VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Mika Leppiaho

ANDROID-SOVELLUSKEHITYS

Tekniikka
2020

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Mika Leppiaho
Opinnäytetyön nimi	Android-sovelluskehitys
Vuosi	2020
Kieli	suomi
Sivumäärä	43
Ohjaaja	Pirjo Prosi

Opinnäytetyön tarkoituksena on tutustua natiivin Android-sovelluksen kehittämiseen. Työssä toteutetaan sovellus, jolla käyttäjä voi ilmoittautua joukkueensa tuleviin tapahtumiin.

Tarkastellaan työkaluja, kirjastoja ja järjestelmiä, joilla voidaan luoda mobiilisovellus Android -käyttöjärjestelmälle. Tutustutaan autentikointiin käytettävää OAuth2 -protokollaa ja tunnistautumiseen käytettävää OpenID -protokollaa ja hyödynnetään näitä Amazon User Pool palvelun käyttämään tunnistautumiseen.

Katsomme työssä, kuinka saadaan sovellus tekemään HTTP - kyselyitä ja vastaanottamaan dataa REST -palvelimelta käyttäen Retrofit 2 -kirjastoa. Tutkimme, kuinka rakennetaan käyttöliittymä usealle kyselylle hyödyntäen otsikkotietoja.

Käyttöliittymä rakennetaan käyttäen Java -ohjelmointikieltä, Googlen virallista IDE:tä Android Studiota ja Android -kehyksen komponentteja ja tarkastellaan kuinka näitä hyödyntää sovelluksessa.

Avainsanat sovelluskehitys, autentikointi, Android, mobiilisovellus

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Mika Leppiaho
Title	Development of an Android Application
Year	2020
Language	Finnish
Pages	43
Name of Supervisor	Pirjo Prosi

The subject of this thesis is the building of a mobile application for Android operating system. The thesis deals with techniques needed to create applications that can communicate with a REST server using API queries.
Authentication with the software was done using Amazon Cognito User Pools. The underlying protocols are used for token-based authentication and identification was also studied.

The application was programmed using Java with Googles official IDE Android Studios. The building of the user interface with activities and layouts were investigated, as well as how the user experience is built with Android framework components.

Thesis work ended with the application successfully sending HTTP requests, receiving data from the application server, and finally processing the received data into readable form that the user can interact with.

Keywords Android, mobile application, Java and Authentication

SISÄLLYS

KUVIOLUETTELO

TAULUKKOLUETTELO

LIITELUETTELO

1 JOHDANTO

Tarkoituksena on rakentaa verkkosovellukselle mobiili käyttöliittymä, jonka avulla käyttäjä voi ilmoittautua urheilujoukkueensa tapahtumiin, kuten otteluihin ja harjoituksiin. Ideana on luoda käyttöliittymä, joka mahdollistaa nopean käytön, että käyttäjän ei tarvitse erikseen hakea puhelimen selaimella ilmoittautumistoimintoa.

Sovellus luodaan Android -käyttöjärjestelmälle johtuen Google Play Storen kevyemmistä vaatimuksista, jotta saadaan sovellus helpommin julkaistua. Työssä tutkitaan natiivia Android kehitystä ja tutustutaan Android -kehykseen ja käytössä oleviin komponentteihin.

Vaikka mobiilille sovelluskehitykselle on jo olemassa useita eri työkaluja ja järjestelmäriippumattomia kehitysvaihtoehtoja, niin työssä valittiin natiivi Android -kehitys olemassa olevan dokumentaation määrän vuoksi ja Java -kieli, jotta vältyttäisiin uuden syntaksin opettelulta.

Ohjelmointityöhön eli ohjelmistokoodin kirjoitukseen on valittu Android Studio, koska tämä on Googlen virallinen IDE Android sovelluskehitykselle.

Työssä olennaiseen toimintaan, eli tietokannan päivittämiseen tarvitaan REST API-kyselyitä, joiden hallintaan valittiin Retrofit 2 niminen kirjasto, jossa on kaikki tarvitsemamme ominaisuudet. Verkkosovellus käyttää tunnistautumiseen Amazon User Pool nimistä palvelua, joka toimii nykyaikaisten autentikointitunnistautumisien mukaan. Näihin tutustutaan työn aikana tarkemmin, jotta ymmärretään mitä ohjelman taustalla tapahtuu.

2 ANDROID -KÄYTTÖJÄRJESTELMÄ

Android on yleinen avoimeen lähdekoodiin pohjautuva käyttöjärjestelmä mobiililaitteille. Käyttöjärjestelmä pohjautuu Linuxiin. Android -sovelluksia voidaan rakentaa esimerkiksi yhdistämällä Java -ohjelmointikieltä ja XML -tiedostoja. Muitakin tapoja on, kuten React Native tai ohjelmointikieli Kotlin. Työssä keskitytään vain natiiviin kehitystyöhön käyttäen Javaa.

2.1 Arkkitehtuuri

Android -alusta koostuu eri tasoista, joita hyödyntämällä voidaan rakentaa sovellus. Taso määritellään tietotekniikassa siten, että pinon tai ns. stackin alin osa pystyy toimimaan ilman sen yläpuolella olevia osia, mutta ilman alla olevia tasoja, kyseinen taso ei pysty operoimaan. Kuva 1 esittää Android -järjestelmän eri osat. /3/

2.1.1 Linux kernel

Android on rakennettu Linux kernelin päälle. Kerneli tarjoaa ajureita laitteistolle, verkkotyöskentelylle, prosessien hallinnalle ja tiedostojärjestelmälle. Androidin ja tavanomainen Linux -kerneli eroavat toisistaan hieman. Tärkeimmät toiminnalliset ominaisuudet, jotka Androidiin on lisätty, ovat Binderit ja paranoid networking. Binder toteuttaa prosessienvälistä kommunikaatiota ja tietoturvamekanismeja. /1/

2.1.2 Hardware Abstraction Layer (HAL)

HAL on abstrakti taso, jonka tarkoitus on piilottaa fyysisen raudan erot, jotta kernelin koodia ei tarvitse muokata alustalle sopivaksi. HAL implementoi rajapinnan raudalle. Kun ohjelmisto tekee API -kyselyn laitteen rautaan (kamera etc.), lataa Android -järjestelmä kyseisen komponentin kirjaston moduulin. /3/

2.1.3 Android Runtime (ART)

Android Runtime (ART) on virtuaalilaite, joka on luotu Android projektia varten. ARTia käytetään Android -laitteilla applikaatioiden suorittamiseen. Android kehityksessä ohjelmakoodi käännetään .dex (**D**alvik **Ex**ecutable) -tiedostoon. Siinä missä JVM kääntää Java -koodin useaan .class -tiedostoon, niin applikaatiot

käännetään normaalisti yhteen .dex -tiedostoon. DEX -formaatti voi kuitenkin vain käsitellä 65,535 metodia, joten suuremmissa applikaatioissa joudutaan käyttämään useampaa .dex -tiedostoa. Androidin 4.4 ”KitKat” -versiossa ART tuotiin esille vaihtoehtona Dalvik VM:lle ja versiossa Android 5.0 ”Lollipop” Dalvik korvattiin kokonaan. /3/

2.1.4 Natiivit C/C++ kirjastot

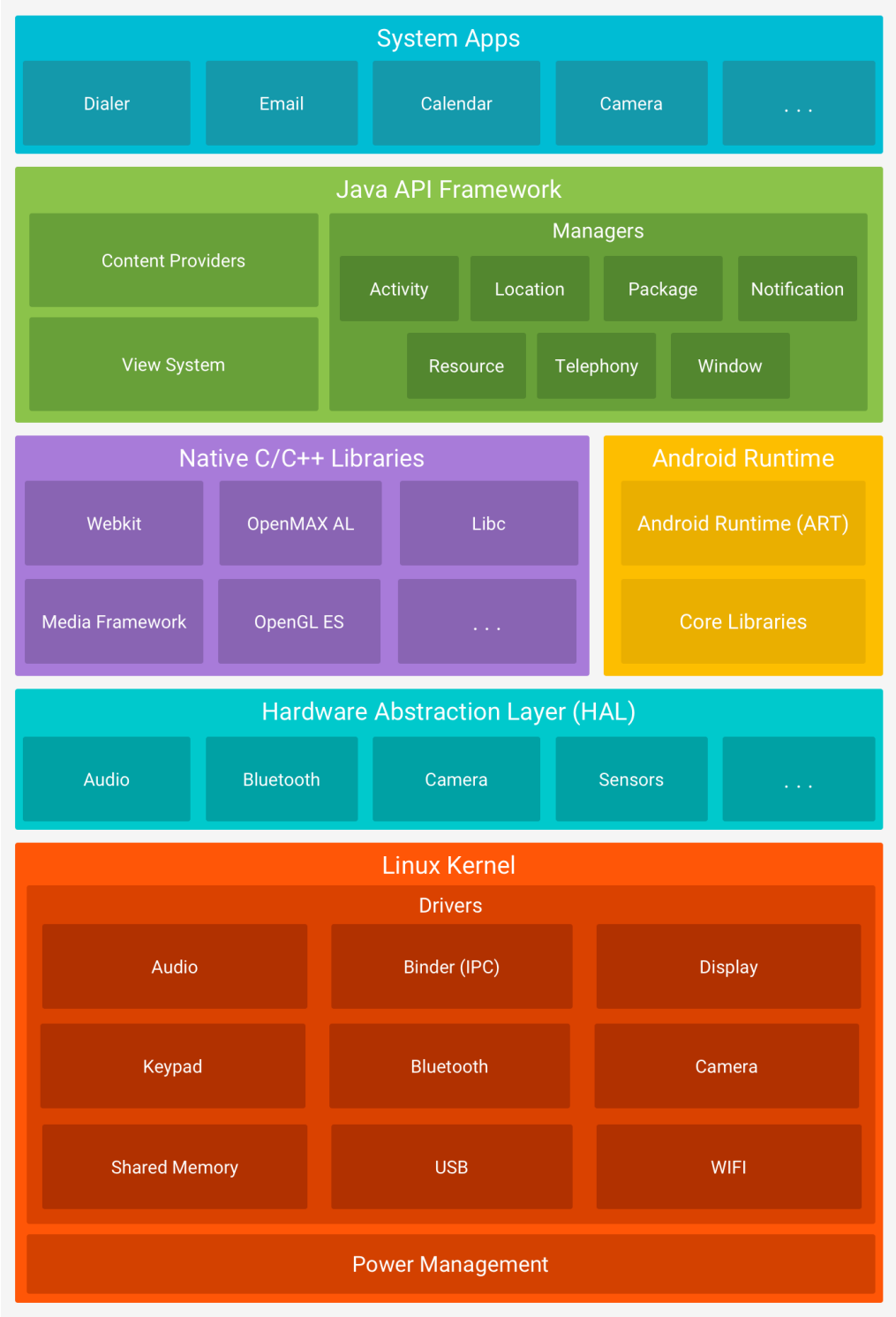
Useat Androidin käyttämät järjestelmäkomponentit ja palvelut on rakennettu natiivista koodista, joka tarvitsee käyttöönsä kirjastoja, jotka on kirjoitettu C:llä ja C++ -ohjelmointikielillä. Sovellusalusta tarjoaa Java kehys API:n, jonka avulla voidaan saada applikaatioille näiden natiivien kirjastojen toiminnallisuutta. /3/

2.1.5 Java API Framework

Toiseksi korkein kerros pinossa on Java API Framework. Android -käyttöjärjestelmän koko ominaisuuskokoelma on saatavilla Java -kielellä kirjoitetun API:n kautta. Näillä API -kutsuilla voidaan helpottaa applikaatioiden rakentamista käyttämällä järjestelmäkomponentteja. Komponenteilla hallitaan sovelluksen elinkaarta, näkymiä ja käyttöliittymää. /3/

2.1.6 Applikaatiot

Pinon ylimpänä on sovellukset. Nämä pitävät sisällään tekstiviestit, sähköpostit, puhelin applikaatiot, selaimet jne. Vaikka Android tulee muutamalla esiasennetulla sovelluksella, esimerkiksi sähköpostin tai tekstiviestin lähettämistä varten, niin sovelluksia ja esimerkiksi virtuaalinäppäimistöjä voi käyttäjä vaihtaa. Muutama poikkeus on olemassa, kuten järjestelmän asetukset. Esiasennetut sovellukset ovat käyttäjiä varten, mutta myös sovelluskehittäjille, jotka voivat kutsua näitä olemassa olevia sovelluksia omissa sovelluksissaan. Esimerkiksi puhelimelle ei tarvitse tehdä kahta sähköpostia käsittelevää sovellusta. /3/



Kuva 1. Android arkkitehtuuri. /3/

3 AUTENTIKOINTI

3.1 OAuth 2.0

OAuth on Internet Engineering Task Force (IETF):n kehittämä autentikointikehys, joka on noussut standardiksi verkkosovelluksien autentikoinnissa. Auktorisointi sallii kolmannen osapuolen applikaatioilla rajatun pääsyn HTTP -palvelulle. OAuthin autentikoinnin pääpiirteenä on digitaalinen tunnistin, jota kutsutaan tokeniksi. Protokolla määrittelee neljä roolia, joiden välillä autentikointi tapahtuu. /4/

- Resource Owner (Käyttäjä)
    - Käyttäjä, joka omistaa datan, johon applikaatio haluaa saada käyttöoikeuksia. Voidaan myös määrittää loppukäyttäjänä, joka antaa luvan suojatun datan käyttöön. /4, s. 5/
- Resource Server (Resurssipalvelin)
    - Suojattua dataa ylläpitävä palvelin. Tämä on myös vastuussa vastaamaan ja vastaanottamaan pyyntöjä käyttäen access tokenia. /4, s. 5/
- Client (Asiakasohjelma)
    - Applikaatio, joka tekee resurssipyyntöjä käyttäjän puolesta käyttäjän valtuutuksella. /4, s. 5/
- Authorization Server (Auktorisointipalvelin)
    - Palvelin, joka toimittaa clientille access tokenin onnistuneen käyttäjän autentikoinnin jälkeen. /4, s. 5/

3.1.1 Authorization Grant

Authorization Grant toimii valtakirjana, joka edustaa loppukäyttäjän auktorisointia suojatun datan pääsyä varten. Ohjelma käyttää tätä tietoa pyytäessä auktorisointipalvelimelta Access Tokenin. OAuth on spesifioinut neljä eri lupatyyppiä: /4, s. 7/

- Authorization Code
    - Valtuutuskoodi hankitaan käyttämällä auktorisointipalvelinta välittäjänä sovelluksen ja sovelluksen loppukäyttäjän välillä. /4, s. 7/
- Implicit
    - Tämä on yksinkertaistettu kulku, joka on optimoitu skriptauskieliä, kuten JavaScript, käyttäviä selaimia varten, jossa sovellus saa access tokenin suoraan. Tässä auktorisointipalvelin ei autentikoi sovellusta. /4, s. 7/
- Loppukäyttäjän salasana
    - Käyttäjän käyttäjänimi ja salasana käytetään access tokenin saamiseen. /4, s. 8/
- Ohjelmavaltuutus
    - Ohjelmalle voidaan antaa auktorisointi siinä tapauksessa, kun ohjelma tarvitsee access tokenin sovelluksen tarpeeseen, kuten logon tai käyttäjästatistiikan päivittämiseen. /4, s. 8/

3.1.2 Scope

Scope on mekanismi OAuth -protokollassa, jolla voidaan rajoittaa sovelluksen pääsyä käyttäjän tiliin. Sovellus pyytää käyttäjältä yhden tai useamman scopen, jolloin käyttäjä voi antaa sovellukselle luvan. Sovellukselle myönnetty access tokenilla oleva pääsy tiliin on rajattu annetun luvan mukaan. /4, s. 22/

3.1.3 Abstrakti OAuthin työnkulku

Kuvassa 2 on kuvattu OAuth -protokollan työnkulku yleisesti. Alla lueteltuna vielä askeleet selitettynä. /4, s. 6/

A. Authorization Request

   Ohjelma pyytää käyttäjältä auktorisointia resursseja varten /4, s. 6/

B. Authorization Grant

   Ohjelma saa loppukäyttäjältä auktorisointisuostumuksen. OAuth protokolla tarjoaa neljä eri toimintoa auktorisoinnin myöntämiseen. /4, s. 6/

C. Authorization Grant to Authorization Server

   Ohjelma pyytää access tokenin autentikointipalvelimelta tarjoten käyttäjän syöttämän autentikointitiedon. /4, s. 6/

D. Access Token

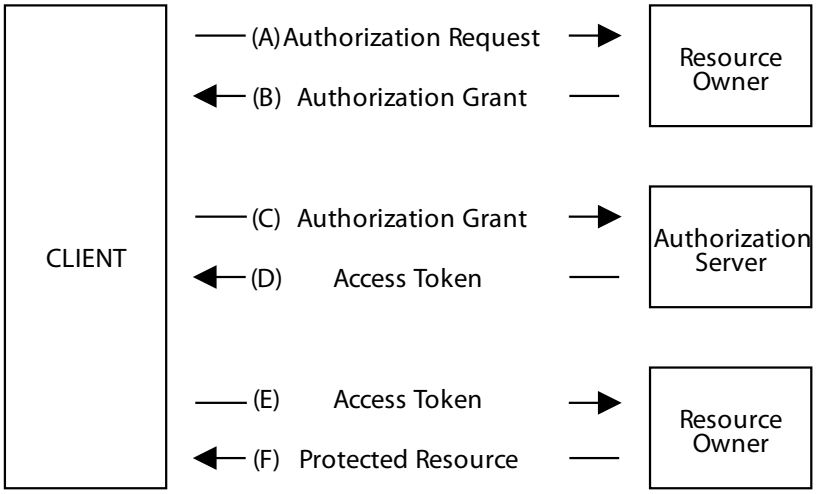
   Autentikointipalvelin validoi autentikoinnin ja autentikoinnin ollessa onnistunut, palauttaa palvelin access tokenin. /4, s. 6/

E. Access Token to Resource backend

   Ohjelma lähettää pyynnön palvelimelle pyytäen suojattua resurssia access tokenin kanssa. /4, s. 6/

F. Protected Resource

   Palvelin validoi Tokenin ja palauttaa pyydetyn datan. /4, s. 6/

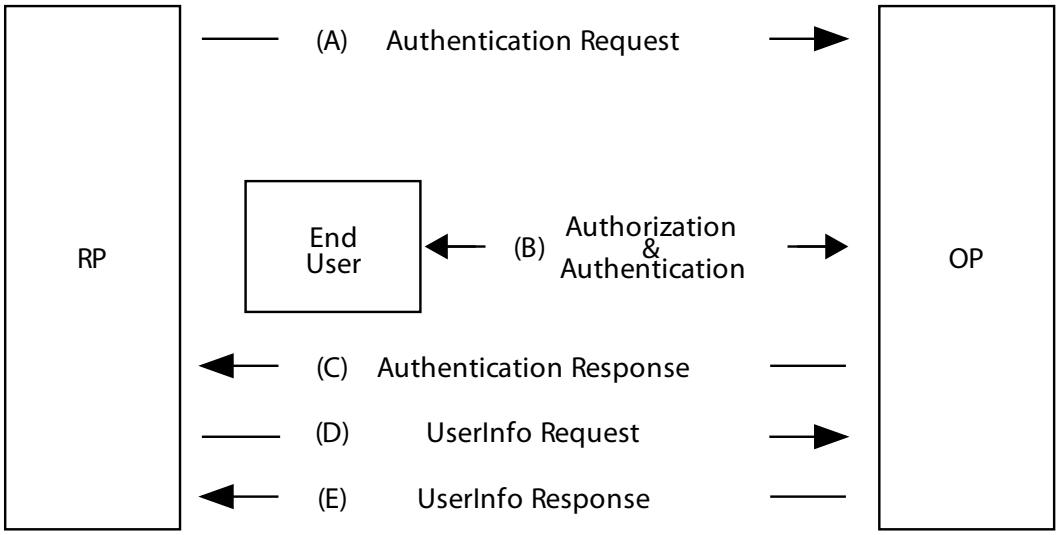


Kuva 2. OAuth 2.0 -työnkulku /4/

3.2 OpenID Connect

OAuth ei tarjoa identifiointia. Tähän tarkoitukseen on tehty Open ID Connect (OIDC) -protokolla, joka toimii identifiointikerroksena OAuthin päällä. Sovellus voi varmistaa loppukäyttäjän identiteetin auktorisointipalvelimen tekemän todennuksen pohjalta käyttäen OpenID:tä.

OpenID:tä käyttävää autentikointipalvelinta kutsutaan OpenID Provideriksi (OP) ja sovellusta kutsutaan Relying Partyksi (RP) /5/



Kuva 3. OpenID Connect -protokollan työnkulku yleisesti /5/

OpenID:n toiminta yleisesti on kuvattu kuvan 3 mukaisesti seuraavalla tavalla: /5/

A. Sovellus eli RP lähettää pyynnön autentikointipalvelimelle eli OP:lle
B. OP autentikoi loppukäyttäjän ja saa valtuutuksen
C. OP palauttaa vastauksena alkuperäiseen pyyntöön ID- ja Access Tokenin
D. Sovellus voi nyt pyytää käyttäjätietoja
E. Autentikointipalvelin palauttaa pyydetyt tiedot.

OpenID autentikointi voidaan suorittaa kolmella eri tavalla. Voidaan käyttää Authorization Code Flow, Implicit Flow, tai Hybrid Flow. Nämä autentikointikulut määrittelevät kuinka ID- ja Access Tokenit toimitetaan loppukäyttäjälle. Toiminta eri tavoille on kuvattu taulukossa 1. /5/

Taulukko 1. OpenID Connect autentikointikulut /5/

	Authorization code flow	**Implicit Flow**	**Hybrid flow**
Kaikki tokenit palautetaan auktorisointi päätepisteeltä	ei	kyllä	ei
Kaikki tokenit palautetaan token päätepisteeltä	kyllä	ei	ei
Tokenia ei näytetä loppu-käyttäjälle	kyllä	ei	ei
Sovellus voidaan autenti-koida	kyllä	ei	kyllä
Refresh Token mahdollista	kyllä	ei	kyllä
Kommunikaatio yhdellä kierroksella	ei	kyllä	ei
Eniten kommunikointia palvelimelta palvelimelle	kyllä	ei	vaihtelee

3.2.1 TOKEN

Token on tekstinpätkä, jolla voidaan valtuuttaa resurssien käyttö sovelluksella. Tokeniin tallennetaan käyttäjätietoja, joita verrataan palvelimella olevaan dataan ja käyttäjä auktorisoidaan näillä tiedoilla. Yleisin token malli on JSON Web Token.

3.2.2 JSON Web Token

JSON Web Token on standardi, joka määrittelee kompaktin vaatimusten esittely-formaatin. Data voidaan todentaa ja siihen voi luottaa, koska se on digitaalisesti allekirjoitettu joko HMAC-, RSA- tai ECDSA -algoritmeilla. JWT koostuu

kolmesta osasta; Otsikko (Header), tietosisältö (Payload) ja allekirjoitus (Signature). /6/

Otsikko koostuu yleensä algoritmista ja tyyppitiedosta. /8/

```
{
"alg": "HS256",
"typ": "JWT"
}
```

Tietosisältö pitää sisällään dataa, kuten lähettäjä tai erääntymisaika. Työssä käytettävän Access Tokenin sisältö on seuraava /7/

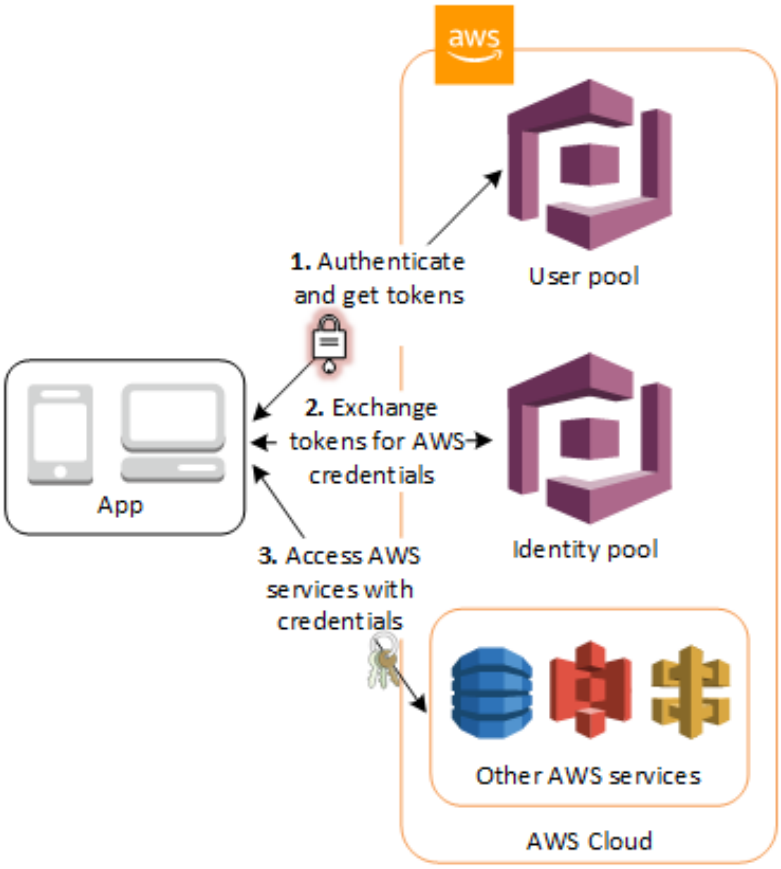
```
{
  "sub":, //subject Henkilö johon tokeni viittaa
  "device_key": ,
  "event_id": ,
  "token_use": "access",
  "scope": "aws.cognito.signin.user.admin",
  "auth_time": 1580935220,
  "iss": ” https://cognito-idp.eu-central-1.amazonaws.com..., //Is-
suer Tokenin tekijä ja allekirjoittaja
  "exp": 1580938820, //Expiration Umpeutumisaika
  "iat": 1580935220, //Issued at luontiaika
  "jti": , //JWT ID Tokenille yksilöllinen ID
  "client_id": ,// Sovellukelle rekisteröity id
  "username": //käyttäjänimi
}
```

Identity provider ottaa tietosisällön ja allekirjoittaa sen headerissä spesifioidun avaimen kanssa ja lisää sen JWT:n perään. Se verifioi, että payload tulee identity providerilta, eikä sitä ole muokattu. /8/

3.3 Amazon Cognito

Amazon Cognito on Amazonin tuottama autentikointipalvelu, joka pohjautuu OAuth- ja OpenID Connect -protokollaan mahdollistaen rekisteröitymisen, kirjautumisen ja pääsyn hallinnan verkko – ja mobiiliapplikaatioita varten. Kuvan 4 mukaisesti Cognito on jakautunut kahteen komponenttiin, User Pools ja Identity Pool. Identity Poolilla voidaan autentikoida ja auktorisoida pääsy Amazonin AWS-

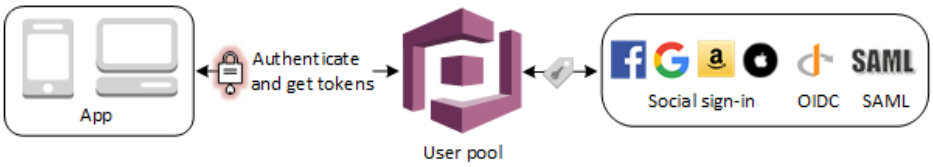
palveluille. Nämä palvelut ovat epärelevantteja tässä työssä, keskitymme vain User Pooliin. /9/



Kuva 4. Amazon Cogniton toimintakuva /9/

3.3.1 User Pool

Amazon Cognito User Pool on osa Amazon Cognitoa ja tarjoaa käyttäjähakemiston (**Kuva 5.**). /10/



Kuva 5. User Poolin toimintamalli /10/

3.4 AWS Android SDK

AWS Android SDK on kolmannen osapuolen kirjasto, joka pitää huolen autentikoinnista, tunnistautumisesta ja yhteydestä autentikointipalvelimelle. Kirjastot otetaan käyttöön ensin määrittelemällä Gradlelle kirjaston riippuvuudet lisäämällä osoite build.gradle -tiedostoon. /11/

```
//AWS riippuvuudet
implementation 'com.amazonaws:aws-android-sdk-ddb:2.15.2'
implementation('com.amazonaws:aws-android-sdk-mobile-client:2.15.+@aar')
{ transitive = true }
```

Jotta saadaan autentikointi toimimaan Amazon Web Servicen kautta, tarvitaan myös awsconfiguration.json -tiedosto, johon määritellään autentikointipalvelimen ID. Käytämme AWS sdk:ta siten, että tarkistamme käyttäjän kirjautumistilan ja annamme toimintaohjeet käyttäjälle riippuen tilasta. Jos käyttäjä on kirjautuneena sisään, voi hän jatkaa sovelluksen käyttöä ja jos hän on kirjautunut ulos, pitää käyttäjän kirjautua sisään ennen kuin voi jatkaa sovelluksen käyttöä. /11/

AWSMobileClient tarjoaa API:n Amazonin verkkopalveluja varten, kuten JWT Tokenin toimittamisen, kirjautumisen tai uloskirjautumisen ja käyttäjän kirjautumistilan seurannan. /12/

Tokenint voidaan hakea järjestelmään tunnistautumista varten seuraavilla metodeilla. /12/

AccessToken:

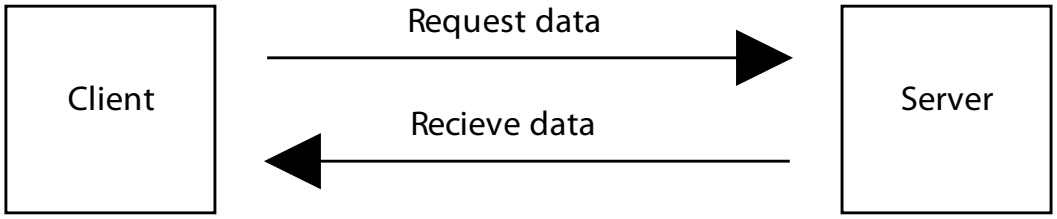
```
AWSMobileClient.getInstance().getTokens().getAccessToken().getTo-
kenString();
```

IdToken:

```
AWSMobileClient.getInstance().getTokens().getIdToken().getTokenString()
```

4 WEB API

Web Application Programming Interface on ohjelmointirajapinta, jonka avulla voidaan tarjota dataa toisille ohjelmille ja käyttäjille verkon ylitse. Työssä käytetään verkon yli tapahtuvaa kommunikaatiota Android -järjestelmän ja palvelimen välillä, jolla päivitetään tietoja loppukäyttäjälle. Tähän kommunikaatioon käytetään API -kyselyitä, johon palvelin vastaa kyselyssä annetun datan mukaan. **Kuva 6.** esittää toimintaperiaatteen yksinkertaistettuna.



Kuva 6. Web APIn toimintaa kuvaava kaavio

4.1 REST

REST tulee sanoista Representational State Transfer ja on arkkitehtuurimalli hypermedian jakelujärjestelmille, jonka on Roy Fielding määritellyt vuonna 2000. Hypermedialla tarkoitetaan siis tekstiä, kuvia, ääntä jne. Protokolla tiedonsiirtoon on nykymaailmassa lähes aina HTTP ja tätä samaa protokollaa käytetään myös työssä. RESTillä on kuusi ohjaavaa rajoitusta, jotka pitää tulla toteen, että käyttöliittymä voidaan nimetä **RESTfuliksi**. Rajoitukset on lueteltu seuraavasti: /23/

1. Asiakas – palvelin

   Tarkoittaa sitä, että järjestelmän on koostuttava clientistä, joka pyytää ja antaa tietoa ja palvelimesta, joka palauttaa ja tallentaa tietoa. /23/

2. Tilattomuus

   Tilattomuudella haetaan sitä, että pyyntö voidaan suorittaa ilman, että palvelimelle tallennetaan käyttäjän tilatietoja kyselyiden suorittamista varten. /23/

3. Välimuisti

Välimuistirajoitukset vaativat, että pyynnön vastauksesta on nähtävä, voidaanko vastaus tallentaa välimuistiin. Näin nähdään, voidaanko vastaus tallentaa välimuistiin myöhempää käyttöä varten. /23/

4. Yhdenmukainen rajapinta

   REST määrittelee neljä rajapinta rajoitusta, joilla saadaan järjestelmä yhteneväiseksi: Resurssien identifiointi, resurssien manipulointi, itseselitteiset viestit ja hypermedia applikaation tilakoneena. /23/

5. Kerroksittainen järjestelmä

   Järjestelmän tulee tarjota hierarkkisesti yläpuolellaan oleville tasoille palveluita ja hyödyntää tason alapuolella olevia tasoja siten, että jokainen komponentti pystyy näkemään vain yhden kerroksen ylä- ja alapuolella olevat tasot. /23/

6. Ladattava koodi

   Viimeinen rajoite on valinnainen. Ladattavalla koodilla tarkoitetaan järjestelmässä mahdollisuutta ladata palvelimelta koodia ja skriptejä. /23/

Avainasemassa REST -palvelussa on palvelimella oleva data, jota käytetään resurssina. Nyrkkisääntönä voidaan sanoa, että kaikki se informaatio, joka voidaan nimetä, toimii resurssina. Resurssit löydetään resurssitunnisteella. RESTful api ei palauta varsinaisesti mediaa vaan mediatyypin, joka määrittelee, kuinka resurssi esitetään. RESTful API näyttää hypertextiltä. Jokainen data kantaa mukanaan osoitteen joko selvästi linkkinä, ID -attribuuttina tai implisiittisesti (esim. mediatyypin määrittelynä)

4.2 Retrofit 2

Retrofit tiivistää REST APIn Java rajapinnaksi. Retrofitillä voidaan rakentaa HTTP -pyyntöjä headereillä ja URL -parametreillä. Kirjasto hoitaa yhdistämisen, keskeytykset, SSL -kättelyt ja asynkroniset tehtävät automaattisesti. /13/

4.2.1 Gradle -riippuvuudet

Retrofit 2:n kirjastot pitää tuoda projektille. Käytetään Gradlea riippuvuuksien tuontiin ja asentamiseen. /13/

```
implementation 'com.squareup.retrofit2:retrofit:2.7.1'
```

Retrofit ei oletuksena kykene parsimaan JSON -vastauksia, joten tuodaan ohjelmaan vielä Gson -konvertteri, jotta voidaan käsitellä vastauksia palvelimelta. /13/

```
implementation 'com.squareup.retrofit2:converter-gson:2.7.1'
```

4.2.2 HTTP -pyynnön määrittely

Rajapintaluokassa määritellään pyynnöt seuraavalla koodilla. Annotaation @GET jälkeen on annettu API -päätepiste, joka palauttaa vastauksena pyydetyn datan.

```
Public interface ApiClient {
        @GET("{user}/repos")
        Call<List<ApiRepo>>
                reposForUser(@Path("user")String user);

        @POST(“events”)
        Call<Events> createEvents(@Body Events events);
}
```

`@GET()`-annotaatio määrittelee APIn päätepisteen ja pyynnön tyypin. Retrofit tukee seuraavia HTTP -metodeja: GET, POST, PUT, DELETE, PATCH ja HEAD. Seuraavaksi on annettu metodin palautustyyppi `Call<List<ApiRepo>>`, metodin nimi `reposForUser()` ja metodille on syötetty parametrejä `@PATH` –annotaatiolla, joita voidaan hyödyntää kyselyssä, esimerkiksi hakemalla tietyn käyttäjän tiedot käyttäen käyttäjän tunnistenumeroa. /13/

4.2.3 Datan mäppäys käyttäen Retrofit Konvertteriä.

Pyyntöjen vastaukset palautetaan työssä JSON -muodossa. Rajapintaluokassa nimetty `ApiRepo` on luokka, joka pitää sisällään objektin ominaisuudet. Samaan luokkaan määritellään myös tarpeen vaatien Javan getterit ja setterit datan muokkaamista ja käyttöä varten. Alla olevalla koodilla voidaan päivittää id- ja name-

muuttujat vastaamaan palvelimelta saatua dataa, joka voidaan tulostaa aktiviteetilla käyttäjälle näkyviin.

```
class UserByID {
    private String id;
    private String name;

    public String getId() {
        return id;
    }
    public String getName() {
        return name;
    }
}
```

4.2.4 Pyynnön tekeminen

Retrofit -luokka implementoi luodun ApiInterface -rajapintaluokan seuraavasti

```
Retrofit retrofit = new Retrofit.Builder()
       .baseUrl("https://api.restful.com/")
       .build();

ApiInterface service = retrofit.create(ApiInterface.class)

Call<List<ApiRepo>> repos = service.
reposForUser("user");
```

Jokainen kutsu luodusta ApiInterfacesta voi tehdä synkronisen tai asynkronisen HTTP -pyynnön palvelimelle. Synkroninen kutsu jää odottamaan pyynnön valmistumista, jolloin ohjelmaa ei voi käyttää. Asynkroninen pyyntö ei estä ohjelman kulkua. Jos ohjelma vaatii useamman HTTP -pyynnön, on Retrofit -luokan implementointi jokaiselle pyynnölle epäkannattavaa ohjelmistokoodin luettavuuden kannalta. Rivien määrä kasvaa, varsinkin kun lisätään pyyntöihin autentikointikäsittely. Suosituksena onkin luoda erillinen luokka, joka vastaa pyynnön rakentamisesta ja tokenin lisäämisestä. /14 s. 13-16/

4.2.5 Autentikointi

Autentikointi on toteutettu työssä käyttäen OAuth 2 -protokollaa, jolloin tunnistautuminen on tehty token -pohjaisesti. Retrofit tukee tokenilla tunnistautumista.

Token voidaan lisätä pyyntöön Authorization otsikkotiedossa. Tapoja käyttää otsikkotietoja on muutama. Yksi tapa on lisätä otsikkotieto suoraan rajapintaluokkaan.

```
@GET("{user}/repos")
Call<List<ApiRepo>>
        reposForUser(
                @Header(”Authorization”) String token,
                @Path("user")String user);
```

Tapa sopii, jos pyyntöjä ei ole montaa. Mutta tilanteessa, jossa pyyntöjä on useampi, tämä menetelmä aiheuttaa paljon turhan koodin toistoa. Parempi toimintatapa on luoda metodi, joka sieppaa pyynnön ja lisää Autentikointitunnisteelle tokenin ja jatkaa pyynnön suorittamista. /14 s. 83/

```
Request.Builder builder = original.newBuilder()
        .header(“Authorization”, authToken)
        .method(original.method(), original.body());

Request request = builder.build();

return chain.proceed(request);
```

Esimerkki päivittää alkuperäisen pyynnön tunnisteelle tokenin.

5 ANDROID-APPLIKAATIO

5.1 Kehitysympäristö

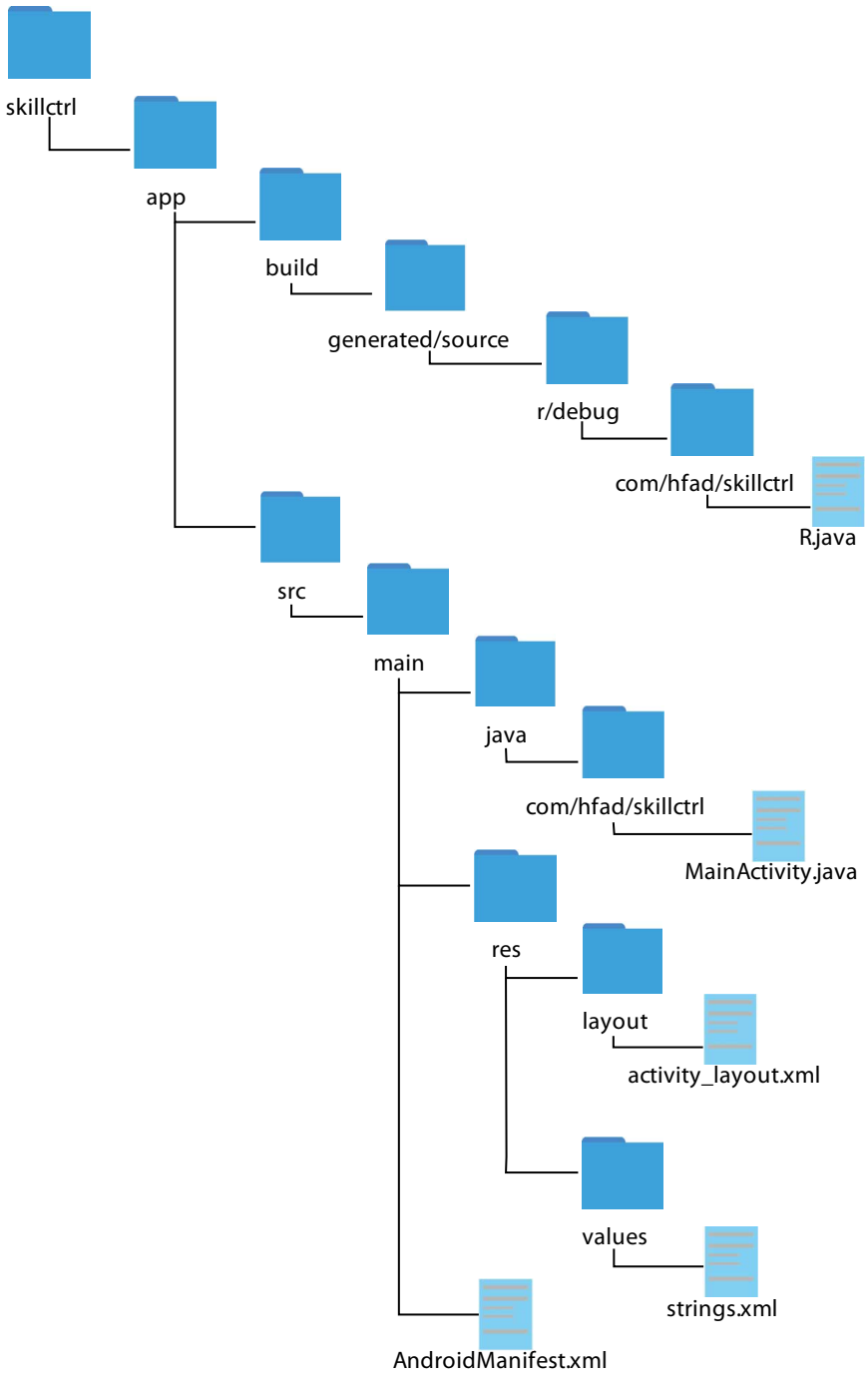
Työ ohjelmoidaan käyttäen Java -ohjelmointikieltä ja käyttämällä Googlen virallisesti tuettua ohjelmistoympäristöä Android Studiota. Android Studio valittiin IDE:ksi Android Studion tarjoamien ominaisuuksien vuoksi, jotka helpottavat sovelluksen rakentamista merkittävästi. Tärkeimpiä ominaisuuksia, joita sovelluksen ohjelmoinnissa käytettiin, on seuraavat: /15/

- Layout editori, jolla voidaan tehdä nopeasti näkymiä käyttäjiä varten. /15/
- Android emulaattori nopeaa testaamista varten. Voimme käyttää emulaattoria testaamaan sovellusta nopeasti eri Android -versioilla ja näytön laajuuksilla. /15/
- Ohjelmakoodin kääntö, joka tapahtuu käyttäen Gradlea. Gradle on käännösautomaatiotyökalu, jota ajetaan JVM:n päällä. /15/
- Koodimalleja tavanomaisille prosesseille, jotka nopeuttavat kehitystyötä. /15/
- AndroidManifest.xml -tiedoston automaattinen päivitys. /15/

Android Studiolla voidaan toteuttaa koko ohjelmiston kehityksen työnkulku työtilan pystyttämisestä ohjelman julkaisuun. /15/

5.2 Android -kansiorakenne

Kuvassa 7 on muutama tärkeä tiedosto ja kansio, joita Gradle tarvitsee ohjelman kääntämiseen ja ohjelman toimimiseen. Rakenne alkaa projektin juuresta. Kaikki tiedostot menevät projektin alle. Android Studio hoitaa kansiorakenteen, joten siihen ei yleensä tarvitse kiinnittää huomiota. Ohjelman kannalta olennainen tiedosto build -kansion alla on R.java, joka hoitaa resurssien hallinnan. Src -kansion alle tulee kaikki kirjoitettu ohjelmakoodi. Res -kansioon tulee kaikki resurssit, kuten kuvat, tekstit ja layoutit, joita ohjelmassa käytetään. /2, s. s.17/



Kuva 7. Standardi Gradle tiedostorakenne /2 s.17/

5.3 Android -komponentit

Android -sovellus koostuu eri komponenteista, jotka toimivat tulokohtana, joista käyttäjä tai järjestelmä saa pääsyn applikaatioon. Erilaisia komponentteja on neljä, joilla on oma erillinen elinkaari. /16/

5.3.1 Aktiviteetti

Käyttäjän näkökulmasta olennaisin komponentti on aktiviteetti ja tähän sidottu layout, joka määrittelee, mitä ruudulla näkyy ja mitä ruudulla tapahtuu. Aktiviteetti siis edustaa yksittäistä ruutua ja pitää yllä tietoa, jota käyttäjä sillä hetkellä tarvitsee. Useimmat applikaatiot pitävät sisällään useamman ikkunan, jolloin applikaatio on tehty useammasta aktiviteetistä. Koska aktiviteetit toimivat itsenäisesti erillään toisistaan, on mahdollista esimerkiksi yhdestä applikaatiosta avata toinen applikaatio tietystä aktiviteetistä. Jotta voimme käyttää aktiviteettejä, pitää nämä rekisteröidä applikaation manifest -tiedostoon. /17/

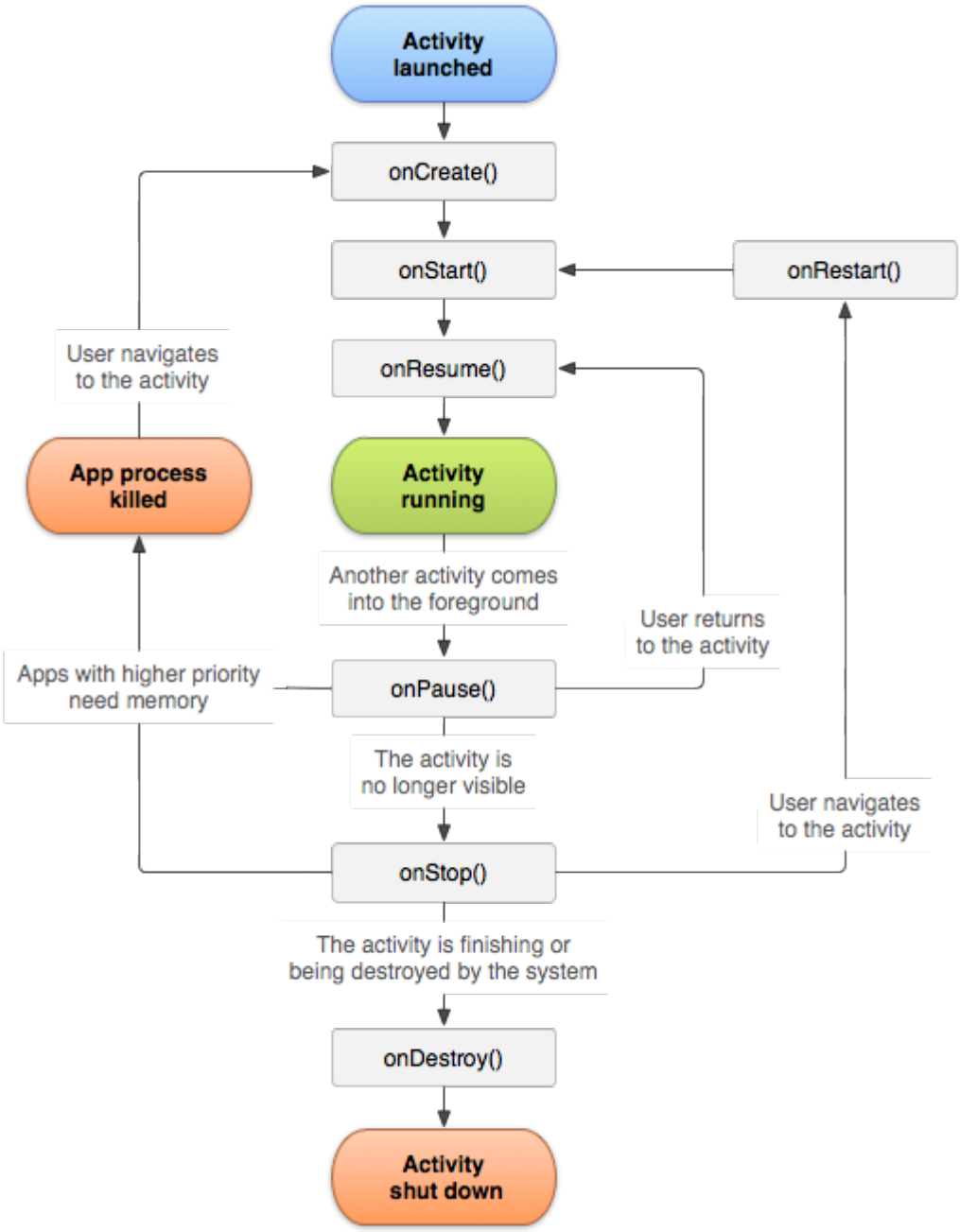
```
<manifest ... >
  <application ... >
      <activity android:name=".ExampleActivity" />
      ...
  </application ... >
  ...
</manifest >
```

Aktiviteetti tulee olemaan elinkaarensa aikana useassa eri tilassa. Näitä tiloja voidaan hallita seuraavilla takaisinkutsuilla. /17/

- `onCreate()`
    - onCreate laukaistaan, kun järjestelmä luo aktiviteetin. onCreate()-metodin sisälle voidaan määritellä logiikkaa, jonka pitäisi käynnistyä vain kerran koko aktiviteetin elinkaaren aikana. /17/
- `onStart()`
    - onStart kutsu aktivoi aktiviteetin käyttäjälle näkyväksi ja applikaatio valmistelee aktiviteetin vastaanottamaan dataa. onStart() -metodissa applikaatio alustaa käyttöliittymän. /17/
- `onResume()`

- Kun aktiviteetti siirtyy resumed -tilaan, aktiviteetti siirtyy etualalle ja järjestelmä kutsuu onResume() -takaisinkutsun. Tämä on tila, jossa käyttäjä voi olla vuorovaikutuksessa ohjelman kanssa ja kaikki aktiviteettiin sidotut komponentit vastaanottavat ON_RESUME -tapahtuman. onResume() -metodiin alustetaan logiikka, joka pitää ajaa aina, kun aktiviteettiin palataan. /17/

- `onPause()`
  - Metodi onPause kutsutaan, kun käyttäjä on poistumassa aktiviteetilta. Metodia voidaan käyttää, kun halutaan pysäyttää operaatioita, jotka ei haluta jatkavan, jos käyttäjällä ei ole aktiviteetti aktiivisena. Metodin toiminta on hyvin nopea eikä välttämättä anna aikaa tehdä tallennusoperaatioita, kuten tietokantapäivityksiä. Nämä on parempi siirtää onStop() -metodille. /17/
- `onStop()`
  - Aktiviteetti on onStop -tilassa, kun aktiviteetti ei ole enää näkyvissä käyttäjälle. Esimerkiksi käyttäjä on vaihtanut aktiviteettia applikaatiossa. onStop() metodissa applikaation pitäisi irrottautua resursseista, joita ei tarvita, kun käyttäjä ei niitä näe. Prosessorille intensiiviset sulkemisoperaatiot kuuluu tehdä myös onStop -metodissa, kuten käyttäjätietojen päivittäminen tietokantaan. /17/
- `onDestroy()`
  - Metodi kutsutaan ennen kuin aktiviteetti tuhoutuu ja metodin pitäisi vapauttaa kaikki resurssit, joita ei ole vielä aikaisemmissa takaisinkutsuissa vapautettu. /17/



Kuva 8. Aktiviteetin elinkaari /17/

5.3.2 Services

Services -palvelulla tarkoitetaan komponenttia, joka ajaa prosesseja taustalla ilman käyttöliittymää. Palveluilla on itsenäiset elinkaaret irrallaan aktiviteeteistä. Tämä mahdollistaa prosessien kulun, kun esimerkiksi aktiviteetti menee onPause -tilaan. Vaikka palveluiden ideana on pyöriä taustalla, on tärkeä huomioida, että oletuksena

palvelut eivät pyöri taustathreadillä, vaan sillä threadillä, jolla kyseinen palvelu luotiin. Palveluita on käytössä kahdenlaisia, *Bound services* ja *Started services.*

Started services kertoo järjestelmälle, että prosessit pidetään päällä, kunnes prosessi on valmis. Bound services ajetaan silloin, kun toinen applikaatio tai järjestelmä haluaa käyttää kyseistä palvelua ja tämä prosessi pidetään käynnissä, kunnes sitä ei enää tarvita. Palvelut implementoidaan ohjelmassa Servicen aliluokkana. /18/

5.3.3 Broadcast receiver

Broadcast receiver on komponentti, joka mahdollistaa järjestelmän toimittamaan tapahtumia applikaatiolle tavanomaisen käyttökulun ulkopuolelta. Komponentti voi vastaanottaa tapahtumia, vaikka sovellus on suljettu. Komponentti itsessään toimii ilman käyttöliittymää, mutta voi esimerkiksi tuottaa popup -ilmoituksen puhelimen näytölle. /16/

5.3.4 Content providers

Content provider on komponentti, jonka kautta toiset sovellukset voivat kysyä dataa jos komponentti sallii kyselyn. Esimerkiksi käyttäjän jokin sovellus voi pyytää puhelimen kontaktilistaa. /16/

5.4 Käyttöliittymä

Käyttäjän käyttöliittymä luodaan layouteilla, jotka on linkitetty aktiviteetteihin AndroidinManifest.xml -tiedostossa. Layout on xml -muotoinen resurssi, jolla voidaan määrittää käyttäjälle näkyvä käyttöliittymä aktiviteetillä. /16/

5.5 Manifest

Manifest kuvailee applikaatiosta tietoja Android käännöstyökaluille, käyttöjärjestelmälle ja Google Playlle. Tiedosto on pakollinen ja pitää sisällään pakollista tietoa, jota sovellus tarvitsee toimiakseen. Manifesti on xml muotoinen tiedosto

projektin lähdekoodin juuressa. XML -tiedoston juurielementissä on määritelty pakkauksen nimi ja applikaatio ID. /16/:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.skillctrl">
```

<uses-permission> elementtiin määritellään myös oikeudet arkaluontoisiin tietoihin, kuten kontaktilistaan tai kameran käyttöön, joita applikaatio haluaa käyttää. /18/:

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

Android.permission.INTERNET pyytää oikeutta yhdistää applikaatio internetiin.

Jokainen applikaatiokomponentti pitää määritellä manifesti -tiedostossa <application> elementtiin. Jokaiselle komponentille pitää määrittää vähintään Java -luokan nimi, joka aktiviteetilla on. /18/:

```
<activity android:name=".AuthenticationActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <action android:name="android.intent.action.VIEW" />

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

Ylläolevassa esimerkissä aktiviteetti AuthenticationActivity on määritelty pääaktiviteetiksi käyttäen elementtiä <action android:name="android.intent.action.MAIN" />. Pääaktiviteetti käynnistetään sovelluksessa ensimmäisenä. Intent-filter -elementin avulla järjestelmä yhdistää aktiviteetin ja komponentille välitetyn Intent -objektin, jolla kuvaillaan aktiviteetille tehtävät toimenpiteet. /18/

Manifest -tiedostoon määritellään myös laite ja ominaisuudet, joita applikaatio tarvitsee. Näillä asetuksilla on vaikutusta esimerkiksi sovelluksen näkyvyyteen Google Play -kaupassa. <uses-feature> elementtiin määritellään ominaisuuksia, joita applikaatio tarvitsee toimiakseen. /18/

Manifestitiedostoon määritellään myös sovellukselle käytettävä tyyli attribuutilla android:theme. Attribuutti määrittelee linkityksen styles.xml -resurssiin, jossa voidaan vaikuttaa sovelluksen ulkonäköön, esimerkiksi aktiviteeteilla käytettäviin

väreihin ja fonttien kokoihin. Tällä tavalla saadaan yhtenäinen teema näkymille siten, että kaikkia näkymiä ei tarvitse erikseen muokata samanlaisiksi. /18/

5.6 Resurssit

Android -projekteissa kaikkia staattisia tietoja ja sisältöjä, kuten kuvat, tekstit ja layoutit, jota sovelluksessa käytetään, kutsutaan resurssiksi. Vaikka nämä on mahdollista määritellä elementeille suoraan, niin virallinen suositus on, että sisältö määritellään erillisiin tiedostoihin ja linkitetään näkymille käyttöön. Näin voidaan toimittaa eri resursseja, kuten näkymiä, erikokoisille laitteille. Resursseihin saa pääsyn käyttäen resurssi ID:tä, jotka on generoitu projektin R -luokassa. Resurssit tallennetaan projektin res -kansion alle resurssille kuuluvaan kansioon. /19/

Taulukko 2 Yleisiä resurssikansioita. /19/

drawable/	Bitmap -tiedostot eli kuvat tulee drawable/ kansion alle.
layout/	XML -tiedostot, jotka määrittelevät aktiviteetille layoutit.
raw/	Omavaltaisia tiedostoja. Työssä raw/ kansio pitää sisällään kirjautumispalvelimen tiedot.
values/	XML -tiedostoja arvoille, kuten teksteille.

5.7 Layout

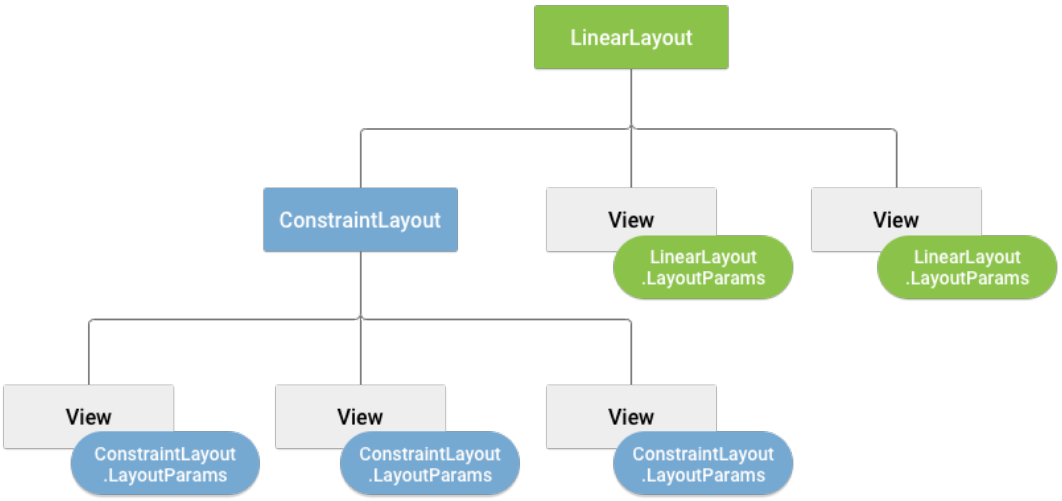
Layout on XML -tiedosto, joka määrittelee aktiviteetin käyttöliitymän rakenteen. Rakenne tehdään layoutille käyttäen View- ja ViewGroup -objekteja. View piirtää ruudulle tietoa ja ViewGroup määrittelee, missä järjestyksessä objektit piirretään. Layout voidaan määrittää sovellukselle kahdella tavalla, joko määrittelemällä käyttöliittymä XML -tiedostossa tai View- ja ViewGroup -objektit voidaan luoda ohjelman pyöriessä. /20/

Layoutin määrittely XML -tiedostolla tehdään seuraavasti. Esimerkissä ConstraintLayout ViewGroupin sisälle on määritelty LinearLayout, jonka sisälle on sijoitettu painike. /20/

```
<androidx.constraintlayout.widget.ConstraintLayout
    …
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <LinearLayout
        android:id="@+id/linearLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        …
        <Button
            android:id="@+id/events"
            android:layout_width="0dp"
            android:layout_height="150dp"
            android:onClick="openEvents"
            android:text="@string/eventsButton" />
    </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Erilaisia näkymiä saadaan siis ryhmittämällä objekteja, kuten LinearLayout, joka määrittää kuinka ryhmän sisällä olevat View -objektit piirretään ruudulle. Ryhmän sisälle määritellään View -objektit, kuten painikkeet, tekstit ja kuvat. /20/



Kuva 9. Esimerkkikaavio käyttöliittymän hierarkkista /20/

Layout -tiedosto tallennetaan projektin resurssikansioon ja voidaan kutsua aktiviteetille metodilla setContentView() /20/

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_authentication);
}
```

Virallinen suositus on, että layout ladataan aktiviteetin onCreate -tilassa. /20/

5.8 Teksti

Applikaatiossa on palvelimelta saadun datan lisäksi myös staattista tekstiä. Tätä ei suositella kirjattavaksi suoraan objekteihin, vaan erilliseen xml -muotoiseen resurssitiedostoon, jota voidaan kutsua objektille seuraavasti: android:text="@string/eventsButton". Toimintatapa mahdollistaa myös lokalisoinnin eri kielille. /20/

5.9 Syöte

Syötteet käyttöliittymässä käsitellään takaisinkutsumetodeilla, joita Android -kehys kutsuu. Tapahtumia seurataan kuuntelijoilla, jotta takaisinkutsumetodit voidaan siepata funktioiden ajamista varten. Kuuntelijat voidaan määrittää näkymäluokkaan layoutissa. Kuuntelijoita voidaan lisätä esimerkiksi painikkeille tai kuville. Erilaisia kuuntelijoita on useita. Useimmiten tarvitaan vain onClick() tai onLongClick(), joka kutsuu metodin, kun objektia painaa. onLongClick() -metodi kutsutaan, kun objektia painetaan yli sekunnin. Jos Android -laitteella on käytössä fyysinen

näppäimistö, voidaan käyttää myös onFocusChange() -kuuntelijaa, kun käyttäjä navigoi objektiin tai siitä pois. /21/

Esimerkki kuuntelijan aktivoinnista osana aktiviteettiä. /21/

```
public class ExampleActivity extends Activity implements OnClick-
Listener {
    protected void onCreate(Bundle savedValues) {
        ...
        Button button = (Button)findViewById(R.id.corky);
        button.setOnClickListener(this);
    }

    // Implement the OnClickListener callback
    public void onClick(View v) {
      // do something when the button is clicked
    }
    ...
}
```

Kuuntelijoita voidaan myös määrittää suoraan objektille Layout.xml -tiedostossa. /21/

```
<Button
    android:id="@+id/events"
    android:layout_width="0dp"
    android:layout_height="150dp"
    android:onClick="openEvents"
    android:text="@string/eventsButton" />
```

Esimerkissä painikkeelle on määritelty onClick -kuuntelija, joka ajaa aktiviteetilla openEvents -funktion. /21/

5.10 Fragmentti

Fragmentti on uudelleenkäytettävä aliaktiviteetti, jolla voidaan hallita tiettyä osaa ruudulla. Fragmenteilla voidaan hyödyntää kirjoitettua koodia uudelleen. Esimerkiksi applikaatiossa on lista esineistä ja nämä voivat aueta yksityiskohta -näkymään. Fragmentin käyttöliittymä määritellään omassa Layout.xml -tiedostossa. /22/

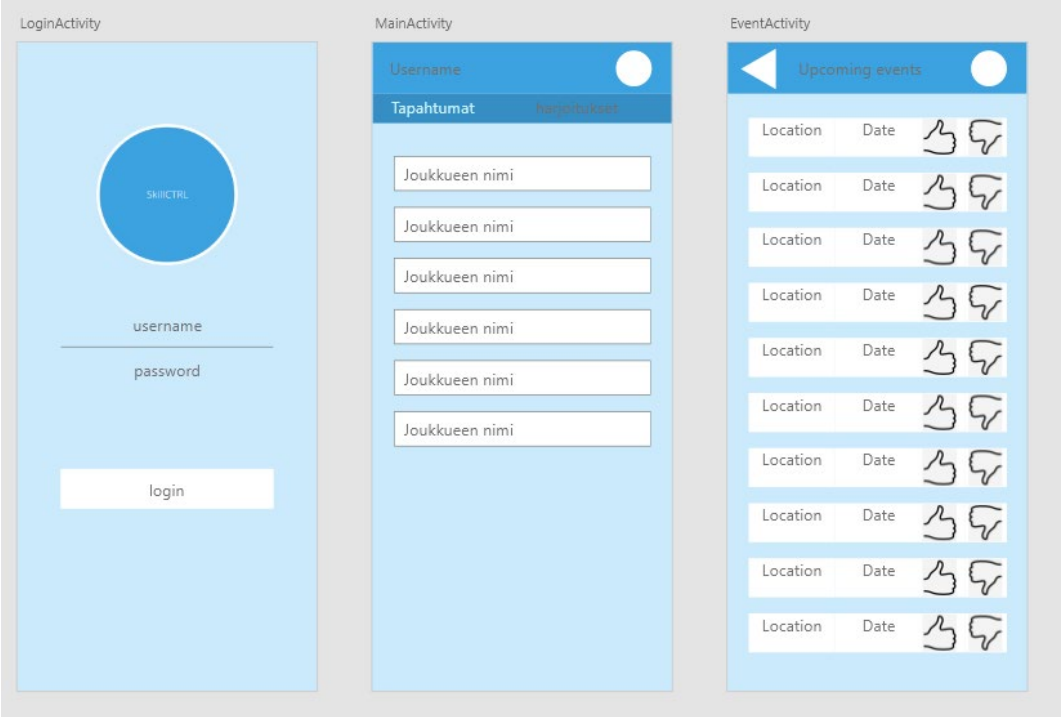
6 TYÖ

Aloitamme sovelluksen rakentamisen määrittelyistä, sitten asennamme kehitysympäristön, kirjoitamme sovelluksen, käännämme koodin, testaamme ja korjaamme ohjelmistovirheitä. Lopuksi julkaistaan ohjelma.

6.1 Use case

Ohjelmiston suunnittelu alkaa määrittelystä. Määrittely tehdään käyttäen käyttötapauskaaviota. Use Case on listaus toimenpiteistä, jotka ovat tietojärjestelmässä mahdollisia. Näiden pohjalta voidaan suunnitella käyttöliittymä ja ohjelmakoodi, jolla käyttöliittymä toteutetaan. Työn tarkoituksena on keskittyä mobiilisovelluksen käyttämiin teknologioihin, jotka mahdollistavat datan hakemisen ja päivittämisen verkkoyhteyden kautta, joten käyttöliittymän suunnittelu tehdään hyvin suppeasti. Haluamme, että käyttäjä pystyy tekemään seuraavat asiat.

- Käyttäjä voi kirjautua ja tunnistautua käyttäjätunnuksella ja salasanalla.
- Käyttäjä näkee joukkueet, joissa hän on jäsenenä.
- Käyttäjä näkee valitun joukkueen tulevat tapahtumat ja harjoitukset.
- Käyttäjä voi ilmoittautua joukkueensa tuleviin harjoituksiin ja tapahtumiin.

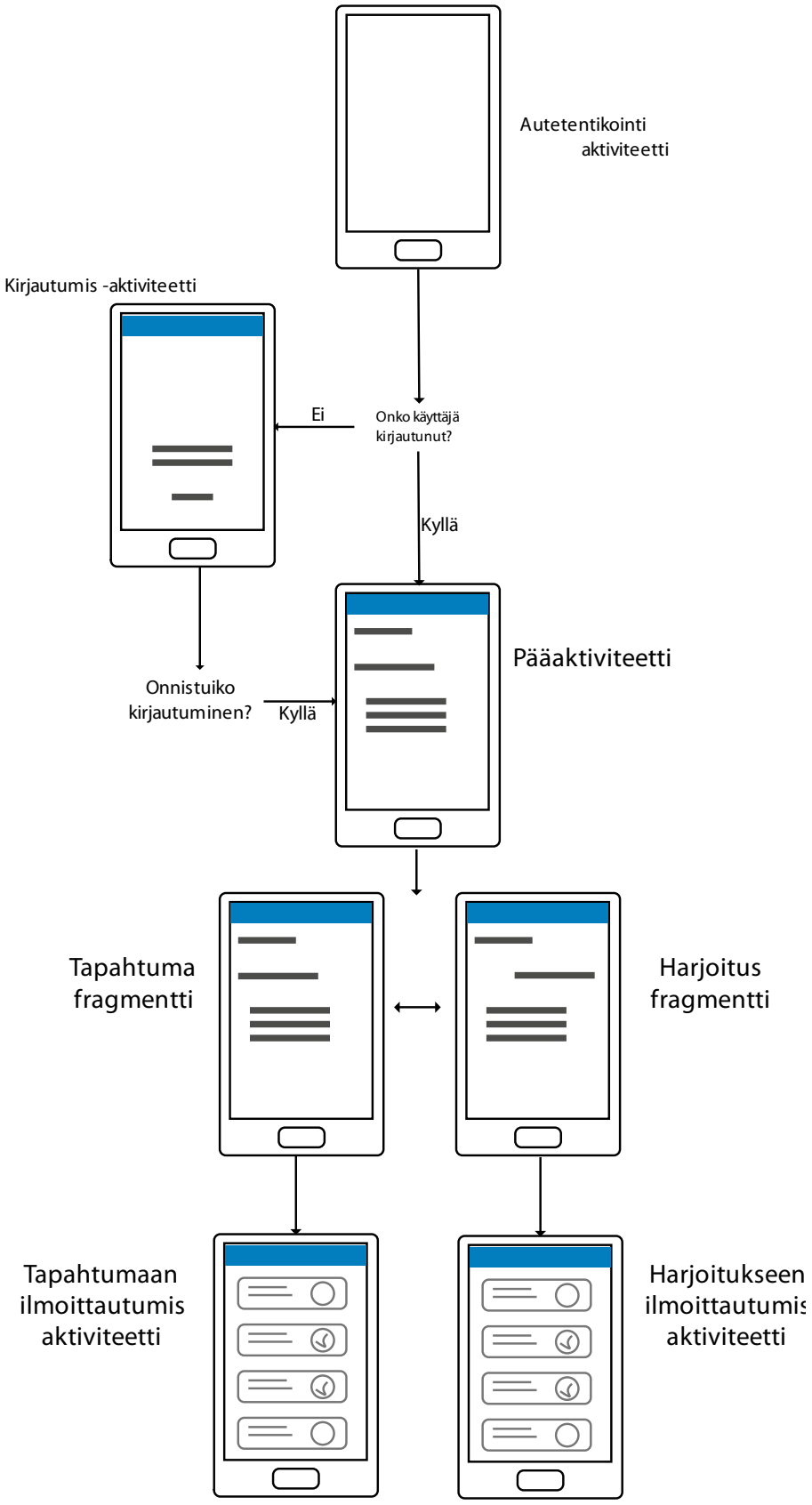


Kuva 10. Prototyyppi käyttöliittymästä annettujen määrittelyjen mukaan

Käyttöliittymän prototyyppi kuvassa 10 on tehty käyttäen Adobe Xd -suunnitteluohjelmaa, joka mahdollistaa nopean käyttöliittymän rakentamisen. Prototyyppiin voidaan lisätä yksinkertainen näkymän siirto painikkeiden tai kosketuksen kanssa.

Työssä käytettiin versionhallintaan Git -palvelua. Android Studiossa on sisäänrakennettu versionhallintatyökalu, joka mahdollistaa helposti koodin viemisen ja hakemisen repositoriosta.

Kuvassa 11 on kuvattu applikaation rakenne aktiviteettitasolla. Sovellus käynnistyy autentikointiaktiviteetistä ja aktiviteetilla on painike, josta voidaan siirtyä seuraavaan yhdistettyyn aktiviteettiin.



Kuva 11. Applikaation työnkulku

6.2 Autentikointiaktiviteetti

AndroidManifest.xml -tiedostossa on määritelty applikaation käynnistyvän aktiviteetillä, joka ohjaa käyttäjää ohjelman kulussa seuraavaan näkymään (kuva 11). Riippuen AWSMobileClient -metodin palauttavasta kirjautumistilasta, siirrytään kirjautumisaktiviteetille tai pääaktiviteetille. Tilan tarkastus tehdään Javan switch statementillä. Tilan tarkastus tehdään aktiviteetin elinkaaren onResume - vaiheessa.

6.3 Kirjautumisaktiviteetti

Kirjautuminen koostuu layoutista, jossa on vertikaalisesti asetettu kaksi <EditText> elementtiä. <EditText> toimii tekstinsyöttökenttänä. Ensimmäinen on käyttäjänimeä varten ja toinen on salasanaa varten. EditText mahdollistaa syötetyypin vaihtamisen salasana -tyyppiseksi, jolloin syöte kyseisessä kentässä ei näy kirjaimina vaan * -merkkeinä. Layoutin pohjalla on painike, joka onClick kuuntelijalla odottaa, että käyttäjä painaa painiketta. Painalluksen jälkeen AWSMobileClient tarkistaa Amazonin UserPoolista syötetyt kirjautumistiedot ja asettaa käyttäjän kirjautumistilan kirjautuneeksi, jos tunnukset ovat oikein. Muussa tapauksessa käyttäjälle annetaan virheilmoitus.

6.4 Päänäkymä -aktiviteetti

Aktiviteettiin on määritelty ViewPager adapteri, joka mahdollistaa fragmenttien käytön. Näkymässä on määritelty omat fragmentit tapahtumille ja harjoituksille, joista klikkaamalla tuodaan käyttäjälle ilmoittautumisaktiviteetti näkyviin. Fragmenteille voidaan siirtyä pyyhkäisemällä ruudulla oikealle tai vasemmalle `ScreenSlidePagerAdapter` -luokkaa hyödyntämällä.

Aktiviteetissa määriteltiin kuuntelija, joka tarkistaa kirjautumisen tilaan liittyviä muutoksia ja aukaisee käyttäjälle tarpeen vaatiessa kirjautumisikkunan.

6.5 Fragmentit

Käyttäen Amazon UserPoolista saamaa autentikointi- ja identifiointitokenia, saamme sovelluksen palvelimelta tiedot joukkueista, jotka kuuluvat käyttäjälle. Joukkuetiedot haetaan restful -palvelimelta käyttäen Retrofit 2 -kirjastoa. Koska on tiedossa, missä järjestyksessä joukkueet listataan, niin fragmentille lisätään kuuntelija, joka joukkueen nimeä painaessaan osaa aukaista ja siirtää joukkueID:n ja pelaajaID -tiedon ilmoittautumisaktiviteetille.

6.6 Ilmoittautuminen

Ilmoittautuminen on tehty omalle aktiviteetille, jossa listataan joukkueen tapahtumat. Aktiviteetin onCreate -tilassa tehdään API -kysely käyttäen joukkue- ja pelaaja -ID:llä. Vastaanotettu JSON -data kirjataan tapahtumaluokalle, josta saadaan haettua tapahtumasta tapahtuman nimi, aika, paikka ja osallistumistieto. Tapahtumat listataan näytölle RecyclerView -luokkaan. RecyclerView on listanäkymä, johon saadaan dynaamisesti lisättyä osia adapterin kautta. Osallistumistieto esitetään listassa pienellä kuvakkeella. Adapterissa on määritelty kuuntelija, joka lähettää HTTP POST -pyynnön palvelimelle, jossa vaihdetaan ilmoittautumisen tilaa. Tilan vaihdos päivittää myös kuvakkeen ilmoittautumislistalla.

7 JOHTOPÄÄTÖKSET JA POHDINTA

Työn tarkoituksena oli luoda applikaatio Android -käyttöjärjestelmälle käyttäen tunnistautumista ja verkkoyhteyttä. Sovellus saatiin lähettämään tunnistautumistiedot tunnistuspalvelimelle käyttäen hyväksi OAuth2 -protokollaa ja vastaanottamaan käyttäjän identifioivan tunnistuskoodin, jolla voidaan tehdä kyselyitä ja vastaanottaa käyttäjän tietoja sovelluspalvelimelta käyttäen hyväksi kolmannen osapuolen kirjastoja ja Android -kehystä. Käyttöliittymään saatiin käytettyä komponentteja siten, että käyttökokemus on dynaaminen.

Android Studio on työkaluna Android -sovelluksilla hyvin kätevä ja valmiit koodimallit, ja layout editori nopeuttavat kehitystyötä huomattavasti, jolloin aikaa voidaan käyttää tehokkaammin ohjelman logiikan käyttöön.

Suurin asia, jonka olen työtä tehdessäni oppinut, on virallisen dokumentaation lukeminen ja kuinka tuo teoria saadaan toimimaan tuotannossa.

Ohjelmaan voidaan helposti lisätä uusia toimintoja ja ominaisuuksia käyttäen aktiviteettejä ja fragmentteja. Tutkittavaksi jää vielä myös kaikki Googlen tarjoamat best practice -ohjeet, jotka voidaan täyttää, että saadaan ohjelma Google Play Storeen verkkosovelluksen käyttäjille.

LÄHTEET

/1/ Elenkov, N. Android Security Internals, V. 2015 No Starch Press

/2/ Griffiths, D., & Griffiths, D. (2017). Head First Android Development. Culem-borg, Netherlands: Van Duuren Media.

/3/ Android Open Source Project. (27.12.2019). Platform Architecture Viitattu 15.02.2020, https://developer.android.com/guide/platform

/4/ Hardt, D., & Internet Engineering Task Force (IETF). (2012). The OAuth 2.0 Authorization Framework. Viitattu 15.02.2020, https://tools.ietf.org/html/rfc6749

/5/ Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., & Mortimore, C. (2014, 11.8). Final: OpenID Connect Core 1.0 incorporating errata set 1. Viitattu 15.02.2020, https://openid.net/specs/openid-connect-core-1_0.html

/6/ Jones, M., Bradley, J., & Sakimura, N. (2015). JSON Web Token(JWT). Viitattu 15.02.2020, https://tools.ietf.org/html/rfc7519

/7/ Auth0.Com. JWT.IO. Viitattu 15.02.2020, https://jwt.io/

/8/ Auth0.Com. JWT.IO. Viitattu 15.02.2020, from https://jwt.io/introduction/

/9/ Amazon Web Services. (n.d.). Amazon Cognito - Simple and Secure User Sign Up & Sign In | Amazon Web Services (AWS). Viitattu 20.02.2020, https://aws.amazon.com/cognito/

/10/ Amazon Web Services. (2020). Amazon Cognito User Pools, https://docs.aws.amazon.com/cognito/latest/developerguide/ cognito-user-identity-pools.html

/11/ Amazon Web Services. AWS Amplify Android SDK Documentation. Viitattu 20.02.2020, 2020, https://docs.aws.amazon.com/sdk-for-android/

/12/ Amazon Web Services. Getting Started. Viitattu 20.02.2020, https://aws-amplify.github.io/docs/sdk/android/start

/13/ Retrofit. Viitattu 17.02.2020, https://square.github.io/retrofit/

/14/ Pöhls, M., & Peitek, N. (2019). Retrofit: Love Working with APIs on Android. Victoria, Canada: Leanpub.

/15/ Android Open Source Project (27.12.2019). Download Android Studio and SDK tools. Viitattu 22.02.2020, 2020, https://developer.android.com/studio

/16/ Android Open Source Project. (27.12.2019). Application Fundamentals. Viitattu 22.02.2020, https://developer.android.com/guide/components/ fundamentals#Components

/17/ Android Open Source Project. (27.12.2019). Application Fundamentals. Viitattu 22.02.2020, https://developer.android.com/guide/components/activities/activity-lifecycle

/18/ Android Open Source Project. (27.12.2019). Application Fundamentals. Viitattu 22.02.2020, https://developer.android.com/guide/topics/manifest/manifest-intro

/19/ Android Open Source Project. (27.12.2019). Application Fundamentals. Viitattu 22.02.2020, https://developer.android.com/guide/topics/resources/providing-resources

/20/ Android Open Source Project. (27.12.2019). Application Fundamentals. Viitattu 22.02.2020, https://developer.android.com/guide/topics/ui/declaring-layout

/21/ Android Open Source Project. (27.12.2019). Application Fundamentals. Viitattu 22.02.2020, https://developer.android.com/guide/topics/ui/ui-events

/22/ Android Open Source Project. (27.12.2019). Application Fundamentals. Viitattu 22.02.2020, https://developer.android.com/guide/components/fragments

/23/ Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Viitattu 18.02.2020, https://www.ics.uci.edu/%7Efielding/pubs/dissertation/rest_arch_style.htm