsojaa. Mikäli kanavalla oli yli seitsemänsataa katsojaa, botti kohtasi virheen ja kaatui. Virhe johtui siitä, että useampi päivitettävä tietue yritti päästä käsiksi samaan elementtiin yhtä aikaa, jolloin elementtiin ei kyetty muodostamaan yhteyttä ja sovellus kaatui. Tämän kiertämiseksi päivitys täytyi tehdä Javan tapahtumatietovirran (EventThread) kautta, jolloin jokainen elementtiin lisättävä tietue pistetään jonoon ja ne muodostavat elementtiin yhteyden yksi kerrallaan.

```
timer.scheduleAtFixedRate(new TimerTask() {
    @Override
    public void run() {
        long timestamp = System.currentTimeMillis() / 1000L;
        for(ArrayList<String> message : messageMap){
            if(messageTimestamps.containsKey(message.get(3))){
                if(message.get(1).equals("interval")){
                    // Tutkitaan onko aikaa kulunut tarpeeksi, jotta viesti voidaan lähettää uudelleen
                    if(timestamp - Long.parseLong(messageTimestamps.get(message.get(3)).toString()) > Long.parseLong(message.get(2))){
                        try{
                            irc.writeToChannel(message.get(3));
                        }catch(Exception e){
                            System.out.println(e);
                        }

                        messageTimestamps.put(message.get(3), timestamp);
                    }
                }
            }else{
                messageTimestamps.put(message.get(3), timestamp);
            }
        }
    }
}, 1*1000, 1*1000);
```

KUVA 21. Ajastettujen viestien lähetys

Botin käynnistymisen jälkeen botti lukee asetustiedostot ja sijoittaa niiden sisällään pitämät tiedot erillisiin muuttujiin. Asetustiedostojen sisällään pitämät tiedot sijoitetaan myös niitä vastaaviin asetuselementteihin tapahtumatietovirran kautta. Asetusten lukemisen jälkeen luodaan kaksi erillistä ajastettua toimintoa Javan Timer-luokan avulla. Näistä harvemmin toistettavan ajastimen tarkoitus on ainoastaan päivittää moderaattorilistaa, pyytämällä päivitettyä listaa palvelimelta. Tämän ominaisuuden tarkoitus on varmistaa, että botilla on edelleen käytössään moderaattorin oikeudet, sekä sijoittaa mahdollisesti nimitetyt uudet moderaattorit oikealle listalle. Useammin toistettavan ajastimen on tarkoitus käydä kerran sekunnissa lävitse kaikki botille määritetyt viestit ja tarkistaa onko sille annetuiden ajastettujen viestien edellisestä lähetyskerrasta kulunut niin kauan aikaa, että ne voidaan lähettää uudestaan (kuva 21).

Botin sydämenä toimii olio, joka muodostaa yhteyden IRC-palvelimelle ja lukee palvelimelta tulevia viestejä. Botin toimintaa ohjataan viestien sisällön ja botin asetusten perusteella. Tämä näkyy erityisesti botin tapahtumalokiin kirjoitettujen viestien kohdalla, joita pystyy rajoittamaan asetusten avulla. Viestien lukemisen yhteydessä myös poimitaan viestin lähettänyt käyttäjä ja asetetaan käyttäjän nimi käyttäjänimilistalle. Tämä tehdään siitä syystä, että palvelimelta tulevat JOIN-viestit tulevat noin kymmenen sekunnin välein, mikä aiheuttaisi viivettä listan päivittämiseen. Listan sisällään pitämistä tiedoista saadaan siis tällä tavalla tarkempia, kun nimiä lisätään automaattisesti viestien saapumisen yhteydessä. Nimiä lisätään listaan myös JOIN-viestien antamien tietojen perusteella, jolloin tarkistetaan se mahdollisuus, että onko esiintyykö kyseinen nimi jo listassa.



KUVA 22. Hallintapaneelin etusivu

Client pitää sisällään useita eri painikkeita ja lomakekenttiä, joiden avulla kanavan tietoja ja chatin tilaa pystyy muokkaamaan. Näistä painikkeista olennaisimmat ovat chatin tilaan vaikuttavat kolme painiketta, jotka vaihtavat automaattisesti väriä sen mukaan, mitkä Twitchin tarjoamista tiloista ovat sillä hetkellä aktiivisena. Slow mode -tilalla tarkoitetaan tilaa, jolloin chatissa olevat käyttäjät voivat kirjoittaa uuden viestin ainoastaan tietyin väliajoin. Oletusviive tälle tilalle on 120 sekuntia, mutta tähän voi vaikuttaa botin asetuksia mukauttamalla. Subscriber mode on tila, jolloin kanavalla pystyvät keskustelemaan ainoastaan moderaattorit sekä kanavan tilanneet ja sitä rahallisesti kuukausittain tukevat käyttäjät. R9K mode on tila, jolloin viesteissä täytyy olla vähintään yhdeksän merkin eroavaisuus aikaisempaan viestiin nähden. Tämä tila on vielä Twitchin mukaan betassa, mutta sen käyttötarkoituksena on todennäköisesti estää spämmiä. (Kuva 22.)

Muita hallintapaneelissa olennaisia elementtejä ovat kaksi lomakekenttää, jotka pitävät sisällään kanavan sen hetkisen otsikon ja pelattavan pelin nimen. Otsikkokenttä on aivan tavallinen tekstikenttä, johon käyttäjä voi antaa täysin vapaamuotoisen tekstin. Pelin nimen sisällään pitävä kenttä sen sijaan on muokattava alas vedettävä valikko, johon voi laittaa haluamansa pelin nimen, mutta joka myös tarjoaa vaihtoehdoiksi listan 100sta botin käynnistyshetkellä pelatuimmasta pelistä. Lista näistä peleistä noudetaan REST-pyynnön avulla.

Kuten aiemmin mainittiin, botti pitää sisällään kaksi erillistä listaa käyttäjistä. Ensimmäinen listoista pitää sisällään ainoastaan paikalla olevista chatin moderaattoreista. Toinen lista pitää sisällään tiedot paikalla olevista tavallisista käyttäjistä. Tätä listaa voidaan suodattaa listan yläpuolella sijaitsevaan tekstikenttään. Tämän toiminnon toteuttaminen vaati sen, että kyseiselle tekstikentälle täytyi asettaa erillinen DocumentListener eli tapahtumakäsittelijä, joka reagoi jokaiseen elementtiin tapahtuneeseen muutokseen ja suorittaa muutoksen tapahtuessa halutun toiminnon. Suodattamisen tarkoituksena on helpottaa timeout- ja ban-painikkeiden käyttämistä, jotka tarvitsevat toimiakseen sen, että käyttäjälistasta on valittu kohde, joka halutaan timeoutata eli estää kohdekäyttäjältä oikeus keskustella kanavalla väliaikaisesti tai bannata eli poistaa käyttäjä kanavalta pysyvästi.

Viimeiset painikkeet ovat tarkoitettu chatin tyhjentämiseen ja mainosten ajamista varten. Chatin tyhjentämisellä tarkoitetaan sitä, että painiketta painamalla chatti tyhjennetään täysin siihen lähetetyistä viesteistä. Chatin tyhjennystä käytetään yleensä merkkinä siitä, että keskustelu on ajautunut asiattomalle polulle ja streamaaja tai moderaattorit haluaisivat sen palautuvan taas asialliseen aihepiiriin. Mainosten ajamiseen Twitch on asettanut astetta suuremmat vaatimukset. Ensimmäinen vaatimus on se, että botilla täytyy olla kanavan muokkaajan eli editorin käyttäjäoikeudet ennen kuin se voi ajaa mainokset. Tämä pitää tehdä erikseen Twitchin tarjoaman hallintapaneelin (dashboard) kautta. Toinen vaatimus on asetettu kanavan suosioon liittyen. Streamaajan täytyy olla Twitchin yhteistyökumppani, jota varten kanavan täytyy olla tarpeeksi suosittu. Vaatimuksia ovat muun muassa ne, että kanavan täytyy saada jatkuvasti vähintään 500 katsojaa ja että streamaaja streamaa säännöllisesti vähintään kolmena päivänä viikossa (Partner Application 2016). Tästä syystä tätä toimintoa on testattu ainoastaan virheilmoitusten avulla. Mainosten ajamiseen tarkoitettu REST-pyyntö palauttaa tietynlaisen virheilmoituksen riippuen siitä, onko pyyntö onnistunut vai onko käyttäjällä edes oikeutta ajaa mainoksia. Tästä syystä kykenin toteamaan painikkeen toimivuuden ilman sellaisen testikanavan hankkimista, joka olisi Twitchin yhteistyökumppani.

Streamin asetukset sisällään pitävä välilehti koostuu ainoastaan kolmesta listasta, joita varten on luotu myös painikkeet ja tekstikentät. Painikkeiden toiminta riippuu siitä, mikä elementti on ollut aktiivisena viimeksi (eli kumpaa on klikattu viimeksi). Mikäli edellisenä aktiivisena on ollut tekstikenttä, painike poimii tekstikentän sisällön ja lisää sen tekstikentän alapuolella olevaan listaan (olettaen ettei kyseinen tietue ole jo listas-

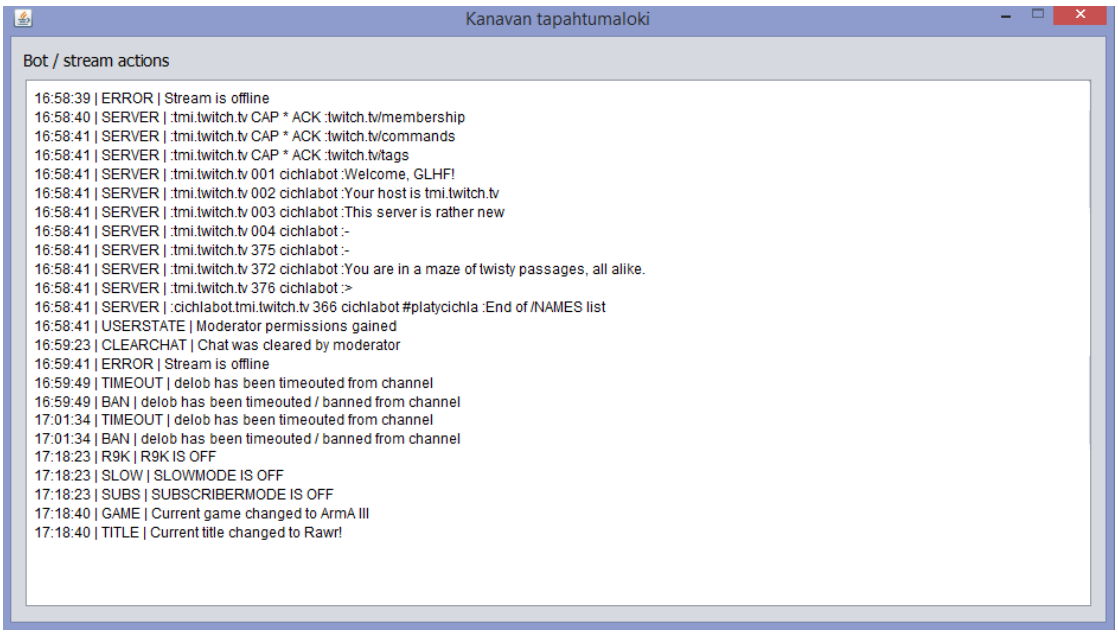
sa). Mikäli edellisenä aktiivisena elementtinä on ollut lista, painike ottaa ylös listassa valittuna olevan tietueen ja poistaa sen listasta ja siitä XML-tiedostosta, johon se on sijoitettu.

Botin asetukset näyttävä välilehti sisältää useita eri lomakekenttiä, joissa olevat tiedot otetaan ylös ja päivitetään asetustiedostoon tallennuspainikkeen painalluksen yhteydessä. Osa botin asetuksista päivittyvät vasta uudelleenkäynnistyksen yhteydessä, kuten loki-ikkunan aukioloon liittyvä asetus. Asetusvälilehti pitää sisällään myös taulukon botin sisällään pitämistä komennoista. Tähän taulukkoon tapahtuvat muutokset tehdään taulukon alapuolella sijaitsevan lomakkeen kautta ja ne otetaan käyttöön suoraan.

3.7 Tulokset

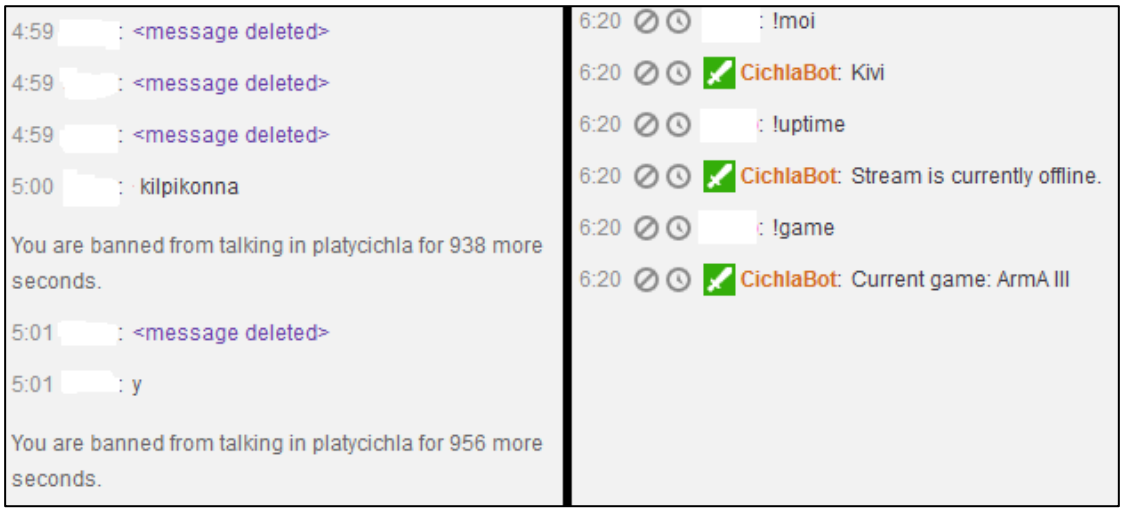
Botin toteutuksen aikana jouduin toteamaan muutamaan otteeseen, että joitain botille asetetuista tavoitteista ei pysty toteuttamaan sellaisenaan tai niitä ei ole järkevää toteuttaa. Tästä syystä botin lopullisessa versiossa on tapahtunut muutoksia luvussa 3.1 annettuihin tavoitteisiin ja liitteessä 1 oleviin rautalankamalleihin nähden. Esimerkkeinä toteutuksen aikana tapahtuneista muutoksista ovat muun muassa linkkien estäminen chatistä ja asetusten aukeaminen omaan ikkunaansa. Linkkien chatistä estäminen poistui toteutuksesta pääasiasta siksi, että Twitch tarjoaa kanavan asetuksissa vaihtoehdon, joka estää kaikki linkit automaattisesti, jolloin vastaavan ominaisuuden toteuttaminen botille jäi tarpeettomaksi. En luonut asetuksille omaa erillistä ikkunaa sen takia, että ne veisivät liikaa tilaa näytöltä ja asetustietojen jatkuvasti esillä pitäminen ei ole tarpeellista.

Botin suunnitteluvaiheessa toteutetut rautalankamalleja vastaavat botin lopullista versiota kohtuullisen tarkasti. Joitain poikkeuksia jouduttiin tekemään toteutuksessa tapahtuneiden muutosten takia. Näkyvin muutos tapahtui Stream settings -välilehden toteutuksen yhteydessä, josta jätettiin täysin pois moderaattorien hallintaan liittyvät toiminnat. Tähän päädyttiin siitä syystä, että moderaattoreita voi nimetä kanavalle ainoastaan streamaajan eli kanavanhaltijan käyttäjätunnuksen kautta. Botilla ei siis ollut tarvittavia oikeuksia moderaattorien hallinnan toteuttamiseksi. Myöskään tapahtumaloki ei vastaa täysin sille suunniteltua ulkoasua, koska se ei ollut tarpeellinen toimivan lopputuloksen aikaansaamiseksi.



KUVA 23. Tapahtumaloki

Kuvassa 23 näkyy botin tapahtumalokin otos siitä, että botti suorittaa sille annettuja komentoja. Viestit, joissa esiintyy ilmoitus SERVER, ovat palvelimen lähettämiä viestejä onnistuneeseen kanavalle liittymiseen liittyen. CLEARCHAT-ilmoitus tapahtuu chatin tyhjennyksen yhteydessä, joka on tapahtunut joko toisen paikalla olevan moderaattorin tai botin napin painalluksesta. Kuvassa näkyvä ERROR-ilmoitus kertoo siitä, että botti ei ole kyennyt suorittamaan jotain sille annettua toimintoa. Muita mahdollisia ERROR-ilmoituksia on useita, kuten asetusten tallentamisessa tai REST-pyyntöjen yhteydessä tapahtuvat virheilmoitukset. TIMEOUT-ilmoitus kirjataan botin suorittaman timeoutin yhteydessä, jonka alapuolella oleva BAN-ilmoitus kertoo siitä, että palvelin on vastaanottanut ilmoituksen ja toteuttanut botin sille antaman käskyn. Loput ilmoituksista liittyvät joko kanavan otsikkoon tai peliin tehtyihin muutoksiin tai chatin tilamuutoksiin.



KUVA 24. Botin viestitapahtumat

Botin avulla voi asettaa rajoituksia ja mahdollisuuksia sen suhteen minkälaisia viestejä käyttäjät voivat kirjoittaa chattiin. Rajoituksilla tarkoitetaan kiellettyjä sanoja, joihin botti reagoi. Niiden seurauksena on väliaikainen timeout tai ban riippuen käyttäjän tekemistä asetuksista. Käyttäjä kykenee muun muassa määrittelemään sen, että toistuva sääntöjen rikkominen kasvattaa annettavan timeoutin kestoa. Mahdollisuudet sen sijaan ovat botin käyttäjän määrittelemiä komentoja, jotka alkava huutomerkillä. Botti vastaa annettuun komentoon sille määritellyllä viestillä. Näihin komentoihin voi myös asettaa oman viiveen, jonka aikana botti ei reagoi katsojien antamiin komentoihin lainkaan. Tämän tarkoituksena on välttää se tilanne, jossa suuri joukko käyttäjiä kykenisi hidastamaan botin toimintaa viestikomentojen avulla.

4 PÄÄTÄNTÖ

Opinnäytetyön tavoitteena oli kehittää botti, joka valvoo Twitchin chattia epäasiallisten viestien varalta ja samalla tarjota graafinen käyttöliittymä, jonka kautta streamaaja kykenee hallinnoimaan streamkanavaansa ja botin toimintaa yksinkertaisesti. Aihe valittiin oman kiinnostuksen pohjalta. Opinnäytetyön alussa tutustuttiin botin toteutuksen kannalta olennaisiin aiheisiin, kuten XML, Web API, REST, OAuth ja IRC.

Opinnäytetyölle asetetut toiminnalliset vaatimukset toteutuivat lähestulkoon täysin. Täyttämättä jääneet vaatimukset johtuivat pääasiassa siitä, ettei botille kyetty myöntämään tarpeeksi korkeita käyttäjäoikeuksia tai sitten ominaisuuksien kannalta tarpeelliseen tietoon ei päästy käsiksi. Opinnäytetyön teoriaosuuteen tarvittavien tietojen hankinta onnistui helposti.

Opinnäytetyön aikana haasteita tuottivat käytettävien tekniikoiden kankeus, mutta niihin tottui nopeasti. Java ohjelmointikielenä taipui varsin heikosti sellaisen graafisen käyttöliittymän suunnitteluun, jossa liikkuu paljon muokattavia ja lennosta luotuja elementtejä. Muuten Java ohjelmointikielenä toimi ihan hyvin. Javan vahvuutena oli esimerkiksi olemassa olevien ohjeiden määrä. Sellaisia ongelmia ei löytynyt, mihin ei olisi löytynyt internetin avulla vastausta. Ohjeita joutui tosin mukauttamaan aina hieman omaan tapaukseen soveltuvaksi. Javan verkkoyhteystekniikkoja olivat kohtuullisen yksinkertaisia ja helppoja ymmärtää.

Tiedon varastoimisen osalta jouduin toteamaan XMLn hieman epäkäytännölliseksi sen takia, että lähes jokaista asetustiedostoa varten täytyi luoda oma luku- ja muokkauskoodinsa, sillä pieninkin eroavaisuus vaikutti tiedoston käsittelyyn. Tämä hidasti botin työstämistä omaan makuun hieman liian paljon. Se kuitenkin ajoi tehtävän sille antamat vaatimukset.

Väittäisin kuitenkin, että opinnäytetyön lopputuloksena oli botti, joka kykenee suoriutumaan aloittelevan tai keskisuuren streamaajan chatin valvomisesta. Se kykenee suoriutumaan sille asetetuista vaatimuksista ja pienemmillä kanavilla ei yleensä ole liian suuria vaatimuksia valvonnalle, joten botin pitäisi kyetä niistä helposti. Kanavien katsojamäärien kasvaessa suosittelisin käyttämään palvelinpohjaisia chatbotteja, sillä tällöin bottia varten varastoitavaa dataa pystyy hallinnoimaan paljon helpommin tietokantojen avulla.

Botin jatkokehityksen osalta sanoisin, että tekisin botista ensin palvelinpohjaisen version. Tällöin asetusten ja muun datan varastointiin voisi käyttää tietokantaa ja käyttöliittymän voi tehdä selainpohjaiseksi, jolloin sen mukauttaminen olisi huomattavasti yksinkertaisempaa. Uusia mahdollisesti kehitettäviä ominaisuuksia kuitenkin olisivat esimerkiksi arvontojen tai äänestysten mahdollistavat työkalut, joita esiintyy muissakin tarjolla olevissa boteissa.

LÄHTEET

Anicas, Mitchell 2014. An Introduction to OAuth 2. WWW-dokumentti. https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2. Päivitetty 21.7.2014. Luettu 14.2.2016.

Apache 2016. Maven – Introduction. WWW-dokumentti. https://maven.apache.org/what-is-maven.html. Päivitetty 15.3.2016. Luettu 3.4.2016.

Authentication. 2012. Twitch. WWW-dokumentti. https://github.com/justintv/Twitch-API/blob/master/authentication.md. Päivitetty 1.9.2015. Luettu 14.2.2016.

Bashore, Andrew 2016. A Guide to Twitch ”IRC”. WWW-dokumentti. http://blog.bashtech.net/a-guide-to-twitch-irc/. Päivitetty 2016. Luettu 29.3.2016.

Beach, Chris 2013. What is an API? WWW-dokumentti. https://www.quora.com/What-is-an-API-4. Päivitetty 24.4.2013. Luettu 2.3.2016.

Binkowski, Brooke 2016. The Saga of Twitter Bot Tay. WWW-dokumentti. http://www.snopes.com/2016/03/25/the-saga-of-twitter-bot-tay/. Päivitetty 25.3.2016. Luettu 28.3.2016.

Bradley, Tony 2014. What is a Bot (or Zombie)? WWW-dokumentti. http://netsecurity.about.com/od/frequentlyaskedquestions/qt/pr_bot.htm. Päivitetty 15.12.2014. Luettu 6.3.2016.

BrainJet 2014. 20 Awesome Twitter Bots You Aren’t Following. WWW-dokumentti. http://www.brainjet.com/pop-culture/3816/20-awesome-twitter-bots-you-arent-following/#slide/0/0. Päivitetty 17.9.2014. Luettu 28.3.2016.

Business Logic. 2016. Techopedia. WWW-dokumentti. https://www.techopedia.com/definition/27382/business-logic. Ei päivitystietoa. Luettu 29.3.2016.

Caraballo, David & Lo, Joseph 2014. The IRC Prelude. WWW-dokumentti. http://www.irchelp.org/irchelp/new2irc.html. Päivitetty 2014. Luettu 16.2.2016.

Class HttpUrlConnection. 2016. Oracle. WWW-dokumentti. https://docs.oracle.com/javase/8/docs/api/. Päivitetty 12.1.2016. Luettu 29.2.2016.

Davis, Thomas 1999. Logging on to Internet Relay Chat (IRC). WWW-dokumentti. http://www.javaworld.com/article/2076502/core-java/logging-on-to-internet-relay-chat--irc-.html. Päivitetty 20.10.1999. Luettu 29.12.2015.

Different Types of Internet Bots and How They Are Used. 2016. Spamlaws. WWW-dokumentti. http://www.spamlaws.com/how-internet-bots-are-used.html. Ei päivitystietoa. Luettu 6.3.2016.

Elkstein, M 2008a. What is Rest? WWW-dokumentti. http://rest.elkstein.org/2008/02/what-is-rest.html. Päivitetty 9.2.2008. Luettu 17.2.2016.

Elkstein, M 2008b. Using REST in Java. WWW-dokumentti. http://rest.elkstein.org/2008/02/using-rest-in-java.html. Päivitetty 2008. Luettu 29.2.2016.

Ellis, John 2015. Good Bot, Bad Bot, Ugly Bot. Battle of the Bots! PDF-dokumentti. http://www.rsaconference.com/writable/presentations/file_upload/tta-r08-good-bot-bad-bot-ugly-bot-battle-of-the-bots.pdf. Päivitetty 23.7.2015. Luettu 28.3.2016.

Events vs. Trees. 2004. SourceForge. WWW-dokumentti. http://sax.sourceforge.net/event.html. Päivitetty 27.4.2004. Luettu 2.3.2016.

Fielding, Roy 2002. Representional State Transfer (REST). WWW-dokumentti. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. Päivitetty 16.3.2002. Luettu 20.3.2016.

Fowler, Martin 2010. Richardson Maturity Model. WWW-dokumentti. http://martinfowler.com/articles/richardsonMaturityModel.html. Päivitetty 18.3.2010. Luettu 20.3.2016.

Gaffan, Marc 2012. What Google doesn't show you: 31% of website traffic can harm your business. WWW-dokumentti. https://www.incapsula.com/blog/what-google-doesnt-show-you-31-of-website-traffic-can-harm-your-business.html. Päivitetty 29.3.2012. Luettu 7.3.2016.

Gai, Dragan 2015. Top 8 Java RESTful Micro Frameworks. WWW-dokumentti. http://www.gajotres.net/best-available-java-restful-micro-frameworks/. Päivitetty 9.7.2015. Luettu 18.4.2016.

Gosling, James 2013. History of Java. WWW-dokumentti. http://www.freejavaguide.com/history.html. Päivitetty 29.10.2013. Luettu 28.2.2016.

Gosnell, Denise 2005. Professional Web APIs : Google, eBay, Amazon.com, MapPoint, FedEx. Indianapolis: Wiley Publishing, Inc.

Hammer, Eran 2012. OAuth 2.0 and the Road to Hell. WWW-dokumentti. http://hueniverse.com/2012/07/26/oauth-2-0-and-the-road-to-hell/. Päivitetty 26.7.2012. Luettu 15.2.2016.

Harvey, Robert 2014. What really is the ”business logic”? WWW-dokumentti. http://programmers.stackexchange.com/questions/234251/what-really-is-the-business-logic. Päivitetty 31.3.2014. Luettu 29.3.2016.

Hazlewood, Les 2014. Top Six Reasons to Use API Keys (and How!). WWW-dokumentti. https://stormpath.com/blog/top-six-reasons-use-api-keys-and-how/. Päivitetty 2014. Luettu 18.4.2016.

Heikkiniemi, Ville 2000. Spanning Tree. PDF-dokumentti. http://heikkiniemi.fi/files/Spanning_Tree.pdf. Päivitetty 1.1.2000. Luettu 16.2.2016.

History of Java programming language. 2013. Free Java Guide. WWW-dokumentti. http://www.freejavaguide.com/history.html. 28.10.2013. Luettu 28.2.2016.

iMasters Expert 2015. REST Architecture Model: Definition, Constraints and Benefits. WWW-dokumentti. http://imasters.expert/rest-architecture-model-definition-constraints-benefits/. Päivitetty 9.1.2015. Luettu 20.3.2016.

Internet Bot. 2016. Techopedia. WWW-dokumentti. https://www.techopedia.com/definition/24063/internet-bot. Ei päivitystietoa. Luettu 5.3.2016.

IRC. 2016. Computer Hope. WWW-dokumentti. http://www.computerhope.com/jargon/i/irc.htm. Päivitetty 1.1.2016. Luettu 15.2.2016.

Kalali, Masoud & Mehta, Bhakti 2013. Developing RESTful Services with JAX-RS 2.0, WebSockets, and JSON. Birmingham: Packt Publishing Ltd.

Kearn, Martin 2015. Introduction to REST and .net Web API. WWW-dokumentti. https://blogs.msdn.microsoft.com/martinkearn/2015/01/05/introduction-to-rest-and-net-web-api/. Päivitetty 5.1.2015. Luettu 29.3.2016.

Kerr, Dara 2013. Bots now running the Internet with 61 percent of Web traffic. WWW-dokumentti. http://www.cnet.com/news/bots-now-running-the-internet-with-61-percent-of-web-traffic/. Päivitetty 12.12.2013. Luettu 5.3.2016.

Kirby, Mark 2013. Creating an efficient REST API with HTTP. WWW-dokumentti. http://mark-kirby.co.uk/2013/creating-a-true-rest-api/. Päivitetty 16.1.2013. Luettu 20.3.2016.

Kramer, Doug 1996. Package java.net. WWW-dokumentti. http://web.mit.edu/java_v1.0.2/www/apibook/javaf.htm. Päivitetty 22.4.1996. Luettu 29.2.2016.

Kumar, Avishek 2015. What is Java? A Brief History about Java. WWW-dokumentti. http://www.tecmint.com/what-is-java-a-brief-history-about-java/. Päivitetty 7.7.2015. Luettu 28.2.2015.

Louvel, Jerome 2013. How much REST should your web API get? WWW-dokumentti. http://restlet.com/blog/2013/05/02/how-much-rest-should-your-web-api-get/. Päivitetty 5.2.2013. Luettu 20.3.2016.

Lublinsky, Boris 2010. Three levels of the REST Maturity Model. WWW-dokumentti. http://www.infoq.com/news/2010/03/RESTLevels. Päivitetty 24.3.2010. Luettu 20.3.2016.

McLaughlin, Brett 2001. Java & XML. Helsinki: Gummerus.

Means, Scott & Bodie, Michael 2002. Book of SAX: The Simple API for XML. San Fransisco: No Starch Press, Inc.

Michel, Jason 2013. Web Service APIs and Libraries. Chicago: American Library Association.

Miessler, Daniel 2016. The Difference Between URLs and URIs. WWW-dokumentti. https://danielmiessler.com/study/url-uri/. Ei päivitystietoa. Luettu 20.3.2016.

Mission. 2012. JDOM. WWW-dokumentti. http://www.jdom.org/mission/index.html. Päivitetty 30.4.2012. Luettu 2.3.2016.

Myer, Tom 2005. A Really, Really, Really Good Introduction to XML. WWW-dokumentti. http://www.sitepoint.com/really-good-introduction-xml/. Päivitetty 24.8.2014. Luettu 27.2.2016.

Obasanjo, Dare 2003. Understanding XML. WWW-dokumentti. https://msdn.microsoft.com/en-us/library/aa468558.aspx. Päivitetty 2003. Luettu 27.2.2016.

Oikarinen, Jarkko & Reed, Darren 1993. Internet Relay Chat Protocol. WWW-dokumentti. http://tools.ietf.org/html/rfc1459. Päivitetty 1993. Luettu 15.2.2016.

Oracle 2015. Reading from and Writing to a URLConnection. WWW-dokumentti. https://docs.oracle.com/javase/tutorial/networking/urls/readingWriting.html. Päivitetty 14.9.2015. Luettu 28.2.2016.

Palmer, Dan 2015. Your API is not RESTful. WWW-dokumentti. https://www.danpalmer.me/blog/your-api-is-not-restful. Päivitetty 4.4.2015. Luettu 20.3.2016.

Partner Application. 2016. Twitch. WWW-dokumentti. https://www.twitch.tv/partner/signup. Ei päivitystietoja. Luettu 8.5.2016.

Patterson, Michael 2015. What Is an API, and Why Does It Matter? WWW-dokumentti. http://sproutsocial.com/insights/what-is-an-api/. Päivitetty 3.4.2015. Luettu 2.3.2016.

Proffit, Brian 2013. What APIs Are And Why They're Important. WWW-dokumentti. http://readwrite.com/2013/09/19/api-defined. Päivitetty 19.9.2013. Luettu 2.3.2016.

RESTful Web Services - Statelessness. 2016. TutorialsPoint. WWW-dokumentti. http://www.tutorialspoint.com/restful/restful_statelessness.htm. Ei päivitystietoja. Luettu 20.3.2016.

Richer, Justin 2016. User authentication with OAuth 2.0. WWW-dokumentti. http://oauth.net/articles/authentication/. Ei päivitystietoa. Luettu 14.2.2016.

Riddel, Jamie 2016. 5 Benefits of using an API in your business. WWW-dokumentti. http://www.digitaltomorrowtoday.com/blog/5-benefits-of-using-an-api-in-your-business. Päivitetty 6.3.2016. Luettu 29.3.2016.

Rosen, Alex 2002. When to use SAXBuilder or DOMBuilder? WWW-dokumentti. http://www.jdom.org/pipermail/jdom-interest/2002-July/010165.html. Päivitetty 2.7.2002. Luettu 1.4.2016.

Rouse, Margaret & Norman, Frank 2005. URI (Uniform Resource Identifier). WWW-dokumentti. http://searchsoa.techtarget.com/definition/URI. Päivitetty 2005. Luettu 20.3.2016.

Rouse, Margaret & Sullivan, John 2008. What is client/server (client/server model, client/server architecture). WWW-dokumentti. http://searchnetworking.techtarget.com/definition/client-server. Päivitetty 2008. Luettu 16.2.2016.

Rouse, Margaret 2014. XML (Extensible Markup Language). WWW-dokumentti. http://searchsoa.techtarget.com/definition/XML. Päivitetty 2014. Luettu 28.6.2015.

Rouse, Margaret, Sargent, Moriah & Linthicum, David 2014. REST (representational state transfer). WWW-dokumentti. http://searchsoa.techtarget.com/definition/REST. Päivitetty 2014. Luettu 17.2.2016.

Ruokonen, Anna 2010. REST ja WADL. WWW-dokumentti. http://www.cs.tut.fi/kurssit/OHJ-5201/materiaali/9.pdf. Päivitetty 8.6.2010. Luettu 17.2.2016.

Sandoval, Jose 2009. RESTful Java Web Services : Master Core REST Concepts and Create RESTful Web Services in Java. Birmingham: Packt Publishing Ltd.

Silmaril Consultants 2014. The XML FAQ – Frequently-Asked Questions about the Extensible Markup Language. WWW-dokumentti. http://xml.silmaril.ie/parsers.html. Päivitetty 14.8.2014. Luettu 27.2.2016.

Stenberg, Daniel 2011. History of IRC (Internet Relay Chat). WWW-dokumentti. https://daniel.haxx.se/irchistory.html. Päivitetty 29.3.2011. Luettu 15.2.2015.

Twitch IRC. 2013. Twitch. WWW-dokumentti. https://github.com/justintv/Twitch-API/blob/master/IRC.md. Päivitetty 19.12.2015. Luettu 29.12.2015.

Twitter 2016. OAuth FAQ. WWW-dokumentti. https://dev.twitter.com/oauth/overview/faq. Ei päivitystietoa. Luettu 14.2.2016.

van Loon, Rober & Lo, Joseph 2004. An IRC Tutorial. http://www.irchelp.org/irchelp/irctutorial.html. Päivitetty 17.8.2004. Luettu 16.2.2016.

Vesterholm, Mika & Kyppö, Jorma 2008. Java-ohjelmointi. Helsinki: Talentum.

Voon, Claire 2016. A Wiki for All the Internet's Bots. WWW-dokumentti. http://hyperallergic.com/280055/a-wiki-for-all-the-internets-bots/. Päivitetty 8.3.2016. Luettu 28.3.2016.

What are Web Services? 2016.TutorialsPoint. WWW-dokumentti. http://www.tutorialspoint.com/webservices/what_are_web_services.htm. Päivitetty 19.3.2016. Luettu 20.3.2016.

What is a Socket? 2016. TutorialsPoint. WWW-dokumentti. http://www.tutorialspoint.com/unix_sockets/what_is_socket.htm. Ei päivitystietoja. Luettu 2.5.2016.

What is a URL? 1997. University of Pennsylvania. WWW-dokumentti. https://www.cis.upenn.edu/~bcpierce/courses/629/papers/Java-tutorial/networking/urls/definition.html. Päivitetty 10.9.1997. Luettu 20.3.2016.

What is difference between Iterator and for loop. 2016. AllInterview. WWW-dokumentti. http://www.allinterview.com/showanswers/61346/what-is-difference-between-iterator-and-for-loop.html. Ei päivitystietoa. Luettu 30.4.2016.

What is the Richardson Maturity Model? 2016. The RESTful cookbook. WWW-dokumentti. http://restcookbook.com/Miscellaneous/richardsonmaturitymodel/. Päivitetty 26.1.2016. Luettu 20.3.2016.

What's a Bot? 2016. Norton. WWW-dokumentti. http://us.norton.com/cybercrime-bots. Ei päivitystietoa. Luettu 6.3.2016.

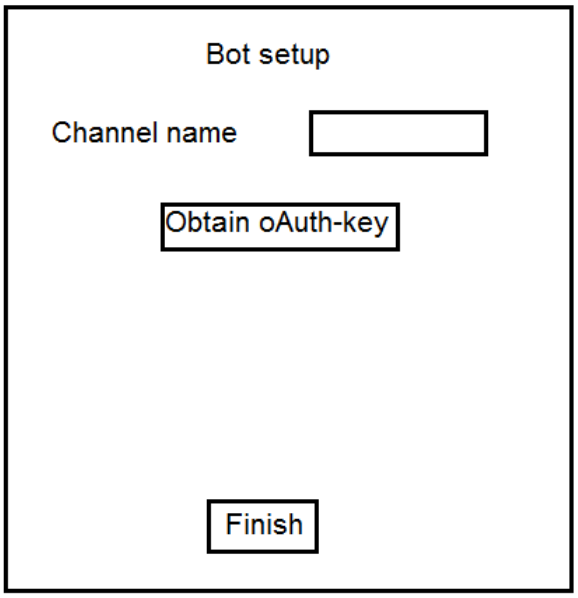
Wiesehan, Robert 2015. Run a Better Twitch Stream With These Tools. WWW-dokumentti. http://www.makeuseof.com/tag/run-a-better-twitch-stream-with-these-tools/. Päivitetty 30.3.2015. Luettu 28.3.2016.

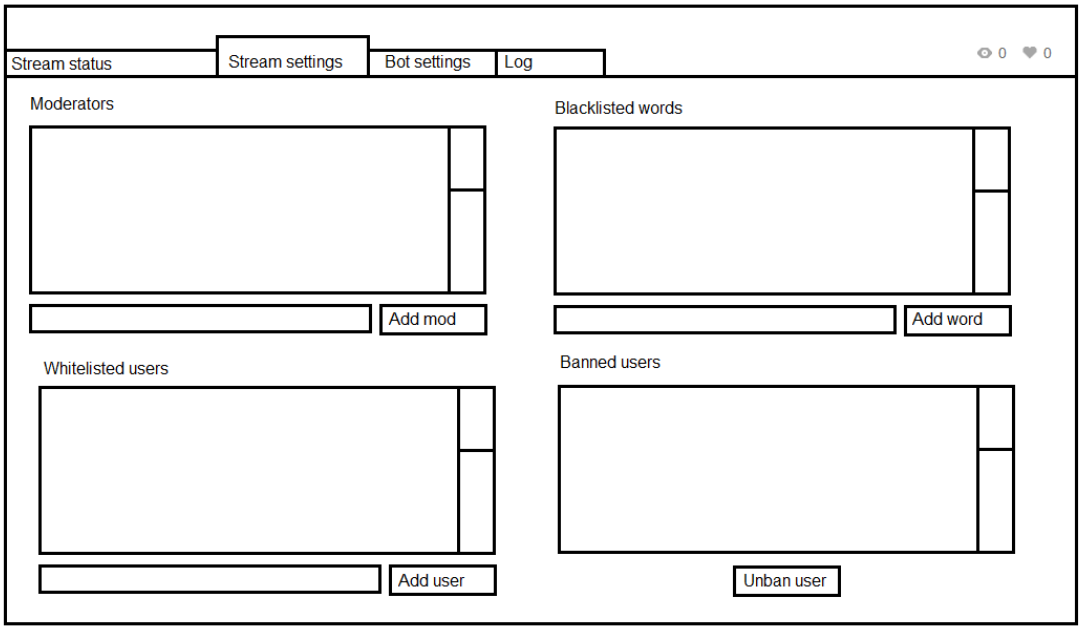
Wulf, 2012. Wireframes: A Great Way to Start Development Projects. WWW-dokumentti. http://www.infoq.com/articles/wireframes-start-development-projects. Päivitetty 7.8.2012. Luettu 4.3.2016.

XML Parser. 2016. Stylus studio. WWW-dokumentti. http://www.stylusstudio.com/xml/parser.html. Ei päivitystietoa. Luettu 27.2.2016.

LIITE 1(1).

Botin rautalankamallit





LIITE 1(2).

Botin rautalankamallit