Tampereen ammattikorkeakoulu

RabbitMQ-jonojen käyttäminen Android sovelluksessa

Niko Ohvo

OPINNÄYTETYÖ
Toukokuu 2019

Tieto- ja viestintätekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

OHVO NIKO:
RabbitMQ-jonojen käyttäminen Android- sovelluksessa

Opinnäytetyö 28 sivua
Toukokuu 2019

Työn tavoitteena oli perehtyä RabbitMQ-jonojen toimintaan ja Android-sovelluskehitykseen luomalla mobiilisovellus, joka ottaa RabbitMQ-palvelimelta vastaan URL-osoitteen sisältäviä viestejä ja vastaanotetuista osoitteista ladata HTML- tai Mp4-sisältö, jota näytetään mobiililaitteen näkymässä. Sovellus toteutettiin Android Studiolla ja sen testaamiseen käytettiin Android 4.2.2-, 4.4.4- ja 6.0.1-käyttöjärjestelmän omaavilla tableteilla.

Työn teoriaosiossa käsiteltiin RabbitMQ:n yleistä toimintaa ja peruskäsitteitä, sekä Android-sovelluskehityksessä käytettäviä Java- ja XML-kieliä. Tämän lisäksi teoriaosioissa esiteltiin yleiskuvaus sovelluksen lopullisesta toiminnallisuudesta.

Käytännön osuudessa käytiin läpi RabbitMQ-palvelimen pystyttäminen ja Android-sovelluksen rakennuksen tärkeimmät ohjelmointiratkaisut, joihin sisältyvät näkymien ja aktiviteettien luominen, palvelimelta viestien vastaanottaminen ja viestin käsittely lataamalla viestistä saadusta osoitteesta sisältö. Näiden lisäksi käytiin myös läpi lokaalin palvelimen isännöinti Python 3.7.2-kielellä.

Pohdintaosuudessa käytiin läpi työn tulokset ja mietittiin jatkokehityksen kannalta mahdollisia tulevaisuuden suunnitelmia ja käyttömahdollisuuksia sovellukselle.

Asiasanat: RabbitMQ, Android, Java, XML, mobiilikehitys

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Information and Communication Technologies
Software Engineering

OHVO NIKO:
Usage of RabbitMQ-queues in an Android application

Bachelor's thesis 28 pages
May 2019

The aim of the thesis was to get familiar with the functionality of RabbitMQ-queues and Android application development by creating a mobile application that receives messages containing URL-addresses from a RabbitMQ-server and downloads HTML- or Mp4-content from the received addresses, which are then displayed in the mobile device view. The application was built in Android Studio and tested on tablets with Android 4.2.2, 4.4.4 and 6.0.1 operating systems.

The theoretical section of the thesis dealt with the general functionality and basic concepts of RabbitMQ, as well as Java and XML languages used in Android application development. In addition, in the theoretical sections, an overview of the final functionality of the application was presented.

The practical section of the thesis dealt with the installation of the RabbitMQ server and the most important programming solutions of the Android application building, including creating views and activities, receiving, and handling messages from the server and downloading content from the address received from the message, were presented. In addition, hosting a local server using Python 3.7.2 was presented.

In the reflection section, the results of the work were reviewed and possible future plans and opportunities for use were considered for further development.

Key words: RabbitMQ, Android, Java, XML, mobile development

SISÄLLYS

ERITYISSANASTO

AMQP	*Advanced Message Queuing Protocol*, avoimen lähdekoodin standardi asynkroniseen viestintään.
HTML	*Hypertext Markup Language*, verkkosivujen luomiseen käytetty standardoitu kuvauskieli
Java	Sun Microsystemsin kehittämä luokkapohjainen oliokieli
JSON	*JavaScript Object Notation*, avoimen standardin tiedostomuoto, jota käytetään tiedonvälitykseen
Mp4	Säiliömuoto, joka voi sisältää videokuvaa, ääntä ja tekstitystä
Python	Monipuolinen korkean tason ohjelmointikieli
TCP	*Transmission Control Protocol*, tietokoneiden välille yhteyksien luomista varten käytetty tietoliikenneprotokolla
URL	*Uniform Resource Locator*, verkkosisältöön osoittava merkkijono.
XML	*eXstensible Markup Language*, metakieli, jolla määritellään rakenteellisia merkkauskieliä

1 JOHDANTO

Työn tarkoituksena on perehtyä RabbitMQ-viestijonojen toimintaan ja niitä käyttävän Android-mobiilisovelluksen suunnitteluun ja toteutukseen. Teknologioina käytetään Java- ja XML-kieliä, joiden lisäksi lokaalin palvelimen luomiseen on käytetty Python-kieltä. Käytetyistä kirjastoista merkittävimmät olivat RabbitMQ:n amqp-client, Googlen Gson ja Apachen HttpComponents.

Lopputuloksena on tarkoitus luoda sovellus, joka ottaa RabbitMQ-palvelimelta viestin, joka sisältää URL-osoitteen ladattavaan HTML-, tai Mp4-sisältöön, lataa sisällön ja asettaa sen mobiilisovelluksessa näkyville.

Toisessa luvussa esitellään yleiskuvaus sovelluksen toiminnasta ja siinä käytetyistä teknologioista ja kirjastoista.

Kolmannessa luvussa esitellään RabbitMQ:n viestinvälitysprosessin yleiskuva ja peruskäsitteet.

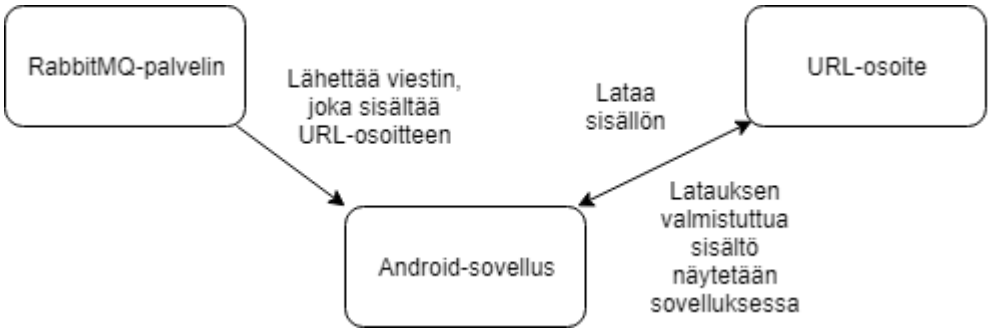
Neljännessä luvussa esitellään Android-mobiilisovelluksessa käytetyt teknologiat.

Viidennessä luvussa esitellään mobiilisovelluksen ja RabbitMQ-palvelimen rakennuksen työvaiheet ja ohjelmointiratkaisut.

Kuudennessa luvussa pohditaan yleisesti projektin lopputulosta ja sovelluksen jatkokehitystä parannuksien ja uusien mahdollisten ominaisuuksien osalta.

2 YLEISKUVAUS SOVELLUKSESTA

Työn tarkoituksena on luoda Android-sovellus, joka ottaa vastaan RabbitMQ-palvelimelta URL-osoitteen sisältäviä viestejä. Kun viesti on vastaanotettu, sovellus lataa osoitteesta .zip-muodossa HTML- tai Mp4-sisällön ja luo uuden aktiviteetin, joka näyttää sisällön (kuva 1).



KUVA 1: Kuvaus sovelluksen toiminnallisuudesta

RabbitMQ-palvelin hankitaan CloudAMQP-palvelusta, jossa on myös helppokäyttöinen rajapinta viestien lähettämiseen. Viestien lataamiseen käytetään Apachen HttpComponents-kirjastoa ja testikäyttöä varten viestissä lähetettävä URL-osoite viittaa paikalliseen palvelimeen, jota isännöidään Python 3.7.2-kielen http.server-moduulilla.

Sovellus koostuu kahdesta aktiviteetista, joista ensimmäinen näyttää vain latausanimaation ja tekstin, joka kertoo sovelluksen tämän hetkisen tilan. Näytettävät tilat ovat sisällön odottaminen, lataaminen ja purkaminen. Sovelluksen toinen aktiviteetti etsii ladatun sisällön laitteen muistista ja lataa sen sovellusnäkymään. Koska sisältö voi olla joko HTML- tai Mp4-sisältöä, aktiviteetissa on kaksi eri koko näkymän kattavaa komponenttia, joista WebView näyttää HTML-sisältöä ja VideoView Mp4-sisältöä.

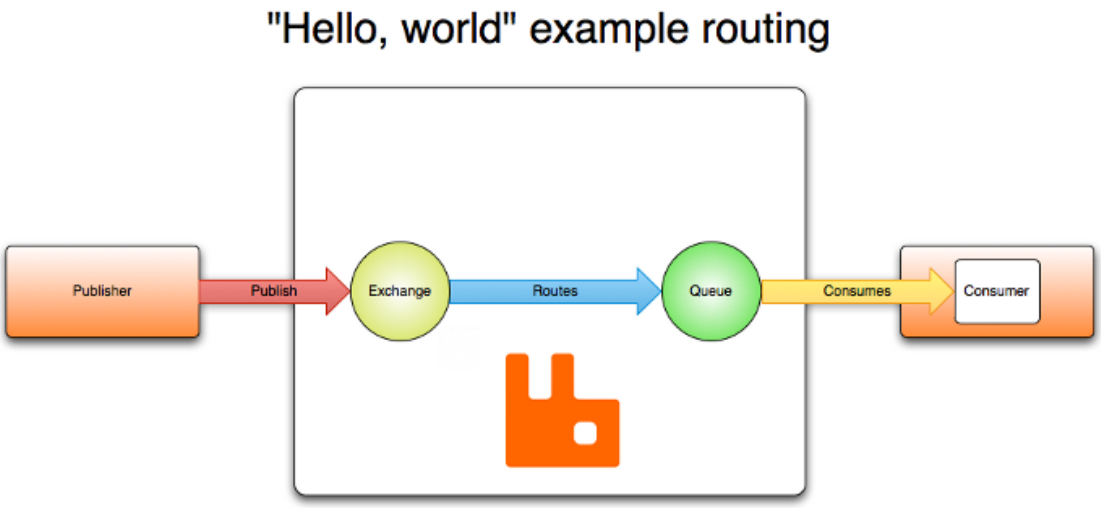
3 RABBITMQ

RabbitMQ on avoimen lähdekoodin viestinvälittäjäohjelmisto, joka antaa sovelluksille yhteisen alustan viestien lähettämiselle, vastaanottamiselle ja varastoinnille ennen, kun ne otetaan vastaan (Pivotal Inc. What can RabbitMQ do for you). RabbitMQ tukee monia eri viestintäprotokollia, mutta tässä työssä keskitytään vain AMQP 0-9-1:een.

AMQP 0-9-1 (Advanced Message Queuing Protocol) on viestintäprotokolla, jonka avulla yhteensopivat asiakassovellukset voivat kommunikoida vastaavien viestintävälineohjelmien välittäjien kanssa.

Viestinvälittäjät saavat viestejä julkaisijoilta (*publisher*), jotka julkaisevat ja ohjaavat ne kuluttajille (*consumer*), eli niitä käsitteleville sovelluksille. Koska kyseessä on verkkoprotokolla, kustantajat, kuluttajat ja välittäjä voivat olla eri koneissa.

AMQP 0-9-1- mallilla on seuraava näkemys maailmasta: viestit julkaistaan vaihtoon (*exchange*), jota verrataan usein postitoimistoihin tai postilaatikoihin. Vaihto-ohjelmat jakavat sitten kopiot viesteistä jonoihin käyttämällä sääntöjä, joita kutsutaan siteiksi (*bindings*). Sen jälkeen välittäjä (*broker*) joko lähettää viestejä jonon (*queue*) tilanneille (*subscribed*) kuluttajille tai kuluttajat hakevat tai vetävät viestejä jonoista kysynnän mukaan (kuva 2).



KUVA 2: Esimerkki yksinkertaisesta viestin välittämisestä (Pivotal Inc. AMQP 0-9-1 Model Explained).

Kun viestejä julkaistaan, julkaisijat voivat määrittää erilaisia viestin määritteitä (*message attributes*), jotka ovat käytännössä viestin metatietoa. Välittäjä voi käyttää jotakin tätä metatietoa, mutta loppuosa on välittäjälle täysin näkymätön ja sitä käyttävät vain viestit vastaanottavat sovellukset.

Verkot ovat epäluotettavia ja sovellukset saattavat epäonnistua viestien käsittelyssä, joten AMQP 0-9-1-mallissa on käsite viestien kuittauksista (*message acknowledgements*): kun viesti toimitetaan kuluttajalle, kuluttaja ilmoittaa välittäjälle joko automaattisesti tai heti sovelluskehittäjän valinnan jälkeen tehdä niin. Kun viestien kuittaukset ovat käytössä, välittäjä poistaa viestin kokonaan jonosta vain, kun se vastaanottaa ilmoituksen kyseisestä viestistä tai viestiryhmästä.

Tietyissä tilanteissa, esimerkiksi kun viestiä ei voi ohjata, viestit voidaan palauttaa julkaisijoille, pudottaa tai, jos välittäjä toteuttaa laajennuksen, se asetetaan ns. "kuolleen kirjeen jonoon". Julkaisijat valitsevat, miten käsitellään tällaisia tilanteita julkaisemalla viestejä tiettyjen parametrien avulla (Pivotal Inc. AMQP 0-9-1 Model Explained).

3.1 Vaihdot ja vaihtotyypit

Vaihdot ovat AMQP 0-9-1 kokonaisuuksia, joissa viestit lähetetään. Vaihdot ottavat viestin ja ohjaavat sen nollaan tai useampaan jonoon. Käytetty reititysalgoritmi riippuu vaihtotyypistä ja siteiksi kutsutuista säännöistä. AMQP 0-9-1-välittäjät tarjoavat neljä vaihtotyyppiä:

- Suora vaihto (*Direct exchange*)
- Fanout-vaihto (*Fanout exchange*)
- Aihevaihto (*Topic exchange*)
- Otsikkovaihto (*Headers exchange*).

Edellä mainittujen vaihtotyyppien lisäksi, vaihdoille annetaan määritteitä, joista tärkeimpiä ovat

- nimi
- kestävyys, joka määrittää, että selviääkö vaihto välittäjän uudelleenkäynnistyksen

- auto-delete, joka määrittää, että poistetaanko vaihto viimeisen siihen sidotun jonon vapauttamisen yhteydessä
- argumentit, joita voidaan käyttää liitännäisten ja välittäjien toiminnassa (Pivotal Inc. AMQP 0-9-1 Model Explained).

3.1.1 Suora vaihto

Suora vaihto välittää viestejä jonoihin viestin reititysavaimen (*routing key*) perusteella. Jonot sitoutuvat tietyllä reititysavaimella välittäjään, ja kun välittäjälle saapuu samalla reititysavaimella viesti, se lähetetään jonolle. Tietyn reititysavaimen omaavasta viestistä lähetetään kopio jokaisella samalla reititysavaimella sidotulle jonolle. Suora vaihto on ihanteellinen viestien yksinkertaiselle reitittämiselle (Pivotal Inc. AMQP 0-9-1 Model Explained.)

3.1.2 Fanout-vaihto

Toisin kuin suora vaihto, fanout-vaihto ei huomioi reititysavaimia, vaan lähettää vaihtoon saapuvasta viestistä kopion kaikkiin siihen sitoutuneisiin jonoihin. Tätä voidaan käyttää sovelluksissa, joissa ei tarvitse lähettää yksilöityä dataa eri laitteisiin, kuten esimerkiksi reaaliaikaisia urheilutuloksia näyttävässä sovelluksessa (Pivotal Inc. AMQP 0-9-1 Model Explained).

3.1.3 Aihevaihto

Aihevaihto reitittää viestejä yhteen tai useampaan jonoon, joihin reitittäminen perustuu reititysavaimen rakenteeseen. Aihevaihtoja käytetään yleisesti viestien monilähetysreitityksessä. Esimerkkinä aihevaihtoa käyttävästä sovelluksesta voisi olla uutissovellus, joka vie tietynlaiset uutiset eri jonoihin (Pivotal Inc. AMQP 0-9-1 Model Explained).

3.1.4 Otsikkovaihto

Otsikkovaihto ei fanout-vaihdon tavoin huomioi reititysavainta ollenkaan, vaan sen sijaan reitittää viestit käyttäen viestien otsake (*header*)- attribuuttia käyttäen. Otsikkovaihtoihin voi sitoa jonoja useammilla kuin yhdellä otsakkeella ja valita,

reititetäänkö viesti kaikkien vai vain yhden otsakkeen täsmätessä sidottuun jonoon (Pivotal Inc. AMQP 0-9-1 Model Explained).

3.2 Jono

Jonot tallentavat sovellusten kuluttamia viestejä. Jonoilla on samanlaisia ominaisuuksia, kuin vaihdoilla, joista tärkeimmät ovat

- nimi
- kestävyys (selviääkö jono välittäjän uudelleenkäynnistyksestä)
- eksklusiivisuus (pystyykö vain yksi yhteys kerrallaan käyttämään jonoa)
- auto-delete (poistetaanko jono, kun viimeinen kuluttaja lopettaa tilauksensa)
- argumentit (voidaan käyttää liitännäisten ja välittäjien toiminnassa).

Ennen kuin jonoa voidaan käyttää, se on ilmoitettava (*declare*). Jonon ilmoittaminen aiheuttaa sen, että se luodaan, jos sitä ei vielä ole. Ilmoituksella ei ole vaikutusta, jos jono on jo olemassa ja sen attribuutit ovat samat kuin ilmoituksessa (Pivotal Inc. AMQP 0-9-1 Model Explained).

3.3 Side

Siteet ovat sääntöjä, joita vaihdot käyttävät viestien reitittämiseen oikeisiin jonoihin. Siteillä voi olla reititysavain, jota jotkut vaihtotyypit käyttävät. Reititysavain on käytännössä suodatin, jonka avulla oikea viesti voidaan lähettää oikeaan jonoon. Jos viestiä ei voida reitittää mihinkään jonoon, se joko hylätään tai lähetetään takaisin julkaisijalle (Pivotal Inc. AMQP 0-9-1 Model Explained).

3.4 Kuluttaja

Kuluttajat ottavat viestejä jonoista ja käyttävät niitä. Ennen kuin kuluttajat voivat käyttää viestejä tietystä jonosta, niiden pitää tilata jono. Yhdessä jonossa voi olla useampi kuin yksi kuluttaja. Jokaisella kuluttajalla on oma merkkinsä (*tag*), jota ne voivat käyttää tilausten peruuttamiseen (Pivotal Inc. AMQP 0-9-1 Model Explained).

3.5 Kanava

Jotkin sovellukset tarvitsevat useita yhteyksiä välittäjään. On kuitenkin epätoivottavaa pitää monet TCP-yhteydet auki samanaikaisesti, koska tämä kuluttaa järjestelmän resursseja ja vaikeuttaa palomuurien määrittämistä. AMQP 0-9-1-liitännät multipleksoidaan kanavilla, jotka voidaan ajatella "kevyiksi yhteyksiksi, joilla on yksi TCP-yhteys".

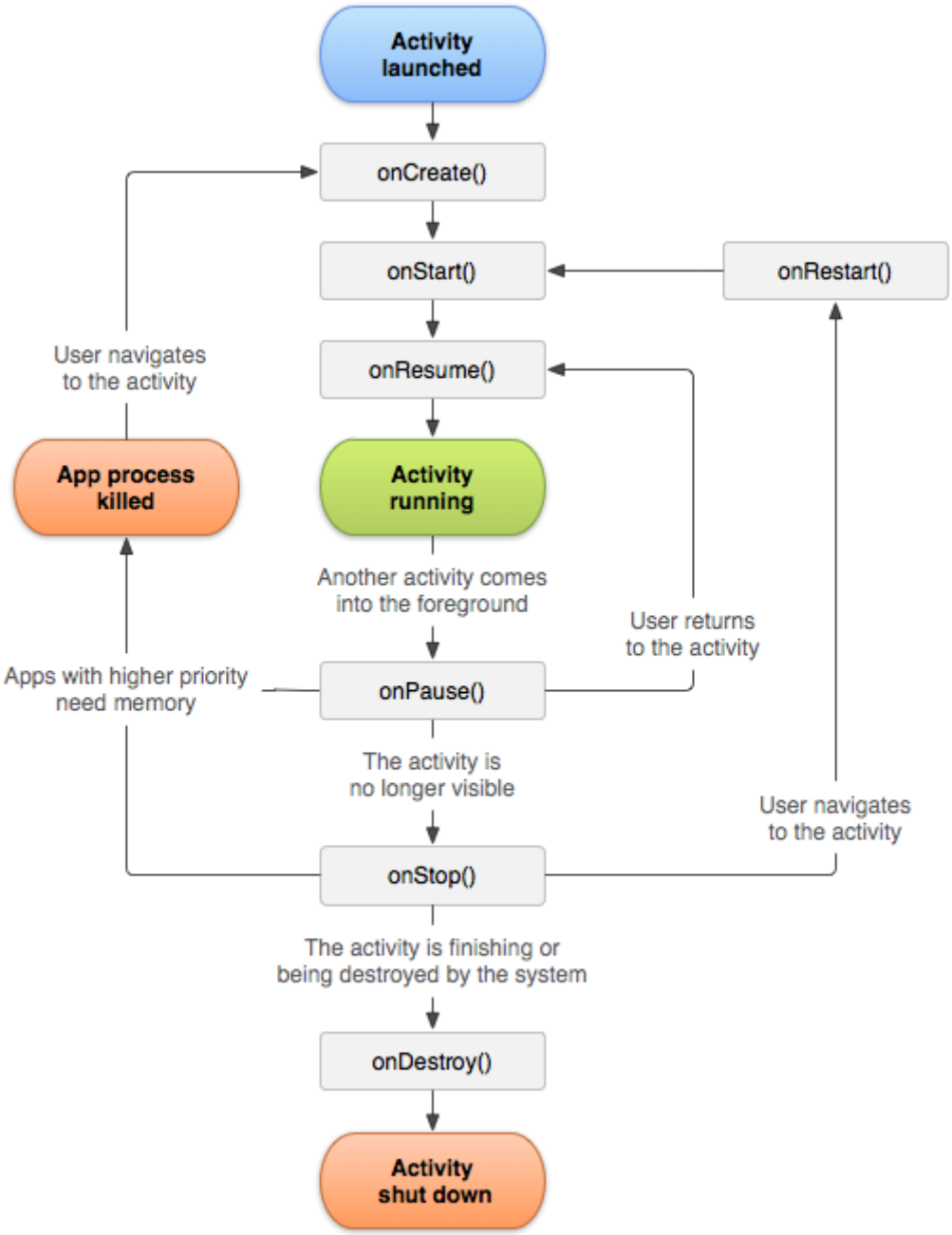
Jokainen asiakkaan (*client*) suorittama protokollatoiminta tapahtuu kanavalla. Tietyllä kanavalla tapahtuva viestintä on täysin erillään toisella kanavalla tapahtuvasta viestinnästä, joten jokaisella protokollamenetelmällä on myös kanavan tunnus (*channel ID*), kokonaisluku, jota sekä välittäjä että asiakkaat käyttävät selvittämään, mihin kanavaan menetelmä on.

Kanava on olemassa vain yhteyden kontekstissa, eikä koskaan yksinään. Kun yhteys on suljettu, kaikki kanavat sulkeutuvat sen mukana. Sovelluksissa, joissa käsitellään useita prosesseja, on hyvin yleistä avata uusi kanava prosessia kohden eikä jakaa kanavia niiden välillä (Pivotal Inc. AMQP 0-9-1 Model Explained).

4 ANDROID

Android on Googlen kehittämä pääosin mobiililaitteille suunnattu käyttöjärjestelmä. Tyypillinen Android-sovellus koostuu näkymistä (*layout*), jotka kirjoitetaan XML-kielellä ja aktiviteeteista (*activity*), jotka kirjoitetaan Java- tai Kotlin-kielellä. Näkymät määrittävät sen, miltä sovellus näyttää ja aktiviteetit määrittävät sen, mitä sovellus tekee.

Aktiviteetilla on oma elinkaarensa, joka koostuu kuudesta eri tilasta, joihin aktiviteetti päätyy olemassaolonsa aikana (kuva 3). Aktiviteetin käyttäytymistä voidaan muuttaa näihin tiloihin tullessa, jotta esimerkiksi käyttäjän poistuessa sovelluksesta ja aktiviteetin saapuessa tuhoutumistilaan (*onDestroy*) voidaan tallentaa käyttäjän tekemiä muutoksia sovelluksen sisällä. Tämä on hyödyllistä esimerkiksi tekstinmuokkaussovelluksissa. Elinkaaressa on ainoastaan yksi tila, jonka aikana on pakko tehdä tietyt asiat. Tämä on aktiviteetin luomishetki (*onCreate*), jossa määritetään aktiviteetin käyttöliittymän ulkonäkö asettamalla näkymä.



KUVA 3: Aktiviteetin elinkaari (Android Developers. Understanding the Activity Lifecycle).

Näkymä koostuu juurielementistä ja sen sisältämistä elementeistä. Nämä voivat olla esimerkiksi tekstikenttiä, nappeja tai kuvia (kuva 4).

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

KUVA 4: Yksinkertainen näkymä, joka sisältää vain tekstikentän

Kaikille elementeille pitää asettaa leveys- ja pituusattribuutit, joiden avulla aktiviteetti asettaa ne halutulla tavalla käyttöliittymään. Tämän lisäksi on suositeltavaa antaa elementeille tunniste (*id*), jonka avulla niitä voi käyttää ja muokata aktiviteetin sisällä.

4.1 XML

XML on HTML-kieltä muistuttava metakieli, jolla määritellään rakenteellisia merkkauskieliä. Itsessään XML ei tee mitään, mutta sen avulla voidaan varastoida ja kuljettaa dataa tunnisteiden avulla, joka helpottaa merkittävästi datan erottelua (kuva 5). Tämän erottelun avulla eri sovellukset voivat käyttää tiettyjä XML-tiedostossa olevia elementtejä ja sijoittaa niitä haluamiinsa paikkoihin tai käyttää niitä haluamallaan tavalla (W3Schools. Introduction to XML).

```
<?xml version="1.0" encoding="UTF-8"?>
<koira>
  <ikä>5</ikä>
  <rotu>Saksanpaimenkoira</rotu>
  <omistaja>Essi Esimerkki</omistaja>
  <väri>Ruskea</väri>
  <nimi>Musti</nimi>
</koira>
<koira>
  <ikä>8</ikä>
  <rotu>Siperianhusky</rotu>
  <omistaja>Elli Esimerkki</omistaja>
  <väri>Harmaa</väri>
  <nimi>Roni</nimi>
</koira>
```

KUVA 5: Esimerkki XML-tiedostosta

Android sovelluksissa XML-tiedostoja käytetään esimerkiksi näkymien luomiseen ja oletusattribuuttien antamiseen elementeille, jotta tietynkaltaisille elementeille saadaan samanlainen ulkonäkö ja attribuutit.

4.2 Java

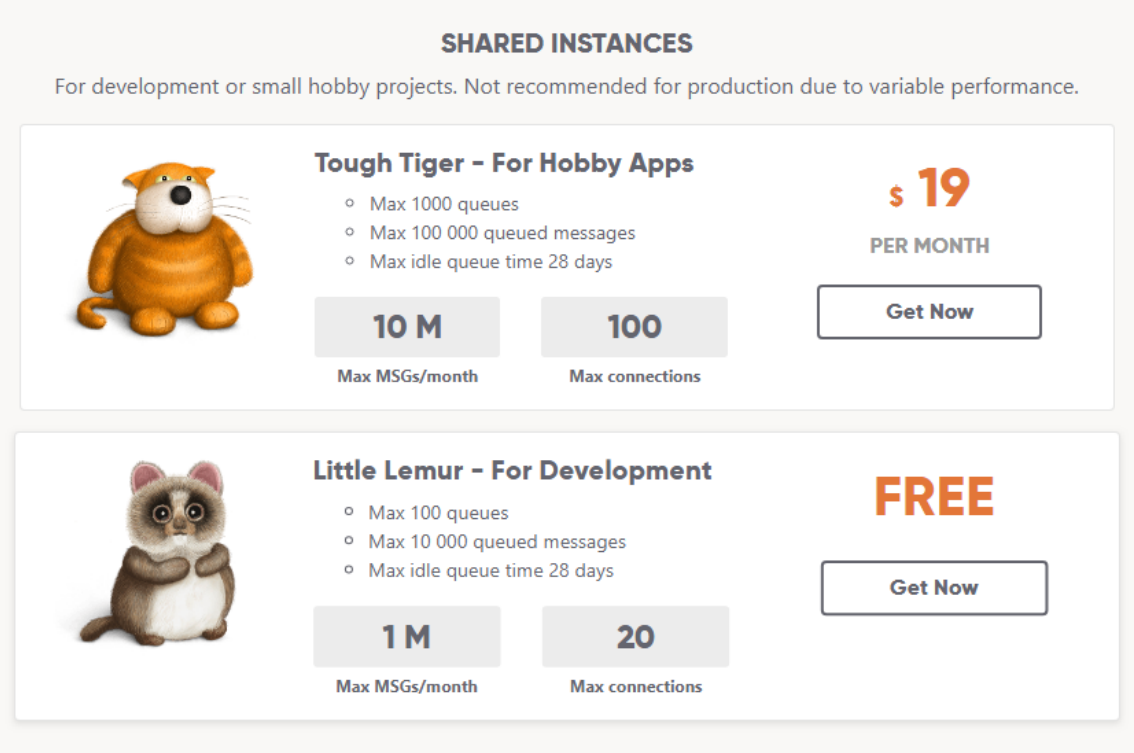
Java on luokkapohjainen oliokieli, jonka Sun Microsystems julkaisi vuonna 1995. Java-koodi toimii kaikilla sitä tukevilla alustoilla käyttöjärjestelmästä riippumatta käyttäen koodin suorittamiseen virtuaalikonetta (*Java Virtual Machine, JVM)*, jota pidetään yhtenä Javan tärkeimmistä ominaisuuksista. Toinen Javan tärkeimmistä ominaisuuksista on automaattinen roskien kerääjä (*garbage collector*), joka poistaa laitteen muistista sovelluksessa luotuja objekteja, joita ei enää käytetä, helpottaen merkittävästi muistin hallinnoimista. Muistivuoto voi silti tapahtua, jos koodissa viitataan olemattomaan objektiin (HowToDoInJava. What is Java programming language?)

5 OHJELMOINTIRATKAISUT

Tässä osiossa käsitellään sovelluksen rakentamista ja sen aikana tehtyjä ohjelmointiratkaisuja. Käsittely aloitetaan AMQP-palvelimen pystytyksestä, jonka jälkeen siirrytään Android-sovelluksen ohjelmointiin aluksi vastaanottamaan, ja sen jälkeen käsittelemään viestejä. Sovellus näyttää aluksi latausnäkymää, kunnes palvelimelta vastaanotetaan viesti, joka sisältää URL-osoitteen, josta sovellus lataa HTML- tai MP4-sisällön ja siirtyy näyttämään sitä.

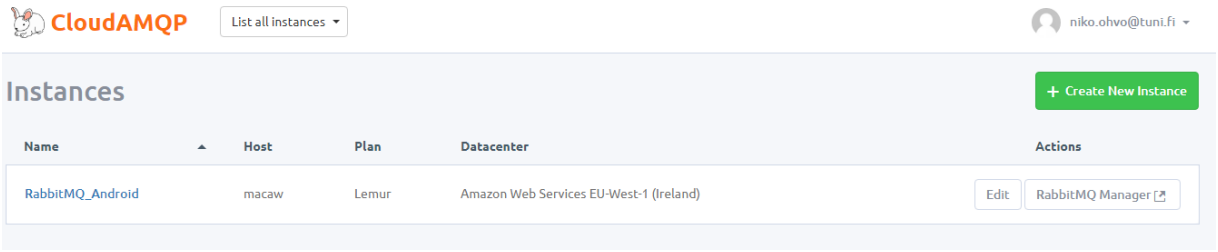
5.1 Palvelimen pystytys

Palvelimen pystyttämiseen käytettiin CloudAMQP- palvelua, jonka avulla RabbitMQ- jonoja pystytään hallitsemaan verkossa käytettävän rajapinnan kautta. CloudAMQP tarjoaa eri laajuisia hinnoittelusuunnitelmia palveluilleen, joista kalliimmat tarjoavat suuremman määrän yhtäaikaisia yhteyksiä palvelimeen ja suuremman maksimimäärän jonoissa käsiteltäviä viestejä kuukaudessa. Tässä työssä käytettiin ohjelmistokehitykseen tarkoitettua, ilmaista hinnoittelusuunnitelmaa (kuva 6).



KUVA 6: Kaksi halvinta palvelun hinnoittelutyyppiä

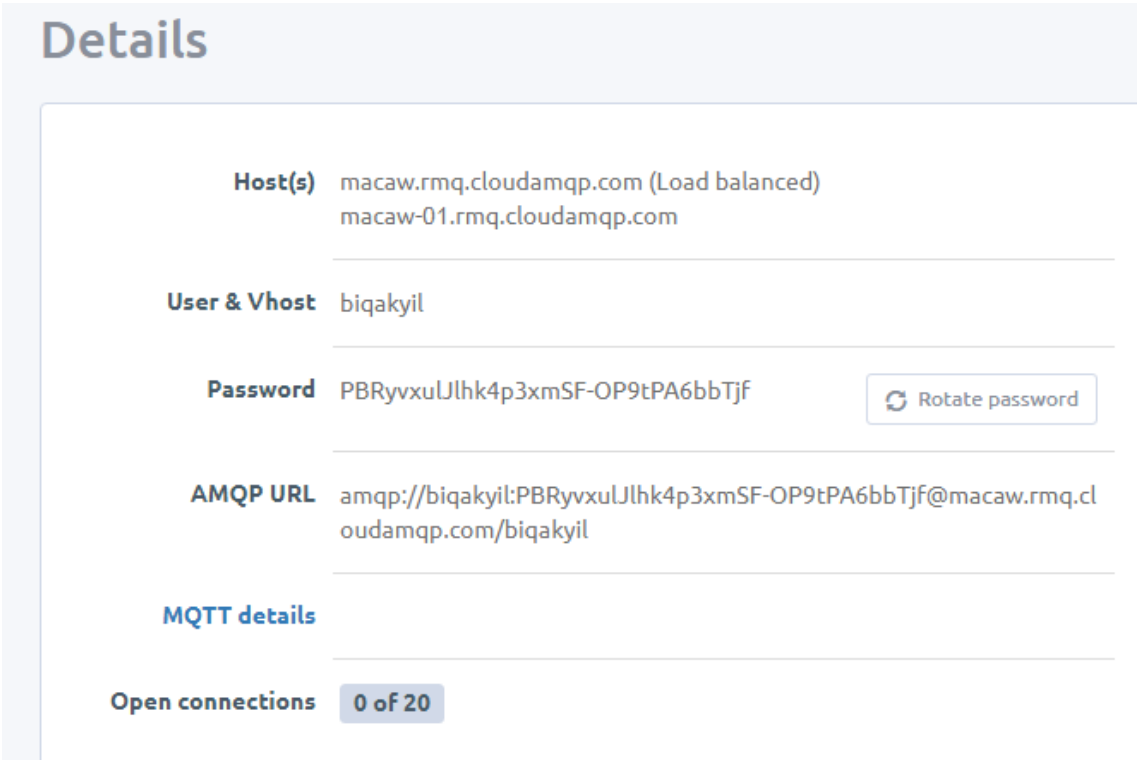
Palvelu otetaan käyttöön rekisteröitymällä asiakkaaksi, jonka jälkeen käyttäjä pääsee instanssien hallintasivulle (kuva 7).



KUVA 7: CloudAMQP-instanssien hallintasivu

Instanssi luodaan painamalla *Create new instance*-painiketta ja valitsemalla siellä nimi ja hinnoittelusuunnitelma instanssille. Tämän jälkeen valitaan instanssille palvelukeskus ja se voidaan ottaa käyttöön.

Kun instanssi on otettu käyttöön, sen nimeä klikkaamalla pääsee näkymään, josta näkee mm. RabbitMQ- yhteyttä varten tarvittavat kredentiaalit ja tietoa instanssin tämän hetkisten yhteyksien ja viestien määrästä (kuva 8).



KUVA 8: CloudAMQP-instanssin tietonäkymä

5.2 Mobiilisovelluksen käyttöliittymän rakentaminen

Kehitysympäristönä käytettiin Android Studiota. Debuggaus ja testaus toteutettiin Android 4.2.2-, 4.4.4- ja 6.0.1- version omaavilla tableteilla.

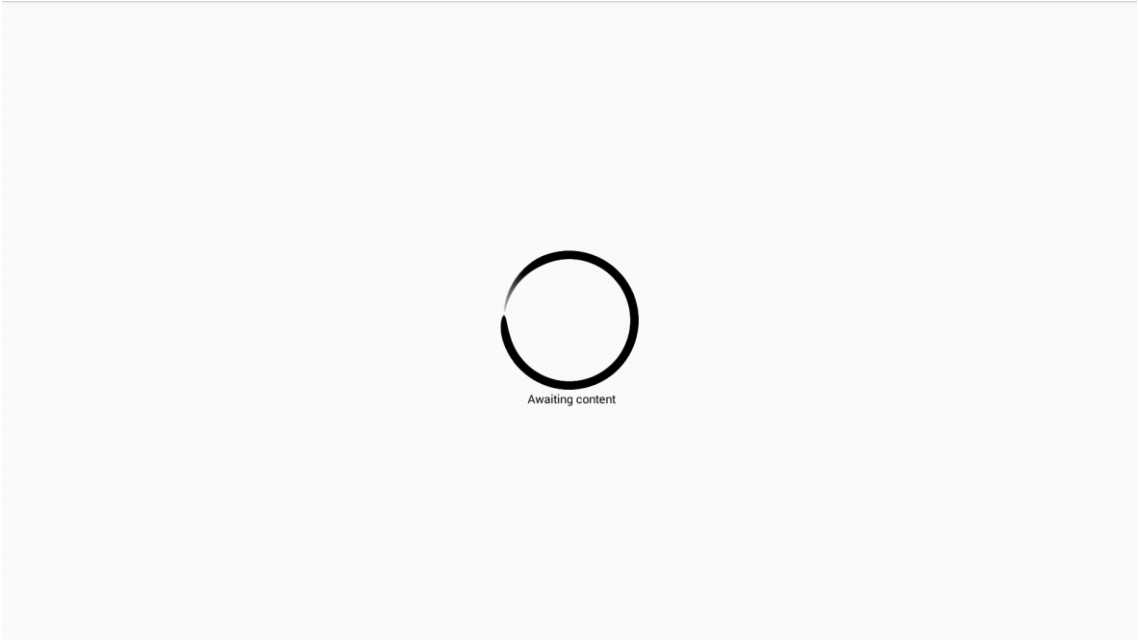
Sovelluksen rakentaminen aloitettiin luomalla Android Studiolla projektipohja, joka luo sovellukselle automaattisesti yhden *layout*-näkymän, jota muokkaamalla lisätään ja poistetaan elementtejä näkymästä ja yhden aktiviteetin (*activity*), jonka kautta voidaan muokata, miten näkymän elementit toimivat ja mitä niistä tapahtuu. Layout kirjoitetaan XML- kielellä ja aktiviteetti Javalla. Projektipohjaan kuuluu myös AndroidManifest.xml-tiedosto, josta hallinnoidaan sovelluksen ja aktiviteettien oikeuksia käyttää mobiililaitteen eri ominaisuuksia. Tätä sovellusta varten tarvittiin oikeudet internet-yhteyteen, ulkoiseen muistiin kirjoittamiseen ja mahdollisuus sovelluksen valitsemiseen käynnistyssovellukseksi. Tämän lisäksi Home-aktiviteetille, jossa näytetään sovelluksen lataama sisältö, asetettiin noHistory-ominaisuus, joka mahdollistaa vain yhden samanlaisen aktiviteetin olemassaolon muiden kanssa (kuva 9).

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.INTERNET" />

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="Advertising_App"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme"
    tools:ignore="GoogleAppIndexingWarning">
    <activity android:name=".Home"
            android:noHistory="true"/>
    <activity android:name=".Init">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
            <category android:name="android.intent.category.HOME" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </activity>
</application>
```

KUVA 9: Sovelluksen AndroidManifest.xml

Layoutin rakentamiseen käytettiin ConstraintLayout- näkymätyyppiä, jonka avulla näkymän elementit voidaan sitoa toisiinsa ilman tarvetta antaa elementeille kovakoodattuja arvoja niiden sijoitusta varten näkymässä helpottaen skaalautuvuutta eri resoluution omaaville laitteille. Skaalautuvuutta helpottaen elementtien sijoitteluun käytettiin myös apuviiva (*guideline*)- elementtejä, jotka ovat näkymättömiä viivoja, joihin voi kuitenkin sitoa elementtejä. Apuviivat voivat olla joko pysty- tai vaakasuunnassa ja niille voi antaa prosentuaalisen arvon niiden sijoitusta varten näkymässä.

Sovelluksen ensimmäiseen näkymään asetettiin *ImageView*- ja *TextView*- elementit, jotka sijoitettiin apuviivoja käyttäen näkymän keskelle. ImageView näyttää kuvan, johon on erillisessä tiedostossa määritelty animaationa 360:n asteen pyöriminen ja TextView näyttää sovelluksen sen hetkisen tilanteen viestin vastaanottamisesta (kuva 10).



KUVA 10: Sovelluksen alkunäkymä

Myöhemmin ladattavan sisällön näyttämistä varten luotiin myös toinen näkymä, joka koostuu *WebView*- ja *VideoView*- elementeistä. WebViewiin ladataan HTML-sisältö ja VideoViewiin MP4-sisältö (kuva 11). Näiden kahden elementin näkyvyyttä säädellään sen mukaan, minkälainen sisältö laitteeseen on ladattu sillä hetkellä.

```
<android.support.constraint.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <WebView
        android:id="@+id/homeWebView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:visibility="invisible"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

    <VideoView
        android:id="@+id/homeVideoView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:visibility="visible"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>
</android.support.constraint.ConstraintLayout>
```

KUVA 11: Sisältönäkymän ohjelmakoodi

Sisällön vastaanottamista varten tarkistetaan sovelluksen käynnistyessä, että laitteen ulkoisessa muistissa on olemassa kansio, johon sisältö ladataan latausviestin saapuessa. Jos kansio ei ole olemassa, niin se luodaan. Sisältö etsitään rekursiivisella funktiolla, joka etsii sisällöstä HTML- tai MP4-sisältöä ja tallentaa sisältönäkymää varten muuttujaan pääsisältötiedoston polun, jotta se voidaan myöhemmin ladata WebView- tai VideoView-elementtiin. Jos käynnistyksen yhteydessä sisältökansiosta löytyy sisältöä, se siirtyy heti sisältönäkymään.

5.3 Viestien vastaanottaminen

Viestien vastaanottamista varten otettiin Android-projektissa käyttöön RabbitMQ Java Client 4.7.0-kirjasto, joka mahdollistaa Java-sovellusten toimivuuden RabbitMQ:n kanssa. Tämä tapahtui muokkaamalla build.gradle-tiedostoa Android-projektin sisällä ja lisäämällä sinne implementaatio tarvittavasta kirjastosta (kuva 12).

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'com.android.support.constraint:constraint-layout:1.1.2'
    implementation 'com.rabbitmq:amqp-client:4.7.0'
    implementation 'com.google.code.gson:gson:2.8.5'
    implementation 'org.apache.commons:commons-io:1.3.2'
    implementation group: 'org.apache.httpcomponents', name: 'httpclient-android', version: '4.3.5.1'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
}
```

KUVA 12: Build.gradle-tiedosto

Tämän jälkeen luotiin uusi Java-luokka nimeltä AMQPConnection, joka voidaan luoda sovelluksen alkunäkymän aktiviteetissa. AMQPConnection-oliota luodessa annetaan aluksi *ConnectionFactory*-muuttujalle URI, joka löytyy CloudAMQP:n instanssin tietonäkymästä (kuva 13).

```
public class AMQPConnection {

    private ConnectionFactory factory = new ConnectionFactory();
    private AMQPConnectionListener listener = null;
    private String downloadUrl;
    private Handler subscribeHandler = new Handler();
    Runnable r = () -> { subscribe(); };

    public interface AMQPConnectionListener {
        void onMessageReceived(final String url);
    }

    AMQPConnection() {
        setFactoryCredentials(factory);
        subscribe();
    }
    @SuppressLint("AuthLeak")
    private void setFactoryCredentials(ConnectionFactory factory) {
        try {
            factory.setUri("amqp://bigakyil:PBRyvxulJlhk4p3xmSF-OP9tPA6bbTjf@macaw.rmq.cloudamqp.com/bigakyil");
            factory.setRequestedHeartbeat(10);
        } catch (URISyntaxException | NoSuchAlgorithmException | KeyManagementException e) {
            e.printStackTrace();
        }
    }
```

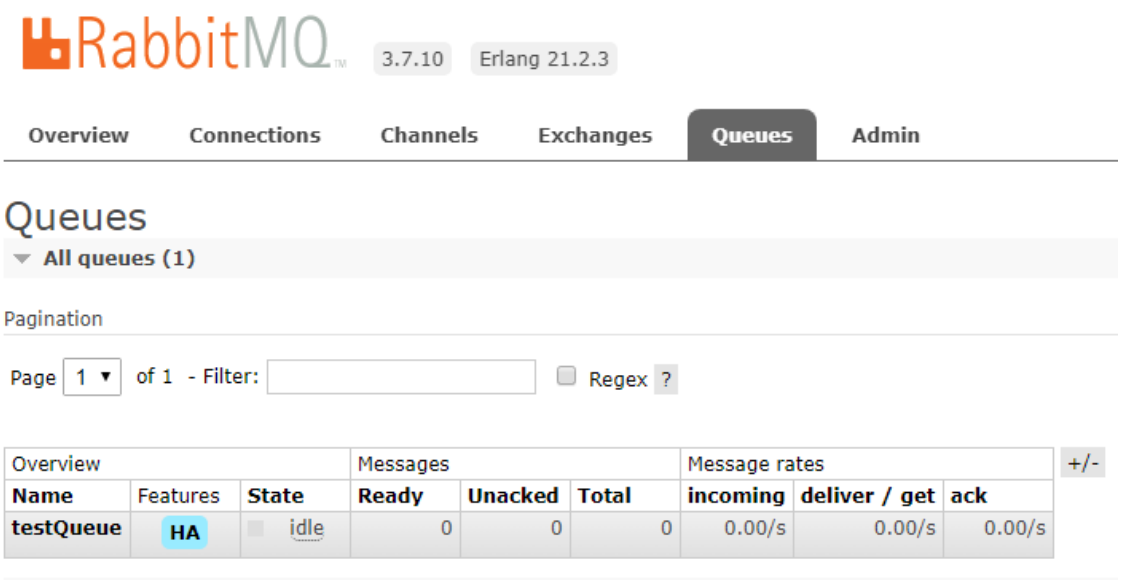
KUVA 13: AMQPConnection-olion luominen

Tämän jälkeen tilataan (*subscribe*) palvelimelta jono ja aletaan kuuntelemaan sitä. Tilaaminen tapahtuu aluksi luomalla yhteys ConnectionFactoryn avulla palvelimelle, jonka jälkeen luodaan jono ja sidotaan se kanavaan palvelimella (kuva 14).

```
Log.d( tag: "AMQPCONNECTION",  msg: "Starting connection");
final Connection connection = factory.newConnection();
Channel channel = connection.createChannel();
channel.basicQos( prefetchCount: 1);
AMQP.Queue.DeclareOk q = channel.queueDeclare( queue: "testQueue",  durable: false,  exclusive: false,  autoDelete: false,  arguments: null);
channel.queueBind(q.getQueue(),  exchange: "amq.direct",  routingKey: "test");
```

KUVA 14: Jonon luominen

Jonon luomisen onnistumisen voi testata CloudAMQP:n RabbitMQ-managerista navigoimalla queues-välilehteen ja tarkistamalla sieltä onko jono luotu (kuva 15).



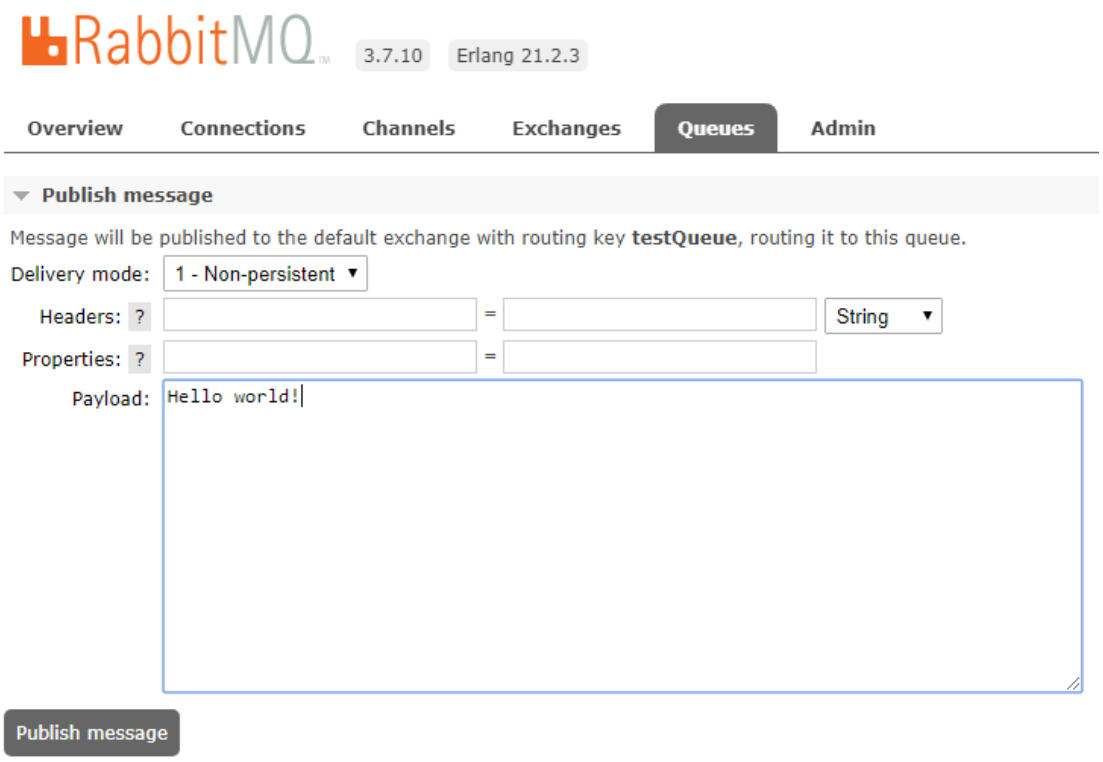
KUVA 15: RabbitMQ-managerin jononäkymä

Viestien vastaanottaminen tapahtuu luomalla kuluttaja (*consumer*), jolle määritellään tapa käsitellä viesti (kuva 16).

```
Consumer consumer = new DefaultConsumer(channel) {
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {
        String message = new String(body, charsetName: "UTF-8");

        // If there was no errors parsing the JSON and the file id is not duplicate, start download
        if (parseDownloadUrlJSON(message)) listener.onMessageReceived(downloadUrl);
        Log.d( tag: "AMQPCONNECTION", msg: "Received message " + message);
    }
};
channel.basicConsume(q.getQueue(), autoAck: true, consumer);
```

KUVA 16: Kuluttajan luominen

Viestien vastaanottamista voidaan testata tulostamalla se kehitysympäristön lokeihin. Haluttu viesti voidaan lähettää CloudAMQP:n RabbitMQ-managerista navigoimalla queues-välilehdeltä oikeaan jonoon ja sitä kautta julkaisemalla viesti jonoon (kuva 17).



KUVA 17: Viestin julkaiseminen RabbitMQ-managerin kautta

5.4 Viestien käsitteleminen

Viestien käsittelyä varten päätettiin, että oikeanlainen vastaanotettava viesti on JSON-formaatissa oleva contentUrl-nimisen muuttujan sisältävä viesti. ContentUrl nimensä mukaan sisältää osoitteen ladattavaan sisältöön. Kun formaatti oli päätetty, tehtiin sovellukseen funktio, joka ottaa viestistä osoitteen käyttämistä varten (kuva 18). Viestin parsimista varten käytettiin Googlen Gson-kirjastoa.

```
private boolean parseDownloadUrlJSON(String json) {
    try {
        JsonParser parser = new JsonParser();
        JsonObject obj = (JsonObject) parser.parse(json);
        downloadUrl = obj.get("contentUrl").getAsString();
        return true;
    }
    catch (Exception e) {
        Log.d( tag: "AMQPCONNECTION",  msg: "JSON parsing error: " + e);
        e.printStackTrace();
        return false;
    }
}
```

KUVA 18: Viestin parsimisfunktio

Kun osoite on saatu viestistä, olion sisällä olevan AMQPConnectionListener-niminen rajapinta (*interface*) laukaisee sovelluksen alkuaktiviteetissa sijaitsevan kuuntelijan (*listener*), joka poistaa sisältökansiosta kaiken ja aloittaa uuden sisällön lataamisen.

Ennen sisällön lataamista poistetaan vanha sisältö, jonka jälkeen uusi sisältö ladataan viestistä saadun osoitteen avulla koodissa määriteltyyn polkuun .zip-tyyppisenä tiedostona ja puretaan erilliseen sisältökansioon.

Sisällön lataamista testattiin luomalla Python 3.7.2-kielen http.server-moduulilla paikallinen palvelin, johon laitettiin .zip-tiedostona HTML-sisältö (kuva 19).

```
C:\Users\Niko\Desktop\DownloadContent>python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

KUVA 19: Paikallisen palvelimen luominen

Tämän jälkeen Android-tabletti ja palvelinta hostaava tietokone kytkettiin samaan verkkoon ja lähetettiin RabbitMQ-managerin kautta JSON-muodossa viesti, joka sisälsi polun sisältöön. Saatuaan polun sisältöön, käytettiin Apachen HttpComponents-kirjastoa sisällön lataamiseen (kuva 20).

```
D/DELETE: Start clearing folder
D/AMQPCONNECTION: Received message {"contentUrl":"http://192.168.2.208:8000/content.zip"}
D/DELETE: Success
D/DOWNLOADER:: CLIENT CREATED
D/DOWNLOADER:: HTML ENTITY NOT NULL, CREATING FILE
D/DOWNLOADER:: DOWNLOADING FINISHED
D/UNZIP: Begin unzipping
D/ZIP: Starting unzipping process
D/ZIP: Unzipping finished
```

KUVA 20: Viestin käsitteleminen

5.5 Sisällön näyttäminen

Kun sisältö on ladattu ja purettu sisältökansioon, lähetetään kuulutus (*broadcast*) sovelluksen sisällä, jotta mahdollinen jo olemassa oleva sisältönäkymä tuhoaa itsensä, jonka jälkeen luodaan uusi sisältönäkymä, johon ladataan uusi sisältö (kuva 21, kuva 22).

```
try {
    Log.d( tag: "LOAD",  msg: "Loading URL");
    homeWebView.loadUrl("file:///" + viewFile);
} catch (Exception e) {
    Log.d( tag: "LOAD",  msg: "Error: " + e);
    e.printStackTrace();
    finish();
}
```

KUVA 21: Sisällön lataaminen näkymään

KUVA 22: Sisältönäkymä tabletissa

6 POHDINTA

Työn tavoitteena oli luoda RabbitMQ-jonoja käyttävä Android-mobiilisovellus, joka lataa jonosta tulevan viestin avulla URL-osoitteesta sovelluksen näkymään sisällön ja sitä kautta ymmärtää syvemmin RabbitMQ-jonojen toimintaa ja Android-sovelluskehitystä. Android-sovelluskehitystä helpotti merkittävästi se, että työkalut ja teknologiat olivat ennestään tuttuja työelämästä, mutta työn alkua hidasti tietämättömyys RabbitMQ:n käyttämisestä.

Projektin lopputuloksena valmistui sovellus, joka tekee kaiken, mitä ensimmäisessä kappaleessa jo mainittiinkin. Jatkokehitystä ajatellen projekti on helposti yhdistettävissä palvelimeen, joka käyttää samaa RabbitMQ-instanssia viestien lähettämiseen tabletille. Yhdellä instanssilla voidaan myös hallinnoida monia tabletteja, joilla on mahdollisesti myös kaikilla erilainen sisältö määrittelemällä jokaiselle tabletille oma reititysavaimensa.

LÄHTEET

Pivotal Inc. What can RabbitMQ do for you. Luettu 15.4.2019
https://www.rabbitmq.com/features.html

Pivotal Inc. AMQP 0-9-1 Model Explained. Luettu 15.4.2019
https://www.rabbitmq.com/tutorials/amqp-concepts.html

Android Developers. Understand the Activity Lifecycle. Luettu 10.5.2019
https://developer.android.com/guide/components/activities/activity-lifecycle

W3Schools. XML Introduction. Luettu 10.5.2019
https://www.w3schools.com/xml/xml_whatis.asp

HowToDoInJava. What is Java programming language? Luettu 10.5.2019
https://howtodoinjava.com/java/basics/what-is-java-programming-language/