Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

JENNY KIISKINEN

Tulostaulujärjestelmän seurannan käyttöönotto

TIETOJENKÄSITTELYN TUTKINTO-OHJELMA
2023

TIIVISTELMÄ

Kiiskinen, Jenny: Tulostaulujärjestelmän seurannan käyttöönotto
Opinnäytetyö, AMK
Tietojenkäsittely
Joulukuu 2023
Sivumäärä: 49

Työn tarkoituksena oli suunnitella ja toteuttaa seurantaa tulostaulujärjestelmään testiympäristössä. Seurantaa toteutettiinn toimeksiantajalle Jidoka Technologies Oy:lle. Seurannan keskeisempään käsitteeseen kuuluu auditointiviesti (Audit Message), erityisesti käyttäjälähtöinen auditointiviesti, joka on tarkempi käsite lokille. Käyttäjälähtöinen auditointiviesti tallentaa ja dokumentoi käyttäjän tekemiä painiketoimintoja ohjainkäytössä. Auditointiviestiä on valmiiksi toteutettu tulostaulujärjestelmään tuotantoympäristössä, minkä vuoksi hyödynnettiin valmiin auditointiviestin komponentteja.

Aluksi määriteltiin seurattavia alueita viestikohdille. Seurannan käyttöönoton varhaisessa vaiheessa valittiin seurattavia painikkeita, minkä rinnalla luotiin seurantakriteeri. Valittuja painikkeita tarkasteltiin toiminnallisuuksien mukaan, joita ryhmiteltiin pelilajikohtaisen, jaetun tapahtuman ja yleisen mukaan. Näille merkittiin potentiaaliset viestikohdat, joiden tilalle korvattiin seurannan viestin lähetyksellä.

Lopputuloksena saatiin valmis seuranta, jonka auditointiviestit päätyivät onnistuneesti portaalin selaimeen. Muutokset julkaistiin toimeksiantajan kehittäjien käytettäväksi. Jatkossa seuranta on käytettävissä, ja sitä voi siirtää tuotantoon tai tarvittaessa parannella omien tarpeiden mukaisesti.

Avainsanat: auditointiviesti, loki, android studio, java, olio-ohjelmointi, tulostaulu, ohjain, painike, käyttäjän syöte, xml

Abstract

Kiiskinen, Jenny: Commissioning of Monitoring in Digital Scoreboard
Bachelor's thesis
Business Information Systems
December 2023
Number of pages: 49

The main purpose of this thesis was to design and implement monitoring for a digital scoreboard system's test environment. The solution was implemented for Jidoka Technologies Oy. The core concept of this commissioning is an audit message, specifically a user-centric audit message, which provides a more detailed concept for the log. This type of audit message captures and documents user's operations via buttons on the controller. In general, the audit message was already implemented in the digital scoreboard system's production environment, so the finished components of the audit message were utilized for this purpose.

At first, the areas to be monitored for message points were defined. In the early stage of the commissioning of monitoring, the buttons to be monitored were selected, and alongside the monitoring criteria, they were created. The selected buttons were reviewed based on their functionalities, which were grouped into sport-specific, shared event and general categories. The potential message points within these functionalities were identified and marked, intending to replace them with the transmission of monitoring messages.

The result is a completed monitoring, and its audit messages were successfully directed to the portal's browser. The changes were pushed to the origin repository for pulling. In the future, the monitoring will be available, and it can be deployed to production or improved as needed based on individual requirements.

Keywords: audit message, log, android studio, java, object-oriented programming, digital scoreboard, controller, button, user input, xml

SISÄLLYS

TERMILUETTELO

AMQP	Advanced Message Queuing Protocol. Protokolla, jota käytetään eri sovellusten väliseen viestin välittämiseen
Debuggaus	Ohjelman virheenjäljitys englannin kielen sanasta *debug.* Debuggauksella etsitään ja korjataan ohjelmakoodin virheitä (bugeja), että ohjelma toimii
Java	Ohjelmointikieli
Käyttöliittymä	Laitteen tai ohjelmiston näkyvä osa, joka on vuorovaikutuksessa käyttäjän kanssa
Palvelin	Tietokone, joka verkossa käsittelee ja toimittaa tietoja toiselle tietokoneelle
Paradigma	Ajattelumalli
Pseudokoodi	Tarkoitettu kuvaamaan ohjelmakoodia luettavassa muodossa ilman varsinaista koodin kirjoittamista
RabbitMQ	Avoimen lähdekoodin viestinvälitysjärjestelmä, joka mahdollistaa sovellusten välistä kommunikointia välittäjän kautta

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena on alustavasti kartoittaa tulostaulujärjestelmän ohjaimen painikkeita ja niiden toimintaa, joita voidaan hyödyntää seurannan jatkototeutuksessa. Tämä käyttäjään keskittyvä seuranta on aloitettu, mistä voi kätevästi jatkaa. Seurannan tietoja kuvataan audit messagella, jota kutsutaan jatkossa auditointiviestiksi. Vastaavanlainen auditointiviestillä varustettu toteutus on tehty tulostaulujärjestelmässä, mikä on keskittynyt tulostaulujärjestelmän prosessiin ja laitteiden väliseen kommunikointiin. Näitä valmiita työkaluja tulen ottamaan käyttöön seurannan toteutuksessa. Tässä opinnäytetyössä seurannan käyttöönottoa toteutan ensisijaisesti ohjaimeen ja toissasijaisesti tulostauluun ohjaimen käyttäjälähtöisyyden vuoksi. Työtä toteutetaan toimeksiantajan Jidoka Technologies Oy:n tulostaulujärjestelmän testiympäristössä, minkä tarkoituksena on vältellä tuotannossa olevien tulostaulujärjestelmän versioita. Testiympäristö koostuu fyysisistä laitteista ja palveluista, jotka simuloivat tulostaulujärjestelmää tuotannossa. Testiympäristöä ajetaan Android Studiolla ja muutoksia tehdään Javan kielellä. Seurannan käyttöönoton toteutus tapahtuu jääkiekko-pelilajissa, kun mennään yksittäisen pelilajin näkymään. Näin tulen testaamaan seurannan toimivuutta yhdessä pelilajissa, minkä pohjalta voi liittää muihin pelilajeihin.

Työn tavoitteena on tuottaa toimiva tulostaulujärjestelmän seuranta, joka mahdollistaa painikkeista saatavaa auditointiviestien keräämistä ja arkistointia yhteen paikkaan eli portaaliin. Portaali on Jidoka Technologies Oy:n käyttämä verkkopalvelu, josta saadaan reaaliaikainen kuva käyttäjän toiminnasta järjestelmän ollessa käynnissä. Auditointiviestin sisältöä pyrin kuvaamaan mahdollisimman lyhyesti tapahtumien luettavuuden helpottamiseksi. Tulostaulujärjestelmä on valmis projekti, joten toteutan seurannan käyttöönottoa valmiin ohjelmakoodin perusteella. Tällöin en tule tekemään muutoksia, mikä vaikuttaisi tulostaulujärjestelmän toimintaan. Lopputuloksena saadaan seurannan

auditointiviesti tulostaulujärjestelmän laitteiden kautta portaaliin. Valmiin työn tuloksena saadaan pohjatietoa käyttäjän tekemistä operaatioista ja mahdollisuutta parannella järjestelmää. Jatkossa tämän seurannan käyttöönottoa voi käyttää sellaisenaan tai käyttää pohjana mahdolliselle jatkokehitykselle.

2 LÄHTÖKOHDAT

2.1 Tulostaulujärjestelmän laitteisto

Tässä luvussa esitellään yleisesti tulostaulujärjestelmän laitteistosta. Asiakkaalle menevät tulostaulujärjestelmän laitteistot koostuvat tulostaulusta, ohjaimesta ja hubista, jonka kautta viestit kulkevat MQTT-protokollaa käyttäen. Digitaalinen tulostaulu on suuri näyttö, joka on suunnattu katsomoita kohti. Se on yksi digitaalisen näytön ratkaisuista, jonka keskiössä on mainonta. Tulostaulun tehtävänä on mainonnan lisäksi näyttää reaaliaikaisesti ottelun tuloksia (erät ja pisteet) ja tilanteita (rangaistukset ja jatkoaika). Digitaalinen tulostaulu tiivistää katsojille ottelun tärkeimpiä tietoja yhteen näyttöön. Digitaalisen tulostaulun taustajärjestelmästä saadaan useita tulostauluja, joiden näkymä muuttuu automaattisesti näyttöjen kokojen mukaan (Digitaalinen Helsinki, n.d.). Tulostaulu on ohjattava, ja se tarvitsee ohjainta toimiakseen. Tulostauluja ollessa monta, tarvitaan yksi ohjain, joka hallitsee kaikkia samaan aikaan.

Ohjaimella hallitaan yhtä tai useampia näyttöjä. Se voi olla tabletti tai suurempikin. Ohjaimen mukautuvan käyttöliittymän ansiosta käyttöliittymän osia (painikkeet, tulokset) voi sovittaa tasaisesti moniin eri kokoisiin näyttöihin (Digitaalinen Helsinki, n.d.). Kaikki tapahtumat ovat lähtöisin ohjaimesta, missä käyttäjä hallitsee tulostaulun näkymää, kuten pelilajivalintaa, ottelun aikaisia tuloksia ja digitaalista mediaa. Näihin toimintoihin vaaditaan kosketusnäytön virtuaalisia painikkeita, joilla hyväksytään haluttuja toimintoja. Tämä yhdistää käyttäjän vuorovaikutusta järjestelmän kanssa. Tulostaulujärjestelmän ohjaimessa on omistettu yksi painike kullekin toiminnolle tai joukko toiminnoille, mikä tekee käytöstä helppoa ja varmistaa tehokkaan toiminnallisuuden. Painike reagoi käyttäjän fyysiseen kosketukseen, minkä aikana toimintoa suoritetaan ja suoritettua muutosta välitetään tulostauluun. Samaan aikaan uusimmat tiedot päivittyvät kaikissa tulostaulun näytöissä eli synkronoivat keskenään. Keskeistä on muistaa, että jokainen painike aktivoi tietyn tapahtuman tulostaulujärjestelmässä, olipa kyseessä suora toiminto tai toiminto lisäkysymysten ja -painikkeiden kanssa. Ohjaimen painikkeen toiminnallisuudella saavutetaan tietojen

tallentamista ja esittämistä, siirtymää seuraavaan vaiheeseen otteluissa, tietyn prosessin käynnistämistä jne. Se tarjoaa käyttäjälle mahdollisuuden tehdä toimintoja tulostaulujärjestelmässä. Painikkeen painamisesta suoriutuu tietty tapahtuma tapahtumakäsittelijän (event handler) avulla. Tapahtumakäsittelijä on koodi tai metodi ja sitä määritellään vastaamaan tiettyä tapahtumaa, jota liitetään kyseiseen käyttöliittymäelementtiin eli painikkeeseen. Yhdistämällä ohjaimen painiketta ja tapahtumakäsittelijää saadaan reagoivan ja vuorovaikutteisen käyttöliittymän, jossa käyttäjän toiminta vastaa tiettyä ohjelmistologiikan toimintoa.

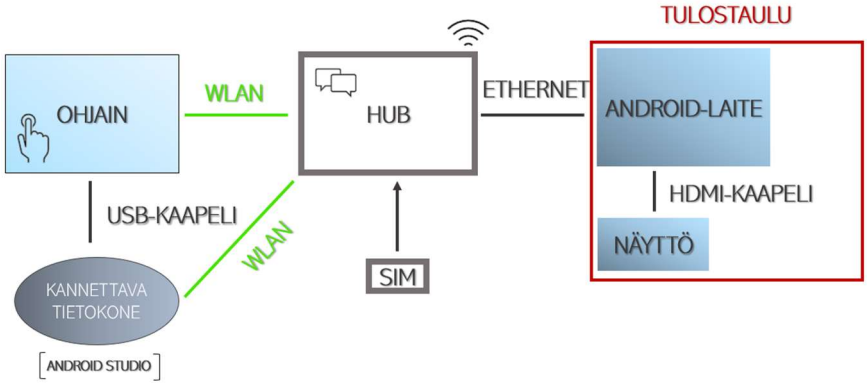
Hub eli keskitin toimii kahden eri näyttöjen signaalien vastaanottamisessa ja lähettämisessä. Hub on verkon komponentti, jolla voidaan liittää useita edellä mainittuja näyttöjä yhteen langallisesti tai langattomasti. Hubin kautta laitteita yhdistetään internettiin kytkemällä joko langallisesti WAN-porttiin tai langattomasti mobiilidatan kautta, joista tullaan kertomaan lisää. Hubilla toimii tulostaulujärjestelmän MQTT-viestivälitin (broker) (kuva 1), joka vastaanottaa aluksi paketteja ohjaimelta (publisher) ja sitten lähettää paketteja eteenpäin tulostauluun (subscriber). MQTT:n viestinkulku on kaksivaiheinen: ohjaimesta hubiin ja hubista tulostauluun. (Cheong, 2023.)

Kuva 1. MQTT publish-subscribe -malli. (Rinne, 2023b)

Ohjain julkaisee tietoa haluttuun aihepiiriin eli topiciin. Tulostaulu tilaa tiettyä topicia ja vastaanottaa siitä kaikkia julkaistuja viestejä. MQTT:n asynkronisessa viestintämallissa ohjain lähettää aluksi viestiä topiciin, jonka jälkeen hub välittää viestiä eteenpäin topicin tilaajalle, joka on tulostaulu. (Cheong, 2023; Rinne, 2023a.)

2.2 Testiympäristö

Testiympäristö koostuu eri fyysistä laitteista ja palveluista, joilla simuloidaan tulostaulujärjestelmää tuotantoympäristössä. Testiympäristöä on kehittäjien keskuudessa pystytetty tulostaulujärjestelmän kehittämistä varten, jota pidetään erillään tuotantolaitteistosta. Seurannan toimivuuden kannalta rakennetaan myös testiympäristö. Kuvassa 2 esitetään toteutetun testiympäristön kuvaus laitteista ja niiden kytkennästä.



Kuva 2. Testiympäristön fyysisen laitteiston osat.

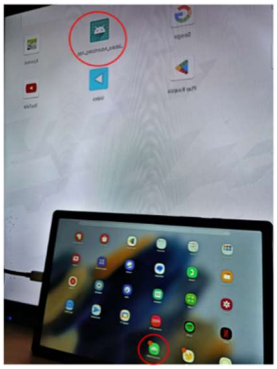
Tässä testiympäristössä ohjaimena toimii kosketusnäyttö ja tulostauluna Android-laite, jota monitoroidaan TV:n näytöllä. Hubin tehtävänä on ottaa vastaan toimeksiantajalta saadusta SIM-kortista langatonta lähiverkkoa (WLAN, tutuimmin WIFI) ja lähettää sellaisenaan eteenpäin. Ohjainta yhdistetään hubin tarjoamaan langattomaan lähiverkkoon ohjaimen asetuksesta, sillä kosketusnäytöllä ei ole ethernet-porttia. Android-laitteessa on ethernet-portti, johon kytketään ethernet-kaapelilla suoraan hubiin. Näiden lisäksi on kannettava tietokone, missä toimii Android Studio debuggamista varten.

Testiympäristön käyttöönottoon tarvitaan Jidoka Technologies Oy:ltä saatu muistitikku, joka sisältää uusimmat versiot APK- ja config.json-tiedostoja laitteiden asennukseen. APK-tiedoston (Android Package Kit) tunnistaa sen nimen perässä olevasta .apk-päätteestä (esim. controller.apk ja scoreboard.apk). APK-tiedostoja asennetaan ohjaimeen ja tulostauluun, jotta saadaan tulostaulujärjestelmän sovellukset käyttöön (kuva 3). Tiedosto config.json tarvitaan laitteiden konfigurointiin. Tämän tiedoston .json-pääte liittyy JSON-

tiedostotyyppiin, jonka avulla dataa tallennetaan tiedostoihin ja luetaan siitä. Lisäksi se sisältää seuraavia asioita:

- user, password ja virtualHost tarvitaan valtuutukseen ja yhdistämään oikeaan virtuaalipalveluun.
- hostName ja portNumber yhdistävät oikeaan palvelimeen ja käyttämään oikeaa porttia.
- serial ja clientid tarvitaan RabbitMQ:n reititykseen.

Näiden arvot talletetaan ja käsitellään Android Studion ConfigGetter-luokassa, josta AMQPConnection-luokka hakee kyseisiä arvoja laitteiden yhteyksien alustamiseen, mutta tätä enempää aihetta ei tulla käsittelemään. Luokista käsitellään tarkemmin luvulla 3.1.1.



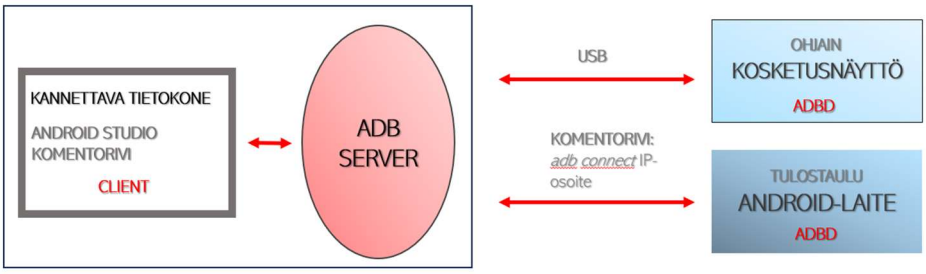
Kuva 3. Asennetut sovellukset laitteissa.

Ohjaimen konfigurointi ja tulostaulun yhteyden muodostaminen adb:hen tehdään kannettavan tietokoneen Windows-käyttöjärjestelmän komentorivin välityksellä (kuva 4). Komentorivi on työkalu, jonka suoritus perustuu syötettyihin komentoihin. Adb (Android Debug Bridge) on komentorivin apuohjelma, ja sen tunnistaa *adb* alkuisilla komennoilla:

- *adb push "config.json" sdcard/dsbController*, config.json-tiedoston kopioimiseen ohjaimeen
- *adb shell reboot*, laitteiden uudelleenkäynnistymiseen
- *adb devices*, kytkettyjen laitteiden listaamiseen

Adb:n komennot helpottavat tietokoneen ja Android-laitteiden välistä tiedonsiirtoa, kuten asentamisessa ja sovellusten kehittämisessä, testaamisessa sekä debuggaamisessa. Adb työkaluna mahdollistaa vuorovaikutusta kehitysympäristön ja virtuaalisen tai fyysisen Android-laitteen kanssa. Tässä

testiympäristössä adb:tä käytetään yhteyttä muodostettujen fyysisten laitteiden kanssa. (Avisekhar, 2016, s. 53.) Yhteyttä muodostetaan joko usb-kaapelilla tai yhdistämällä laitteen ip-osoitteeseen komentorivistä.



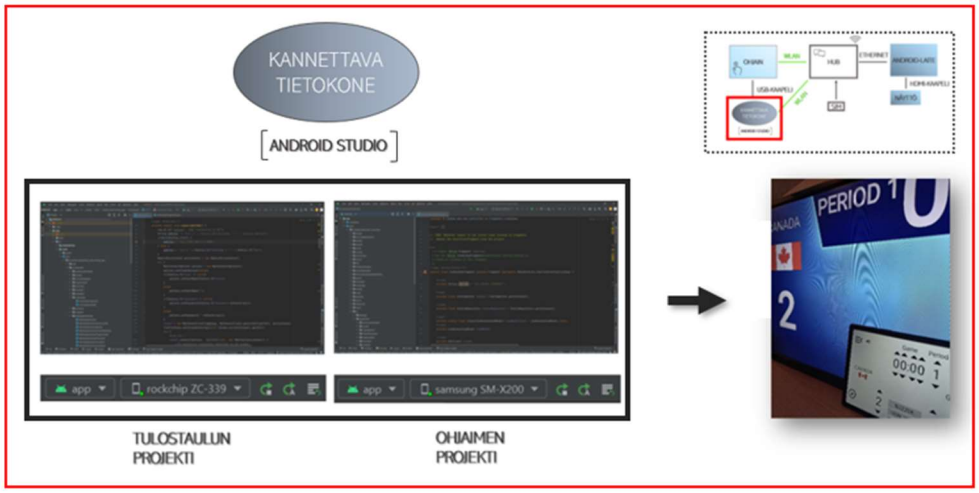
Kuva 4. Fyysiset laitteet kommunikoivat adb:n välityksellä kannettavasta tietokoneesta. (Android Developers, n.d.-b)

ADB:ssä on kolme komponenttia:

1. Client edustaa tietokonetta, joka lähettää adb-komentoja
2. Server hallitsee client:n ja adbd:n välistä viestintää ja toimii tietokoneen taustaprosessina
3. Daemon tai adbd suorittaa adb-komentoja ja toimii ohjaimen ja tulostaulun taustaprosessina (Android Developers, n.d.-a.)

Jäljelle jäävät tulostaulun konfigurointi ja ohjaimen yhteyden muodostaminen adb:hen, mikä tapahtuu ilman komentoriviä. Tulostaulun konfiguroinnissa kytketään muistitikku suoraan Android-laitteeseen, minkä aikana asennus tapahtuu automaattisesti. Ohjain muodostaa välittömästi yhteyden adb:hen, kun se on kytketty kannettavaan tietokoneeseen kuvan 2 mukaisesti.

Kannettavassa tietokoneessa on kaksi Android Studion projekteja, joista yksi on ohjaimen ja toinen on tulostaulun projekti (kuva 5). Adb:n ansiosta Android Studio tunnistaa liitettyjä laitteita omissa projekteissaan.



Kuva 5. Kannettavan tietokoneen rooli kuvana.

Kyseisiin projekteihin tullaan tekemään muutoksia, joiden toimivuutta tulisi testata debuggauksella. Android Studion tunnistaessa laitteen, voi sovellusta debugata Debug ’app’ tai Run ’app’ -ilmoituksella varustetulla painikkeella, minkä aikana sovellukset automaattisesti ottavat muutokset käytäntöön käynnistymällä uudelleen omissa laitteissaan.

2.3 Auditointiviesti

Tulostaulujärjestelmän tuotantolaitteistosta ja testiympäristöstä tulee säännöllisesti auditointiviestejä, jotka tallentuvat yrityskohtaiseen tallennuspaikkaan, kuten tietokantaan, kansioon, tiedostoon yms. Nämä viestit liittyvät järjestelmän tapahtuman tai käyttäjän toiminnon varmistamiseen, valvomiseen tai seurantaan. Auditointiviesti käsitteenä voisi liittää lokiin (log). Loki on aikajärjestyksessä oleva tallenne tai dokumentti, joka kerää säännöllisesti tapahtumia ja toimintoja järjestelmissä, tietoverkoissa yms. Lokin käytöllä katetaan tietoturvaa, järjestelmävalvontaa ja käyttäjätoimintaa. Näiden lokien säilyttämisellä saadaan helpoiten käsitystä järjestelmän tilasta, jolloin mahdollisten poikkeusten syiden selvittäminen ja ratkaiseminen on tehokkaampaa. Lokitietojen arkistoinnilla palataan myöhempiin havaintoihin esim. asioiden varmistamiseen tai virhetilanteiden sattuessa. (Traficom, 2023; Viestintävirasto, 2016, s. 2.) Loki on osana aukotonta kirjausketjua eli audit trailiä, jolla ilmaistaan muutostapahtumia katkottomasti tallennetussa paikassaan (Taloushallinto, n.d.).

Audit trail käytännössä tarkoittaa, että lukija kykenee tarkistamaan järjestelmän tapahtumia ja muutoksia asiakirjassaan tai tallenteessaan. Audit trail siis tallentaa ja ylläpitää edellä mainittuja toimintoja, missä on lokeja, mukaan lukien myös auditointiviestiä. Tämä tarjoaa kokonaisvaltaisen kuvan järjestelmän tilasta. (DNSstuff, 2020.) Loki on enemmänkin yleiskäsite erilaisille lokin luokille, joiden mm. tarkoitukset, muodot ja toimintatavat eroavat toisistaan (Traficom, 2023). Tämän vuoksi auditointiviestillä ja esim. käyttölokilla (tai tapahtumaloki) ovat erillisiä käsitteitä, mutta niissä saattaa silti esiintyä päällekkäistä tietoa.

Auditointiviesti on tarkempi käsite lokille: auditointiviesti on kuin yksi lokiin tallennettu tapahtuma. Auditointiviesti tallentaa ja dokumentoi tiivistämällä ohjelmiston prosessin ja käyttäjän toiminnan tietoja tietyssä formaatissa. Sen käyttö liittyy tietoturvaan ja tarkastukseen, joka tallentaa järjestelmän tai sovelluksen auditoinnista. Auditointiviestejä käsitellään monella eri tapaa esim. tietokannassa tai itsetehdyssä kirjauksessa, joka mahdollistaa järjestelmän tapahtumien tarkastelua ja analyysia. Auditointiviestin muotoa ja sisältöä räätälöidään kehittäjä- tai yrityskohtaisen ja niiden tarpeiden mukaan, mutta on kuitenkin tiettyjä yhteisiä komponentteja. Nämä komponentit (ohjelma 1) ovat:

- aikaleima (timestamp) on aika, milloin tapahtuma tapahtui
- suorittaja (executor) on tieto, kuka suoritti toiminnon
- toiminto (action) on kuvaus, mitä tapahtui
- lisätieto (info) on lisätieto tapahtumasta, mikä ei ole välttämättä pakollinen

(International Business Machines, 2021; NetApp, n.d.; Particular Software, n.d.; PTC, n.d.)

```
{”timestamp”: ”2023-11-20 15:30:30”, ”executor”:
”kayttaja”, ”action”: ”Kirjautuminen onnistui”, ”info”:
””}
{”timestamp”: ”2023-11-20 15:40:40”, ”executor”:
”kayttaja”, ”action”: ”Kirjautuminen epäonnistui”,
”info”: ”Yhteys katkesi 2023-11-20 15:30:30”}
```

Ohjelma 1. Esimerkki yksinkertaisesta auditointiviestistä JSON-muodossa.

Yllä oleva ohjelma on kuvitteellinen esimerkki auditointiviestistä. Todellisuudessa auditointiviesti voi olla monimutkaisempia ja rakenne voi vaihdella. Esimerkkiohjelmassa JSON-muoto on yksi mahdollinen tapa rakentaa auditointiviestiä. JSON (JavaScript Object Notation) on tiedonvaihtomuoto ja perustuu objekteista eli avain-arvo -pareihin, joita määritellään {} -aaltosulkeilla.

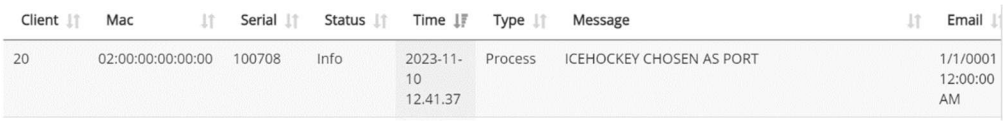
2.3.1 Käyttäjän syötteet: käyttäjälähtöinen auditointiviesti (User Input)

Käyttäjälähtöisen auditointiviestin tarkoitus on dokumentoida ja tallentaa käyttäjän tekemiä toimintoja tulostaulujärjestelmässä. Tallennuspaikkana toimii hubin ja kytkettyjen laitteiden yhteinen portaali (ks. 3.2). Käyttäjän syötteisiin keskittyvä seuranta liittyy vahvasti käyttäjään ja sen vuorovaikutukseen järjestelmän kanssa. Tässä seurannan ensimmäisessä versiossa keskitytään ohjaimen painikkeisiin, joka on kytköksissä tulostaulujärjestelmän tapahtumaan ja käyttäjän vuorovaikutukseen. Myöhemmin voi toteutetun seurannan perustalta kehittää ja laajentaa seurantaa kattavammaksi, kuten havaitsemaan muita havaintoja erilaisella näkökulmalla. Koska käyttäjä hallitsee tulostaulua vain ohjaimen välityksellä, toiminta koskee ensisijaisesti ohjaimeen. Käyttäjälähtöistä auditointiviestiä suunnitellaan muodostuvan ohjaimen painikkeen fyysisestä kosketuksesta, minkä rinnalla tulostaulujärjestelmässä tapahtuva toiminto tapahtuu. Käyttäjästä tapahtuvasta toiminnasta esimerkkejä voi olla luo, muokkaa, poista, aloita ja lopeta. Näiden tietojen perusteella saadaan hyvät lähtökohdat seurata ja ymmärtää, mitä käyttäjä tekee järjestelmässä. Käyttäjälähtöisen auditointiviestien avulla voi:

- seurata käyttäjän suorittamia toimintoja ja tunnistaa mahdollisia virhetilanteita tai sovelluksen kaatumisen syytä
- tarkastella käyttäjän tekemiä toimintoja, minkä pohjalta voisi parannella järjestelmää
- virheitä selvittäessä toistaa mahdollisimman tarkasti käyttäjän tekemiä toimia tietojen perusteella

(International Business Machines, 2021; NetApp, n.d.; Particular Software, n.d.; PTC, n.d.)

Portaali määrittää auditointiviestien esiintymismuotoa. Auditointiviestin muotoa muunnetaan valmiiksi JSON-muotoon, jonka objektit esitetään portaalin taulukkomuodossa sarakkeiden mukaan. Portaalin auditointiviestillä ei ole toistaiseksi säilytysaikaa. Portaalin valmiita sarakkeita ei voi muokata, paitsi jos on valtuudet siihen. Auditointiviestille on määritetty sarakkeet ja niiden arvot, joiden arvot järjestäytyvät portaalissa sarakkeiden alle (kuva 6). Jatkossa, kun tulee useampia auditointiviestejä, niitä kerääntyy taulukossa allekkain.



Client	Mac	Serial	Status	Time	Type	Message	Email
20	02:00:00:00:00:00	100708	Info	2023-11-10 12.41.37	Process	ICEHOCKEY CHOSEN AS PORT	1/1/0001 12:00:00 AM

Kuva 6. Käyttäjätoiminnan auditointiviesti.

Edellisessä kuvassa testattiin onnistuneesti auditointiviestin lähetystä. Portaalin sarakkeista havaitaan tuttua komponenttia eli time (aika). Client, mac ja serial saadaan Jidoka Technologies Oy:n muistitikun config-tiedostosta. Tavoitteena on jatkossa tehdä muutoksia type- ja message-kentille. Type auttaa käyttäjiä ja kehittäjiä jäsentämään ja hahmottamaan auditointiviestin sisältöä ja tarkoitusta. Suodattaessa type-kenttää voidaan tarkastella tiettyä kenttää, mikä helpottaa tiedon lukemista ja ymmärtämistä. Tällä hetkellä portaalin type-kentällä ei ole käyttäjän syötölle vaihtoehtotyyppiä. Siksi luodaan uutta tyyppiä nimeltään User Input portaalin koodien puolella. Tämän jälkeen User Input -tyyppiä voidaan käyttää auditointiviestin lähetyksen yhteydessä, mitä toteutetaan hetken päästä. Auditointiviestin message-kentässä on yksityiskohtainen kuvaus suoritetun painikkeen toiminnoista. Opinnäytetyön toteutusvaihe tulee keskittymään vahvasti tähän message-kenttään, sillä käyttäjälähtöiset tiedot tallennetaan ja esitetään message-kentällä.

2.3.2 Valmiin auditointiviestin tarkastelu

Seurannan vaiheet sisältävät useita toimenpiteitä ja prosesseja. Koska auditointiviestin valmis palvelu on tehty tulostaulujärjestelmässä, seurannan ensimmäisiä vaiheita ei ole kannattavaa tehdä. Tämä johtuu siitä, että valmiin ja toimivan auditointiviestin lisäksi kyseistä palvelua on integroitu järjestelmään,

johon liittyy käytössä oleviin tukivälineisiin, kuten RabbitMQ ja tiedon tallentaminen tietokantaan sekä esittäminen toiseen järjestelmään eli portaaliin. Uuden toimenpiteen tai prosessin kehittäminen veisi tarpeettomasti resursseja ja aikaa. Vanhasta toteutuksesta voidaan mukauttaa eli tehdä omia muutoksia ja lisäyksiä uuden auditointiviestin toteutukseen vastaamaan tarvittavia ominaisuuksia käyttäjälähtöiseksi auditointiviestiksi. Mukautetussa auditointiviestissä aluksi valikoidaan, mitä ohjaimen painikkeita tullaan keräämään, minkä jälkeen jatko kehitellään ohjelmakoodissa objektien arvoja eli mitä tietoa esitetään portaalissa. Näiden toimenpiteiden käyttöönoton lopputuotteeksi saadaan käyttäjälähtöistä auditointiviestiä tai lyhyesti seurantaa.

Tarkastellaan valmiin auditointiviestin osia. AMQPConnection-luokassa käsitellään auditointiviestin muotoa ja rakennetta ja valmiin viestin lähettämistä tallennuspaikkaan. Kaikki auditointiviestin lähetykset kulkevat AMQPConnection-luokan kautta kohti tallennuspaikkaan. AMQPConnection-luokassa käsitellään sisäisesti AMQP:n liittyviä asioita ja julkisesti auditointiviestin lähetystä. Auditointiviestin lähetyksen ollessa julkinen, sitä voi kutsua muissa luokissa. Kyseisessä luokassa on monta auditointiviestin lähetyksen metodeja, jotka määrittävät auditointiviestin tyyppiä, kuten sendMessageToDeveloper-metodi on suunnattu erityisesti käyttäjälähtöisille auditointiviesteille. Luokista ja metodeista käsitellään luvulla 3.1.1. AMQPConnection-luokan metodeissa on channel-niminen muuttuja, jonka tehtävänä on luoda viesteille reitti haluttuun kohteeseen eli portaaliin. Muuttujassa julkaistaan viesti viestijonoon basicPublish()-metodin avulla (ohjelma 2) ja sen sulkuihin määritetään parametrit, jotka määrittävät viestijonon tapahtumaa.

```
void basicPublish(String  exchange,
                  String  routingKey,
                  AMQP.BasicProperties props,
                  byte[] body)
```

Ohjelma 2. basicPublish()-metodin parametrit.

Exchange toiminta tapahtuu palvelimen puolella ja käsittelee laitteilta tulleita viestejä. Exchange toimii producerin ja viestijonon välisenä viestinviejänä. Se vastaanottaa viestejä producerilta ja vie ne viestijonoon (kuva 7). Tässä

exchange on tiedettävä, mitä tehdä saapuneen viestin kanssa, sillä viestiä voi halutessaan hävittää, lisätä tiettyyn jonoon tai moneen eri jonoon. (RabbitMQ, n.d.-b.)

Kuva 7. Viestijono. (RabbitMQ, n.d.-a)

Exchangen roolin vuoksi, sitä voisi visualisoida olevan producerin ja viestijonojen välissä. Parametrin routingKey tehtävänä on tunnistaa, mistä laitteesta clientId:n, pattern:n ja serialin tulevat, sillä näitä tietoja käsitellään palvelimen puolella, ja lopuksi esitetään portaalissa. AMQP.BasicProperties props:n paikalle määritetään auditointiviestin objektien eli portaalin taulukon kenttien arvoja. Viimeiseen parametriin byte[] bodyyn kiinnitetään viestin sisältö JSON-muodossa. Parametrien määrittelyjen jälkeen auditointiviestiä käsitellään exchangen määrittämässä palvelimessa, minkä jälkeen viestit päätyvät tietokannasta portaaliin ja portaalin selaimeen.

Painikkeen toiminnan taustalla tapahtuu samanaikaisesti useita eri tapahtumia, jotka vaativat monia eri metodia, luokkia ja muita ohjelmistollisia komponentteja. Näitä komponentteja voi tarvittaessa sisällyttää seurannan viestiin, jotta saadaan yksityiskohtaisempaa tietoa, mitä painikkeen toiminnossa tapahtuu. Painikkeen toiminnan tapahtumista voi valikoida tiettyä tapahtumaa näkökulmasta ja tavoitteesta riippuen. Näitä voi olla tietyn muuttujan arvon, funktion, luokan tai metodin etsiminen jne. Näihin kohtiin merkitään seurannan viestin lähetystä, josta alkaa seurannan viestin lähetys portaaliin. Viestin lähetystä tapahtuu Javalla kahdella (ohjelma 3) eri tavoilla: 1. luodaan erikseen olio tai 2. viitataan suoraan haluttua luokkaa. //-merkki tarkoittaa yksirivistä kommenttia ohjelmakoodissa.

```
// luodaan olio
AMQPConnection olio = AMQPConnection.getInstance();
olio.sendMessageToDeveloper("PICK ICEHOCKEY ");
```

```
// viitataan luokkaa
AMQPConnection.getInstance().sendMessageToDeveloper("PICK
ICEHOCKEY ");
```

Ohjelma 3. Auditointiviestin lähetystä voi luoda kahdella eri tavoilla.

Kun luodaan olio, se tarkoittaa uuden yksilöllisen ilmentymän tekemistä tietylle luokalle. Tässä tapauksessa kyse on AMQPConnection-luokasta. Käytetty tapa, AMQPConnection.getInstance().sendMessageToDeveloper("PICK ICE-HOCKEY ");, viittaa kyseiseen luokkaan eli AMQPConnection. Näitä kahta tapaa voidaan pitää toiminnallisesti samankaltaisina, ja niiden välillä ei ole merkittävää eroa. Molemmissa tapauksissa pyritään käsittelemään AMQPConnection-luokan instanssia tai toimintoja siihen liittyen. Käyttäjän valitessaan jääkiekon pelilajilistalta, jääkiekon pelinäkymä avautuu automaattisesti ja tämä tieto tallentuu portaaliin. Viestin lähetystä olion avulla tai suoraan luokan metodin kutsumalla riippuu tilanteesta ja siitä, mitä tavoitellaan. Jos haluaa säilyttää ja käsitellä tietoa halutussa luokassa, valitaan muuttujaan tallettaminen. Jos tieto on tilapäistä ja käsitellään yhden metodin sisällä, valitaan luokan metodin kutsuminen. Loppujen lopuksi näiden käytössä ei ole tarkkoja sääntöjä tai ”normeja”, joita olisi pakko noudattaa. Yleensä valitaan se vaihtoehto, joka tuntuu luontevalta ja sopivalta tilanteeseen.

3 KÄYTETYT TEKNIIKAT

3.1 Kehitysympäristö

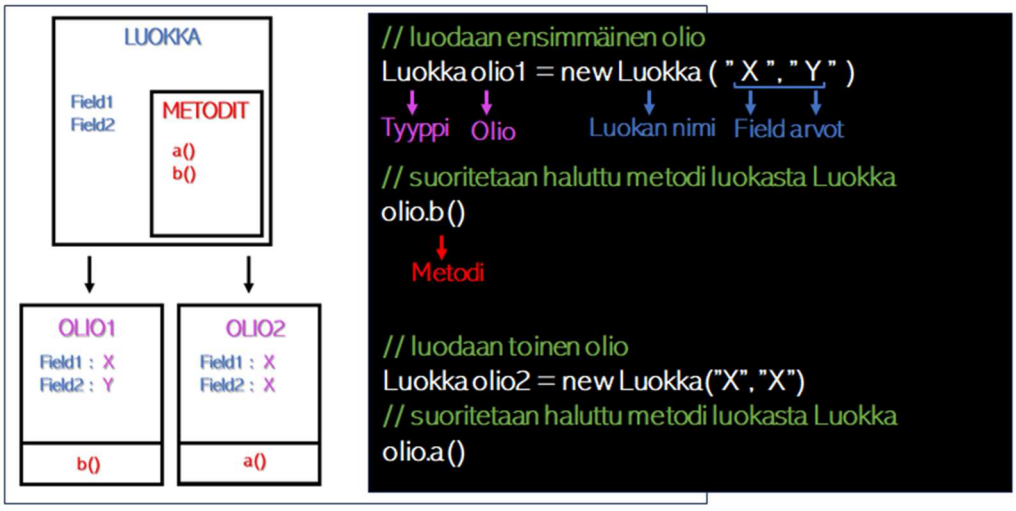
Tulostaulu- ja ohjainjärjestelmää luokitellaan Android-sovellukseksi, minkä vuoksi tulostaulun ja ohjaimen ohjelmiston kehittämisessä on käytetty Android Studiota. Android Studio on Androidin virallinen integroitu kehitysympäristö (IDE, Integrated Development Environment) Android-sovellusten kehittämiseen. Se tarjoaa monipuolisesti työkaluja ja resursseja tulostaulun ja ohjaimen suunnitteluun, kehittämiseen ja testaamiseen, kuten:

- Gradle-pohjaista työkalua Android-sovellusten koonnin hallinnointiin ja automatisointiin
- Sovellusten toiminnan jäljittäminen emulaattorilla
- XML:ää käyttöliittymäsuunnitteluun (ks. luku 3.1.1)
- Lokien tarkkailutyökalua tunnistamaan ongelmia sovelluksissa (ks. luku 3.1.2)
- Versionhallintaa koodin ja GitHubin integrointiin (ks. luku 3.1.3)

Android Studion käyttämä Gradle-työkalu mahdollistaa luomaan useita APK-tiedostoja yhdestä sovelluksesta laitekohtaisten ominaisuuksien tukemiseksi, kuten eri laitteen tai näyttökoon mukaan. Lisäksi Gradle tarjoaa tehokkaan tavan hallita projektin tarvittavia riippuvuuksia lopulliseksi sovellukseksi, esim. tarpeettomien moduulien tai kirjastojen poistaminen tai lisääminen. (Android Developers, n.d.-d.) Android Studiolla pystyy emuloimaan eli ajamaan Android-sovellusta eri laitteissa ja käyttöjärjestelmissä virtuaalisesti avd:llä (Android Virtual Device) tai fyysisesti adb:llä testiympäristön mukaisesti. AVD Managerin avulla luodaan virtuaalisia laitteita ja määritetään laitteille ominaisuuksia, kuten laitteen nimeä, näytön kokoa ja resoluutiota sekä RAM-muistia. (Avisekhar, 2016, s. 52.) Android-sovelluksen kehittämisessä on käytetty Javaa ja käyttöliittymän määrittelyssä XML:ää. Kyseiset kielet määritetään omissa tiedostoissaan, joista tunnistaa .java- ja .xml- päätteellä.

3.1.1 Android-ohjelmointi

Android-sovellusten kehityksen aikana on hyödynnetty olio-ohjelmoinnin paradigmaa, jolla hyödynnetään koodin jäsentämistä ja selkeyttämistä. Olio-ohjelmointi (OOP, Object-oriented programming) on yksi käytetyimmistä paradigmoista ohjelmistojärjestelmän rakentamisessa. Se on yksi nopeimmin kehittyvistä paradigmoista monien vuosien ajan ohjelmistokehityksen alalla. Hallitsevia olio-ohjelmoinnin kieliä ovat Java, Ruby, Python, C# ja C++, joita käytetään suurissa yrityksissä, vaikkakin kyseiset ohjelmointikielet eivät täydellisesti noudata olio-ohjelmoinnin puhtaita periaatteita. Loppujen lopuksi olio-ohjelmoinnin puhtaan periaatteen omaavaa kieltä tällä hetkellä ole. (Yakuba & Zykov, 2022.) Tulostaulujärjestelmän kehittämisessä on olio-ohjelmoinnin lisäksi käytetty Javaa. Android Studio tarjoaa valmiita luokkia ja rajapintoja sovelluksen luomiseen, kuten Activity- ja View -luokkia käyttöliittymässä. Näitä käsitellään tarkemmin opinnäytetyön toteutusvaiheessa. Olio-ohjelmoinnilla saavutetaan uudelleenkäytettävyyttä, jonka vuoksi samaa koodia ei kirjoiteta uudelleen. Koodia kirjoitetaan ja kootaan pienistä ohjelman osista, jonka toiminnallisuus on tarkasti määriteltävissä. Tällöin tulostaulujärjestelmän ohjelma rakentuu siis erotettavista pienistä yhteistoiminnassa olevista olioista. (Doherty, 2020.) Olio-ohjelmoinnin peruskäsitteistä keskitytään luokkaan, metodeihin eli toiminnallisuuksiin ja olioihin. Näiden toimintamalli näkyy seuraavassa kuvassa (kuva 8), missä lisäksi on esitetty yksi ratkaisu luoda Javalla koodillisesti olioita.



Kuva 8. Olioiden muodostaminen tapahtuu asteittain.

Luokka määrittelee olion ominaisuutta eli siihen liittyviä tietoja. Luokan sisällä määritellään ja käsitellään luokan käyttämien field:ien arvoja. Kun luokka on määritetty, olio luodaan kyseisen luokan perusteella, mikä tekee oliosta luokan ilmentymän (instance). Luokasta voidaan luoda yhtä tai useampaa olioita, joilla jokaisella on identtiset metodit, mutta eri field:ien arvoja eli uniikkisia datoja. Olio-ohjelmoinnissa nämä oliot jakavat tietoa keskenään viestien välityksellä.

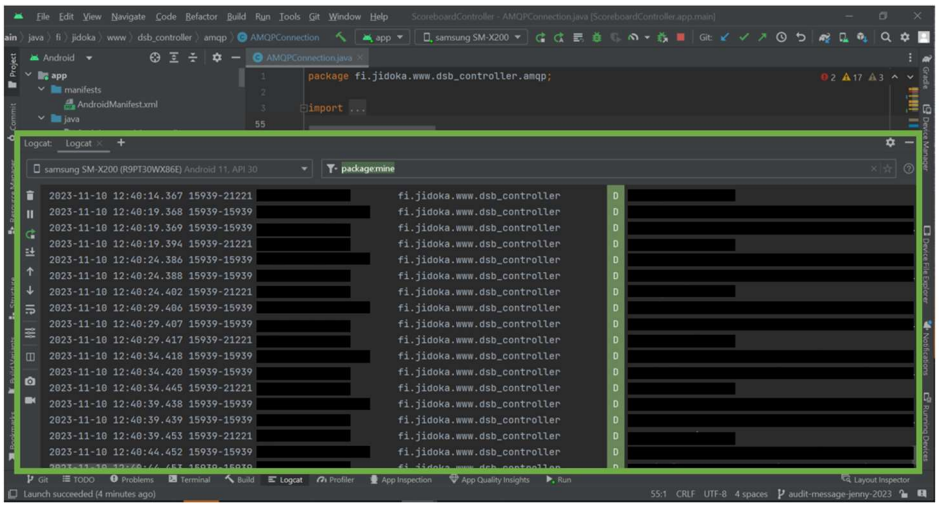
Tulostaulujärjestelmän toiminnallisuutta on hyödynnetty olio-ohjelmoinnin luokilla, metodeilla ja olioilla. Tämä toiminnallisuus on jaettu useisiin java-tiedostojen osiin, ja näitä osia on edelleen organisoitu kansioihin selkeyden lisäämiseksi. Kansiorakenne heijastelee ohjelmistokomponentteja, jotka määrittävät, mitä komponentti tavoittelee. Komponentteja tai kansioita voi olla esim. clock (kello), language (kieli), goal (maali) jne. Jokaiseen java-tiedostoon on suunnattu oma olio-ohjelmoinnin luokka, joiden nimet ovat identtiset, kuten AMQPConnection-luokka AMQPConnection.java-tiedostossa. Tulostaulujärjestelmän toiminnallisuutta ohjataan ohjaimen painikkeen toiminnallisuudella, jota määritellään erillisessä java-tiedostossa. Painike muodostuu useista luokista, metodeista ja olioista tulostaulujärjestelmän toiminnallisuudesta riippuen tarkoituksesta. Painiketta luodaan omassa luokassaan, mihin kutsutaan jopa useita toisen luokan ilmentymiä ja niiden metodeja, jotka tarjoavat keinoja muuttaa tulostaulujärjestelmän toiminnallisuuden tilaa ja tapahtumia.

Painikkeen tehtävänä on toiminnallisuuden lisäksi linkittää toimintoa käyttöliittymän elementtiin XML:n avulla painikkeen näköiseksi. Tällöin yhdellä painikkeella on sekä java- että XML-tiedostoja, jotka jakavat yhteisen id:n eli yksilöllistä tunnistetta. Java-tiedostossa painikkeelle annetaan id:tä, jotta siihen voidaan viitata ohjelmallisesti XML-tiedostosta, minkä seurauksena painike saa visuaaliset ja toiminnalliset ominaisuudet. XML-merkintäkielellä (Extensible Markup Language) siirretään ja tallennetaan tietoja. Näitä tietoja jäsennellään merkinnöillä, jotka kuvaavat tiedon rakennetta. XML-tiedosto sisältää käyttöliittymäelementtejä, joilla tallennetaan painikkeille visuaalista esitystä, kuten väriä, kokoa ja muotoa. XML:ssä attribuuteilla voi myös tuoda drawable-kansiosta (resurssi) piirrettyä grafiikkaa, esim. ikonia tai tekstiä. Sovellusten

debuggauksen aikana nämä liitetyt resurssit generoituvat, minkä seurauksena painikkeet saavat lopullisen muotonsa. Näiden ominaisuuksien tavoitteena on helpottaa käyttäjiä ymmärtämään, mitä painikkeessa tapahtuu. Esim. rangaistuksen poistoa kuvaavassa painikkeessa voi olla punainen väri ja roskakoriikoni, mikä antaa visuaalisen vihjeen toiminnon luonteesta.

3.1.2 Lokien tarkkailutyökalu

Logcat on Android-järjestelmän työkalu lokitietojen tarkkailemiseen, missä näytetään reaaliajassa tietoja laitteen toiminnasta ja sovelluksen suorituksesta. Logcat on hyödyllinen työkalu tulostaulujärjestelmän tarkasteluun ja tutkimiseen. Tämän avulla löydetään sopivia metodeja tai luokkia, jotka vastaavat parhaiten tulostaulun muuttujien ja ohjaimen painikkeiden toimintaa. Näitä lokeja korvataan myöhemmin auditointiviesteillä. Android Studion Logcat-ikkuna täyttyy erilaisilla lokiviesteillä, jotka liittyvät sovellusten suorituksen lisäksi esim. virheisiin, vianetsintään ja muihin järjestelmätapahtumiin. Sieltä pääsee näkemään itse lisättyjä lokeja Android-sovellusten ollessa käynnissä (kuva 9). Konkreettisia lokiviestejä ei tulla näyttämään seuraavassa kuvassa.



Kuva 9. Logcat-ikkunaan on kerätty tulostaulujärjestelmän monenlaisia tarpeellisia lokeja.

Android Studiolla on oma Logcat-ikkuna, jota avataan valitsemalla ”Logcat”-välilehteä alareunasta. Logcat-ikkunasta pääsee muokkaamaan lokin järjestystä tai valikoimaan tiettyjä lokeja. Suodattamalla lokiviestejä eri kategorioihin

esim. prioriteetteihin tai tageihin, näyttää ikkuna vain näitä kyseisiä lokeja. Hakutoiminnolla etsitään tiettyjä lokiviestin tiettyjä sanoja tai ilmaisuja. Luettavuuden tukemiseksi lokiviestit ovat värikoodattuja prioriteettien mukaan, kuten kuvasta x on D-kirjain (DEBUG) vihreällä laatikolla. Logcat-ikkunan viestit muodostuvat Log-luokasta, jota lisätään itse ohjelmaan kuvan 10 mukaisesti. (Android Developers, n.d.-e.)

Kuva 10. Loki ohjelmassa ja logcat-ikkunassa.

Javan puolella lokia luodaan Log-luokalla, joka tulostaa lokiin tietoa. Löytäessään painikkeen mahdollista luokkaa tai metodia, niihin voi lisätä itse Log-luokkaa. Painikkeen, ja sen taustalla olevan metodin tai luokan aktivoituessa tulostuu itse lisäämä loki Logcat-ikkunaan nähtäväksi. Logcatin loki koostuu monista eri osista, kuten tunnisteesta (tag), prioriteetista (priority) ja viestistä (message). Tunniste on lokissa lyhyt merkkijono, joka liittyy komponenttien, luokkien tai toiminnallisuuksien nimeen. Tunnisteen käyttö auttaa erottamaan lokiviestit toisistaan ja paikallistaa viestien alkuperää etenkin, kun projektissa on useita komponentteja. Prioriteetit määrittävät lokiviestin tärkeysastetta. Prioriteetteja ovat FATAL, ERROR, WARNING, INFO, DEBUG ja VERBOSE, joita ilmoitetaan yhtenä kirjaimena. Näistä tullaan käyttämään INFO ja DEBUG, sillä halutaan tarkempaa tietoa sovelluksen suorituksesta virheiden ja muiden ongelmien tarkastellessa (kuva x). (Android Developers, n.d.-e.)

3.1.3 Versionhallinta

Versionhallinta on palvelu, joka säilöö koodien eri versioita. Näitä versioita pääsee muokkaamaan ja seuraamaan ohjelman kehitystä. Versionhallinnan avulla tallennetaan ja hallitaan ohjelman lähdekoodin muutoksia ajan saatossa sekä jakamaan muokattua versiota muille saatavaksi. Tämä tarjoaa samassa projektissa työskenteleville kehittäjille mahdollisuuden ratkaista konflikteja, kun useita muutoksia tulee samaan tiedostoon.

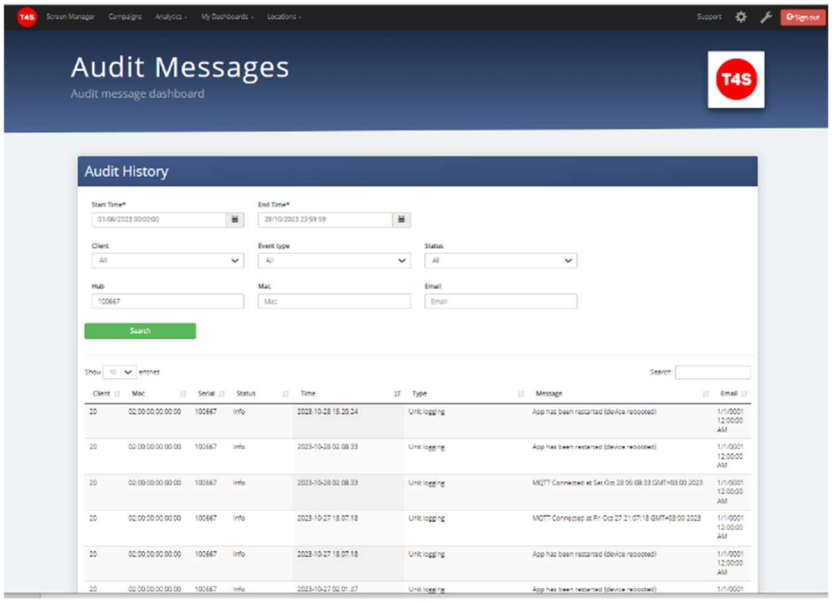
Sourcetree on Git-versionhallinnallinen työkalu sekä graafisella käyttöliittymällä (GUI) varustettu ilmainen työpöytäsovellus tarjoten versionhallintajärjestelmien hallintaan. Ohjelman toiminnot korvautuvat konsolista käyttöliittymäelementteihin. Sen toiminta on versionhallinnassa visuaalista, mutta suoraviivaista. Sourcetreen käyttö helpottaa Gitin käyttöä ja hallintaa, etenkin niille, jotka eivät ole perehtyneitä komentorivikäyttöön. Sourcetreetä käytetään seurannan auditointiviestien lisäämisen jälkeen, kun muutokset tallennetaan muille nähtäväksi. SourceTree tarjoaa miellyttävän vaihtoehdon Gitin käytölle ja samalla helpottaa merkittävästi ohjelmistojen hallintaa. (Topchyi, 2022.)

Gitin branchillä luodaan samasta ohjelmasta erillistä haaraa, joka mahdollistaa eri ominaisuuksien tai muutosten tekemistä. Branch kuvastaa puun oksaa, missä on päähaara eli ”main”, joka edustaa ohjelman pääversiota. Uuden ominaisuuden tai virheen korjatessa luodaan uusi haara päähaaran pohjalta. Näin uudempi haara on päähaarasta erillinen haara. Tällä tavalla voi muokata ohjelmaa ilman, että se vaikuttaisi ohjelman pääversiota. Branchille annetaan jokin nimi, joka kuvastaa muokkauksen tavoitetta, esim. seurannan käyttöönoton tapauksessa loin ”audit-message-jenny-2023”. Kun tarvittavat muutokset on tehty, yhdistetään luotu branch takaisin päähaaraan, minkä jälkeen ohjelma saa uudet ominaisuudet tai muutokset.

3.2 Portaalin selain

Hubin ja siihen kytkettyjen laitteiden tapahtumien tietoja pystyy lukemaan ja muokkaamaan hubin portaalista etänä. Portaalin selain tarjoaa pääsyn tietyn hubin tai useamman hubien tietoihin. Portaali on verkkopalvelu, joka vaatii selaimen avaamisen ja kirjautumisen tähän palveluun. Hubin tietoja ovat esim. sijainti, signaalin voimakkuus, käyttöoikeuksien hallinta ja pääsynhallinta sekä käyttö. Tiedot päivittyvät portaalissa automaattisesti, jonka ansiosta tapahtumia voi seurata reaaliaikaisesti. Tällä tiedonhallinnalla voidaan säädellä datan käyttöä ja kulutusta hubissa. (Track4Services, n.d.) Kun tulostaulujärjestelmä on integroitu hubiin, portaalissa visualisoidaan ohjelmallisesti luotuja

tapahtumia, jotka on määritelty Android Studiossa. Visualisoinnilla muunnetaan tapahtumia graafiseen muotoon, kuten taulukoksi, havainnollistavampaan muotoon hahmottamaan muutosten tapahtumia. Näistä tapahtumista keskitytään seurannan viesteihin (kuva 11), jotka tallentuvat portaalin Audit Messages -osion alle.



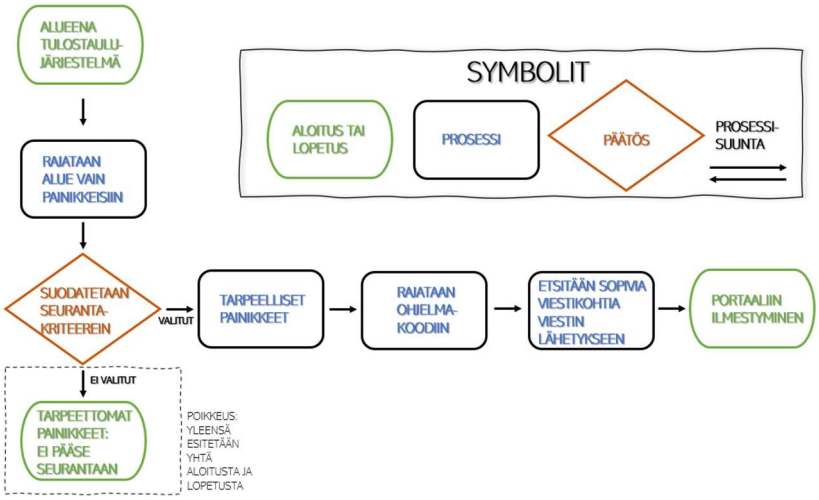
Kuva 11. Auditointiviestit portaalissa.

Seurannan viestien visualisoinnin lisäksi portaali arkistoi näitä auditointiviestejä tulevaa seurantaa varten. Auditointiviestiä kerääntyy riveinä portaaliin reaaliajassa järjestelmän ollessa käynnissä, etenkin painikkeen fyysisestä kosketuksesta. Näitä auditointiviestejä pystyy suodattamaan viestin sisällön mukaan tai vaihtamaan järjestystä nousevaan tai laskevaan muotoon. Auditointiviestin komponentit sijoittuvat portaalissa taulun sarakkeisiin seuraavassa järjestyksessä: client, mac-osoite, serial, status, time, type, message ja email, joista message-kenttään tehdään ensisijaisesti jatkotoimenpiteitä. Portaalissa voi myös käsitellä auditointiviestejä ja niiden esiintymistä, esim. suodattamalla auditointiviestin client:n, event typen, statuksen, hubin serialin, mac-osoitteen tai emailin mukaan.

4 KÄYTTÖÖNOTTO

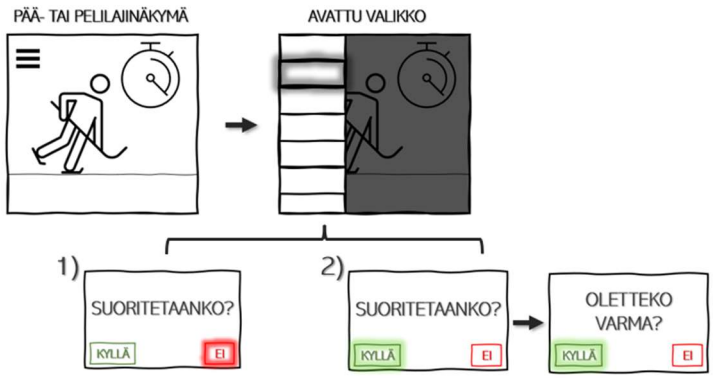
4.1 Viestikohtien määrittäminen

Seurannan käyttöönottoa testataan jääkiekossa, vaikka pyrkimyksenä on löytää ratkaisu, joka hyödyttäisi kaikkia pelilajeja. Jääkiekkoa on paranneltu jatkuvasti ja se antaa erinomaisen pohjan seurannan käyttöönotolle, sillä jos seuranta toimii jääkiekossa, toimii seuranta myös muissa pelilajeissa. Jatkossa tulevat määrittelyt ja toteutukset tapahtuvat jääkiekon toiminnallisuuden ja käyttöliittymän mukaan, mukaan lukien painikkeet ja niiden toiminnallisuudet. Seurannan käyttöönoton varhaisessa vaiheessa määritetään viestikohdat ennen viestin lähetystä. Kun määritellään seurattavaa aluetta, on tärkeää olla selvillä valituista viestikohdista. Tällä tavalla varmistetaan tässä laajassa järjestelmässä tarpeettomien sekaannusten välttämistä ja työajan tehokkaampaa hyödyntämistä, sillä jokaisen komponentin tutustumisessa ja sen ymmärtämisessä voi viedä runsaasti aikaa, ennen kuin päästään tekemään muutoksia. Viestikohtien määrittämisen prosessia kuvataan vuokaaviona (kaavio 1). Tällä kaavalla kuvataan viestikohtien määrittämiseen liittyvien prosessin toimintoja graafisesti. Näin helpotetaan ymmärtämistä ja seuraamista visualisoimalla viestikohtien määrittämisen toimintojen vaiheita ja päätöksiä standardisoiduilla symboleilla. (Aspose, n.d.)



Kaavio 1. Viestikohtien määrittelyn prosessi vuokaaviossa.

Aloitetaan prosessi painikkeiden valinnasta, eli määritellään, mitkä painikkeet halutaan sisällyttää auditointiviestiin. Näin saadaan kavennettua tutkittavaa aluetta koko järjestelmästä vain painikkeisiin. Tässä vaiheessa prosessia ohjaimen painikkeita on huomattava määrä: päänäkymässä on useita erilaisia pelilajeja, joista valita ja sivupalkkivalikko (kuva 12), josta ilmenee lisää painikkeita käyttäjän syötön perusteella dialogin mukaan. Dialog on kuin pieni ikkuna ja se ei täytä ohjaimen näkymää. Dialogin tarkoitus on pyytää käyttäjältä lupa tai varmennusta ennen järjestelmän suorittamista. Käyttäjä vastaa dialogeihin heille tarjoamilla painikkeilla (kyllä-ei, ok-poistu). Kun tiettyä pelilajia valitaan, avautuu uusi näkymä eli valitun pelilajin näkymä sisältäen lisää painikkeita ottelun hallintaan. Näihin painikkeisiin kuuluu mm. pisteiden lisääminen tai vähentäminen, PLAY ja STOP sekä sivupalkkivalikko päänäkymän mukaisesti. Pelilajien näkymä perustuu fragmentin toimintaan. Fragment on kuin katkelma ohjaimen käyttöliittymästä tai tapahtumasta, joka on vahvasti kytköksissä Activityyn. Activity on yksittäinen asia, mitä käyttäjä voi tehdä järjestelmässä. Fragment-järjestelmä määrittää itse elinkaarensa, joka on riippuvainen toiminnostaan, jos jääkiekkoa lopetetaan, silloin myös kyseistä fragmenttiä lopetetaan, jolloin palataan päänäkymään. (Android Developers, n.d.-c.)



Kuva 12. Valikko ja välikysymyksiin tarvittavia painikkeita yksinkertaisesti visualisoituna.

Sivupalkkivalikosta voi määritellä tarkemmin ottelun kulkua, ohjaimen ja tulostaulun asetuksia sekä muita tärkeitä vaihtoehtoja. Sivupalkkivalikon avaamiseen liittyy kuvake kolmesta viivasta pää- tai pelilajinäkymän vasemmassa ylälaidassa. Kun valikkoa avataan, sivupalkkivalikko ilmestyy näytön reunaan, josta näkyy pääsisältöjä. Sivupalkkivalikko sisältää käyttäjälle joukko valintoja,

joissa voi vaatia käyttäjältä välikysymysten vastaamista tai lisätoiminnan määrittelyä.

Vaikka turvallisinta on valita kaikkia mahdollisia painikkeita seurantaan, niiden joukossa voi kuitenkin olla tarpeettomia painikkeita, joiden tieto ei ole olennaista seurannassa. Tämän vuoksi tulostaulujärjestelmän ohjaimen painikemäärää kavennetaan entisestään, jotta tarpeettomat painikkeet eivät häiritse seurantaa. Nämä tarpeettomat tiedot voivat täyttää portaalin auditointiviestien tallennuspaikkaa liikaa ja ylikuormittavat nopeasti seurannan kapasiteettia. Tällöin tarpeettomat tiedot vievät keskittymisen tärkeistä tiedoista, ja ne hankaloittavat portaalin luettavuutta ja seurannan keräämän potentiaalisen tiedon tutkimista. Koska kyseiseen tilanteeseen ei ole asetettu kriteeriä tarpeettomista ja tarpeellisista painikkeista, luodaan lyhyt seurantakriteeri, mistä voi lähteä liikkeelle. Ohjaimen painikkeita on analysoitu sekä fyysisesti kosketusnäytöllä että Android Studion kautta, mukaan lukien koodillinen tarkastelu ja testit logcatilla. Näiden käsittelyistä tarkemmin luvulla 4.2. Painikkeiden käyttöjen ja tutustumisen aikana on havaittu asioita, jotka on otettu huomioon seurantakriteerissä. Jatkossa painikkeet, jotka löytyvät taulukon (taulukko 1) kriteereistä, luokitellaan tarpeettomiksi.

Taulukko 1. Seurantakriteeri

Kriteeri	Kuvaus	Lisätieto
1. Painikkeita ei pysty testaamaan	Testaattomilla painikkeilla ei voi taata niiden toimivuutta oikeassa ympäristössä	
2. Vaatii ylläpitäjän roolia	Kehittäjille tarkoitetut painikkeet eivät sovellu tavalliselle käyttäjälle	
3. Toistuvat toiminnot	Sama toiminto on saatavilla useammalla kuin yhdellä painikkeella	*Erilaisten valikkojen avaaminen & välikysymysten tai lisätoiminnan määrittelyn välivaiheet* *Poikkeuksena aikalisän määritysnäkymä, missä määritetään aikalisää*

4. Välimuotoon tarkoitetut painikkeet	Näillä painikkeilla taataan mahdollisuutta perua painikkeen aiheuttamaa toimintoa etenkin, kun käyttäjä totuttelee ohjaimen käyttöä	
5. Automaattiset vaihdot tai siirtymät	Automaattiset vaihdot tai siirtymät	*Erilaiset siirtymät otteluissa & tietojen automaattinen muuttuminen tai vaihtuminen*
6. Ovat valmiiksi luotu	Ei ole tarvetta tehdä kopioita	*Osa valmiista luoduista auditointiviestistä liittyvät vahvasti käyttäjän syöttöihin, minkä vuoksi näitä kyseisiä auditointiviestejä voi halutessaan siirtää tähän seurantaan.*
7. Painikkeen toimintaa ei voi eristää Android Studiossa	Tietyissä metodeissa tai olioissa on päällekkäisyyksiä muiden metodien tai olioiden kanssa	*Tietyn toiminnan yhdistäminen moneen eri painikkeisiin. Yhden painikkeen aktivoituminen vaikuttaa muihinkin painikkeisiin*

Tämä seurantakriteeri on alustava suunnitelma, jolla helpotetaan omaa työtä painikkeiden rajauksessa. Seurantakriteeri voi muuttua tai jopa poistua kokonaan tämän opinnäytetyön käyttöönoton jälkeen. Seurantakriteerien asettamisen tavoitteena on vain määrittää, mitkä painikkeet otetaan alustavasti mukaan seurantaan. Seurantakriteerin poikkeuksena on aikalisän määritysnäkymiä, missä käyttäjä määrittelee tarvittavat asetukset ennen toiminnan hyväksymistä. Aikalisässä esiintyy satunnaisia ohjelmavirheitä, jotka aiheuttavat ohjainjärjestelmässä odottamattomia kaatumisia, mikä saattaa hankaloittaa käyttöä, etenkin silloin, kun käyttäjä testaa ohjainta ensimmäistä kertaa. Toistaiseksi olisi suositeltavaa ilmoittaa erikseen, kun aikalisän määritysnäkymä avataan, jotta tiedetään vielä yksityiskohtaisemmin käyttäjän viimeisempää toimintaa ennen ohjainjärjestelmän kaatumista.

4.2 Toimintojen rajaus ja viestikohtien merkitseminen

Seurantakriteerien laatimisen yhteydessä on käyty ohjaimen jokaista painikkeita yksitellen läpi, jotka ovat koottu alla olevaan taulukkoon (taulukko 2). Taulukossa on huomioitu ne painikkeet, jotka näyttävät ja toimivat identtisesti muissakin pelilajeissa ja niiden arkkitehtuurillinen sijainti ohjelmakoodissa ovat samoissa kohdissa. Painikkeita esitetään yhtenä kokonaisuutena eli niitä ei jaotella valikon ja näkymien perusteella. Taulukossa sarakkeina ovat painike, toiminta, auditointiviesti ja kategoria. Painikkeessa esitetään aakkosjärjestyksessä valittuja painikkeita, joita otetaan huomioon seurannassa. Painike-sarakkeessa ilmenee ++-merkki, jolla indikoidaan painikkeen sijaitsevan useassa eri paikassa. Toiminnassa kuvataan painikkeen toimintaa tulostaulujärjestelmässä ja auditointiviestissä näytetään painikkeen auditointiviestiä message-kentässä, jossa esitetään painikkeen potentiaalista auditointiviestiä seurannassa. Painikkeiden auditointiviestin tavoitteena on selittää viestin sisältö preesensissä eli nykymuodossa mahdollisimman selkeästi ja ymmärrettävästi vastaten kysymyksiin: mikä painike on aktivoitu ja millaisia muutoksia se aiheuttaa. Auditointiviestissä hyödynnetään oman ja toisen painikkeen muuttujien arvoja, jotta saadaan kattavampi tieto järjestelmästä. Muuttujia ovat esitetty taulukossa kursivoituina pienin kirjaimin. Totuusarvomuuttujasta voi saada kahta mahdollista arvoa: tosi (true) tai epätosi (false). Tällä muuttujalla kerrotaan, onko suorittama toiminto kytketty käyttöön vai ei. Muut sanat isoilla kirjaimilla auditointiviesti-sarakkeessa säilyvät muuttumattomina. Näiden sarakkeiden lisäksi kategoria-sarake kertoo, mihin kategoriaan painikkeen valitsema toiminto sijoittuu. Tästä tarkemmin taulukon jälkeen. Painike-sarakkeen ++-merkin vuoksi kategoria määräytyy painikekohtaisesti.

Taulukko 2. Valitut painikkeet ja niiden toiminta, auditointiviesti sekä kategoria

Painike	Toiminta	Auditointiviesti	Kategoria
Aikalisä	Aloittaa aikalisän pyytävälle joukkueelle välittömästi	START TIMEOUT: *joukkue, kesto*	*lajikohtainen*
Aikalisän määritysnäkymä	Määritysnäkymässä määritetään aikalisän kestoa ja joukkuetta	TIEMOUT CONFIG VIEW tai	*lajikohtainen*

		TIMEOUT CONFIG VIEW: CANCEL	
Aloita ottelu	Aloittaa ottelun käyttäjän hyväksyttyä ottelun määritysnäkymässä	START: *ottelutiedot*	*lajikohtainen*
Kieli	Vaihtaa kieltä	CHANGE LANGUAGE: *EN/FI/SE*	*geneerinen*
Lisää rangaistus	Luo rangaistusta (joukkue, pelaaja, kesto)	ADD PENALTY: *joukkue, pelaaja* & *rangaistuksen kesto*	*lajikohtainen*
Maalivideo ++	Näyttää maalivideon, sijaitsee joko pelilaji- tai mediahallintanäkymässä	PLAY GOAL VIDEO	*geneerinen & lajikohtainen*
Mediahallintanäkymässä olevat mediat	Näyttää, soittaa tai pysäyttää median (video, musiikki)	MEDIA CONTROLLER: START *median tyyppi, nimi, kesto ja sijainti* tai MEDIA CONTROLLER: STOP VIDEO/ GOAL VIDEO/BUZZER	*geneerinen*
Muokkaa rangaistus	Muokkaa voimassa oleva rangaistusta	EDIT PENALTY: TEAM/PLAYER NUMBER/DURATION *alkuperäinen arvo* CHANGED TO *uusi arvo*	*lajikohtainen*
Ottelun määritysnäkymä	Määrittää ottelun tietoja (mm. erämäärän määrittäminen ja erä-, erätauko-, jatkoaika- ja lämmittelykello vaativat tarkkoja asetuksia, mukaan lukien summeriäänet)	MATCH CHANGES: joko *tieto* FROM *alkuperäinen arvo* TO *uusi arvo* tai *tieto & totuusarvo*	*lajikohtainen*
Piste (koti)	Nostaa tai vähentää pistettä kotijoukkueelle	HOME SCORE: *kotiPiste-vierasPiste*, CLOCKTYPE: *kellotyyppi*	*lajikohtainen*

Piste (vieras)	Nostaa tai vähentää pistettä vierasjoukkueelle	GUEST SCORE: *vierasPiste-kotiPiste*, CLOCKTYPE: *kellotyyppi*	*lajikohtainen*
Poista rangaistus	Poistaa voimassa olevaa rangaistusta (joukkue, pelaaja, kesto)	REMOVE PENALTY: *joukkue, pelaaja, rangaistuksen kesto*	*lajikohtainen*
Pysäytä media ++	Pysäyttää kesken jääneen median, sijaitsee joko valikossa tai mediahallintanäkymässä	STOP MEDIA	*geneerinen & lajikohtainen*
Summeri ++	Soittaa summerin, sijaitsee joko pelilaji- tai mediahallintanäkymässä	PLAY BUZZER	*geneerinen & lajikohtainen*
Valitse pelilaji	Valitse mikä tahansa pelilajin listatuista pelilajista päänäkymässä	PICK ICEHOCKEY	*lajikohtainen*
Välitauko	Aloittaa erätauon välittömästi	START INTERMISSION: *kesto*	*geneerinen*

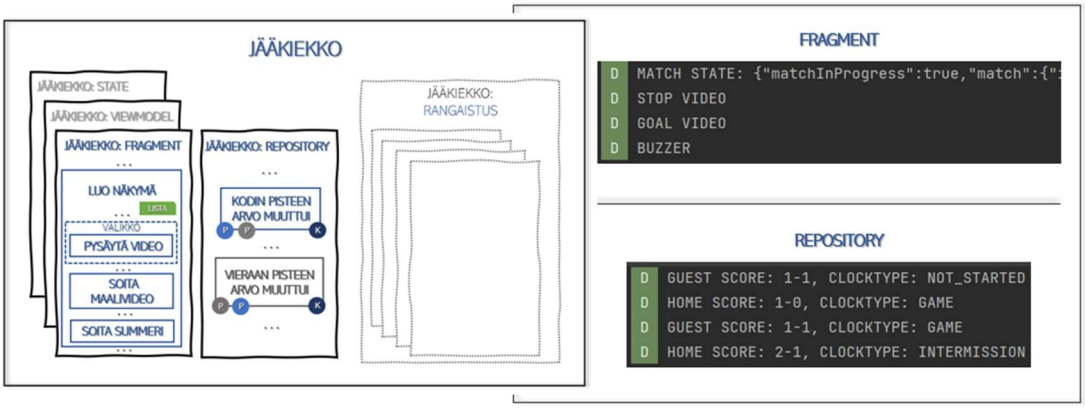
Seurattavan alueen eli tutkittavan alueen määrittämisen ja kaventamisen jälkeen suodatettiin pois tarpeettomat painikkeet, minkä lopputuloksena saatiin valittuja painikkeita seurantaan. Kun painikkeet on konkreettisesti valittu, tutkitaan tämän jälkeen perusteellisemmin näiden koodillista toimintaa testiympäristössä. Testiympäristössä ohjain on kytköksissä Android Studioon, joten sovellusten ollessa käynnissä, painikkeiden toiminnat näkyvät logcat-ikkunasta. Logcatin avulla voidaan tarkastella ja testata tarkemmin tietyn painikkeen toimintoja yksityiskohtaisesti. Logcatillä siis kirjataan mm. muuttujan aiempien ja/tai nykyisten arvojen muutoksia, tietyn metodin aktivointeja sekä fragmentin tilaa (onko se aloitettu vai lopetettu) ja muita vastaavia tapahtumia.

Valitun painikkeen toiminto koostuu monipuolisesti yhdestä tai useammasta luokasta, metodista, oliosta ja funktiosta. Nämä komponentit voivat olla keskenään samankaltaisia ja sijaita hajallaan useissa eri java-tiedostoissa ja kansioissa. Näistä toiminnoista tarkennetaan yhteen toimintoon, joka kuvastaisi parhaiten painikkeen tavoitetta eli valikoidaan oleellisinta toimintoa seurantaa varten. Tämä prosessi saattaa olla välillä haastavaa, kun ei esim. tiedetä toimintojen lopullista määrää, minkä vuoksi niiden etsiminen ja navigoiminen voi

viedä aikaa. Tämä siis vaatii koodin huolellista tarkastelua tiedostoissa ja kansioissa sekä useita eri lokimerkintöjen tekemistä. Valittujen painikkeiden ja niiden toimintojen yksityiskohtaisen tarkastelun sekä onnistuneesti logcatilla testaamisen jälkeen, voidaan luokitella ja esittää näitä painikkeita kategorioina. Painikkeita jaetaan kolmeen eri kategoriaan: pelilajikohtainen toiminnallisuus eli oman pelilajin vaikuttava toiminto, jaetun tapahtuman toiminnallisuus eli lajiominaisuuksiin perustuva toiminto ja yleinen toiminnallisuus eli geneerinen toiminto kaikissa lajeissa. Nämä toiminnallisuudet muodostavat yhdessä ohjaimen valittua pää- ja pelilajinäkymää sekä ottelutapahtumia.

4.2.1 Pelilajikohtainen toiminnallisuus

Lajikohtaisella toiminnallisuudella tarkoitetaan tässä yhteydessä painikkeen toimintoja, jotka toimivat vain tietyssä. Tässä tapauksessa kyseinen toiminto ei ole nähtävissä muissa pelilajissa, kun siirrytään toiseen pelilajiin. Pelilajikohtaisessa kansiossa on tiedostoja, jotka ovat räätälöityjä tähän tiettyyn pelilajiin sekä pelilajin mukaan jaotellut alikansiot rangaistuksille, jossa puolestaan on tarvittavia java-tiedostoja (kuva 13). Ohjaimen tietty pelilajinäkymä on suunniteltu, toteutettu ja testattu pelilajikohtaisten pelisääntöjen mukaan. Pelisääntöjen tarkastelussa on otettu huomioon eri tasojen ja pelaajaryhmien vaikutusta, joiden säännöt voivat vaihdella vähäisesti. Pelilajinäkymässä sivupalkkivalikon sisältö ja otteluhallintaan liittyvät painikkeet on räätälöity vastaamaan pelilajikohtaisen ottelun tarpeita. Pelilajikohtainen kansio voi hakea ottelutietoja kansion ulkopuolelta täydentääkseen valitun pelilajin ottelun alustamista. Nämä ulkopuoliset kansiota kutsutaan jaetun tapahtuman kansioiksi, joissa ottelutiedot alustetaan yhteisesti kaikille lajeille. Jaetun tapahtuman toiminnallisuudesta käsitellään myöhemmin lisää.



Kuva 13. Yksinkertaistettu hahmotus pelilajikohtaisista toiminnoista pelilajikohtaisessa kansiossa.

Kansioiden, tiedostojen ja niiden sisältöjen nimet ja merkinnät ovat keksittyjä eivätkä vastaa todellisiin. Jatkossa tämä käytäntö säilyy. Jääkiekon kansiossa on neljä eri java-tiedostoa pelilajinäkymän arkkitehtuurin rakentamista varten. Näihin tiedostoihin sovelletaan pelilajikohtaisilla säännöillä ja muiden luokan ilmentymillä. Näistä neljästä tiedostoista tai luokista keskitytään vain kolmeen:

1. Fragment.java: koostuu monista itsenäisistä toimintokatkelmista jääkiekon käyttöliittymässä. Tiedostoa voi liittää tämän jääkiekkonäkymän XML-tiedostoon, toisiin pienempiin fragmentteihin tai aktiviteetteihin (rangaistukset).
2. Repository.java: tarjoaa yhtenäisen tavan tietojen hankkimiseen, kuten paikallisesta tietokannasta ja muista tiedostoista. Tiedostossa voi mm. hallita tietojen noutamista ja asettamista (get-set).
3. State.java: toimii sovelluksen tilatietona tai pankkina, missä säilytetään ottelun tietojen ja niiden arvojen (mm. pisteiden, erän, kellonajat yms.) nykyistä tilaa.

Näistä kahdesta tiedostosta löydettiin kuusi sopivaa toimintoa, johon voi sisällyttää viestin lähetystä. Toiminnot ovat joko suoraan kytköksissä painikkeisiin (fragment.java, lukuun ottamatta vihreää pientä laatikkoa) tai osana painikkeiden omissa tapahtumakäsittelyissä (repository.java). Näiden tavoitteet on havainnollistettu omissa tiedostoissaan. Kohdat, jotka on merkitty kolmella pisteellä ('...'), sisältävät koodia. Havainnollistetut toiminnot on siis suoraan otettu koodien seasta. Tiettyihin toimintoihin on lisätty vielä ehtolauseita (if-else -

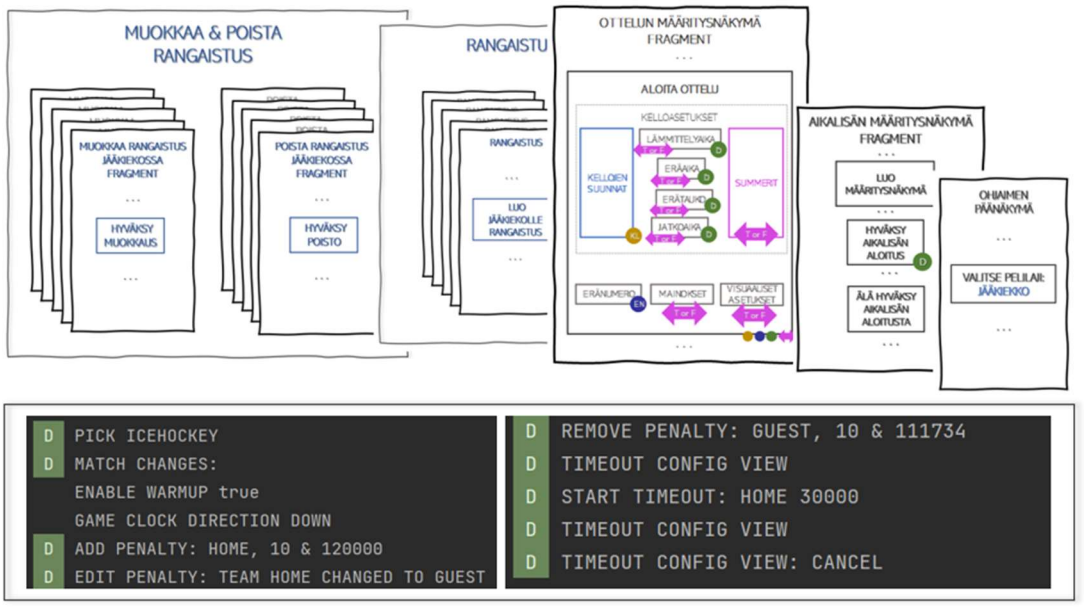
lauseke) ja käytetty listaa (vihreä laatikko) sekä muuttujia (värilliset pallot), jotka ovat peräisin olioista tai omasta luokasta. Omat lisätyt koodit on listattu pseudokoodeina opinnäytetyön viimeiseen sivuun liitteeseen 1. Kun käyttäjä suorittaa toimintoja eli koskettaa painikkeita, tämä käynnistää näitä valittuja toimintoja, minkä jatkeeksi näiden tiedot päätyvät logcat-ikkunoihin (kuvan oikealla olevat osiot). Poikkeuksena on vihreä laatikko fragment-java-tiedostossa, missä on luotu lista state.javan ottelun tiedoista. Kyseinen lista ei synny suoraan painikkeesta, vaan muodostuu heti ottelun käynnistyessä. Ottelun yksityiskohtainen tieto on pyydetty ominaisuus, jotta voidaan tarkastella ottelun lähtökohtaisia tietoja. Ottelutietojen muuttuessa, mukaan lukien kellonaikamuutokset, state.javasta tulee herkästi päivitettyjä ottelutietoja pitkinä riveinä, joka täyttää logcat-ikkunaa ja todennäköisesti myös portaalia. Tämän vuoksi on luotu ehtolause luomaan lista, kun ottelu on vain käynnissä (liite 1). Listalta otetaan ensimmäinen lähtökohtainen tieto (objekti), jossa on tarvittavat tiedot tarkastelua varten.

Tiedostojen rinnalla on jääkiekon rangaistukselle alakansio, jota ei oteta mukaan seurantaan, minkä takia alakansiota ja sen sisältöä on piirretty katkoviivoilla. Pelilajikohtaisen rangaistuksen toiminnallisuus sisältää jatkuvaa toistoa, mikä ei ole toivottava ominaisuus (ks. seurantakriteeri luvulla 4.1). Sen sijaan pelilajikohtaisen rangaistuksen alustamiseen tarvitaan jääkiekkoa koskevat tiedostot tai toiminnot jaetun tapahtuman kansioista, joissa toiminnallisuudet ovat yksiselitteisempiä ja selkeämpiä.

4.2.2 Jaetun tapahtuman toiminnallisuus

Jaetun tapahtuman kansio toimii tietyn tapahtuman yhteisenä paikkana vaikuttaen kaikkiin pelilajeihin ja sijaitsee hajautuneesti Android Studio -projektissa. Koska pelilajien ottelutapahtumat ja -menetelmät eroavat toisistaan, jaetun tapahtuman kansion sisällä yksittäiset tiedostot ja toiminnot määrittävät, miten kyseinen tapahtuma toteutuu tietyissä pelilajeissa. Kyseisen kansion tiedostoissa (kuva 14) käynnistetään käyttäjän toimesta tapahtumien tietojen alustus pelilajikohtaisesti. Tämä voi tapahtua joko ottelun määrityksen aikana tai

ottelun aikana, riippuen tapahtumasta. Nämä alustetut tiedot ja toiminnot siirretään pelilajikohtaiseen kansioon, missä näitä tietoja viimeistellään pelilajikohtaisilla ottelutiedoilla ja -tapahtumilla.



Kuva 14. Yksinkertaistettu hahmotus pelilajikohtaisista toiminnoista jaetun tapahtuman kansioissa.

Jääkiekosta löytyi kaksi jaetun toiminnallisuuden kansioita: yhdessä luodaan rangaistuksia jääkiekkosääntöjen mukaisesti, ja toisessa muokataan tai poistetaan aiemmin luotuja rangaistuksia. Rangaistusten toiminnot liittyvät käyttäjän syötön vastaanottamiseen ja käsittelemiseen. Kun käyttäjä vahvistaa tekemänsä muutokset, lopulliset tiedot vahvistetaan jääkiekon rangaistuksen alakansiossa, ja tässä vaiheessa kyseiset tiedot ilmestyvät logcat-ikkunaan. Rangaistuksen muokkaustiedostossa on lisätty myös ehtolauseita (if-lauseke), jotka avaavat konkreettisesti, mitä muutoksia on tehty (liite 1). Lisäksi on esitetty kolme erillistä java-tiedostoa: ohjaimen päänäkymä ja ottelun sekä aikalisän määritysnäkymä. Ohjaimen päänäkymä toimii ensisijaisena näkymänä käyttäjälle ohjainsovelluksen avautuessa. Tässä näkymässä on useita erilaisia pelilajeja, josta käyttäjä valitsee haluaman pelilajin. Ottelun ja aikalisän määritysnäkymässä asetetaan ottelun tietoja jääkiekkosääntöjen mukaisesti esim. mitä tietoja halutaan esittää ja miten ne näytetään. Nämä tiedot alustetaan joko pelilajikohtaisesti tai yhteisesti eri lajeja, jos niillä on samankaltaisia ominaisuuksia. Pelilajeja voidaan siis asettaa ryhmiin, joissa alustetaan yhteisiä ottelutietoja ja -tapahtumia. Lopulta valittu pelilaji hyödyntää näitä tietoja ja

tapahtumia. Eli tässä tapauksessa jaetun tapahtuman kansioista valitaan vain jääkiekolle alustetut toiminnot. Nämä toiminnot ovat joko samasta tiedostosta, jossa käsitellään muitakin pelilajeja (ottelunMääritysnäkymä.java, aikalisä.java, päänäkymä.java ja rangaistus.java) tai erikseen omista tiedostoista (muokkaa & poista rangaistus -kansio).

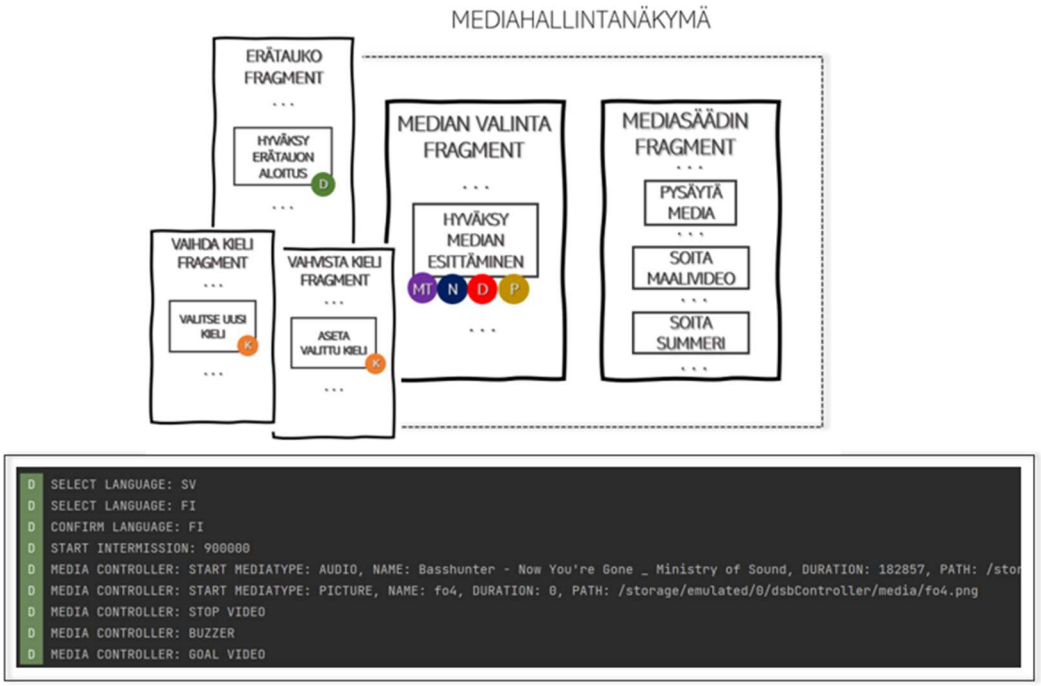
Ottelun määritysnäkymässä on sovellettu rangaistuksen muokkaustiedostossa käytettyjä ehtolauseita ja alustettua message-muuttujaa sekä sen käyttöä. Tässä tiedostossa metodit käsittelevät erikseen kellojen asettamista, niiden suuntaa, kestoa ja summeriääniä. Lisäksi näiden metodien rinnalla on muita metodeja, jotka liittyvät eränumeron asettamiseen, mainosten kytkemiseen ja muihin yksittäisiin visuaalisiin asetuksiin. Olioiden (värilliset pallot) lisäksi on havainnollistettu totuusarvomuuttujan (violetit kaksinuoliset) roolia ottelun aloittamisessa. Totuusarvomuuttuja ilmaisee, onko kyseinen ominaisuus kytketty päälle vai pois (true tai false). Kun näissä metodeissa tapahtuu muutos, näiden tiedot lisätään message-muuttujaan. Tämän jälkeen käyttäjä aloittaa ottelun, ja sen aikana logcat-ikkunaan listataan message-muuttujan tiedot. Aikalisän määritysnäkymässä on ainoastaan lisätty yksinkertainen ehtolause (if-lauseke), jolla löydettiin Peruuta-toiminto määritysnäkymästä poistumiseen.

Vaikka pelilajikohtaiset ja jaetun tapahtuman kansiot kattavatkin täysin pelilajin sääntöjä ja toimenpiteitä, heikkoutena on kuitenkin se, että nämä merkatut viestikohdat eivät tule näkymään kaikissa pelilajeissa. Painike, jonka toiminnallisuus on lajikohtainen, toimii vain jääkiekossa ja jaetun tapahtuman toiminnallisuus toimii jääkiekon lisäksi vain niissä lajissa, joiden säännöt ovat lähes identtiset jääkiekon kanssa. Jos näistä toiminnallisuuksista löydyttäviä viestikohtia halutaan käyttää toisissa pelilajeissa, kehittäjän on aktiivisesti etsittävä näitä viestikohtia ja sovellettava näitä toisiin lajeihin.

4.2.3 Yleinen toiminnallisuus

Painikkeen yleiset toiminnot (kuva 15) ovat nähtävissä kaikissa pelilajissa. Tällöin painikkeen toiminto toteutuu muissakin pelilajissa, kun siirrytään toiseen

pelilajiin. Tämä johtuu siitä, koska samaa luokan ilmentymää käytetään kaikissa pelilajissa, mikä tarkoittaa sitä, että pelilajit jakavat tämän saman luokan. Tässä tapauksessa näiden toimintojen viestikohdista ei tarvitse huolehtia tämän työn jälkeen, ellei ilmene merkittäviä arkkitehtuurillisia muutoksia. Yleisten kansioiden riskinä on, että ne sijaitsevat hajanaisesti Android Studion projektissa, aivan kuten jaetun tapahtuman kansioiden kohdalla. Eräs helpottava tekijä yleisen toiminnallisuuden painikkeen viestikohtien etsimiselle on XML-tiedostot, joissa on määritelty painikkeen rakenteellista muotoa. Vaikka XML-tiedostoja on runsaasti, niiden tarkastelu helpottaa työtä merkittävästi, koska java-tiedostoja on paljon enemmän. XML-tiedostoissa jokaisella painikkeella on attribuuttina id, joka liittyy painikkeen vastaavaan java-tiedostoon, missä toteutetaan sen toiminnallisuutta. Tästä tiedostosta päästään tutkimaan tarkemmin painikkeen toimintoja. Toinen vaihtoehto on käydä jokaista java-tiedostoja yksityiskohtaisesti läpi, ja etsiä niistä luokka, metodi tai olio, jota on kutsuttu jokaisessa pelilajissa.



Kuva 15. Yksinkertaistettu hahmotus yleisistä toiminnoista erilaisissa tiedostoissa.

Yleisten painikkeiden toimintoja määritellään niiden omissa kansioissaan, jossa tarkennettiin tiedostoihin ja yksittäisiin toimintoihin. Median valintaa ja

mediasäädintä esitetään yhtenä kokonaisuutena eli mediahallintanäkymänä, sillä ohjaimessa tämä näkyy yhtenä näkymänä. Yleisen toiminnallisuuden toiminnot ovat yksinkertaisempia ja selkeämpiä, joten niihin ei ole lisätty ehtolauseita, paitsi tarvittaessa muuttujien tai olioiden kutsumista omista luokista.

Tarkasteltaessa painikkeiden toimintaa, voidaan luokitella ne toiminnallisuuksien mukaan eli lajikohtaiseen, jaetun tapahtumaan ja yleiseen toiminnallisuuksiin, joiden vaikutukset pelilajinäkymässä ja ottelutapahtumissa eroavat toisistaan. Yksittäisen painikkeen toiminnallisuuden syvällisemmässä tarkastelussa on havaittu, että pelkkä toimintoihin rajaaminen ei ole aina riittävä. Erityisesti tilanteissa, joissa halutaan näyttää tietyssä metodissa tietyn muuttujan arvoa, voidaan havaita, että kyseistä muuttujaa ei ole hyödynnetty missään vaiheessa. Tällöin tarvitaan omia lisäyksiä, jotta halutun muuttujan arvo saadaan näkyviin. Toimintojen rajausten ja niiden esittämisen yhteydessä ovat viestikohdat olleet selvillä logcatin avulla. Viestikohtia on merkitty kutsumalla haluttuihin toimintoihin Log-luokkaa, minkä aikana näiden viestikohtien toimintojen tiedot näkyvät logcat-ikkunasta. Nämä viestikohdat toimivat indikaattoreina siitä, mistä kohdista seurannan viestin lähetys alkaa. Voidaan todeta, että logcat-ikkunassa olleiden lokitietojen siirrettäessä portaaliin message-kenttään, mahdollistaa samojen lokitietojen näkymisen käyttöliittymässä auditointiviestimuodossa.

4.3 Löydösten esittäminen portaalissa

Kun viestikohtia on määritelty, on havaittu mahdollisia viestikohtia seurannan viestien lähettämiseen. Nämä viestikohdat sijaitsevat painikkeiden toiminnallisuuksissa, joita on käyty läpi. Seuraavassa vaiheessa, jossa korvataan näiden viestikohtia suoraan viestin lähetyksellä, on yksinkertaisempi toteuttaa. Edellisten toiminnallisuuksien (lajikohtaisen, jaetun tapahtuman ja yleisen) viestikohtia on merkitty Log()-luokalla, jonka ansiosta saatiin havainnollistettu kuva painikkeiden toiminnoista logcat-ikkunoissa. Näitä merkittyjä Log()-luokkia korvataan joko viitataan suoraan AMQPConnection-luokkaan ja haluttua metodia luokan sisältä tai vastaavasti luokan ilmentymillä eli olioilla, ja annetaan oliolle

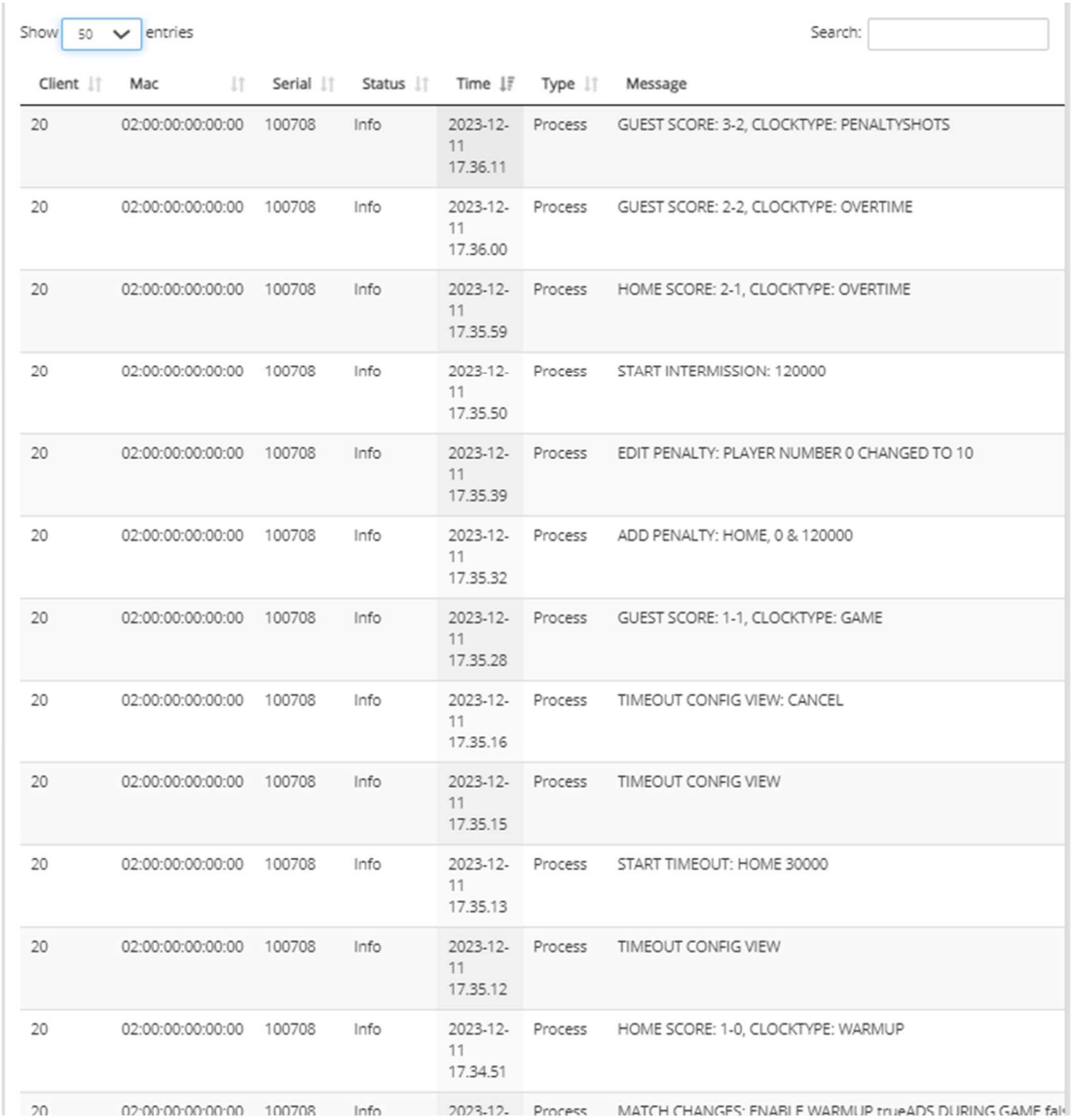
toteutettavaksi luokan metodi. Kertauksena metodina toimii AMQPConnection-luokasta sendMessageToDeveloper, johon on havainnollistettu basicPublish():n lisäksi viestin käsittely ennen palvelimeen lähettämistä (ohjelma 4).

```
public void sendMessageToDeveloper(String message){
    try {
        JsonObject body = new JsonObject();
        body.addProperty("message", message);
        String messageJson = body.toString();

        channel.basicPublish(String  exchange,
                             String  routingKey,
                             AMQP.BasicProperties props,
                             byte[] body);
    } catch (IOException e) {
        Log.e(LOG_TAG, "Error", e);
    }
}
```

Ohjelma 4. Portaaliin viestin lähetyksen metodi.

Metodissa sendMessageToDeveloper tarvitaan merkkijonoista parametria (message), johon lisätään haluttua viestiä, kun viitataan kyseistä metodia, esim. AMQPConnection.getInstance().sendMessageToDeveloper("Haluttu viesti"). Kun sendMessageToDeveloper-metodi saa parametrikseen jotakin viestiä, annetaan se message-kentän arvoksi body.addProperty():n sulkujen sisällä. Tämän jälkeen viestiä julkaistaan basicPublish():n avulla, jonka jälkeen viesti näkyy portaalissa (kuva 16). Kokonaisuudessaan sendMessageToDeveloper-metodia on viitattu 27 kertaa, joka tarkoittaa sitä, että käyttäjälähtöisiä auditointiviestejä on luotu 27 kertaa.



Show 50 entries Search:

Client	Mac	Serial	Status	Time	Type	Message
20	02:00:00:00:00:00	100708	Info	2023-12-11 17.36.11	Process	GUEST SCORE: 3-2, CLOCKTYPE: PENALTYSHOTS
20	02:00:00:00:00:00	100708	Info	2023-12-11 17.36.00	Process	GUEST SCORE: 2-2, CLOCKTYPE: OVERTIME
20	02:00:00:00:00:00	100708	Info	2023-12-11 17.35.59	Process	HOME SCORE: 2-1, CLOCKTYPE: OVERTIME
20	02:00:00:00:00:00	100708	Info	2023-12-11 17.35.50	Process	START INTERMISSION: 120000
20	02:00:00:00:00:00	100708	Info	2023-12-11 17.35.39	Process	EDIT PENALTY: PLAYER NUMBER 0 CHANGED TO 10
20	02:00:00:00:00:00	100708	Info	2023-12-11 17.35.32	Process	ADD PENALTY: HOME, 0 & 120000
20	02:00:00:00:00:00	100708	Info	2023-12-11 17.35.28	Process	GUEST SCORE: 1-1, CLOCKTYPE: GAME
20	02:00:00:00:00:00	100708	Info	2023-12-11 17.35.16	Process	TIMEOUT CONFIG VIEW: CANCEL
20	02:00:00:00:00:00	100708	Info	2023-12-11 17.35.15	Process	TIMEOUT CONFIG VIEW
20	02:00:00:00:00:00	100708	Info	2023-12-11 17.35.13	Process	START TIMEOUT: HOME 30000
20	02:00:00:00:00:00	100708	Info	2023-12-11 17.35.12	Process	TIMEOUT CONFIG VIEW
20	02:00:00:00:00:00	100708	Info	2023-12-11 17.34.51	Process	HOME SCORE: 1-0, CLOCKTYPE: WARMUP
20	02:00:00:00:00:00	100708	Info	2023-12-	Process	MATCH CHANGES: ENABLE WARMUP trueADS DURING GAME fal

Kuva 16. Auditointiviestit portaalissa.

Viestikohdat korvattiin viestin lähetyksellä, ja kaikkien painikkeiden toimintojen tiedot tallennettiin onnistuneesti portaaliin. Message-kentän alapuolella olevat auditointiviestit vastaavat täysin logcat-ikkunan lokiviestejä, vaikka kaikki viestit eivät näy kuvassa. Jäljelle jäivät viestien pienet kirjoitus- ja välilyöntivirheet. Lisäksi toimintojen rajauksen ja viestikohtien merkitsemisen aikana on käytetty logcatin käytön aikana rivinvaihtomerkkiä (”\n”), joka ei näy portaalissa, minkä vuoksi rivinvaihtomerkkiä poistetaan käytöstä. Auditointiviestin tyyppi on type-kentässä toistaiseksi Process User Input:n sijasta, koska ei ollut pääsyä portaalin ohjelmakoodiin. Auditointiviestin tyyppiä vaihdetaan lähiaikoina, kuitenkin vasta tämän toteutuksen jälkeen.

Versionhallinnassa Android Studion omassa branchissä tehtyjä muutoksia ei yhdistetä suoraan ohjaimen pääversioon, vaan siirretään omaa branchiä paikallisesta tietokoneesta (local) etäiseen repositoryyn (origin). Originissa oma branch on jaettavissa muiden kanssa, kun taas paikallisessa koneessa muutokset näkyisivät vain itsellään. Branchin ollessa originissa, kyseinen branch on vieläkin erillään main branchistä (päähaara). Tässä välissä muilla on vielä mahdollisuus parannella seurannan toiminnallisuutta, ennen main branchiin yhdistämistä.

5 LOPUKSI

Tässä opinnäytetyössä suunniteltiin, testattiin ja toteutettiin seurannan käyttöönottoa onnistuneesti tulostaulujärjestelmän testiympäristössä. Alussa tutustuttiin yleisesti tulostaulujärjestelmän fyysisiä laitteistoja ja testiympäristön tarvittavia komponentteja, kuten fyysiset laitteistot eli hub, kosketusnäyttö, Android-laite, tarvittavat kaapelit ja SIM-kortti sekä muistitikkua ohjain- ja tulostaulusovellusten asentamiseen (apk-tiedostot) ja laitteiden konfigurointiin (config.json-tiedostot) ADB:n välityksellä. Lisäksi tarvitaan oma kannettava tietokone, johon on asennettu Android Studio, testiympäristön toimimista varten. Testiympäristön tarkoituksena on simuloida tulostaulujärjestelmän toimintaa tuotannossa, minkä vuoksi toteutettiin seurannan käyttöönottoa tähän ympäristöön. Seurannan käyttöönoton vaiheet sisältävät: viestikohtien määrittäminen, niiden etsiminen ja merkitseminen sekä lopuksi viestikohtien korvaaminen viestin lähetyksellä, minkä ansiosta saatiin käyttäjälähtöiset auditointiviestit portaalin selaimeen näkyviin.

Tässä työssä olen tutustunut uusiin käsitteisiin ja teknillisiin työvälineisiin, joita mainittiin edellisessä kappaleessa. Olen varautunut, että opinnäytetyöprosessi on aikaa vievää ja monimutkaista, varsinkin kunnollisen suunnitelman ja tutkimuskysymysten puuttuessa. Näiden puitteiden osalta sain onneksi kuntoon opinnäytetyön keskimmäisessä vaiheessa. Jos aloittaisin työn uudestaan, aloittaisin toteuttamaan seurannan käyttöönottoa ensimmäisenä asiana enkä lähtisi liikkeelle opinnäytetyörakenteen kronologisessa järjestyksessä. Tällä tavoin olisin saanut paremman käsityksen siitä, mitä kappaleita tulisi sisällyttää opinnäytetyöhön ennen varsinaista toteutusta.

Suurimmat haasteet työni kannalta olivat auditointiviestin käsitteen ymmärtäminen ja testiympäristön pystyttäminen. ”Auditointiviesti”-nimikään ei ole standardoidu sana, vaan keksin sen Audit Messagesta. Auditointiviestiin liittyviä artikkeleita on ollut vaikeuksia löytää sekä suomeksi että englanniksi. Löysin muutamia dokumentteja auditointiviestiin liittyen, mutta niissä tarkastellaan lähinnä laitteiden lähettämiä auditointiviestejä eikä samalla tavalla kuin esim.

lokissa tai ADB:ssä. Siitä huolimatta löysin ja sain tarvittavat tiedot ja ymmärryksen seurannan käyttöönoton toteutukseen. Testiympäristön pystyttäminen on edellyttänyt monia eri vaiheita ja laitteita, mikä on kuluttanut odotettua enemmän aikaa. Erityisesti tietyn kaapelin unohtaminen toimeksiantajan toimistolle tai yllättävät yhteysongelmat ovat olleet odottamattomia haasteita. Onneksi olen kuitenkin saanut arvokasta apua toimeksiantajan työntekijöiltä. Työn loppuvaiheessa olen havainnut, että viestikohdissa olisi ollut parannettavaa. Esim. olisin voinut tehdä enemmän lisäyksiä ohjelmakoodiin, jotta voisin tarkentaa vielä yksityiskohtaisemmin painikkeiden toimintoihin. Painikkeiden toiminnallisuuksia voisi varmasti nimetä selkeämmin. Lisäksi olisi ollut tarpeen tutustua perusteellisemmin AMQPConnection-luokan sendMessageToDeveloper-metodiin, sillä huomasin vasta loppuvaiheessa, että olisin voinut luoda lisää objekteja message-kentän sisälle. Tämä olisi selkeyttänyt auditointiviestin luettavuutta entisestään.

Seurannan tulevaisuus on epäselvä, sillä tulostaulujärjestelmän ohjelmakoodiin todennäköisesti tullaan tekemään ajoittain muutoksia, mikä voi vaikuttaa viestin lähetyksen kohtia. Lähitulevaisuudessa tämän seurannan käyttöönoton pohjalta voisi tarkastella käyttäjän tekemien toimintojen lisäksi myös toteutetun seurannan käyttöä ja toimintaa. Tämän perusteella on mahdollista kerätä palautetta kehittäjiltä seurannan jatkokehittämistä varten. Kehittäjiltä voitaisiin kysyä esim., mitkä tiedot eivät vastaa heidän tarpeitaan tai ovat epäselviä, jotka auttaisivat optimoimaan seurannan toimintaa ja tarkoitusta entisestään.

LÄHTEET

Android Developers. (n.d.-a). Android Debug Bridge (adb). Haettu 19.10.2023 osoitteesta https://developer.android.com/tools/adb

Android Developers. (n.d.-b). [Android Debug Bridge (adb) selitys kuvana]. Haettu 19.10.2023 osoitteesta https://developer.android.com/tools/adb

Android Developers. (n.d.-c). Fragment. Haettu 21.9.2023 osoitteesta https://developer.android.com/reference/android/app/Fragment

Android Developers. (n.d.-d). Meet Android Studio. Haettu 14.12.2023 osoitteesta https://developer.android.com/studio/intro

Android Developers. (n.d.-e). View logs with Logcat. Haettu 14.12.2023 osoitteesta https://developer.android.com/studio/debug/logcat

Aspose. (n.d.). What is a Flowchart? Haettu 25.11.2023 osoitteesta https://products.aspose.app/diagram/pages/what-is-flowchart

Avisekhar, R. (2016). The Android Game Developer's Handbook : Discover an All in One Handbook to Developing Immersive and Cross-Platform Android Games. Packt Publishing Ltd.

Cheong, S. (9.1.2023). Essential Guide to MQTT Topics and Wildcards. https://cedalo.com/blog/mqtt-topics-and-mqtt-wildcards-explained/

Digitaalinen Helsinki. (n.d.). Rajapinnat ja käyttöliittymät. Haettu 26.9.2023 osoitteesta https://digi.hel.fi/digipalveluopas/tietoa-teknologiasta/rajapinnat-ja-k%C3%A4ytt%C3%B6liittym%C3%A4t/

DNSstuff. (20.4.2020). What Is an Audit Log? Audit Trails and How to Use Audit Logs. https://www.dnsstuff.com/what-is-audit-log

Doherty, E. (15.4.2020). What is object-oriented programming? OOP explained in depth. https://www.educative.io/blog/object-oriented-programming

International Business Machines. (11.8.2023). About Audit Messages. Haettu 24.8.2023 osoitteesta https://www.ibm.com/docs/en/sgfmw/5.3.1?topic=notification-about-audit-messages

NetApp. (n.d.). Audit message examples. Haettu 18.11.2023 osoitteesta https://library.netapp.com/ecmdocs/ECMP12481619/html/GUID-1FD2FE07-A18F-44FA-A3B3-C7860E739A72.html

Particular Software. (n.d.). Auditing Messages. Haettu 18.11.2023 osoitteesta https://docs.particular.net/nservicebus/operations/auditing

PTC. (n.d.). ThingWorx Audit Messages. Haettu 18.11.2023 osoitteesta https://support.ptc.com/help/thingworx_hc/thingworx_8_hc/en/index.html#page/ThingWorx/Help/Composer/System/Subsystems/AuditSubsystemMessages.html

RabbitMQ. (n.d.-a). AMQP 0-9-1 Model Explained: Fanout exchange routing. [valokuva]. RabbitMQ. Haettu 13.8.2023 osoitteesta https://www.rabbitmq.com/tutorials/amqp-concepts.html

RabbitMQ. (n.d.-b). Publish/Subscribe. Haettu 11.8.2023 osoitteesta https://www.rabbitmq.com/tutorials/tutorial-three-java.html

Rinne, J. (17.7.2023a). Henkilökohtainen keskustelu Jidoka Technologies Oy:n työntekijän, Juuso Rinteen, kanssa.

Rinne, J. (17.7.2023b). [Valokuva MQTT publish-subscribe -mallista].

Taloushallinto. (n.d.). Aukoton kirjausketju eli audit trail. Haettu 17.11.2013 osoitteesta https://taloushallintoliitto.fi/tietopankki/kirjanpidon-abc/aukoton-kirjausketju-eli-audit-trail/

Topchyi, L. (21.7.2022). What is a SourceTree and How to Use It. https://www.alphaservesp.com/blog/what-is-a-sourcetree-and-how-to-use-it/

Track4Services. (n.d.). Portable Outdoor Wifi. Haettu 5.8.2023 osoitteesta https://www.t4s.com/portablehub/

Traficom. (2023). Näin keräät ja käytät lokitietoja. Haettu 20.11.2023 osoitteesta https://www.kyberturvallisuuskeskus.fi/fi/ajankohtaista/ohjeet-ja-oppaat/nain-keraat-ja-kaytat-lokitietoja

Viestintävirasto. (2016). Lokien keräys ja käyttö. https://www.kyberturvallisuuskeskus.fi/sites/default/files/media/file/Lokitusohje.pdf

Yakuba, D. & Zykov, S. (2022). Development of an object library for the EO programming language. ScienceDirect. https://doi.org/10.1016/j.procs.2022.09.496

LIITE 1: OPINNÄYTETYÖSSÄ KÄYTETTÄVÄT KOODIT

Java-tiedosto	Lisätyt muutokset pseudokoodeina

Jääkiekon fragment

```
// Alustetaan tyhjä lista
private Lista<State> stateList = uusi Taulukko-
lista<>();
                              …
public View luoNäkymä(@EiSallitaNull parametri, par-
ametri, parametri) {

                              …

   Jos (state.onkoOtteluKäynnissä() == true) {
       lisää nykyinenState listaan: stateList;
       Jos (stateList:ssä on ensimmäinen
            nykyinenState) {
          Tulosta(tunniste: "AUDIT_MSG",
          viesti:"MATCH STATE: " + stateString);
       }
   }

                              …
}
                              …
```

Jääkiekon repository

```
                              …
// Koti piste
Jos (state.haeOttelu ei ole käynnissä) {
Tulosta(tunniste: "AUDIT_MSG", viesti: "HOME SCORE:
" + repository.haeKotiPiste + "-" + repository.hae-
VierasPiste + rivinvaihto + ", CLOCKTYPE: " + repo-
sitory.haeNykyinenKelloTyyppi);
} Muuten {
Tulosta(tunniste: "AUDIT_MSG", viesti: "HOME SCORE:
" + repository.haeKotiPiste + "-" + repository.hae-
VierasPiste + rivinvaihto + ", CLOCKTYPE: " + repo-
sitory.haeKelloTyppi);
}

                              …

// Vieras piste
Jos (state.haeOttelu ei ole käynnissä) {
Tulosta(tunniste: "AUDIT_MSG", viesti: "GUEST SCORE:
" + repository.haeVierasPiste + "-" + repo-
sitory.haeKotiPiste + rivinvaihto + ", CLOCKTYPE: "
+ repository.haeNykyinenKelloTyyppi);
} Muuten {
Tulosta(tunniste: "AUDIT_MSG", viesti: "GUEST SCORE:
" + repository.haeVierasPiste + "-" + repo-
sitory.haeKotiPiste + rivinvaihto + ", CLOCKTYPE: "
+ repository.haeKelloTyppi);
}
                              …
```

Muokkaa rangaistus

```
                                …

// Alustetaan message-muuttujaa tyhjällä merkkijonolla
String message = "";

Jos (alkuperäinen.haeJoukkue ei ole sama kuin nykyinenRangaistus.haeJoukkue) {
message += "TEAM " + alkuperäinen.haeJoukkue + " CHANGED TO " + nykyinenRangaistus.haeJoukkue + rivinvaihto;
}
Jos (alkuperäinen.haePelaajaNumero ei ole sama kuin nykyinenRangaistus.haePelaajaNumero) {
message += "PLAYER NUMBER " + alkuperäinen.haePelaajaNumero + " CHANGED TO " + nykyinenRangaistus.haePelaajaNumero + rivinvaihto;
}
Jos (alkuperäinen.haeKesto ei ole sama kuin nykyinenRangaistus.haeKesto) {
message += "DURATION " + alkuperäinen.haeKesto + " CHANGED TO " + nykyinenRangaistus.haeKesto + rivinvaihto;
}

Jos (message on tyhjä) {
Tulosta(tunniste: "AUDIT_MSG", viesti: "EDIT PENALTY: " + rivinvaihto + message);
}
                                …
```