OULUN AMMATTIKORKEAKOULU

Laura Pietikäinen

PAINO- JA TULOSTUSVÄRIEN VARASTONHALLINTASOVELLUS

PAINO- JA TULOSTUSVÄRIEN VARASTONHALLINTASOVELLUS

Laura Pietikäinen
Opinnäytetyö
Syksy 2018
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä: Laura Pietikäinen
Opinnäytetyön nimi: Paino- ja tulostusvärien varastonhallinta-sovellus
Työn ohjaaja: Lasse Haverinen
Työn valmistumislukukausi ja -vuosi: Syksy 2018
Sivumäärä: 44

Tämän opinnäytetyön tarkoituksena oli tuottaa tilaajalle sovellus, jonka tarkoituksena on selkeyttää ja helpottaa värivaraston käyttöä ja hallintaa yhdistämällä käytettyjä menetelmiä yhteen sovellukseen.

Sovellus toteutettiin Android Studiolla ja liittämällä siihen Firebase-konsolissa Real Time Database-tietokannan sekä Firebase Authentication-palvelun. Sovelluksella käyttäjä voi lisätä värireseptin ja hakea reseptejä tietokannasta. Tämän lisäksi käyttäjä voi täyttää tilauslistan ja hakea tilauksia. Sovelluksessa on myös sävytys osio, jolla käyttäjä voi hakea värin RGB- ja HEX-arvot, sekä vuorolista, johon käyttäjät voivat määrittää vuoronsa.

Lopputuloksena syntyi toivotunlainen sovellus, jolle voisi ajatella vielä laajempaa roolia ja monia muitakin ominaisuuksia, jos kehittäminen jatkuu käyttöönoton jälkeen.

Asiasanat: Android Studio, Firebase, varastonhallinta

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Software Engineering

---

Author: Laura Pietikäinen
Title of thesis: Warehouse management application for printing inks
Supervisor: Lasse Haverinen
Term and year when the thesis was submitted: Autumn 2018
Pages: 44

The purpose of this thesis was to provide an application that is meant to clarify and facilitate the use and management of the color warehouse by digitalizing them into one application.

Application was implemented in Android Studio and by connecting it to the Firebase console Real Time database, as well as the Firebase Authentication service.

The application allows the user to add a color recipe and search for recipes from the database, and user can also fill up order lists and search for orders. The application includes a color search section that allows the user to search for the RGB and HEX values of the color, there is all so a section where user can add their shifts when they have a responsibility for the color warehouse.

My opinion is, that the result was a successful application that could have a bigger role in the future and many more features could be possible to implemented if the development continues after this.

Keywords: Android Studio, Firebase, Warehouse Management.

ALKULAUSE

Haluan kiittää työn tilaajaa mielenkiintoisesta ja haastavasta projektista. Kiitokset myös työntekijöille, joilta sai palautetta ja kehitysideoita projektin edetessä, sekä kiitos opinnäytetyön ohjaajalle hyvästä ohjauksesta.

Oulussa 11.12.2018

Laura Pietikäinen

SISÄLLYS

SANASTO

API	Application Program Interface. Sovelluksen ohjelmointirajapinta.
APK	Android Application Package. Android-sovelluksen asennuspaketti.
AVD	Android Virtual Devices. Android Studion virtuaalikoneet.
Bitmap	Bittikartta. Tapa, jossa esimerkiksi jokaisen pisteen bittimäärä määritellään tietyllä bittimäärällä.
GUI	Graphical User Interface. Tarkoittaa käyttöliittymää, jonka käyttäjä näkee käyttäessään sovellusta.
HEX	Heksadesimaali. Lukujärjestelmä, jossa kantaluku on 16. Merkitään yleisesti väriarvoja.
IDE	Integrated Development Environment. Sovelluskehitysympäristö.
JAVA	Oliopohjainen ohjelmointikieli.
JSON	JavaScript Object Notation. Avoimen standardin tiedostomuoto tiedonvälitykseen.
RGB	Red, Green, Blue. Grafiikassa käytetty värijärjestelmä.
SDK	Software Development Kit. Esim. Android Studion ladattavat ohjelmistokehityspaketit.

1 JOHDANTO

Tämän opinnäytetyön tilaajana toimi Tekniseri Oy. Työn tavoitteena oli digitalisoida värivarastonhallinta yhteen sovellukseen.

Ajatus paino- ja tulostusvärien varastonhallintasovelluksesta lähti alkusyksyn kuukausipalaverissa, kun työntekijät toivovat käyttöön nykyaikaisempaa järjestelmää, jolla pystyisi hakemaan värireseptit ja katsomaan, onko värisävylle olemassa jo resepti vai pitääkö sellainen laatia. Toivottiin myös selkeämpää tapaa löytää värisävyt, sillä niitä on valtava määrä käytössä, ja luonnollisesti etsiminen vie työaikaa.

Tästä lähti ajatus, että muutkin toiminnot, kuten tilaus- ja vuorolistat voisi laittaa samaan sovellukseen. Digipainajan töissä taas tarvitsee ennakoida ajoissa värien tilaaminen, joten tähän sovellukseen sisällytettiin tilauslista myös tulostusvärеille. Sovellukseen sisällytin myös kaksi erilaista sävytystoimintoa, joilla voi etsiä RGB- ja HEX-arvoina sopivanlaista sävyä.

Koska sovellus ei ole julkinen ja sen sisältämä tieto on tarkoitettu yrityksen työntekijöiden käyttöön, sovellus tarvitsi myös kirjautumis- ja rekisteröitymissivun.

Firebase-konsolin kautta hallitaan sovelluksen kirjautumiseen liittyvä autentikointi ja siellä sijaitsee sovelluksen tallennettavien tietojen reaaliaikainen tietokanta. Tietokantaan kerätään talteen tiedot väriresepteistä, vuoroista ja tilauslistoista. Firebase-tietokannasta tieto välittyy reaaliaikaisesti kaikille sovelluksen käyttäjille.

Lopputuloksena syntyi kaksi Android-sovellusta, jotka sisällöllisesti vastaavat toisiaan, mutta toinen on tarkoitettu älypuhelimelle ja toinen tabletille.

2 ANDROID-KÄYTTÖJÄRJESTELMÄ

Sovelluksen toteutukseen valitsin käyttöjärjestelmäksi Androidin sillä se on älypuhelimien yleisin käyttöjärjestelmä. Android on Googlen ylläpitämä avoimeen lähdekoodiin pohjautuva järjestelmä, joka käyttää Linux-ydintä. Opinnäytetyön Android-sovellus on kehitetty Android Studiolla, joka on yhtä lailla Googlen lanseeraama kehitysympäristö Android-sovelluskehitykseen.

2.1 Android-arkkitehtuuri

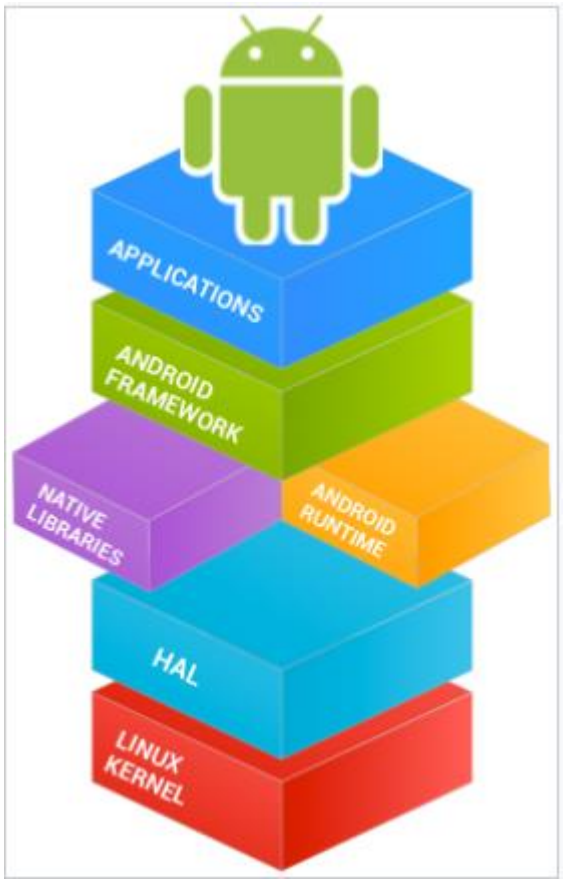
Android-alusta rakentuu Linux-pohjaisen ytimen päälle. Linux-ydin tarjoaa Androidille tietoturvallisuus ominaisuuksia sekä mahdollistaa laiteohjaimien kehityksen **Linux-kernelille** (kuva 1). Kerrokset lyhyesti:

- **Hardware Abstraction Layer** eli laitteiston abstraktiokerros (**HAL**) koostuu useista moduuleista, jotka toteuttavat rajapinnan tiettyihin laitteistokomponentteihin. Näitä komponentteja ovat esimerkiksi Bluetooth- ja kameramoduulit. (1.)
- **Android Runtime (ART)** on uudempien laitteiden ominaisuus (alkaen Android API-tasolta 21), jossa jokainen sovellus toimii omana prosessinaan. ART tarjoaa Androidille virheenkorjaustuen, yksityiskohtaiset diagnoosit ja kaatumisraportoinnit sekä kyvyn asettaa tarkkailupisteet tietyn kentän valvontaan. Android sisältää myös joukon keskeisiä runtime-kirjastoja, jotka tarjoavat erilaisia Java-ohjelmointikielen toimintoja. (1.)
- **Native C/C++ libraries** -kerrosta tarvitaan sillä monet Android-järjestelmän peruskomponentit ja -palvelut, kuten ART ja HAL, on rakennettu alkuperäisestä koodista, jonka toiminta edellyttää C: n ja C ++:n kirjoittamia natiivikirjastoja. Android-alustalla on Java-kehyssovellusliittymät, joiden avulla jotkut näistä natiivikirjastoista toimivat sovelluksissa. Eli jos kehitetään sovellusta, joka vaatii C- tai C++ -koodin, voi käyttää Android-NDK:ta, jotta päästään suoraan natiivikirjastoihin alkuperäisestä koodista. (1.)

- **Java API Framework** -kerroksessa on Android-käyttöjärjestelmän kaikki ominaisuudet käytettävissä Java-kielellä kirjoitettujen käyttöliittymien

kautta. Nämä sovellusliittymät muodostavat rakennusosat, jotka tarvitaan Android-sovellusten luomiseen yksinkertaistamalla ydinosien, moduuli järjestelmäkomponenttien ja palveluiden uudelleenkäyttöä. (1.)

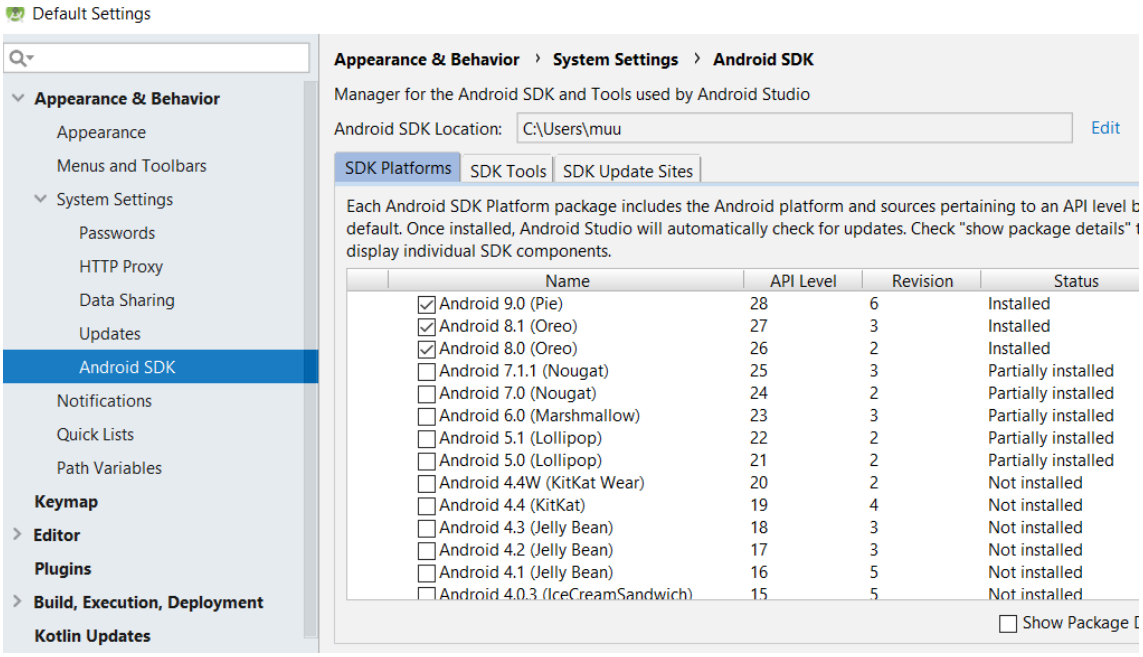
- **System Apps** -kerros sisältää joukon ydinsovelluksia, kuten sähköposti, tekstiviestien lähettäminen, kalenteri, internet-selain ja yhteystiedot. Sovelluskehittäjät voivat käyttää näitä ydinsovelluksia omassa sovelluksessaan. Esimerkiksi jos sovellukseen halutaan tekstiviestin lähetys ominaisuus, ei sovelluskehittäjän tarvitse erikseen rakentaa kyseistä toimintoa. (1.)



KUVA 1. Android-arkkitehtuuri (2)

2.2 Android-sovelluskehitys

Android-sovellus voidaan kirjoittaa Kotlin-, Java- ja C / C++ -kielillä. Android SDK -työkalut kokoavat koodin, resurssitiedostot ym. tiedostot APK-tiedostoksi. Yksi APK-tiedosto sisältää siis kaiken Android-sovelluksen sisällön, ja on tiedosto, joka asennetaan Android-laitteelle. Android Studion Android Software Development Kit eli SDK sisältää paljon erilaisia Android-sovelluskehitykseen liittyviä kehitystyökaluja (kuva 2). SDK-kehitystyökaluja ovat esimerkiksi virheenkorjaustyökalut, kirjastot, emulaattori, dokumentaatio, näytekoodit ja monet erilaiset opetusohjelmat. (2.)



KUVA 2. Android Studion Android SDK:sta on mahdollista ladata erikseen sovellukseen tarvittavat komponentit.

3 VAATIMUSMÄÄRITTELY

Vaatimusmäärittely on tässä projektissa jaettu kolmeen osioon, joista ensimmäisessä määritellään sovelluksen toiminnalliset vaatimukset, toisessa tekniset vaatimukset ja kolmannessa on kuvattu käyttötapaukset. Vaatimusmäärittely luo perustan sille, miltä pohjalta sovellusta kehitetään. Sovelluksen käyttötapauksia on kolme erilaista. Ne ovat peruskäyttäjä, kehittäjä ja ylläpitäjä (kuva 3).

Toiminnalliset vaatimukset

- Helppokäyttöinen, yksinkertaistetut toiminnot nopeaan työskentelyyn. (vinkit ja valikot).
- Selkeä ja suomenkielinen (ohjeet ja vaadittavat kentät ja opasteet olisivat osa koodia).

Tekniset vaatimukset

- Android-sovellus, joka toimii älypuhelimessa ja tabletissa. Toteutus Android Studiolla
- Reaaliaikainen tietokanta ja autentikointi. Toteutus Firebase-konsolilla.

Käyttötapaukset

Sovelluksen peruskäyttäjä voi

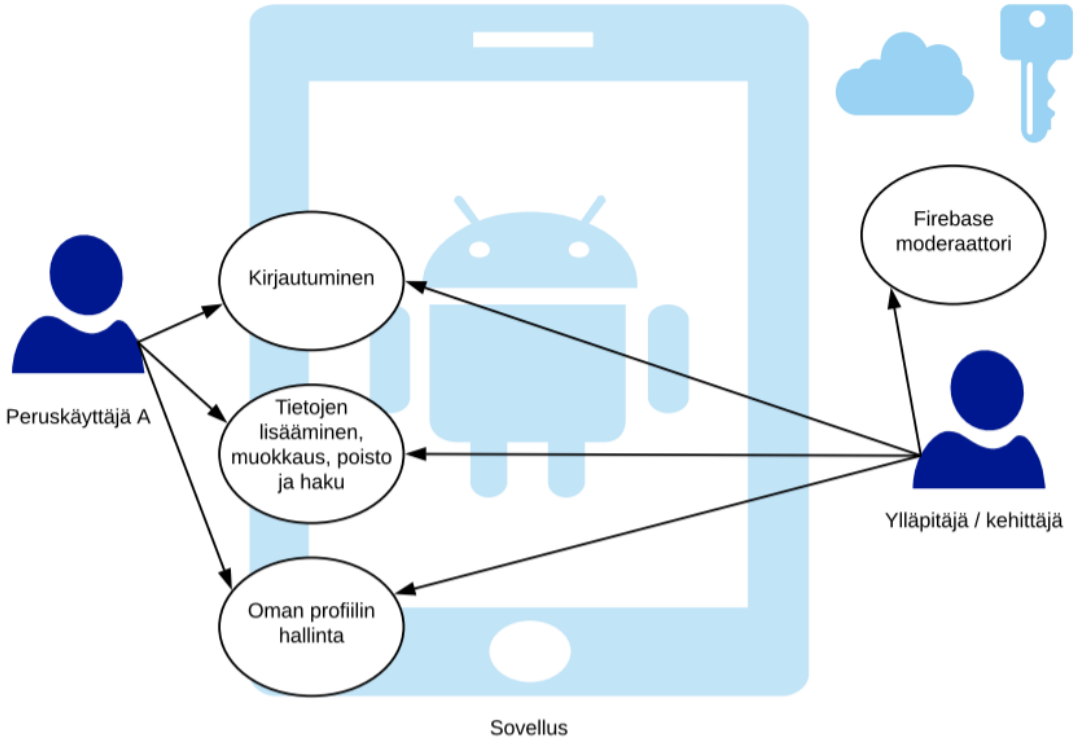
- luoda ja poistaa omat käyttäjätunnukset sekä vaihtaa salasanansa
- kirjautua sovellukseen ja ulos
- käyttää hakutoimintoa sekä lisätä, muokata ja poistaa tietoja
- käyttää sovelluksen muita osia.

Sovelluksen ylläpitäjä ja kehittäjä voi edellä mainittujen lisäksi

- toimia moderaattorina ja ongelmatilanteissa poistaa käyttäjän
- hallinnoida tietokantaa.
- hallinnoida käyttäjätilejä.

Sovelluksen ylläpitäjän täytyy vielä lisäksi

- varmistaa, että sovellus on päivitetty viimeisimpään versioon ja, että sovellus toimii moitteitta eri laitteissa.
- varmistaa, että henkilöt, joilla on sovellus omalla laitteella saavat päivitettyä viimeisimmän version, jos sovellukseen on tullut uusia päivityksiä.
- huolehtia sovelluksen varmuuskopioinnista ja toimia teknisenä tukena sovelluksen käyttäjille.



KUVA 3. Käyttötapaukset.

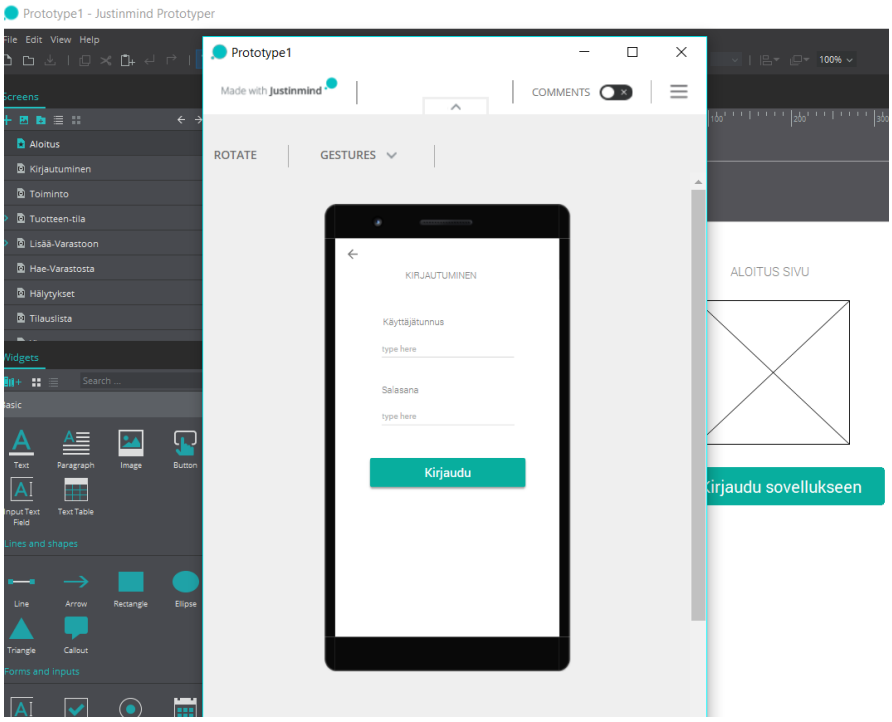
4 MENETELMÄT JA TYÖKALUT

Android Studion lisäksi sovelluksen kehittämiseen tarvitsi myös suunnittelutyökaluja, versionhallintaa ja aikataulun seurantaa. Ensimmäisenä arvioin, mistä osista sovellus tulee koostumaan, ja kuinka paljon aikaa niiden ohjelmointi veisi. Tästä etenin suunnittelemaan sovelluksen käyttöliittymää, jonka jälkeen pystyi vasta ohjelmoimaan varsinaista ohjelmakoodia.

4.1 Käyttöliittymäsuunnittelu

Sovelluksen kehittäminen alkoi graafisen käyttöliittymän (GUI) suunnittelusta, jonka toteuttamiseen käytin käyttöliittymäsuunnitteluun tarkoitettua ohjelmaa nimeltä JustInMind Prototyper.

JustInMind Prototyper on työkalu web -ja mobiilisovelluksien suunnitteluun. Sillä pystyy muotoilemaan layoutin sopivanlaiseksi ja oikeaan kokoon kohdelaitteen mukaan. Komponentit voidaan linkittää esimerkiksi navigoimaan seuraavaan näkymään ja tällä tavoin yhdistää sovellukseen suunnitellut layoutit ja aktiviteetit toisiinsa halutussa järjestyksessä. Valikosta löytyy tyypillisimmät komponentit kuten erilaiset näppäimet, tekstikentät, kuva, kalenteri jne. Asetelman lisäksi komponenteille voi antaa ominaisuuksia, kuten muokata niitä sopivan kokoisiksi, vaihtaa värejä yms. ja linkittää niitä tiettyihin toimintoihin. Kun layoutit ja aktiviteetit on yhdistetty, voi suunnitelman testata ohjelman simulaatiolla (kuva 4).



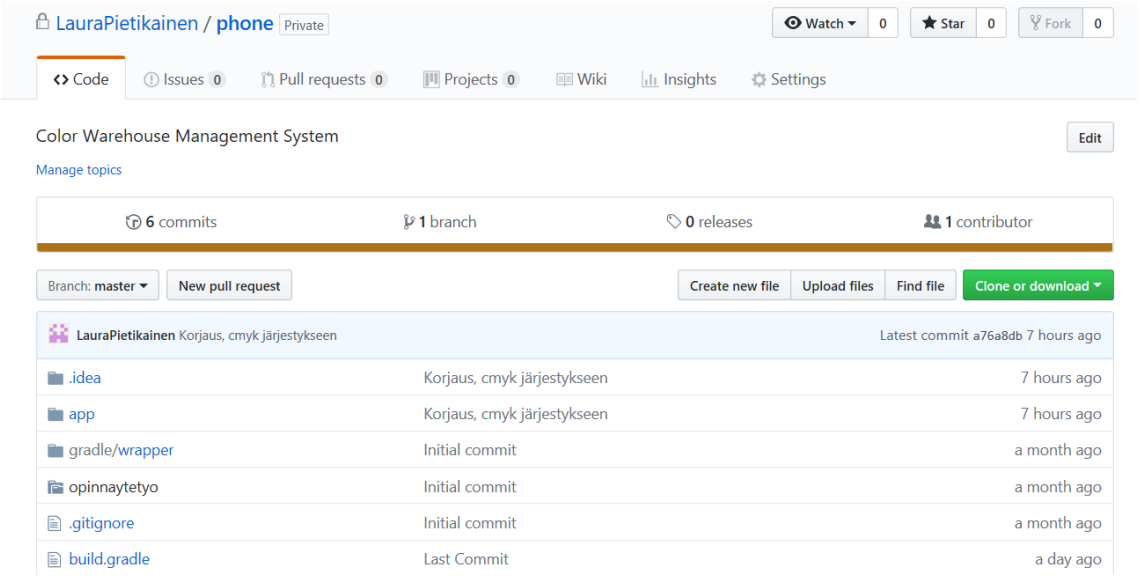
KUVA 4. Simulaatio käyttöliittymästä.

4.2 Projektinhallinta

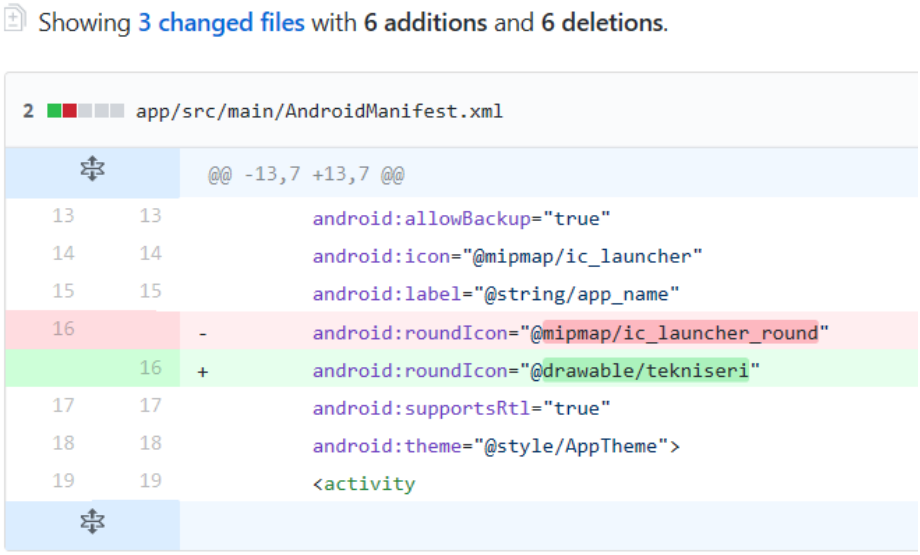
Opinnäytetyön tekeminen edellytti aikataulun hallintaa, suunnittelua ja raportointia. Opinnäytetyön ohjaaja sai viikoittain raportin työn etenemisestä ja tämän lisäksi raportoin eri vaiheista Trelloon, sekä tallensin eri vaiheiden suunnitelmia opinnäytetyön Google Drive kansioon. GitHub taas toimi projektissa versionhallinta työkaluna (kuva 5).

4.2.1 Versionhallinta GitHubissa

Git-versionhallintaa tarvitaan sovelluskehityksessä, koska normaalisti sovelluksella on useampi kehittäjä ja muiden kehittäjien tarvitsee tietää, missä projekti etenee. Tässä tapauksessa versionhallinta oli käytössä, jotta ohjaaja näkee, kuinka projekti on edennyt. Kun viimeisin muokattu koodi ladataan Githubin projektikansioon, siitä syntyy tiedosto, josta erot edelliseen versioon näkyvät (kuva 6). GitHubissa on mahdollistaa jakaa lähdekoodi kaikille tai vaihtoehtoisesti pitää se yksityisenä. GitHub on hyödyllinen siitä, että kehittäjä voi ladata lähdekoodin käyttämälleen koneelle ja jatkaa ohjelmointia laitteesta riippumatta.



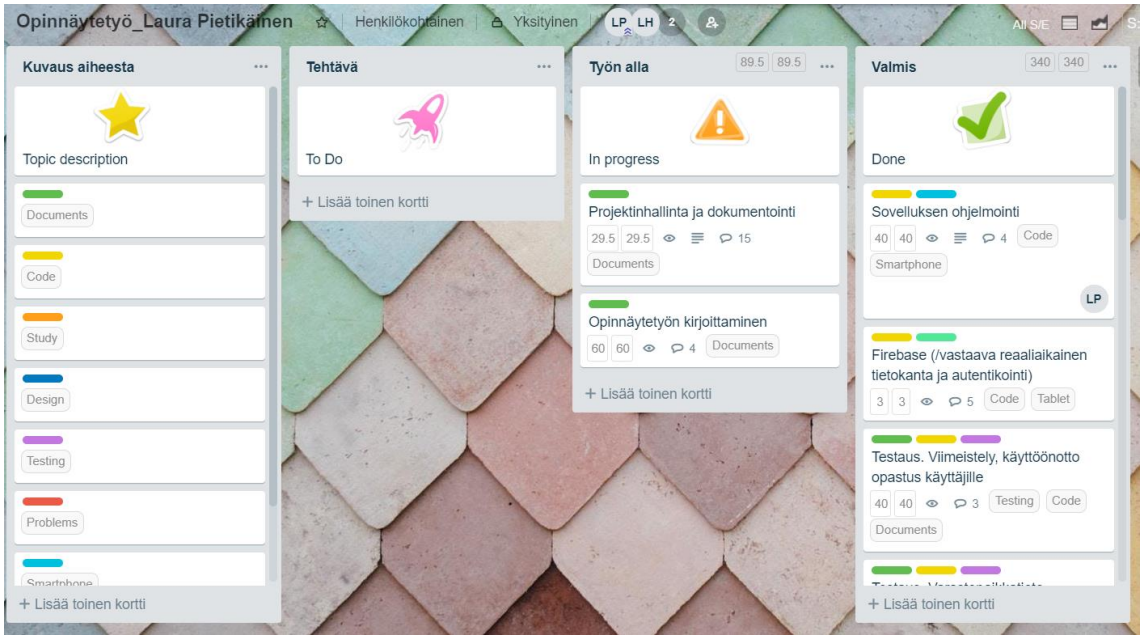
KUVA 5. Versionhallinta GitHubissa.



KUVA 6. GitHub-versionhallinta, esimerkki muutoksesta AndroidManifest.xml-tiedostossa.

4.2.2 Projektinhallintajärjestelmä Trello

Trello on verkossa toimiva palvelu, jossa voi suunnitella aikatauluja, asettaa takarajan, kirjata tuntimäärät, sekä jakaa sisältöä ja linkkejä osallistujien kesken (kuva 7).

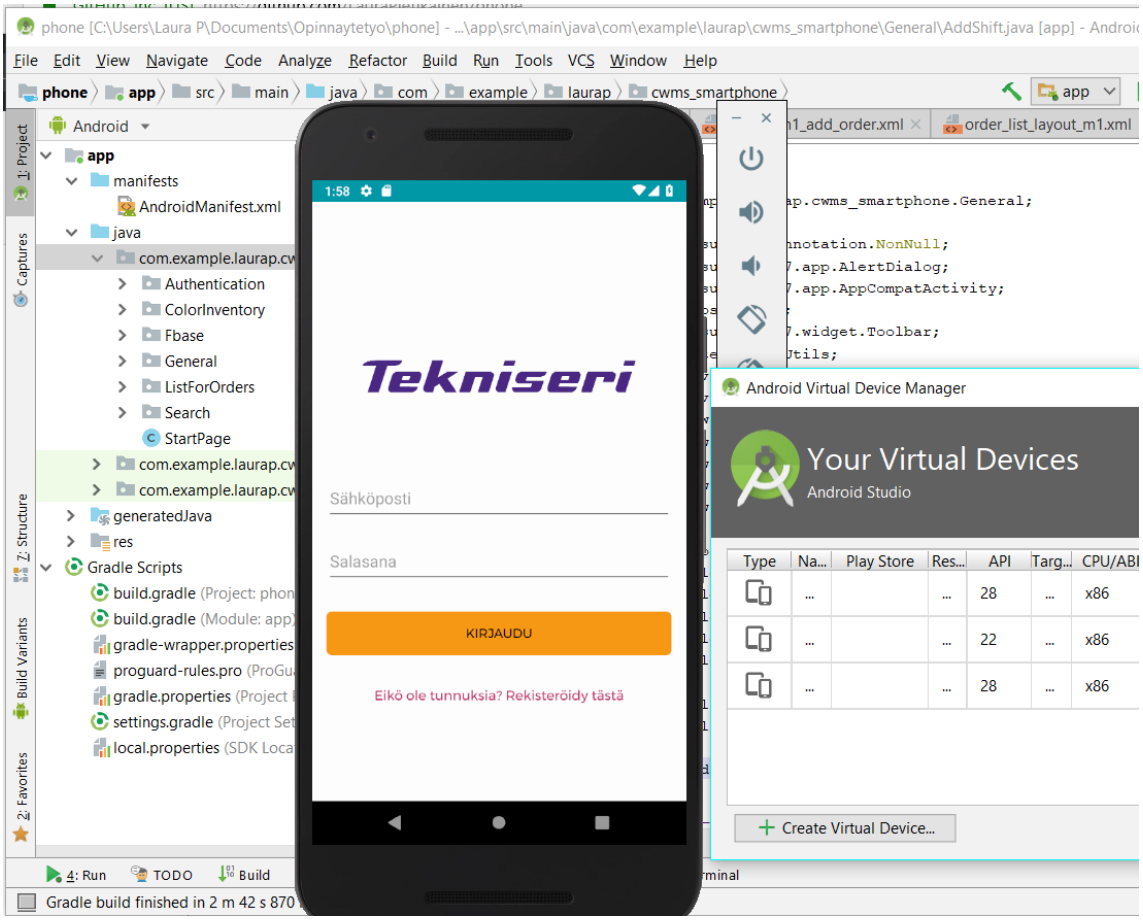


KUVA 7. Projektinhallinta, Trello

4.3 Ohjelmointiympäristö Android Studio IDE

Android Studio on Android-sovelluksille tarkoitettu ohjelmointiympäristö (IDE). Se on mahdollista ladata ilmaiseksi Windows-, MacOs- ja Linux-käyttöjärjestelmille. (3.) Android Studio IDE tarjoaa ohjelmoijalle käyttöliittymän, jossa voi kehittää sovellustaan ja saada sovelluksen käyttöön tietyt laajennukset, kuten Android SDK, sekä tuen ja valmiit kirjastot peruskomponenteille. Android Studiolla on mahdollista suorittaa koodi käyttämällä virtuaalikonetta eli emulaattoria, joka simulaattorin tapaan näyttää ohjelman toiminnan, mutta myös vastaa oikean sovelluksen toimintaa (kuva 8). Emulaattori voidaan valita halutun laitteen ja API-version mukaan ja niitä voi luoda useita erilaisia. Yleensä kun uusi laite tai uusi

Android-versio saapuu markkinoille, on mahdollista ladata uusimmat versiot myös Android Studion AVD-virtuaalikoneisiin.



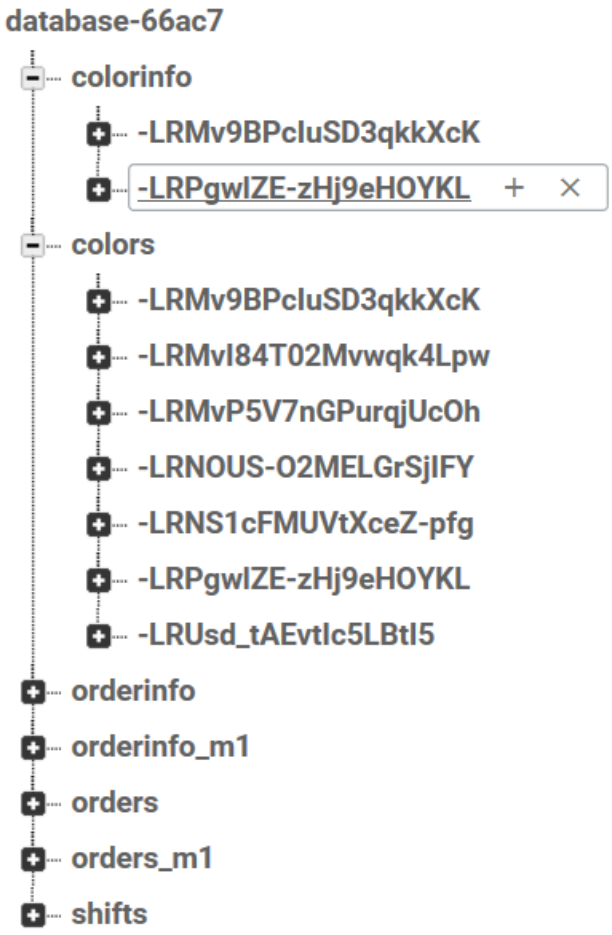
KUVA 8. Android Studio IDE, jonka emulaattorissa testataan opinnäytetyön sovellusta.

4.4 Firebase-konsoli

Firebase on Googlen hallinnoima pilvipalvelu, jossa voi monella tavalla hallinnoida sovelluksesta kerättyä dataa, sekä sovelluksen käyttöoikeuksia. Opinnäytetyössä oli tarpeellista saada sieltä käyttöön reaaliaikainen tietokanta ja autentikointi, jotta pystytään hallinnoimaan kuka sovellusta saa käyttää.

4.4.1 Real Time Database -tietokanta

Firebase Real Time Database on pilvipalvelu, jonne sovelluksen keräämät tiedot tallentuvat reaaliaikaisesti. Tiedot tallentuvat JSON-muodossa ja synkronoituvat lähes välittömästi käyttäjilleen, sillä tyypillisten http-kutsujen sijaan Firebase käyttää datasynkronointia ja lähettää käyttäjälle uusia tietoja heti, kun se päivitetään (kuva 9) . Jos sovelluksen käyttäjä on offline-tilassa, tiedot synkronoituvat tietokantaan, kun yhteys palautuu. Realtime Databasen Security Rules-asetuksista kehittäjä voi määritellä, minkä laajuiset käyttöoikeudet sovelluksen käyttäjille sallitaan. (4.)



KUVA 9. Firebase Realtime Database, JSON-rakenne

4.4.2 Autentikointi, Firebase Authentication

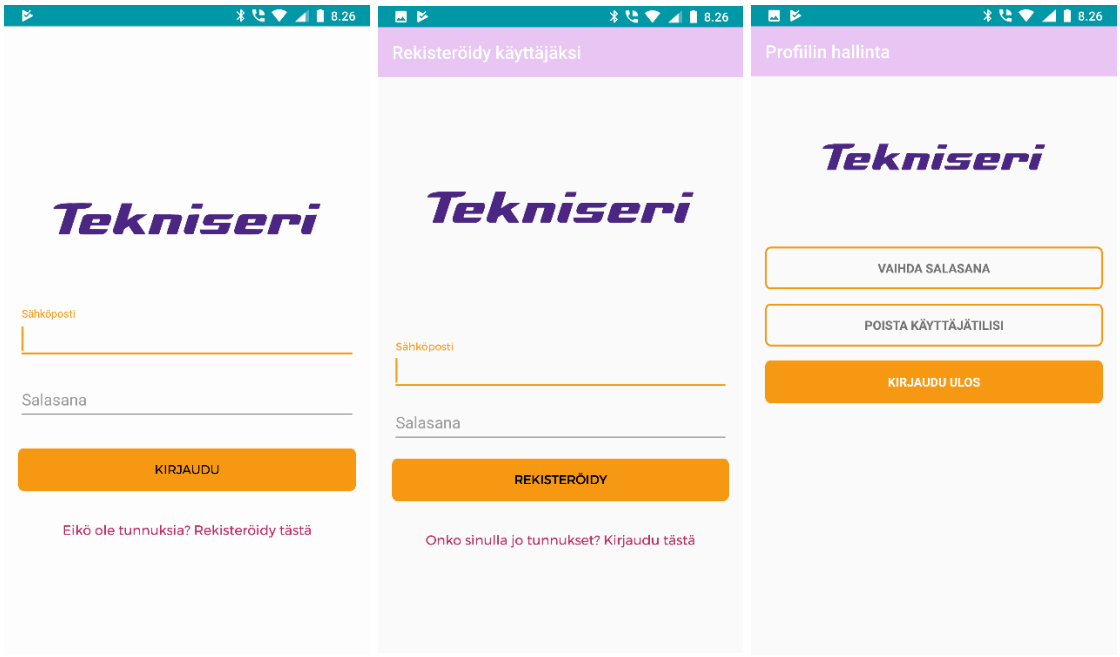
Firebase mahdollistaa käyttäjän identiteetin tunnistamisen erilaisilla tunnistautumismahdollisuuksilla (kuva 10). Ongelmatilanteissa sovelluksenkehittäjän tai ylläpitäjän on mahdollista estää haluttu käyttäjätili, jolloin hän ei enää pääse kirjautumaan takaisin sovellukseen. Vaihtoehtoisesti tarvittaessa ylläpitäjä voi myös resetoida käyttäjän salasanan tai poistaa koko käyttäjätilin. (5.)

KUVA 10. Erilaisia palveluntarjoajia Firebase-tunnistautumiseen (5.)

5 KÄYTTÖLIITTYMÄN OSAT JA TOTEUTUS

5.1 Tunnistautuminen

Sovellukseen kirjautuminen edellyttää käyttäjätilin luomista. Kun käyttäjä on asentanut sovelluksen, aloitussivu ohjaa kirjautumaan sovellukseen tai vaihtoehtoisesti rekisteröitymään käyttäjäksi (kuva 11). Käyttäjän rekisteröidyttyä sovellukseen, hänen käyttäjätietonsa tallentuvat Firebasen käyttäjätietoihin.



KUVA 11. Sovelluksen kirjautuminen, rekisteröinti ja profiilin hallinta.

5.1.1 Rekisteröityminen

Jotta rekisteröityminen onnistuu, on käyttäjän sähköpostiosoitteen oltava oikea osoite ja salasanan riittävän pitkä. Kun tietojen syöttämisen jälkeen käyttäjä painaa REKISTERÖIDY-painiketta, painikkeelle asetettu *setOnClickListener*-metodi tarkastaa, että täytetyt kentät täyttävät ehdot. Jos ehdot täyttyvät, uusi käyttäjätili luodaan *CreateUserWithEmailAndPassword* -metodilla (kuva 12). Tämän jälkeen sovellus ohjautuu seuraavaan aktiviteettiin. Jos käyttäjätilin luominen epäonnistuu, tämä ilmoitetaan käyttäjän näytölle.

```
//Create new user account
auth.createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener( activity: SignupActivity.this, (task) → {
                //If sign in is successful, displays Toast message:
                if (task.isSuccessful())
                    Toast.makeText( context: SignupActivity.this,
                            text: "Uusi käyttäjätunnus luotu!", Toast.LENGTH_SHORT).show();
                progressBar.setVisibility(View.GONE);
                // If sign in fails, displays a Toast message to the user.
                if (!task.isSuccessful()) {
                    Toast.makeText( context: SignupActivity.this,
                            text: "Käyttäjätunnuksen luonti epäonnistui!" + task.getException(),
                            Toast.LENGTH_SHORT).show();
                } else {
                    //Intent to next Activity
                    startActivity(new Intent( packageContext: SignupActivity.this, StartPage.class));
                    finish();
                }
        });
```

KUVA 12. CreateUserWithEmailAndPassword-metodi

5.1.2 Kirjautuminen

Rekisteröitymisen jälkeen käyttäjä pystyy jatkossa kirjautumaan sovellukseen tunnuksillaan. Kirjautumissivulla KIRJAUDU-painikkeelle on vastaavasti asetettu kuuntelija, joka varmistaa, että käyttäjä ja salasana täsmäävät. Jos tiedot on syötetty virheellisesti, käyttäjän näytölle lähetään virheilmoitus (kuva 13). Kun kentät on täytetty oikein, käyttäjä vahvistetaan signInWithEmailAndPassword -metodilla (kuva 14). Tämän jälkeen Firebase Authentication -tiedoista vahvistetaan kirjautumistunnuksilla rekisteröitynyt henkilö getCurrentUser-metodilla (kuva 15). Onnistuneen tunnistautumisen jälkeen sovellus ohjautuu pääsivulle (kuva 16).

```
btn_login.setOnClickListener((v) → {
        String email = et_email.getText().toString();
        final String password = et_password.getText().toString();
        //If email field is empty, displays a Toast:
        if (TextUtils.isEmpty(email)) {
            Toast.makeText(getApplicationContext(),
                    text: "Syötä sähköpostiosoite!", Toast.LENGTH_SHORT).show();
            return;
        }
        //If password fiels is empty, diaplays a Toast:
        if (TextUtils.isEmpty(password)) {
            Toast.makeText(getApplicationContext(),
                    text: "Syötä salasana!", Toast.LENGTH_SHORT).show();
            return;
        }
```

KUVA 13. setOnClickListener-metodi

```
//authenticate user
auth.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener( activity: LoginActivity.this, (task) → {
                progressBar.setVisibility(View.GONE);
                if (!task.isSuccessful()) {
                    // min.password
                    if (password.length() < 6) {
                        et_password.setError("Salasana on liian lyhyt, min. 6 merkkiä!");
                    } else {
                        //Authentication failed Toast
                        Toast.makeText( context: LoginActivity.this,
                                getString(R.string.auth_failed),
                                Toast.LENGTH_LONG).show();
                    }
                } else {
                    Intent intent = new Intent( packageContext: LoginActivity.this, StartPage.class);
                    startActivity(intent);
                    finish();
                }
        });
```

KUVA 14. signInWithEmailAndPassword -metodissa vahvistetaan käyttäjä.

```
//Get Firebase auth instance
auth = FirebaseAuth.getInstance();

if (auth.getCurrentUser() != null) {
    startActivity(new Intent( packageContext: LoginActivity.this, StartPage.class));
    finish();
}
```

KUVA 15. getCurrentUser-metodilla saadaan käyttöön käyttäjätilintiedot. Kun kirjautuminen on onnistunut, sovellus siirtyy aloitussivulle.

KUVA 16. Onnistuneen tunnistautumisen jälkeen sovellus ohjautuu pääsivulle.

5.1.3 Profiilinhallinta

Profiilinhallinnassa käyttäjän on mahdollista vaihtaa salasanansa (kuva 17), poistaa käyttäjätilinsä ja kirjautua ulos sovelluksesta (kuva 18). Salasanan vaihto tapahtuu samankaltaisesti kuin salasanan luominen ja se edellyttää, että sille annetut ehdot täyttyvät.

```
//Confirm Password change
btn_confirm_pw_change.setOnClickListener((v) → {
        progressBar.setVisibility(View.VISIBLE);
        if (user != null && !et_new_password.getText().toString().trim().equals("")) {
            if (et_new_password.getText().toString().trim().length() < 6) {
                et_new_password.setError("Salasana on liian lyhyt, minimi 6 merkkiä");
                progressBar.setVisibility(View.GONE);
            } else {
                user.updatePassword(et_new_password.getText().toString().trim())
                        .addOnCompleteListener((task) → {
                                if (task.isSuccessful()) {
                                    //If change is succesfull display Toast:
                                    Toast.makeText( context: ProfileActivity.this,
                                            text: "Salasanasi on vaihdettu!", Toast.LENGTH_SHOR
                                    signOut();
                                    progressBar.setVisibility(View.GONE);
                                } else {
                                    //If change is not successfull diaplays Toast:
                                    Toast.makeText( context: ProfileActivity.this,
                                            text: "Salasanan vaihto epäonnistui!", Toast.LENGTH_
                                    progressBar.setVisibility(View.GONE);
                                }
                            });
            }
        } else if (et_new_password.getText().toString().trim().equals("")) {
            //If password field is empty display Toast:
            et_new_password.setError("Syötä salasana");
            progressBar.setVisibility(View.GONE);
        }
});
//Button Delete Account OnClickListener
```

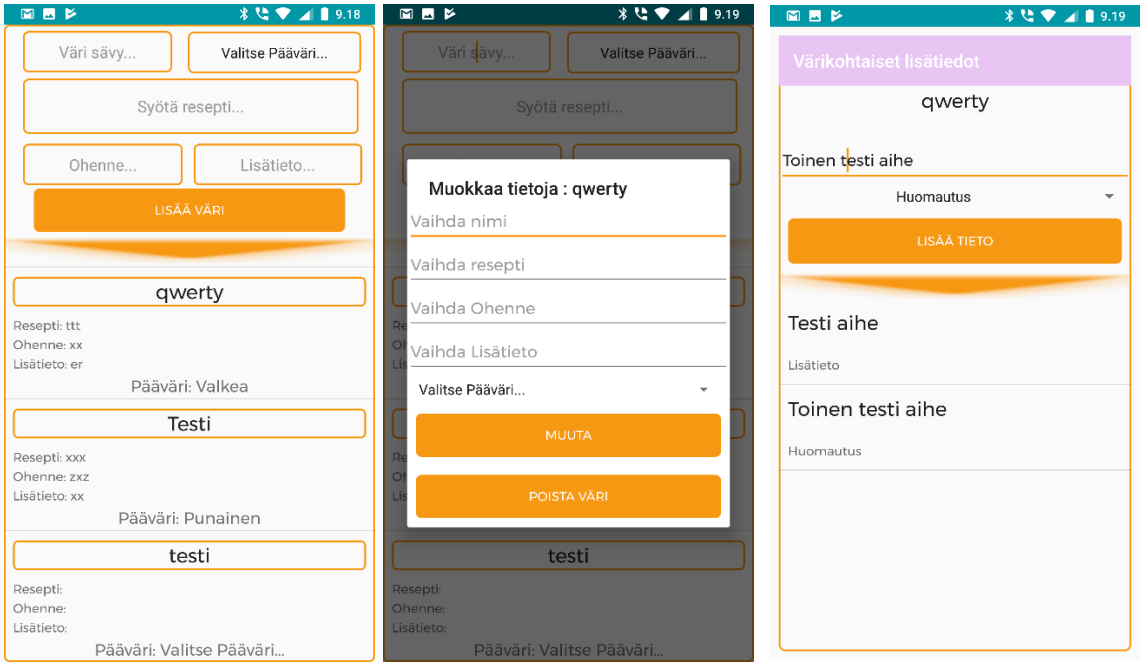
KUVA 17. Salasanan vaihto.

```
    //Button Delete Account OnClickListener
    btn_delete_account.setOnClickListener((v) → {
            progressBar.setVisibility(View.VISIBLE);
            if (user != null) {
                user.delete()
                        .addOnCompleteListener((task) → {
                                if (task.isSuccessful()) {
                                    //If task is successful display a Toast:
                                    Toast.makeText( context: ProfileActivity.this,
                                            text: "Käyttäjätilisi on poistettu!", Toast.LENGTH_SHORT).show();
                                    startActivity(new Intent( packageContext: ProfileActivity.this, SignupActivity.class));
                                    finish();
                                    progressBar.setVisibility(View.GONE);
                                } else {
                                    //If task is not successfull display a Toast:
                                    Toast.makeText( context: ProfileActivity.this,
                                            text: "Käyttäjätilin poisto epäonnistui!", Toast.LENGTH_SHORT).show();
                                    progressBar.setVisibility(View.GONE);
                                }
                            });
            }
    });
    //Sign Out onClickListener
    btn_sign_out.setOnClickListener((v) → { signOut(); });

}

//Sign out method
public void signOut() { auth.signOut(); }
```

KUVA 18. Käyttäjätilin poisto ja uloskirjautuminen

5.2 Lisää väri -toiminto

Lisää väri -toiminnon tarkoituksena on se, että kun painaja on määrittänyt värikartan mukaisen reseptin, hän voi sen tallentaa tietokantaan, josta se on taas tarvittaessa uudelleen löydettävissä. Kun käyttäjä on täyttänyt yläosaan tarvittavat tiedot ja painanut LISÄÄ VÄRI -painiketta, tieto siirtyy alaosan listaan. Jos tietoja tarvitsee vielä jälkeenpäin muokata, valitaan listalta muokattava kohde ja pitkällä painalluksella saadaan auki tietojen muokkaus dialogi. Normaali painalluksesta taas avautuu aktiviteetti, jossa voi lisätä värikohtaisia lisätietoja. (kuva 19)



KUVA 19. Ensimmäisenä perusnäkymä, toisena muokkausnäkymä ja kolmantena lisätietojen täyttö.

5.2.1 Väritiedon lisääminen listalle

Tiedon lisäys listalle on toteutettu onStart-metodissa, jossa luodaan uusi addValueEventListener. Tässä luokassa taas kutsutaan onDataChange()-metodia, jotta

for-loopissa saataisiin käyttöön DataSnapshot, joka sisältää tietokantakohtaiset tiedot (kuva 20).

```
//Generated Override method onStart
@Override
protected void onStart() {
    super.onStart();
    //ValueEventListener for database reference object

    databaseColors.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {

            colorList.clear();

            for(DataSnapshot colorSnapshot : dataSnapshot.getChildren()){
                Color color = colorSnapshot.getValue(Color.class);
                //Sorting the last added color to show up first
                listViewColors.setStackFromBottom(true);
                colorList.add(color);
            }
            //Create ArrayAdapter
            ColorList adapter = new ColorList( context: AddColor.this, colorList);
            listViewColors.setAdapter(adapter);
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
    });
}
```

KUVA 20. onStart-metodi.

5.2.2 Väritiedon päivitys ja poisto

Väritiedon päivitys- ja poistotoiminto on toteutettu käyttämällä dialogia eli valintaikkunaa, joka on ohjelmoitu ilmestymään pitkästä painalluksesta. Tämä tapahtuu määrittämällä setItemOnLongClickListener värilistan kohteelle (kuva 21). Dialogissa käyttäjä voi päivittää tiedot täyttämällä pakolliset kentät tai vaihtoehtoisesti poistaa tiedot kokonaan. Dialogissa tapahtuvia muutoksia kuuntelee kaksi erillistä näppäimille asetettua kuuntelijaa (kuva 22).

```
    //setItemOnLongClickListener for listViewColors
    listViewColors.setOnItemLongClickListener((parent, view, i, l) -> {

            Color color = colorList.get(i);
            showUpdateDialog(color.getColorId(), color.getColorName(),
                    color.getColorName2(), color.getColorName3(), color.getColorName4());
            return false;
    });
}
```

KUVA 21. setItemOnLongClickListener.

```
btn_update.setOnClickListener((v) -> {
        String name = et_colorname.getText().toString().trim();
        String name2 = et_colorname2.getText().toString().trim();
        String name3 = et_colorname3.getText().toString().trim();
        String name4 = et_colorname4.getText().toString().trim();

        String main = spinner_main.getSelectedItem().toString();

        if(TextUtils.isEmpty(name)){
            et_colorname.setError("Pakollinen tieto");
            return;
        }
        if(TextUtils.isEmpty(name2)){
            et_colorname.setError("Pakollinen tieto");
            return;
        }
        if(TextUtils.isEmpty(name3)){
            et_colorname.setError("Pakollinen tieto");
            return;
        }
        if(TextUtils.isEmpty(name4)){
            et_colorname.setError("Pakollinen tieto");
            return;
        }
        updateColor(colorId, name, name2, name3, name4, main);

        alertDialog.dismiss();
});

btn_delete.setOnClickListener((v) -> { deleteColor(colorId); });
```

KUVA 22. Dialogin näppäinten kuuntelijat.

5.2.3 Värikohtaiset lisätiedot

Valitsemalla värilistalta yksittäisen kohteen normaalilla painalluksella pääsee lisäämään lisätietoja kyseisen värin kohdalle. Ohjelmassa setOnItemClickListener kuuntelee, kun listalta valittua kohdetta painetaan. Painalluksesta se luo uuden Intentin ja siirtyy AddInfoActivityyn, jossa lisätiedot voi lisätä (kuva 23). Tämä toteutus toimii samalla periaatteella kuin värinlisäys, mutta täytettäviä kenttiä on vähemmän ja lisäyksestä luodaan oma JSON-säie Firebase-tietokantaan (kuva 24). Molemmilla säikeillä on yhteinen ColorId, joka toimii ikään kuin Primary keyn roolissa.

```
//OnItemClickListener
listViewColors.setOnItemClickListener((adapterView, view, i, l) → {
        //Open next activity
        Color color = colorList.get(i);
        //new intent
        Intent intent = new Intent(getApplicationContext(), AddInfoActivity.class);

        intent.putExtra(COLOR_ID, color.getColorId());

        intent.putExtra(COLOR_NAME, color.getColorName());
        intent.putExtra(COLOR_NAME2, color.getColorName2());
        intent.putExtra(COLOR_NAME3, color.getColorName3());
        intent.putExtra(COLOR_NAME4, color.getColorName4());

        startActivity(intent);
});
```

KUVA 23. setOnItemClickListener.

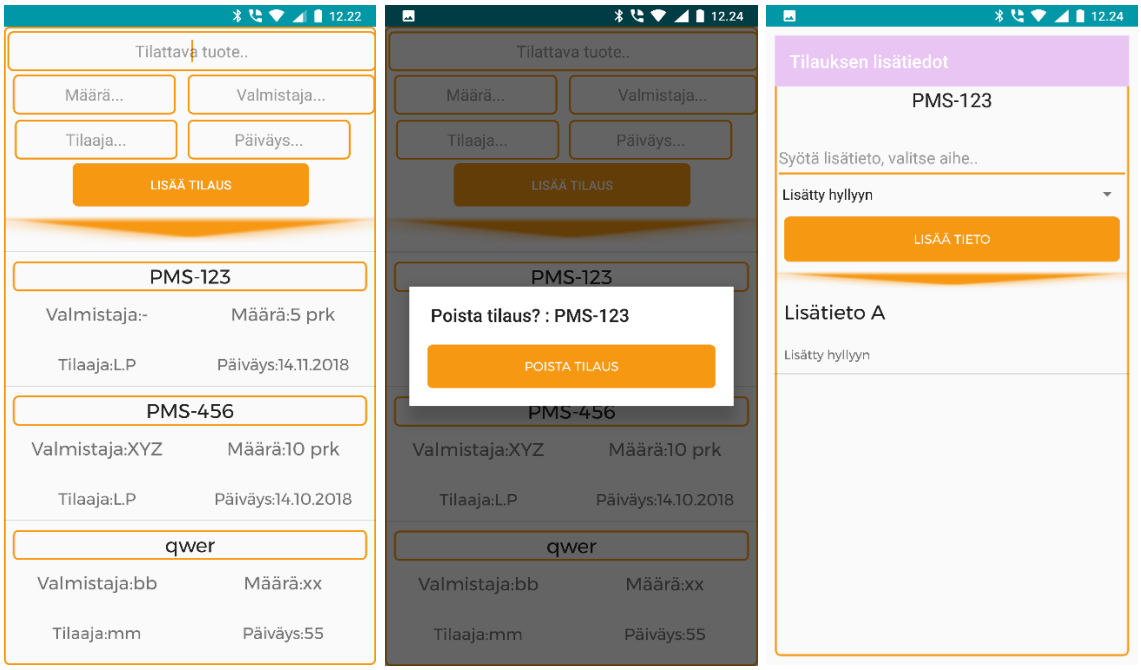
KUVA 24. colorinfo ja colors ovat erilliset säikeet, mutta niitä yhdistää sama ColorId.

5.3 Tilauslistat

Paino- ja tulostusväreille on sovelluksessa omat tilauslistat. Toteutukset on tehty samalla periaatteella kuin värinlisäys, joten en lähde koodin yksityiskohtia toistamaan uudelleen, mutta kerron pääpiirteittäin, millaisia nämä tilauslistat ovat.

5.3.1 Painovärien tilauslista

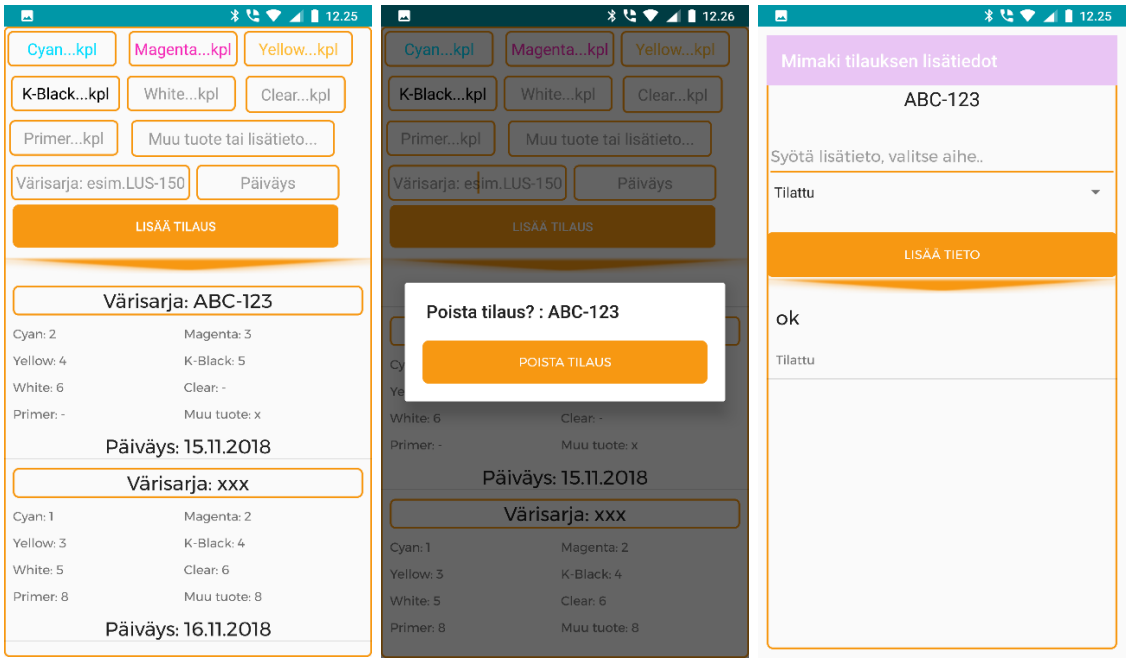
Painovärien tilauslistaan voi lisätä tuotteen, määrän, valmistajan, tilauspäivämäärän sekä tilauksen tekijän. Tilaus siirtyy LISÄÄ TILAUS -painikkeesta alaosan listalle ja lista siirtyy automaattisesti näyttämään viimeisimmän tilauksen. Dialogista on mahdollista poistaa tilaus ja se tapahtuu painamalla pitkään haluttua listan kohdetta. Normaalipainalluksesta avautuu lisätiedot, johon käyttäjä voi merkitä tilaukseen liittyviä tapahtumia muistiin kirjoittamalla ne lisätietokenttään ja painamalla LISÄÄ TIETO -painiketta. (kuva 25)



KUVA 25. Tilauslista, tilauksen poisto ja tilauksen lisätiedot.

5.3.2 Tulostusvärien tilauslista

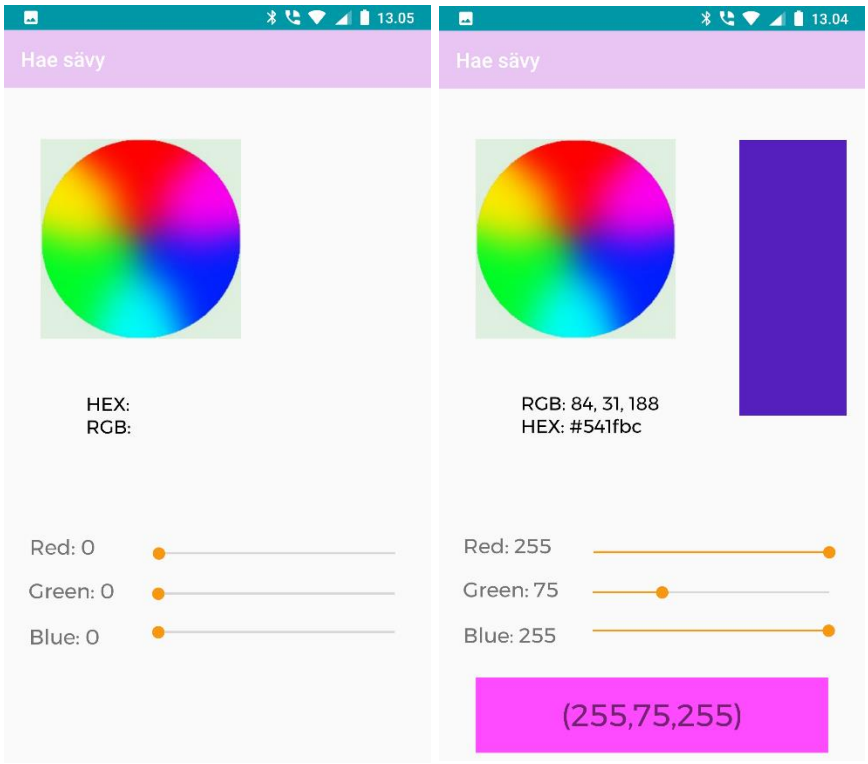
Tulostusvärit lisätään myös samalla periaatteella. Myös siinä pitkästä painalluksesta avautuu dialogi tilauksen poistoon ja lisätiedot normaalista painalluksesta. (kuva 26)



KUVA 26. Tulostusvärien tilauslista, tilauksen poisto ja tilauksen lisätiedot.

5.4 Sävytys

Sävytystoimintoja on kaksi. Toisessa pystyy hakemaan sävyt liikuttamalla sormea väriympyrän päällä, jolloin se määrittää sen kohdan RGB- ja HEX-arvot näkyviin näytölle. Toisena vaihtoehtona sävy voidaan hakea liikuttamalla kolmea eri hakupalkkia, joista se laskee sävyn RGB-arvon. (kuva 27)



KUVA 27. Sävyt voidaan hakea joko säätämällä sävy sopivaksi palkeista tai hakemalla väriympyrästä.

5.4.1 Sävyn hakeminen kuvasta

Kuvaan valikoitu väriympyrä, josta löytyy mahdollisimman kattavasti eri sävyjä. Sävyn hakeminen väriympyrästä tapahtuu liikuttamalla sormea ympyrän päällä. Väriympyrälle on asetettu ohjelmassa setOnTouchListener, joka kuuntelee näytön kosketusta (kuva 28). Jotta pikseliarvo voidaan määrittää värille, täytyy bitmap.getPixelin lukea kosketusta kohdasta RGB-arvo. Kun RGB-arvo on saatu, siitä voidaan muuntaa HEX-arvo. Tämän jälkeen valittu RGB-arvo määrittää kuvaan arvoa vastaavan sävyn. RGB- ja HEX -arvot tulostuvat ja kuvan sävy muuttuu sitä mukaa, kun sormea liikutetaan näytöllä.

```
//image view on touch listener
mImageView.setOnTouchListener((v, event) → {
        if(event.getAction()== MotionEvent.ACTION_DOWN || event.getAction() == MotionEvent.ACTION_MOVE){
            bitmap = mImageView.getDrawingCache();

            int pixel = bitmap.getPixel((int)event.getX(), (int)event.getY());

            //getting RGB values
            int r = red(pixel);
            int g = green(pixel);
            int b = blue(pixel);

            //getting Hex Values
            //String hex = "#"+ Integer.toHexString(pixel);
            String hex = String.format("#%02x%02x%02x", r, g, b);
            //set background color of view according to the picked color
            mColorView.setBackgroundColor(rgb(r,g,b));

            //set RGB, HEX values to textview
            mResultTv.setText("RGB: "+r +", "+ g +", "+ b +"\nHEX: "+ hex);

        }
        return true;
});
```

KUVA 28. setOnTouchListener.

5.4.2 Sävyn hakeminen palkeista

Koska RGB-värimallissa on kolme eri värikanavaa (Red, Green ja Blue), tarvitaan jokaiselle ohjelmassa oma säätöpalkki. Jokaisella kuvapisteellä voi olla 256 erilaista kirkkaustasoa, ja jokaiselle palkille on layoutissa määritelty sen mukaan maksimiarvo. Android Studiosta löytyy tuki RGB-arvoille. Arvot on ohjelmassa alustettu nollaan ja lukema voi kasvaa aina määriteltyyn maximiarvoonsa asti. Palkeille on määritelty setOnSeekBarChangeListener kuuntelija, joka kuuntelee, mihin arvoon palkki on säädetty. Jokainen kuuntelija kuuntelee vain yhtä väriä ja tulostaa vain yhden palkin arvon. Nämä kaikki kolme palkkia yhdessä muokkaavat kuvan väriä ja RGB-arvoa sitä mukaa, kun niitä säädetään. (kuva 29)

```
Sb_red.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener(){
    @Override
    public void onProgressChanged(SeekBar Sb_red, int i, boolean bo) {
        r = i;
        Text_r.setText("Red: "+r);
        Text_rgb.setText("("+r + ","+ g +"," +b+")");
        Text_rgb.setBackgroundColor(rgb(r, g, b));
    }
    @Override
    public void onStartTrackingTouch(SeekBar Sb_red) {}
    @Override
    public void onStopTrackingTouch(SeekBar Sb_red) {}
});

Sb_green.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener(){
    @Override
    public void onProgressChanged(SeekBar Sb_green, int i, boolean bo) {
        g = i;
        Text_g.setText("Green: "+g);
        Text_rgb.setText("("+r + ","+ g +"," +b+")");
        Text_rgb.setBackgroundColor(rgb(r, g, b));
    }
    @Override
    public void onStartTrackingTouch(SeekBar Sb_red) {}
    @Override
    public void onStopTrackingTouch(SeekBar Sb_red) {}
});

Sb_blue.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener(){
    @Override
    public void onProgressChanged(SeekBar Sb_blue, int i, boolean bo) {
        b = i;
        Text_b.setText("Blue: "+b);
        Text_rgb.setText("("+r + ","+ g +"," + b +")");
        Text_rgb.setBackgroundColor(rgb(r, g, b));
    }
```

KUVA 29. RGB-palkkien kuuntelijat.

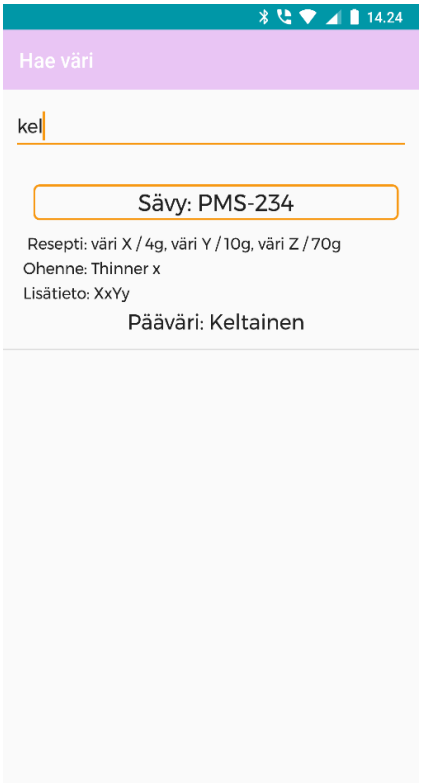
5.5 Haku

Sovelluksessa on kolme erilaista hakuvaihtoehtoa, yksi resepteille, toinen painoväreille ja kolmas tulostusväreille. Tässä sovelluksessa olen toteuttanut kyselyt niin, että käyttäjä pystyy hakemaan kirjaamansa reseptin joko reseptin nimellä tai sen päävärillä. Tilauslistat puolestaan voidaan hakea joko tuotteen nimellä, värisarjalla tai päiväyksellä. Käytännössä jokaiseen hakuvaihtoehtoon on ohjelmoitu

kaksi arvoa, millä niitä voi hakea (kuva 32). Haettu resepti on tallennettuna JSON-muodossa Firebase tietokannassa (kuva 31). Tietoa haetaan tietokannasta for-loopissa, jossa on määritelty if- ja else if-lauseet, joihin on lisätty mitä arvoja näytetään, kun resepti näkyy näytöllä (kuva 32). Lauseisiin on määritelty myös, kuinka monta hakutulosta näytölle listautuu ja haku huomioi pienellä ja isoilla kirjaimilla kirjoitetut vastaavat sanat hakutuloksiin. Hakutulokseen myös lisätään teksti siitä, mitä haettu arvo edustaa (kuva 33).

5.5.1 Värireseptin haku

Värireseptejä voi hakea tietokannasta kahdella eri hakuarvolla, jotka ovat sävyn nimi ja pääväri. Haettavan sävyn nimen yleensä määrittää värikartan mukainen sävy, eli esim. RAL- tai PMS-arvo, joka on määritetty silloin, kun väri on tietokantaan tallennettu. Toinen sovellukseen ohjelmoitu hakuvaihtoehto on pääväri. Päävärillä hakien käyttäjä voi etsiä suuntaa antavaa reseptiä (kuva 30).



KUVA 30. Värireseptiä haetaan päävärillä.

KUVA 31. Väriresepti JSON-muodossa.

```
//Search all colors for matching searched string
for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
    String color_name = snapshot.child("colorName").getValue(String.class);
    String color_name2 = snapshot.child("colorName2").getValue(String.class);
    String color_name3 = snapshot.child("colorName3").getValue(String.class);
    String color_name4 = snapshot.child("colorName4").getValue(String.class);

    String color_main = snapshot.child("colorMain").getValue(String.class);

    if (color_name.toLowerCase().contains(searchedString.toLowerCase())) {
        colorNameList.add(color_name);
        recipeNameList.add(color_name2);
        thinnerNameList.add(color_name3);
        otherNameList.add(color_name4);
        mainNameList.add(color_main);
        counter++;


    } else if (color_main.toLowerCase().contains(searchedString.toLowerCase())) {
        colorNameList.add(color_name);
        recipeNameList.add(color_name2);
        thinnerNameList.add(color_name3);
        otherNameList.add(color_name4);

        mainNameList.add(color_main);
        counter++;
    }

     //Get maximum of 50 searched results
    if (counter == 50)
        break;
}
```

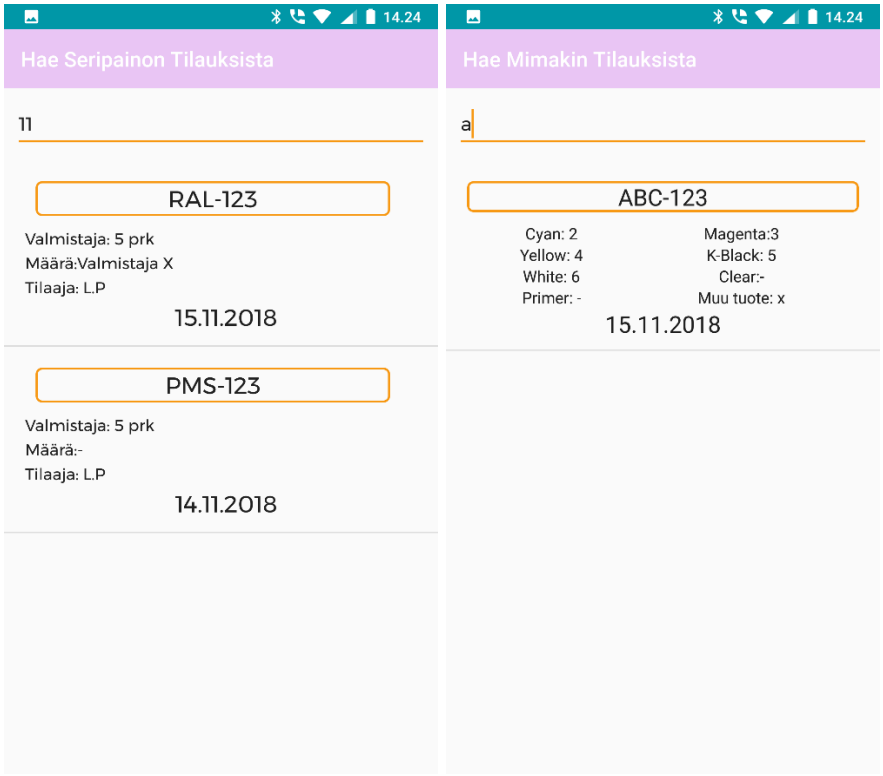
KUVA 32. Haku Firebase-tietokannasta.

```
@Override
public void onBindViewHolder(SearchViewHolder holder, int position) {

    holder.color_name.setText("Sävy: " + colorNameList.get(position));
    holder.recipe_name.setText(" Resepti: " +colorRecipeList.get(position));
    holder.thinner_name.setText("Ohenne: " + colorThinnerList.get(position));
    holder.other_name.setText("Lisätieto: " + colorOtherList.get(position));

    holder.main_name.setText("Pääväri: " + colorMain.get(position));

}
```

KUVA 33. SearchAdapter.java luokassa kutsutaan onBindViewHolder metodia. Tässä kohtaa myös lisätään haettavaan tietoon määrite siitä, mitä haettu tieto on.

5.5.2 Tilauslistojen haku

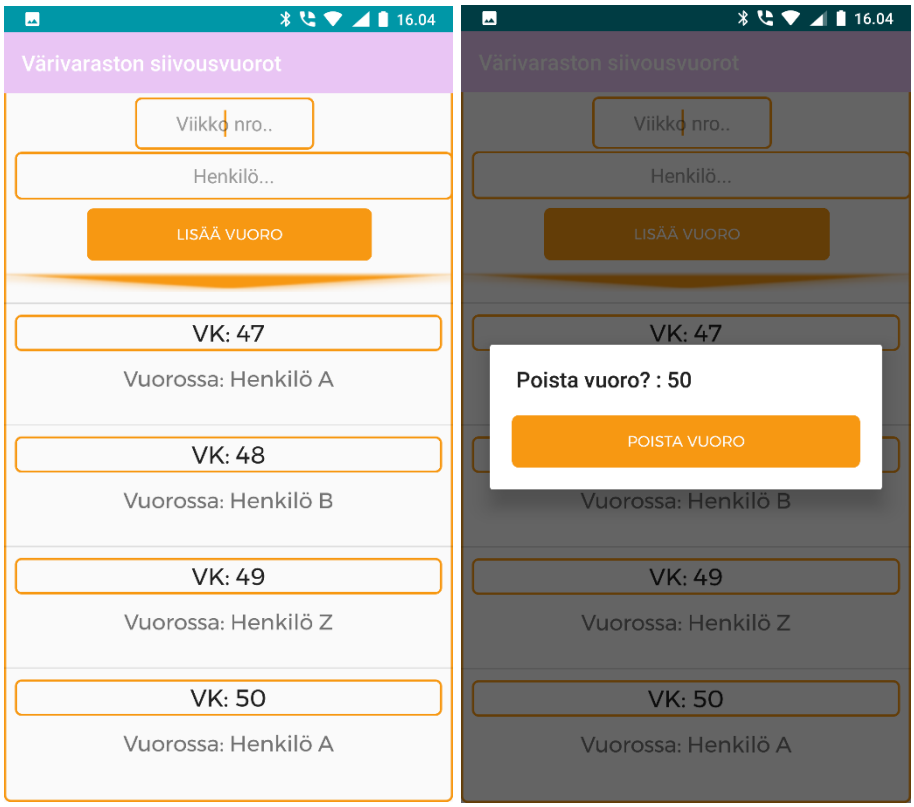
Tilauslistojen haku toimii täysin samalla tavalla kuin väriresepteissäkin, mutta ne haetaan Firebase-tietokannan eri säikeistä ja eri laajuisina. Hakutulos perustuu siihen, mitä tietoja tilauskohteesta on ollut tarpeellista tallentaa. Hakuarvoiksi on määritelty tilattavan tuotteen nimitys, värisarja ja tilauspäivämäärä (kuva 34).



KUVA 34. Tilausten haku.

5.6 Vuorolista

Sovellukseen on lisätty vuorolista, johon viikoittaisen vastuuhenkilön voi merkitä. Lista toimii samalla periaatteella ja toteutuksella kuin värinlisäys- ja tilauslista -toiminnot. (kuva 35)



KUVA 35. Vuorolista ja vuoron poistaminen.

6 TULOKSET

Lopputuloksena syntyi sovellus, jolla käyttäjä voi kirjata värireseptinsä sekä hakea olemassa olevia reseptejä. Käyttäjä voi myös täyttää tilauslistaan tilauksensa sekä etsiä aikaisempia tilauksia päiväyksen, tuotteen tai värisarjan perusteella. Sovelluksen käyttäjä voi rekisteröityä sovelluksen käyttäjäksi sekä vaihtaa salasanansa tai poistaa käyttäjätilinsä. Lisäksi käyttäjä voi merkitä ja poistaa värivaraston siivousvuoroja. Sävytys-osiossa puolestaan käyttäjä voi hakea sopivanlaisen sävyn RGB- ja HEX-arvoja.

Sovellus on helppokäyttöinen, se ohjaa käyttäjäänsä täyttämään vaaditut kentät ja sen toiminnot on keskitetty yhteen näkymään, jotta ne ovat siitä mahdollisimman helposti nähtävillä. Näiden lisäksi täytettävissä kentissä on vinkki siitä, mitä kenttään täytyy täyttää. Jos käyttäjä on täyttänyt kentän väärin tai se on täyttämättä, sovellus lähettää käyttäjän näyttöön virheviestin. Eri listoista on tehty helppolukuisia, ja ne on otsikoitu selkeästi, jotta käyttäjän on helppo nähdä, mitä listalle sisältyy. Näiden lisäksi myös tietokannasta haettuun dataan on lisätty teksti, mitä haettu arvo edustaa, jotta haettu data olisi myös helppolukuisessa muodossa.

Jatkokehityksessä sovellukseen tuli vielä lisää hakuvaihtoehtoja. Listauksiin tuli järjestysnumero, jolla erotetaan toisistaan samalla arvolla määritetyt kohteet. Muutoksia tuli myös moneen layouttiin, sillä eri resoluutioiset laitteet näyttävät komponentit eri tavoin. Ohjelmaa myös muokattiin niin, että tilauslistoihin voi merkitä tiedon siitä, missä vaiheessa tilaus etenee.

7 POHDINTA

Opinnäytetyön aiheena oli sovellus, jonka tarkoituksena on selkeyttää ja helpottaa värivaraston käyttöä ja hallintaa yhdistämällä käytettyjä menetelmiä yhteen sovellukseen. Tuloksena syntyi myös sen mukainen sovellus.

Aihe oli mielenkiintoinen, sillä työelämässä on tullut vastaan paljon sellaisia asioita, joita voisi digitalisoida. Oikein toteutettuna digitalisointi edesauttaa tiedon kulkemista kaikille osapuolille ja voi pienentää virheiden riskejä monella tavalla. Esimerkiksi, jos sovelluksen käyttäjä tekee värireseptin, jonka arvot poikkeavat toisen tekemästä vastaavasta sävystä, voidaan nopeasti reagoida siihen, onko toinen niistä väärä sävy. Toisaalta taas tilauslistoista, voi laskea keskimääräistä kulutusta ja näin ennakoida varsinaisia tilauksia etukäteen. Tämä edesauttaa välttämään tuotantokatkoksia ja toisaalta myös turhia tilauksia, jos jotakin tuotetta kuluu harvoin. Sovellukseen sisältyvä sävyn määritys RGB- ja HEX -arvoilla puolestaan voi auttaa käyttäjää värin määrityksessä, mutta koska tässä toteutuksessa se on varsin suppea, näkisin sen tulevaisuudessa hyvänä kehityskohteena.

Mielestäni tavoitin opinnäytetyöhön asetetut tavoitteet ja lopputuloksena oli toimiva sovellus, joka ajaa asiansa. Pieniä parannuksia ja jatkokehitys ideoita ilmeni vielä loppumetreillä, ja näillä näkymin kehitystyötä jatketaan käyttöönoton jälkeenkin. Tämä oli mielestäni hyvin mielenkiintoinen ja haastava projekti. Toivon, että tästä sovelluksesta on sekä työnantajalle että työntekijöille iloa ja hyötyä.

LÄHTEET

1. Platform Architecture. 2018. Android Developers. Saatavissa: https://developer.android.com/guide/platform/. Hakupäivä 18.11.2018.

2. Android software development. 2018. Wikipedia. Saatavissa: https://en.wikipedia.org/wiki/Android_software_development. Hakupäivä 18.11.2018.

3. Android Studio. Wikipedia. Saatavissa: https://en.wikipedia.org/wiki/Android_Studio. Hakupäivä 18.11.2018.

4. Firebase Real Time Database. 2018. Firebase. Saatavissa: https://firebase.google.com/docs/database/ Hakupäivä 18.11.2018.

5. Firebase Authentication. 2018. Firebase. Saatavissa: https://firebase.google.com/docs/auth/. Hakupäivä 18.11.2018.