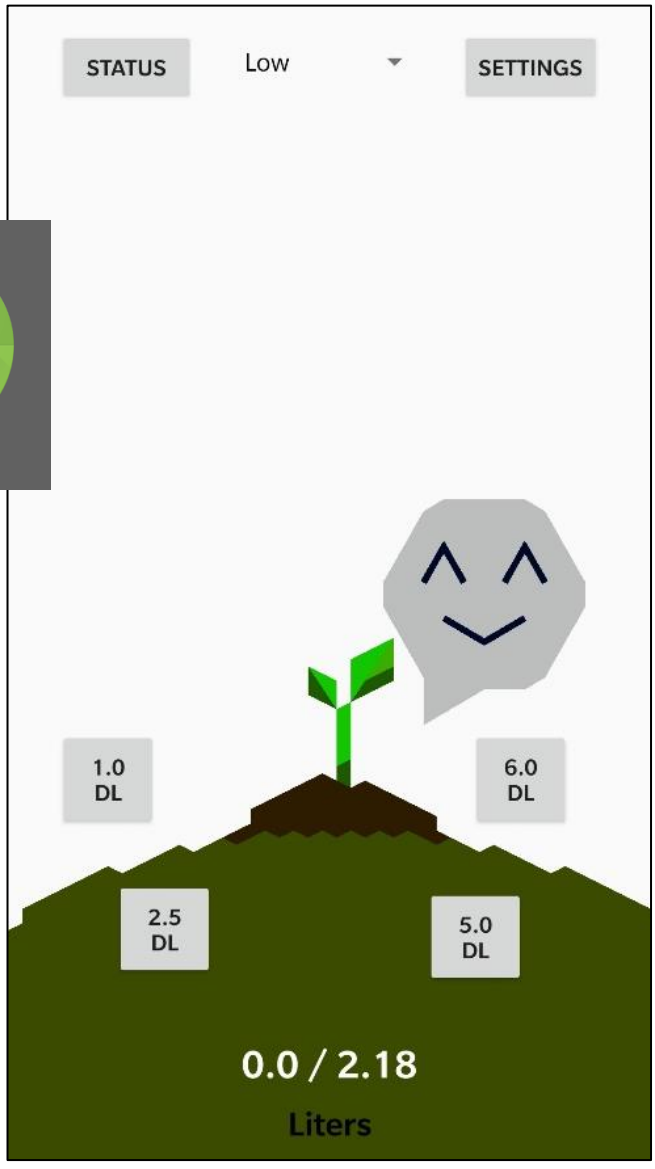
Mika Lintula

Sovelluksen kehittäminen Android Studiolla



Tradenomi

Tietojenkäsittely

Kevät 2020

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä(t): Lintula Mika

Työn nimi: Sovelluksen kehittäminen Android Studiolla

Tutkintonimike: Tradenomi, tietojenkäsittely

Asiasanat: Android Studio, ohjelmointi, Android, Sovelluskehitys, mobiililaite

Opinnäytetyön tarkoituksena oli kehittää Android Studiolla mobiililaitteelle sovellus, jonka tarkoituksena on motivoida käyttäjää juomaan päivittäin tarpeellisen vesimäärän. Sovelluksen kehityksessä käytettiin Java-ohjelmointikieltä.

Android on Googlen hallinnoima ja jakama avoin käyttöjärjestelmä, joka pyörii usealla erilaisella mobiililaitteella. Puhelimien ja tablettien lisäksi Android-käyttöjärjestelmää käytetään Android Auto ja Android Wear laitteissa. Android-käyttöjärjestelmälle voidaan kehittää sovelluksia käyttämällä Android Studio-ohjelmointiympäristöä. Android Studiossa on visuaalinen käyttöliittymä ja sisältää valmiit työkalut APK-tiedoston paketointiin.

Sovellus laskee käyttäjälle juotavan vesimäärän hänen antamiensa tietojen mukaan. Käyttäjää motivoidaan esittämällä visuaalinen efekti veden juonnin vaikutuksesta puulla, joka kasvaa tai kuihtuu. Sovellukseen kehitettiin ominaisuuksia, joilla lisätään sovelluksen monipuolisuutta ja käytettävyyttä sekä miten ne keskustelevat keskenään.

Tulokseksi saatiin toimiva sovellus, jonka ominaisuudet toimivat toivotulla tavalla. Sovelluksen grafiikkaan ei panostettu muuten kuin tekemällä sovellukseen kolme puuvaihtoehtoa ja niiden neljä eri kasvuvaihetta. Jatkokehitystä on suunniteltu sovellukselle, jolloin tullaan keskittymään grafiikan ja käyttöliittymän kehitykseen.

KAMK • University of Applied Sciences

Abstract

Author(s): Lintula Mika

Title of the Publication: Application development with Android Studio

Degree Title: Bachelor of Information Technology

Keywords: Android Studio, programming, Android, application development, Java, mobile device

The purpose of this Bachelor's thesis was to use the Android Studio to develop an application for a mobile device that motivates the user to drink enough water daily. Java programming language was used to develop this application.

Android is Google's controlled and distributed open operating system that runs on multiple devices. Android operating system is used in phones and tablets, but it also runs in the Android Auto and Android Wear devices. It is possible to develop applications for the operating system using Android Studio integrated development environment. Android Studio has a visual interface and the tools that build the APK-file.

The application calculates the amount the user should drink daily from the information the user applies. The user is motivated with the visual effect of a tree that grows or withers based on how well the user is drinking water. Throughout the thesis, the features are presented, as well as how they work together to add versatility and usability to the application.

The result is a working application where the features work as intended. The application has three tree types with four growing states but otherwise, visuality wasn't the focus of the development. It has been planned to continue the development of this application, where visual design and user interface development are in focus.

Sisällysluettelo

1 Johdanto

Opinnäytetyön tarkoituksena on kehittää Android Studiolla mobiililaitteelle sovellus, jonka tarkoituksena on motivoida käyttäjää juomaan päivittäin tarpeellisen vesimäärän. Sovelluksen kehityksessä käytetään Java-ohjelmointikieltä. Sovellus laskee vesimäärän, jonka käyttäjän olisi juotava päivittäin hänen antamiensa tietojen pohjalta. Käyttäjää motivoidaan esittämällä visuaalinen efekti veden juonnin vaikutuksesta puulla, joka kasvaa tai kuihtuu. Työn aikana keskitytään enemmän ominaisuuksiin, joilla lisätään sovelluksen monipuolisuutta sekä miten ne keskustelevat keskenään.

Mobiiliympäristöt ja niille kehittäminen on ollut aina mielenkiitoinen aihe, koska mobiililaitteilla on paljon kehitystä rajoittavia tekijöitä, joita pitää huomioida niille kehittäessä. Pelien kehittäminen mobiililaitteille on tuttua ammattikorkeakouluopintojen kautta. Tavoitteena olisi kuitenkin monipuolinen ohjelmointiosaaminen ja sovellukset ovat jotain, minkä kehityksestä ei ole paljon kokemusta. Oli siis luontevaa valita opinnäytetyön aiheeksi sovelluksen kehittäminen Android Studiolla, Java-ohjelmointikieltä käyttäen.

2 Android

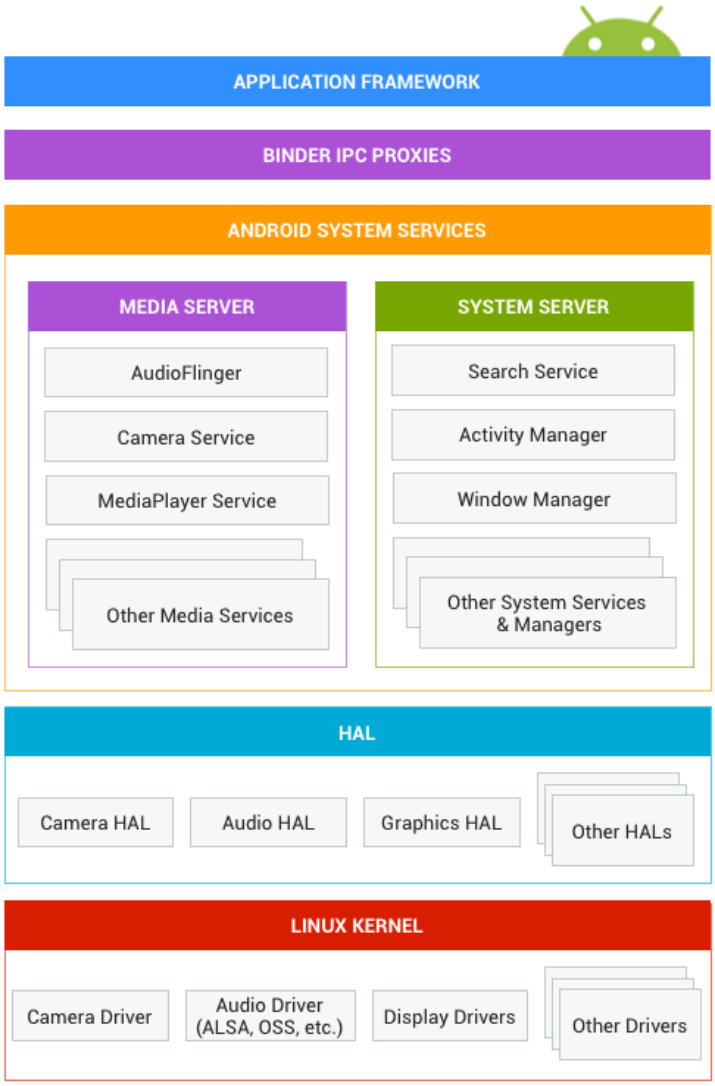
2.1 Yleistä

Android on alun perin Android Inc:in kehittämä mobiilikäyttöjärjestelmä, jonka Google osti vuonna 2005 [1]. Android julkaistiin Googlen toimesta vuonna 2007, josta vuoden päästä vuonna 2008 julkaistiin ensimmäinen kaupallinen Android-laite [2]. Google jatkoi Open Handset Alliance -konsortion käyttöjärjestelmän kehittämistä Googlen johtamana [3]. Uusin Android Versio on Android 10, joka julkaistiin elokuussa 2019 Googlen Pixel-laitteille [4].

Androidin käyttäminen on maksutonta kuin myöskin sovellusten kehittäminen Androidille. Google mahdollistaa tämän pitämällä Androidin lähdekoodin avoimena ja tarjoamalla ilmaiset kehittämistyökalut. Pitämällä lähdekoodin avoimena Android välttää suurta epäonnistumista, mikä olisi mahdollista, jos yksi yhtiö rajoittaisi ja kontrolloisi Androidin innovaatioita [5]. Tällä mahdollistetaan kilpailua ja innovaatiota, joka puolestaan on monipuolistanut paljon, millaisilla laitteilla Android pyörii. Laitevalmistajan ottaessa Android käyttöjärjestelmän käyttöön laitteelleen hän voi räätälöidä käyttöjärjestelmän ulkoasun juuri kyseiselle laitteelle sopivaksi ja sille uniikiksi. Android käyttöjärjestelmää käyttäviä laitteita ei ole vain mobiililaitteet vaan nykyisin on myös Android autoja tai Android Wear eli älyrannekello. Hyvin usea mobiililaite sisältää tehdasasetuksiin monia Googlen toimintoja kuten esimerkiksi Gmail ja Gmaps.

2.2 Arkkitehtuuri

Android on käyttöjärjestelmältään kohtuullisen monimutkainen järjestelmä, mutta sovelluksia tehdäkseen sitä ei tarvitse ymmärtää kokonaan. On hyvä kuitenkin käydä läpi, mitä osa-alueita käyttöjärjestelmässä on, mitä ne tekevät siten hahmottaa, millä tasolla sovellus pyörii ja miten se keskustelee käyttöjärjestelmän kanssa. Kuvassa 1 nähdään Android käyttöjärjestelmän arkkitehtuuri ja miten se rakentuu eri osa-alueista.



Kuva 1. Android käyttöjärjestelmäarkkitehtuuri [6].

Android arkkitehtuurin pohjalla toimii Googlen kehittämä Linux-käyttöjärjestelmäydin, joka on muokattu mobiililaitteelle sopivaksi. Tämän takia ajureiden kehittäminen Android-laitteille on hyvin samanlaista kuin ajureiden kehittäminen tietokoneella pyörivälle Linuxille. Androidissa käytettävässä Linux-versiossa on muutamia erikoisempia ominaisuuksia, jotka ovat tärkeitä mobiilikäyttöjärjestelmälle. Laitteessa voidaan käyttää mitä tahansa käyttöjärjestelmäydinversiota, kunhan se tukee niitä ominaisuuksia, mutta suurimmalta osin ne eivät vaikuta ajureiden kehittämiseen. [5.]

Hardware Abstraction Layer (HAL) tarjoaa käyttöjärjestelmälle kirjastot, joita laitteen komponentit käyttävät toimiakseen oikein. Kun joitain komponenttia halutaan käyttää, Android-järjestelmä

aukaisee kyseiselle komponentille kuuluvan kirjaston. Näitä kirjastoja käyttämällä on mahdollista toteuttaa toimivuuksia ilman, että ne häiritsevät korkeamman tason järjestelmää. Nämä komponenttikirjastot on lajiteltu jo modulaarisesti mutta Android 8.0-version ilmestyessä markkinoille kirjastot toteutettiin uudelleen. Uusi kirjasto kirjoitettiin HIDL-kielellä ja kirjastot lajiteltiin kahteen lohkoon Binderized HAL ja Passingthrough Hal. [7.]

Android System Services eli Androidin järjestelmäpalvelut ovat modulaarisia keskitettyjä komponentteja, jotka on jaettu kahteen lohkoon: järjestelmä ja media. Järjestelmä sisältää esimerkiksi ilmoitusmanagerin ja ikkunamanagerin ja media huolehtii sitten palveluista, jotka hoitavat median toiston tai sen tallentamisen. [6.]

Binder IPC oli aikaisemmin laitteen prosessien kommunikaation käyttöön varattuna, mutta Android 8.0-versiossa tämä huolehtii myös sovelluskehyksen ja HAL:n välisestä kommunikaatiosta [8]. Tämä mekanismi mahdollistaa sovelluskehyksen ylittää prosessirajan ja kutsua Android-järjestelmän palveluita. Sovelluskehyksen tasolla tämä kommunikaatio on piilotettu kehittäjältä [6].

Application framework on sovelluskehys. Tämä taso on meille tärkein, koska juuri tälle tasolle kehitetään sovellukset, joita laitteen käyttäjä näkee ja käyttää. Sovelluskehys on kokoelma työkaluja, jotka helpottavat sovellusten kehittämistä ja kommunikointia Android-laitteen kanssa [6]. Näitä on esimerkiksi Aktiviteettihallinta, jolla hallitaan aktiviteettien elämänkaarta ja miten aktiviteetit pinotaan taustalla.

- Activity Manager eli aktiviteettien hallinta nimensä mukaan hallitsee aktiviteettien elämänkaarta ja miten ne pinotaan taustalla, jotta sovelluksiin, jotka pyörivät eri prosesseissa, saadaan sulavat animaatiot [9].

- Package Manager on pakettienhallinta, joka seuraa mitä, sovelluksia on asennettu laitteellesi ja tarjoaa niistä tietoa [9].

- Window Manager, ikkunoidenhallinta, vastaa, mikä ikkuna on aktiivisena näytöllä ja ikkunoiden järjestyksestä [9].

- View Manager, näkymähallinta vastaa sovelluksen ja laitteen käyttöliittymän näkymistä ja näkymäryhmistä [9]. Näkymä on objekti, joka mahdollistaa käyttäjän vuorovaikutuksen piirtämällä jotain näytölle. Kaikki sovellukset sisältävät näkymiä ja niiden ryhmiä.

- Telephone Manager eli puhelinpalveluidenhallinta, hallitsee ohjelmointirajapintaa, jota esimerkiksi puhelinsovellukset käyttävät [9]. Se tarjoaa puhelinpalveluiden tiloja, jotka ovat kyseisellä hetkellä käytettävissä.

- Content Providers on sisällöntarjoaja. Sen tehtävä on hallinnoida dataan ja mahdollistaa sen jakamisen useiden sovellusten välillä. Siten sovellukset voivat hyödyntää toisten sovellusten tietoa tai jakaa omia tietoja. [10.]

Järjestelmäsovellukset on Android-arkkitehtuurin ylin kerros, jota ei ole kuvattuna kuvassa. Tätä ei aina lasketa käyttöjärjestelmän arkkitehtuuriin. Tässä kerroksessa ovat Androidin ydinsovellukset kuten esimerkiksi internetselain, sähköposti ja kamera. Muut sovellukset voivat hyödyntää tämän kerroksen sovelluksia omien toimintojen suorittamiseen [10]. Sovelluskehittäjätkin hyödyntävät järjestelmäsovelluksia omissa sovelluksissaan [10].

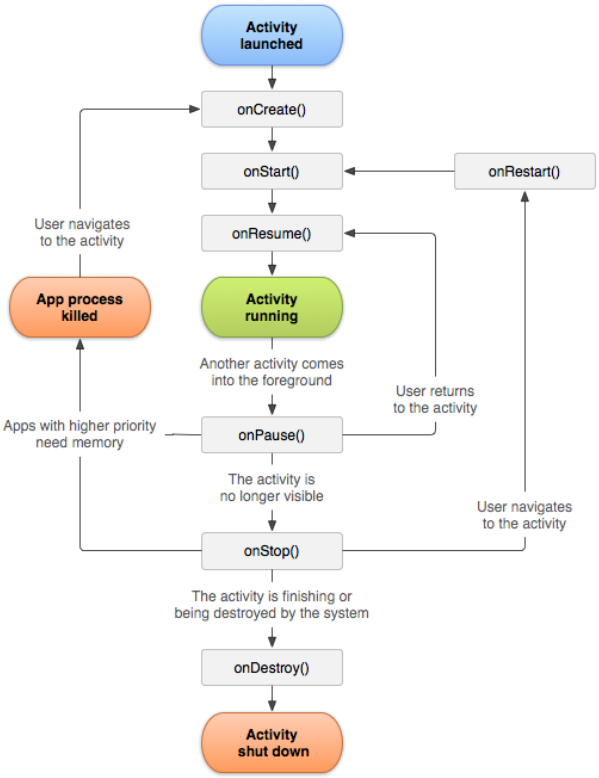
Tämä Android arkkitehtuuri muuttui taas Android 10:n tullessa kehitykseen. Android 10 muuttaa osan järjestelmästä moduuleiksi, jotta niitä voidaan päivittää Androidin omia järjestelmäpäivitysten ulkopuolella [3]. Osa moduuleista tulee käyttämään APK-formaattia ja osa APEX-säiliöformaattia, joka esitellään tämän Android-version kanssa. Tämä tulee mahdollistamaan yksittäisten komponenttien päivittämisen kriittisillä bugikorjauksilla ja muilla parannuksilla tarvittaessa ilman että päivitys vaikuttaa laitteen muihin komponentteihin tai sovelluksiin.

2.3 Sovelluksen aktiviteetit

Aktiviteetti on käyttöliittymällä varustettu näkymä sovelluksessa, jonka kanssa käyttäjä voi olla vuorovaikutuksessa. Sovellukset koostuvat useista aktiiveista, ja näiden avulla hallitaan, mitä osaa sovelluksesta näytetään käyttäjälle.

2.3.1 Aktiviteetin elinkaari

Aktiviteeteillä on elinkaaret, jotka ovat tärkeitä ottaa huomioon sovellusta kehittäessä. Elinkaaren eri vaiheilla kehittäjä voi hallita, mitä tapahtuu sovelluksen avaamisen ja sulkemisen yhteydessä. Näihin elinkaariin vaikuttavat toiset aktiviteetit, aktiviteettien tehtävät ja pinotietorakenne. Kuvassa 2 nähdään aktiviteetin elinkaari sekä miten erilaiset vuorovaikutukset vaikuttavat elinkaaren kulkuun.



Kuva 2. Aktiviteetin elinkaari [11].

Kun sovellus käynnistetään, aloitetaan myös aktiviteetin käynnistykseen kuuluvat metodit. OnCreate(), OnStart() jaOnResume()-metodit suoritetaan aina tässä järjestyksessä, sillä kun ensimmäinen metodi on suorittanut itsensä loppuu se kutsuu seuraavaa.

OnCreate()-metodi kutsutaan, kun aktiviteetti luodaan. Tämä metodi on ainoa pakollinen elinkaarimetodi, joka lisätään aktiviteettiin, sillä se on vastuussa näkymän luonnista. On suositeltavaa, että tässä metodissa tehdään kaikki toimet, jotka tehdään vain kerran aktiviteetin elinkaaren aikana kuten esimerkiksi tietojen hakeminen ja lisääminen listaan. [12.]

OnStart()-metodi huolehtii, että näkymä, joka luotiin on näkyvissä käyttäjälle [11]. Mutta jos kehittäjä haluaa käyttää käyttöliittymää muokkaavia toimintoja, niiden alustaminen suositellaan tehtäväksi tässä [12].

OnResume()-metodia kutsutaan OnStart()-metodin jälkeen ja kun tämän metodin suorituksen jälkeen sovellus on aktiivisessa tilassa ja valmiina vastaamaan käyttäjän vuorovaikutukseen [12].

Seuraavat kolme metodia tapahtuvat heti, kun kyseinen aktiviteetti ei ole käyttäjälle ylimpänä näkyvänä aktiviteettina.

OnPause()-metodi kutsutaan heti, kun käyttäjä poistuu aktiviteetista. Tämä voi johtua siitä, että käyttäjä vaihtaa toiseen aktiviteettiin, toiseen sovellukseen tai toinen sovellus ottaa pääprosessin käyttöönsä. Tätä metodia käytetään yleensä muistin vapauttamiseen. Jos aktiviteetti on taukotilassa, Android-järjestelmä pitää aktiviteetin kokonaisena muistissa. Kun aktiviteettia jatketaan, se ei menetä dataa tai näkymiä, eikä tarvitse alustaa uudelleen. Tämä metodi on hyvin lyhyt, joten tässä ei kannata suorittaa mitään isoja ja raskaita toimintoja. [12.]

OnStop()-metodi kutsutaan, kun aktiviteetti ei ole enää käyttäjälle näkyvissä. Tämä metodi on pitempi kuin OnPause()-metodi, jonka takia OnStop()-metodia suositellaan käyttämään raskaampien ja pitempien toimintojen kuten esimerkiksi tietokannan päivittämiseen. Vaikka suurin osa alustetuista objekteista ja toiminnoista tuhoutuu tämän metodin aikana, näkymät pysyvät ehjinä. [12.]

OnDestroy()-metodi on viimeinen metodikutsu, jonka aktiviteetti saa ennen kuin aktiviteetti tuhoutuu [11]. Resurssit, joita ei vapautettu OnStop()-metodin aikana, vapautetaan viimeistään tässä metodissa [12].

2.3.2 Elinkaaren tilat

Elinkaaren metodeihin liittyy metodien nimenmukaisia tiloja ja ne ovat vuorovaikutuksessa muihin elinkaarimetodeihin. Jälkimmäisten metodien vuorovaikutus on vähän erilainen kuin kolmen elinkaarimetodin, jotka tapahtuvat käynnistyksen yhteydessä, sillä käynnistyksen metodit tapahtuvat järjestyksessä.

Aktiivinen tila (Active State)

On silloin, kun aktiviteetin näkymä on mobiililaitteella päällimmäisenä odottaen käyttäjän vuorovaikutusta. Tämä tila on päällä heti, kun sovellus on käynyt käynnistymisen elinkaarimetodit läpi. [11.]

Keskeytetty tila (Pause State)

Aktiviteetti on siirrettynä taustalle, joissakin tapauksissa aktiviteetti on näkyvissä edelleen käyttäjälle. Tämä johtuu yleensä siitä, että jokin aktiviteetti peittää vain osan mobiililaitteen ruudusta tai päällimmäisen aktiviteetin ikkuna on läpinäkyvä. Jos aktiviteetti nostetaan takaisin käyttöön kun se on keskeytetyssä tilassa, se kutsuu ainoastaan OnResume()-metodia. Jos jokin toinen sovellus, joka on korkeammalla tärkeysjärjestyksessä, tarvitsee enemmän muistia, järjestelmä voi vapauttaa muistia taukotilassa olevilta aktiviteeteilta. Tämän jälkeen kun aktiviteettia jatketaan, on kutsuttava OnCreate()-metodia initialisoitavakseen kaikki uudestaan. Sovellukset kutsuvat OnPause() ja OnResume()-metodeja useasti, ja suositellaan, että niitä ei käytetä initialisointeihin. [12.]

Pysäytettytila (Stop State)

Pysäytetty tila käynnistyy, kun aktiviteetti ei ole enää näkyvissä käyttäjälle. Tämän tilan ollessa päällä aktiviteetti jatkaa OnStart()-metodista, kun aktiviteetti nostetaan takaisin käyttöön. Mutta samalla tavalla kuin keskeytetty tilassa, jos sovelluksen resursseja tarvitaan, sovellus vapauttaa niitä ja käynnistää itsensä uudelleen OnCreate()-metodilla. [12.]

Tuhottu tila (Destroy state)

Aktiviteetin elinkaaren viimeinen tila, joka on umpikuja. Kun aktiviteetin tila muuttuu tuhotuksi, se vapauttaa kaikki resurssit ja näkymäobjektit muiden sovellusten käyttöön [12]. Tämän tilan jälkeen sovelluksen on pakko käynnistää itsensä uudelleen.

2.4 Manifestfile

Manifestfile.xml on tiedosto, jonka tulee olla jokaisella sovelluksella projektin juuressa, täsmälleen tällä nimellä [13]. Tämän tiedoston tarkoitus on kertoa kaikki olennainen tieto sovelluksesta Android-käyttöjärjestelmälle, sovelluksen kokoamistyökaluille ja Google Playlle [13]. Kuten esimerkiksi, mitä komponentteja sovellus sisältää, missä tilanteessa niiden tulee käynnistyä ja mitkä prosessit ylläpitävät näitä komponentteja [14]. Käyttöjärjestelmä tarvitsee tämän informaation, jotta voidakseen käynnistää sovelluksen ja toimia oikein.

Manifestfile.xml nimeää Java-paketin sovellukselle. Paketin nimi toimii uniikkina tunnisteena sovellukselle, ja yleensä se on sama kuin koodin nimiavaruus [13]. Android rakennustyökalut käyttävät tämän paketin nimeä löytääkseen koodin olentojen sijainnit projektia rakentaessa [14]. Manifestfile.xml tiedosto kertoo myös minimi API-tason, mitkä ovat sovelluksen luvat käyttää laitteen toimintoja ja mihin kirjastoihin se tarvitsee yhteyden toimiakseen [14].

Tiedot voivat olla jaettuina hieman hämmentävästi Android Studioissa Manifestfile.xml ja gradle.build-tiedostojen kesken. Kunhan muistaa pitää tiedostojen tiedot yhtäläisinä ei pitäisi ilmestyä ongelmia. Jos kuitenkin sovellusta paketoidessa tiedot eroavat toisistaan, työkalut käyttävät gradle.build-tiedoston tietoja. Lopuksi rakennustyökalut korvaavat nimitunnisteen sovelluksen ID:llä, joka löytyy Gradle rakennustiedostoista. Tätä ID:tä käytetään uniikkitunnisteena käyttöjärjestelmälle ja Google Playlle [13].

Manifestfile.xml listaa myös instrumentaatio-objektit, jotka tarjoavat profiloinnin ja muuta informaatiota, kun sovellus on käynnissä. Nämä deklaraatiot ovat manifestfile.xml-tiedostossa vain, kun sovellus on kehityksessä ja testauksessa. Nämä poistetaan, kun sovellus julkaistaan. [14.]

2.5 APK

APK (Android Package Kit) eli Android Paketointipaketti. On tiedostomuoto, jota Android käyttää jakamaan ja asentamaan sovelluksia. Se sisältää kaikki elementit, jotka Android tarvitsee asentaakseen sovelluksen oikein laitteelle. Aivan kuten EXE-tiedostoa käytetään asentamaan sovelluksia Windows-tietokoneille. [15.]

Google Play Store-palvelu lataa ja asentaa APK-tiedoston automaattisesti. On mahdollista ladata APK-tiedosto muilta sovelluskauppapalveluista tai lataamalla sen netistä, joita ei esimerkiksi löydy Play Storesta. Tietenkin kuten tietokoneen kanssa yleensäkin ei ole aina hyvä idea, koska ne voivat sisältää viruksia tai muita haitallisia lisäohjelmia. [15.]

2.6 SDK

SDK (Software development kit) on sovellustyökalusetti, joka sallii kehittäjien kehittää sovelluksia Android käyttöjärjestelmälle. Android SDK sisältää lähdekoodin sisältäviä esimerkkiprojekteja, kehitystyökaluja, emulaattorin, debuggerin ja tarvittavat koodikirjastot. [16.]

Julkaistessaan uuden Android-käyttöjärjestelmän Google julkaisee myös vastaavan SDK-version. Jotta kehittäjien on mahdollista hyödyntää Android-käyttöjärjestelmäversioiden ominaisuuksia, jokaista versiota varten on ladattava ja asennettava SDK-versio. [16.]

3 Android Studio

Android Studio on Android-käyttöjärjestelmän virallinen ohjelmointiympäristö, joka on suunniteltu Android-kehitykseen [17]. Android Studio ilmoitettiin kehitettäväksi toukokuussa 2013, Google I/O konferenssissa [17]. Sen jälkeen sitä on päivitetty säännöllisesti aina nykyiseen versioon 3.5 asti, joka päivitettiin elokuussa 2019.

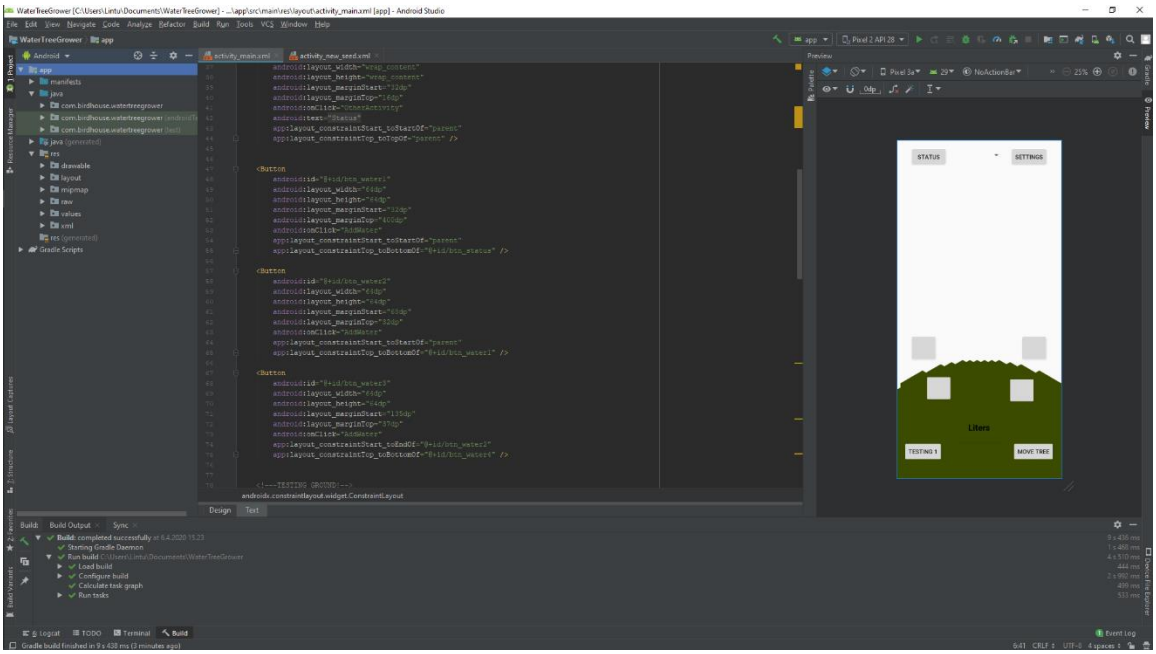
Android studio sisältää työkalut, joita tarvitaan sovelluskehitykseen ja mahdollistaa useamman ohjelmointikielen käyttämisen, kuten Java, Kotlin ja C++. Tosin projektia tehtäessä on muistettava lisätä Kotlin ja C++-tuki, jos tavoitteena on kehittää sovellusta niillä ohjelmointikielillä. Android Studio olettaa, että kehittäjä käyttää Java-ohjelmointikieltä.

Toisin kuin muut Android-käyttöjärjestelmälle soveltuvat kehitysympäristöt, Android Studioon on kerätty useita erilaisia toimintoja yhteen ja siihen on kehitetty editori, jolla mahdollistetaan monipuolinen käyttöliittymän kehittäminen [18, s. 10]. Kehitystä helpottavia ominaisuuksia on myös joustava gradle-pohjainen sovelluksen rakennusympäristö, useita vaihtoehtoja APK-tiedoston generointiin sekä ohjelmointipohjia, joiden tarkoitus on auttaa kehittäjiä rakentamaan yleisiä ominaisuuksia sovelluksiin [19].

Android Studio ei ole pelkästään keskittynyt kehittämään sovelluksia mobiilipuhelimille ja -tableteille, vaan mahdollistaa kehittämisen muillekin laitteille, kuten Android Watch tai Android Auto [18]. Helpottaakseen kehittämistä ja testaamista Android Studiosta löytyy emulaattorit, joilla on mahdollista testata sovelluksia jo koneella, kehitysympäristössään [20].

3.1 Käyttöliittymä

Android Studion käyttöliittymä on jaettu alueisiin, joka helpottaa kehitysympäristössä navigointia. Oletuksena struktuurinavigointi-ikkuna on käyttöliittymän vasemmalla puolella, editori on isona keskellä ja käyttöliittymää kehittäessä esikatseluikkuna on oikealla, jos käyttää koodipohjaista kehitystä. Kuva 3 esittelee miltä käyttöliittymä näyttää tummassa teemassa, kun tehdään sovelluksen käyttöliittymän koodipohjaista kehitystä.



Kuva 3. Android Studion käyttöliittymä tummalla teemalla.

3.1.1 Editori-ikkuna

Editori-ikkuna on käyttöliittymän keskiosassa, joissa pääosa kehitystyöstä tehdään. Näihin pääosin kuuluu tekstieditori, jossa oletuksena tehdään kaikki ohjelmoinnit. Tämän lisäksi on sovelluksen käyttöliittymän kehitykseen erikoistunut editori. Kaikki tiedostot, jotka avataan editoriin, aukeavat omille välilehdille, jolloin niitä on helppo muokata, kun kehitetään esimerkiksi kahden objektin tai aktiviteetin välistä interaktiota.

Sovelluksen käyttöliittymäeditori

Sovelluksen käyttöliittymän kehittämistä varten Android Studioon on rakennettu oma editori käyttöliittymän kehittämiseen. Tämä käyttöliittymäeditori tukee edelleen tekstipohjaista käyttöliittymän kehittämistä kuin myös visuaalisempaa vedä ja pudota ominaisuudella. Nämä eri kehittämisen tavat on jaettu välilehdillä mutta käyttöliittymän kehittämisessä voi käyttää molempia tapoja.

Vedä ja pudota-ominaisuus on hyvä nopeaan aloittamiseen kehityksessä. Tässä menetelmässä käyttöliittymä, jota ollaan kehittämässä, näkyy keskellä Android Studiota, vasemmassa ylänurkassa on lista moduuleja, joita voidaan vetää käyttöliittymään ja moduulilistan alapuolella on hiearkialista. Hiearkialistasta aukaistaan näkymä moduulien ominaisuuksiin ja asetuksiin, tämä sijoittuu oletuksella Android Studion oikeaan reunaan. On mahdollista nähdä käyttöliittymä pohjapiirustusnäkymässä, jolloin on helpompi nähdä, mitkä näkymät ovat mihinkin yhteyksissä tai tukeutuneet relatiivisessa positiossaan.

Koodipohjainen käyttöliittymä käyttää XML-kieltä. Kehitys muuttuu nopeammaksi ja tehokkaammaksi, kun sitä käyttää enemmän ja sen kanssa on helpompi tehdä tarkkoja muutoksia. Tätä menetelmää käyttäessä oletuksena Android Studion oikeassa reunassa on esikatseluikkuna, jonka avulla nähdään visuaalisesti, miltä kehitettävä käyttöliittymä näyttää sillä hetkellä.

3.1.2 Navigointi-ikkuna

Navigointi-ikkuna löytyy oletuksena Android Studion vasemmasta reunasta. Tämä ikkuna pitää sisällään projektin kansiostruktuurin. Näkymä, minkä tämä ikkuna esittää, voidaan muuttaa niin, että se näyttää Android Studion projektistruktuurein tai esimerkiksi, miten sovelluksen kansiostruktuuri olisi laitteella. Kun tästä kansiosta aukaisee tiedoston, se aukeaa editoriin välilehdelle, missä sitä voi tarvittaessa muokata.

3.1.3 Emulaattori

Android Studion sisälle on rakennettu oma emulaattori helpottamaan sovellusten testausta ja siten nopeuttamaan kehittämisprosessia. Tosin on huomioitava, että virtuaalinen emulointi laitteesta ei aina suorita kaikkia toimintoja, joita oikea laite suorittaa. Tämän takia joitain ominaisuuksia voidaan testata vain fyysisellä laitteella.

Android Studiossa on valmiiksi tehtyjä virtuaalisia laitteita, puhelimien ja tablettien kohdalla ne keskittyvät enemmän Nexus- ja Pixel-laitteisiin. Studio ei kuitenkaan ole jättänyt pois mahdollisuutta tehdä mukautettuja laitteita. Laitteen omat ominaisuudet voidaan kertoa ja rajoittaa mitä

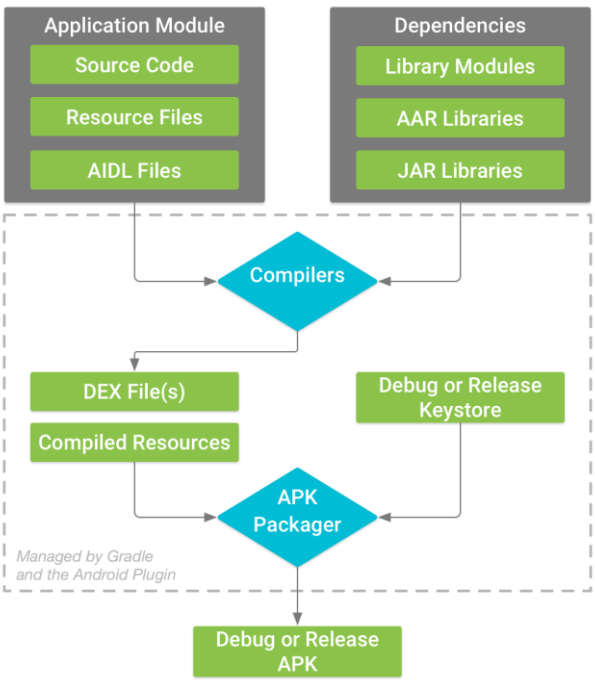
Android-käyttöjärjestelmä versiota kyseinen laite käyttää. Tällä tavalla kehittäjä voi testata vanhemmille laitteille ja käyttöjärjestelmäversiolle, ilman että joutuu hankkimaan jokaista varten oman fyysisen laitteen. Virtuaaliset laitevaihtoehdot yltävät myös muihin Android Studion tukemiin laitteisiin. Tällä tavoin mahdollistetaan, että voidaan kehittää esimerkiksi Android TV:lle sovellusta vaikka kyseistä laitetta ei omistaisi. On silti mahdollista kehittää ja testata sellaiselle laitteelle virtuaalisen laitteen avulla.

3.2 Sovelluksen paketointi

Androidin sovelluksen rakennusjärjestelmä mahdollistaa joustavan tavan tehdä mukautettuja paketointimuokkauksia, ilman että sovelluksen lähdekoodeja täytyy muokata. Se kokoaa sovelluksen resurssit, lähdekoodin ja paketoi ne APK:ksi, jonka avulla kehittäjä testaa, ottaa käyttöön, liittää tunnisteen ja jakaa. Android Studio käyttää paketoinnin automatisointiin ja hallinnointiin Gradlea, joka on edistynyt paketointityökalusetti. Gradle mahdollistaa paketointiasetusten mukauttamisen projektin tarpeisiin. Jokainen mukautettu paketointi voi määrittää oman osuuden koodista ja resursseista, samalla kun kyseinen paketointiversio käyttää muita yhteisiä osia sovelluksesta. Gradlen sisällä oleva Android lisäosa toimii rakennustyökalun kanssa tarjotakseen prosesseja ja muokattavat asetukset, jotka on erityisesti tarkoitettu Android sovellusten rakentamiseen ja testaamiseen. [21.]

Gradle ja Android-lisäosa pyörii riippumattomana Android Studiosta. Eli Android sovelluksia voidaan paketoida tietokoneen komentoriviltä tai tietokoneelta, jolle ei ole asennettuna Android Studiota. Paketoinnin tulos pysyy samana, ja koska nämä ovat riippumattomia Android Studiosta, nämä osat on päivitettävä erikseen, kun uusin päivitys on julkaistu. [21.]

Androidin sovelluksen rakennustyökalut kokoavat lähdekoodit ja resurssit yhteen tehden niistä APK-tiedoston, joka voidaan sitten asentaa laitteelle [21]. Tämä prosessi visualisoidaan kuvassa 4, jossa nähdään tarkemmin mitä osia rakennustyökalut paketoivat.



Kuva 4. Paketointiprosessi [21].

Prosessin ensimmäisessä vaiheessa kootaan projektin moduulit ja kirjastoriippuvuudet yhteen. Moduulit sisältävät sovelluksen lähdekoodin ja sovelluksessa käytettävät resurssit. Tämän jälkeen käännetään tämä DEX (Dalvik Executable) tiedostoiksi, nämä sisältävät tavukoodin, joka pyörii Android-laitteilla, ja muut tiedostot resursseiksi. Jotka paketoidaan lopuksi yhdeksi APK-tiedostoksi, johon samalla lisätään tunniste. Tunniste kertoo APK-tiedoston käyttötarkoituksen, eli onko se tarkoitettu testaukseen vai jaettavaksi yleisölle. Android Studio lisää testaamiseen tarkoitetun tunnisteen uuteen projektiin automaattisesti. Prosessin lopussa kehittäjällä on joko testattava tai julkaistava APK-tiedosto sovelluksesta. [21.]

Ennen kuin tehdään lopullinen APK-versio, joka julkaistaan yleisölle, paketointi optimoi sovelluksen käyttämään vähemmän muistia käyttämällä zipaling nimistä työkalua [21]. Tämän työkalun tehtävä on sovittaa arkiston pakkaamaton data mahdollisimman optimoidusti [22].

3.3 Gradle

Gradle on avoimen lähdekoodin sovelluksen rakennustyökalu, joka on suunniteltu tarpeeksi joustavaksi rakentamaan melkein minkä tahansa sovelluksen [16]. Gradle tekee oletuksia, joiden

avulla se määrittelee, minkälaista sovellusta kehittäjä on tekemässä ja miten se tulisi rakentaa. Tämän lisäksi Gradle helpottaa yleisten projektien kuten esimerkiksi Java-projektien rakentamista lisäämällä valmiiksi rakennettuja toimintoja lisäosilla, mukautetuilla työtehtävämalleilla ja monella muulla hyvin muokattavissa olevilla alueilla. Olisi suositeltavaa, että rakennetaan projektia, joka käyttää pelkästään Gradlen omia asetuksia, mutta realistisesti projekteissa on yleensä erikoisia ominaisuuksia, jotka vaativat oman rakennuskoodin. Tämän takia Gradle mahdollistaa omien lisäosien ja rakennuskoodien kehittämisen. Näin saadaan juuri niitä toimintoja, joita tarvitaan sovelluksen paketointiprosessissa. [23.]

Gradle:n rakennusskriptiä on helppo katsoa suoritettavana koodina, koska sitä ne ovatkin. Mutta se on implementaatioyksityiskohta, hyvin suunnitellut rakennusskriptit kuvailevat, mitä askelmia tarvitaan, että voidaan rakentaa sovellus, ei kuinka näiden askelmien tulisi toimia. Se työ kuuluu muokatuille työtehtävämalleille ja lisäosille. [23.]

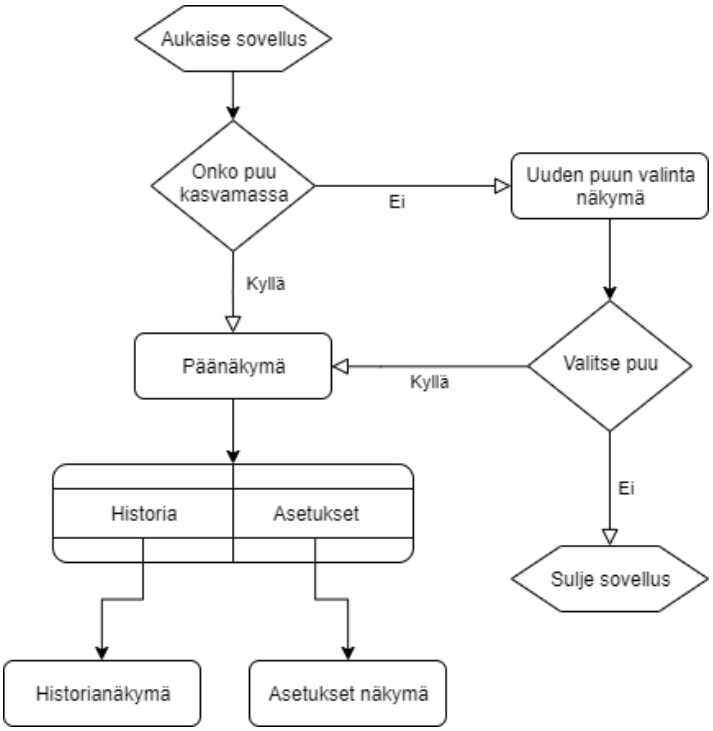
4 Kehitettävä sovellus

Sovelluksen tarkoituksena on auttaa ja motivoida juomaan vettä tarpeellinen määrä päivän aikana. Motivoimiseen käytetään apuna puuta, joka kasvaa, kun juodun veden määrä lisätään sovellukseen. Tarkoittaen, että kun juo vettä, se kasvattaa puuta ja puu voi kuihtua, jos ei juo tarpeeksi.

Sovellusta ei ole tarkoitus saada täysin valmiiksi tämän opinnäytetyön aikana, vaan tarkoituksena on saada mekaniikat ja perustoiminnot kehitettyä. Niitä tekemällä oppia oikeanlaisia ohjelmointikäytäntöjä Android Studion kanssa. Sovelluksen olisi tarkoitus olla lopussa sellaisessa tilassa, että siihen ei olisi tarvetta tehdä suuria mekanismeja, vaan se olisi enemmän grafiikan implementointia ja pieniä korjauksia, sekä lisäämään selkeyttä ja visuaalista informaatiota käyttäjälle tekstipohjaisen informaation tueksi.

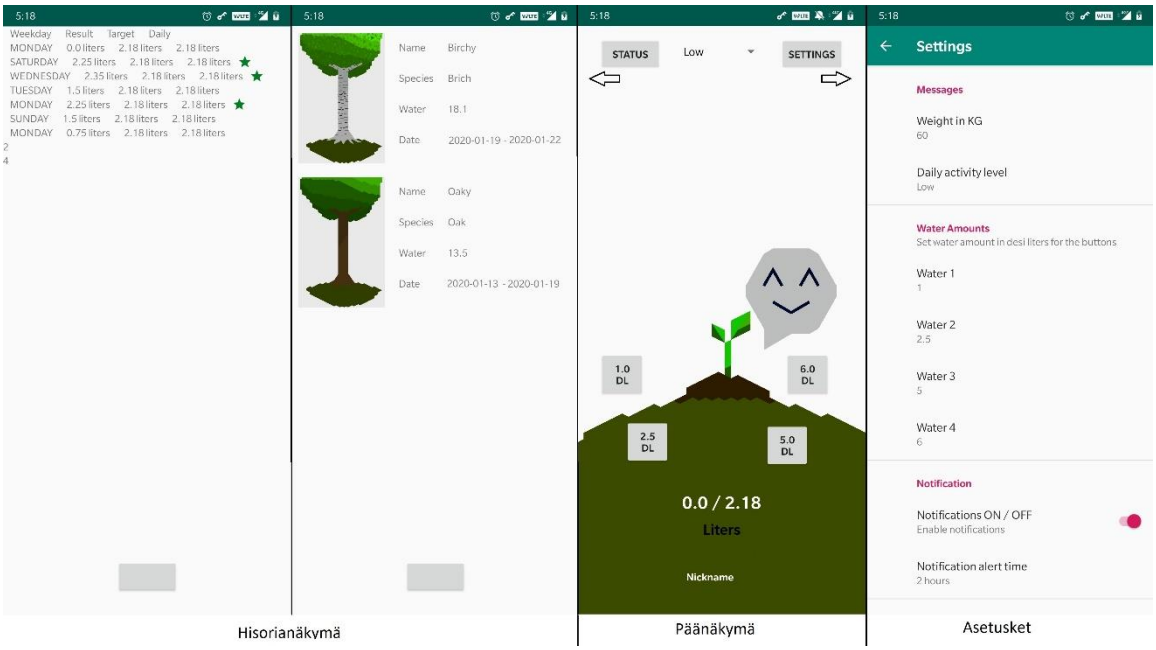
Projektin alussa sovelluksen aktiviteettien käyttöjärjestys vaikutti hyvin yksinkertaiselta, sillä se tarvitsisi vain muutaman näkymän joiden avuilla sovellusta käytetään. Tämän käyttöjärjestyksen kehityksen jälkeen sitä jatkettiin siten, että sovellus käynnistyy oikein riippuen siitä, onko sovelluksella jo puu kasvamassa. Tulevaisuudessa olisi tarkoituksena kehittää ominaisuus, jossa ensimmäisellä käynnistyskerralla kysytään käyttäjän paino ja aktiivisuustaso, millä lasketaan päivittäisen veden määrä. Tulisi myös kehittää jonkinlainen tutoriaalimekanismi, joka pyörii läpi ensimmäisen kerran, jos käyttäjä niin haluaa, tai sen voi käynnistää asetuksista.

Seuraava kaavio kuvaa, mitä tapahtuu sovelluksen käynnistymisen yhteydessä. Kun sovellus käynnistyy pääaktiviteettiin, se tarkistaa useita asioita ennen kuin mitään näytetään käyttäjälle, mutta tämä kaavio on yksinkertaistettu näyttämään vain, miten liikutaan aktiviteettinäkymien välillä.



Kuva 5. Liikkuminen aktiviteettien välillä.

Kun sovellus avataan ensimmäisen kerran tai sillä ei ole puuta, jota se kasvattaa, sovellus vie käyttäjän puunvalintanäkymään, jossa käyttäjä voi selata eri puuvaihtoehtoja. Tästä sitten palataan päänäkymään puunvalinnan jälkeen. Kuvassa 6 nähdään kaikki yleisimmät näkymät, joiden kanssa käyttäjä on vuorovaikutuksessa. Päänäkymä käynnistyy suoraan, jos sovelluksella on jo puu kasvamassa. Tästä näkymästä voidaan sitten liikkua historia- ja asetukset-näkymiin painikkeilla, joiden avulla käyttäjä voi olla vuorovaikutuksessa sovelluksen kanssa.



Kuva 6. Sovelluksen yleisimmät näkymät.

Käyttäjä on eniten vuorovaikutuksessa päänäkymän kanssa. Tässä näkymässä käyttäjä näkee puun, paljonko vettä on juotu ja mikä on tavoite. Näkymässä lisätään veden määrä, vaihdetaan päivän aktiivisuustasoa tai liikutaan muihin aktiviteettinäkymiin, jotka ovat historia- tai asetukset-näkymä.

Asetukset-näkymässä käyttäjä voi muuttaa tietoja, joilla lasketaan, paljonko vettä käyttäjän olisi juotava päivässä sekä asettaa päänäkymän painikkeille vesimäärät, jotka lisätään puulle. Lisäksi voidaan valita kellonaika, jolloin sovellus nollaa päivittäisen tuloksen sekä ovatko ilmoitukset käytössä ja paljonko aikaa menee, että sovellus muistuttaa käyttäjää.

Historia-näkymän tarkoitus on kertoa käyttäjälle, kuinka hyvin on suoriutunut viikon aikana ja nähdä, montako puuta on saanut kasvatettua. Tiedot on tallennettu tekstitiedostoihin, joista tämä aktiviteetti lukee ne.

Sovelluksessa on myös kaksi muuta näkymää, jotka aktivoituvat tietyissä tilanteissa. Nämä ovat uuden puun valintanäkymä ja pienohjelman näkymä.

Kuva 7. Puunvalintanäkymä.

Puunvalintanäkymässä valitaan, minkä puun kasvattamisen käyttäjä aloittaa. Puut näytetään yksi kerrallaan, ja niitä voi selata kahden painikkeen avulla. Puun valittuaan käyttäjä voi antaa puulle lempinimen.

Pienohjelman näkymä on hieman erikoinen, koska se käynnistetään vain, kun pienohjelmaa painautetaan ja silloinkin se vain näyttää valikon. Tästä valitaan, paljonko vettä lisätään sovelluksessa olevalle puulle.

5 Vesimäärien lisääminen ja seuraaminen

Veden lisääminen sovelluksessa on tehty helpoksi siten, että käyttäjä voi asettaa neljälle painikkeille omat desilitra-arvot asetukset-näkymässä. Päänäkymässä painikkeen avulla voidaan lisätä kyseisen vesimäärä puulle. Sovelluksen päänäkymän alalaidassa on tekstinäkymä, joka ilmoittaa, paljonko vettä on juotu ja paljonko olisi tavoitteena.

5.1 Vesitavoitteen laskeminen

Tarvittavan veden laskemisen kaavana käytetään käyttäjän painoa, joka kerrotaan 0,033 ja sen tulokseen lisätään liikunnalliseen aktiviteettitasoon sopiva desilitramäärä. Tämä on yleisin kaava mitä käytetään laskettaessa päivittäistä vesimäärää, mikä henkilön tulisi juoda. Vesitavoitteen kaavan liikunnallisen aktiviteettitaso on säädetty pienemmäksi, kuin mitä yleisimmät kaavat käyttävät. Tällä tavoitellaan sitä, että otetaan huomioon veden määrä jonka käyttäjä saa ruuasta ja muista juomista.

Vesitavoitteen kaavat:

perusta: (paino * 0,033) + liikkumisen aktiviteettitaso

Matala: (paino * 0,033) + 0,2

Normaali: (paino * 0,033) + 0,3

Aktiivinen: (paino * 0,033) + 0,4

Hyvin Aktiivinen: (paino * 0,033) + 0,5

Vesimäärät, käyttäjän paino, oletus liikunnallinen aktiviteettitaso ja oletus veden juomisen tavoite tallennetaan sovelluksen muistiin käyttämällä jaetut asetukset ominaisuutta. Tällä tavalla, kun tulee uusi päivä, niin oletusarvot on helppo palauttaa, sillä päänäkymässä käyttäjän on mahdollista muuttaa kyseisen päivän liikunnallista aktiviteettia pienemmälle tai suuremmalle tarpeen mukaan, jolloin vesimäärän tavoite mukautuu tarpeisiin.

Käyttäjillä on erilaisia päivärytmejä, jonka takia päivän nollaaminen tapahtuu oletuksena kello 24:00, mutta asetuksista se on mahdollista muuttaa tasatunnein kello 03:00 asti. Kun päivä nollautuu, sovellus tallentaa edellisen päivän vesimäärän, oletustavoitteen ja päivän oman tavoitteen tekstitiedostoon. Käyttäjä voi sitten katsoa näitä arvoja historianäkymästä. Tekstitiedostoon tallennetaan myös puiden tiedot, kun ne ovat kasvaneet täyteen kokoon.

5.2 Käyttäjän motivointi

Puut lisäävät motivointia sillä, ne kasvavat aina, kun on juonut tietyn määrän. Jotta puiden kasvattaminen ei olisi liian helppoa, kasvaminen hidastuu, mitä suuremmaksi puu kasvaa. Tämä on laskettu siten, että tarpeellinen veden määrä kerrotaan joko kahdella tai kolmella, jonka tulokseen lisätään prosenttimäärä, joka on laskettu vesitavoitteesta.

Kaavat:

Taimi: (Vesitavoite * 2) + (vesitavoite * 0,1)

Pieni: (Vesitavoite * 2) + (vesitavoite * 0,25)

Keskikokoinen: (Vesitavoite * 3) + (vesitavoite * 0,5)

Täysikasvuinen: (Vesitavoite * 3) + (vesitavoite * 0,75)

Tämän takia puilla on omat prosessi- ja tavoitearvot tallennettuna, kun niitä kasvatetaan. Prosessiarvoa päivitetään joka kerta, kun käyttäjä lisää vettä, ja tavoitearvo päivitetään, kun alkuperäinen tavoite on saavutettu ja puu kasvaa. Kun puu on kasvanut täyteen pituuteensa, kysytään käyttäjältä, haluaako hän liikuttaa puun puutarhaan, jolloin saa uuden puun kasvatettavakseen. Kun puu liikutetaan puutarhaan, tallennetaan päivämäärä, koko vesimäärä, mikä puun kasvattamiseen on merkitty, sekä lajin ja käyttäjän antama nimi. Näitä voidaan tarkastella historianäkymästä.

Puille kehitettiin myös alustava hyvinvointitaso, jolla ilmaistaan, kuinka hyvin käyttäjä joi vettä edellisenä päivänä. Kun uusi päivä alkaa, edellisen päivän vesimäärää verrataan tavoitteeseen. Jos se on parempi kuin 75 prosenttia tavoitteesta, hyvinvointitaso kasvaa, ja jos se on alle 50

prosenttia, hyvinvointitaso laskee. Kun hyvinvointitaso on matalimmalla, puu kuolee ja käyttäjän on aloitettava puun kasvatus alusta.

Tulevaisuudessa olisi tarkoituksena kehittää hyvinvointitasoihin mekanismi, joka ottaa huomioon, mitä puuta kasvatetaan, että joidenkin puiden kasvattaminen olisi haastavampaa. Sekä mahdollisuus määrittää asetuksissa valmiiksi, mitkä päivät ovat korkeita ja mitkä matalia liikunnallisen aktiivisuuden kannalta.

6 Tietojen tallentaminen

Sovelluksen tietojen tallentamiseen on monia vaihtoehtoja, ja kaikissa on omat hyvät ja huonot puolensa. Tämän takia on tärkeää arvioida, paljonko ja minkälaista tietoa sovellukseen täytyy tallentaa. Sen avulla voi valita sopiva tallennusominaisuus. Tässä projektissa käytettiin jaetut asetukset ominaisuutta ja tekstitiedostoon tallentamista sekä tietoja, jotka eivät muutu, on tallennettu projektin resurssitiedostoihin.

6.1 Jaetut asetukset

Jaetut asetukset (Shared preferences) on yksi monista ominaisuuksista, miten tallentaa tietoa Android-sovelluksissa, että tieto ei katoa käyttäjän sulkiessa sovelluksen. Jaetut asetukset tallentaa tiedon avainnimi- ja arvopareina. Se on helppo ottaa käyttöön projektissa, toisin kuin muut tiedon tallennusvaihtoehdot, jotka tarvitsevat usein paljon koodia. Jaettujen asetusten tallentamat tiedot poistetaan joko erillisellä ”Clear”-komennolla sovelluksen sisällä, tai laitteen asetuksista tyhjentämällä sovelluksen välimuisti tai kun sovellus poistetaan laitteelta. [24.]

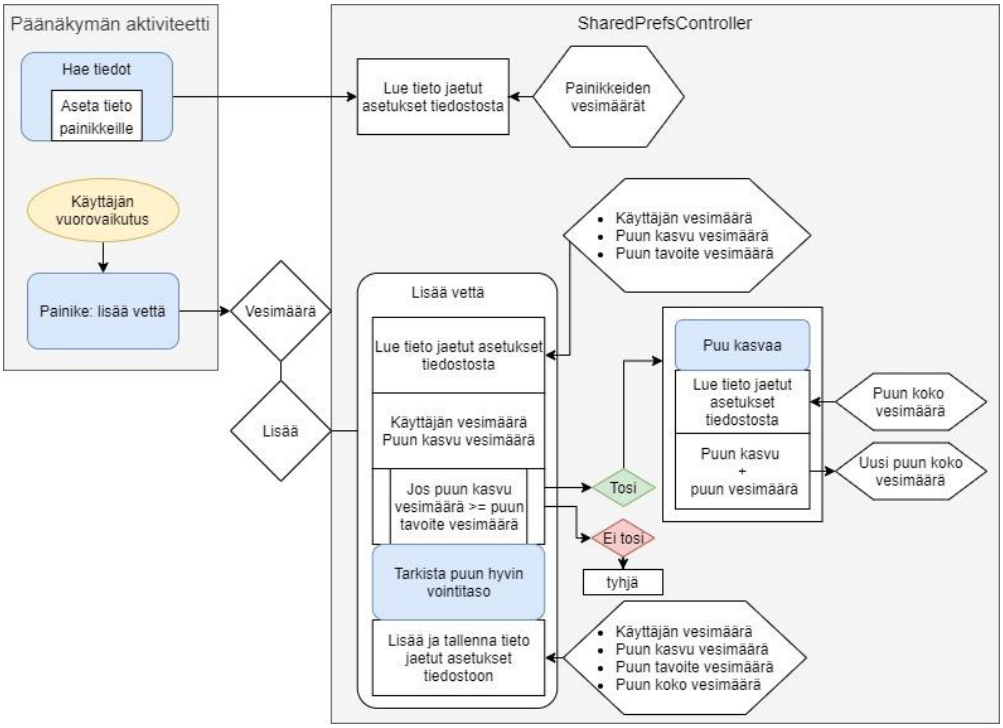
Tätä tallennustyyppiä käytetään yleensä tallentamaan sovelluksen asetusten tilat, kuten esimerkiksi onko käyttäjä kirjautunut sisään tai ovatko ilmoitukset päällä vai ei [24]. Tässäkin sovelluksessa sitä käytetään siihen, mutta sen lisäksi se tallentaa myös paljon tietoa sovelluksesta, kuten käyttäjän vesimäärän tavoite ja paljonko käyttäjä on juonut tämän päivän aikana. Mutta sovellus ei tallenna päivittäisiä tuloksia tai puun tilastoja, kun puu on kasvanut kokonaan. Nämä tiedot tallennetaan erillisiin tekstitiedostoihin. Tämä tehdään, koska on hyvä tapa pyrkiä pitämään jaettujen asetusten tallentamat tiedot pienenä määränä [25].

Jaetut asetukset on helppo ottaa käyttöön muutamalla rivillä koodia. Jaetut asetukset tiedostoja voi olla useita, ja ne erotellaan toisistaan String-muuttujalla, joka toimii tiedoston nimenä [25]. Tämän nimen tulisi olla täysin uniikki kehitettävälle sovellukselle [25]. Näin tallennettavia arvoja voidaan lajitella useampaan tiedostoon ja rajoittaa, mihin tietoihin mikäkin ominaisuus pääsee tekemään muutoksia tai lukemaan niitä. Tällä erottelulla voidaan myös varmistaa, että aukaistaan aina sama jaetut asetukset-tiedosto.

Jotta jaettujen asetusten tiedostoon voidaan tallentaa, on hyvä ensin initialisoida Preferences Editori, tämä helpottaa koodin lukua. Tämän jälkeen editoria käyttämällä lisätään tietoja tallennettavaksi ja lopuksi kutsutaan editor.apply()- metodia, joka tallentaa nämä tiedot.

Tämän projektin täytyy tarkistaa ja muokata jaettujen asetusten tiedoston tietoja suhteellisen useasti. Tätä varten jaettujen asetusten käyttöä helpottamiseksi tehtiin olento nimeltä SharedPrefController, joka lisätään aktiviteettiin, mikäli aktiviteetti tarvitsee tallennettuja tietoja. Koska tämä on omassa olennossa, täytyy objektin initialisoinnissa antaa olennolle sovelluksen konteksti. Initialisointimetodissa luodaan Preferences Editori valmiiksi, että sitä ei tarvitse luoda uudelleen joka kerta, kun sovelluksen tulee käyttää sitä. Poikkeuksellisesti jaetut asetukset on lisätty suoraan historia-aktiviteetille, koska sen tulee lukea vain muutamaa arvon eikä sen tarvitse tallentaa sinne mitään.

SharedPrefsController-olentoon tehtiin metodeja, joiden tarkoitus on joko muokata tai verrata jaettujen asetusten tietoja. Seuraavassa kaaviossa kuvataan, miten päänäkymän aktiviteetti kommunikoi SharedPrefsController-olennon kanssa hakemalla käyttäjän asettamat vesimäärät jaetuista asetuksista sovelluksen painikkeiden käyttöön ja mitä olennon metodissa tehdään, kun puulle lisätään vettä.



Kuva 8. Vesimäärän lisäämisen prosessikaavio.

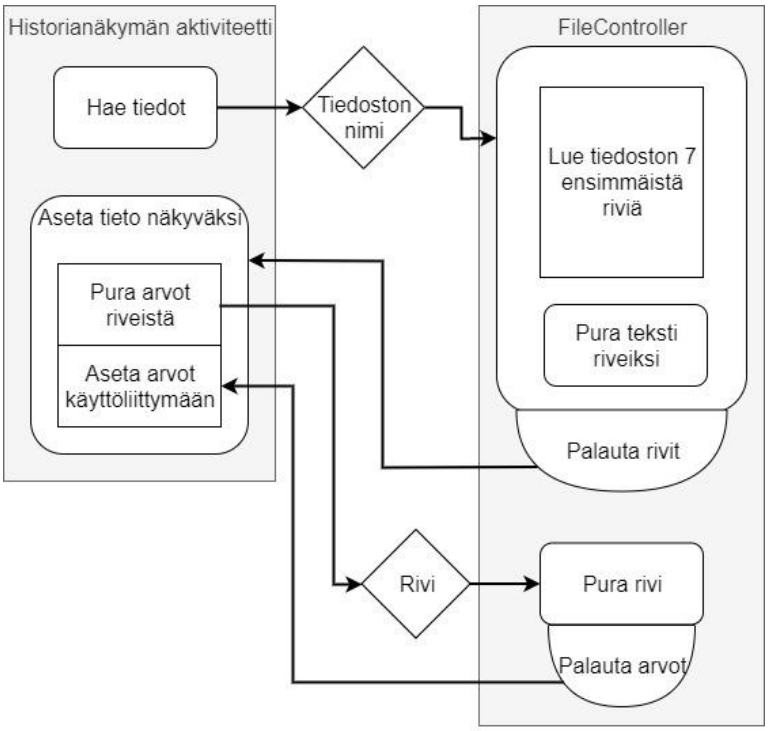
Vesimäärät on helppo tallentaa metodilla, koska kaikki on float-arvoja, mutta esimerkiksi kun asetuksia tallennetaan, niin asetukset-aktiviteetti kirjoittaa rivi kerrallaan ja lopuksi kutsutaan metodia, joka lisää vielä pari arvoa tallennettavaksi ennen kuin kutsutaan editorin apply()-metodia tallentamaan tiedot.

6.2 Tekstitiedoston käsittely

Tekstitiedosto on kevyempi kuin tietokanta, kun otetaan huomioon tiedon määrä, joka pitää tallentaa tietokantaan, joka liian yliampuva tälle sovellukselle. Varsinkin kun olisi tarkoituksena pitää sovellus kevyenä. Tietenkin tekstitiedostoon tallentaminen voi olla raskasta, jos tietoa on paljon. Tekstitiedosto kirjoittaa itsensä uudelleen, kun sinne lisätään tietoa. Tämän takia täytyy tietoa rajoittaa, paljonko tiedostoon tallennetaan. Esimerkiksi tämä projekti tallentaa omiin tekstitiedostoihin vain 50 riviä. Tämä valintaperustelu tehtiin siksi, että käyttäjälle näytetään vain viikon historia tai mahdollisesti tulevaisuudessa kyseisen kuukauden historia. Tämän hetken tarpeisiin 50:en puun tallennus on tarpeeksi, ehkä jopa hieman liikaa. Kun nämä rivit on täytetty tiedolla, sovellus alkaa kirjoittamaan uusimman rivin vanhimman tilalle.

Tekstin kirjoittaminen ja lukeminen tekstitiedostosta on suhteellisen yksinkertainen. Kaikki tekstitiedostosta lukeminen ja kirjoittaminen on siirretty omaan FileController-nimiseen olentoon koodin selventämisen vuoksi. Olennon alussa varmistetaan, että tiedostoille on kansio. Se tehdään, jos sitä ei ole. Metodit joko kutsuvat tämän olennon sisällä olevia Async Task objekteja tai parsii niiden tuloksia ennen kuin palauttaa ne aktiviteetille, joka metodia kutsui.

Tekstitiedoston lukemiseen käytetään kahta menetelmää kyseisellä hetkellä. Toinen on historian lukemiseen, jolloin luetaan vain seitsemän ensimmäistä riviä. Enempää ei ole tarvetta lukea, koska tällä hetkellä käyttäjälle näytetään vain viikon aikainen historia. Kuvassa 9 näytetään prosessi minkä historianäkymän aktiviteetti käy läpi, saadaksemme tiedot näkymään käyttäjälle. Toiseen tiedoston lukemiseen käytetään AsyncTask-objektia, joka voi lukea molemmat tiedostot. Tämä onnistuu siten, että AsyncTaskin initialisoinnissa se saa String-muuttujat, jotka ovat tiedoston sijainti ja nimi. Sitä käytetään kyseisellä hetkellä vain valmiiden puiden tietojen lukuun.



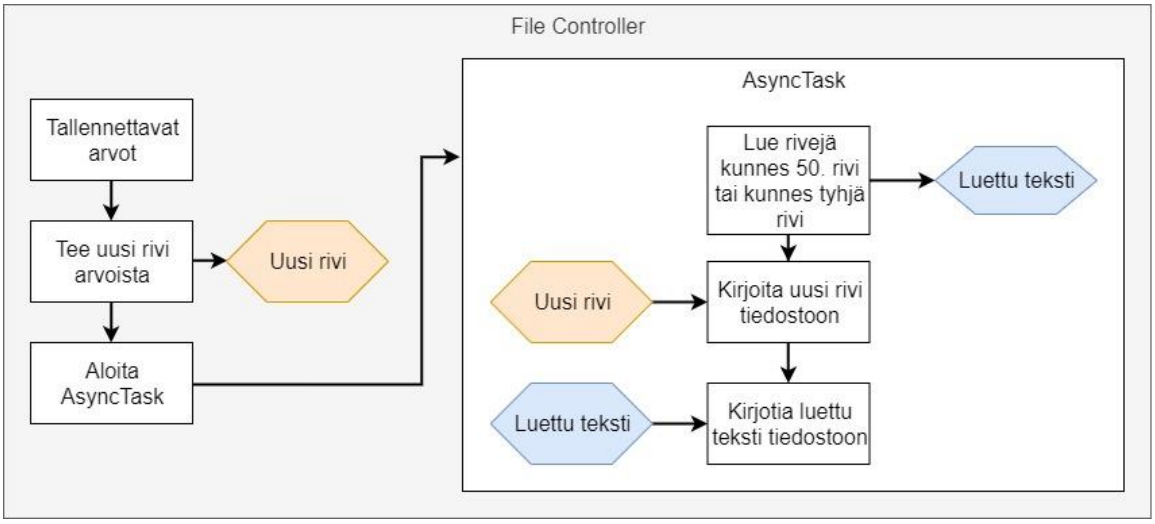
Kuva 9. Viikon aikaisen historian lukeminen tekstitiedostosta.

6.2.1 AsyncTask

Joskus sovelluksen täytyy suorittaa resursseille raskaita tehtäviä kuten esimerkiksi ladata tiedostoja tehdä tietokantakyselyjä, soittaa mediaa tai laskelmoida monimutkaisia analyyttisia laskentoja. Tämän tyyppinen intensiivinen prosessointi voi estää käyttöliittymän suoritusketjun, jolloin sovellus ei reagoi käyttäjän syötteeseen tai ei piirrä, eikä päivitä näyttöä. Tästä syystä käyttäjä voi turhautua, kun sovellus ei toimi kunnolla ja poistaa sen. [26.]

Jotta sovelluksen käyttökokemus olisi mukava ja sulava Android kehys tarjoaa avustajaobjektin nimeltä AsyncTask, joka prosessoi toisessa suoritusketjussa kuin käyttöliittymälle varatussa [25]. Käyttämällä AsyncTask-toimintoa tekemään raskaammat prosessit tarkoittaa, että käyttöliittymän suoritusketju pysyy reagoivana [26]. Tämän toiminnon käyttöä suositellaan lyhyille prosesseille, jotka kestävät maksimissaan muutaman sekunnin [27]. Jos prosessi on pidempi tai sen pitää pyöriä todella pitkään aikaan, suositellaan käyttämään kirjastoja, jotka on suunniteltu taustaprosessiketjujen suorittamiseen [27].

AsyncTask-ominaisuutta käytetään tässä projektissa, koska on hyvin mahdollista, että sovellus suljetaan, kun kirjoittaminen on kesken. Silloin sovellus tuhoaa sen prosessin ja tiedot eivät tallennu. Async jää suoriutumaan taustalle, kunnes se saa työnsä tehtyä. AsyncTask-objektit on lisätty FileController-olennon sisään, että kaikki tiedoston lukemiseen ja kirjoittamiseen on samassa kooditiedostossa. AsyncTask tekee lukemisen ja kirjoittamisen metodissa, jonka nimi on doInBackground(). Seuraavassa kaaviossa on kuvattuna, miten tekstitiedostoon kirjoitetaan uusi rivi käyttämällä AsynkTask-ominaisuutta.



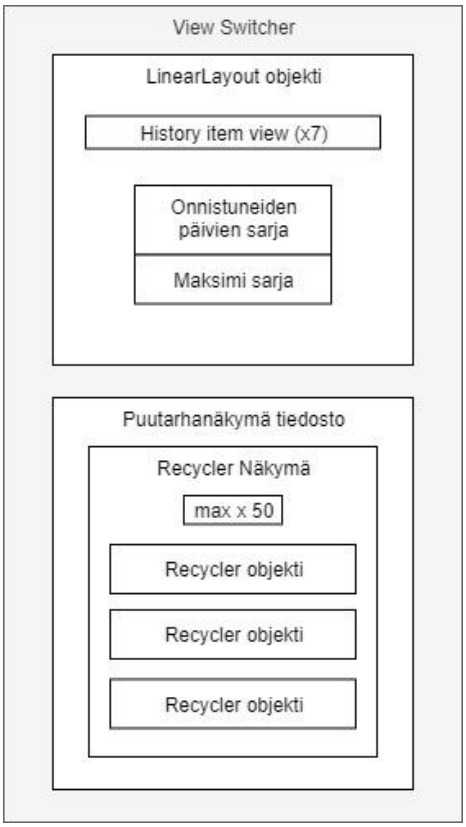
Kuva 10. Uuden rivin kirjoittaminen AsycTaskissa.

Uuden rivin kirjoittaminen tehdään enimmäkseen kokonaan AsyncTaskissa. Ensin kutsutaan metodia, joka luo tallennettavan tekstin annetuista arvoista. Sen jälkeen käynnistetään AsyncTask, mille annetaan tiedoston sijainti ja nimi sekä lisättävä teksti. AsyncTask-objekti lukee tekstitiedoston, ja tallennetaan luettu teksti muuttujaan. Huomiona kuitenkin, että tekstin lukeminen lopetetaan, jos rivejä on enemmän kuin 50. Tämän jälkeen uusi rivi kirjoitetaan ylimmäiseksi ja sitten lisätään tallennettu muuttuja.

6.2.2 Tekstitiedoston tallenteiden esittäminen

Historia-aktiviteetin näkymässä käytetään ViewSwitcher-nimistä näkymäluokkaa. Tämän luokan sisälle voidaan laittaa vain kaksi näkymäobjektia [28]. ViewSwitcher hallitsee kumpi, näkymistä

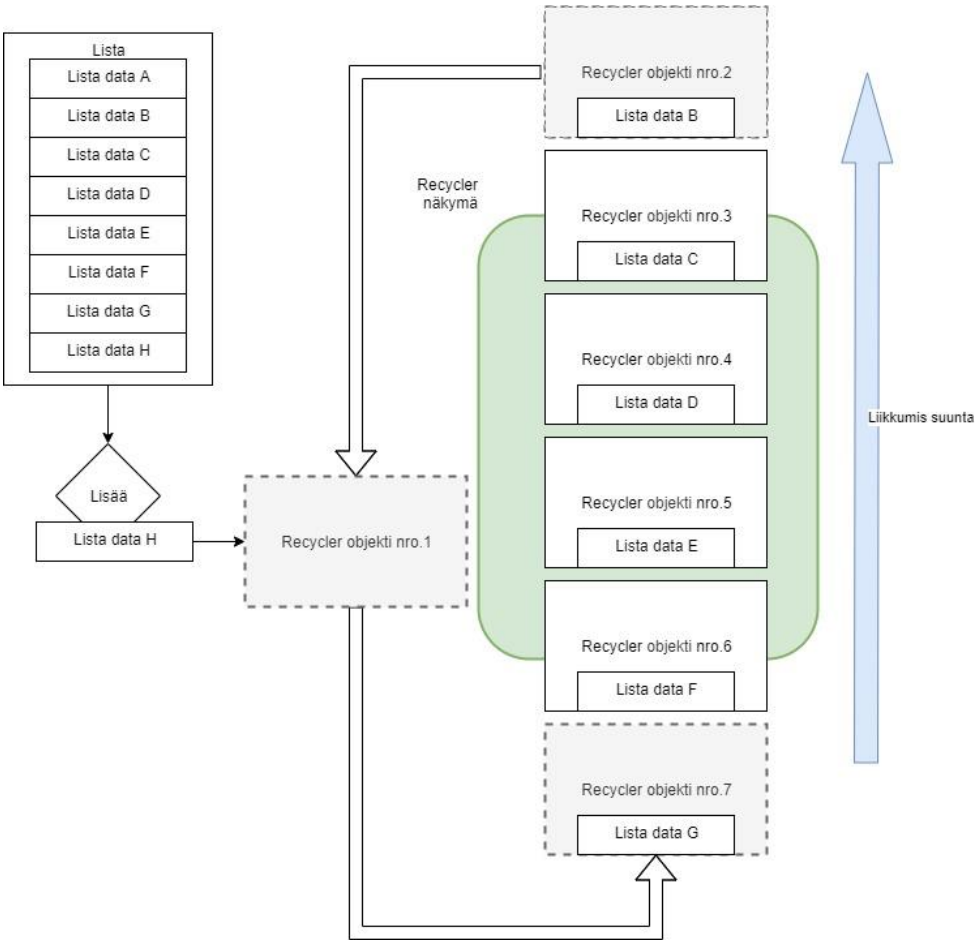
on aktiivisena käyttäjälle [28]. Kahden näkymän rajoituksen voi kuitenkin kiertää lisäämällä LinearLayout-nimisen näkymäorganisointiobjektin tai lisäämällä erillisen näkymäobjektitiedoston, joiden sisälle voidaan laittaa useampi näkymäobjekti. Kuvassa 11 on kuvattuna näkymä ja miten näkymäobjektit asettuvat hierarkkisesti.



Kuva 11. Historia-aktiviteetin näkymäkaavio.

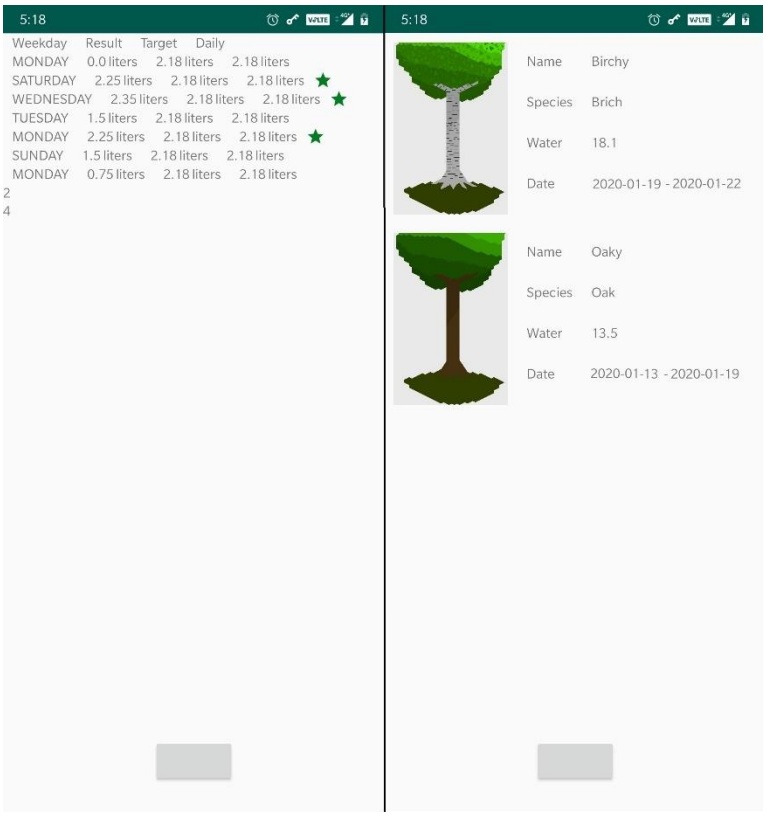
Seitsemän päivän historiaosuus näyttää tiedot lineaarisessa näkymälistassa, minkä sisällä on useita näkymäobjekteja, yksi jokaiselle viikonpäivälle. Nämä objektit pitävät sisällään tekstinäkymiä ja yhden kuvanäkymän, missä näytetään tähti, mikäli on juonut tavoitteen vaatiman määrän. Listan loppuun on lisätty myös kaksi tekstinäkymää, joissa näkyy, montako päivää peräkkäin käyttäjä on suoriutunut tavoitteessaan ja monenko päivän sarja on maksimissaan saavutettu.

Puutarha esitetään Recycler-nimisessä näkymässä erillisinä näkymäobjekteina. Recycler-näkymä on yksi suosituimmista näkymistä, joita käytetään, kun tehdään isoja tai loputtomia listoja. Tämä johtuu siitä, että recycler näkymä säästää muistia, koska se ei tee kaikkia listan objekteja valmiiksi, vaan käyttää uudelleen listan objektit päivittämällä niiden tiedot seuraavalla listassa [29]. Seuraavassa kaaviossa on kuvattu miten recycler kierrättää näkymä objekteja näyttääkseen listan tiedot.



Kuva 12. Recycler-näkymän toimintakaavio.

Tätä käytetään kasvatettujen puiden esittämisessä, koska listan objekteihin voi laittaa useamman näkymän, jotka näyttävät tiedot ja kuvan puusta. Jossakin välissä täytyy mahdollisesti näyttää kaikki 50 puuta, jotka on tallennettu ja tällä tavalla sovelluksen tulisi pysyä kevyempänä proses-siltaan sekä muistin käytöltään.



Kuva 13. Historia-aktiviteetin näkymistä.

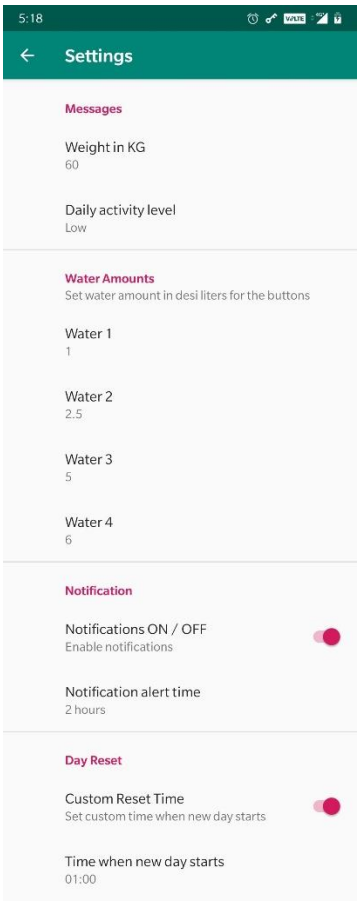
Ylemmässä kuvassa nähdään miltä historia-aktiviteetin näkymäobjektit näyttävät sovelluksessa. Tulevaisuudessa tästä näkymästä on tarkoitus tehdä paljon selkeämpi ja visuaalisesti mukavampi. Tämä toteutettaisiin kehittämällä pylväskaaviot päivittäisille vesimäärille. Tällöin olisi helpompi seurata, miten hyvin käyttäjä on juonut vettä ilman, että täytyy verrata lukuja sekä mahdollisesti lisätä muita käyttäjän tietoja, kuten esimerkiksi kyseisen päivän vesimäärätavoite ja vesimäärän oletustavoite.

7 Asetukset

Asetukset mahdollistavat käyttäjää muuttamaan sovelluksen toimintoja ja käyttäytymisiä sallituilla tavoilla. Ne voivat vaikuttaa sovelluksen taustalla pyöriviin käyttäytymisiin, kuten esimerkiksi, kuinka usein sovellus synkronoi itsensä pilvipalveluiden kanssa, tai asetukset voivat olla paljon laajemmin vaikuttavia kuten muuttaa käyttöliittymän ulkonäköä. [30.]

Asetukset toiminto on niin yleinen sovelluksissa, että Android Studiosta löytyy valmis pohja, joka voidaan lisätä projektiin helposti. Tämä pohja käyttää AndroidX-kirjastoa, jonka käyttö on juuri asetuksia tehtäessä suositeltavaa [30]. Kirjasto hallinnoi käyttäjälle näkyvää käyttöliittymää ja kommunikoi tiedon tallennuksen kanssa, että kehittäjä voi määritellä yksittäiset asetukset, joita käyttäjä voi muokata [30]. AndroidX-kirjasto sisältää myös materiaaliteeman, joka varmistaa vakaan käyttäjäkokemuksen kaikilla laitteilla ja käyttöjärjestelmäversioilla [30].

Kun lisää asetukset aktiviteettipohjan huomaa, että käyttöliittymän xml-koodi eroaa hieman muiden aktiviteettien käyttöliittymän xml-koodista ja sen tiedosto tallentuu xml-kansioon. Asetusten käyttöliittymä käyttää preferenceScreen-luokkaa, joka on osa Preferences kirjastoa. Asetukset ovat aktiviteetissa fragmentin sisällä, fragmentit helpottavat asetusten kehittämistä, kun asetusten valikossa tarvitaan useampaa valikkonäkymää. Niin uusi tai sisempi valikko on uusi fragmentti, joka näytetään käyttäjälle [31]. Tällä hetkellä tässä sovelluksessa ei ole niin paljon asetuksia, että olisi suositeltavaa käyttää useampaa valikkonäkymää. Kuvassa 14 nähdään kaikki asetukset, joita käyttäjä voi muokata.



Kuva 14. Sovelluksen asetukset valikko.

Sovelluksen asetusten valikossa ylimpänä on käyttäjän paino- ja aktiivisuusvalinnat, joiden alla on vesimäärien asetukset. Näiden alla on ilmoitukset ja päivän nollaamiseen kuuluvat asetukset. Ilmoitukset voi ottaa pois päältä tai valita listan vaihtoehdoista, minkä ajanjakson välein sovelluksen kuuluu muistuttaa käyttäjää. Asetusten valikon koodissa myös määritellään, että vaihtoehtoista aikaa ei voi valita, elleivät ilmoitukset ole päällä. Päivän nollaamiseen liitettävää asetusta ei voi ottaa pois päältä, mutta sitä on mahdollista muokata, kuinka myöhään yöllä nollaamisen tulee tapahtua.

Asetukset tallentavat omaan jaettujen asetusten tiedostoon automaattisesti, kun asetuksia muokataan. Mutta tämä sovellus tallentaa asetusten valinnat uudelleen sovelluksen muiden tallennettujen arvojen kanssa, aktiviteetin OnStop()-metodissa. Tämä on tehty siksi että testauksen aikana täytyy seurata tallennettuja arvoja, mikä on helpompaa kaikkien ollessa samassa tiedostossa. Aktiivisen kehitystyön kannalta on tässä tilanteessa järkevämpää pitää ne samassa tiedostossa, kunnes tehdään lopullinen ja julkaistava versio.

Jälkikehityksessä lisätään vaihtoehtoja, jotka muokkaavat käyttöliittymää kuten esimerkiksi kielen vaihto tai käyttöliittymän värien vaihto. Sekä muita asetuksia, joilla muokata sovelluksen antamaa käyttäjäkokemusta.

8 Ilmoitukset

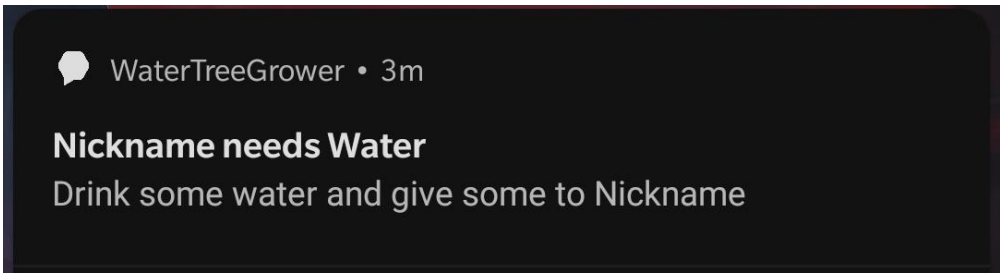
Joskus kehittäjät tahtovat sovelluksen näyttävän informaatiota käyttäjälle, kun sovellus itsessään ei ole aktiivisena. Tämä tehdään esimerkiksi silloin, kun uutta sisältöä on tarjolla tai kun käyttäjän suosima urheilujoukkue teki maalin. Androidin ilmoitusmekanisminkehys mahdollistaa sovelluksen tehdä ilmoituksen, kun sovellus on suljettuna. [32.]

Ilmoitus on viesti, joka näyttää informaatiota sovelluksen normaalin käyttöliittymän ulkopuolella. Ilmoitukset ilmaantuvat kuvakkeena ilmoituspalkkiin, josta sitä voi katsella tarkemmin aukaisemalla ilmoituslaatikon, esimerkiksi pyyhkäisemällä ilmoituspalkkia alaspäin. Ilmoitusalue ja ilmoituslaatikko on käyttöjärjestelmän hallinnoimia alueita, joita käyttäjä voi tarkastella milloin haluaa. [32.]

Laitteet, joissa pyörii käyttöjärjestelmäversio Android 8.0 tai uudempi, saavat ilmoituksen. Käyttöjärjestelmä lisää ilmoituspallon myös sovelluksen kuvakkeeseen. Kun käyttäjä painaa sovelluskuvaketta pitkään, ilmoitus ilmaantuu sovelluksen kuvakkeen ylle. [32.]

Projektiin tehtiin yksinkertainen ilmoitusmekanismi, joka muistuttaa tietyn tuntimäärän jälkeen, että käyttäjän olisi juotava vettä lisää. Koskettamalla tätä ilmoitusta sovellus käynnistyy, jotta vettä voi lisätä. Tällaisen ilmoituksen tekeminen oli suhteellisen yksinkertainen, sen ymmärsi nopeasti, miten se toimii ja mitä muokata toimiakseen oikein.

Sovellus tarkistaa käynnistämisen yhteydessä, ovatko ilmoitukset valittuna vai ei. Jos on valittuna, selvitetään, minkä ajan jälkeen sovelluksen tulee tehdä ilmoitus. Tämä aika löytyy jaetuista asetuksista. Android 8.0 versio ja sitä uudemmat versiot vaativat, että ilmoituskanavaa luodessa kanavalle annetaan uniikki nimi [33]. Ilmoituskanavaa tehtäessä voidaan määrittää esimerkiksi ledin valo sekä käytetäänkö ilmoituksessa värinätoimintoa. Ilmoituksen luomisen kohdalla määritellään tarkemmin, miltä ilmoitus näyttää, tarkoittaen, mitä ilmoituksen tekstit kertovat ja mitä tapahtuu ilmoitusta painettaessa.



Kuva 15. Sovelluksen ilmoitus.

Tämän projektin ilmoitukselle asetettiin ledin valo vihreäksi ja värinätoiminto päälle. Ilmoitukseen lisättiin teksti, joka lisää ilmoitukseen puun lempinimen, jonka käyttäjä on antanut.

Jatkokehityksen tarkoituksena on lisätä torkkutoiminto, muistuttamaan esimerkiksi puolet aikaisemmin ilmoitetusta ajasta, sekä toiminto varmistamaan, ettei yön aikana tule ilmoituksia. Toiveena olisi myös tehdä toinen ilmoitustyyppi, joka on tarkoitettu vesipullon juomiseen. Siten, että sen voi erikseen laittaa päälle, sille määritellään vesipullon koko sekä intervalliaika, jolloin sovellus päivittää ilmoituksen. Joka kerta kun ilmoitus päivitetään, se muistuttaa juoda pullosta ja ilmoituksessa olisi valinnat, joilla käyttäjä voi kertoa sovellukselle, että pullosta juotiin, myöhemmin ja pullo on tyhjä. Kun pullo on tyhjä, sovellus lisää kyseisen vesimäärän puulle automaattisesti sekä kysyy, aloitetaanko tämä toiminto uudestaan.

9 Pienohjelma

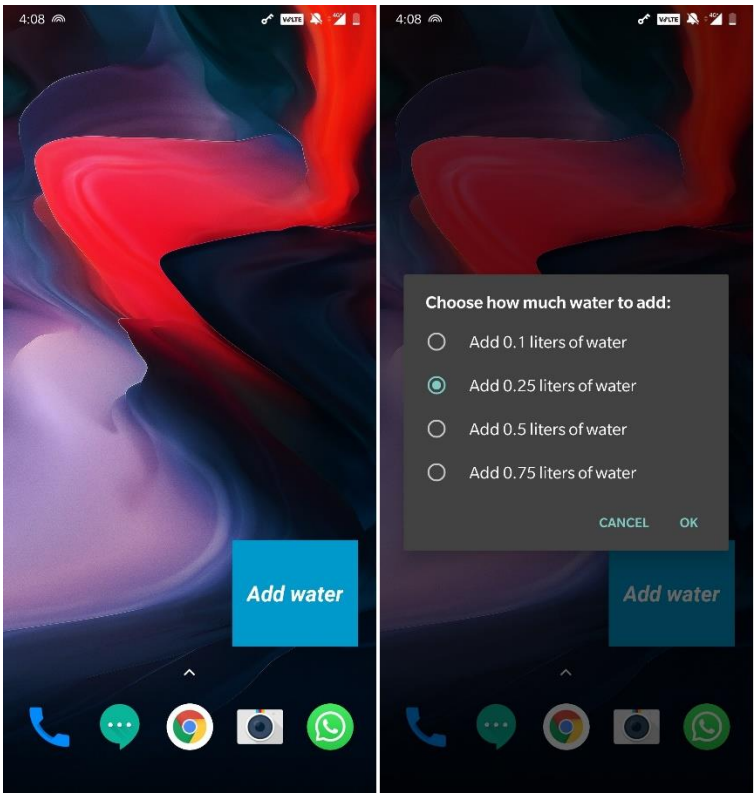
Widget eli pienohjelmat ovat yksi keskeisistä osista kotiruudun mukauttamiseen. Niitä voidaan ajatella nopeina näkyminä sovelluksen tärkeimpiin tietoihin tai sovelluksen toiminto suoraan käyttäjän kotiruudussa. Käyttäjällä on mahdollisuus liikuttaa pienohjelma haluamaansa paneeliin ja kohtaan kotiruudussa. Sekä jos pienohjelma tukee, niin voidaan muokata sen kokoa ja sitä kuinka paljon tietoa näytetään. [34.]

Kun pienohjelmaa suunnitellaan, on hyvä miettiä, minkälaista pienohjelmaa ollaan rakentamassa. Pienohjelmat lajitellaan yleensä neljään eri kategoriaan, ja niistä yleisin on hybridipienohjelma. Hybridipienoisohjelma yleensä keskittyy yhteen kolmesta muusta kategoriasta lainaten muista kategorioista elementtejä. Muut kategoriat ovat informaatiopienohjelma, joka näyttää käyttäjälle informaatiota sovelluksesta. Kokoelmapienohjelma nimensä mukaisesti näyttää kokoelman samantyyppisiä elementtejä, joita voi tarkastella. Hallintapienohjelmaa, voi ajatella sovelluksen kauko-ohjauksena, jolla voi käyttää sovelluksen toimintoja, vaikka sovellus ei ole aktiivinen kyseisellä hetkellä. [34.]

Musiikkipienohjelma on hyvä esimerkki hybridipienohjelmasta, sillä se on pääsääntöisesti hallintapienohjelma, jolla käyttäjä voi pysäyttää tai vaihtaa kappaletta ilman, että sovellus aukaistaan. Se myös antaa käyttäjälle informaatiota mikä kappale on soimassa, eli se yhdistää hallintapienohjelman informaatiopienohjelman elementtien kanssa. [34.]

Android Studiossa on pohja, joka voidaan lisätä projektiin. Pohja tekee automaattisesti pienohjelman layout xml-tiedoston ja koodin, joka antaa pohjan pienohjelman lisäämiseen ja päivittämiseen. Pohjaa lisätessä voidaan määritellä pienohjelman koko, voiko kokoa muokata kotiruudussa ja valita, lisätäänkö projektiin aktiviteetti, jossa käyttäjä voi määritellä, mitä informaatiota pienohjelma näyttää.

Alla esitettävässä kuvassa nähdään miltä pienohjelma näyttää laitteen kotiruudulla ja mitä tapahtuu kun sitä painetaan.



Kuva 16. Kotiruutuun asetettu pienohjelma ja pienohjelman aktiviteetti.

Tämän sovelluksen pienohjelman on tarkoitus olla informaatio- ja hallintapienohjelman hybridi. Mutta se on tässä mekaniikkojen kehitysvaiheessa vain hallintapienohjelmasta tehty kompromissiversio. Pienohjelmaa painamalla se käynnistää sovelluksesta aktiviteetin, joka näyttää vain valikon läpinäkyvällä pohjalla, jossa on käyttäjän asettamat vesimäärät. Valikosta voi valita, minkä vesimäärän käyttäjä lisää puulle, minkä jälkeen sovellus sulkee itsensä.

10 Sovelluksen tulos ja jatkokehitys

Lopputuloksena on sovellus, joka sisältää kaikki tarvittavat mekaniikat toimiakseen kunnolla. Sovelluksella voidaan seurata käyttäjän juoman veden määrää, ja se tallentaa päivän aikana saavutetun vesimäärän. Ainoa ominaisuus, mitä ei saatu kehitettyä kunnolla toimivaksi, on kotiruutuun lisättävä pienohjelma. Se on ensimmäinen asia, mikä kehitetään loppuun, ja todennäköisesti tehdään useampi variaatio.

Kehityksen aikana saatiin paljon ideoita ominaisuuksiin, joita kehittää tulevaisuudessa ja miten lisätä käytettävyyttä. Seuraavana olisi kuitenkin tarkoitus keskittyä lisäämään grafiikkaa pienillä animaatioilla ja optimoimaan koodia. Tämä tulee olemaan paljolti käyttöliittymän käyttökokemuksen kehittämistä ja parantamista. Esimerkiksi historia-aktiviteetin näkymä muotoillaan paljon enemmän käyttäjäystävällisemmäksi. Päänäkymään tehtäisiin palkki juodun veden määrän seuraamiseksi ja puulle, jolla ilmaistaan, paljonko vettä puu tarvitsee vielä vettä ennen kuin se kasvaa.

Jotta sovelluksella olisi mahdollisimman laaja käyttäjäkunta, on tarkoituksena lisätä kaikki käyttöliittymän String-muuttujat resursseihin. Tällä tavalla voidaan helposti kääntää sovellus usealle kielelle. Toiveena olisi myös automatisoida tietynlaisten käyttöliittymäasetusten valinnat. Tällä tarkoitetaan sitä, että kun sovellus asennetaan ja käynnistetään, se valitsee kielekseen sen, mitä käyttöjärjestelmä käyttää, jos sovellus on käännetty vastaavalle kielelle ja sovellus käyttäisi myös järjestelmän valitsemaa kirjaimen fontin kokoa.

Lähteet

(1) Callaham John. The history of Android OS: its name, origin and more. 2019; Julkaistu: https://www.androidauthority.com/history-android-os-name-789433/. Haettu 11.01.2020.

(2) Android is for everyone. Julkaistu: https://www.android.com/everyone/. Haettu 9.01.2020.

(3) Set up for Android Development. Julkaistu: https://source.android.com/setup?hl=fi. Haettu 9.01.2020.

(4) Android 10 Release Notes. Julkaistu: https://source.android.com/setup/start/android-10-release?hl=fi. Haettu 9.01.2020.

(5) Android Open Source Project. Julkaistu: https://source.android.com/?hl=fi. Haettu 9.01.2020.

(6) Android Architecture. Julkaistu: https://source.android.com/devices/architecture?hl=fi. Haettu 9.01.2020.

(7) HAL Types. Julkaistu: https://source.android.com/devices/architecture/hal-types?hl=fi. Haettu 10.01.2020.

(8) Using Binder IPC. Julkaistu: https://source.android.com/devices/architecture/hidl/binder-ipc?hl=fi. Haettu 10.01.2020.

(9) Android System Architecture. Julkaistu: https://android.tutorialhorizon.com/android-system-architecture/. Haettu 10.01.2020.

(10) Platform Architecture. Julkaistu: https://developer.android.com/guide/platform?hl=fi. Haettu 10.01.2020.

(11) Activity. Julkaistu: https://developer.android.com/reference/android/app/Activity?hl=fi. Haettu 15.01.2020.

(12) Soni N. Android Activity Explained. 2017; Julkaistu: https://medium.com/@n.nikhil.ns65/android-activity-explained-2b651a8d2c79. Haettu 15.01.2020.

(13) App Manifest Overview. Julkaistu: https://developer.android.com/guide/topics/manifest/manifest-intro?hl=fi. Haettu 10.01.2020.

(14) The AndroidManifest.xml File. Julkaistu: https://stuff.mit.edu/afs/sipb/project/android/docs/guide/topics/manifest/manifest-intro.html. Haettu 12.01.2020.

(15) What Is an APK File and What Does It Do? Julkaistu: https://www.makeuseof.com/tag/what-is-apk-file/. Haettu 12.01.2020.

(16) What is Android SDK? - Definition from Techopedia. Julkaistu: https://www.techopedia.com/definition/4220/android-sdk. Haettu 12.01.2020.

(17) Android Studio: An IDE built for Android. 2013 May 15; Julkaistu: https://android-developers.googleblog.com/2013/05/android-studio-ide-built-for-android.html. Haettu 11.01.2020.

(18) Mew K. Mastering Android Studio 3. 2017; Haettu 11.01.2020.

(19) Android Platform. Julkaistu: https://developer.android.com/about?hl=fi. Haettu 11.01.2020.

(20) Meet Android Studio. Julkaistu: https://developer.android.com/studio/intro?hl=fi. Haettu 11.01.2020.

(21) Configure your build. Julkaistu: https://developer.android.com/studio/build?hl=fi. Haettu 15.01.2020.

(22) zipalign. Julkaistu: https://developer.android.com/studio/command-line/zipalign?hl=fi. Haettu 15.01.2020.

(23) What is Gradle. Julkaistu: https://docs.gradle.org/current/userguide/what_is_gradle.html#what_is_gradle. Haettu 15.01.2020.

(24) Pillai AS. Shared Preferences in Android explained in detail. 2017; Julkaistu: http://www.gadgetsaint.com/android/android-shared-preferences-detail/. Haettu 18.01.2020.

(25) Save key-value data. Julkaistu: https://developer.android.com/training/data-storage/shared-preferences?hl=fi. Haettu 18.01.2020.

(26) Android fundamentals 07.1: AsyncTask. Julkaistu: https://codelabs.developers.google.com/codelabs/android-training-create-asynctask/index.html?index=..%2F..%2Fandroid-training#0. Haettu 18.01.2020.

(27) AsyncTask. Julkaistu: https://developer.android.com/reference/android/os/AsyncTask?hl=fi. Haettu 17.01.2020.

(28) ViewSwitcher. Julkaistu: https://developer.android.com/reference/android/widget/ViewSwitcher. Haettu 18.01.2020.

(29) Android fundamentals 04.5: RecyclerView. Julkaistu: https://codelabs.developers.google.com/codelabs/android-training-create-recycler-view/index.html?index=..%2F..android-training#0. Haettu 18.01.2020.

(30) Settings. Julkaistu: https://developer.android.com/guide/topics/ui/settings?hl=fi. Haettu 20.01.2020.

(31) Organize your settings. Julkaistu: https://developer.android.com/guide/topics/ui/settings/organize-your-settings?hl=fi. Haettu 20.01.2020.

(32) Android fundamentals 08.1: Notifications. Julkaistu: https://codelabs.developers.google.com/codelabs/android-training-notifications/index.html?index=..%2F..%2Fandroid-training#0. Haettu 20.01.2020.

(33) Create a Notification. Julkaistu: https://developer.android.com/training/notify-user/build-notification?hl=fi. Haettu 22.01.2020.

(34) App Widgets Overview. Julkaistu: https://developer.android.com/guide/topics/appwidgets/overview?hl=fi. Haettu 22.01.2020.

Kuvat