Timo Piippo

Bingojärjestelmän testaus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

13.1.2014

Metropolia

Tekijä Otsikko Sivumäärä Aika	Timo Piippo Bingojärjestelmän testaus 41 sivua 13.1.2014
Tutkinto	insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Senior Programmer Kalle Kärkkäinen Yliopettaja Auvo Häkkinen

Tässä insinöörityössä tutkitaan testausympäristön kehittämistä Bingobooster-järjestelmälle. Työtä aloitettaessa järjestelmän testaus oli melko satunnaista, ja järjestelmän kehittäneellä yrityksellä oli tarve kattavammalle testaustavalle.

Työssä käydään ensin läpi Bingobooster-järjestelmän rakenne ja komponentit ja kerrotaan tekniikoista, joita sen toteutuksessa on käytetty. Tämän jälkeen käsitellään käytetyt testaustavat.

Järjestelmän suoritus- ja kestokykyä testattiin tekemällä sille stressitesti suurella määrällä asiakasohjelmia. Testissä havaittuja ongelmia tutkittiin ja ne korjattiin, jonka jälkeen korjausten vaikutus todennettiin ajamalla testi uudestaan.

Työssä suoritettiin myös yksikkötestejä järjestelmän asiakasohjelmien ja palvelinten välissä olevalle yhteysmanagerille, jonka tehtävänä on välittää viestejä ja hallinnoida yhteyksien tilaa. Testeissä käytettiin JUnit-yksikkötestauskehystä.

Lopuksi työssä tutkittiin myös hyväksymistestaukseen tarkoitettuja Robot Framework- ja Jasmine-testauskehyksiä ja niiden soveltuvuutta tämän projektin testaamiseen. Jasmine testaa JavaScript-koodia ja tutkimisen myötä syntyi myös JavaScript-versio rasitustestaukseen käytetystä asiakasohjelmasta. Tämän avulla järjestelmän suorituskykyä voitiin testata myös toista rajapintaa käyttäen.

Avainsanat	Testaus, Java, JUnit, Jasmine, rasitustestaus, yksikkötestaus, hyväksymistestaus, testauskehys

Author(s) Title Number of Pages Date	Timo Piippo Testing of Bingo System 41 pages 13 January 2014
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Kalle Kärkkäinen, Senior Programmer Auvo Häkkinen, Principal Lecturer

The thesis studies the development of a testing environment to the Bingobooster system. The testing of the system was rather sporadic when the study was started so there was a need for a more comprehensive testing environment.

First the structure, the components and the techniques of the Bingobooster system are analyzed. After this the used testing methods are covered.

The performance and durability of the system was tested with a stress test that included a large number of clients. The problems found were investigated and fixed. After this the effects were verified by rerunning the test.

Unit tests were also built for the manager responsible for the connections between the servers and the clients and the delivering of messages. The JUnit testing framework was used for this.

In the end the acceptance testing frameworks Robot and Jasmine were studied and their usefulness for this project was analyzed. Jasmine is meant for testing JavaScript programs and by studying it a JavaScript version of the stress test client was made. With this the performance of the system could be tested by using an alternative interface.

Keywords	Software testing, Java, JUnit, Jasmine, stress testing, unit testing, acceptance testing, testing framework

Sisällys

Lyhenteet

BSS	Bingo Socket Server eli bingon liitäntäpalvelin. Ohjelmisto, joka muuntaa ECM-järjestelmän lähettämät viestit ECM-liitännäiselle.
ECM	Järjestelmä, joka hoitaa bingopelin etenemistä, lappujen myyntiä ja arvontaa. Koostuu SRCe-, XSCe- ja RNGe-laitteista. Myös samanniminen yhtiö on kehittänyt em. laitteiston ja toimittaa sitä.
JUnit	Yksikkötestausympäristö Javalle.
MCB	Mechanical Cash Bingo eli käteispeli. Bingon väliajalla pelattava bingopeli, johon ei tarvitse ostaa etukäteen pelipaketteja.
MSB	Mainstage Bingo eli bingon pääpeli, joka on pääasiallisesti pelattava bingopeli ECM-järjestelmässä. Pelaaminen tapahtuu ostamalla paketteja, joihin kuuluu bingokirjoja, jotka ovat MSB-pelejä.
POS	Point of Sale. Ohjelmisto, joka on tehty hoitamaan hallissa tapahtuvaa myyntiä, kassan hallintaa ja raportointia.
Protobuf	Protocol Buffer. Googlen kehittämä viestityyppi. Proto-tiedostossa määritellään viestin rakenne (kuten luokka ja jäsenmuuttujat), jonka jälkeen viesti käännetään halutulle kielelle ja sitä voi käyttää koodissa oliomaisesti.
R	Ohjelma kaavioiden tekemiseen tilastodatasta.
SRCe	ECM-järjestelmän osa, joka hoitaa pelin kulkua, kuten bingon oikeellisuuden vahvistusta ja pelin vaihtamista.
RNGe	ECM-järjestelmän osa, joka hoitaa pallojen arvontaa.
XSCe	ECM-järjestelmän osa, joka hoitaa myyntisessiossa oleviin peleihin tulleita ostoja ja niiden raportointia.

1 Johdanto

Työn tavoitteena on kehittää Airdice Oy:lle testausympäristö internetissä ja halleissa pelattavalle Bingobooster-bingojärjestelmälle. Bingojärjestelmä on käytössä Suomessa ja tulee jonkin verran erilaisena käyttöön myös Englannissa. Tähän asti testausmenetelmänä ovat toimineet manuaalisesti käsitellyt testitapaukset, joissa on määritelty lähtötilanne, tehtävät toimenpiteet ja oletettu lopputulos. Bingon kehitys on kuitenkin edennyt siihen pisteeseen, että tarvitaan manuaalista testausta kehittyneempiä testausmenetelmiä, jotta järjestelmää voidaan testata perusteellisemmin. Tarkoituksena on tutkia erilaisia testaustapoja ja toteuttaa niistä parhaat testausjärjestelmään.

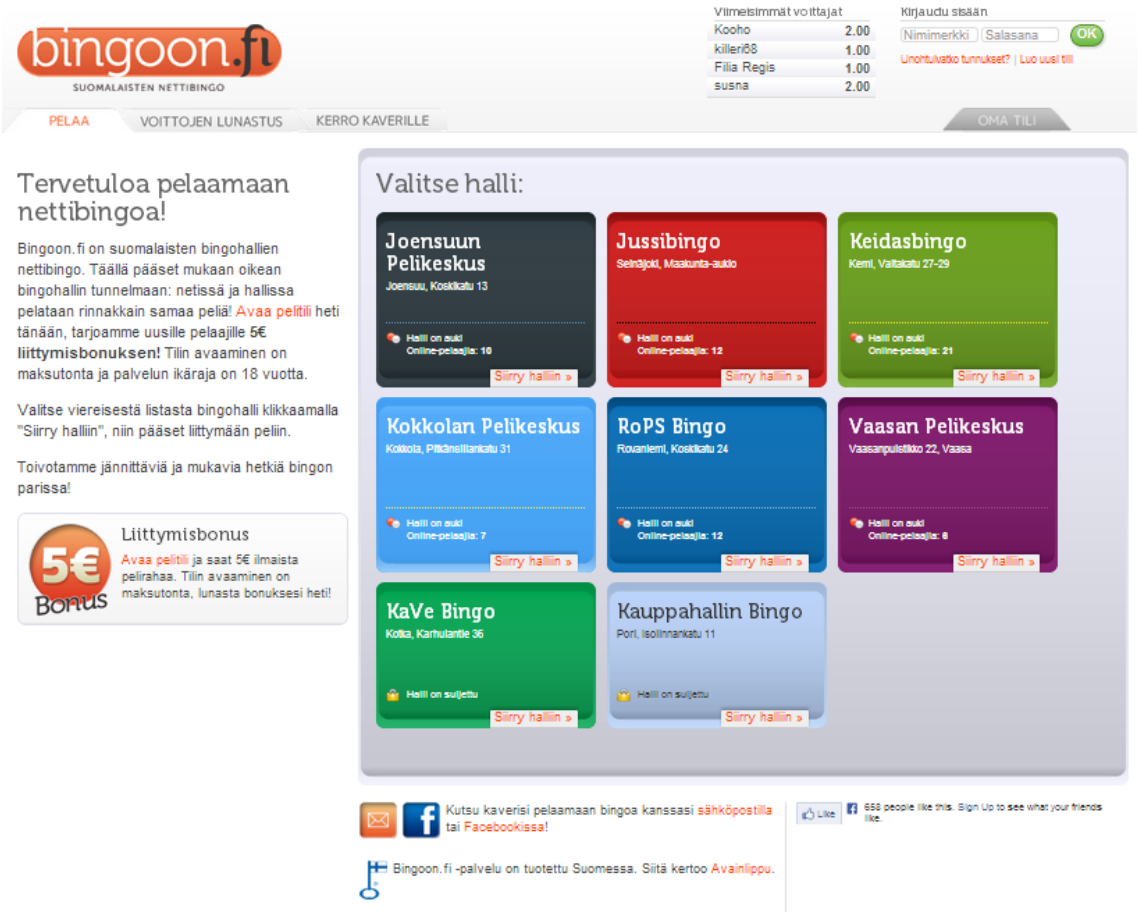
Bingon pelaamisessa käytetään tätä varten kehitettyjä laitteita, jotka arpovat pallot ja tarkistavat huudettujen bingojen oikeellisuuden. Tämän lisäksi käytössä ovat pelaamista varten tabletit ja pelien myyntiin tarkoitettu ohjelmisto. Tarkoituksena on ohjelmoida näille laitteille vastineet ja automatisoida ympäristöä siten, että järjestelmää voi testata esimerkiksi suurella määrällä asiakasohjelmia.

Työssä kerrotaan alkutilanteesta, asiakkaalla tällä hetkellä käytössä olevasta järjestelmästä ja sen tekniikasta. Tämän jälkeen perehdytään siihen, millainen on uusi, tämän rinnalle tuleva ja osittain korvaava järjestelmä ja mitä tekniikkaa siinä käytetään. Lopuksi kerrotaan varsinaisesta testausjärjestelmästä. Testausjärjestelmä koostuu käytännössä järjestelmätestaukseen kuuluvasta rasitustestistä ja järjestelmän asiakasohjelmistojen yhteyksiä hallinnoivan komponentin yksikkötesteistä. Rasitustestiä ajetaan suoraan palvelinta vasten sekä palvelimen ja asiakasohjelman välissä olevan nettipalvelun kautta. Rasitustestauksessa järjestelmää kuormitettiin suurella määrällä asiakasohjelmistoja ja mitattiin järjestelmän suorituskykyä.

Työssä tutustutaan myös mahdollisuuteen käyttää hyväksymistestaukseen tarkoitettuja ohjelmistokehyksiä Robot Framework ja Jasmine, joista jälkimmäisellä ohjelmoidaan testitapaukset JavaScriptillä. Tätä mahdollisuutta alettiin tutkia, koska JavaScriptillä ohjelmoitua asiakasohjelmaa voisi käyttää myös muihin tarkoituksiin bingojärjestelmän kokonaisuutta ajatellen.

2 Lähtötilanne

Airdice on kehittänyt Suomessa pelattavaa bingoa varten Bingoon.fi-palvelun. Palvelun ideana on mahdollistaa perinteisesti hallissa tapahtuvaan bingopeliin liittyminen myös netin kautta. Palveluun luodaan käyttäjätunnus netissä ja valitaan oikea fyysinen halli, jossa olevaan peliin halutaan osallistua. Hallissa oleva laitteisto saa tiedon netin kautta samaan peliin liittyneistä käyttäjistä, jonka jälkeen sekä hallissa että kotikoneilla olevat pelaajat pelaavat yhtä aikaa samaa peliä. Kuvassa 1 on Bingoon.fi-sivuston etusivu.



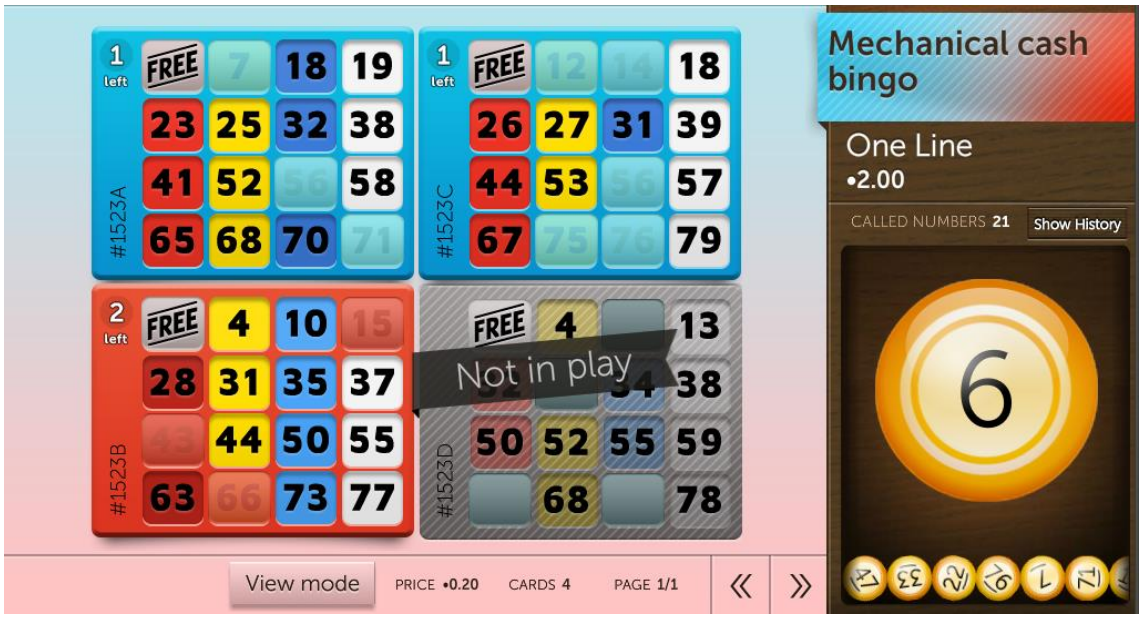
Kuva 1 Bingoon.fi-sivusto [Bingoon.fi, 2013]

Sivuston etusivulla on listattu järjestelmässä mukana olevat hallit. Halleista kerrotaan, ovatko ne sillä hetkellä auki ja montako online-pelaajaa niissä on. Käyttäjä valitsee haluamansa hallin ja näkee kyseessä olevan hallin aukioloajat, yleiset tiedot ja pelattavien pelin säännöt. Jos halli on auki ja käyttäjä on kirjautunut sisään, käyttäjä voi avata hallin peli-ikkunan ja liittyä käynnissä olevaan peliin ostamalla pelilappuja.

Vastaava järjestelmä on tarkoitus toteuttaa myös Englannissa, mutta laajemmassa mittakaavassa sekä ominaisuuksilla, joita Suomessa ei ole käytössä, kuten bingon pelaaminen tableteilla hallissa. Haasteen projektiin on asettanut se, että Suomessa käytettävän bingolaitteiston tekniikka on aivan erilainen kuin Englannissa käytössä olevan eri valmistajan laitteiston. Laitteiden lähettämät viestit ovat erityyppisiä, ja suurimpana syynä bingoa ajava laitteisto on ohjelmoitu ajamaan erityyppisiä bingopelejä eri säännöillä. Tästä johtuen kaikkea olemassa olevaa tekniikkaa ei voi käyttää sellaisenaan, vaan Englannissa käytössä olevaa laitteistoa on jouduttu opiskelemaan alusta alkaen ja kehittämään tämän perusteella ohjelmistoja.

2.1 Bingopelin kulku

Tavallisesti pelatessa bingoa bingohallissa pelaajat ostavat bingolappuja, joissa oleviin ruudukkoihin on merkitty valmiiksi satunnaisia numeroita tietyltä väliltä. Lappuja voi ostaa, kunnes bingohallin henkilökunta aloittaa arvonnan. Hallissa oleva laitteisto arpoo satunnaisesti numeroita, jotka henkilökunta kuuluttaa ja joita pelaajat merkitsevät lapulle arvotun numeron täsmätessä lapussa olevaan numeroon. Yhdessä pelissä voi olla useampi kierros, esimerkiksi kolme, joiden välillä vaihtuvat säännöt, joiden mukaan pelaaja saa bingon. Seuraavassa kuvassa 2 on pelaajan näkymä bingopelin aikana.



Kuva 2 Pelaajan näkymä pelin aikana

Kuvassa oikeassa alakulmassa näkyy, että viimeisin arvottu numero on kuusi. Sen yläpuolella oleva ”Called numbers” kertoo, että palloja on arvottu 21 ja aivan oikealla alhaalla näkyy aiemmin arvottuja numeroita. Bingoruudukoiden vasemmassa yläkulmassa näkyy, kuinka monta numeroa tarvitaan vielä käynnissä olevan kierroksen vaadittuun voittoyhdistelmään. Kaikkien ruudukoiden ylhäällä vasemmalla oleva ruutu on ilmainen ruutu, joka on automaattisesti merkitty ja mukana voittolinjoissa. Peli on ensimmäisellä kierroksella, koska yhtään voittolinjaa ei ole vielä löytynyt.

Yhden pelin kulku voi olla esimerkiksi seuraava: arvottavien numeroiden kokonaismäärä on 80. Ensimmäisellä kierroksella pelaaja tarvitsee ruudukostaan yhden voittolinjan, joka voi pelistä riippuen olla esimerkiksi vaakarivi, pystyrivi, vinorivi, ruudukon keskusta tai kulmat. Kun ensimmäinen voittolinja löytyy, vaihtuu kierros, jolloin lapusta pitää löytyä kaksi edellä mainituista voittotavoista. Kun kaksi linjaa on löytynyt, alkaa kolmas ja viimeinen kierros, jolloin voittoon vaaditaan kaikki numerot yhdestä lapusta. Jos pelaaja haluaa parantaa voittomahdollisuuksiaan, hänellä on mahdollisuus ostaa ennen pelin alkua useita erilaisia bingoruudukoita, jolloin todennäköisyys voittoon luonnollisesti kasvaa. Kuvassa kaikki ruudukot on ostettu lukuun ottamatta ruudukkoa 1523D.

2.2 Bingo Englannissa

Suomessa bingo on totuttu mieltämään urheiluseurojen järjestämäksi rahankeräystavaksi, jossa palkintoina on esimerkiksi tuotepalkintoja tai lahjakortteja, mutta ei rahaa, johtuen lakiteknisistä seikoista. Bingon suosio ei myöskään ole ollut kovin suurta verrattuna muihin rahapeleihin kuten lottoon tai kenoon.

Englannissa puolestaan bingo on laajemmassa suosiossa. Bingoa pelataan rahalla, kuten Suomessakin, mutta peleistä on myös mahdollista voittaa rahaa. Ecm Systems -nimisellä yrityksellä on käytännössä monopoli bingohalleissa käytettäviin laitteistoihin. Laitteistot sisältävät SRCe-nimisen laitteen, joka pitää yllä tietoa pelinkulusta, pelattavasta pelistä, pelaajilta tulevista viesteistä sekä muusta vastaavasta. Laitteistoon kuuluvat myös XSCe, joka vastaa bingosessioissa olevien pelien myynnin hoitamisesta, sekä RNGe, jolla on pallojen arpomiseen liittyviä tehtäviä.

Bingo ennen

Ennen Englannissa bingohallissa pelatessa käytössä ollut tietotekniikka on rajoittunut SRCe-, XSCe- ja RNGe-laitteisiin. Pelaajien käyttämät bingoruudukot on painettu paperilapuille ja ne on ostettu hallihenkilökunnalta. Pelin edetessä merkinnät on tehty näihin lappuihin ja bingon tullessa henkilökunta on pyydetty paikalle tarkastamaan lapusta bingon oikeellisuus. Varsinaisten bingopelien väliajalla pelattavan Mechanised Cash Bingo -pelin ruudukot on kiinnitetty pelipöytiin, ja ne ovat siis pelipaikkakohtaisia eivätkä vaihdu pelien välillä.

Mainstage Bingo (MSB) eli bingon pääpeli

Mainstage Bingo eli bingon pääpeli on bingohalleissa pelattava peli, jota pelataan eniten. Bingolaitteistossa valitaan ja käynnistetään päivän tietty peliohjelma, jolloin laitteisto lähettää kyseisessä ohjelmassa pelattavat pelit. Ohjelma koostuu niin sanotuista kirjoista, jotka sisältävät yhden tai useampia sivuja, ja jokainen sivu taas on oma pelattava pelinsä. Hallin henkilökunta koostaa kirjoista paketteja, jotka sisältävät tietyn määrän lappuja eri peleihin. Pelaajat ostavat laput sisältävät paketit peleihin etukäteen. Pelissä on 75 tai 90 arvottavaa numeroa. Peliruudukot ovat 9x3- (joissa on kuitenkin 15 numeroa), 5x5- tai 4x4-kokoisia. Pelattavien kirjojen välillä on taukoja, joiden aikana on mahdollista pelata ns. Mechanised Cash Bingo -pelejä.

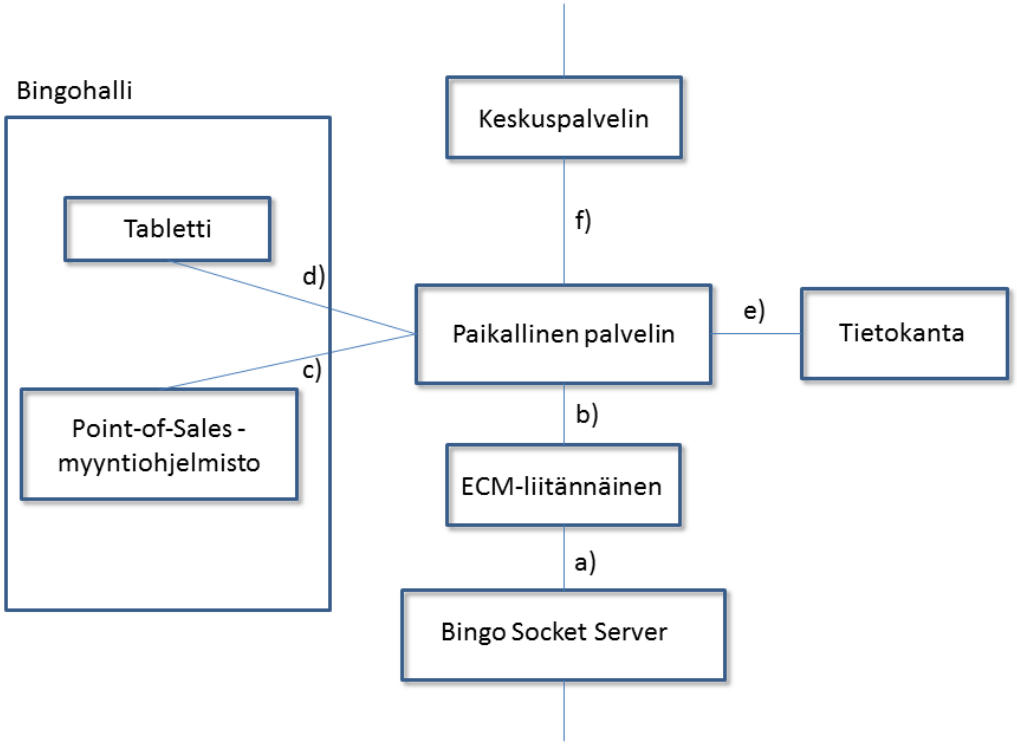
Mechanised Cash Bingo (MCB) eli käteisbingo

Varsinaisten pelien välillä pelaajat voivat osallistua Mechanised Cash Bingo- eli käteisbingopeleihin, joihin ei tarvitse ostaa paketteja etukäteen. Pelaajien pelipaikalla on valmiina tiettyjä ruudukkoja, jotka aktivoituvat pelaajan ostaessa ne laittamalla kolikon pöydällä olevaan raha-aukkoon. Käteisbingopelit pelataan 80 numerolla 4x4-ruudukolla valituilla sääntöyhdistelmillä. Pelejä pelataan vapaavalintainen määrä odotellessa seuraavan varsinaisen pääpelikierroksen alkamista.

3 Bingobooster-järjestelmä

Bingobooster-projektin myötä entisiä paperisia pelilappuja ja peliruudukkoja ei ole tarkoitus korvata vaan pelitarjontaa voidaan laajentaa tuomalla mukaan tabletteja käyttä-

vät pelaajat sekä netin kautta liittyvät asiakkaat. Kaikki pelaajat pelaavat siis samassa pelissä. Järjestelmän valmiina olevat komponentit ovat ECM-järjestelmä (SRCe, XSCe ja RNGe) sekä näiden jäljessä oleva Bingo Socket Server (BSS) eli bingon liitäntäpalvelin. Kaikki ECM-järjestelmän lähettämä tieto menee liitäntäpalvelimen kautta. Lähetetty tieto muunnetaan ja lähetetään eteenpäin. Kun lähetetään tietoa ECM-järjestelmälle, liitäntäpalvelin vastaavasti muuntaa viestit sen ymmärtämään muotoon. Järjestelmään Airdicellä tehdyt osat ovat bingon liitäntäpalvelimen ja paikallisen palvelimen välille tehty liitännäinen, paikallinen palvelin, joka pääasiallisesti hoitaa tiedon käsittelyä ja tallentamista, Point-of-sales-myyntiohjelmisto myyntiä varten sekä keskuspalvelin, joka liittyy muihin asiakaskoneisiin ja halleihin. Tämän lisäksi bingopeliä voi pelata myös tableteilla, joihin on asennettu asiakasohjelma. Kuvassa 3 on esitetty järjestelmä paikallisen palvelimen ympärillä.



Kuva 3 Järjestelmä paikallisen palvelimen ympärillä

Bingolaitteisto lähettää viestit Bingo Socket Serverille, joka muuntaa ne ja lähettää edelleen ECM-liitännäiselle (a), joka ne käsiteltyään lähettää edelleen paikalliselle palvelimelle protobuf-viestin (b). Protobuf-viestit ovat Googlen kehittämä tapa muodostaa

viestejä samaan tapaan kuin xml:ssä, mutta viestit ovat reilusti pienempiä ja nopeampia. Kun viestit on käännetty, niitä voi myös käyttää helposti olioina koodin seassa. Palvelin lähettää pelipäivän tiedot myyntiohjelmistolle (c), josta tehdyt ostot lähetetään puolestaan palvelimelle. ECM-järjestelmältä saadut pelitapahtumat lähetetään tableteille (d), jotka taas lähettävät bingoilmoituksia ja lappuostoja palvelimelle. Pelitapahtumat ja ostot tallennetaan tietokantaan (e). Paikallinen palvelin voi välittää tietoja myös keskuspalvelimelle (f), joka siirtää ne edelleen nettipelaajille.

ECM-liitännäinen

Liitäntäpalvelimen lähettämiä viestejä vastassa on niin sanottu ECM-liitännäinen. Viestin tyypin perusteella se ohjataan oikealle käsittelijälle, jotka viestin perusteella päivittävät liitännäisessä muistissa olevaa pelitilannetta. Käsittelijät voivat esimerkiksi päätellä viestin sisällön ja tallennetun pelitilanteen perusteella, että uusi peli on alkanut ja päivittää sisäisen tilan vastaamaan sitä. Tämän jälkeen saadusta tiedosta rakennetaan protobuf-viesti, joka lähetetään paikalliselle palvelimelle.

Paikallinen palvelin

Paikallinen palvelin on liikenteen solmukohta, joka käsittelee saapuneet viestit, tekee niiden edellyttämät toimenpiteet ja välittää viestit tarvittaessa edelleen. Kun viestit tulevat ECM-liitännäiseltä, päivitetään viestien sisällön mukaan esimerkiksi pelilokia ja lähetetään viestejä edelleen pelin eri osapuolille, kuten tableteille tai POS-myyntiohjelmistolle. Paikallinen palvelin on keskeinen osa järjestelmää, sillä kaikki tieto, jonka ECM-järjestelmä lähettää, kulkee sen kautta ja välittyy järjestelmän muille osille.

Palvelimella on joukko viestinkäsittelijöitä, joille palvelimelle saapuvat protobuf-viestit ohjataan viestin tyypin mukaan. Viestit voivat tulla joko ECM-liitännäiseltä, tableteilta, POS-myyntiohjelmistolta tai keskuspalvelimelta.

Keskuspalvelin

Keskuspalvelimen tehtävänä on hallita järjestelmän paikallispalvelimia laajempana kokonaisuutena. Paikallispalvelimet ovat yhteydessä keskuspalvelimeen, joka pitää yllä tietoa mm. järjestelmän yhteyksien tilasta ja synkronoi tietoja muiden keskuspalvelimi-

en kesken. Keskuspalvelin myös ohjaa nettipelaajien asiakasohjelmilta tulevat pyynnöt oikealle paikalliselle palvelimelle, jotta pelaaja voi liittyä oikeaan peliin.

Point-of-sales -myyntiohjelmisto

POS eli Point of Sales on ohjelmisto, jota bingohallin henkilökunta käyttää hoitamaan bingoon liittyvä myyntiä. POS-myyntiohjelmisto saa ECM-järjestelmältä sen hetkisen peliohjelman, joka sisältää kyseisen ohjelman Mainstage Bingon pelit (kirjat ja sivut). Näistä peleistä hallihenkilökunta voi koostaa pelipaketteja, jotka sisältävät erilaisia yhdistelmiä pelejä. Kun pelaaja haluaa ostaa paketteja, hänen käyttäjätunnuksensa validoidaan, jonka jälkeen hän voi ostaa paketteja joko pelitilillään olevalla rahalla tai käteisellä. POS-myyntiohjelmistossa voi myös katsoa raportteja päivän myynneistä sekä tarkastella käyttäjän aiempia tilitapahtumia ja esimerkiksi hyvittää niitä.

Pelitabletti

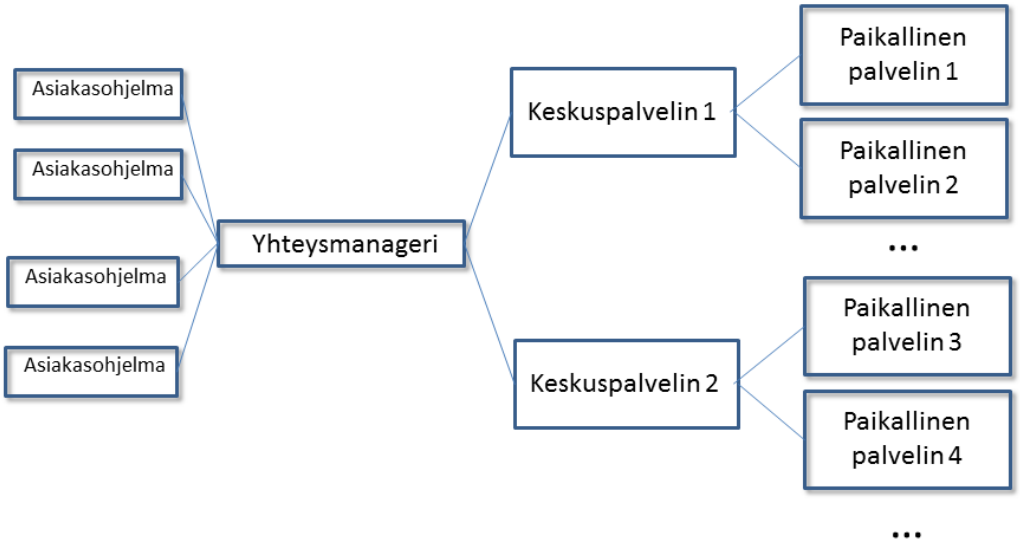
Tablettien käyttö on järjestelmän olennaisimpia lisäyksiä perinteiseen bingon pelaamiseen. Halleissa on saatavilla tabletteja, joissa pyörii asiakasohjelma bingon pelaamiseen. Asiakas tunnistautuu hallissa hallihenkilökunnalle POS-myyntiohjelmiston avulla, jolloin hän voi saada käyttöönsä tabletin, joka liitetään hänen käyttäjätiliinsä. Tabletilla näkyvät käyttäjän joko etukäteen tai tabletilla ostamat bingokirjat. Tabletilla käyttäjä voi pelata bingoa samalla tavalla kuin hän pelaisi sitä fyysisillä lapuilla; arvotut pallot näkyvät näytöllä ja pelaaja voi napin painalluksella merkata lappuihinsa numerot, jotka on siihen mennessä saanut. Kun tarvittavat numerot ovat tulleet, pelaaja voi tabletilla ilmoittaa saaneensa bingon ja voittorahat maksetaan suoraan tämän pelitilille. Tabletti saa tietoa ECM-järjestelmältä paikallispalvelimen kautta ja päivittyy sen mukaan.

Kun käyttäjä on tunnistautunut tabletin avulla, hän voi ostaa yksittäisiä käteisbingoruudukoita ja mahdollisesti myös pääpelin bingokirjoja suoraan tabletin valikon kautta ilman, että hänen tarvitsee ostaa niitä henkilökunnan avulla POS-myyntiohjelmistosta.

Järjestelmän kokonaisrakenne

Paikalliset palvelimet kommunikoivat pelintarjoajan (ECM) sekä tablettien, tietokannan ja POS-myyntiohjelmiston kanssa. Järjestelmä on kuitenkin kokonaisuudessaan laa-

jempi kuin vain paikallinen ympäristö. Seuraavassa kuvassa 4 on esitetty järjestelmän kokonaisrakenne.



Kuva 4 Järjestelmän kokonaisrakenne

Paikallinen palvelin on yhteydessä myös keskuspalvelimeen, jolla voi olla yhteyksiä myös muihin paikallispalvelimiin. Keskuspalvelimet taas ovat yhteydessä yhteysmanagerin kautta toisiinsa. Yhteysmanageri ohjaa keskuspalvelimien yhdistämisen lisäksi nettipelaajilta tulevat pyynnöt oikealle keskuspalvelimelle, josta ne menevät edelleen paikalliselle palvelimelle ja oikeaan peliin.

4 Käytetyt tekniikat

BingoBooster-projektissa on käytetty useita tekniikoita, mutta suurin osa sovelluksista, kuten palvelimet, POS-myyntiohjelmisto ja ECM-liitännäinen on ohjelmoitu Javalla. Käytännössä tablettiohjelmistot ovat suurin yksittäinen osa, joka on toteutettu muulla kuin Javalla.

Viestien lähettämiseen verkossa käytetään Googlen kehittämää Protocol Buffer -viestintätekniikkaa, jolla viestit saadaan mahtumaan pieneen tilaan. Palvelimet on rakennettu Apache-säätiön tarjoaman Mina-ohjelmistokehyksen päälle, jonka avulla yhteyksien ja viestien hallinta on helppoa [Apache Mina, 2013].

4.1 Protocol Buffer -viestintätekniikka

Protocol Buffer (protobuf) on viestintätekniikka, joka on käytössä laajasti Googlen projekteissa. Protobuf on ohjelmointikielestä riippumaton tekniikka rakentaa viestejä rakenteisesta datasta [Protocol Buffer Developer Guide, 2012]. Protobuf-viestit ovat periaatteessa rakentuneet kuten xml-viestit, mutta ne ovat 3-10 kertaa pienempiä, 20-100 kertaa nopeampia ja niitä voi käyttää helpommin muun koodin seassa olioina. Prototiedostoissa määritellään tietorakenne kuten xml:ssäkin. Tämän jälkeen viesti käännetään halutulle kielelle kääntäjällä, joka tekee viestistä luokan. Google tarjoaa kääntäjät Javalle, C++:lle ja Pythonille. Luokalla on funktiot sen muuttujien asettamiseen ja saamiseen ja muuhun operointiin. Koodiesimerkissä 1 on rakennettu viesti, joka kuvaa henkilöä ja tämän yhteystietoja.

```
package tutorial;

option java_package = "com.example.tutorial";
option java_outer_classname = "AddressBookProtos";

message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}

message AddressBook {
  repeated Person person = 1;
}
```

Koodiesimerkki 1 Protobuf-viesti [Protocol Buffer Developer Guide Java Tutorial, 2012]

Henkilöllä on kentät nimelle, id:lle, sähköpostiosoitteelle ja puhelinnumerolle, jolle on tehty oma aliviestityyppi, jossa määritellään myös numeron tyyppi. Kentät voi asettaa vaadituksi, vapaaehtoiseksi tai niillä voi olla useita arvoja. Koodissa on määritelty myös osoitekirja, joka sisältää valinnaisen määrän henkilöitä. Koodiesimerkissä 2 on käännetty aiempi henkilöä kuvaava protobuf-viesti Javalle.

```
Person john =
  Person.newBuilder()
    .setId(1234)
    .setName("John Doe")
    .setEmail("jdoe@example.com")
    .addPhone(
      Person.PhoneNumber.newBuilder()
        .setNumber("555-4321")
        .setType(Person.PhoneType.HOME))
    .build();
```

Koodiesimerkki 2 Protobuf-viestin käyttö Java-ohjelmassa [Protocol Buffer Developer Guide Java Tutorial, 2012]

Henkilöstä tehdään uusi ilmentymä, johon voidaan asettaa arvot eri kentille. Jos kenttä sisältää toisen viestin, kuten henkilön puhelintiedot, luodaan siihen uusi viesti, johon puolestaan asetetaan halutut arvot. Lopuksi viesti rakennetaan, jolloin sitä ei voi enää muokata.

4.2 Apache Mina -ohjelmistokehys

Apache-säätiön Mina on ohjelmistokehys, jolla voidaan tehdä tehokkaita ja skaalautuvia verkkosovelluksia [Apache Mina, 2013]. Minalla on helppoa tehdä palvelin, sillä se tarjoaa valmiit metodit tapahtumiin reagointiin, kuten yhteyden avaamiseen, viestin vastaanottamiseen ja käsittelyyn, lähettämiseen jne. Sen suorituskyky on myös hyvä.

Palvelin luodaan käyttämällä Minan IoAcceptor-luokkaa, jolle asetetaan IoHandlerAdapter-luokan perivä tapahtumankäsittelijä, jonka halutut metodit ylikirjoitetaan. Koodiesimerkissä 3 on tehty tapahtumankäsittelijäluokan IoHandlerAdapter perivä luokka, jonka tarvittavat metodit on ylikirjoitettu.

```
public class TimeServerHandler extends IoHandlerAdapter
{
    @Override
    public void exceptionCaught( IoSession session, Throwable cause )
throws Exception
    {
```

```
        cause.printStackTrace();
    }

    @Override
    public void messageReceived( IoSession session, Object message )
throws Exception
    {
        String str = message.toString();
        if( str.trim().equalsIgnoreCase("quit") ) {
            session.close();
            return;
        }

        Date date = new Date();
        session.write( date.toString() );
        System.out.println("Message written...");
    }

    @Override
    public void sessionIdle( IoSession session, IdleStatus status )
throws Exception
    {
        System.out.println( "IDLE " + session.getIdleCount( status ));
    }
```

Koodiesimerkki 3 Minan tapahtumankäsittelijä [Apache Mina Quick Start Guide, 2013]

Poikkeuksen sattuessa printataan sen aiheuttaja, ja kun viesti saapuu, siitä tehdään String-merkkijono. Jos se sisältää sanan ”quit”, suljetaan istunto. Muuten istuntoon kirjoitetaan sen hetkinen aikaleima. Kun viesti saapuu, sen tyyppi on Object eli siitä voi tarpeen mukaan tehdä tarvittavan luokan edustajan. Jos istunto on tarpeeksi kauan tekemättä mitään, tulostetaan, kuinka monesti se on ollut toimettomana.

Minalla on toteutettu useat palvelun tarvitsemat yhteydet, kuten yhteydet paikalliselta palvelimelta tabletteihin, POS-myyntiohjelmistolle ja ECM-liitännäiselle. Kun viesti saapuu messageReceived-metodille, siitä tehdään protobuf-viesti ja viestityypille rekisteröity viestinkäsittelijä käsittelee sen.

4.3 Tabletit ja tietokanta

Tableteille on eri versioita bingo-ohjelmistosta riippuen tabletin käyttöliittymästä. Linux-tableteille on tehty ohjelmisto Flashilla, kun taas Android-tableteille on tehty Android-käännös.

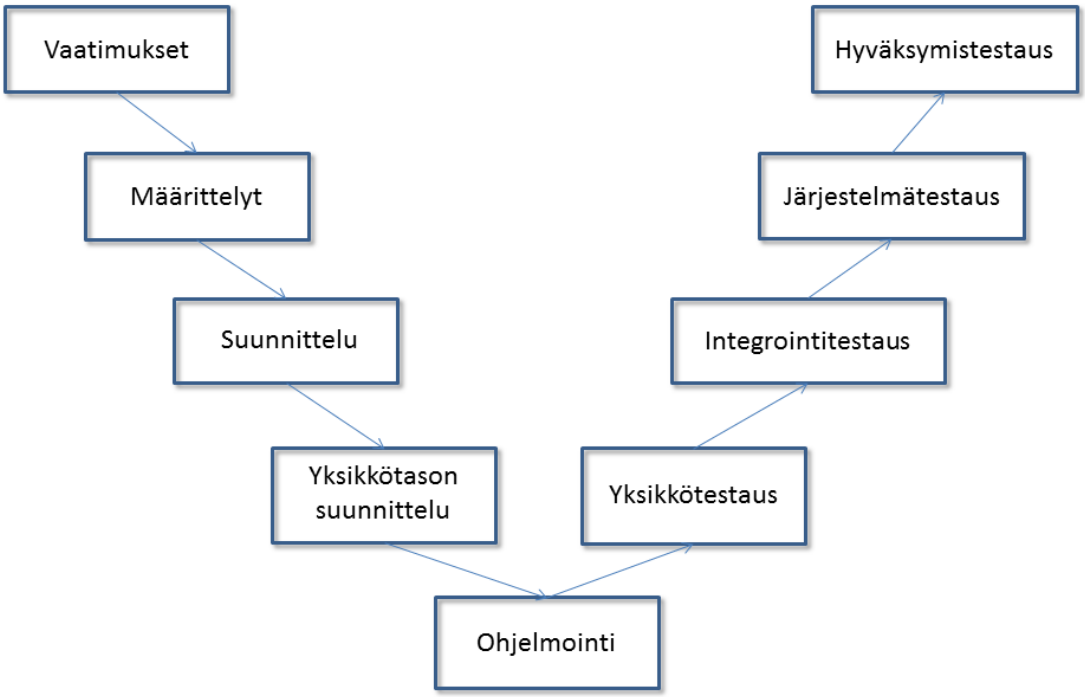
Järjestelmä käyttää tietokantana PostgreSQL-kantaa. Palvelin tallentaa tietokantaan käyttäjätietojen lisäksi muun muassa tehdyt tilit eri käyttäjille ja halleille, transaktiot ti-

leistä toiselle, hallien tiedot, omistetut laput eri peleihin ja pelien lokit, joissa on arvotut numerot ja kyseessä olevassa pelissä käytetyt sääntöyhdistelmät.

5 Testauksesta yleensä

Ohjelmistotestauksessa on useita testaustyyppejä, jotka liittyvät ohjelman kehityksen eri vaiheisiin. Perinteisessä vesiputousmallissa jokaista ohjelman kehitysvaihetta vastaa siihen liittyvä testauksen kohde (ks. kuva 5). Vaikka projektin kehittäminen onkin vesiputousmallia ketterämpää, testauskohteet ovat kuitenkin osittain samoja.

Järjestelmän testausta alettiin suunnitella tekemällä erilaisia testejä, jotka sijoittuvat eri testaustasoihin. Osa testeistä oli järjestelmätestejä, jotka testasivat järjestelmää ja sen kestokykyä kokonaisuutena. Osa taas oli yksikkötestejä, jotka keskittyivät järjestelmän yksittäisiin osiin ja niiden toimintaan eri tilanteissa, kuten yhteyskatkoksien sattuessa.



Kuva 5 Ohjelmistokehityksen perinteinen vesiputousmalli

Kuvassa 5 on ohjelmistokehityksessä perinteisesti käytetty vesiputousmalli. Projektissa testaustavat keskittyivät sekä alimman tason yksikkötestaukseen että ylempänä olevaan järjestelmätestaukseen. Myöhemmin tutkittiin myös mahdollisuutta käyttää ylimmän tason hyväksymistestaukseen tarkoitettuja ohjelmistokehyksiä.

6 Järjestelmätestaus

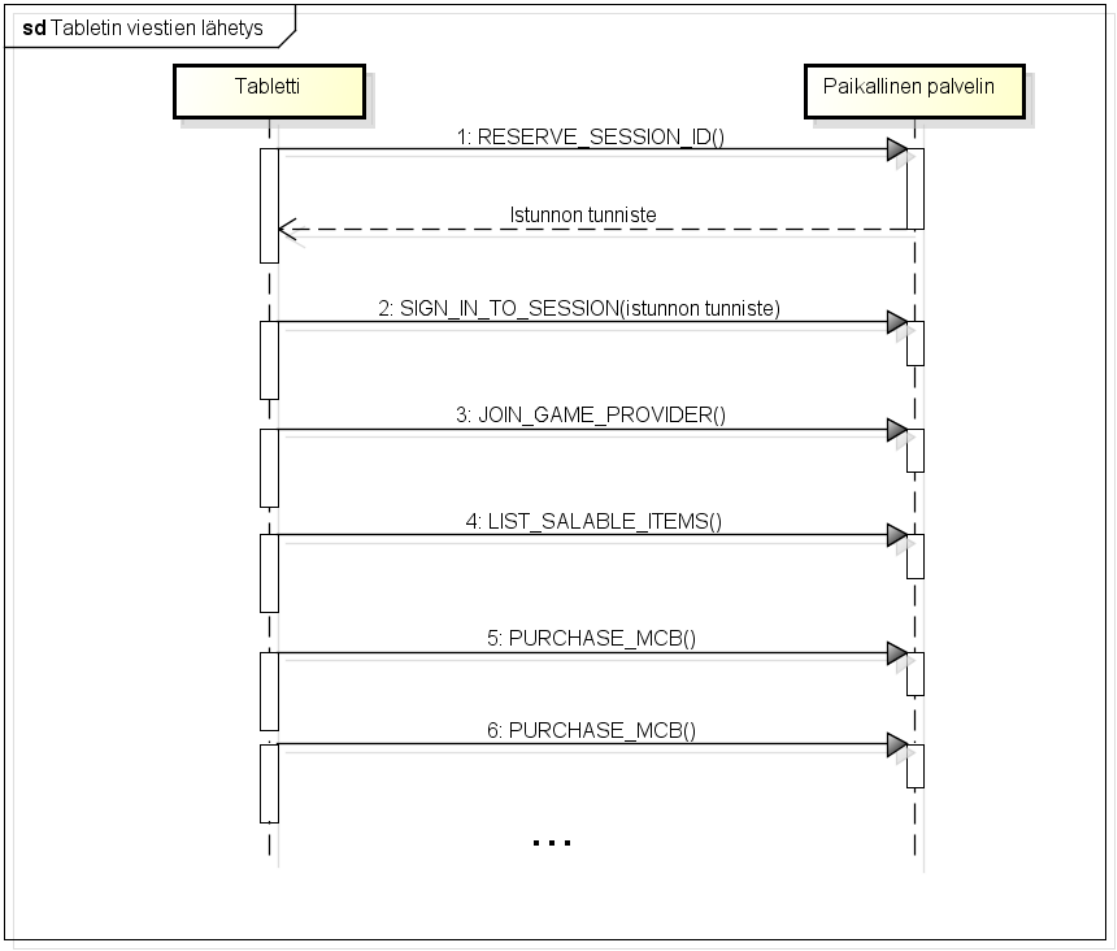
Järjestelmätestauksen tarkoituksena on testata järjestelmää kokonaisuutena, jotta nähdään, toimiiko se, kuten sen kuuluisi ja kuten sille asetetut vaatimukset edellyttävät [Software Testing Fundamentals]. Järjestelmätestaukseen kuuluvat esimerkiksi stressitestit, käytettävyystestit ja turvallisuustestit. Ensimmäisenä alettiin tehdä itsenäisesti toimivaa jäljitelmätablettia, joka toimii asiakasohjelmana testattaessa järjestelmää kokonaisuutena. Tablettia voisi käyttää tarvittaessa muutenkin järjestelmää testattaessa, jos esimerkiksi halutaan luoda tietokantaan iso määrä ostoja tai voittotapahtumia. Kuitenkin alkuperäinen tavoite jäljitelmätabletille oli mahdollistaa järjestelmän rasitustestaus ja kuormittaminen.

6.1 Rasitustestaus

Järjestelmää kehitettäessä sitä testaa yleensä vain yksi kehittäjä kerrallaan omassa kehitysympäristössään tai kaksi esim. käyttäjän kahdelta koneelta kirjautumista testattaessa. Tämä on kuitenkin epärealistinen tilanne verrattuna todellisuuteen, jossa käyttäjiä on useita samanaikaisesti. Palvelua tulisi siis pyrkiä testaamaan suurellakin käyttäjämäärällä, joko järjestämällä testaajajoukko etsimään bugeja tai sitten simuloimalla rasitusta. Tällä tavoin järjestelmästä pyritään löytämään hidastavia pullonkauloja ja niihin korjaukset.

Testiympäristön rakentaminen alkoi jäljitelmätabletin ohjelmoimisella. Tarkoituksena oli tehdä ohjelma, joka saa peliin liittyviä viestejä palvelimelta ja kommunikoi tämän kanssa kuten normaali tabletti, mutta ei vaadi käyttäjän interaktiota. Ohjelman tuli toimia täysin itsenäisesti ja sitä tuli pystyä ajamaan valitulla käyttäjämäärällä, jota simuloidaan säikeillä.

Kun tabletti liittyy peliin, se suorittaa tietyn tapahtumasyklin, joka on esitetty seuraavassa sekvenssikaaviossa kuvassa 6.



Kuva 6 Tabletin viestit liityttäessä peliin

Ensin tabletti lähettää viestin, jolla se rekisteröi istuntonsa palvelimella ja saa tunnisteen istunnolleen (viestityyppi RESERVE_SESSION_ID). Tämän jälkeen se kirjautuu istuntoon saamallaan tunnisteella (SIGN_IN_TO_SESSION), jonka jälkeen se saa tiedon pelintarjoajista, joista johonkin se liittyy (JOIN_GAME_PROVIDER). Tämän onnistuttua se lähettää kyselyn tuotteista, joita on ostettavissa käynnissä olevaan peliin tai mitä yleensä pääpelin tapauksessa on jo ostettu (LIST_SALABLE_ITEMS). Saatavilla olevat ostamattomat tuotteet ostetaan kaikki (PURCHASE-viestityyppi, joka tässä on nimetty PURCHASE_MCB, koska ostetut ruudukot olivat käteisbingoruudukoita. Ostettavia ruudukoita oli neljä/tabletti). Sitten tabletti alkaa vastaanottaa palvelimelta pelin edetessä arvottuja numeroita ja merkkaa bingolappuja sen mukaan. Kun oikea voittoyhdistelmä löytyy jostain lapusta, tabletti lähettää bingoviestin palvelimelle, joka lähet-

tää sen edelleen ECM-järjestelmälle0. Kun kaikki kierrokset on pelattu, palvelin aloittaa uuden pelin, ja pelaaminen jatkuu.

Jotta vasteaikoja voitaisiin mitata luotettavasti, säikeiden lähettämiin viesteihin liitettiin vastauksenkäsittelijä. Kun lähetettyyn viestiin saadaan vastaus, otetaan aika, joka viestillä kesti, vastauksen sisältö käsitellään, ja seuraava viesti voidaan lähettää. Ohjelma on siis täysin tapahtumiinreagoiva ja viestit pystytään ketjuttamaan peräkkäin ilman turhaa odotusta. Viestien vastaanottamisen jälkeen kuitenkin lisättiin satunnainen lyhyt (500–1000 ms) viive, jotta pelaajien normaalia reaktioaikaa voitaisiin mallintaa luonnollisemmin ja kuorma jakautuisi realistisemmin. Tätä aikaa ei kuitenkaan laskettu ajanottoon.

6.2 Rasitustestauksen tulokset

Järjestelmä toimi hyvällä vasteajalla, kun bingoa pelattiin yhdellä tai muutamalla tabletilla. Kun laitteiden määrää lisättiin ensialkuun sataan, alkoivat vasteajat kasvaa. Ensimmäisenä lähetettävään viestiin (RESERVE_SESSION_ID) suurin osa tableteista sai vastauksen, ja se tuli myös kohtalaisessa ajassa. Tämän jälkeen kasvu oli eksponentiaalista ja laitteiden, jotka saivat viimeisinä lähetettyä viestin, käytettävyys oli huono, sillä vasteaika oli minuutteja. Ylipäätään peliin liittyminen oli mahdotonta ja epäonnistui aikakatkaisun tullessa. Jos tabletit olivat päässeet liittymään peliin, saatiin pelin aikana pelitapahtumat lähetettyä tableteille jouhevasti ilman viivettä, sillä kyseessä olivat vain yksittäiset viestit numeroita arvottaessa ja bingosta ilmoitettaessa. Ongelma ilmeni vain pelin käynnistyksen aikana.

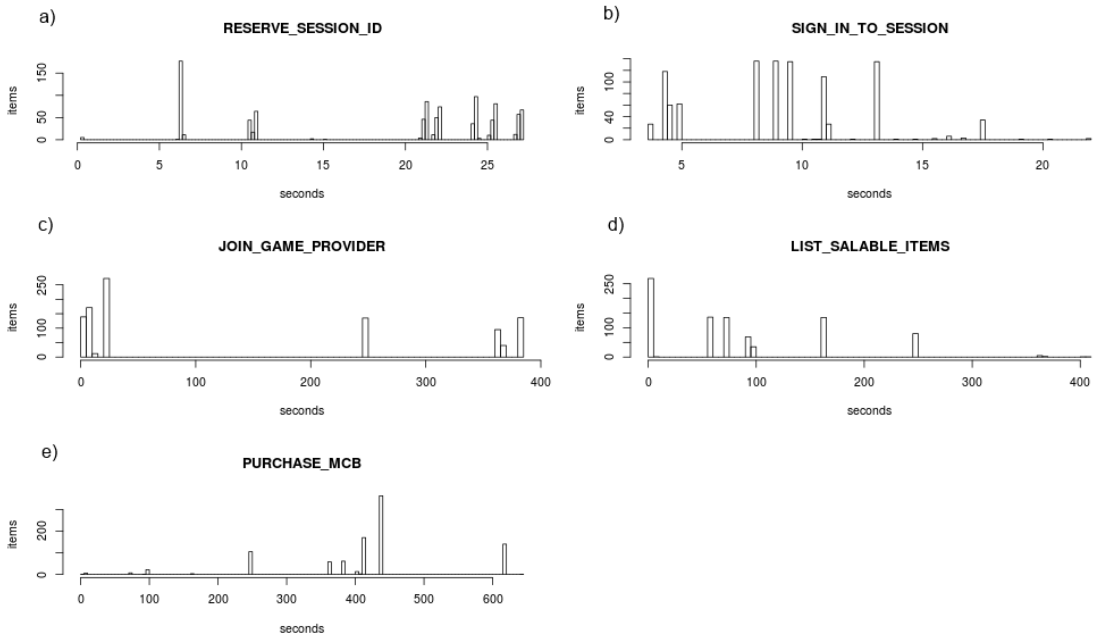
Kun ongelmaa kartoitettiin, piti ensimmäiseksi varmistaa, että mitatuissa testaustuloksissa havaitut ongelmat johtuvat todella järjestelmästä eikä testausmenetelmissä olevasta virheestä. Viestin lähetyksen ja vastaanottamisen välinen viive mitattiin tarkasti ennen viestin lähtöä ja saapumista eikä ajanottoon ollut otettu mukaan epähuomiossa muita säikeiden suorittamia laskuoperaatioita tai nukkumista. Aika lisättiin listaan, joka sisälsi ko. tapahtumatyypin mitatut ajat. Kun tabletit olivat saaneet viesteihinsä vastauksen, tallennettiin muistissa olevat ajat kiintolevylle tekstitiedostoon. Tämän jälkeen ajat sisältävät tiedostot avattiin R-ohjelmaan käsittelyä varten. R sisältää useita valmiita kirjastoja tilastotiedon käsittelyyn ja esittämiseen. Tässä tapauksessa sillä tehtiin yksin-

kertainen muutos aikayksiköstä toiseen ja histogrammien piirtäminen selitekenttineen. Seuraavassa koodiesimerkissä 4 on datan käsittelyyn käytetty skripti.

```
#ladataan tabletin ja palvelimen mitatut ajat kahteen muuttujaan
results        <-        read.table("/home/tpiippo/Documents/oppari/R-
data/test_results_after.txt", header=TRUE, sep="\t")
resultsls       <-       read.table("/home/tpiippo/Documents/oppari/R-
data/test_ls_results_after_rearranged.txt", header=TRUE, sep="\t")

#histogrammit piirretään kolmeen riviin ja kahteen sarakkeeseen
par( mfrow = c( 3, 2 ) )

#iteroidaan sarakkeet läpi
for ( i in 1:ncol(results) ) {
  #muunnos millisekunneista sekunneiksi
  data <- sapply(results[,i], function(x){x/1000});
  #tehdään histogrammi datasta ja määritetään välien määrä ja otsikot
  hist(data,       breaks=120,       xlab="seconds",       ylab="items",
main=colnames(results)[i])
  #printtaus
  print(colnames(results)[i]);
}

#samat operaatiot palvelimen datalle
par( mfrow = c( 3, 2 ) )

for ( i in 1:ncol(resultsls) ) {
  data <- sapply(resultsls[,i], function(x){x/1000});
  hist(data,       breaks=120,       xlab="seconds",       ylab="items",
main=colnames(resultsls)[i])
  print(colnames(resultsls)[i]);
}
```

Koodiesimerkki 4 R-ohjelman skripti datan käsittelyyn

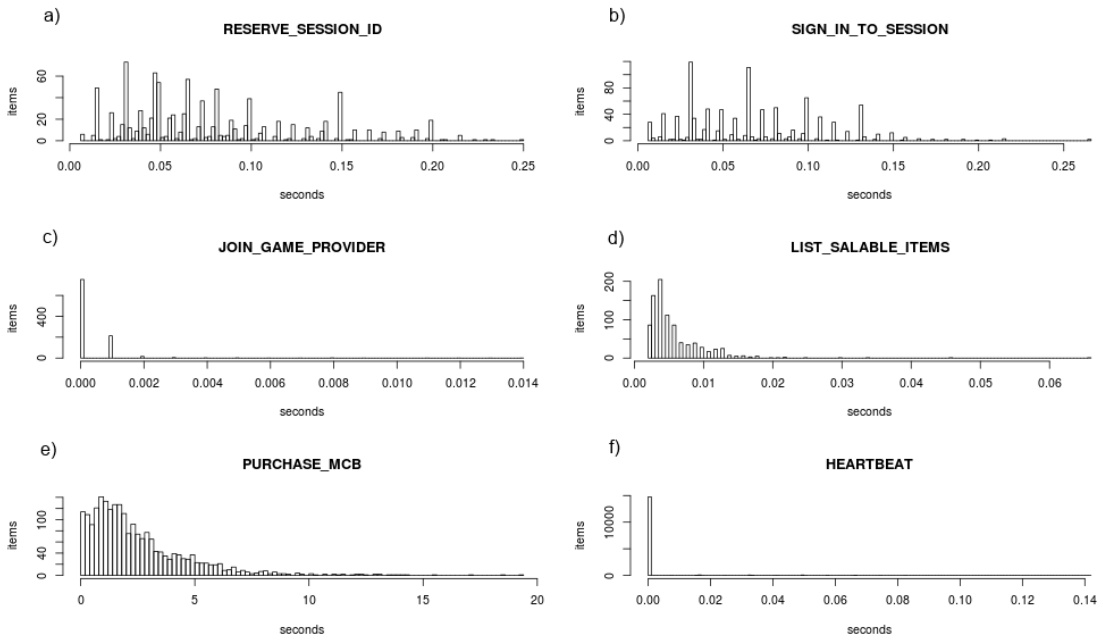
Koodiesimerkissä ladataan ensin data tiedostoista taulukkomuuttujiin. Ladattaessa määritellään, että datasarakkeiden ylimmällä rivillä on otsikko ja erotinmerkki on tabulaattori. Tämän jälkeen määritellään histogrammien järjestys: kolme riviä ja kaksi saraketta. Sitten taulukkomuuttujan sarakkeet käydään läpi, ja jokainen arvo jaetaan tuhannella sekunneiksi muuntamisen vuoksi. Lopuksi tehdään histogrammi, jolle annetaan välien määrä ja otsikot ja se printataan. Sama tehdään vielä palvelimen tuloksille. Seuraavassa kuvassa 7 on tabletin testausdatasta piirretyt histogrammit.



Kuva 7 Histogrammit tabletin viestien vasteajoista

Kaaviosta a) näkee, että suuri osa tableteista sai vastauksen ensimmäisenä lähettyyn viestiin kohtuullisessa ajassa. Histogrammi on korkeimmillaan noin kuuden sekunnin kohdalla, ja seuraava piikki on 10 ja 11 sekunnin tienoilla. Tämä on vielä siedettävän rajamailla, mutta toisaalta noin puolet mitatuista ajoista kesti yli 20 sekuntia. Seuraavana lähetetyn viestin käsittelyajat kaaviossa b) ovat suurimmaksi osaksi alle 10 sekuntia eikä kovin moni mene yli 15 sekunnin. Suurimmat erot viestien keskinäisiin latensseihin alkavat syntyä kolmannesta viestistä lähtien, kuten näkyy kaaviossa c). Vaikka suurin osa ajoista jää kaavion alkupäähän, vie huomattava osa yli 250 sekuntia, ja osa vieläpä vajaa 400 sekuntia. Myös neljännen viestin käsittelyajat vaihtelevat suuresti kaaviossa d), mutta kaaviosta e) huomaa, että ostoviestien viive oli järjestään todella pitkä: ensimmäinen merkittävä piikki näkyy noin 250 sekunnin kohdalla, ja loput viiveet olivat vielä pidempiä. Pahimmillaan käsittely kesti yli 600 sekuntia. Jokaisen tabletin lähettämästä kahdeksasta viestistä neljä oli ostoviestejä, joten niitä oli määrällisesti eniten, ja ne aiheuttivat samantapaista kuormitusta palvelimen päässä. Huomaamisen arvoista on, että ostoviestejä kuvaavasta kaaviosta puuttuu suuri osa viesteistä, jotka olisi pitänyt lähettää. Jo tällä viestimäärällä vastauksen saaminen kesti testin loppuessa yli 10 minuuttia, joten kaikkien viestien vastauksen odottaminen olisi kestänyt todella pitkään. Jo tässä vaiheessa ongelmat kävivät ilmi.

Samaa vertailua tehtiin myös palvelimen päässä eri viestityyppien käsittelyyn kuluneesta ajasta. Seuraavassa kuvassa 8 on histogrammit ajoista palvelimen päässä.



Kuva 8 Histogrammit palvelimen viestien käsittelyajoista

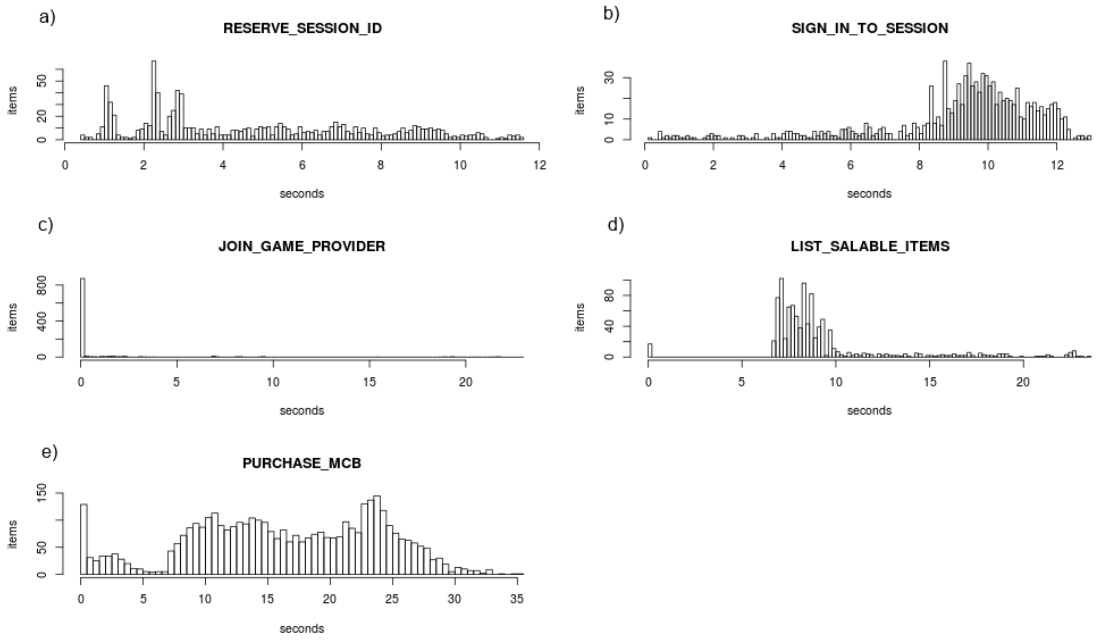
Toisin kuin oli etukäteen odotettavissa, palvelimen käsittelyajat eivät kasvaneet samalla tavalla kuin tabletin vastaavat, vaan tulokset olivat joko jakautuneet melko tasaisesti, kuten kaavioissa a) ja b) tai sitten jäivät histogrammin alkupäähän. Kaavioista pistää silmään myös, että mitatut ajat ovat hyvin lyhyitä, reilusti alle sekunnin. Pelintarjoajaan liittyminen kaaviossa c) kestää siksi niin vähän aikaa, koska tällä pyynnöllä ei jouduta tekemään tietokantahakuja, vaan tarvittavat tiedot voidaan nopeasti hakea muistissa olevista tietorakenteista ja palauttaa asiakasohjelmalle. Ainoa joukosta poikkeava on ostoviesti, joista suurin osa vei alle viisi sekuntia ja maksimissaankin 20 sekuntia. Tästä heräsi kysymys siitä, onko ongelma esimerkiksi Minan tavassa käsitellä viestejä vai siinä, että sekä tabletteja, palvelinta että ECM-järjestelmää ajettiin lokaalisti samalla koneella. Jos ongelma olisi johtunut jälkimmäisestä syystä, olisi se luonnollisesti ollut epäolennaista käytettäessä järjestelmää, kuten sitä pitäisi käyttää, eli jokaista komponenttia ajettaisiin erillisellä koneellaan. Histogrammissa f) oleva HEARTBEAT-viestityyppi on viesti, joka kertoo, että yhteys on vielä auki eikä sitä tule sulkea, eli niin sanottu keepalive-viesti. Sen käsittely ei vaadi kummempia toimenpiteitä, joten se ei myöskään vie paljon aikaa.

6.3 Muutokset ja uudet tulokset

Kun tutkittiin syytä viestien käsittelyn hidastumiseen, huomattiin, että se alkoi pelilappujen oston aikana ja hidasti myös muita myöhemmin tulevia viestejä. Oston aikana haettiin tietokannasta peliruudukkoja käynnissä olevan pelin perusteella. Tämän jälkeen, jos haluttua lappua ei vielä ollut ostettu, tehtiin ostotoimenpiteet. Lopuksi lähetettiin ECM-liitännäiselle raportti ostetuista lapuista, jonka se välitti edelleen ECM-järjestelmälle. Näissä toimenpiteissä löytyi muutama ongelma. Yksi aiheutui siitä, että jo ostettuja lappuja haettaessa tehtiin useaan kertaan samantapaisia tietokantahakuja, jotka olivat hitaita suorittaa. Tämä korjattiin yhdistämällä hakuja ja muuttamalla koodia siten, että yhdellä haulla haettuja tietoja pystyttiin käyttämään myös myöhemmin paikassa, missä ennen oli tehty uusi tietokantahaku. Toisena korjauksena poistettiin myös synkronoituja lohkoja. Useassa kohdassa oli synkronoitu lohkoja, vaikka niiden sisällä suoritettava koodi ei edes muuttanut sellaisia tietoja, mitä toiset säikeet olisivat käyttäneet.

Yksi ongelmista puolestaan johtui myyntiraportin lähettämisestä. Myyntiraportti jouduttiin koostamaan uudelleen jokaisen yksittäisen oston jälkeen, joten tämä luonnollisesti oli pullonkaula suurella määrällä samanaikaisia ostoja. Ratkaisuna oli tehdä raporttien lähettämisestä säikeessä suoritettava toimenpide, joka tehtiin säännöllisin väliajoin uusien ostojen tultua. Toisin sanoen jokainen osto ei aiheuttanut raportin koostamista ja lähettämistä, vaan ostot lähetettiin kerralla isompina kimppuina. Tästä seurasi heti käsittelyaikojen jakaantuminen tasaisemmin tabletin päässä.

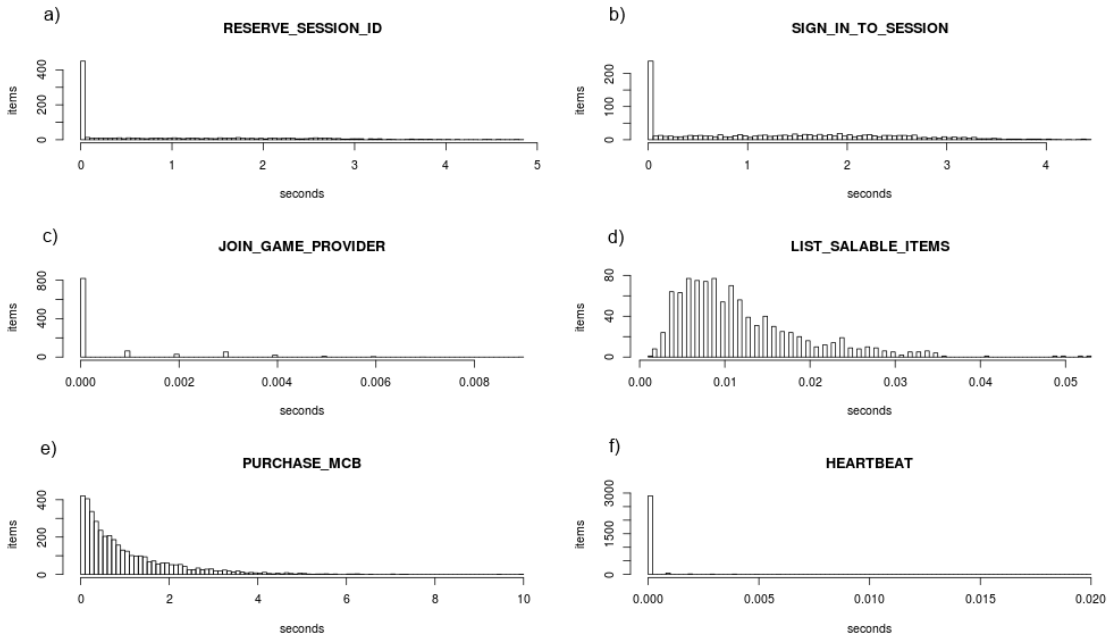
Kuvassa 9 on nähtävissä histogrammit tabletin viestinkäsittelyajoista muutosten jälkeen.



Kuva 9 Histogrammit tabletin viestien vasteajoista parannusten jälkeen

Kuvaajista näkee, kuinka isoimmat piikit ovat levittäytyneet jonkin verran tasaisemmin. Kahdessa ensimmäisessä kuvaajassa a) ja b) ajat ovat suurin piirtein samaa luokkaa kuin ennen muutosta, mutta suurin hidastuminen aikaisemmin tapahtuikin näiden viestien jälkeen. Ajat lyhenivät eniten näiden viestien jälkeen pelintarjoajaan liittymisessä kaaviossa c), myytävien tuotteiden kyselyssä kaaviossa d) ja erityisesti viimeisenä lähetetyssä ostoviestissä kaaviossa e). Kaavioissa c) ja d) oli aikaisemminkin osa viesteistä käsitelty nopeasti, mutta pisimmät yli 100 sekunnin käsittelyajat hävisivät. Kaavion e) ostoviestit olivat ennen muutoksia suurimmaksi osaksi 400 sekunnin ympärillä ja pahimmillaan yli 600 sekuntia. Korjausten jälkeen yksikään viesti ei vienyt noin 35 sekuntia kauempaa.

Kuvassa 10 näkyvät paikallispalvelimen viestien käsittelyajat korjausten jälkeen.



Kuva 10 Histogrammit palvelimen viestien käsittelyajoista parannusten jälkeen

Myös palvelimen ajat ovat jakaantuneet paremmin kuvaajien alkupäähän. Jotkut viestit käsiteltiin itse asiassa hieman hitaammin kuin aiemmin, kuten istunnon tunnisteen varaaminen kuvaajassa a) ja istuntoon kirjautuminen kaaviossa b). Ennen muutoksia tunnisteen varaamisessa aikaa meni korkeimmillaan 0,25 sekuntia, kun taas muutosten jälkeen aikaa meni enimmillään vajaa viisikin sekuntia. Sama ilmiö on havaittavissa istuntoon kirjautumisessa kuvaajassa b); ennen muutoksia aikaa meni pisimmillään hieman yli 0,25 sekuntia ja muutosten jälkeen pisimmillään noin 4,5 sekuntia. Tällä ei kuitenkaan ole niin suurta merkitystä, sillä ostoviestien käsittelyssä on tullut melko suuri parannus. Kuvaajat ovat suurin piirtein samanmuotoisia, mutta siinä missä suurin osa viesteistä jäi aikaisemmin 10 sekunnin alle, jäävät ne nyt 4-5 sekunnin alle. Pisimmillään ostoviestin käsittely kesti nyt hieman alle 10 sekuntia ja aikaisemmin 19,3 sekuntia.

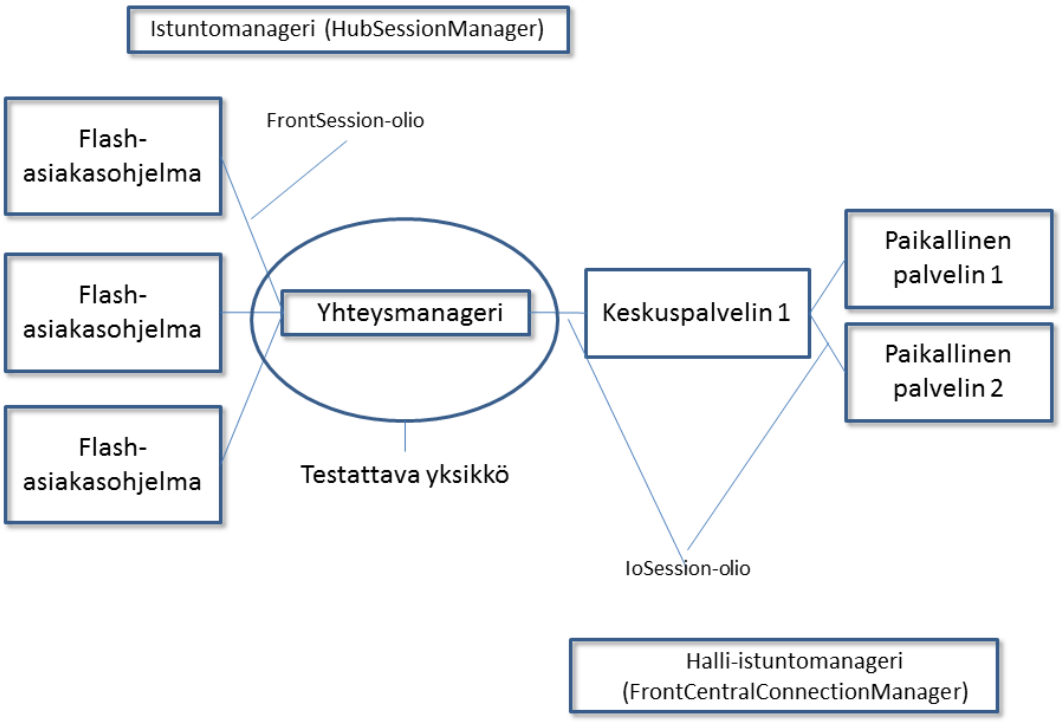
7 Yksikkötestaus

Yksikkötestauksessa testataan nimensä mukaisesti ohjelman erillisiä yksiköitä. Testattavana on usein ohjelman luokka, mutta testissä voi olla myös luokan osat [Myers, 2013, 85]. Näin pystytään testaamaan tarkemmin koodi, koska keskitytään pienempiin osiin, ja löydetyt ongelmat voidaan paikallistaa nopeammin. Yksikkötestaus on luon-

teeltaan lasilaatikkotestausta, sillä siinä osan toiminta tunnetaan yksityiskohtaisesti, jolloin testattavalle osalle pystytään laatimaan testaustapauksia kattavasti. Koodin tuntemuksen lisäksi tulee tietää ohjelman osan määrittely, jotta testattavat syötteet pystytään valitsemaan niin, että kaikki tarvittavat ekvivalenssiluokat tulee testattua. Jotta testattavan osan saa eristettyä järjestelmän muista komponenteista, voi joutua kirjoittamaan tynkätoteutuksia tai ns. jäljittelyolioita muille palveluille, joita se käyttää. Nämä tyngät toteuttavat saman rajapinnan kuin aidot vastineensa, mutta niiden toiminnallisuus on rajoitettu testiin sopivaksi. Ne eivät esimerkiksi käytä välttämättä verkkoyhteyksiä muihin palveluihin, mikä normaalisti tapahtuisi. Ne voivat myös palauttaa metodikutsun tuloksena testitapauksen mukaan haluttuja tuloksia.

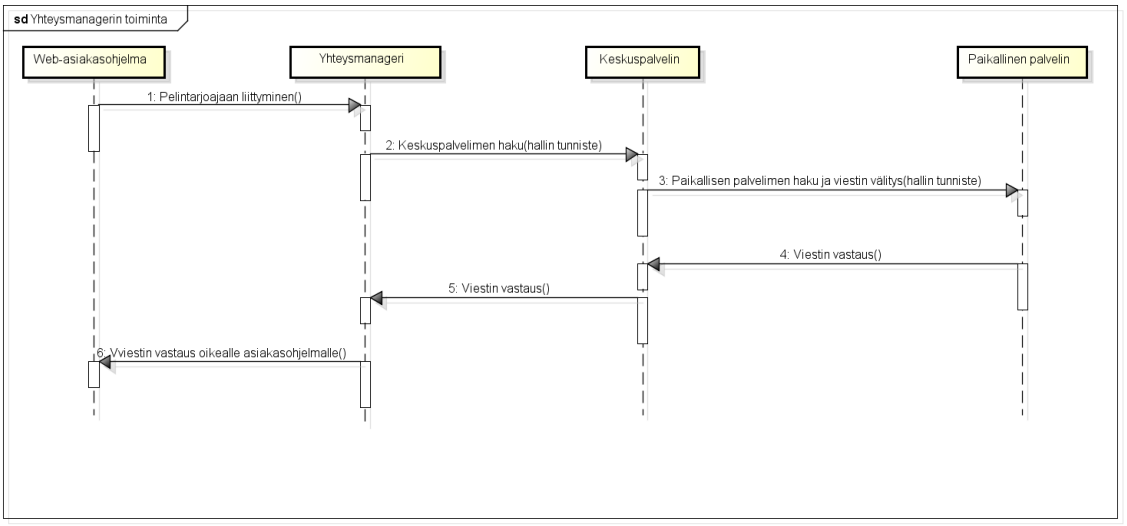
7.1 Yhteysmanagerin testaus

Palvelun käyttäjille tarkoitetut web-sivustot käyttävät niiden ja keskuspalvelimen välissä olevaa yhteysmanageriluokkaa. Tälle luokalle oli alettu tehdä yksikkötestiluokkaa JUnitilla. Luokan tarkoituksena on testata yhteysmanagerin toimintaa erilaisissa tilanteissa. Testit olivat kuitenkin jääneet kesken ja niitä ei ollut kovin montaa. Kuvassa 11 on esitetty testattavan osan rakenne.



Kuva 11 Testattava ohjelman osa ja liittyvät luokat

Kuvassa näkyy yhteysmanagerin asema asiakasohjelmien ja palvelimien välissä. Yhteysmanagerin tehtävänä on välittää viestejä web-asiakasohjelmien ja palvelimen välillä. Tämä on esitetty seuraavassa kuvassa 12.



Kuva 12 Yhteysmanagerin toiminta

Viestit web-sivustoilta, kuten pelisivuston flash-asiakasohjelmalta, välitetään yhteysmanagerille. Tämä välittää ne oikealle keskuspalvelimelle viestissä olevan hallin tunnisteen perusteella. Kun viesti saapuu keskuspalvelimelle, se ohjataan sieltä edelleen oikean hallin paikallispalvelimelle. Kun viesti on käsitelty, lähetään sen vastaus vastaavasti takaisin hallista keskuspalvelimelle, siitä yhteysmanagerille, joka palauttaa sen edelleen alkuperäisen viestin lähettäneelle flash-asiakasohjelmalle. Testitapauksissa otettiin huomioon useat asiakasohjelmat ja mahdolliset yhteyksien katkeamiset eri osien välillä. Tässä testiluokassa testattiin vain yhtä keskuspalvelinta, mutta niitä voisi periaatteessa olla useampiakin.

Yhteysmanagerin ja keskuspalvelimen sekä keskuspalvelimen ja paikallisten palvelimien välissä on Apache Minan tarjoama IoSession-luokka, joka hoitaa viestien kirjoittamisen datavirtaan. Testiluokkaa varten tehtiin tätä vastaava TestIoSession-luokka. Se toteutti saman rajapinnan, mutta metodien toteutukset olivat tyhjiä. FrontCentralConnectionManager-luokka hallinnoi näitä istuntoja. Tältä voi saada IoSession-istunnon hallin nimen perusteella tai hallin sitä vastaavan istunnon perusteella.

Asiakasohjelmille, kuten netin bingo-ohjelmistolle, on tehty FrontSession-luokka, joka sisältää muun muassa yhteyden istunnon tunnisteet ja nimen. Istunnoilla on myös viestien kirjoittamista ja lukemista varten edellä mainittu Apache Minan tarjoama IoSession-olio. Näitä FrontSession-istuntoja hallinnoi HubSessionManager-luokka, jolta voi saada istunnon tunnisteen perusteella tai rekisteröidä sen sisäänkirjautuessa. Testattavat istunnot olivat tätä tyyppiä.

7.2 JUnitin käyttö testeissä

JUnit on Javalle kehitetty yksikkötestauskehys, joka tarjoaa hyvät työkalut yksikkötestien rakentamiseen, valmisteluun, tulosten varmistamiseen ja testien ajamiseen ryhminä automaattisesti muutosten jälkeen [Unit Testing Your Application with JUnit, 2010]. JUnit onkin käytännöllinen regressioiden testaamiseen; kun ohjelman koodia muutetaan myöhemmin esimerkiksi uuden ominaisuuden lisäämisen tai suorituskyvyn optimoimisen myötä, pystytään JUnitilla testaamaan, toimivatko aiemmat ominaisuudet vielä halutulla tavalla. Testeissä käytettiin myös EasyMock-kirjastoa, jonka avulla voi luoda luokista jäljitelmäversioita, jotka palauttavat haluttuja olioita, ja varmistaa esimerkiksi, onko luokan metodia kutsuttu riittävän monta kertaa.

JUnit tarjoaa testiluokkien metodeihin annotaatioita, joilla voidaan määrittää, missä vaiheessa testejä tehdään mitäkin toimenpiteitä. Yhteysmanagerin yksikkötestiluokassa käytettiin annotaatioita BeforeClass, After ja Test. Tässä metodissa luotiin TestIoSession-oliot korvaamaan Minan IoSessioneita. Alustusmetodissa luotiin myös halli-istuntomanageriolio, jolle lisättiin myös jäljitellyt metodit, joilla saatiin IoSession-olio hallin tunnisteen perusteella. Halli-istuntomanagerille lisättiin myös metodi, jolla saatiin bingohallit, jotka tässä tapauksessa olivat vain merkkijonoja tyyliin ”H41-” ja ”H42-”.

Yksikkötestiluokan @After-purkumetodissa, nimeltään tearDown, kaikki web- istuntojen IoSessionit nollattiin, koska muuten testit epäonnistuivat, sillä IoSession oli jo kiinnitetty istuntoon.

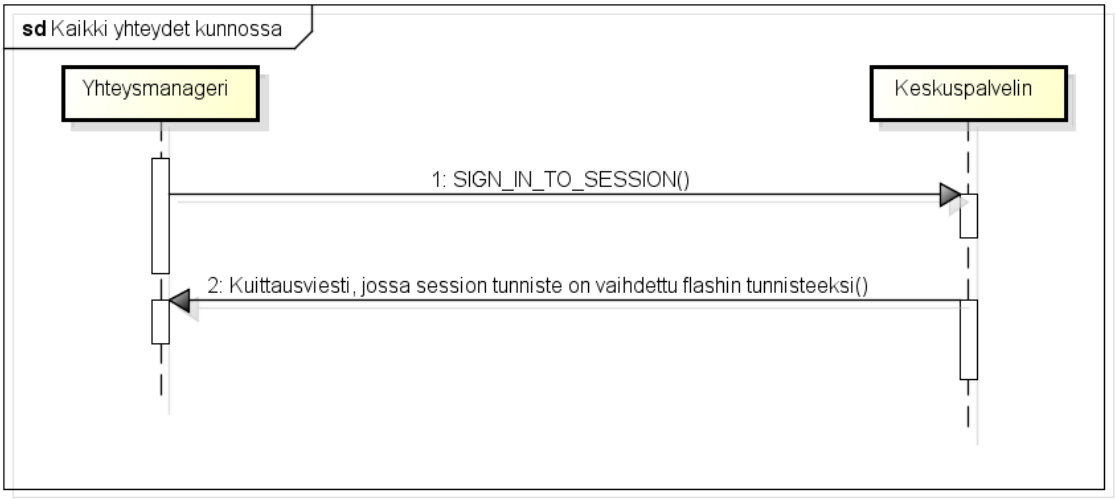
Tehtyjä yksikkötestejä voisi myös automatisoida ajettavaksi säännöllisin väliajoin jatkuvan integroinnin palvelimella, jolloin myöhemmin koodiin tulevien muutosten mahdolliset regressiot kävisivät ilmi helpommin.

7.3 Testitapaukset

Testitapauksissa testattiin, miten yhteysmanageri välittää viestejä asiakasohjelmilta keskuspalvelimelle, joka välittää ne edelleen haluttuun halliin. Manageria testattiin myös usealla asiakasohjelmalla, jolloin viesti tuli toimittaa oikealle vastaanottajalle tai mahdollisesti kaikille, jos vastaanottajaa ei ollut määritelty. Testeissä käsiteltiin myös poikkeustilanteita, joissa yhteys oli poikki tai katkesi myöhemmin.

a) Testi, jossa kaikki yhteydet ovat kunnossa

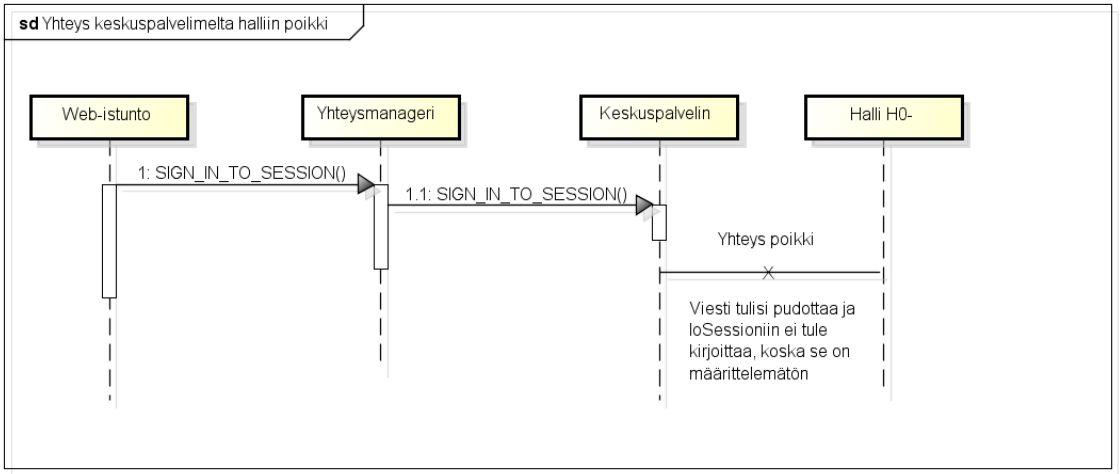
Tämän JUnit-testin tarkoituksena oli testata viestin lähetystä ja sille tehtäviä toimenpiteitä tilanteessa, jossa kaikki yhteydet toimivat. Tässä testissä luotiin TestIoSession-olio ja sisäänkirjautumisviesti ja nämä annettiin yhteysmanagerin messageReceived-metodille. Lähetetty viesti oli käärityt keskuspalvelimelle menevään CentralMessage-viestityyppiin. Vastauksen tullessa varmistuttiin JUnitin assertTrue-metodilla, että viestiin oli lisätty niin sanottu välitysviesti (Assert.assertTrue(message.hasForwardingMessage())) ja istunnon tunniste oli vaihdettu.



Kuva 13 Testi, jossa kaikki yhteydet kunnossa

b) Testi, jossa yhteys keskuspalvelimelta halliin poikki

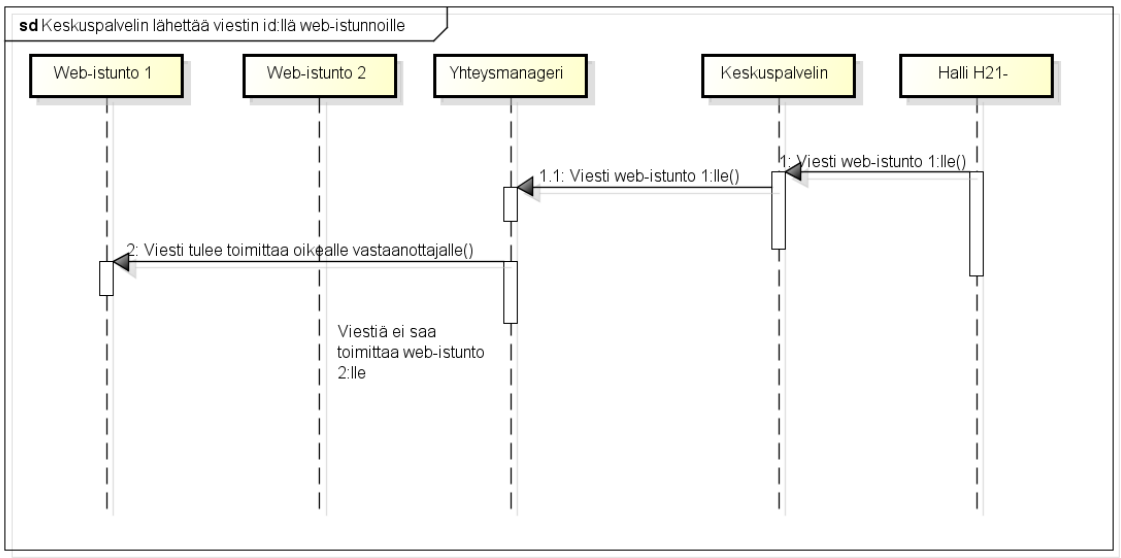
Tässä testissä tehtiin samat toimenpiteet kuin edellisessä testissä, mutta simuloitavassa tilanteessa keskuspalvelimen IoSession-olio oli määrittelemätön. Tämä vastaa tilannetta, jossa yhteys keskuspalvelimelta halliin on poikki. Testin piti epäonnistua, jos tähän sessioon yritettäisiin kirjoittaa (JUnitin Assert.fail()).



Kuva 14 Testi, jossa yhteys keskuspalvelimelta halliin poikki

c) Testi, jossa keskuspalvelin lähettää viestin tunnisteella web-istunnoille

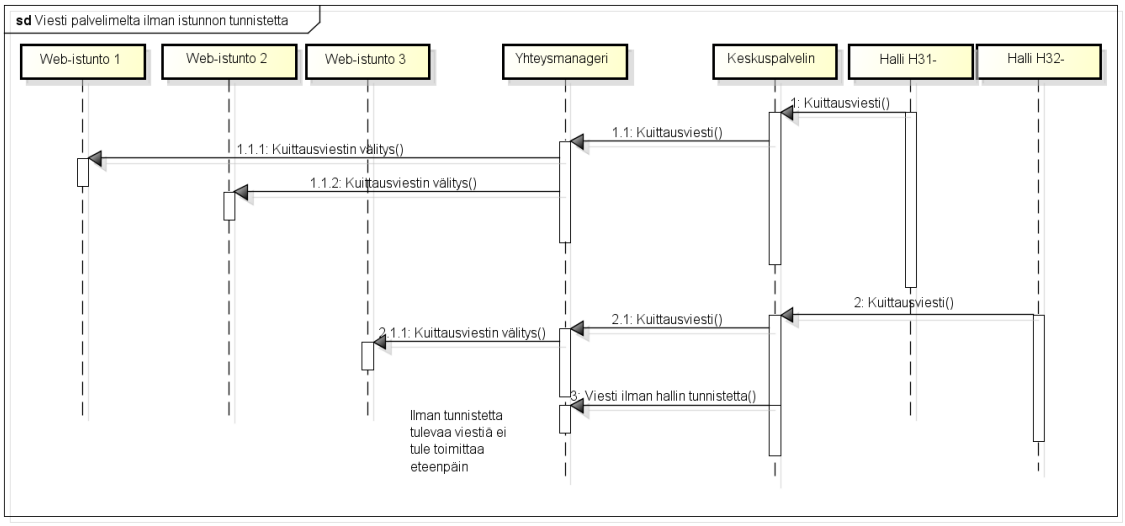
Tässä testissä luotiin kaksi web-istuntoa, ja istunnot kirjautuivat samaan halliin H21- (kuva 15). Lisäksi istunnoille luotiin jäljitelty metodi ”write”, joka matki viestin istuntoon kirjoittamista. EasyMockin expect-avainsanan avulla määriteltiin, että ensimmäisen istunnon write-metodin tuli tulla kutsutuksi kerran ja toiseen istuntoon ei saanut kirjoittaa kertaakaan. Tämä varmistettiin testin lopussa EasyMockin verify-metodilla. Testin tavoitteena oli todeta, osataanko hallista tuleva viesti välittää istunnon tunnisteen perusteella oikealle asiakasohjelmalle.



Kuva 15 Testi, jossa keskuspalvelin lähettää viestin tunnisteella web-istunnoille

d) Viesti palvelimelta ilman istunnon tunnistetta

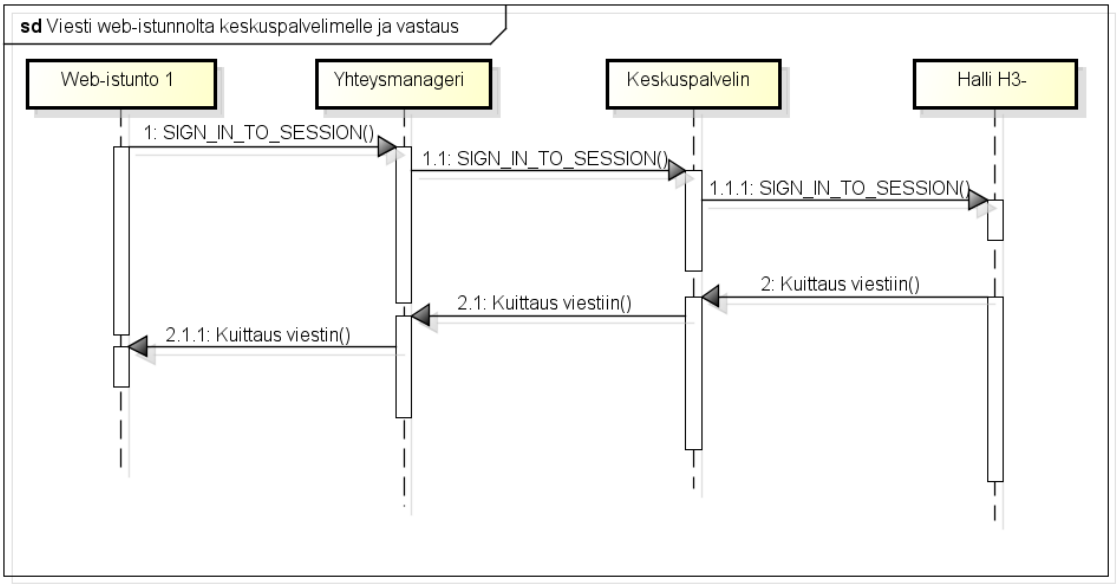
Tässä testissä luotiin kolme web-istuntoa ja niille EasyMockilla write-metodi. Kaikki istunnot kirjautuivat halleihin, kaksi ensimmäistä halliin H31- ja kolmas halliin H32-. Tämän jälkeen luotiin myös kolme testiviestiä. Ensimmäinen viesti oli keskuspalvelimelta, joka välitti sen hallilta H31-. Tämä tulisi siis toimittaa kahdelle ensimmäiselle istunnolle. Toinen viesti oli hallista H32- ja tulisi toimittaa kolmannelle istunnolle. Kolmas viesti ei sisältänyt hallin tunnistetta, ja tämä viesti tulisi tiputtaa (kuva 16). Myös tässä testissä käytettiin expect- ja verify-avainsanoja varmistamaan, että jokaisen istunnon write-metodin tuli tulla kutsutuksi tasan kerran eli testi epäonnistuisi, jos istunnot eivät saa yhtä viestiä tai jos kolmas viesti välitetään jollekin istunnolle.



Kuva 16 Viesti palvelimelta ilman istunnon tunnistetta

e) Viesti web-istunnolta keskuspalvelimelle ja vastaus

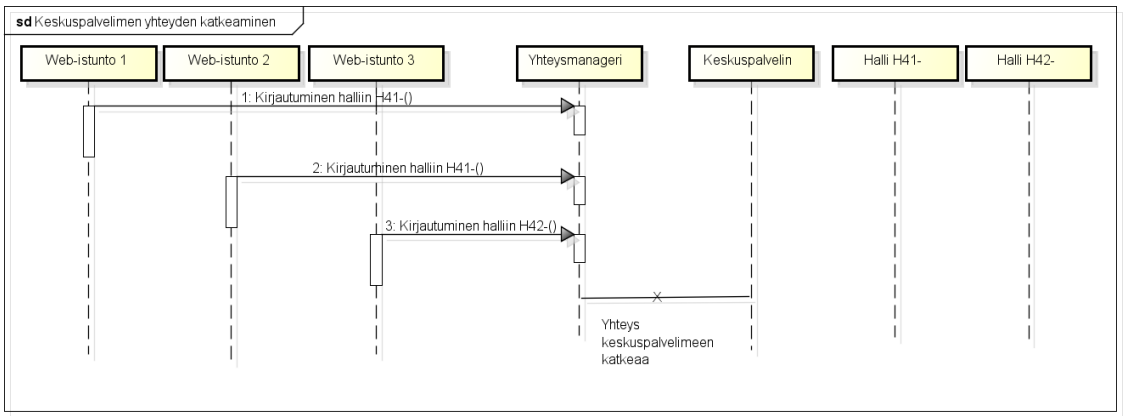
Tässä testissä web-istunto kirjoitti peliviestin keskuspalvelimeen yhdistettyyn halliin H3- (kuva 17). Viestiin tuli saada kuittaus, ja se tuli välittää takaisin viestin lähettäneelle web-istunnolle, toisin sanoen EasyMockilla varmistettiin, että web-istunnon write-metodia kutsuttiin kerran.



Kuva 17 Viesti web-istunnolta keskuspalvelimelle ja vastaus

f) Keskuspalvelimen yhteyden katkeaminen

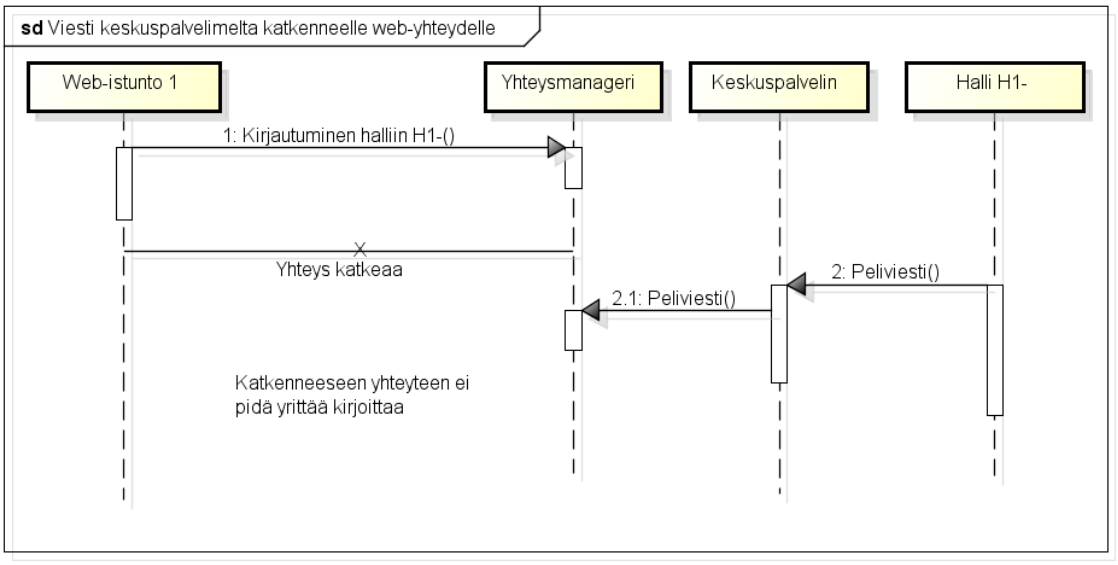
Tässä testissä tehtiin kolme web-istuntoa, joilla kirjauduttiin sisään halleihin H41- ja H42- (kuva 18). Tämän jälkeen kutsuttiin yhteysmanagerin exceptionCaught-metodia, joka vastasi keskuspalvelimen ja yhteysmanagerin välisen yhteyden katkeamista. Tämän jälkeen tarkastettiin JUnitin Assert.assertTrue-metodilla, että kaikkien web-istuntojen IoSession-oliot on suljettu.



Kuva 18 Keskuspalvelimen yhteyden katkeaminen

g) Viesti keskuspalvelimelta katkenneelle web-yhteydelle

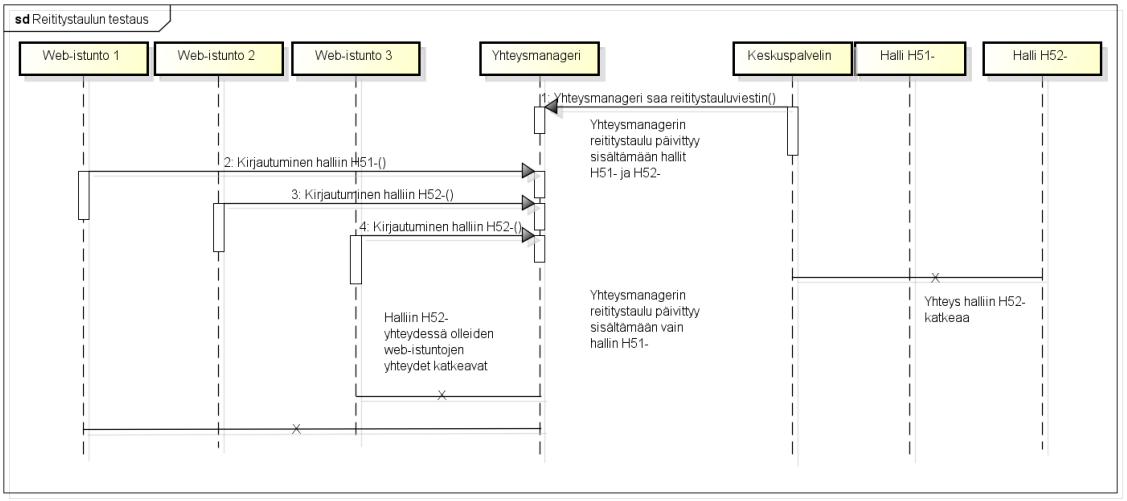
Tässä testissä tehtiin web-istunto, jolla kirjauduttiin halliin H1-. Tämän jälkeen web-istunto sulkeutui ja yhteys yhteysmanagerista siihen katkesi. Seuraavaksi keskuspalvelimelta tuli viesti, joka oli osoitettu tälle sulkeutuneelle web-istunnolle (kuva 19). Testissä asetettiin ehto EasyMockin expect-metodilla, jonka mukaan testi epäonnistui, jos jo sulkeutuneeseen web-istuntoon yritettiin kirjoittaa jotain. Jo sulkeutuneille istunnoille osoitetut viestit tulee pudottaa.



Kuva 19 Viesti keskuspalvelimelta katkenneelle web-yhteydelle

h) Reititystaulun testaus

Tässä testissä luotiin kaksi hallia: H51- ja H52-, jotka yhdistettiin yhteen keskuspalvelimeen. Tämän jälkeen lähetettiin nämä hallit sisältävä reititystauluviesti hallien yhteyksistä vastaavalla managerille. Sen jälkeen luotiin kolme web-istuntoa, joista kaksi yhdistettiin halliin H52- ja yksi halliin H51- (kuva 20). Tässä vaiheessa käytettiin JUnitin Assert.assertTrue-metodia tarkistamaan, että web-istunnoilla oli toimiva yhteys. Tämän jälkeen simuloitiin tilannetta, jossa hallin H52- yhteys keskuspalvelimeen olisi sulkeutunut, ja lähetettiin uusi reititystaulu, joka sisälsi vain jäljelle jääneen hallin H51- tiedot. Lopuksi tarkastettiin JUnitilla, olivatko sulkeutuneeseen halliin liittyneet kaksi web-istuntoa sulkeutuneet oikein eli niiden IoSession-olio oli määrittelemätön.



Kuva 20 Reititystaulun testaus

8 Hyväksymistestaus

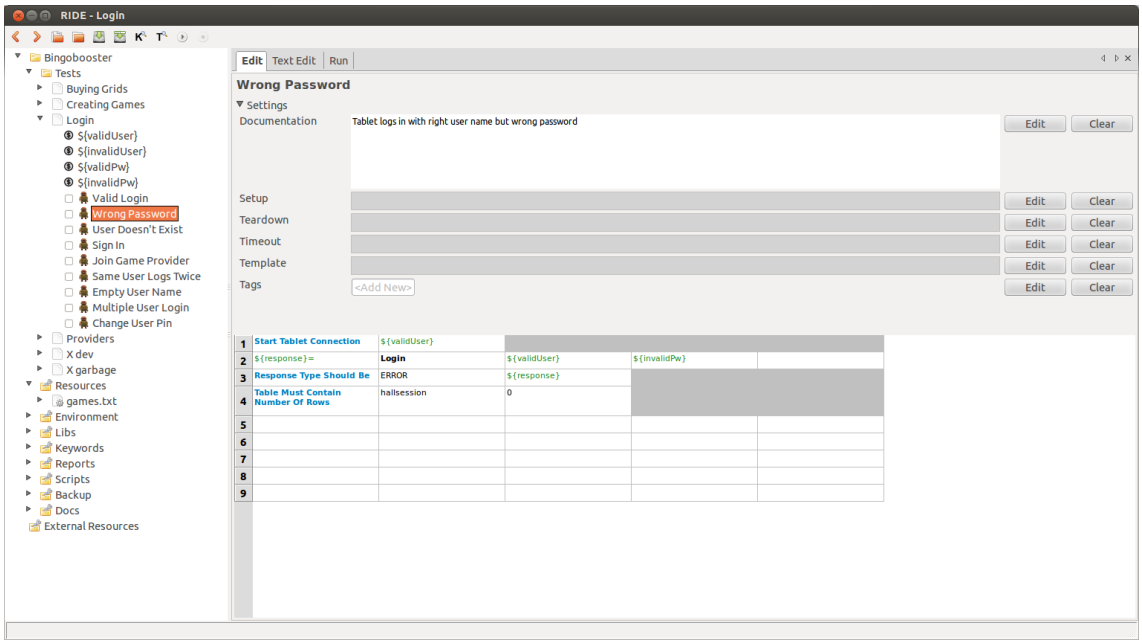
Testausjärjestelmää varten tutkittiin erilaisten testauskehysten hyödyntämistä. Testauskehyksiä on tehty moneen tarkoitukseen ja niitä löytyykin lukuisia. Ongelmana oli löytää tähän ympäristöön ja ohjelmointikieleen sopiva, joka olisi myös tarpeeksi laajassa käytössä, hyvin dokumentoitu ja aktiivinen projekti.

8.1 Robot Framework

Robot Framework on ohjelmistokehys hyväksymistestien automatisointiin ja hyväksymistestivetoiseen kehitykseen (acceptance test-driven development) [Robot Framework, 2013]. Testien kirjoittaminen on yksinkertaista, koska testidata ja testin askelet kirjoitetaan selkeästi taulukkomuotoon ja siinä voidaan hyödyntää avainsanoja. Avainsanat sisältävät yhden tai useampia toimenpiteitä ja niiden avulla toimintasarjoista voi tehdä komponentteja, joita voi hyödyntää useissa testeissä. Robotin testausmahdollisuuksia ja avainsanavalikoimaa voi laajentaa tätä varten tehdyillä valmiilla kirjastoilla, kuten kirjastoilla Androidille tai Swing-käyttöliittymille. Kirjastoja voi tehdä myös itse ohjelmoimalla ne Pythonilla tai Javalla. Jos esimerkiksi halutaan testata palveluun sisäänkirjautumista, joka sisältää useita askelia, nämä voi sisällyttää yhteen metodiin, jolla on avainsana ”Kirjaudu sisään”. Vaihtoehtoisesti jokainen askel voisi olla oma avainsanansa, ja näistä koostetaan avainsana ”Kirjaudu sisään”. Robot laatii myös

testien ajamisen jälkeen erillisen html-muotoisen raporttisivun testeistä, josta näkee, mitkä testit on suoritettu onnistuneesti ja mitkä ovat epäonnistuneet.

Testitapauksia ja testikokonaisuuksia voi kirjoittaa suoraan tekstieditorilla tai vaihtoehtoisesti RIDE-nimisellä editorilla, joka tarjoaa graafisen käyttöliittymän testien rakentamiseen. RIDE:n avulla näkee nopeasti kokonaiskuvan testeistä ja näissä mahdollisesti käytettävistä resurssitiedostoista hakemistoittain. Tämän lisäksi sen tärkeimpiin ominaisuuksiin kuuluu muun muassa avainsanojen täydentäminen, niiden dokumentaation nopea saaminen ja tarvittavien parametrien määrän ja tyypin tarkastaminen. Seuraavassa kuvassa 21 on RIDE:n käyttöliittymä.



Kuva 21 RIDE:n käyttöliittymä

Käyttöliittymän vasemmassa reunassa on ryhmitelty testit kokoelmittain. Valikossa näkyvät myös muut kansiot, kuten resurssikansio. Valtaosan käyttöliittymästä vie testieditori. Kun testikokoelma valitaan, voi määrittää kokoelmaan tuotavat kirjastot ja resurssitiedostot. Kuvassa on auki yksittäisen testin näkymä, missä voi asettaa testille alustus- ja lopetustoimenpiteitä sekä rakentaa testin. Kuvassa sinisellä näkyvät RIDE:n tunnistamat avainsanat ja niiden oikealla puolella annettavat parametrit. Dollarilla alkavat nimet ovat muuttujia. Testissä testataan tabletin kirjautumista oikealla käyttäjänimellä, mutta väärällä salasanalla. Rivillä kaksi koitetaan kirjautumista ja tallennetaan Login-funktion palauttama arvo muuttujaan ”response”. Kirjautumisen vastauksen pitäisi olla tyyppiä ”ERROR” ja testin jälkeen tietokannassa ei pitäisi olla tallennettuna yhtään is-

tuntoa. Kuten edellä mainittiin, testejä voi kirjoittaa myös tekstieditorilla. RIDE:lla kirjoitetun testin näkee tekstimuodossa yläreunan ”Text Edit” -välilehdestä. Seuraavassa koodiesimerkissä 5 on sama testi tekstimuodossa.

```
*** Settings ***
Documentation      This is a test suite to test the tablet logging in
with different credential combinations
Suite Setup         Run Keywords     Run     scripts/empty-databases.sh
AND    Start Local Server    AND    Seed Test Data
...                            100       AND       Connect To Database
org.postgresql.Driver          jdbc:postgresql://localhost:5432/dbname
username    password
Suite Teardown    Disconnect From Database
Test Setup        Delete All Rows From Table    hallsession
Library             bingobooster.autotest.tests.keywords.KeywordWrapper
WITH NAME    all
Library             org.robot.database.keywords.DatabaseLibrary    WITH
NAME    db
Library           OperatingSystem    WITH NAME    OS

*** Variables ***
${validUser}      0
${invalidUser}    kekkonen
${validPw}        0000
${invalidPw}      4567

*** Test Cases ***
Wrong Password
    [Documentation]     Tablet logs in with right user name but wrong
password
    Start Tablet Connection    ${validUser}
    ${response}=    Login    ${validUser}    ${invalidPw}
    Response Type Should Be    ERROR    ${response}
    Table Must Contain Number Of Rows    hallsession    0
```

Koodiesimerkki 5 Robot-testi tekstimuodossa

Testin alussa on testikokoelman kuvaus, testikokoelman alussa suoritettavat avainsanat (tietokannan tyhjentäminen, paikallisen palvelimen käynnistäminen, testidatan tuominen tietokantaan ja tietokantaan yhdistäminen), testikokoelman lopetustoimet (tietokantayhteyden sulkeminen) ja kirjastojen tuominen. Tämän jälkeen on määritelty testeissä käytettävissä olevia muuttujia. Lopuksi ovat varsinaiset testit. Testit sisältävät dokumentaation ja käytettävät avainsanat ja niiden parametrit.

Vaikka Robotiin löytyy jonkin verran valmista dokumentaatiota, sitä löytyy selvästi vähemmän Javalle tehtyihin sovelluksiin kuin esimerkiksi Python-sovelluksiin tai nettisivustojen testaamiseen tarkoitettuun Selenium-kirjastoon. Javan käyttö myös vaatii joitain ylimääräisiä toimenpiteitä: itse tehdyt kirjastot esimerkiksi pitää lisätä virtuaaliko-

neen classpathiin joko testien käynnistysvaiheessa erikseen komentoriviltä ajettaessa tai RIDE:a käytettäessä ohjelman käynnistysparametrina.

RIDE:n käyttö toikin vielä yhden muuttujan lisää. Jotta RIDE tunnistaa ja täydentää avainsanat ulkoisista kirjastoista, täytyy kirjastojen Java-luokista kääntää erillinen avainsanakirjasto, joka tuodaan RIDE:en. Ei ollut esimerkiksi selvää, että vaikka RIDE ei tunnista ja täydennä avainsanoja, ne on silti voitu tuoda ohjelmaan oikein ja testit voidaan ajaa onnistuneesti. Vastaavasti RIDE voi näyttää tunnistavan avainsanat, mutta testejä ei voi ajaa, koska varsinaiset kirjastot puuttuvat. Avainsanatiedostot piti myös olla nimetty oikean käytännön mukaisesti siten, että tiedoston nimessä oli sekä varsinaisen luokan nimi, mutta myös pakkaus, jossa se oli.

Ulkoisista kirjastoista löytyi kaksi käyttökelpoista kirjastoa projektin käyttötarkoituksiin. Javalle tehty tietokantakirjasto tarjoaa valmiita metodeja tietokannan käsittelyyn. Kirjastolla pystyy esimerkiksi avaamaan testitapauksen alussa yhteyden tietokantaan, tarkastamaan testin lopussa tietokannan sisältämät tietueet ja sulkemaan yhteyden. Hyödylliseltä vaikutti myös JavalibCore-kirjasto, jonka avulla luokat pystyy annotaatioilla merkitsemään Robotin käytettäväksi ja nimeämään avainsanoja. Kirjastolla voi paketoida avainsanoja yhteen, jolloin kaikkia ei tarvitse tuoda erikseen RIDE:en.

Ensimmäiset testit tehtiin tabletin sisäänkirjautumiselle. Testitapauksissa koetettiin kirjautua palvelimelle käyttäen erilaisia oikeiden ja väärien käyttäjänimen ja salasanan yhdistelmiä. Lisää testitapauksia tehtiin tablettien bingolappujen ostolle. Testeissä esimerkiksi ostettiin lappuja bingon pääpeliin ja testin jälkeen tarkastettiin tietokannasta, onko oikeat ruudukot myyty.

Testien suunnittelu Robotilla osoittautui kuitenkin jonkin verran työlääksi. Järjestelmässä on monta erilaista komponenttia, kuten paikallinen palvelin, ECM-järjestelmä, POS-myyntiohjelmisto ja tabletti. Tarvittavien komponenttien automatisointi ja toiminnan muuntaminen avainsanoiksi tuntui melko aikaa vievältä. Jotkut toimenpiteet, kuten sisäänkirjautuminen, ovat yksinkertaisia eivätkä jäljitelmätabletin tekemisen jälkeen vaatineet kovin paljon lisätyötä, kun Robotin toimintalogiikka oli selvinnyt. Toisaalta taas esimerkiksi bingon pääpelin pelaaminen vaatii bingokirjojen myymistä asiakkaille etukäteen myyntiohjelmistosta, jota ei saanut suoraan tehtyä, vaan ohjelmiston kirjautuminen ja myynti olisi pitänyt eristää ohjelmasta.

Järjestelmän laajuuden, useiden toimintavariaatioiden (eri bingopelityypeillä on erilaisia ruudukoita käytössä jne.) ja kirjastojen laatimisen takia mietittiin Robotin käyttöönottoa ja tultiin siihen tulokseen, että sen kiinnittäminen projektiin tässä vaiheessa ei ehkä toisi merkittävää hyötyä verrattuna kuntoon saamisen vaivaan. Robot onkin omimmillaan ehkäpä esimerkiksi kompaktimpien projektien tai nettisivujen testaamisessa.

8.2 JavaScript-testaus

Kun suunniteltiin testausjärjestelmän seuraavaa suuntaa, työn ohjaajalta tuli ajatus JavaScript-kielellä suoritettavasta testauksesta. Testauksen voisi suorittaa Jasmine-nimisellä JavaScriptillä toteutetulla testauskehyksellä. Ajatuksena oli myös, että testejä varten tehtyä JavaScript-versiota tabletista voisi käyttää myös muussa testaamisessa, kuten nettipalvelun rasitustestauksessa, sekä mahdollisesti myöhemmin muihinkin tarkoituksiin, kuten korvaamaan nettisivulla käytettävää flash-asiakasohjelmaa.

8.2.1 Jasmine-testauskehys

Jasmine on testauskehys, joka on ohjelmoitu kokonaan JavaScriptillä. Jasmine ei ole riippuvainen selaimista eikä muista JavaScript-kehyksistä, joten sillä voi testata nettisivuja, Node.js:llä tehtyjä projekteja ja muita JavaScript-sovelluksia [Jasmine, 2013].

Testien tekeminen Jasminella on yksinkertaista. Testikokoelma määritellään funktiolla ”describe”, joka saa parametreikseen testikokoelman nimen, sekä funktion, joka sisältää testit. Testit määritellään testikokoelman sisällä puolestaan funktiolla ”it”, joka saa myös kaksi parametria; testin nimen ja funktion, joka sisältää testin logiikan. Testien lopputuloksen todentamiseen käytetään expect-funktiota, joka vastaa suurin piirtein JUnitin assert-metodia. Funktion avulla voi verrata esimerkiksi muuttujaa toiseen arvoon, tarkastaa onko muuttuja määrittelemätön tai tarkastaa muuttujan totuusarvon.

Jasminella pystyy myös testaamaan JavaScriptissä tavallisia asynkronisia funktioita. Testissä määritellään runs-funktio, joka loppuu asynkroniseen kutsuun. Tätä seuraa waitsFor-funktio, joka pysäyttää suorituksen, kunnes asynkroniseen kutsuun on vastattu tai kunnes aikakatkaisu tulee. Seuraavassa koodiesimerkissä 6 on sisäänkirjautumista testaava testi.

```
var clientFile = require("../Client");
var nodeComms = require("../NodeComms");

describe("A testing suite for tablet", function() {

            var connectionTimeout = 5000;
            var client;
            var comms;

            beforeEach(function() {
                        comms = nodeComms.NodeComms();
                        client = clientFile.Client(comms);
            });

            /*
             * Test login with wrong credentials
             */
            it("Login should fail", function() {
                        runs(function() {
                                    isLoginFinished = false;

                                    client.login("localhost",        5252,
"username", "wronpassword", function() {
                                                console.log("Logging     in
succeeded");
                                                isLoginFinished = true;
                                    }, function(reason) {
                                                console.log("Logging     in
failed. Reason: " + reason);
                                                isLoginFinished = true;
                                    });
                                    waitsFor(function() {
                                                return isLoginFinished;
                                    },  "Failing  miserably",  connection-
Timeout);

                                    runs(function() {
                                                ex-
pect(client.self.connection).toBeNull();
                                                ex-
pect(client.self.sessionId).toBeNull();
                                    });
                        });
            });
});
```

Koodiesimerkki 6. Jasmine-testi, jossa kirjaudutaan väärällä salasanalla

Testissä esitellään ensin tabletti ja sen tietoliikenneyhteyksiä hoitava comms-olio sekä aikakatkaisun aiheuttava aika eli 5000 ms. Ennen jokaista testiä luodaan beforeEach-funktiossa uusi tabletti sekä comms-olio. Testissä runs-funktiossa suoritetaan asynkroninen sisäänkirjautuminen. Funktiolle annetaan parametreina vastauksen tullessa suoritettavat funktiot. Ensimmäinen funktio suoritetaan, jos kirjautuminen onnistui ja toinen, jos se epäonnistui. Molemmissa funktioissa muutetaan isLoginFinished-muuttujan tila todeksi, mutta printataan eri viesti. WaitsFor-funktio odottaa korkeintaan viisi sekuntia,

ja tänä aikana se tarkistaa jatkuvasti isLoginFinished-muuttujan tilaa. Testin suoritus etenee, jos sen tila on tosi eli kirjautumiseen on tullut vastaus. Muutoin testi epäonnistuu aikakatkaisun takia. Tämän jälkeen tarkastetaan tabletin yhteyden ja session tunnisteen tila. Molempien pitäisi olla määrittelemättömiä, koska kirjautumisen pitäisi olla epäonnistunut.

Jasmine vaikuttaa olevan toiminnallisuudessaan lähellä JUnitia. Testien rakentaminen on yksinkertaista, ja Jasmine sisältää paljon käyttökelpoisia ominaisuuksia, kuten komponenttien jäljittelyyn käytettäviä spy-tynkiä. Jasminessa on myös ominaisuuksia, jotka tekevät siitä juuri JavaScriptin testaamiseen sopivan, kuten tuen asynkronisille funktioille. Projektissa ei tehty paljon Jasmine-testejä, mutta huomattiin, että tarvittaessa JavaScriptin testaamiseen soveltuvaa kehystä, se on varteenotettava vaihtoehto.

8.2.2 Rasitustestaus JavaScriptillä

Testausta varten oli Javalla tehdystä jäljitelmätabletista tehty melko suora käännös JavaScriptille. Tätäkin tablettia pystyi käyttämään palvelun kuormittamiseen, mutta toisin kuin Javalla tehty tabletti, tämä versio ei ottanut yhteyttä suoraan palvelimeen vaan sen sijaan sen edustalla olevaan nettipalveluun. Tätä palvelua käyttävät myös netin kautta pelaavat pelaajat, tosin viestit lähetetään flash-asiakasohjelmasta. Tämän palvelun yhteysmanagerille laadittiin yksikkötestejä aiemmassa luvussa.

Testissä asiakasohjelmat suorittivat samat toimenpiteet kuin aiemmin kerrotussa Java-versiossa: istunnolle varattiin tunniste erikseen testiä varten tehdystä palvelusta, istuntoon kirjauduttiin, pelintarjoajaan liityttiin, myytävänä olevat tuotteet listattiin ja kaikki tarjolla olevat ruudukot ostettiin. Suurella määrällä asiakasohjelmia järjestelmässä huomattiin jonkin verran viestien käsittelyn hidastumista, suurimmaksi osaksi ostoviesteissä, koska niitä oli eniten. Palvelin käsitteli kyllä viestit nopeasti, mutta hidastuminen tuntui johtuvan yhteysmanagerin ja palvelimen välissä olevasta yhteydestä samanaikaisissa viesteissä. Tämä hidastuminen ei kuitenkaan tuntunut olevan merkittävää. Palvelimen päässä tehtiin kuitenkin parannuksia muuttamalla viestien käsittelytapaa lisäämällä pyyntöjen käsittelyyn säikeitä ja vähentämällä synkronoituja lohkoja.

9 Yhteenveto

Opinnäytetyön alkuperäisenä tavoitteena oli rakentaa BingoBoosterille koko projektin laajuinen testausjärjestelmä, jonka avulla olisi mahdollista testata järjestelmää automaattisesti säännöllisin väliajoin ja löytää nopeasti regressiot sekä uudet bugit. Lähtötilanteessa järjestelmällistä testausta ei juuri ollut, ja jo olemassa ollut testaus oli manuaalista ja epätäydellistä. Järjestelmälle oli siis tilaus.

Työtä aloitettaessa ensimmäinen ongelma oli, mitä testausjärjestelmä käytännössä tarkoittaa eli mitä testaustasoja siihen kuuluu, minkälaisia testejä tehdään ja miten ne voi automatisoida. Etäinen mielikuva ideaalista testausjärjestelmästä oli, että järjestelmän yksittäiset osat ja koko järjestelmän saa testattua säännöllisesti automaattisesti, mutta mistä tällaista järjestelmää voi alkaa rakentamaan?

Projekti aloitettiin tekemällä ensimmäisenä ohjaajan ehdotuksen perusteella jäljitelmätabletti, jonka avulla pystyttiin testaamaan järjestelmän käyttäytymistä ja kestävyyttä suurella määrällä samanaikaisia käyttäjiä peliin liityttäessä ja sen aikana. Yksikkötestien tekeminen yhteyksistä vastaavalle managerille puolestaan testaa koodin toimivuutta ja paljasti myös bugeja, jotka olivat jääneet huomaamatta. Projektin aikana tutkittiin myös testauskehyksiä ja niiden soveltuvuutta testausjärjestelmään. Samalla saatiin tehtyä myös JavaScript-versio asiakasohjelmasta, jolla pystyttiin testaamaan järjestelmän toimintaa myös liityttäessä peliin nettipalvelun kautta.

Projektia aikana tuli mieleen, että valmiin projektin testauksen aloittaminen ei ole aivan yksinkertaista ja tätä työtehtävää voisi olla mahdollista helpottaa esimerkiksi käyttämällä myös monessa lähteessä vastaan tullutta testivetoista kehitystä, etenkin yksikkötestien laatimiseksi. Aikaa projektissa meni myös tutkittaessa testaustapoja, jotka eivät kuitenkaan tulleet käyttöön. Esimerkiksi projektin alussa alettiin tehdä ECM-järjestelmän jäljitelmästä helpommin parametrisoitavaa versiota. Tälle ei kuitenkaan tullut käyttöä myöhemmin. Samaten Robot-ohjelmistokehykseen perehtymiseen ja hyödyllisyyden tutkimiseen meni aikaa, ja lopputuloksena päädyttiin siihen, että vaikka Robot onkin sinänsä käyttökelpoinen työkalu, sen soveltaminen tähän projektiin oli vaikeaa.

Loppuyhteenvetona projektin tuloksena ei ollut aivan sellainen kokonaisvaltainen testausjärjestelmä kuin mielikuva aloitettaessa oli. Projektista oli kuitenkin hyötyä, sillä sitä

varten tehdyillä testeillä saatiin tietoa järjestelmän kestokyvystä eri ympäristöissä, sillä rasitustesteissä käytettyjä ohjelmia pystyttiin pienillä muutoksilla käyttämään järjestelmän testaamiseen myös eri rajapinnan kautta. Rasitus- ja yksikkötesteissä ja niiden rakentamisen aikana saatiin myös selville lukuisia yksittäisiä ongelmia, jotka saatiin korjattua saman tien. Testijärjestelmää rakentaessa suoritettiin siis myös koodin arviointia.

Lähteet

Apache Mina. Verkkodokumentti. <http://mina.apache.org/mina-project/index.html>. Luettu 20.10.2013.

Apache Mina Quick Start Guide. Verkkodokumentti. <http://mina.apache.org/mina-project/quick-start-guide.html>. Luettu 20.10.2013.

Myers, Glenford J., 2012. The Art of Software Testing. New Jersey: Word Association.

Bingoon.fi. Verkkodokumentti. <https://www.bingoon.fi>. Luettu 17.11.2013.

Jasmine. Verkkodokumentti. <https://github.com/pivotal/jasmine>. Luettu 9.12.2013.

Niemelä, Jouni. 2006. Ohjelmistojen konfiguraationhallinta. Verkkodokumentti. <http://www.mit.jyu.fi/opetus/kurssit/jot/2006/luennot/SCM_luento.pdf>. 12.12.2006. Luettu 11.8.2013.

Pan, Jiantao. 1999. Software Testing. Verkkodokumentti. https://www.ece.cmu.edu/~koopman/des_s99/sw_testing/. 1999. Luettu 11.8.2013.

Protocol Buffer Developer Guide. 2.4.2012. Verkkodokumentti. <https://developers.google.com/protocol-buffers/docs/overview>. Luettu 20.10.2013.

Protocol Buffer Developer Guide Java Tutorial. 2.4.2012. Verkkodokumentti <https://developers.google.com/protocol-buffers/docs/javatutorial>. Luettu 20.10.2013.

Robot Framework. Verkkodokumentti. < http://robotframework.org/>. Luettu 8.12.2013.

Software Reliability: A Federal Highway Administration Preliminary Handbook. Verkkodokumentti. <http://www.fhwa.dot.gov/publications/research/safety/04080/02.cfm>. 2004. Luettu 11.8.2013.

Software Testing Fundamentals. Verkkodokumentti. <http://softwaretestingfundamentals.com/system-testing/> Luettu 10.11.2013.

Unit Testing Your Application with JUnit. Helmikuu 2010. Verkkodokumentti. <http://www.oracle.com/technetwork/articles/adf/part5-083468.html>. Luettu 10.11.2013.

Unit Testing with JUnit. 28.10.2013. Verkkodokumentti. <http://www.vogella.com/articles/JUnit/article.html>. Luettu 10.11.2013.