Tuomas Bergholm

Android Auto -toiminnallisuus audiontoistomobiilisovellukseen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

8.5.2023

Tiivistelmä

Tekijä:	Tuomas Bergholm
Otsikko:	Android Auto -toiminnallisuus audiontoistomobiilisovellukseen
Sivumäärä:	44 sivua
Aika:	8.5.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Mobile Solutions
Ohjaajat:	Kehitysmanageri Kalle Toivonen Yliopettaja Jarkko Vuori

---

Insinöörityön tarkoituksena oli kehittää Supla-podcast- ja nettiradiopalvelun Android-mobiilisovellukseen tuki Android Auto -alustalle ja tutkia mediasovelluksen arkkitehtuuria, jonka Android Auto -toiminnallisuus vaatii. Sovelluksen käyttökokemusta ja helppokäyttöisyyttä autossa käytettäessä haluttiin parantaa, ja Android Auto -tuki tuo käyttäjille sovellukseen lisäarvoa.

Toteutuksen vaaditut ominaisuudet olivat suositeltujen podcastien, omien suosikkien ja radiokanavien selaaminen sekä toiston hallinta Android Auto -järjestelmää tukevan auton medianäytöllä puhelimen ollessa yhdistettynä auton tietoviihdejärjestelmään.

Android Auto -mediasovelluksen toiminta perustuu mobiilisovelluksen mediasoitintoteutuksen MediaLibraryServiceen, joka toimii sovelluksessa taustalla. Palvelu tarjoaa mediasisältöä asiakassovelluksille ja mahdollistaa mediantoiston hallintaan liittyvät toiminnot. Uuden Media3-kirjaston avulla voidaan toteuttaa entistä helpommin suositellun arkkitehtuurin mukainen mediasoitintoteutus, joka toimii myös Android Auto -järjestelmän kanssa hyvin yhteen.

Ominaisuus kehitettiin yrityksen vaatimusten pohjalta noudattaen Googlen määrittelemiä standardeja ja ohjeistuksia aiheeseen liittyen. Sovellusta testattiin kehitysprosessin aikana jatkuvasti emulaattorilla.

Työn lopputulos oli toimiva Android Auto -toteutus sovellukseen, ja siinä on kaikki vaaditut ominaisuudet. Toiminto julkaistaan kesän 2023 kuluessa tuotantoversioon käyttäjille. Android Auto -tuen lisäämisen olemassa olevaan mediasovellukseen todettiin olevan suhteellisen yksinkertaista, jos sovelluksessa on jo ennestään käytössä Androidin suositellun media-arkkitehtuurin mukainen mediasoitintoteutus.

Avainsanat: Android Auto, Android, mediantoistosovellus, podcast, nettiradio, tietoviihdejärjestelmä, mediasoitin, audiomedia

Abstract

Author:	Tuomas Bergholm
Title:	Android Auto functionality into an audio playback mobile app
Number of Pages:	44 pages
Date:	8 May 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Mobile Solutions
Supervisors:	Kalle Toivonen, Development Manager Jarkko Vuori, Principal Lecturer

---

The purpose of the thesis was to develop Android Auto support for the Android mobile application of the Supla podcast and internet radio service. In addition, the media app architecture that Android Auto functionality requires was researched. The goal was to improve the user experience and ease of use of the application while using it in a car.

The required features of the implementation included browsing and managing the playback of recommended podcasts, favorites and radio stations on an Android Auto supported infotainment screen. The operation of an Android Auto media app is based on the MediaLibraryService of a mobile app's media player implementation which runs in the background.

The feature was developed based on the requirements of the company while following the standards and instructions on the subject defined by Google. The app was being tested constantly on an emulator during the development process.

The result of the thesis was a working Android Auto implementation to the app that included all the required features. The feature will be released to production for the users during the summer of 2023. Adding Android Auto support to an existing media app was found to be relatively simple as long as it already has a media player implementation according to the Android's recommended media app architecture.

Keywords: Android Auto, Android, media playback app, podcast, internet radio, infotainment system, media player, audio media

Sisällys

Lyhenteet

ADB: *Android Debug Bridge*. Monikäyttöinen komentorivityökalu, jolla voidaan kommunikoida Android-laitteen kanssa.

API: *Application Programming Interface*. Ohjelmointirajapinta eli tapa, jolla ohjelmistot tarjoavat tietoa ja palveluita muille sovelluksille ja järjestelmille.

APK: *Android Package Kit*. Android-sovelluksen asentamiseen ja jakeluun käytetty asennuspakettitiedostomuoto.

DHU: *Desktop Head Unit*. Android Auto -sovelluksen testaamiseen tarkoitettu työasemaympäristössä toimiva auton Android Auto -pääteyksikön mukainen emulaattori.

JSON: *JavaScript Object Notation*. Yksinkertainen, helposti luettava ja laajasti eri ympäristöissä käytössä oleva tiedostomuoto datan välittämiseen ja tallentamiseen.

TCP: *Transmission Control Protocol*. Tietoliikenneprotokolla luotettavaan tiedonsiirtoon verkossa laitteiden ja sovellusten välillä.

UID: *User ID*. Android-laitteeseen asennetun sovelluksen yksilöllinen tunniste.

URL: *Uniform Resource Locator*. Yksittäisen resurssin tai kohteen tarkkaan sijaintiin verkossa osoittava osoite.

1 Johdanto

Opinnäytetyössä toteutetaan Supla-podcast- ja nettiradiopalvelun Android-mobiilisovellukseen tuki Android Auto -järjestelmälle.

Autojen tietotekniset järjestelmät ovat kehittyneet viime vuosien aikana merkittävästi, ja uusissa autoissa onkin nykyään varsin paljon erilaisia tietokonejärjestelmiä. Autoilijoille näistä näkyvin ja merkittävin on kosketusnäyttöpohjainen tietoviihdejärjestelmä, joka nykyään mahdollistaa varsin monipuolisesti erilaisia hyödyllisiä toimintoja ja sovelluksia ja jonka käyttökokemus vastaa pitkälti älypuhelimia ja tabletteja.

Vielä kymmenisen vuotta sitten, kun kosketusnäytölliset tietoviihdejärjestelmät alkoivat yleistyä autoissa, ne olivat yleisesti käytettävyydeltään kankeita, hitaita ja selvästi mobiililaitteita jäljessä. Melkein kaikilla autonvalmistajilla oli oma järjestelmänsä, ja muun muassa navigointi- ja karttatoimintojen laatu oli vaihtelevaa. Vuonna 2014 Apple julkaisi CarPlay-järjestelmänsä, ja vuonna 2015 Google toi markkinoille Android Auto -järjestelmän, jotka mahdollistivat älypuhelimen yhdistämisen auton tietoviihdejärjestelmään ja puhelimen toimintojen hallinnan auton kosketusnäytöltä (1; 2). Tämä paransi huomattavasti muun muassa navigaatio- ja mediasovellusten käyttökokemusta autossa, ja yhtenäinen sovellusalusta saatiin käyttöön useiden erimerkkisten autojen kesken.

Nykyään Android Auto- ja Apple CarPlay -järjestelmät kuuluvat melkein kaikkien uusien autojen vakiovarusteisiin, ja monilla suosituilla musiikki-, podcast- ja äänikirjasovelluksilla on tuki näille järjestelmille. Yrityksen Sanoma Media Finland/Nelonen Media Supla-sovelluksen käytettävyyttä autossa halutaan parantaa, ja opinnäytetyön tarkoituksena on kehittää Android Auto -tuki sovellukseen. Lisäksi työssä tutustutaan tarkemmin Android-mediasovellusten yleiseen ja suositeltuun arkkitehtuuriin ja rakenteeseen sekä tutkitaan, mitä Android Auto -tuen lisääminen olemassa olevaan mediasovellukseen vaatii.

2 Työn lähtötilanne

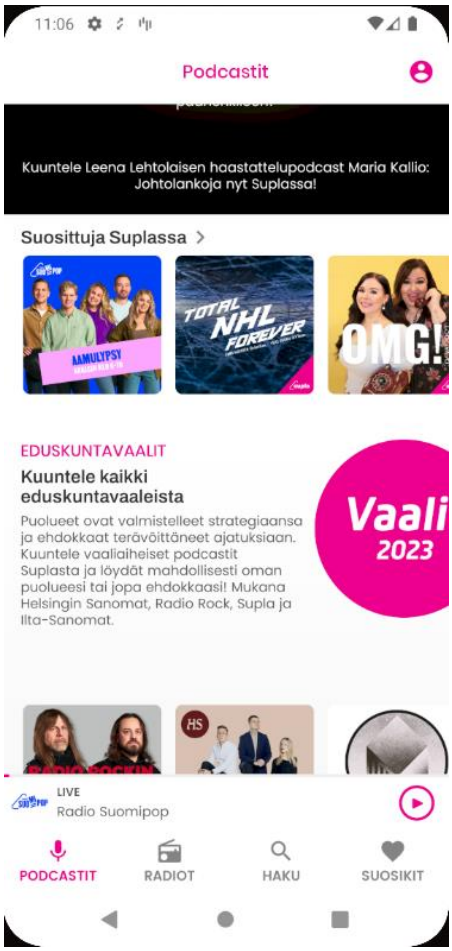
Tässä luvussa esitellään työn tilannutta yritystä ja sen Supla-Android-sovellusta. Lisäksi kuvataan Android Auto -toteutuksen vaatimukset.

2.1 Tilaajayritys

Sanoma Media Finland on Suomen suurin kaupallinen mediayhtiö, jonka tuotteisiin kuuluu muun muassa Helsingin Sanomat, Ilta-Sanomat, useita radiokanavia, Nelonen-tv-kanavat, Ruutu-suoratoistopalvelu sekä Supla-podcast- ja netti-radiopalvelu. Yrityksen palvelut ja sovellukset ovat saatavilla useille eri alustoille, joista Android on yksi tärkeimmistä median siirtyessä entistä enemmän digitaaliseen muotoon ja erityisesti mobiililaitteilla kulutettavaksi. (3.)

2.2 Supla-Android-sovellus

Supla-palvelun Android-sovelluksen tärkeimpiin ominaisuuksiin kuuluu podcastien selaaminen niin etusivulla kuin myös kategorioittain, radiokanavat, omat suosikit ja näiden audiosisältöjen toistaminen ja toiston hallinta soitinnäkymässä (kuva 1). Sovellus on kirjoitettu pääosin Kotlin-ohjelmointikielellä, ja käyttöliittymä on toteutettu XML-layouteilla ja osin myös Jetpack Compose -kirjaston avulla. Sovellus käyttää useita yrityksen API-rajapintoja eri toimintojen toteuttamiseen. Sovellusta käyttävät päivittäin kymmenet tuhannet suomalaiset.



Kuva 1. Supla-Android-sovelluksen Podcastit-etusivu (4).

2.3 Android Auto -toteutuksen vaatimukset

Työssä tuli toteuttaa Supla-sovellukseen toimiva ja Googlen vaatimusten ja standardien mukainen Android Auto -toiminnallisuus, joka

- sisältää päävälilehtivalikon, johon kuuluvat Podcastit-, Radiot- ja Suosikit-sivut
- listaa Podcastit-sivulla suosituimpia podcast-sarjoja ruudukkotyyppisessä listassa
- listaa Radiot-sivulla kaikki Suplan radiokanavat
- listaa Suosikit-sivulla käyttäjän suosikeiksi lisäämät podcastit
- listaa podcast-sarjan uusimmat jaksot, kun käyttäjä valitsee jonkin podcastin
- aloittaa kohteen toiston, kun käyttäjä valitsee jonkin sisällön

- mahdollistaa 30 sekuntia taakse- ja eteenpäin kelaamisen pysäytä- ja toista-kontrollien lisäksi
- tukee puhekomentoja Google Assistantin kautta soitinkontrollien ja sisältöjen haun osalta.

Android Auto -toiminnon tulee siis tarjota vain tärkeimmät perusominaisuudet sovelluksesta, eikä kaikkia lisätoimintoja ole järkevää tai tarpeellista sisällyttää sovelluksen autoversioon. Syitä tähän ovat Googlen standardit, joiden mukaan sovelluksen käytön tulee olla mahdollisimman suoraviivaista ja vähän ajamista häiritsevää, sekä Android Auto -järjestelmän rajoitetut mahdollisuudet erikoisempien toimintojen toteuttamiseen (5).

3 Android Auto -järjestelmä

Tässä luvussa tutustutaan tarkemmin Android Auto -järjestelmän toimintaperiaatteeseen, Android Auto -mediasovellusten toteuttamiseen sekä niiden eri toimintoihin ja rajoituksiin.

3.1 Toimintaperiaate

Android Auto on erityisesti autoissa käytettäväksi kehitetty versio Googlen Android-mobiilikäyttöjärjestelmästä. Järjestelmä toimii siten, että kun käyttäjä yhdistää Android-puhelimen auton tietoviihdejärjestelmään, aukeaa auton näytöllä Android Auto -käyttöliittymä, joka tarjoaa pääsyn puhelimen tärkeimpiin toimintoihin (6). Järjestelmä toimii puhelimessa taustalla toimivan Android Auto -sovelluksen avulla. Se ei siis varsinaisesti ole oma täysimittainen käyttöjärjestelmänsä.

Autonvalmistajilla on tyypillisesti ollut haasteita pysyä älypuhelinten ja sovellusten teknologisen kehityksen perässä tietoviihdejärjestelmien toteutuksessa, ja usein autojen omat navigointi- ja mediajärjestelmät ovat tuntuneet vanhentuneilta jo muutaman vuoden jälkeen auton myyntiintulosta. Tähän Google on tuonut ratkaisun Android Autollaan, joka käyttää puhelimen resursseja sovellusten pyörittämiseen, peilaa ne auton näytölle ja mahdollistaa hallinnan auton

painikkeilla ja mikrofoneilla. Puhelin toimii siis niin sanotusti järjestelmän aivoina, ja auto puolestaan hoitaa sisältöjen näyttämisen ja toistamisen sekä komentojen välittämisen puhelimelle.

Käyttäjä näkee auton näytöllä yksinkertaistetun käyttöliittymän, jonka kautta voidaan käyttää tiettyjä puhelimen ominaisuuksia, mutta ei kaikkia. Käytettäviin toimintoihin kuuluvat muun muassa navigointi, musiikintoisto, viestit ja puhelut (kuva 2). Rajoitettuihin ominaisuuksiin on syynä helppokäyttöisyyden ja ajoturvallisuuden varmistaminen.

Kuva 2. Android Auton päävalikko auton tietoviihdekeskusnäytöllä (7).

Android Auton lisäksi on olemassa Android Automotive OS, joka on itsenäinen, erityisesti autoja varten kehitetty versio Android-käyttöjärjestelmästä. Sen käyttö ei ole vielä läheskään yhtä yleistä kuin Android Auton, ja sen onkin ottanut käyttöön vasta muutama autonvalmistaja. Android Automotive OS ei tarvitse älypuhelinta toimiakseen. (8.)

Nykyisin lähes kaikilla automerkeillä on automalleja, jotka tukevat Android Autoa, mutta se ei ole silti jokaisessa autossa tuettuna. Jälkeenpäin asennettavia

tietoviihdejärjestelmäyksiköitä (Infotainment Head Unit), joissa on mukana Android Auto, on myös saatavilla. Android Auto on julkaisustaan vuodesta 2015 lähtien yleistynyt hiljalleen ja alkaa nykyään olemaan perusvaruste, vaikkakin alkuun jotkin autonvalmistajat olivat vastentahtoisia ottamaan Android Autoa käyttöön oman järjestelmänsä rinnalle (9).

Käyttäjän näkökulmasta Android Auto on varsin helppokäyttöinen ja hyödyllinen toiminto, joka parantaa myös ajoturvallisuutta (10).

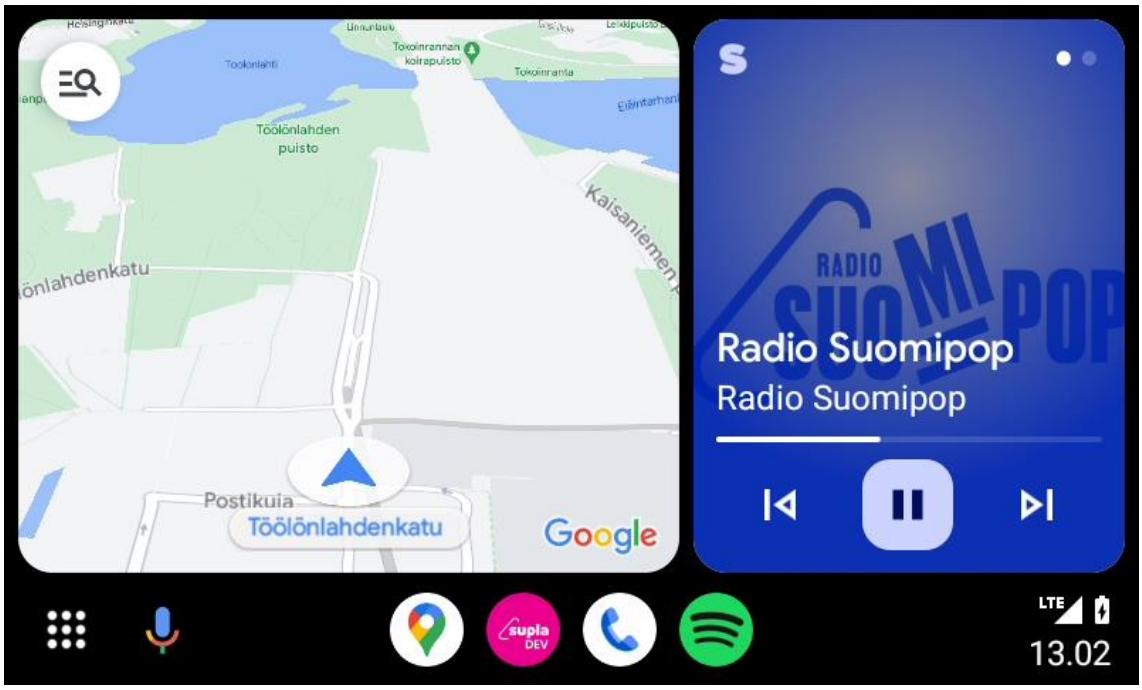
3.2 Android Auton mediasovellukset

Google on määritellyt tietyt Android Auto -sovelluskategoriat, jonka tyyppisiä sovelluksia kehittäjät voivat kehittää alustalle. Yksi yleisimmistä on audiomediasovellukset. Muita kategorioita ovat viestittely-, navigaatio-, kohdepiste-, IoT- ja videosovellukset. (8.)

Mediasovellukset mahdollistavat musiikin, radioiden, äänikirjojen ja muiden audiosisältöjen selaamisen ja toistamisen (8). Esimerkiksi Spotify ja YouTube Music ovat suosittuja mediasovelluksia Android Autossa. Tässä työssä keskitytään nimenomaan audiomediatyyppisen Android Auto -sovelluksen kehitykseen ja ominaisuuksiin.

3.3 Toiminnot ja rajoitukset

Android Auton yksi keskeisimmistä ominaisuuksista on navigointi Google Maps -sovelluksen avulla, joka aukeaa automaattisesti yhdistettäessä puhelin auton tietoviihdejärjestelmään. Toinen keskeinen toiminto on musiikin tai audiosisällön hallintapainikkeet, jotka myös näkyvät automaattisesti käyttöliittymässä jaetun ruudun tilassa Google Mapsin rinnalla (kuva 3).



Kuva 3. Google Maps ja mediasovelluksen soitin Android Auton jaetun näytön näkymässä (11).

Lisäksi Android Auton käyttöliittymään kuuluu yksinkertainen sovellusvalikko, jossa näkyvät kaikki puhelimeen asennetut Android Autoa tukevat sovellukset. Alareunan palkissa on myös viimeksi käytettyjen sovellusten pikanäppäimet, Google Assistant -puhekomentopainike sekä ilmoitusvalikon avaava oikean alareunan tilanäkymä.

Android Autoa hallitaan siis kosketusnäytöllä hyvin samaan tapaan kuin älypuhelinta. Kosketusalueiden ja painikkeiden koko on kuitenkin suurempi ajon aikana operoimisen helpottamiseksi. Lisäksi on mahdollista käyttää Google Assistant -puhekomentoja lähes kaikkiin toimintoihin mediantoiston hallinnasta ja sisältöjen hakemisesta puheluihin vastaamiseen. Google Assistant ei kuitenkaan ainakaan vielä tue suomen kieltä, mutta englannin kielelläkin komennot toimivat sujuvasti.

Kuten luvussa 2.3 mainittiin, ovat Android Auto -sovellukset rajoitetumpia toiminnoiltaan kuin niiden täysversiot. Tämä tarkoittaa esimerkiksi sitä, että kaikkia mahdollisia asioita ei pysty tekemään Auto-sovelluksella. Olennaisimmat

ominaisuudet ovat kuitenkin käytettävissä. Lisäksi järjestelmä keskeyttää käyttäjän operoinnin kosketusnäytöllä vähäksi aikaa, jos käyttöliittymän listoja vierittää liian kauan. Nämä rajoitukset ovat hyvä asia liikenneturvallisuuden kannalta, ja Google ottaakin vakavasti turvallisuusasiat autojärjestelmissään (5).

4 Android-mediasovellukset

Tässä luvussa käsitellään mediasovelluksia ja niiden toimintoja Androidissa. Lisäksi tutustutaan mediasovellusten arkkitehtuuriin, teknisiin ominaisuuksiin ja eri käyttömahdollisuuksiin. Jotta Android Auto -toiminnallisuus voidaan toteuttaa, on ensin varmistettava, että mediasovelluksen arkkitehtuuri on oikeanlainen, ja että tietyt kehysympäristöt ovat käytössä sen teknisessä toteutuksessa.

4.1 Toiminnot

Android-mediasovellukset, eli sovellukset, jotka toistavat jonkintyyppistä digitaalista mediaa, voidaan jakaa video- ja audiosovelluksiin. Tässä työssä keskitytään audiomediasovelluksiin. Audiosovelluksia on monenlaisia musiikkisovelluksista äänikirjasovelluksiin.

Audiosovellusten keskeisimpiin ominaisuuksiin kuuluu soitinnäkymä, jossa on toiston hallintapainikkeet, kuten toista-, tauko-, seuraava ja edellinen kappale - ja eteen- ja taaksepäinkelauspainikkeet. Lisäksi sovelluksissa tulee olla mahdollisuus selata eri mediakohteita ja aloittaa niiden toisto.

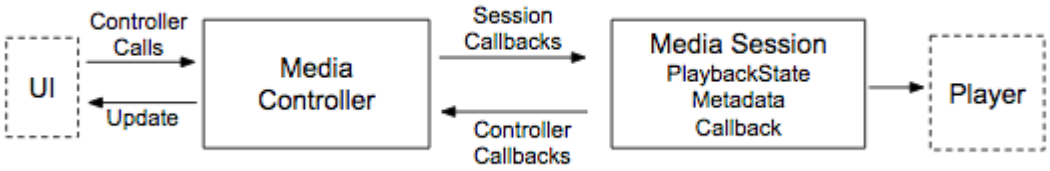
Androidissa toistettavan audion soitinnäkymä on myös lukitusnäytöllä ja pikaasetusvalikossa näkyvillä, ja sen avulla on helppoa kontrolloida audiontoistoa ilman itse sovelluksen avaamista.

4.2 Suositeltu arkkitehtuuri ja kirjastot

Androidin suositeltu mediasovellusarkkitehtuuri perustuu asiakasohjelma ja palvelin -malliin. Käytännössä se tarkoittaa, että soitin vastaanottaa ja soittaa

digitaalista mediaa ja käyttöliittymän kautta kontrolloidaan soitinta ja näytetään sen tila. (12.)

Androidin media-kehysympäristö tarjoaa tähän arkkitehtuuriin kuuluvat ohjelmointirajapinnat ja luokat, joilla jokaisella on oma paikkansa ja tehtävänsä mediasovelluksessa. Keskeisimmät luokat ovat MediaSession ja MediaController, jotka toimivat käyttöliittymän ja soittimen välissä ja mahdollistavat niin kontrollien toiminnan kuin reaaliaikaisen median datan esittämisen (kuva 4). (12.)



Kuva 4. Suositellun mediasovellusarkkitehtuurin toimintaperiaate. Käyttöliittymä (UI) kutsuu MediaControlleria, joka päivittää sen dataa. MediaController ja MediaSession kommunikoivat keskenään takaisinkutsujen avulla, ja MediaSession kutsuu soitinta. (12.)

ExoPlayer-mediasoitin

Android-mediasovelluksissa voidaan käyttää mediasoittimen toteuttamiseen joko MediaPlayer-luokkaa, joka tarjoaa yksinkertaiset perustoiminnot, tai ExoPlayer-kirjastoa, jossa on enemmän muokkausmahdollisuuksia ja laajempi tuki eri toiminnoille ja formaateille. ExoPlayer on avoimen lähdekoodin kirjasto, joka on rakennettu Androidin matalan tason media-kehysympäristöille. Nykyään ExoPlayer kuuluu androidx.media3-kirjastoon, jota käsitellään luvussa 4.3. Google suosittelee ExoPlayerin käyttämistä Android-mediasovelluksissa, ja se on varsin suosittu ja laajasti käytössä oleva mediasoitinratkaisu. (12.)

MediaSession

MediaSession-luokka vastaa kaikesta kommunikoinnista soittimen, kuten ExoPlayerin, kanssa. Soitinta kutsutaan vain MediaSessionista, joka kontrolloi sitä.

MediaSession pitää yllä soittimen tilaa, johon kuuluu tieto siitä, toistaako soitin sisältöä vai onko toisto tauolla, toiston aikaleima ja monenlaista tietoa sisällöstä, jota toistetaan.

MediaSession vastaanottaa takaisinkutsuja (callback) MediaControllereilta, joita voi olla useampi kuin yksi. Tämä mahdollistaa myös sen, että Android Auto voi kontrolloida mediantoistoa sen ollessa yksi MediaController. (12.)

MediaController

MediaController on yhteydessä käyttöliittymään. Käyttöliittymä kommunikoi vain MediaControllerin kanssa, ei suoraan soittimen kanssa. MediaController muuntaa niin kutsutut transport-kontrollit, kuten toista ja tauko, takaisinkutsuiksi MediaSessionille. Lisäksi MediaController vastaanottaa MediaSessionilta takaisinkutsuja, kun sen tila muuttuu, minkä avulla käyttöliittymässä voidaan näyttää ajantasaista dataa median toistosta. MediaController voi olla yhteydessä vain yhteen MediaSessioniin kerrallaan. (12.)

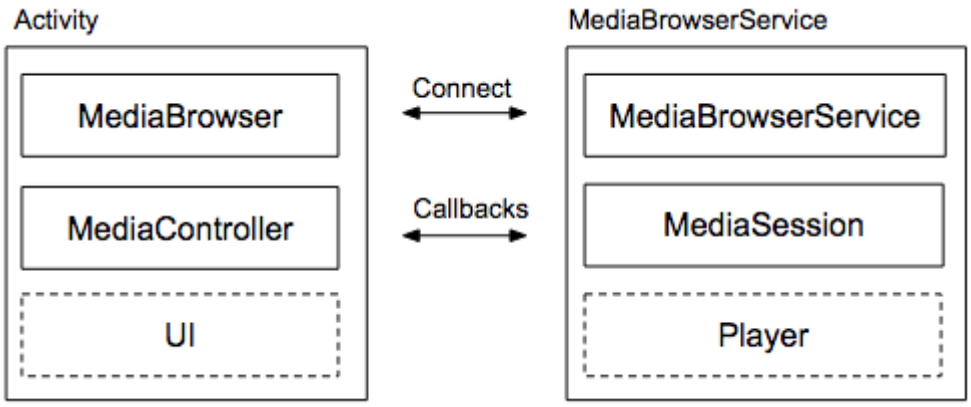
MediaBrowserService ja MediaBrowser

Toisin kuin videontoistosovelluksissa, on audiontoistosovelluksissa mahdollisuus jatkaa toistoa taustalla ilman, että sovelluksen käyttöliittymä on koko ajan esillä. Käyttäjä pystyy siis samaan aikaan käyttämään muita sovelluksia audion soidessa taustalla.

Androidissa tämä toteutetaan siten, että aktiviteetti (activity), joka sisältää käyttöliittymän, ja palvelu (service), joka on itse mediasoitin, ovat erillisiä komponentteja, jotka toimivat itsenäisesti riippumattomina toisistaan. Androidin media-API:t tarjoavat tämän arkkitehtuurin toteuttamiseen kaksi luokkaa: MediaBrowserService ja MediaBrowser.

Palvelu toteutetaan MediaBrowserServicen perivän luokan avulla, joka sisältää MediaSessionin ja sen soittimen. Käyttöliittymäaktiviteetissa taas on MediaBrowser, joka kommunikoi MediaBrowserServicen kanssa (kuva 5).

MediaBrowserServicen avulla monet erilaiset lisälaitteet, kuten Google Wear OS -älykellot ja Android Auto -järjestelmä, voivat löytää mediasovelluksen, kontrolloida toistoa ja saada pääsyn sen tarjoamaan selattavaan mediasisältöön. Sovelluksen tavallista käyttöliittymäaktiviteettia ei siis tarvita ollenkaan näissä käyttötapauksissa, ja palveluun voi olla yhteydessä useampi asiakassovellus yhtä aikaa. (12; 13.)



Kuva 5. MediaBrowserServiceen ja sovelluksen aktiviteettiin kuuluvat mediasovelluksen komponentit. MediaBrowser yhdistää MediaBrowserServiceen ja MediaController kommunikoi MediaSessionin kanssa takaisinkutsujen avulla. (12.)

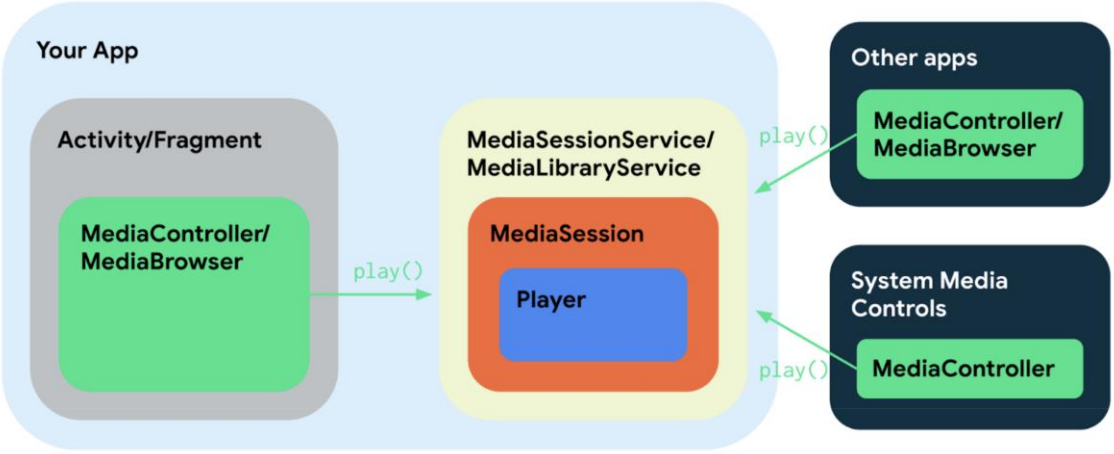
Media-compat-kirjasto

Android-mediasovelluksissa suositellaan käyttämään media-compat-tukikirjaston MediaSessionCompat- ja MediaControllerCompat-luokkia, joiden avulla saadaan tuki myös vanhemmille Android-versioille (12).

4.3 Media3-kirjasto

Jo useamman vuoden käytössä ollut yleinen ja suositeltu tapa toteuttaa mediasovellus on käyttää luvussa 4.2 mainittuja Androidin media-kehysympäristöjä ExoPlayer-kirjaston kanssa. Hiljattain on kuitenkin julkaistu uusi, entistä parempi Jetpack Media3 -kirjasto, joka on ollut jonkin aikaa beetavaiheessa, ja on nyt uusi suositeltu Android-mediakirjasto.

Media3-kirjasto kokoaa yhteen kaikki tarvittavat Androidin mediakirjastot mukaan lukien ExoPlayerin ja tekee arkkitehtuurista yksinkertaisemman, mikä helpottaa sovellusten kirjoittamista ja ylläpitämistä. Media3:n komponentit ovat pitkälti samat kuin aiempien Androidin media-API:en, mutta joitain muutoksia on nimeämisissä ja ominaisuuksissa. Niihin kuuluvat Player, MediaSession, MediaSessionService/MediaLibraryService, MediaController ja MediaBrowser (kuva 6). (14.)



Kuva 6. Media3-kirjaston eri komponentit ja niiden yhteydet. Eri MediaControllerit lähettävät komentoja MediaSessionServicen tai MediaLibraryServicen kautta MediaSessionille, joka välittää ne edelleen Playerille. Saman tai muiden sovellusten MediaBrowserit löytävät mediasisältöä MediaLibraryServicen kautta. (14.)

Keskeisin Media3:n komponentti on MediaLibraryService. Tämä luokka pitää sisällään MediaSessionin ja Playerin ja mahdollistaa soittimen toiminnan taustalla palveluna. MediaLibraryServicessä on myös MediaLibrarySession.Callback-objekti, joka sisältää erilaisia takaisinkutsufunktioita, joiden avulla voidaan muun muassa aloittaa kohteiden toisto, määrittää soitinkomentoja ja tarjota mediasisältökirjasto asiakassovelluksille. (14.)

Yksi Media3:n merkittävimmistä muutoksista on se, että siinä ei tarvita enää komponenttien välille yhdistäjäluokkia. Esimerkiksi MediaSessionille voidaan nyt suoraan antaa player-objekti, kuten ExoPlayer, mikä yksinkertaistaa arkkitehtuuria (esimerkkikoodi 1). (14; 15.)

```
val player = ExoPlayer.builder(context).build()
val session = MediaSession.Builder(context, player).build()
val controller = MediaController.Builder(
     context,
     session.token
).build()
```

Esimerkkikoodi 1. Playerin, MediaSessionin ja MediaControllerin alustaminen Media3:ssa on varsin yksinkertaista. Player-instanssi annetaan parametrina MediaSessionin Builder-funktiolle ja MediaControllerin Builder-funktiolle annetaan MediaSessionin token-avain (16).

Media3:n täysi julkaisu tapahtui maaliskuussa 2023, ja Google suosittelee kehittäjien ottavan sen käyttöön lähitulevaisuudessa, kun vanhempien media-kirjastojen kehitys vähitellen loppuu (16).

4.4 Käyttömahdollisuudet

Androidille pystyy kehittämään monenlaisia mediasovelluksia, ja käyttömahdollisuudet ovat laajat. Kun mobiilisovellukseen kehittää mediantoistotoiminnallisuuden noudattaen suositeltua arkkitehtuuria, pystyy samaa koodia käyttämään helposti myös Google Wear OS -älykellon ja Android Auton mediantoiston hallinnan mahdollistamiseen.

5 Android Auto -tuen toteuttaminen

Tässä luvussa kuvataan Supla-sovelluksen Android Auto -tuen kehittämisen teknistä toteutusta ja havainnollistetaan Android Auto -mediasovelluksen toimintaperiaatetta käytännössä.

5.1 Sovelluksen lähtötilanne

Supla-sovellukseen oli juuri tehty suuri mediasoittimen uudistus, kun opinnäytetyötä aloitettiin tekemään. Sovelluksessa otettiin käyttöön luvussa 4.2 esitelty Androidin suositeltu mediasovellusarkkitehtuuri aiemman mukautetun soitinratkaisun sijaan. Uusi soitin on huomattavasti kevyempi ja koodipohjaltaan selkeämpi kuin vanha.

Myös Android Auto -tuen lisäämisen kannalta soittimen uudistus oli välttämätön ja tarpeellinen, koska mediasovellustyyppinen Android Auto -toteutus perustuu tähän moderniin Android-mediasovellusarkkitehtuuriin.

Vähän Android Auto -kehityksen aloituksen jälkeen siirryttiin sovelluksessa vielä käyttämään uutta Media3-kirjastoa, mikä entisestään yksinkertaisti koodia ja varmisti toteutuksen ajantasaisuuden myös pitkälle tulevaisuuteen. Android Auto -kehitys siirtyi siis Media3-toteutuksen päälle, mikä vaati vain pieniä muutoksia koodiin.

5.2 Android Auto -mediasovelluksen toimintaperiaate

Android Auto -tuen lisääminen olemassa olevaan mediasovellukseen vaatii, että sovellus on Androidin mediasovellusarkkitehtuurin mukainen. Avainkomponentti on MediaBrowserService tai Media3:n MediaLibraryService, jonka avulla Android Auto ja muut sovellukset, joissa on MediaBrowser, voivat löytää, esittää ja toistaa sovelluksen mediasisältöä. Ennen Android Auto -tuen lisäämistä kannattaakin varmistaa, että mediasovellus toimii mobiililaitteella kunnolla. (17.)

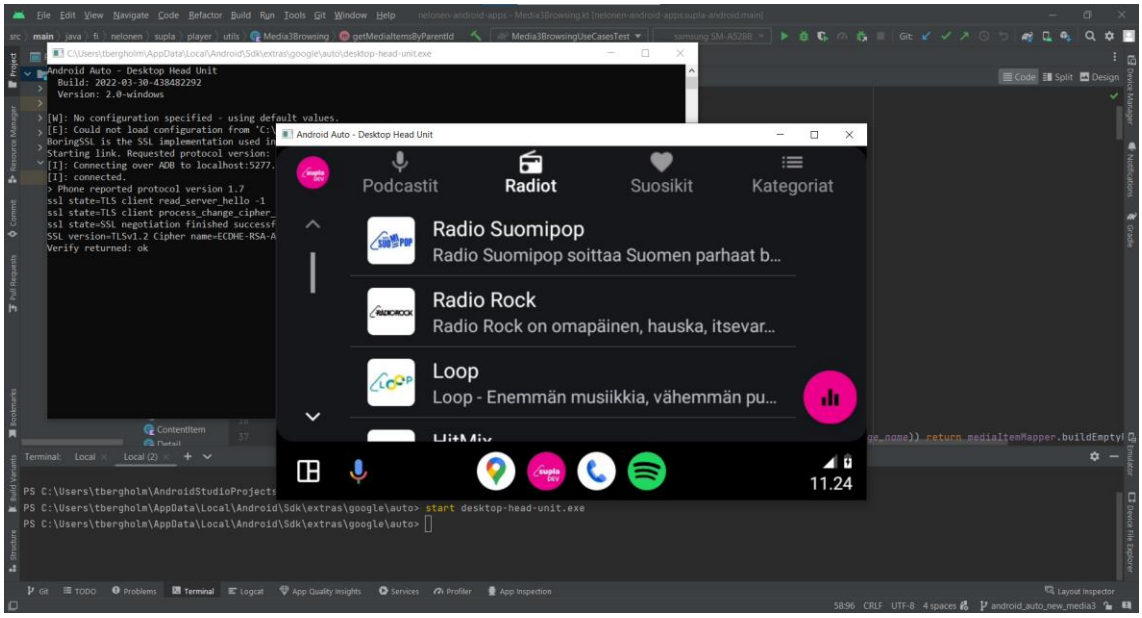
Käytännössä sama mobiilisovellus toimii Android Auto -toiminnon taustalla, eikä Android Autoa varten tehdä omaa versiota tai APK-pakettia (Android Package Kit) sovelluksesta. Kaikki Android Auton ominaisuudet käyttöliittymästä mediantoistoon toteutetaan Androidin media-API:en kautta. Käyttöliittymää ei siis toteuteta ollenkaan itse, vaan Android Auto -järjestelmä piirtää sen.

5.3 Kehitystyökalut, laitteisto ja testaus

Android Auto -tuen kehittämiseen tarvitaan Android Studio -kehitysympäristön lisäksi puhelin, jonka käyttöjärjestelmäversio on vähintään Android 6, ja Android Auto Desktop Head Unit (DHU) -emulaattori (18). Kehitykseen ja testaukseen ei siis tarvita oikeaa auton mediakeskusyksikköä (head unit) saati Android Auto -tuen sisältävää ajoneuvoa.

Puhelin yhdistetään työasemaan ja aktivoidaan puhelimen Android Auto -järjestelmäsovelluksen kehitystila, josta käynnistetään pääyksikön palvelin. Sitten työasemalla muodostetaan yhteys puhelimeen ADB:n (Android Debug Bridge) kautta luomalla TCP (Transmission Control Protocol) -porttiyhteys, minkä jälkeen voidaan käynnistää työasemalla ajettava DHU-emulaattori. Puhelin on yhteydessä emulaattoriin samalla tavalla, kuin se olisi oikeassa käyttötilanteessa yhteydessä auton Android Auto -pääyksikköön (kuva 7).

Puhelimeen voidaan asentaa tavalliseen tapaan uusia kehitysversioita sovelluksesta ja muutokset näkyvät heti myös DHU-emulaattorilla Android Auto -sovelluksessa, mikä tekee testaamisesta nopeaa ja vaivatonta.



Kuva 7. Android Studio -kehitysympäristö ja Android Auto Desktop Head Unit -emulaattori käynnissä (19).

5.4 Alkuvalmistelut

Kuten luvussa 5.1 mainittiin, Supla-sovelluksessa oli media-arkkitehtuuri jo valmiiksi kunnossa, joten varsinaisen Android Auto -tuen kehittäminen voitiin aloittaa saman tien.

Ensimmäiset vaiheet ovat käynnistys- ja attribuuttikuvakkeiden sekä teeman määrittäminen AndroidManifest.xml-tiedostoon. Lisäksi varsinainen Android Auto -tuki lisätään AndroidManifestiin viittaamalla automotive_app_desc.xml-tiedostoon (esimerkkikoodi 2) (20).

```
<meta-data android:name="com.google.android.gms.car.application"
     android:resource="@xml/automotive_app_desc"/>
<meta-data android:name="com.google.android.gms.car.application.theme"
     android:resource="@style/AutoTheme" />
<meta-data
     android:name="androidx.car.app.TintableAttributionIcon"
     android:resource="@drawable/supla_notification_logo" />
```

Esimerkkikoodi 2. Android Auto -tuen sekä teeman ja attribuuttikuvakkeen määrittäminen AndroidManifest-tiedostossa.

Tiedostossa automotive_app_desc.xml määritetään Auto-sovelluksen tyyppi, joka on tässä tapauksessa ”media” (esimerkkikoodi 3).

```
<automotiveApp>
     <uses name="media"/>
</automotiveApp>
```

Esimerkkikoodi 3. Android Auto -sovelluksen tyypin määrittäminen automotive_app_desc.xml-tiedostossa.

5.5 MediaLibraryService ja sisältöhierarkia

MediaLibraryService, tai MediaBrowserService riippuen käytössä olevasta kirjastosta, on olennainen osa Android Auto -toteutusta. Android Auto -järjestelmä kommunikoi sovelluksen kanssa Servicen kautta ja löytää sen kautta mediasisällön, jonka sovellus tarjoaa.

Android Auto -mediasovelluksen käyttöliittymän ja sisältöjen näyttämisen toteuttaminen perustuu MediaItem-tyyppisiin objekteihin, joista muodostetaan sisältöhierarkia. MediaItemillä on useita ominaisuuksia, joihin palataan myöhemmin, mutta yksi olennaisimpia on mediaId.

MediaLibraryServicessä on kaksi takaisinkutsufunktiota, joiden avulla sisältöhierarkia muodostetaan: onGetLibraryRoot ja onGetChildren (onGetRoot ja

onLoadChildren MediaBrowserServicessä). OnGetLibraryRoot-funktiossa palautetaan mediasisältöhierarkian juurikohde. OnGetChildren-funktiossa taas palautetaan parentId-parametrin perusteella lista MediaItemeitä, jotka ovat emokohteen (parent item) lapsikohteita (child item).

OnGetLibraryRoot-funktiota kutsutaan vain kerran aluksi, kun juurikohde haetaan, mutta onGetChildren-funktiota kutsutaan useita kertoja hierarkian mennessä syvemmälle, jolloin parentId-parametri vaihtuu sen mukaan, mikä on kyseisen kohteen mediaId, jonka lapsikohteita haetaan. Funktiossa palautetaan siis parentId:n perusteella eri MediaItem-listoja.

Sovelluksessa käytetyn Media3:n MediaLibraryServicen perivän SuplaPlayerService-luokan MediaLibrarySession.Callback-objektin onGetLibraryRoot- ja onGetChildren-funktiot käyttävät ListenableFuture-luokkaa palautusarvossa, mikä mahdollistaa asynkronisten sisältöhakujen suorittamisen. Guava-kirjasto, joka on varsin laaja ja suosittu Java-kirjasto, sisältää muun muassa Future-luokasta laajennetun ListenableFuturen, jossa on mahdollisuus rekisteröidä takaisinkutsuja heti, kun laskentatehtävä on suoritettu (21). Kotlinin Coroutines-samanaikaisuussuunnittelumalli (concurrency design pattern) on mahdollista integroida Guavan ListenableFuturen kanssa kotlinx-coroutines-guava-kirjaston avulla. CoroutineScopen future-funktion koodilohkossa kutsutaan suspend-funktiota, joka palauttaa asynkronisesti arvon ja future-funktio palauttaa siitä automaattisesti ListenableFuture-tyyppisen objektin.

OnGetLibraryRoot-funktiossa haetaan Media3Browsing-luokan getRootMediaItem-funktiosta juuri-MediaItem (esimerkkikoodi 4). MediaItemiin asetetaan tässä tapauksessa vain ID ja metadata-objekti, mutta MediaItemilla on monia ominaisuuksia, joita käsitellään tarkemmin luvussa 5.6.3.

```
override fun onGetLibraryRoot(
     session: MediaLibrarySession,
     browser: MediaSession.ControllerInfo,
     params: LibraryParams?,
): ListenableFuture<LibraryResult<MediaItem>> {
     return Futures.immediateFuture(
          LibraryResult.ofItem(
               media3Browsing.getRootMediaItem(
                    packageValidator,
                    browser,
               ),
               params,
          ),
     )
}
```

Esimerkkikoodi 4. OnGetLibraryRoot-funktiossa palautetaan juuri-MediaItem LibraryResult-objektissa, joka on ListenableFuture-tyyppisessä objektissa.

OnGetChildren-funktiossa haetaan MediaItemit sovelluksen Media3Browsing-luokan getMediaItems-funktiosta, jolle annetaan parametrina parentId (esimerkkikoodi 5).

```
override fun onGetChildren(
     session: MediaLibrarySession,
     browser: MediaSession.ControllerInfo,
     parentId: String,
     page: Int,
     pageSize: Int,
     params: LibraryParams?,
): ListenableFuture<LibraryResult<ImmutableList<MediaItem>>> {
     return loadContentScope.future {
          LibraryResult.ofItemList(
               media3Browsing.getMediaItems(
                    parentId,
                    browser
               ),
               params,
          )
     }
}
```

Esimerkkikoodi 5. OnGetChildren-funktiossa haetaan asynkronisesti Coroutinen ja Guava-kirjaston future-funktion avulla MediaItem-listoja parentId:n perusteella.

5.6 Käyttöliittymän ja mediasisältöjen luominen

Android Auto -mediasovelluksen käyttöliittymä rakentuu MediaItem-listoista, jotka muodostavat sisältöhierarkian. Android Auto -järjestelmä piirtää käyttöliittymän MediaItemien ja niiden ominaisuuksien mukaan.

Juuri-MediaItemin lapsikohteita ovat päävälilehtikohteet, jotka ovat ”Podcastit”-, ”Radiot”-, ”Suosikit”- ja ”Kategoriat”-sivut. Nämä MediaItemit palautetaan, kun Android Auto -järjestelmä kutsuu onGetChildren-takaisinkutsufunktiota selattavan sisällön juuri-ID:llä. Samalla tavalla esimerkiksi ”Podcastit”-välilehtikohteen ID:llä funktiota kutsuttaessa palautetaan sille kuuluvat lapsikohteet, ja niin edelleen.

Media3Browsing-luokan getMediaItems-funktiossa päätellään parentId:n perustella, minkä sivun MediaItemeitä haetaan. Jos parentId on jokin muu kuin ennalta määritelty sivun tai näkymän ID, kuten podcast-sarjan tai kategorian ID, katsotaan MediaItemin tyyppi ID-merkkijonon alkuun lisätystä tyypistä ja sen mukaan haetaan ja palautetaan oikeanlaiset lapsikohteet. (Esimerkkikoodi 6.)

```
suspend fun getMediaItems(parentId: String): List<MediaItem> {
     return when (parentId) {
          applicationContext.getString(R.string.empty_media_root_id),
          applicationContext.getString(R.string.text_display_me-
dia_item_id),
          -> emptyList()
          applicationContext.getString(R.string.media_root_id)
          -> getMainTabItems()
          applicationContext.getString(R.string.podcasts_tab_id)
          -> getPodcastsTabItems()
          applicationContext.getString(R.string.radios_tab_id)
          -> getRadiosTabItems()
          applicationContext.getString(R.string.favorites_tab_id)
          -> getFavoritesTabItems()
          applicationContext.getString(R.string.categories_tab_id)
          -> getCategoriesTabItems()
          else -> {
               val itemType = parentId.split(";").firstOrNull()
               val itemId = parentId.split(";").lastOrNull()
               if (itemType == null || itemId == null)
                    return emptyList()

               when (itemType) {
                    BrowsableMediaItemType.PodcastSeries.toString()
                    -> getPodcastEpisodeItems(itemId)
                    BrowsableMediaItemType.Category.toString()
                    -> getItemsOfCategory(itemId)
                    else -> emptyList()
               }
          }
     }
}
```

Esimerkkikoodi 6. GetMediaItems-funktiossa päätellään parentId:n perusteella, mistä funktiosta MediaItemeitä haetaan.

Vaikka alustavasti suunniteltiin toteutettavan vain ”Podcastit”-, ”Radiot”- ja ”Suosikit”-pääsivut, päätettiin kehityksen myöhemmässä vaiheessa lisätä vielä ”Kategoriat”-sivu, jossa listataan eri podcast-kategorioita.

5.6.1 Pakettivalidointi

Sovelluspaketin validointi tehdään, ennen kuin palautetaan Android Autolle tarkoitetun MediaItem-sisällön juurikohdetta. Koska mikä tahansa muu sovellus pystyisi pääsemään käsiksi mediasisältöön, jota sovellus tarjoaa MediaLibraryServicen kautta, halutaan rajoittaa, mitkä sovellukset tai järjestelmät voivat löytää sisällön.

Pakettivalidointi toteutettiin Googlen esimerkkinä tarjoaman PackageValidator-luokan avulla (22). Sovelluksen MediaBrowserPackageValidator-luokka tarkistaa Serviceen yhteyden ottavan sovelluksen tai järjestelmän sovelluspakettinimen (package name) ja sovelluspaketin UID:n (user ID) perusteella, onko kutsuja tunnettu. Sallitut sovellukset pakettinimineen ja signeerauksineen (signature), joihin kuuluu muun muassa Android Auto, on listattu xml-resurssitiedostossa. MediaBrowserPackageValidator käyttää tätä tiedostoa tarkistuksessa ja pitää kirjaa jo tarkistetuista sallituista sovelluspaketeista, jolloin varsinaista tarkistusoperaatiota ei tarvitse tehdä toistamiseen sovelluksen ajon aikana. (Esimerkkikoodi 7.)

MediaBrowserPackageValidatorin isKnownCaller-funktiota kutsutaan Media3Browsing-luokan getRootMediaItem-funktiossa. Jos paketti ei ole sallittu, palautetaan tyhjä juuri-MediaItem, mutta jos paketti on Android Auton, palautetaan sisältöhierarkiaan johtava juuri-MediaItem.

```
val isCallerKnown = when {
     // If it's our own app making the call, allow it.
     callingUid == Process.myUid() -> true
     // If it's one of the apps on the allow list, allow it.
     isPackageInAllowList -> true
     // If the system is making the call, allow it.
     callingUid == Process.SYSTEM_UID -> true
     // If the app was signed by the same certificate as the platform
itself, also allow it.
     callerSignature == platformSignature -> true

     callerPackageInfo.permissions.contains(
          BIND_NOTIFICATION_LISTENER_SERVICE
     ) -> true
     // If none of the previous checks succeeded, then the caller is
unrecognized.
     else -> false
}

if (!isCallerKnown) {
     logUnknownCaller(callerPackageInfo)
}

// Save our work for next time.
callerChecked[callingPackage] = Pair(callingUid, isCallerKnown)
return isCallerKnown
```

Esimerkkikoodi 7. MediaBrowserPackageValidatorissa tarkistetaan, onko kutsuva sovelluspaketti tunnettu.

5.6.2 Component API -taustajärjestelmä

Supla-sovellus käyttää sivujen ja niiden komponenttien ja mediasisältöjen hakemiseen Component API -taustajärjestelmää. Component API -rajapintaan tehdään verkkokutsu tietyllä sivun tai komponentin ID:llä, jolloin se palauttaa JSON-muotoisen vastauksen, joka sisältää monenlaista dataa, kuten components- ja items-kohteet. Items-lista sisältää CapiItem-tyyppisiä kohteita, joilla on useita metatietoja, joita käytetään kohteiden näyttämiseen käyttöliittymässä.

Myös Android Auto -toteutus käyttää Component API:a mediasisältöjen hakemiseen. Media3BrowsingUseCases-luokassa on funktioita, jotka hakevat halutun sivun CapiItemit ja rakentavat niistä MediaItemit (esimerkkikoodi 8). Media3Browsing-luokan päävälilehtikohteiden lapsikohteiden hakemiseen käytetyt funktiot kutsuvat Media3BrowsingUseCases-funktioita MediaItem-listojen hakemiseen.

```
suspend fun getPodcastsPageMediaItems(): List<MediaItem> {
     return getMediaItems(PageId.AUTO_PODCASTS, itemLimit = 24)
}

suspend fun getRadiosPageMediaItems(): List<MediaItem> {
     return getMediaItems(PageId.AUTO_RADIOS)
}

suspend fun getFavoritesPageMediaItems(): List<MediaItem> {
     return getMediaItems(PageId.AUTO_FAVORITES)
}

private suspend fun getMediaItems(
     pageId: String,
     params: Map<String, String> = emptyMap(),
     itemLimit: Int = 40,
): List<MediaItem> {
     val components = componentApiRepository.getCapiV2PageSuspend(
          pageId,
          PageType.page,
          params,
     ).components

     return components.map { component ->
          componentApiRepository.getCapiV2ComponentSuspend(
               component.content.query.url,
               component.content.query.params,
               itemLimit,
          ).content.items.map {
               mediaItemMapper.buildMediaItemFromItem(
                    it as Item,
                    component.label?.text,
               )
          }
     }.flatten()
}
```

Esimerkkikoodi 8. Media3BrowsingUseCases-funktiot hakevat ennalta määritellyllä sivun ID:llä Component API:sta CapiItemeitä ja muuttavat ne MediaItemeiksi MediaItemMapper-luokan avulla.

Vaikka MediaLibraryServicen onGetChildren-funktio tukeekin sivutusta (paging) eli kohteiden lataamista pienemmissä osissa, ei Android Auto -järjestelmä tue sitä. Sen sijaan tietyn näkymän kaikki MediaItemit täytyy ladata ja palauttaa kerralla. Tämän takia Component API:sta haettavien kohteiden määrä rajoitettiin useimmilla sivuilla 40:een, mikä ei ole kuitenkaan ongelma, koska Android Auto -sovelluksessa ei ole tarkoituskaan selata näkymiä pitkiä aikoja.

5.6.3 MediaItemien rakentaminen

MediaItem-luokka on keskeinen niin Android Auton käyttöliittymän kuin koko Media3:n mediantoiston kannalta. MediaItemit voidaan jakaa toistettaviin (playable) ja ei-toistettaviin eli selattaviin (non-playbale, browsable) kohteisiin. Jos MediaItem on ei-toistettava, yrittää Android Auto -järjestelmä hakea kyseisen kohteen ID:llä sen lapsikohteita sitä klikattaessa käyttöliittymässä. Tällöin MediaItem on siis selattava ja sillä oletetaan olevan lapsikohteita. Jos MediaItem on toistettava, aloitetaan sen toisto, mitä käsitellään luvussa 5.7.1.

Media3:n MediaItem- ja Androidin vanhemman media-kehysympäristön MediaItem-luokat ovat lähes samanlaiset, mutta niissä on pieniä eroja muuttujien nimeämisissä ja rakenteessa. Tässä työssä käsitellään Media3:n MediaItemiä.

MediaItem-objektin rakentamisessa kaksi keskeisintä muuttujaa on mediaId ja mediaMetadata. MediaMetadata-objektin tässä tapauksessa tärkeimpiä muuttujia ovat title, subtitle, description, artworkUri, extras, isPlayable ja folderType. MediaItemeja luodaan luokan Builder-funktiolla (esimerkkikoodi 9). Yksinkertaisimmillaan MediaItem rakennetaan asettamalla mediaId ja mediaMetadata, johon on asetettu isPlayable-Boolen arvo ja folderType, joka määrittää, minkä tyyppistä sisältöä kohde sisältää. Esimerkiksi juuri-MediaItem rakennetaan näillä muuttujilla.

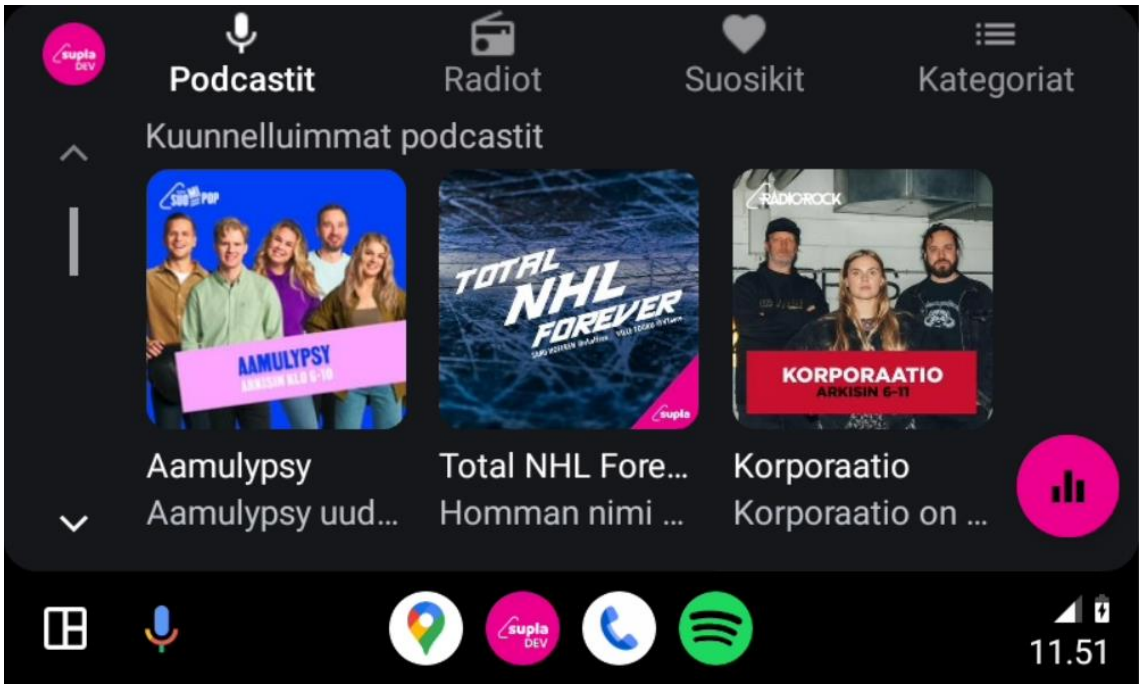
Media3BrowsingUseCases-funktiot käyttävät MediaItemMapper-luokan buildMediaItem-funktiota MediaItemien rakentamiseen CapiItemien datan pohjalta.

```
val mediaMetadata = MediaMetadata.Builder()
     .setTitle(item.title ?: item.link?.label ?: "")
     .setSubtitle(item.subtitle ?: "")
     .setDescription(item.description ?: "")
     .setArtworkUri(Uri.parse(imageUrl).asMediaItemImageContentURI())
     .setExtras(mediaMetadataExtras)
     .setIsPlayable(false)
     .setFolderType(FOLDER_TYPE_MIXED)
     .build()

val mediaItem = MediaItem.Builder()
     .setMediaId(item.id)
     .setMediaMetadata(mediaMetadata)
     .build()
```

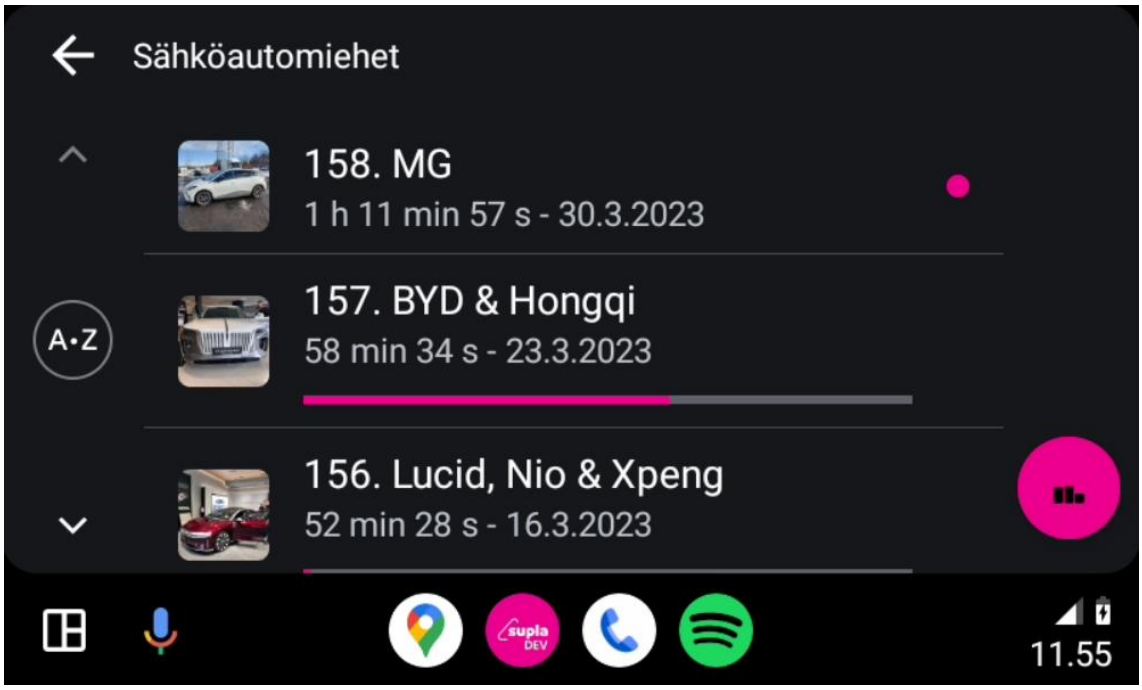
Esimerkkikoodi 9. MediaItemin rakentaminen.

MediaMetadata-objektin Bundle-tyyppiseen extras-muuttujaan voidaan lisätä erilaisia arvoja. Extroihin lisätään muun muassa mediasisällön tyyppi eli ToBePlayedItem.ItemType, joka on podcast tai radio. Lisäksi kohteiden esittämistapaa käyttöliittymässä määritetään extroissa. MediaConstants.EXTRAS_KEY_CONTENT_STYLE_SINGLE_ITEM-avaimen arvoilla MediaConstants.EXTRAS_VALUE_CONTENT_STYLE_LIST_ITEM ja MediaConstants.EXTRAS_VALUE_CONTENT_STYLE_GRID_ITEM määritetään, näytetäänkö kyseinen kohde käyttöliittymässä lista- vai ruudukkotyyppisessä asettelussa (kuva 8). Myös ryhmäotsikko voidaan asettaa extroihin, minkä avulla Android Auton käyttöliittymä osaa ryhmitellä MediaItemeitä, joilla on sama ryhmäotsikko.



Kuva 8. Podcastit-sivu, jossa mediakohteita ruudukkotyyppisessä asettelussa ryhmäotsikoituna.

Android Auton käyttöliittymässä voidaan näyttää myös MediaItemien kuunteluedistyksen palkkeja (progress bar) ja kuuntelemattomien ja kuunneltujen kohteiden indikaattoreita. Tämä toteutettiin hakemalla sovelluksen kuunteluhistoriakuvauskannasta podcast-jaksojen kuunteluedistysdata ja asettamalla sen mukaan MediaMetadata-objektin extras-muuttujaan MediaConstants.EXTRAS_KEY_COMPLETION_STATUS-avaimella loppuun toistetun, osittain toistetun tai ei-toistetun arvo. Lisäksi MediaConstants.EXTRAS_KEY_COMPLETION_PERCENTAGE-avaimella asetetaan kuunteluprosentti. Näiden tietojen perusteella Android Auto -järjestelmä osaa näyttää MediaItemien kuunteluedistyksen sekä päivittää toistettavan kohteen edistyspalkkia automaattisesti reaaliajassa (kuva 9).



Kuva 9. Podcastin jaksot -näkymä, jossa mediakohteita listatyyppisessä asettelussa. Kuunteluedistyspalkit ja kuuntelemattomien kohteiden indikaattorit näytetään tässä näkymässä.

5.6.4 Kuvien lataaminen

Kun MediaItem-objekteja rakennetaan, voidaan niille antaa artworkUri-objekti, joka mahdollistaa kansikuvien näyttämisen käyttöliittymässä. Googlen Android Auto -mediasovellusten ohjeistuksessa suositellaan käyttämään ContentProvider-luokkaa kuvien hakemiseen ja välimuistiin lataamiseen (17). Kuvien lataaminen laitteelle sovelluksen tiedostoihin nopeuttaa käyttöliittymän kuvien latautumista, kun niitä ei tarvitse ladata verkosta toistamiseen.

Sovelluksessa käytetään kuvien lataamiseen MediaItemImageProvider-luokkaa, jonka mapUri-funktiolle annetaan parametrina CapiItemin kuvan verkko-osoitteesta (URL) parsittu Uri-objekti. Uri:sta rakennetaan paikallinen Uri, ja ne tallennetaan Map-objektiin. (Esimerkkikoodi 10.)

```
class MediaItemImageProvider : ContentProvider() {

     companion object {
          private val uriMap = mutableMapOf<Uri, Uri>()

          fun mapUri(uri: Uri): Uri {
               val path = uri.encodedPath?.substring(1)?.replace('/',
':') ?: return Uri.EMPTY
               val contentUri = Uri.Builder()
                    .scheme(ContentResolver.SCHEME_CONTENT)
                    .authority(uri.authority)
                    .path(path)
                    .build()
               uriMap[contentUri] = uri
               return contentUri
          }
     }
```

Esimerkkikoodi 10. MediaItemImageProviderin mapUri-funktio.

Kun Android Auto -järjestelmä aloittaa MediaItemin kuvan lataamisen, kutsutaan MediaItemImageProviderin openFile-takaisinkutsufunktiota, joka lataa kuvan verkosta Glide-kirjaston avulla, jos sitä ei ole vielä ladattu, ja tallentaa sen File-tyyppisenä tiedostona. Funktio palauttaa paikallisen kuvatiedoston ParcelFileDescriptor-tyyppisenä objektina. (Esimerkkikoodi 11.)

```
override fun openFile(uri: Uri, mode: String): ParcelFileDescriptor? {
     val context = this.context ?: return null
     val remoteUri = uriMap[uri] ?:
          throw FileNotFoundException(uri.path)

     var file = uri.path?.let { File(context.cacheDir, it) }

     if (file != null && !file.exists()) {
          val cacheFile = Glide.with(context)
               .asFile()
               .load(remoteUri)
               .submit()
               .get(DOWNLOAD_TIMEOUT_SECONDS, TimeUnit.SECONDS)

          cacheFile.renameTo(file)

          file = cacheFile
     }
     return ParcelFileDescriptor.open(
          file,
          ParcelFileDescriptor.MODE_READ_ONLY
     )
}
```

Esimerkkikoodi 11. OpenFile-funktio, jossa kuva ladataan verkosta, tallennetaan tiedostona laitteelle ja avataan se.

Kuvien lataamisen toteuttamisessa oli aluksi jonkin verran ongelmia. Googlen dokumentaatiossa oli puutteita, eikä siellä oleva koodi toiminut suoraan. Googlen UAMP (Universal Android Music Player) -esimerkkisovelluksesta löytyi kuitenkin toimiva toteutus (22). Lisäksi AndroidManifestissa piti määrittää Provider ja authorities, eli verkko-osoitteet, joista kuvia haetaan. Tämän puuttuessa toteutus ei toiminut. Loppujen lopuksi kuvien lataaminen saatiin toimimaan hyvin ja kuvat latautuvat nopeasti.

5.6.5 Virheiden hallinta

Jos MediaItem-listojen hakeminen epäonnistuu, hallitaan virhetilanne näyttämällä virheviestin sisältävä tekstinäkymä käyttöliittymässä. Virhetilanne voi tulla, jos Component API ei vastaa tai se palauttaa jostain syystä vääränlaista dataa tai jos laitteen verkkoyhteys katoaa hetkellisesti.

Media3Browsing-luokan MediaItemeitä hakevissa funktioissa käytetään Kotlinin runCatching- ja getOrDefault -funktioita virheiden hallintaan. Jos virhe tapahtuu runCatching-koodilohkossa kutsuttavassa Media3BrowsingUseCases-luokan funktiossa, palautetaan getOrDefault-lohkossa text display -tyyppinen yksinkertainen MediaItem, jonka tekstinä on yleinen virheviesti. (Esimerkkikoodi 12.)

Virhetilan pystyisi myös asettamaan MediaSessionin PlaybackStateen, mutta koska se pysäyttäisi toistettavan kohteen toiston myös silloin, kun yhdenkin sivun lataus epäonnistuu, päädyttiin käyttämään virheviesti-MediaItem-ratkaisua mieluummin.

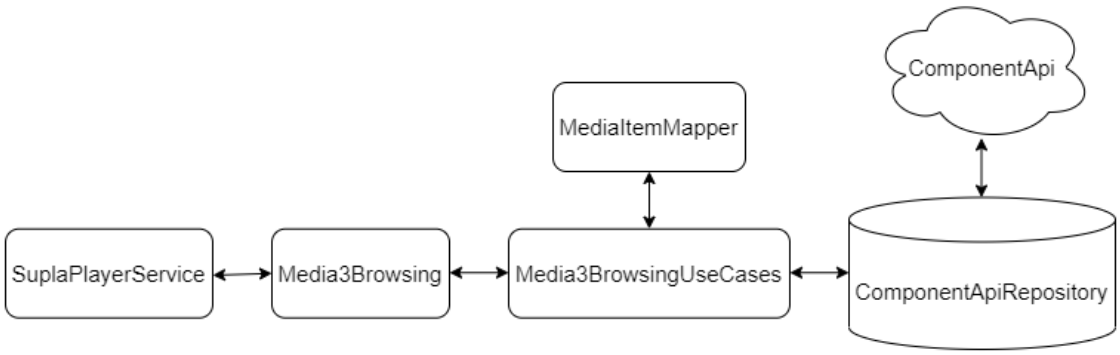
```
return runCatching {
     media3BrowsingUseCases.getPodcastsPageMediaItems()
}.getOrDefault(
     listOf(
          mediaItemMapper.buildTextDisplayMediaItem(
               applicationContext.getString(
                    R.string.loading_items_failed_message,
               ),
          ),
     ),
)
```

Esimerkkikoodi 12. Virheenhallinta runCatching-funktiolla MediaItemeita haettaessa.

5.6.6 Browsing-luokat ja arkkitehtuuri

Sovelluksessa noudatetaan Androidin Clean-arkkitehtuuria yhdessä MVVM-suunnittelumallin (Model-View-ViewModel) kanssa. Clean-arkkitehtuurissa sovellus koostuu eri tasoista, joiden koodi on erotettu toisistaan ja jotka ovat mahdollisimman vähän riippuvaisia toisistaan. Tasot ovat kokonaisuustaso (entity layer), käyttötapaustaso (use case layer), rajapinta-adapteritaso (interface adapter layer) ja kehysympäristö- ja ajuritaso (frameworks and drivers layer). Komponenttien välille muodostetaan rajoja, jolloin muutos yhdessä komponentissa ei vaikuta muihin komponentteihin. (23.)

Myös Android Auton ja MediaLibraryServicen selattavan sisällön hakemisessa ja luomisessa noudatetaan Clean-arkkitehtuuria. SuplaPlayerServicessä kutsutaan Media3Browsing-luokan funktioita, jotka puolestaan kutsuvat Media3BrowsingUseCases-luokan funktioita. Media3BrowsingUseCases-luokan funktioissa taas kutsutaan ComponentApiRepository- ja MediaItemMapper-luokkien funktioita, joiden avulla saadaan haettua CapiItemeitä ja rakennettua niistä MediaItemeitä. (Kuva 10.)



Kuva 10. Sovelluksen MediaLibraryServicen selattavan sisällön hakemiseen ja luomiseen liittyvät luokat ja arkkitehtuuri.

Media3BrowsingUseCases-luokalla on siis keskeinen rooli Android Auto -sovelluksessa käytettävien mediasisältöjen hakemisessa ja rakentamisessa. Luokka sisältää funktioita, jotka palauttavat erilaisia MediaItem-listoja. Media3BrowsingUseCases-funktioita käytetään vain SuplaPlayerServicen kautta esille tuotavan selattavan mediasisällön käyttötapauksissa, ei siis itse mobiilisovelluksen näkymien mediasisältöjen hakemisen tapauksissa.

5.7 Soitin ja toistaminen

Android Auto -mediasovelluksen toinen päätoiminto sisältöjen selaamisen ohella on niiden toistaminen ja toiston hallinta. Koska Android Auto, joka on käytännössä yksi MediaBrowser ja MediaController, käyttää sovelluksen SuplaPlayerServiceä soittimen toiston tilan (playback state) löytämiseen, toimi sisältöjen toistaminen saman tien ilman ylimääräisiä muutoksia.

5.7.1 Toiston aloitus

Kun käyttäjä klikkaa MediaItemiä, joka on tyypiltään toistettava (playable), kutsutaan SuplaPlayerServicen onAddMediaItems-takaisinkutsufunktiota. Funktio saa parametrina mediaItems-listan, joka sisältää MediaItemin, joka valittiin. Funktiossa kutsutaan SuplaPlayerManagerin funktiota getUpdateMediaItem, jossa tarkistetaan muun muassa, onko käyttäjällä oikeus sisältöön. Sitten

rakennetaan toistettavan tyypin mukainen uusi MediaItem ja asetetaan se onAddMediaItems-funktiosta palautettavaan ListenableFuture-objektiin.

Aluksi toiston aloituksessa ilmeni virhe, joka johtui siitä, että MediaItemin extra-metadatasta puuttui sisällön tyyppi. Tyypin lisääminen ratkaisi ongelman. Muuten itse toiston aloitus toimi sujuvasti.

5.7.2 Toistokontrollit

Soitinnäkymässä näytetään MediaItemin metatietoja, kuten nimike, kansikuva ja kesto (kuva 11). Android Auto -järjestelmä löytää tiedot MediaSessionin PlaybackStatesta. Lisäksi soittimessa on toistokontrollipainikkeet, kuten toista, tauko ja siirry seuraavaan ja edelliseen kohteeseen. Nämä peruskontrollit luodaan automaattisesti, mutta lisäksi voidaan luoda mukautettuja komentopainikkeita.

Sovelluksessa on 30 sekuntia taakse- ja eteenpäinkelauspainikkeet, jotka lisättiin asettamalla ne CommandButtoneina MediaLibrarySessionille setCustomLayout-funktiolla. Niiden toiminnot asetetaan onCustomCommand-takaisinkutsufunktiossa. Aluksi lisättiin myös toistonopeudenvaihtopainike, mutta se jätettiin pois, koska sitä olisi pitänyt napauttaa useita kertoja asettaakseen nopeuden halutuksi, mikä ei ole hyvä käyttökokemus ajaessa. Toistonopeuden pystyy kuitenkin vaihtamaan puhelimella etukäteen.

Kuva 11. Android Auton soitinnäkymä, jossa taakse- ja eteenpäinkelauskontrollit.

Kelauskontrollit piti piilottaa mainoksia ja suoria radiokanavia toistettaessa ja palauttaa takaisin näkyviin podcasteja toistettaessa. Lisäksi toistonopeus piti asettaa normaaliksi ja takaisin käyttäjän valitsemaksi näissä tapauksissa. Tämä aiheutti hieman haasteita, etenkin kun DHU-emulaattori ei aina päivittänyt kontrolleja oikein tai reagoinut komentoihin. Siihen auttoi emulaattorin uudelleenkäynnistys.

Kontrollien muokkaamiseen liittyvä dokumentaatio ainakin Media3:n osalta on heikkoa, ja kirjastosta puuttuukin tuki tietyille kontrollien muokkaamisille. Esimerkiksi siirry seuraavaan ja edelliseen kappaleeseen -kontrollien ja edistyspalkin (progress bar) poistaminen ilmoitusvalikon pienoissoittimesta ja Android Auton soittimesta ei ole vielä mahdollista. Tämä olisi haluttu tehdä suorien radiokanavien toistossa. Paremman tuen ja dokumentaation pitäisi kuitenkin olla tulossa pian (24). Ongelmista huolimatta tarpeelliset kontrollit saatiin melko helposti toteutettua.

5.7.3 Puhekomennot ja -haku

Android Auto -mediasovellusten toimintoja pystyy hallitsemaan myös äänikomennoilla Google Assistant -ääniavustajan avulla, ja tuki äänikomennoille onkin Googlen vaatimus. Äänikomentojen käyttäminen parantaa ajoturvallisuutta, kun kosketusnäyttöä ei tarvitse katsoa ja operoida niin paljon.

Toista- (play), tauko- (pause) ja lopeta (stop) -komennot toimivat automaattisesti, eikä niitä tarvinnut erikseen toteuttaa mitenkään. Äänihaulla toiston aloittaminen on myös vaadittu toiminto. Se toimii niin, että käyttäjä sanoo komennon ”Play <sisällön nimike>” tai ”Play <sisällön nimike> on <sovelluksen nimi>”, jos kyseinen sovellus ei ole sillä hetkellä avoinna.

Vaikka Google Assistantissa ei ole ainakaan vielä tukea suomen kielelle, voi sitä käyttää myös Suomessa englannin kielellä. Tämä kuitenkin tarkoittaa, että Assistant ei osaa tulkita suomenkielisiä hakusanoja oikein, mikä tekee äänihausta suurelta osin käyttökelvottoman. Teoreettinen tuki sille on kuitenkin vaatimus, joten äänihaku kehitettiin sovellukseen.

Kun käyttäjä tekee äänihaun, kutsutaan SuplaPlayerServicen onAddMediaItems-takaisinkutsufunktiota, jonka mediaItems-parametrin ensimmäisen ja ainoan MediaItemin requestMetadata-objektin searchQuery-muuttujasta löytyy kyseinen hakusana. Tämä hakusana annetaan parametrina Media3Browsing-luokan getSearchResultPlayable-funktiolle, joka kutsuu Media3BrowsingUseCases-luokan getSearchResultPlayableMediaItem-funktiota. Funktio hakee Component API:sta hakutulossivulta CapiItemin, rakentaa siitä MediaItemin ja palauttaa sen. Sitten tämän hakutulos-MediaItemin toisto aloitetaan normaalisti.

Haun jälkeen soittimeen ilmestyy myös ”Hakutulokset”-painike, jota klikkaamalla pääsee listaan, jossa on kaikki hakutulokset. Listasta voi valita muita tuloksia ja aloittaa niiden toiston (kuva 12). Hakutuloksien näyttäminen tapahtuu SuplaPlayerServicen onSearch-takaisinkutsufunktion avulla, joka ilmoittaa MediaLibrarySessionille, että hakutulokset ovat muuttuneet, ja onGetSearchResult-

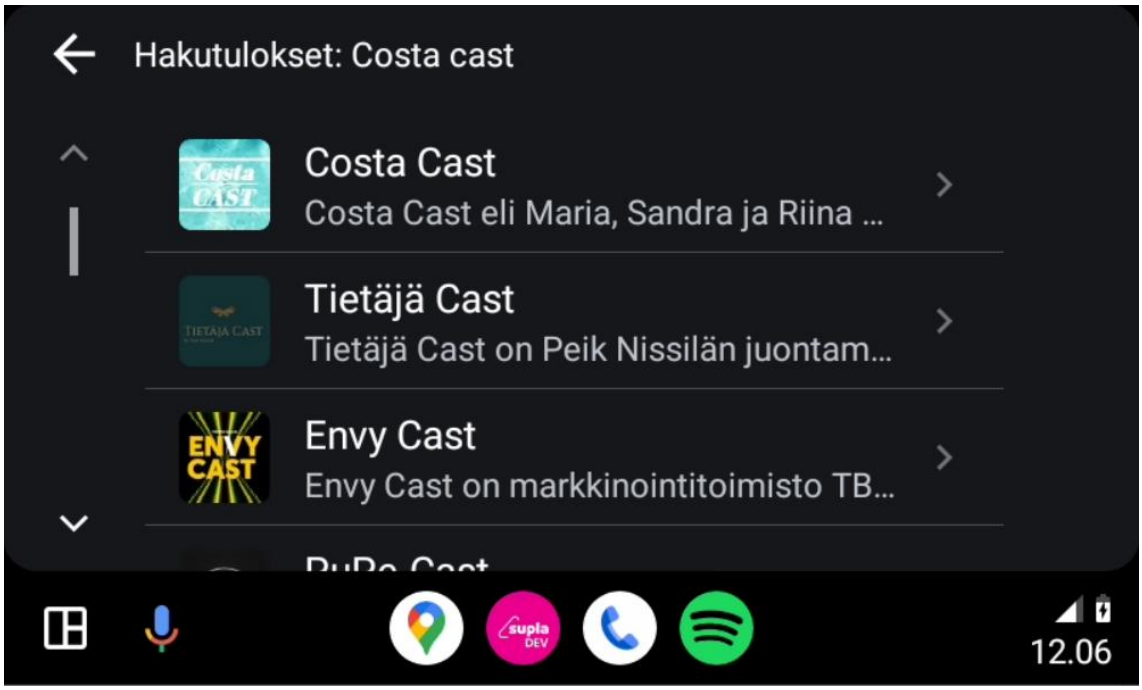
takaisinkutsufunktiolla, jossa haetaan ja palautetaan kaikki hakutulokset query-parametrin eli hakusanan perusteella (esimerkkikoodi 13).

```
override fun onSearch(
     session: MediaLibrarySession,
     browser: MediaSession.ControllerInfo,
     query: String,
     params: LibraryParams?,
): ListenableFuture<LibraryResult<Void>> {
     mediaLibrarySession.notifySearchResultChanged(
          browser,
          query,
          12,
          params
     )
     return Futures.immediateFuture(LibraryResult.ofVoid(params))
}

override fun onGetSearchResult(
     session: MediaLibrarySession,
     browser: MediaSession.ControllerInfo,
     query: String,
     page: Int,
     pageSize: Int,
     params: LibraryParams?,
): ListenableFuture<LibraryResult<ImmutableList<MediaItem>>> {
     return loadContentScope.future {
          LibraryResult.ofItemList(
               media3Browsing.getSearchResultItems(query), params
          )
     }
}
```

Esimerkkikoodi 13. OnSearch- ja onGetSearchResult-takaisinkutsufunktiot.

Puhehaku saatiin toimimaan hyvin käyttäen englanninkielisten sisältöjen nimiä hakusanana. Esimerkiksi ”Play Costa Cast” -komennolla sovellus alkaa toistamaan uusinta Costa Cast -podcastin jaksoa. Suomenkielisillä hakusanoilla tulokset voivat olla mitä vain, koska Google Assistant ei osaa tulkita suomea.



Kuva 12. Hakutulokset-näkymä, jossa listattuna parhaat hakutulokset.

Puhehaku oli melko helppoa toteuttaa, vaikkakin dokumentaatio toteutuksesta Media3:n kanssa oli olematonta. Toimintaperiaate selvisi kokeilemalla MediaLibrarySessionin takaisinkutsufunktioita.

Puhehaun testauksessa ilmeni ongelma, kun hakutuloksen MediaItemin tyyppi oli väärä, kun toistoa yritettiin aloittaa. Tämä aiheutti virhetilan sovelluksessa, ja sovellus kaatui heti käynnistyksen jälkeen. Syynä oli se, että Component API:ssa oli virhe, jossa Android Autolle tarkoitetulla hakusivulla joillain CapiItemeillä oli väärä tyyppi.

5.8 Kehitysprosessi ja testaus

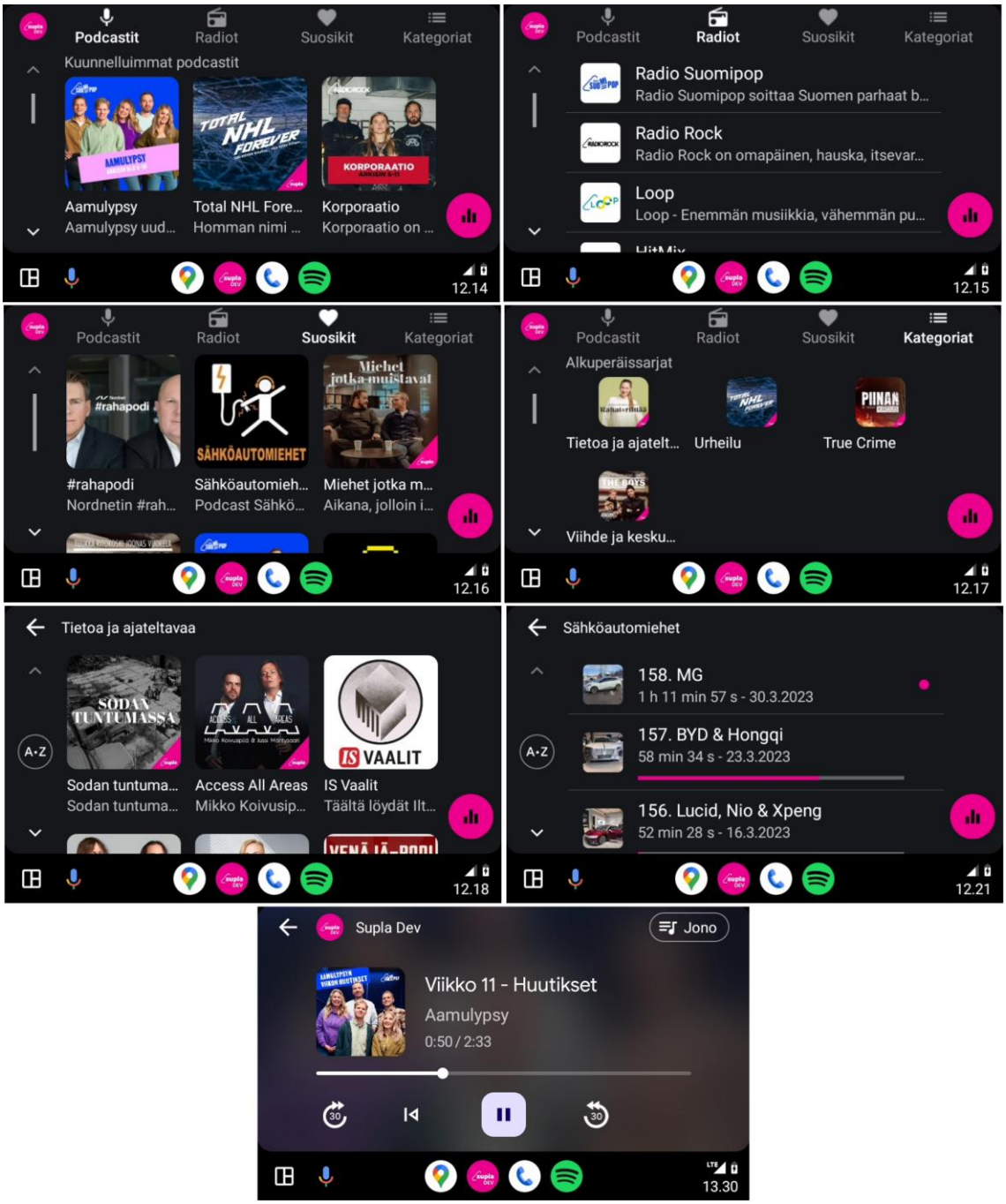
Kehitystyö eteni heti alusta alkaen sujuvasti, ja melko nopeasti ensimmäinen toimiva versio oli valmis. Kehityksessä hyvä apu oli Googlen Android Auto -mediasovellusten dokumentaatio, jossa käytiin pääasiat läpi ja opastettiin eri toimintojen toteuttamiseen (17). Pienempiä ongelmia ilmeni jonkin verran, mutta ne ratkesivat suhteellisen helposti.

Sovellusta testattiin jatkuvasti emulaattorilla, joka toimi pääosin hyvin. Lisäksi ennen tuotantoon julkaisua sovellusta aiotaan testata oikeiden autojen Android Auto -järjestelmillä kehitysmanagerien toimesta. Toteutuksen UseCase-funktioille tehtiin myös yksikkötestit (unit test) koodin toimivuuden varmistamiseksi.

6 Tulokset ja jatkokehitys

Opinnäytetyön lopputulos on toimiva Android Auto -toteutus Supla-sovellukseen. Kaikki suunnitellut ominaisuudet saatiin toteutettua, ja kehitystyö sujui hyvin ja ilman suurempia ongelmia tai viivästyksiä.

Sovelluksella pystyy selaamaan ”Podcastit”-sivulla suosituimpia podcasteja, ”Radiot”-sivulla eri radiokanavia”, ”Suosikit”-sivulla käyttäjän suosikeiksi lisäämiä podcasteja ja ”Kategoriat”-sivulla eri podcast-kategorioita ja niihin kuuluvia podcasteja. Sovelluksella pystyy aloittamaan sisältöjen toiston ja hallitsemaan toistoa kelauspainikkeiden ja äänikomentojen avulla. Lisäksi puhehaku on tuettuna, tosin vain englanniksi. (Kuva 13.)



Kuva 13. Valmiin Supla Android Auto -sovelluksen kaikki eri näkymät.

Tutkimus- ja kehitystyön aikana selvitettiin Android-mediasovelluksen arkkitehtuuria, joka vaaditaan Android Auto -tuen toteuttamiseen. Suositellun mediaarkkitehtuurin todettiin olevan hyvä ja toimiva. Avainkomponentti Android Auton kannalta on MediaLibraryService tai MediaBrowserService. Kun sovelluksen mediasoitin on toteutettu tällä arkkitehtuurilla, toimii moni asia Android Auto -

sovelluksessakin ilman lisätyötä eikä Android Auto -tuen lisääminen ole valtavan suuri työ.

Myös Media3-kirjaston käytöstä Android Auton kanssa saatiin kokemusta. Media3 toimi pääosin hyvin, ja Android Auto -toteutus onnistui hyvin myös siinä. Media3:n dokumentaatio osoittautui kuitenkin erittäin suppeaksi, mikä vaikeutti hieman kehitystyötä. Media3 on julkaistu virallisesti työn loppuvaiheessa, ja kattavampia ohjeita ja dokumentaatiota sen käyttöön luvataan ilmestyvän lähiaikoina (16). Media3:ssa on myös vielä joitain pieniä puutteita, joihin on tulossa ratkaisuja tulevaisuudessa (24; 25). Pääosin Media3 on kuitenkin käyttökelpoinen ja toimintavarma jo nyt. Media3:n käyttöä Android Auton kanssa voi suositella, koska se on uusin ja suoraviivaisin kirjasto mediasovelluksen kehitykseen.

Android Auto -sovellusten kehityksestä ei ole kovin paljon tietoa ja dokumentaatiota saatavilla, mutta Googlen Android Auto -mediasovellusten ohjeistuksesta löytyivät kaikki tärkeimmät asiat, joiden avulla kehitystyössä pääsi hyvin alkuun. Työn aikana huomattiin kuitenkin, että monista yksityiskohtaisemmista seikoista ei löytynyt juuri ollenkaan tietoa. Googlen Android Auto -ohjeet ovat melko suppeat, ja niihin kaipaisi päivitystä ja laajennusta.

Kokonaisuutena Android Auto -tuen lisääminen mediasovellukseen todettiin melko helpoksi, jos sovelluksessa on jo ennestään oikeanlainen media-arkkitehtuuri. Android Auton valikkojen ja sivujen luominen oli nopeaa, eikä käyttöliittymää tarvinnut itse luoda ollenkaan, koska Android Auto -järjestelmä luo sen MediaItemien ja niiden muuttujien pohjalta. Soitin ja mediantoisto toimivat automaattisesti ilman lisätyötä. Android Auto -tuen lisäämistä voi siis suositella kaikentyyppisiin audiomediasovelluksiin, eikä työ ole erityisen suuri tai hankala.

Android Auto -järjestelmä itsessään todettiin erittäin toimivaksi, käteväksi ja käyttökokemukseltaan hyväksi. Navigointi valikoissa on helppoa, ja järjestelmä toimii ripeästi.

Kaikki suunnitellut toiminnot saatiin toteutettua, ja sovellus toimii hyvin. Jatkokehityksenä myöhemmin voisi lisätä toistojonon, josta podcastin muita jaksoja

pystyisi selaamaan ja valitsemaan toistettavaksi. Myös ”Suosikit”-sivulle voisi lisätä uudet mobiilisovellukseen tulevat ”Uudet jaksot”- ja ”Tallennetut jaksot” -näkymät. Lisäksi äänihaun toimivuutta voisi parantaa suomen kielellä, joskin se riippuu täysin siitä, milloin Google lisää suomen kielen Google Assistantiin.

7 Yhteenveto

Insinöörityössä kehitettiin Android Auto -tuki Supla-podcast- ja nettiradiosovellukseen ja tutkittiin Android-mediasovellusten arkkitehtuuria.

Android Auto -mediasovelluksen toteuttamisen edellytys on, että sovelluksen mediasoitin on toteutettu käyttäen Androidin suositeltua mediasovellusarkkitehtuuria. Siinä voidaan käyttää joko vanhempia Androidin mediarajapintoja tai uutta Media3-kirjastoa. Mediasoitin toimii palveluna taustalla MediaLibraryServicen avulla, ja erilaiset asiakassovellukset pystyvät yhdistämään siihen.

Android Auto -sovellus yhdistää MediaLibraryServiceen, esittää sen tarjoaman mediasisällön ja mahdollistaa mediantoiston hallinnan. Android Auto -järjestelmä luo sovelluksen käyttöliittymän monin tavoin muokattavien MediaItemien perusteella.

Media3-kirjasto yksinkertaistaa ja helpottaa Android-sovelluksen media-arkkitehtuuria ja tuo hyviä parannuksia. Media3 on kuitenkin vielä suhteellisen uusi kirjasto, ja siinä on vielä pientä parannettavaa joissain asioissa, kuten soitinkontrollien muokkaamisessa ja dokumentaatiossa. Media3:n todettiin toimivan hyvin Android Auto -toteutuksen kanssa, vaikka ohjeistusta ei juurikaan ollut saatavilla tästä yhdistelmästä.

Android Auto -tuen toteuttamisen olemassa olevaan mediasovellukseen todettiin olevan melko yksinkertaista. Avainasia on sovelluksen mediasoitintoteutus, jonka täytyy olla suositellun mediasovellusarkkitehtuurin mukainen, jotta Android Auto -tuen toteuttaminen onnistuu. Kehityksen aikana kohdattiin joitakin pieniä ongelmia, jotka johtuivat lähinnä dokumentaation ja ohjeistuksen

puutteista. Niihin kuuluivat muun muassa soitinkontrollit ja kuvien lataaminen. Ongelmat ratkesivat kuitenkin melko nopeasti.

Sovellukseen toteutettiin ”Podcastit”-, ”Radiot”-, ”Suosikit”- ja ”Kategoriat”-sivut, soitinkontrollit sekä äänikomennot. Lopputuloksena saatiin toimiva ja vaatimusten mukainen Suplan Android Auto -sovellus, joka julkaistaan kesän 2023 kuluessa tuotantoversioon käyttäjille.

Lähteet

1 Dredge, Stuart. 2014. Apple CarPlay debuts with Ferrari, Mercedes-Benz and Volvo. Verkkoaineisto. The Guardian. <https://www.theguardian.com/technology/2014/mar/03/apple-carplay-ferrari-mercedes-benz-volvo>. Päivitetty 3.3.2014. Luettu 24.3.2023.

2 Welch, Chris. 2015. Google's launch of Android Auto starts today with Pioneer head units. Verkkoaineisto. The Verge. <https://www.theverge.com/2015/3/19/8259565/google-launches-android-auto>. Päivitetty 19.3.2015. Luettu 24.3.2023.

3 Sanoma Media Finland. Verkkoaineisto. <https://www.sanoma.fi/>. Luettu 7.5.2023.

4 Supla. 2015. Sanoma Media Finland Oy.

5 Android app quality for cars. Verkkoaineisto. Android Developers. <https://developer.android.com/docs/quality-guidelines/car-app-quality>. Luettu 7.5.2023.

6 Short guide to developing apps for Android Auto. Verkkoaineisto. appssemble. <https://appssemble.com/blog/android-car.html>. Luettu 3.2.2023.

7 Dow, Cat. 2023. What exactly is Android Auto and what does it do? Verkkoaineisto. Top Gear. <https://www.topgear.com/car-news/top-gear-advice/what-exactly-android-auto-and-what-does-it-do>. Päivitetty 20.1.2023. Luettu 3.4.2023.

8 Android for Cars overview. Verkkoaineisto. Android Developers. <https://developer.android.com/training/cars>. Luettu 28.2.2023.

9 O'Kane, Sean. 2019. Toyota finally caves and announces cars with Android Auto compatibility. Verkkoaineisto. <https://www.theverge.com/2019/2/7/18215741/toyota-android-auto-compatibility-tundra-sequoia-tacoma-4runner>. Päivitetty 7.2.2019. Luettu 6.4.2023.

10 Walker, Steve. 2023. What is Android Auto? Full review and user guide. Verkkoaineisto. Auto Express. <https://www.autoexpress.co.uk/car-tech/96496/what-is-android-auto-full-review-and-user-guide>. Päivitetty 17.2.2023. Luettu 27.2.2023.

11 Google Maps. 2008. Google.

12 Media app architecture overview. Verkkoaineisto. Android Developers. <https://developer.android.com/guide/topics/media-apps/media-apps-overview>. Luettu 3.3.2023.

13 Lake, Ian. 2016. MediaBrowserServiceCompat and the modern media playback app. Verkkoaineisto. Medium. <https://medium.com/androiddevelopers/mediabrowserservicecompat-and-the-modern-media-playback-app-7959a5196d90>. Päivitetty 2.3.2016. Luettu 3.2.2023.

14 Media3 overview. Verkkoaineisto. Android Developers. <https://developer.android.com/guide/topics/media/media3>. Luettu 7.3.2023.

15 Turner, Don. 2021. Introducing Jetpack Media3. Verkkoaineisto. Android Developers Blog. <https://android-developers.googleblog.com/2021/10/jetpack-media3.html>. Päivitetty 27.10.2021. Luettu 7.3.2023.

16 Mital, Nevin. 2023. Media3 is ready to play! Verkkoaineisto. <https://android-developers.googleblog.com/2023/03/media3-is-ready-to-play.html>. Päivitetty 23.3.2023. Luettu 3.4.2023.

17 Build media apps for cars. Verkkoaineisto. Android Developers. <https://developer.android.com/training/cars/media>. Luettu 3.2.2023.

18 Test Android apps for cars. Verkkoaineisto. Android Developers. <https://developer.android.com/training/cars/testing>. Luettu 3.2.2023.

19 Android Studio. 2014. Google.

20 Add support for Android Auto to your media app. Verkkoaineisto. Android Developers. <https://developer.android.com/training/cars/media/auto>. Luettu 3.2.2023.

21 ListenableFutureExplained. 2020. Verkkoaineisto. GitHub (Google). <https://github.com/google/guava/wiki/ListenableFutureExplained>. Päivitetty 3.3.2020. Luettu 28.4.2023.

22 Universal Android Music Player Sample. 2022. Verkkoaineisto. GitHub (Android). <https://github.com/android/uamp>. Päivitetty 14.12.2022. Luettu 13.2.2023.

23 Dumbravan, Alexandru. 2022. Clean Android Architecture. E-kirja. Packt Publishing.

24 Reflecting available commands in buttons of the Notification. 2022. Verkkoaineisto. GitHub (androidx). <https://github.com/androidx/media/issues/140>. Päivitetty 10.1.2023. Luettu 7.5.2023.

25 Media buttons in notification. 2022. Verkkoaineisto. GitHub (androidx). <https://github.com/androidx/media/issues/216>. Päivitetty 19.1.2023. Luettu 7.5.2023.