Antti Joutsenniemi

Android-sovelluksen valmistusprosessi ideasta Play-kauppaan

Opinnäytetyö

Liiketalouden ammattikorkeakoulututkinto

Tietojenkäsittelyn koulutus

2023

Kaakkois-Suomen ammattikorkeakoulu

Tutkintonimike	Tradenomi (AMK)
Tekijä/Tekijät	Antti Joutsenniemi
Työn nimi	Android-sovelluksen valmistusprosessi ideasta Play-kauppaan
Toimeksiantaja	-
Vuosi	2023
Sivut	32
Työn ohjaaja(t)	Janne Turunen

TIIVISTELMÄ

Opinnäytetyön tavoitteena on selvittää Android-sovelluksen valmistusprosessin eri vaiheet sekä toteuttaa sovelluskauppaan julkaistava sovellus. Opinnäytetyö on muodoltaan raportti, eli siinä käydään läpi itse tekemisen vaiheita, sekä sitä tukevaa ja taustalla olevaa teoriaa.

Lähtökohtana työssä on idea riimittelysovelluksesta, jota voisi käyttää esimerkiksi rap-muusikot, laulunsanoittajat tai runon kirjoittajat. Käyttäjä pystyisi kirjoittaa sovelluksella pitkiäkin tekstejä, ja etsiä mille vain sanalle hyvin sointuvia pareja. Näitä tekstejä käyttäjä pystyisi sitten tallentamaan ja palata niiden ääreen myöhemmin.

Työ alkaa suunnitteluvaiheesta, jossa etsitään sovellukselle sen tarvitsema data. Tämän jälkeen suunnitellaan sovelluksen vaatimat visuaaliset elementit, kuten suunnitelma sovelluksen rakenteesta, värit, ilme sekä logo.

Suunnitteluvaiheen jälkeen siirrytään ohjelmointivaiheeseen, jossa ensimmäiseksi luodaan komponenttirakenne. Komponentteihin lähdetään sitten luomaan toiminnallista ja käyttäjille näkyvää koodia, joista molempia visualisoidaan kuvin.

Tarkemmassa tarkastelussa ohjelmointivaiheessa on kaksi funktiota. Ensimmäinen funktio on riimittelyfunktio, joka etsii aiemmin etsitystä datasta sointuvia pareja käyttäjän kirjoittaman tekstin sanojen perusteella. Toinen funktioista on automaattisen tallennuksen funktio, jossa käyttäjän kirjoittamat tekstit tallennettaan automaattisesti tiedostolle joka kerta, kun muutoksia tapahtuu.

Seuraavaksi käydään läpi sovelluksen koontivaihe, jossa luodaan ensimmäiseksi paikallinen testiversio. Tämän jälkeen kootaan varsinainen, Google Play-kauppaan julkaistava versio.

Viimeisenä käydään läpi kootun version julkaiseminen Play-kauppaan. Julkaisemiseen liittyy myös muita vaiheita, kuten mainoskuvien ja tietosuojakäytännön luomista, joita käydään myös tässä vaiheessa läpi. Lopuksi näytetään valmis tuote, sekä pohditaan projektin herättämiä ajatuksia.

Asiasanat: mobiilisovellus, ohjelmointi, graafinen suunnittelu

South-Eastern Finland University of Applied Sciences

Degree title	Bachelor of Business Administration
Author	Antti Joutsenniemi
Thesis title	Bachelor's / Master's thesis title in English
Commissioned by	-
Time	2023
Pages	32
Supervisor	Janne Turunen

ABSTRACT

The objective of the thesis was to explain the different stages of an Android application development process and create an application to be published in the Google Play Store. The thesis was in the form of a report, i.e. it went through the steps of making the app itself, as well as the supporting and underlying theory.

The starting point of the thesis was an idea of a rhyming app that could be used, for example, by rap musicians, songwriters or poem writers. The user would be able to write long texts, and search for rhyming pairs for any word. The user could then save these products and return to them later.

The work started with the planning phase, which involved finding the data the application needed. The visual elements required for the application were then designed, such as a plan for the structure of the application, the colours, the look, and the logo.

After the planning phase was the programming phase, where the first step was to create the component structure. Inside the components were then created functional and user-visible code, both of which were visualised with images.

In the programming phase the main point was to look closely into two functions. The first one was a rhyming function that searched the previously sought up data for word matches based on the words of the text entered by the user. The second function was an autosave function, which automatically saved the text entered by the user to a file each time a change was made.

The next step was to go through the building phase of the application, where a local test version was created first. Then the actual version that was published to the Google Play store was built.

The last step was the process of publishing the assembled version to the Play Store. There were also other steps involved in the publishing process, such as the creation of advertising images and the privacy policy, which were also discussed here. Finally, the finished product was shown and the ideas generated by the project were discussed.

Keywords: mobile-app, programming, graphic design

SISÄLLYS

1 JOHDANTO

Opinnäytetyön aiheena on mobiilisovellustuotanto. Tavoitteena on luoda toimiva android-sovellus Googlen Play-sovelluskauppaan. Rajaan aihetta sen verran, että keskityn enemmän tuotantoprosessin eri vaiheisiin, kuin itse koodaamiseen. Kiteytettynä kehittämis-/tutkimusongelmana on selvittää itsenäinen mobiilisovelluksen tuotantoprosessi ideasta Play-kauppaan.
Työn tulokset ovat hyödyllisiä itselleni, sillä voin jatkossa tuottaa erilaisia sovelluksia samalla kaavalla. Myös muut, joilla on samanlaisia tavoitteita, voivat etsiä työstä apua omiin projekteihinsa. Tämä opinnäytetyön raportti on koottu niin, että kuvaan lineaarisesti etenevästi sovellustuotannon eri vaiheet alusta loppuun. Rakenteeltaan raportti on jaettu prosessin eri vaiheisiin, jotka ovat selkeästi omia kokonaisuuksiaan ja sisältävät toisistaan poikkeavia työtehtäviä. Joihinkin työvaiheisiin ja käsitteisiin liittyy tarvittavaa pohjatietoa ja teoriaa, joita käyn läpi lähdeviitteineen sitä mukaa kun niitä tekstissä ilmaantuu.

Olen aina halunnut tehdä mobiilisovelluksen, jonka kuka vain voisi ladata sovelluskaupasta omalle puhelimelle. Sovellusohjelmointi 3 -kurssilla minulle selvisi, että sellainen on mahdollista luoda JavaScriptin kirjastoilla, joita olen tutkinnon aikana opiskellut paljon. Riittävä pohjatieto oli siis tutkinnon kursseilla hankittu, joten päätin lähteä selvittämään kokonaisvaltaista tuotantoprosessia ideasta valmiiksi Play-sovelluskauppaohjelmaksi. Ideana minulla oli riimittelysovellus, jota voisi käyttää rap-muusikot, laulunsanoittajat, runoilijat jne.

Sovellukseen voisi kirjoittaa esimerkiksi runoa ja helppokäyttöisen ja simppelin käyttöliittymän kautta etsiä mistä vain sanasta rimmaavia pareja, tallentaa tuotoksia ja palata niiden pariin myöhemmin.

Projekti kiteyttää hyvin yhteen tutkinnon aiheita, sillä pääsen siinä koodauksen lisäksi suunnittelemaan toimivaa ja visuaalisesti miellyttävää käyttöliittymää, sekä luomaan logoja ja esittelykuvia Adoben ohjelmilla. Valmiin sovelluksen koonti, allekirjoittaminen ja lataaminen Play-kauppaan on myös oma prosessinsa, joten käyn sitä laajasti läpi opinnäytetyössä. Aihe on myös oman kehitykseni kannalta mieluisa, sillä olen miettinyt vaihtoehtoa tuottaa itsenäisesti sovelluksia sovelluskauppaan ja myöhemmin ansaita niillä.

2 SUUNNITTELUVAIHE

Suunnitteluvaiheessa lähden liikkeelle ideasta, joka on riimittelysovellus. Ohjelman ominaisuudet, sisältö ja visuaaliset elementit on helppo suunnitella ja konkreettisesti lyödä lukkoon ennen koodaamisen aloittamista, eikä ne tule todennäköisesti elämään tuotantoprosessin aikana. Näistä syistä sovelluksen tuotantoon voi käyttää lineaarisesti etenevää vesiputousmallia. Ensimmäisen kerran vesiputousmalli on kuvattu Walter Roycen artikkelissa vuodelta 1970. Siinä sovelluksen rakentamisprosessi etenee vaiheittain ylhäältä alaspäin kuin vesiputous, alkaen esimerkiksi suunnitteluvaiheesta koodaukseen ja sitten testaamiseen. (Rakenne kuvassa 1) Ideana on, että kun yksi vaihe on saatu päätökseen, ei aikaisempiin vaiheisiin enää kajota. (Petersen ym. 2009, 1.)

Kuva 1: Esimerkki vesiputousmallin mukaisesta etenemistavasta (mukaillen Petersen ym. 2009, 1)

Menetelmä on saanut kritiikkiä sen huonoista valmiuksista vastata muutokseen, minkä vuoksi nykyään suurissa ja muuttuvissa projekteissa on usein käytetty ketterän kehityksen menetelmiä. Agile Alliance on ketterän kehityksen menetelmiä ajava organisaatio, ja sen manifestossa (2022) asia tiivistetään näin: ”Responding to change over following a plan”, eli muutokseen vastaaminen on menetelmässä tärkeämpää kuin suunnitelman noudattaminen. Projek-

tini koon ja selkeyden vuoksi se on kuitenkin helpompi tuottaa vesiputousmallin keinoin. Ketterässä kehityksessä tuodaan myös usein asiakkaat mukaan arvioimaan ja testaamaan sovellusta tai sen sisältöä välivaiheissa. Minulla ei ole asiakasta, joten siitäkin syystä katsoin vesiputousmallin selkeämmäksi vaihtoehdoksi.

Vesiputousmallin mukaisesti oli siis ensin luotava kattava suunnitelma, jotta ohjelmointivaihe olisi mahdollisimman mutkaton. Tässä vaiheessa visio sovelluksen rakenteesta ja arkkitehtuurista olivat hämärän peitossa, joten sitä selkeyttääkseni lähdin kirjaamaan paperille toimintoja, joita halusin sovelluksessa olevan. Halusin esimerkiksi, että käyttäjä voi kirjoittaa runoa, ja hänen kirjoittamat sanat ilmestyvät automaattisesti ruudulle aina, kun hän painaa välilyöntiä. Halusin myös mahdollisuuden tallentaa runoja, sekä automaattitallennuksen, jos käyttäjä esimerkiksi sulki sovelluksen kesken käytön. Tallennettuja runoja käyttäjä voisi tietenkin jatkaa myöhemmin, tai kopioida niiden koko sisällön puhelimensa leikepöydälle. Jokaiseen tallennettuun runoon tulisi runon nimi, jonka käyttäjä itse antaisi, itse runon sanat, sekä muokkauspäivämäärä. Vaihtoehtoiseksi haasteeksi laitoin kielen vaihtamisen. Viimeisenä kirjasin ylös toisen vaihtoehtoisen haasteen: mahdollisuus vaihtaa sovelluksen teemaa tummasta vaaleaan ja päinvastoin. Sovelluksen tulevat ominaisuudet olivat nyt kirjattuna ylös, joten visio alkoi selkeytyä.

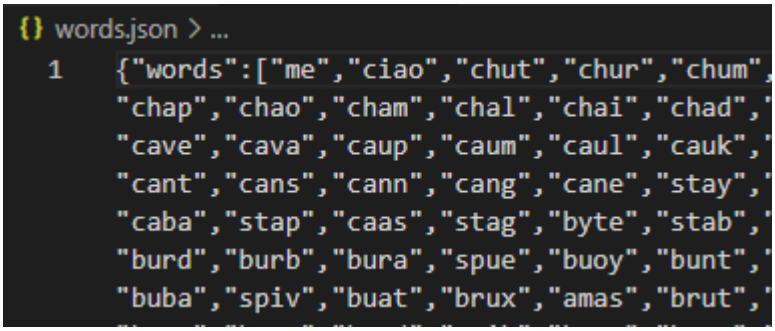
2.1 Sovelluksen tarvitsema data

Tiesin jo tässä vaiheessa, että sovellus tulisi tarvitsemaan ulkopuolista dataa jonkinlaisen sanalistan muodossa, joten kaksi kieltä tarkoittaisi myös kahta sanalistaa. Seuraavaksi lähdin etsimään sopivia tietokantoja, joista etsisin rimmaavia sanoja. Aloitin englanninkielisestä sanalistasta. Etsin alkuun JSON-muotoisia sanalistoja, sillä ne ovat JavaScript-koodausympäristöissä hyvin tuettuja. JSON (JavaScript Object Notation) on tekstipohjainen tiedonvälitysmuoto, jota käytetään usein eri ohjelmointikielten välisessä tiedonsiirrossa. Se on yleensä hyvin luettavaa ja helposti käsiteltävissä, ja se perustuu JavaScript-ohjelmointikieleen. JSON-muoto sisältää muuttujien ja niiden arvojen

joukon, joka voidaan tallentaa tekstitiedostoon tai lähettää verkon yli muiden ohjelmien käytettäväksi. (MDN 2022a.) Esimerkiksi kuvassa 2 JSON-muoto kengästä, joka voisi sijaita kenkäkaupan simppelissä inventaariotietokannassa:

Kuva 2: JSON-esimerkki (mukaillen MDN, 2022.)

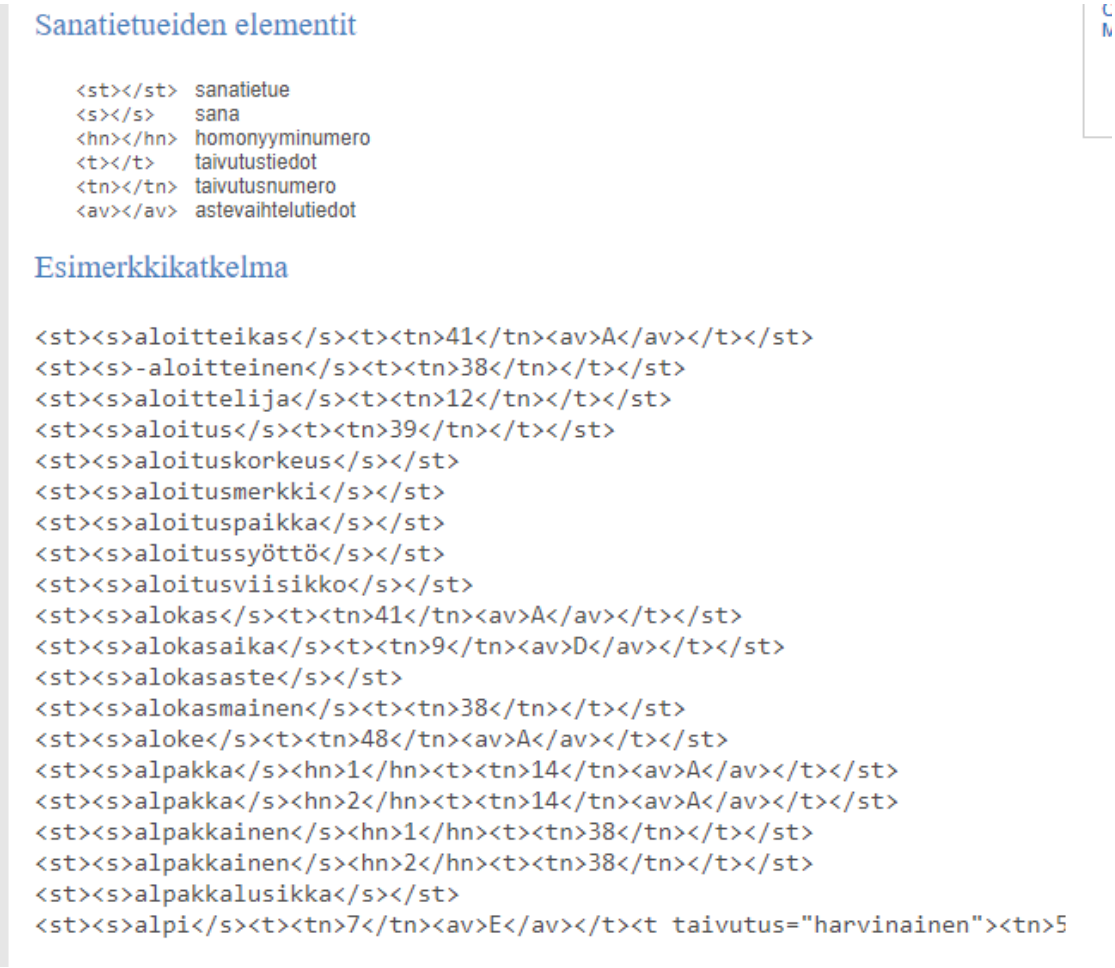
Erilaisin hakusanoin Googlesta tovin etsittyäni erilaisia JSON-muotoisia sanalistoja erilaisin rakentein löytyikin odotettua enemmän, oli mm. eri laajuisia sekä vain yleisimpiä sanoja sisältäviä olevia sanalistoja. Lopulta löysin word-list-json node.js-paketin, joka sisälsi laajan sanarepertuaarin ja oli helposti käsiteltävässä muodossa (sanat olivat kasvavassa järjestyksessä kirjaimien määrän mukaan, kuvassa 3 sanalistan alkupäästä osa.)



Kuva 3: word-list-json JSON-tiedoston alkupää.

Kuten kuvasta näkyy, sanalista sisälsi myös hyvin harvinaisia sanoja, sekä kaiken kaikkiaan tiedoston pituus oli niin suuri, ettei koodieditori pystynyt laskea sen yksittäisiä rivejä, tai värikoodata sen eri elementtejä kuten kuvan 2 kenkäesimerkissä. Harvinaiset sanat eivät haitanneet minua, sillä halusinkin, että käyttäjä saisi sovelluksen kautta tulokseksi sanoja, joita hän ei itse olisi tullut ajatelleeksi. Toisaalta myös täysin omituiset sanat eivät yleensä soinnu hyvin yhteen yleisien sanojen kanssa, joita käyttäjä todennäköisesti kirjoittaa, eivätkä siksi edes näkyisi lopullisessa sovellusnäkymässä. Nyt minulla oli siis

englanninkielinen sanalista, mutta tarvitsin sen rinnalle myös suomenkielisen version, jotta sovelluksen kielen pystyisi vaihtamaan. Lähdin taas etsimään netistä, mutta hakuni päättyi tällä kertaa vain yhteen vaihtoehtoon, nimittäin Kotimaisten kielten keskuksen nykysuomen sanalistaan, (2007.) joka oli muodossa XML. XML-muotoa (Extensible Markup Language) voi JSON:in tapaan käyttää datan säilömiseen, jonka syntaksiin kuuluu nuolimerkein esitettävien ”tagien” käyttöä. (MDN 2022b.) Kuvassa 4 Kotimaisten kielten keskuksen oma esimerkki tagien käytöstä heidän toimittamassa sanalistassaan:



Sanatietueiden elementit

<st></st>	sanatietue
<s></s>	sana
<hn></hn>	homonyyminumero
<t></t>	taivutustiedot
<tn></tn>	taivutusnumero
<av></av>	astevaihtelutiedot

Esimerkkikatkelma

```
<st><s>aloitteikas</s><t><tn>41</tn><av>A</av></t></st>
<st><s>-aloitteinen</s><t><tn>38</tn></t></st>
<st><s>aloittelija</s><t><tn>12</tn></t></st>
<st><s>aloitus</s><t><tn>39</tn></t></st>
<st><s>aloituskorkeus</s></st>
<st><s>aloitusmerkki</s></st>
<st><s>aloituspaikka</s></st>
<st><s>aloitussyöttö</s></st>
<st><s>aloitusviisikko</s></st>
<st><s>alokas</s><t><tn>41</tn><av>A</av></t></st>
<st><s>alokasaika</s><t><tn>9</tn><av>D</av></t></st>
<st><s>alokasaste</s></st>
<st><s>alokasmainen</s><t><tn>38</tn></t></st>
<st><s>aloke</s><t><tn>48</tn><av>A</av></t></st>
<st><s>alpakka</s><hn>1</hn><t><tn>14</tn><av>A</av></t></st>
<st><s>alpakka</s><hn>2</hn><t><tn>14</tn><av>A</av></t></st>
<st><s>alpakkainen</s><hn>1</hn><t><tn>38</tn></t></st>
<st><s>alpakkainen</s><hn>2</hn><t><tn>38</tn></t></st>
<st><s>alpakkalusikka</s></st>
<st><s>alpi</s><t><tn>7</tn><av>E</av></t><t taivutus="harvinainen"><tn>5
```

Kuva 4: Sanalistan ohjeet (Kotimaisten kielten keskus, 2007.)

Halusin kuitenkin myös suomenkielisen listan JSON-muotoon, jotta samaa koodia voisi myöhemmin suoraan hyödyntää molemmissa versioissa. Kuten ohjelmoinnissa yleensä, asian voi ratkaista monella tapaa. Totesin parhaaksi tavaksi käyttää netistä löytyvää muuntajaa, joka muunsikin tiedoston suoraan haluamaani muotoon ilman virheitä.

2.2 Visuaaliset elementit

Sovelluksen tekeminen alusta loppuun on paljon muutakin kuin koodaamista. Vaikka koodi toimisi täydellisesti, ei sen hyötyjä pystytä välittämään käyttäjälle, jos sovelluksen visuaaliset elementit kuten värit, ilme ja käyttöliittymä eivät kohtaa itse käyttäjää. Siksi lähdinkin seuraavaksi suunnittelemaan kaikkea käyttäjälle näkyvää visuaalista sisältöä, joista ensimmäisenä päätin tehdä suunnitelman käyttöliittymästä. Visiona minulla oli selkeä, simppeli ja helppokäyttöinen. Hain lopputuloksesta modernia ja minimalistista ilmettä, jolloin interaktiivisia painikkeita ja toimintoja tulisi mahdollisimman vähän. Tällaisessa asetelmassa on hyvä tukeutua käyttöliittymäsuunnittelun toimiviksi todettuihin sääntöihin, kuten Fittsin lakiin. Paul Fitts oli psykologi, joka teki vuonna 1954 tutkimuksia ihmisen motoriikan parissa. Hän huomasi, että aika joka esim. sormella kuluu kohteeseen liikkuessa, riippuu tietenkin matkasta, mutta on myös riippuvainen kohteen koosta. Eli mitä pidempi matka ja pienempi kohde, sitä enemmän aikaa sinne liikkumiseen kuluu. (Interaction Design Foundation 2022.)

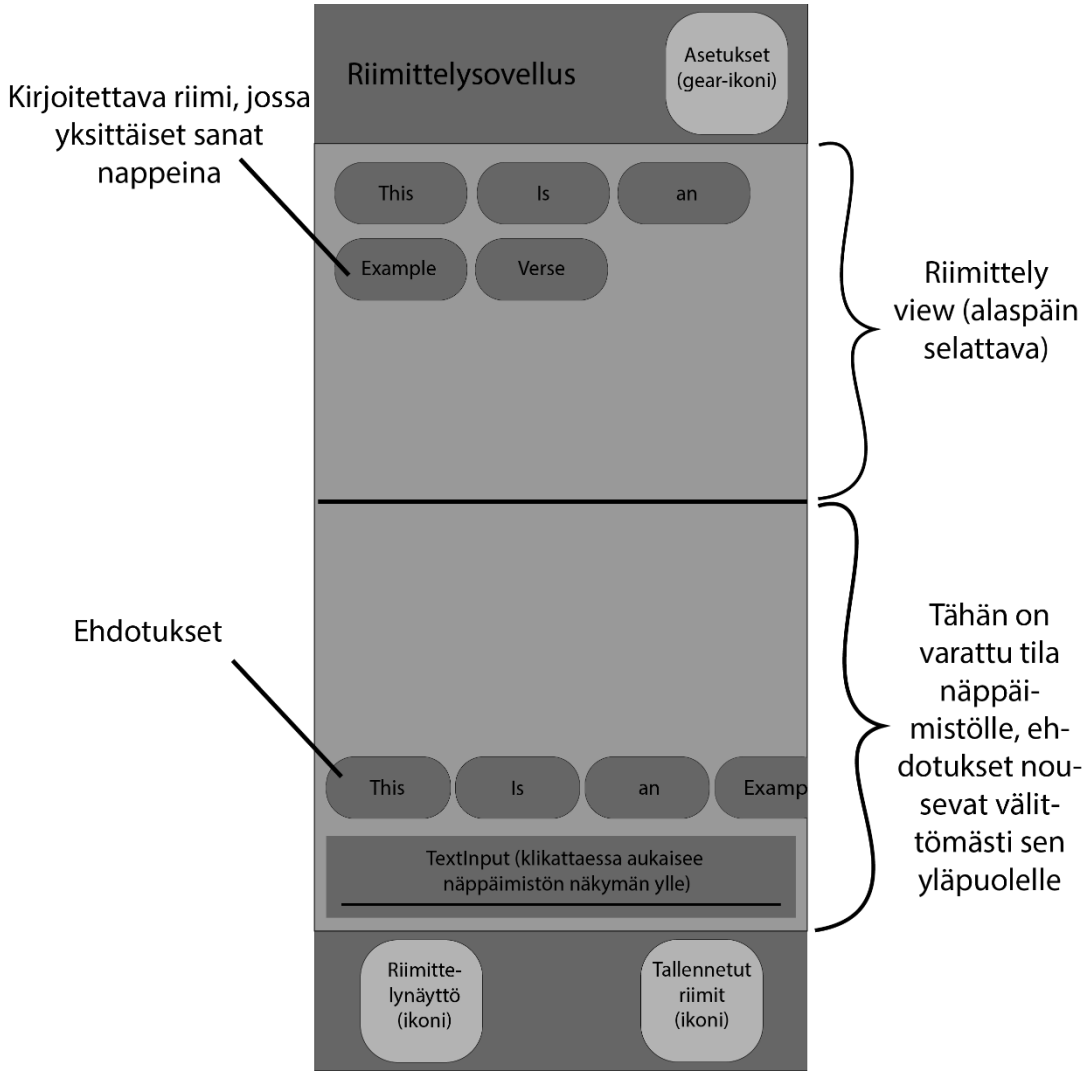
Miten tätä lakia voi sitten hyödyntää käyttöliittymäsuunnittelussa? Merkittäviä toimintoja sisältävät elementit tai painikkeet voisi esimerkiksi sijoittaa näytöllä lähelle peukaloa tai etusormea, sekä tehdä niistä suuria. Suuret ja selkeät painikkeet ja toiminnot eivät turhauta käyttäjää ja parantavat produktiivisuutta. Lisäksi painikkeiden kokoeroilla ja sijainnilla voi viestittää käyttäjälle niiden käyttötarkoitusta. Halusin tämän mukaan sijoittaa käyttäjälle ehdotettavat sanat lähelle näppäimistöä, esimerkiksi suoraan sen yläpuolelle, ja itse kirjoitettavan runon taas sen yläpuolelle, missä on eniten tilaa jopa pitkälle runolle.

Varsinkin mobiilisovellukset tarvitsevat myös usein navigaatioita, sillä kaikki tarvittavat ominaisuudet eivät mahdu pienille ruuduille. Navigaatioiden avuilla liikutaan ruuduista ja näkymista toisiin, ja niiden on hyvä olla itsestään selviä, ettei käyttäjälle tarvitse erikseen ohjata mistä löytyy mitäkin. Halusin sovellukseeni kaksi pääasiallista navigaatiota, ylös toiseen yläkulmaan ikonin, jonka

kautta pääsisi asetusvalikkoon, sekä alas navigaatiopalkin, jonka kautta vaihdettaisiin näkymiä. Aiempaa ajatusta minimalistisesta ilmeestä jatkaen halusin alapalkkiin vain kaksi eri näkymää, toisessa tehtäisiin itse runon kirjoittaminen ja toisessa näkymässä sijaitsisi tallennetut runot.

2.2.1 Rautalankamalli

Minulla oli nyt hyvä kuva päässäni käyttöliittymän rakenteesta, joten oli aika luoda rautalankamalli. Rautalankamallilla tarkoitetaan visuaalista asetelmaa/suunnitelmaa, joka näyttää selkeästi missä lopullisen sovelluksen toiminnot ja sisällöt sijaitsevat, sekä miten ne asettuu toisiinsa nähden. (Fisher 2022) Aloitin siis uuden tiedoston Adobe Illustratoriin, ja asettelin siihen suorakaiteilla kaikki haluamani elementit, sekä linkitin aiemmin tekemästäni ominaisuuslistasta ominaisuuksia kuhunkin elementtiin. Halusin, että jokaiselle listasta löytyvälle elementille löytyy paikkansa sovelluksesta, ettei koodaamisvaiheessa tulisi mutkia matkaan, kun yhdelle tai useammalle ominaisuudelle ei löytyisikään järkevää kohtaa. Rautalankamallia katsoessani tajusin, ettei kaikki elementit mahtuisi mitenkään kahteen näkymään. Päätin, että teen kumpaankin näkymään alaspäin selattavan kentän, jota tarvittaisiin vasta kun käyttäjä kirjoittaisi niin pitkän runon, että se vaatisi tilaa näytön ulkopuolelta. Tämä johti toiseen ongelmaan. Jos käyttäjä kirjoittaisi pitkän runon ja haluaisi nähdä samalla myös rimmaavien sanojen listan, ne eivät mahtuisi mitenkään samalle ruudulle. Ongelman ratkaistakseni kirjasin ylös, että ehdotetuista sanoista pitäisi tehdä sivusuuntaan selattava rivi, joka veisi kiinnitetysti oman tilansa näppäimistön yläpuolelta. Nämä kaikki asiat huomioon ottaen päädyin lopulta kuvassa 5 näkyvään rautalankamalliin.



Kuva 5. Rautalankamalli sovelluksen rakenteesta.

2.2.2 Dark mode

Seuraavaksi lähdin miettimään sovelluksen värimaailmaa. Käyttöliittymäsuunnittelun maailmassa on viime aikoina yleistynyt tummat teemat, näistä käytetään yleensä nimeä ”dark mode”. Päätin seurata trendiä ja luoda myös tähän projektiin tumman väripaletin. Dark mode ei ole vain kosmeettinen designtrendi, vaan sen suosimiseen tavallisen vaalean teeman sijaan löytyy myös tieteellistä perustaa.

Floridalaisessa yliopistossa AR-lasein tehdyssä tutkimuksessa (Kangsoo ym. 2019) testattiin eroja vaalean ja tumman teeman välillä. 19 tutkimukseen osallistunutta testihenkilöä mm. lukivat tekstejä ja tulkitsivat symboleja erilaisilla pohjilla ja kontrastieroilla. Tutkimuksen tulokset puhuivat selkeästi tumman

teeman puolesta. Testihenkilöiden näkö oli dark modessa tarkempi optisien näkötestien perusteella, sekä he kokivat silmien väsyvän vähemmän sitä käytettäessä.

Dark mode on siis käyttäjälle mieluisa, mutta se voi olla sitä myös laitteiden akuille. Vuoden 2017 jälkeen yhä useammassa mobiililaitteessa on ns. OLED näyttö. (Organic Light-Emitting Diode) Vanhemmissa LCD-näytöissä (Liquid Crystal Display) on aina päällä oleva takavalo, kun taas OLED-näytöissä jokainen yksittäinen pikseli saa oman valaistuksensa. Käytännössä tämä tarkoittaa sitä, ettei tummempiin pikseleihin tarvitse käyttää yhtä paljoa energiaa kuin vaaleisiin. Energian säästön määrä on kuitenkin suuresti riippuvainen näytön kirkkaudesta. Monet käyttävät puhelimensa kirkkausasetusta automaattitilassa, joka on huonevalaistuksessa suuren osan ajasta noin 30-40%. Näillä tasoilla dark mode säästää akkua vain 3-9%, kun taas täydessä 100 % kirkkaudessa se voi säästää akkua jopa 39-47%. (Hu ym. 2021.)

2.2.3 Väripaletti

Päädyin siis valitsemaan tumman teeman, joten oli aika luoda miellyttävä väripaletti. Verkko on täynnä tähän tarkoitettuja väripalettiohjelmia. Käytin sellaista, joka generoi satunnaisia värejä, ja pistin sen asetuksiin harmaan ja tummansinisen sävyjä. Satunnaista generaatiota hetken pyöritettyäni päädyin 4 tummaan sävyyn, jotka menivät portaittain tummemmasta vaaleampaan. Kirjasin sävyjen heksakoodit ylös tekstitiedostoon, mistä ne olisi myöhemmin helppo kopioida suoraan koodiin.

Mikä sitten on heksakoodi? Kaikki digitaaliset tuotteet perustuvat kolmeen väriin, punaiseen, vihreään ja siniseen. Tätä värijärjestelmää kutsutaan lyhenteellä RGB, ja pohjimmiltaan tietokoneen näyttökin näyttää vain näytä värejä, joista muut värit ovat vain yhdistelmiä. Heksakoodi-järjestelmä ei ole tästä poikkeus, vaan sen jokainen 2 merkin pari kuvaa joko sinistä, vihreää tai punaista. Esimerkiksi heksakoodi #0000FF on sininen, sillä sen ensimmäiset kaksi arvoa 00 ja 00 tarkoittavat, että punaista ja vihreää on 0%, ja sinistä on 100% eli FF. Kun kaikkia värejä laitetaan täydet 100% eli #FFFFFF tulee valkoista, ja kun kaikkea laitetaan 0% eli #000000 tulee mustaa. (iStock 2022.)

Halusin tummien värien kumppaniksi jonkin kirkkaan värin, ja mielessäni oli limenvihreä, joka toisi raikkautta sovellukseen. Etsin kuvahausta vihreää sisältävää taidetta, ja löysin miellyttävän maalauksen kasvista tummassa ympäristössä. Avasin kuvan Photoshopissa, poimin sen vihreän värin, ja kirjasin sen ylös. Kuvassa 6 lopullinen väripalettini.

Kuva 6. Sovelluksen väripaletti heksakoodeineen.

2.2.4 Sovelluksen logo

Yksi isoimmin näkyvillä olevista elementeistä mobiilisovelluksissa on sovelluksen logo tai ikoni. Hyvin suunnitellulla logolla voi tehdä mahdollisiin lataajiin laadukkaan vaikutuksen, kun taas huono logo voi karkottaa potentiaalisia lataajia, vaikka itse sovellus olisi laadukas.

Avasin Adobe Illustratorin ja lähdin hahmottelemaan mahdollisia muotoja ja ideoita. Visiona minulla oli särmikäs muoto, joka sisältää sovelluksen nimen ”VerseGenerator” ensimmäiset kirjaimet. Suunnitteluprosessissa piti ottaa huomioon ikonin tavoiteformaatti, joka on neliön muotoinen. Asetin siis projektin asetukset neliön mukaan, sekä tarkistin pikselikoon Expon dokumentaatiosta, joka oli 512 x 512. Neliön sisään sommittelin toisen neliön johon piirtää, jotta logo irtoaa reunoista ja saa ”tilaa hengittää”. Tätä tyhjän tilan käyttöä kutsutaan suunnittelun maailmassa ”white spaceksi”. Se on tärkeä elementti, joka tuo visuaaliseen viestintään laadukkuutta. (Babich. 2021.)

Aloitin piirtämisen ja käytin siinä vain 45, ja 90 asteen kulmia. Mittasin myös tarkkaan jokaisen välin kirjaimien ja muotojen välissä ja uudelleenkäytin samoja mittoja pitkin logoa. Taide ja suunnittelu on yleisesti ottaen subjektiivista,

mutta mielestäni tiettyjen säännöllisyyksien käyttö suunnittelussa auttaa luomaan harmonisen kokonaisuuden. Tämän vuoksi käytin logossa myös täysin samoja värejä, jotka olin aiemmin väripalettiini valinnut. Lopputuloksena oli kuvan 7 mukainen logo.

Kuva 7. VerseGenerator-logo

Logon on tarkoitus viestiä laatua, trendikkyyttä ja minimalistisuutta, sillä ne olivat koko sovelluksen teon aikana keskeisiä teemoja, joihin tähtäsin kaikilla valinnoillani.

3 OHJELMOINTIVAIHE

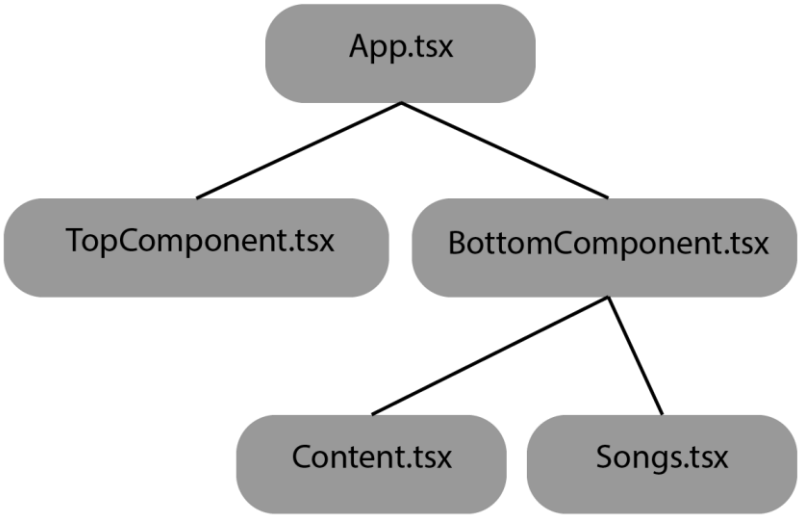
Minulla oli nyt selkeä suunnitelma sovelluksen ominaisuuksista, ne olivat linkitetty rautalankamalliin, josta näkyi sovelluksen rakenne ja värimaailma oli mietitty valmiiksi. Näiden lisäksi olin hakenut valmiiksi kaiken datan, jota ohjelma tulisi tarvitsemaan ja suunnitellut valmiiksi logon. Seuraavaksi oli vuorossa itse ohjelmointi. Kieleksi valitsin Typescriptin, joka on datatyypitetty versio JavaScriptistä. JavaScript-ohjelmointikielen keksi vuonna 1995 Brendan Eich. Se kehitettiin alkujaan Netscape 2 -selaimelle, josta se levisi myöhemmin saataville kaikkiin yleisiin selaimiin. (Aston 2015.)

TypeScriptin valitsin siksi, että se antaa mahdollisuuden tyypittää muuttujia, jolloin virheellistä koodia on vaikeampi tuottaa ja helpompi havaita. En esimerkiksi pysty TypeScriptillä tallentamaan tiedostolle vääräntyyppistä dataa, sillä se vaatii jo koodin kirjoitusvaiheessa, että data on oikeassa muodossa.

TypeScriptillä tehdään ennen kaikkea nettisivuja ja web-sovelluksia, joten miten sillä voisi tuottaa mobiilisovelluksen? Tässä kohtaa kuvaan astuu frameworkit ja kirjastot, joista minä käytin React Nativea ja Expoa. Kaikki konkreettinen ohjelmointi tapahtuu siis TypeScript-kielellä tai kirjastojen valmiiksi rakennetuilla palikoilla, jotka ovat itsessään myös valmiiksi kirjotettuja TypeScript-koodinpätkiä. (Mohan 2019)

Jotta voi käyttää React Nativea, Expoa tai muita TypeScript-pohjaisia koodikirjastoja tai frameworkkeja, tarvitsee myös työkalun, jolla ladata ja hallinnoida niitä. Käytin tähän Node.js:n npm-työkalua. Npm (Node Package Manager) on pakettihallintajärjestelmä, joka tulee Node.js:n mukana. Se tarjoaa tapoja ladata ja asentaa TypeScript-paketteja, jotka ovat valmiita koodikirjastoja tai -työkaluja. Npm:llä voi myös julkaista omia pakettejaan avoimessa pakettivarastossa, jotta ne ovat saatavilla muille kehittäjille. Npm:ää käytetään yleisesti kehitysympäristöissä, joissa käytetään Node.js:ää. Npm:llä on yli 1,3 miljoonaa pakettia ja se on asennettu yli 12 miljoonaan laitteeseen. Sitä voi käyttää komentoriviltä tai integroimalla sen kehitysympäristöön. (Node.js, 2011.)

Aloitin sovelluksen koodausvaiheen asentamalla ja luomalla uuden projektin kansioon Expon omilla npm-komennoilla. Expon projektinluomiskomento tekee juurikansioon samanlaisen rakenteen kuin kaikissa React-projekteissa, eli juurikomponentiksi nimeytyy App.tsx niminen komponentti. Juurikomponenttiin tein aiemmin luomani rautalankamallin mukaisesti komponentit yläpalkille, joka sisältäisi asetukset, sekä alapalkille, joka sisältäisi navigaation. Sovelluksen sisältö oli helpointa sijoittaa alapalkin komponentin sisään, joten sinne loin seuraavaksi komponentit riimittelysivulle ja tallennettujen riimien sivulle. Sovelluksen rakenne oli nyt siis kuten kuvassa 8.



Kuva 8. Sovelluksen komponenttirakenne.

Halusin heti ensimmäiseksi kokeilla aiemmin valitsemiani värejä, joten loin komponentteihin koodit teeman vaihtamiseksi. Käytännössä loin siis kaksi eri React Nativen Stylesheet tyyliobjektia, joista toisen nimi oli styles ja toisen dark. Komponentit tarkistaisivat renderöityessään ehtolauseella kumpaa käytettäisiin, ja tyyliä voisi vaihtaa asetuksista. Tämän toiminallisuuden koodi olisi valmiina, jos haluaisin ottaa sen tulevaisuudessa käyttöön. Toiminto jäi kuitenkin lopullisesta versiosta uupumaan, ja muokkasin koodin valitsemaan oletuksena aina tumman teeman. Valitsemani tummat värit sopivatkin hyvin jo luomiini komponentteihin, joten seuraavaksi lähdin luomaan itse riimittelyn toiminnallisuuksia.

3.1 Riimittelyfunktio

Aloitin koodaamisen BottomComponentin sisällä olleeseen Content komponenttiin. Ensiksi piti luoda tekstikenttä, johon käyttäjä kirjoittaa riimiä, sekä alue, johon riimi ilmestyisi nappeina. Hain näihin tarkoituksiin komponentit React Nativen kirjastosta, lisäsin omat tyylini ja asettelin ne rautalankamallin mukaisesti. Halusin, että käyttäjän kirjoittamat sanat ilmestyisivät näyttöön aina, kun käyttäjä painaa välilyöntiä. Linkitin siis tekstikenttään kirjoitettavaan tekstiin React Nativen useState-muuttujan. UseEffect-toiminto taas aktivoitui joka kerta, kun tekstimuuttuja loppui välilyöntimerkkiin ja työnsi kirjoitetun tekstin sisältämän sanan array-objektiin, jonka jälkeen tekstikenttä tyhjeni. Kuvissa 9 ja 10 toimintojen koodi.

```
<TextInput
        style={(darkmode) ? dark.input : styles.input}
        placeholderTextColor='white'
        placeholder="Start versing..."
        value={verse}
        autoCapitalize='none'
        onChangeText={verse => setVerse(verse)}
    >
</TextInput>
```

Kuva 9. Tekstikenttä, johon käyttäjä kirjoittaa riimiä.

```
useEffect(() => {
    if(verse.endsWith(" ")){
        createButton();
    }
    setError("");
    setMatchesExist(false);
}, [verse]);

const createButton = () => {
    let verseArray = verse.trim().split(" ");
    let lastWord = verseArray[verseArray.length - 1];
    setButtons([...buttons, lastWord]);
    setVerse("");
    setButtonsExist(true);
    updateSessionData(lastWord)
}
```

Kuva 10. Use-effect toiminto ja sananapin luominen.

Loin komponentin return-osioon map-funktion, joka tulosti näkymään buttons-tilamuuttujan sisältämät sanat. Näkymään ilmestyi nyt pötkönä sanat, jotka käyttäjä kirjoitti. Seuraavaksi napeille piti tehdä funktio, joka etsisi sanalle rimmaavan vastineen, kun sitä painetaan. Lisäsin map-kohtaan napeille funktiokutsun, joka lähettäisi parametrinä sen sisältämän sanan. Map-kohta kuvassa 11.

```
{buttons.map((word : string, idx : number) => {
    return <Button
                style={(darkmode) ? dark.wordButton : styles.wordButton }
                key={idx}
                labelStyle={{ color: 'white' }}
                onPress={() => findMatch(word)}
                onLongPress={() => deleteWord(idx)}
            >{word}</Button>
})}
```

Kuva 11. Sanojen tulostus näkymään ja funktiokutsu.

Sitten pääsin koodaamaan itse rimmausfunktiota, joka sai parametrinä riimistä klikatun sanan. Aloitin englanninkielisestä versiosta, joten ensiksi oli mietittävä miten sanat siinä sointuvat hyvin yhteen. Päätteellinen rimmaus oli ilmiselvä, joten lisäsin funktioon muuttujan, joka sisältää sanasta sen viimeiset kolme kirjainta. Sitten lisäsin muuttujan, joka sisälsi koko sanan, sillä joissain tilanteissa sana voi sisältää kokonaan toisen sanan, ja sointua täten todella hyvin, esimerkiksi land -> gland. Lisäsin vielä kolmannen muuttujan, joka sisälsi viimeiset kaksi kirjainta, jota käytettäisiin, mikäli ensimmäisillä ei löytyisi rimmaavia sanoja.

Kirjoitin sitten funktion sarjaksi for-looppeja, joissa käytiin läpi sanalistan sanoja, ja mikäli pareja ei löytynyt tarpeeksi, siirryttiin aina seuraavaan for-looppiin. Sanalista on todella pitkä ja sen loppupäässä sijaitsivat pitkät sanat, joten lisäsin toiminallisuuden, jossa ensimmäisellä napin painalluksella etsittäisiin parhaimpia pareja. Toisella painalluksella tutkittaisiin sanalista toisesta päästä (pitkiä sanoja) ja kolmannella painalluksella lisättäisiin hakuun satunnaisuutta, jotta sai erilaisia tuloksia. Jos yritin hakea koko sanalistan sanoista, sovellus kaatui sen valtavan koon vuoksi, joten muutin funktion sulkemaan for-loopit sadan tuhannen käydyn sanan jälkeen kummastakin päästä. Kuvassa 12

funktion alku, jossa satunnaisuutta lisäävät muuttujat sekä sanojen päätemuuttujat. Kuvassa 13. esimerkki for-loopista, jossa käytetään satunnaisuusmuuttujia.

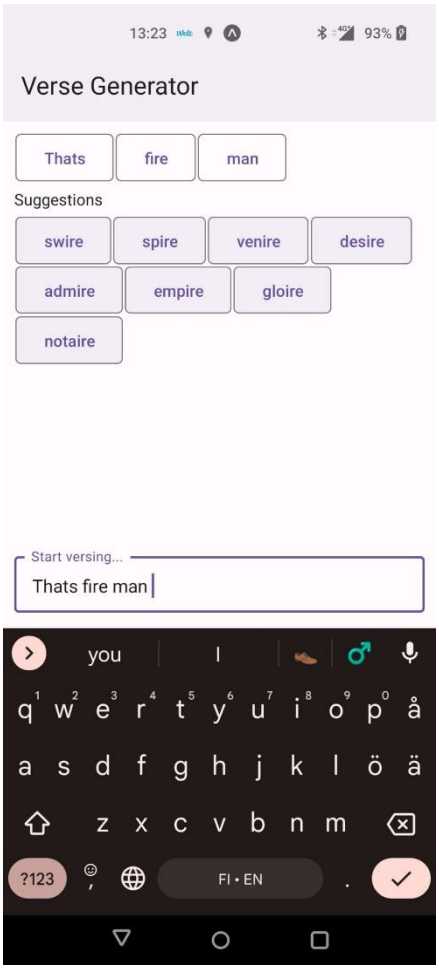
```
const findMatch = (word : string) => {
    setLastWord(word);
    let startingNumber : number = Math.floor(Math.random() * 1000) + 1;
    let increment : number = Math.floor(Math.random() * 20) + 1;
    let end3 : string = word[word.length - 3] + word[word.length - 2] + word[word.length - 1];
    let end2 :  string = word[word.length - 2] + word[word.length - 1];
    let end1 :  string = word.substring(1);
```

Kuva 12. Satunnaisuus- sekä päätemuuttujat.

```
// search random words on third press
if(buttonPressed === 1){
    for(let i = startingNumber; i < wordlist.words.length; i = i + increment){
        loopCount++;
        if(wordlist.words[i].endsWith(end1)){
            results.push(wordlist.words[i]);
            matchCount++
        }
        if(matchCount >= 40){
            break;
        }
        if(loopCount >= 100000){
            break;
        }
    }
```

Kuva 13. Yksi for-loopeista, jossa sanalistaa käydään läpi satunnaisesta aloituskohdasta ja satunnaisista väleistä.

Tässä vaiheessa olin saanut sovelluksen päätoiminnallisuuden toimimaan ja sovelluksen visuaalinen ilme vaalean teeman mukaisesti oli kuten kuvassa. 14.



Kuva 14. Vaalea teema tähän asti.

Näkymä oli tässä kohtaa vielä hyvin paljas ja tyylitön, sillä keskityin ensimmäisenä päätoiminnallisuuksien luomiseen. Tein kuitenkin myös visuaalista ilmettä koko ajan eteenpäin samalla kun työstin toiminnallista koodia.

3.2 Automaattisen tallennuksen funktio

Mobiilisovelluksia voidaan käyttää missä vain. Käyttäjä voi esimerkiksi selata sovellusta junassa. Kun juna saapuu asemalle, sovellus saatetaan nopeasti sulkea tai jättää taustalle, jolloin muuttujien säilymisestä tai manuaalisen tallentumisen tapahtumisesta ei ole takeita. Tämän ongelman vuoksi päätin alkaa tutkimaan mahdollisuuksia automaattiselle funktiolle, joka tallentaa käyttäjän tekemiä tuotoksia tiedostoon joka välissä, kun muutoksia sisällössä tapahtuu. Tiesin, että dataa voi tallentaa tiedostoon noden fs-tiedostokirjastolla, joten lähdin siitä liikkeelle. Aluksi keskityin manuaalisen tallennuksen tekemiseen, enkä lähtenyt ratkaisemaan koko ongelmaa kerralla. Ensimmäinen ratkaisuni täytti konsolin punaisilla virheillä ja kävikin ilmi, ettei kaikkia paketteja

voi React Native-mobiilisovelluksissa käyttää. Niinpä lähdin etsimään vaihtoehtoista järjestelmää ja päädyin käyttämään Expon filesystemiä. Luin dokumentaatiosta, että filesystemillä oli mahdollista tallentaa dataa kahteen eri paikkaan, joista toinen (documentDirectory) oli käyttäjiltä piilotettu kansio, jonka käyttämiseen ei tarvinnut kysyä lupaa puhelimen tiedostoihin, sillä sovellus pystyi tätä reittiä käyttämällä pääsemään käsiksi vain ja ainoastaan kyseiseen kansioon. Tein pakettia käyttäen siis funktiot datan lataamiselle useState-muuttujaan, datan poistamiseen, sekä datan päivittämiseen, jotka ovat kaikki näkyvissä kuvassa. 15.

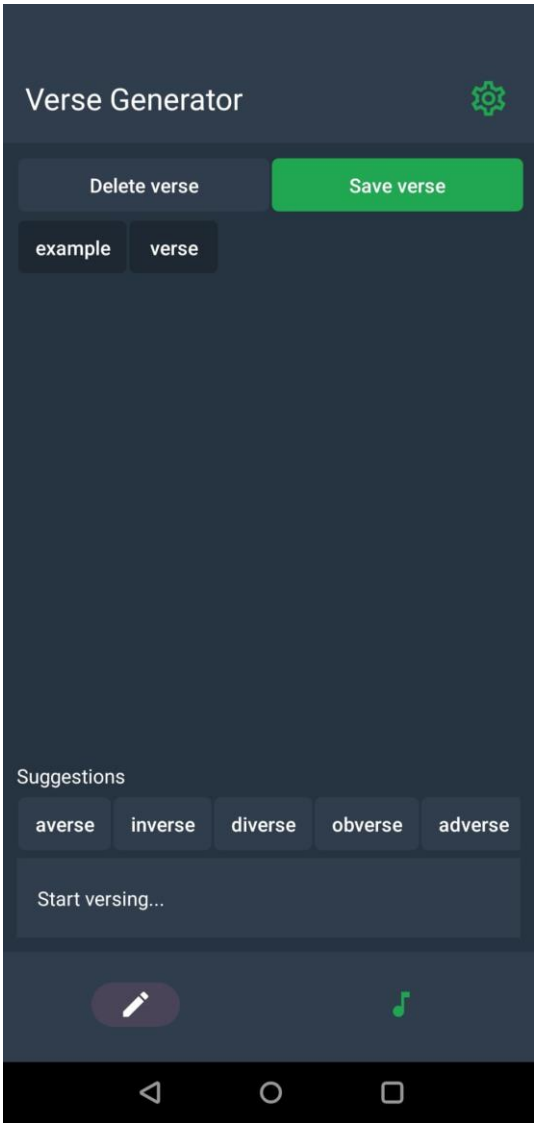
```
const loadSessionData = async () => {
    let filename = "sessiondata.json"
    let fileUri: string = `${FileSystem.documentDirectory}${filename}`;
    if((await FileSystem.getInfoAsync(fileUri)).exists === false){
        let empty : string[] = [];
        await FileSystem.writeAsStringAsync(fileUri, JSON.stringify(empty));
    }
    let json = await FileSystem.readAsStringAsync(fileUri);
    setButtons(JSON.parse(json));
    setButtonsExist(true);
}

const clearSessionData = async () => {
    let filename = "sessiondata.json";
    let fileUri: string = `${FileSystem.documentDirectory}${filename}`;
    if((await FileSystem.getInfoAsync(fileUri)).exists){
        await FileSystem.deleteAsync(fileUri);
    }
    setButtonsExist(false)
}

const updateSessionData = async (lastWord : string) => {
    let filename = "sessiondata.json";
    let fileUri: string = `${FileSystem.documentDirectory}${filename}`;
    let json = [...buttons, lastWord];
    await FileSystem.writeAsStringAsync(fileUri, JSON.stringify(json));
}
```

Kuva 15. Ylimpänä funktio datan lataamiseen tiedostolta muuttujaan, keskellä datan poistaminen ja sen alla datan päivitys, joka saa parametrinään uusimman sanan riimistä.

Sitten yhdistin datan lataamisen funktion useEffect-hookkiin, jotta data luettaisiin tiedostolta, kun komponentti on renderöitynyt. Datan päivittämisfunktion yhdistin funktioon, joka loi uuden sanan. Käytännössä data siis tallennettiin sessiondata.json-nimiseen tiedostoon string-tietotyyppinä automaattisesti joka kerta, kun käyttäjä kirjoitti uuden sanan näkymään. Sovellusta avattaessa data

luettiin tiedostosta, parsittiin takaisin jsoniksi ja sijoitettiin useState-muuttujaan, josta se tulostettiin näkymään map-funktiolla.

Seuraavaksi muutin ja hioin hieman sovelluksen ulkonäköä, lisäsin rautalankamallin mukaiset elementit, joten sovellus näytti lopuksi kuvan 16. mukaiselta.



Kuva 16. Tumman teeman mukainen lopullinen ulkoasu.

Mielestäni ulkoasukin vastasi nyt tavoitteitani. Visuaalinen ilme oli tumman teeman mukainen ja vihreät elementit toivat värimaailmaan raikkautta. Käyttöliittymä oli minimalistinen ja ainakin omasta mielestäni helppokäyttöinen, kun elementtejä ei ollut ruudulla liikaa kerralla. Toki tässä kohtaa erilaisten käyttäjien tekemät laajat testit paljastaisivat toiminnallisuuden ja käyttökokemuksen puutteet.

4 SOVELLUKSEN KOONTIVAIHE

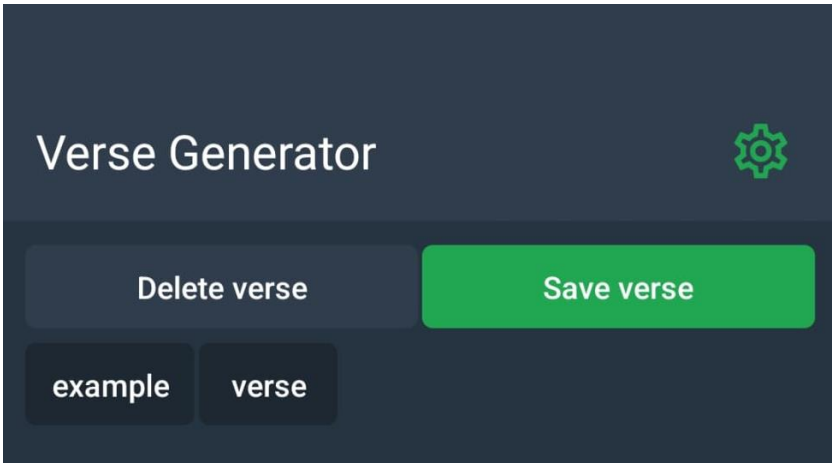
Kirjoittamistani koodeista tulee Android-sovellus koonti- tai ”buildaus”-vaiheessa. Siinä käännetään TypeScript-pohjainen sovellus ja kootaan apk tai aab-tiedosto, jota Android-käyttöjärjestelmä osaa lukea. Projektissani käytän Expo-kirjaston EAS Build-palvelua. Siinä koonti suoritetaan EAS:in (Expo Application Services) palvelimilla, ja valmis .apk (Android Package) tai .aab (Android App Bundle) ladataan myös EAS:in palvelimille, mistä sen voi esimerkiksi ladata omalle koneelle. Koonnin lisäksi palvelussa voi luoda ja tallentaa Googlen vaatimat allekirjoitusavaimet. Allekirjoitusavaimilla (Android Keystore) sovellus kirjataan sen ohjelmoijalle kuuluvaksi. Ne esimerkiksi estävät luvatonta sovelluksen käyttöä; vain yksilölliset kryptografialla luodut avaimet omaava henkilö voi esimerkiksi päivittää tai ladata samoihin avaimiin kirjatun sovelluksen Google Play-kauppaan. (Expo 2022a.)

Ennen kuin kokosin sovelluksesta viimeisen, sovelluskauppaan ladattavan version, halusin koota demoversion ja kokeilla sen toimintoja. Niimpä aloitin koonnin pyörittämällä sovelluksen terminaalissa node-komennon: npx eas-cli build -p android --profile preview. Tämä kokoaa Expon palvelimilla apk-tiedoston, jonka linkin voi koonnin päätyttyä (n.2-20 min) kopioida terminaalista.

Kun linkin avaa Android-puhelimen selaimella, saa sieltä ladattua Androidin sisäiseen jakeluun tarkoitetun sovelluksen. Sisäisen jakelun sovellus on siis tiedostomuodossa apk, eikä sitä ole tarkoitus ladata sovelluskauppaan, vaan käyttää lähinnä vain testaamiseen. Testaamissovellus ilmestyi Androidin käyttöliittymään ikonina, samalla tavoin kuin normaali sovellus, ja käyttäytyi myös samalla tavalla kuin lopullinen, sovelluskaupasta ladattava sovellus käyttäytyisi. Napautin auki sovelluksen ja lähdin testaamaan sen toimintoja. Kaikki toimi kuten kehitysvaiheessakin, yhtä isoa virhettä lukuun ottamatta. Kun yritin aloittaa tekstin kirjoittamisen, tekstikenttä ei noussut näppäimistön yläpuolelle vaan jäi sen taakse piiloon, jolloin käyttäjä ei nähnyt mitä kirjoitti. Ongelmaa oli hyvin vaikea lähteä korjaamaan, sillä sitä ei esiintynyt kehitysvaiheessa, vaan vasta kootussa sovelluksessa.

Minun piti muokata koodia, koota sovellus joka vei aina noin 2-20 minuuttia, sekä sen jälkeen tarkistaa ratkaisiko tekemäni muutokset ongelmaa. Jouduin

lopulta kokoamaan sovelluksen 15 kertaa, ennen kun löysin ongelman syyn. Kuten koodausvaiheen raportissa kuvailin, tekstikenttä laskee sijaintinsa alimman elementin mukaan, joka on näppäimistö sen ollessa esillä. Koottu versio eroaa kuitenkin kehitysversiosta Android-statuskentän osalta, joka on kohta, johon puhelimessa ilmestyy usein mm. ilmoitukset sekä kellonaika. Ero on siinä, että koottu versio laskee Androidin statuskentän pikselit mukaan koko näytön leveyteen ja pituuteen, jolloin tekstikenttä laski sijaintinsa väärin. Poistin Expon koontiasetuksista Android-statuskentän kokonaan, jolloin ongelma ratkesi. Ratkaisu toi kuitenkin pienen kosmeettisen haitan näkymään. Nyt nuo puuttuvat pikselit ilmestyivät sovelluksen yläreunaan tyhjänä alueena. Haitta oli kuitenkin pieni, eikä vaikuttanut toiminnallisuutteen, joten päätin jättää sen sellaiseksi. Kuvassa 17. Näytön yläreunaan jäävä tyhjä tila.



Kuva 17. Tyhjät pikselit Verse Generator tekstin yläpuolella.

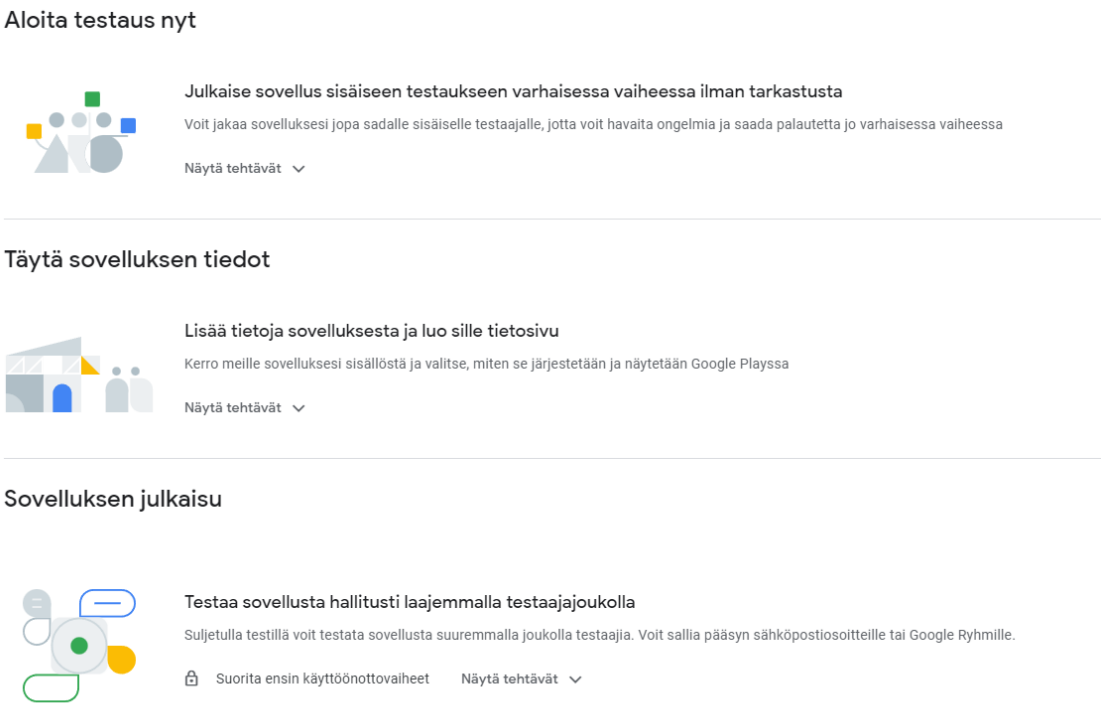
Kun sain apk-version toimimaan, oli aika koota aab-tiedosto, jonka Google asetti pakolliseksi tiedostomuodoksi kaikille Google Play-sovelluksille elokuusta 2021 lähtien. Mikä sitten on apk:n ja aab:n ero? Aab antaa Googlelle mahdollisuuden optimoida sovellukset jokaiselle laitteelle. Jos puhelimessasi on esimerkiksi fullHD-näyttö, on turha tuoda sovelluksen mukana 4k-materiaalia. Loppukädessä siis puhelimellesi latautuu apk-tiedostoja, ne ovat vain optimoituja versioita aab:n vuoksi. (Möllenhoss, 2021)

Sain aab:n koottua vakiokomennolla npx eas-cli build. Minulla oli nyt siis sovelluksen puolesta kaikki tarvittava materiaali valmiina sovelluskauppaan lataamiseksi, joka olikin seuraavana vuorossa.

5 GOOGLE PLAY-STOREEN LATAAMINEN

Sovelluskauppaan ladatakseni tarvitsin ensimmäiseksi Google Play Console käyttäjän, josta löytyy kaikki tarvittavat työkalut sovellusten julkaisuun, päivittämiseen sekä statistiikan seuraamiseen. Sovellusten julkaisemisen valtuudet saadakseni minun oli suoritettava 25 dollarin maksu Googlen developer-ohjelmaan. Koin hinnan reiluksi verrattaessa Applen vastaavaan, joka maksaa 99 dollaria vuodessa. (Apple, 2023)

Google Play Consolen julkaisuprosessissa on monta vaihetta, ennen kuin pääsee näkemään sovelluksensa julkisena Play-kaupassa. Prosessi näytetään sivulla ”tehtäväkortteina” joita pitää suorittaa, että pääsee prosessissa eteenpäin. Tehtävät on kuitenkin aseteltu hämäävästi, sillä niissä on monta testaukseen liittyvää korttia, joita ei kuitenkaan tarvitse tehdä, jos on suorittanut testauksen jo itse ja haluaa vain julkaista sovelluksen. Kuvassa 18. Esimerkkejä tehtäväkorteista.

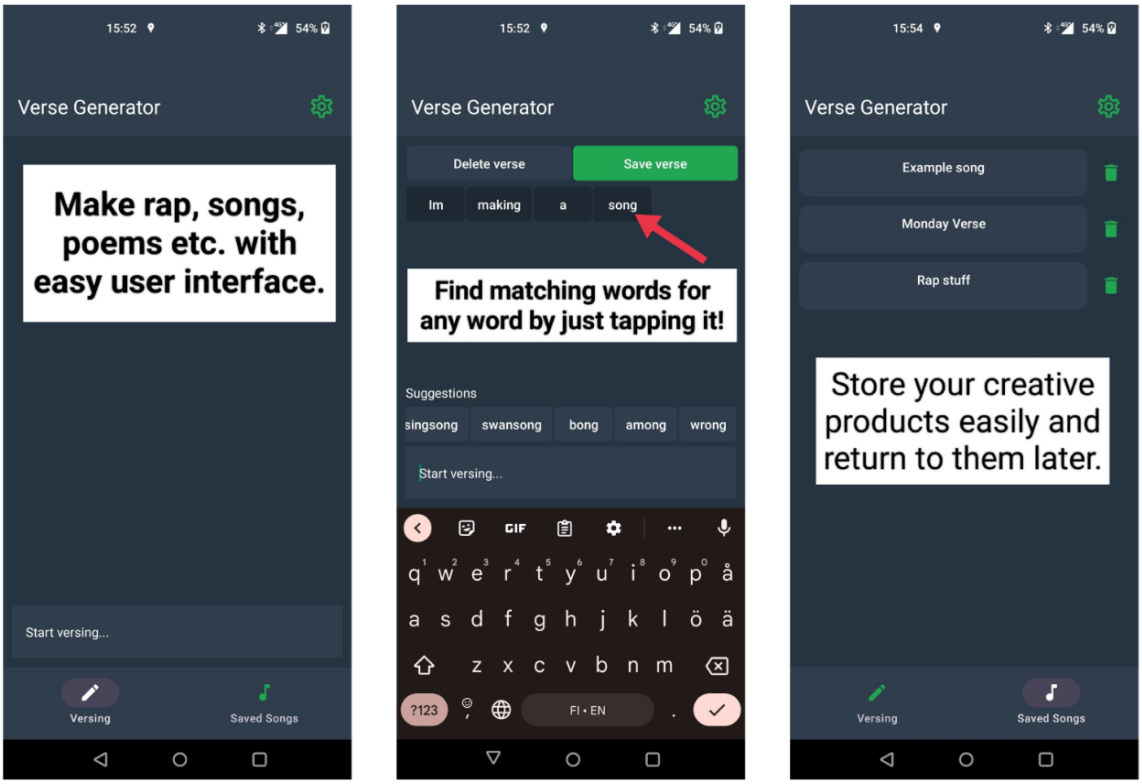


Kuva 18. Esimerkkejä tehtäväkorteista. (Google, 2023)

Jokainen tehtäväkortti pitää sisällään monta tehtävää, enkä käy niitä kaikkia tässä läpi niiden suuren määrän vuoksi. Kortit kysyvät kysymyksiä mm. sovelluksen kohdeyleisöstä, mahdollisista mainontatavoista, ikärajoituksista jne. Tässä kohtaa kysytään myös, onko sovellus viranomaisten sovellus, tai COVID-19 jäljitys-, tai statussovellus. Ymmärtääkseni tätä kysytään, sillä tällaisten sovellusten hyväksymisprosessia vauhditetaan. Seuraavaksi tehtävät kysyivät linkkiä verkkosivulle, josta löytyy sovelluksen tietosuojakäytäntö. Ratkaisin tämän tekemällä julkisen Github-repositorion, jonka readme-tiedostoon lisäsin netistä löytämäni tietosuojapohjan.

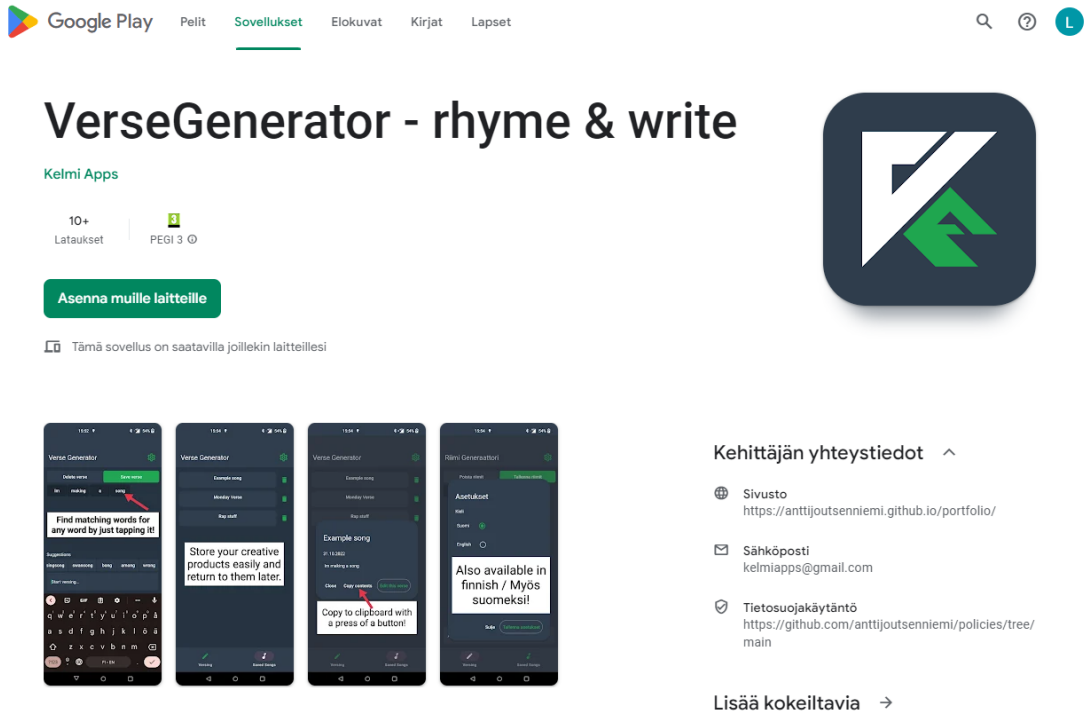
Pohjaan muokkasin sovellukseni yksilölliset tietosuojakäytännöt. Edukseni kääntyi tässä kohtaa se, ettei sovellukseen tarvitse kirjautua tai antaa mitään yksilöllistä dataa. Tämän lisäksi tiedostot, joita sovellus automaattisesti tai käyttäjän toimesta tallentaa tallentuvat Expon suljettuun filesystemiin, joten sovelluksella ei ole tarvetta päästä käyttäjän puhelimen tiedostoihin. Niinpä pystyin tietosuojakäytäntöön kirjoittaa, ettei sovellus tallenna käyttäjistään muuta dataa, kuin heidän kirjoittamansa tuotokset. Kun olin saanut vastattua kaikkiin tehtäviin, pääsin seuraavaksi viimeiseen vaiheeseen, eli sovelluksen tietosivun luomiseen, jossa siis luotiin kaikki elementit, jotka näkyvät sovelluskaupan näkymässä.

Seuraavaksi pääsin vihdoin lataamaan aab-tiedoston Google Play Consoleen. Tämän jälkeen kysyttiin sovelluksen ikoni- ja mainoskuvat, joista jokaisen pikseli- ja tiedostokoot olivat tarkasti määritelty. Avasin siis jälleen Illustratorin ja loin vaatimusten mukaiset projektit, lopullisista kuvista tuli kuvan 19 mukaiset.



Kuva 19. Tietosivun mainokset.

Kuvien lisäksi tietosivulle kirjotettiin esittelytekstit, jotka pystyi vielä kääntämään itse niin monelle kielelle kuin halusi. Tekstit kirjoitettuani pääsin viimein esikatselemaan kaikkia tietoja ja lopulta lähettämään sovelluksen arviointiin. Arviointiprosessissa Google käy läpi lataamani sisällöt ja koodin, näissä menisi heidän tietojen mukaan muutama päivä, mutta luin Android-foorumeilta, että joillakin oli mennyt prosessissa jopa useita kuukausia. Kolmea päivää myöhemmin sähköpostiini kuitenkin kilahti tieto, että sovellus on julkaistu Play-kaupassa. Kuvassa 20 siis viimein näkymä Play-kaupasta.



Kuva 20. Tietokonenäkymä sovelluskauppasivusta.

Sovellus oli nyt siis julkisesti saataville kaikille Android-laitteille, ja pystyin jakamaan sen linkin tuttaville. Tuttavien sitä käyttäessä tulikin esiin bugeja ja virheitä joita sittemmin päivitteli in. Päivityksien lataamisen prosessi toimii käytännössä samalla tavalla kuin alkuunkin, eli sovelluksen koodia muokataan, kootaan uusi aab-tiedosto ja lähetetään se Google Play Consoleen. Päivitetyn sovelluksen Android-manifestiin (konfigurointi tiedosto) pitää muistaa vain lisätä aikaisempaa versiota suurempi versiokoodi, jotta Google hyväksyy sen. Tämän voi myös automatisoida lisäämällä koonnin konfigurointitiedostoon (eas.json) ”production” kohdan alle ”autoIncrement”: true, jolloin koonnissa lisätään aina automaattisesti yhtä arvoa suurempi versiokoodi.

6 PÄÄTÄNTÖ

Opinnäytetyön tavoitteisiin pääsemistä oli helppo mitata, sillä nyt sovellus oli konkreettisesti ladattavissa sovelluskaupassa. Kaikki mainokset ja logot näkyivät kuten kuuluikin, ja sovellus asentui erilaisille Android-laitteille moitteettomasti.

Tässä kohdin sovellusta olisi tietenkin mahdollista jatkokehittää käyttäjien palautteiden mukaan, sillä kaikki tarvittava infrastruktuuri koodin päivittämiseen ja päivitettyjen aab-tiedostojen lähettämiseen oli luotu valmiiksi. Näin kuitenkin projektin ennen kaikkea ponnahduslautana mobiilikehityksen maailmaan, jonka ansiosta olen tätä päätäntöä kirjoittaessani julkaisemassa toista, suurempaa sovelluskokonaisuutta, joka on treenaamis- ja itsekehittämissovellus BecomeChad. Siinä lähdin selvittämään Androidin lisäksi myös iOS-sovellusten koontia, johon React Native ja Expo antavat hyvät mahdollisuudet. Sain riimittelysovelluksesta hyvät eväät uuteen projektiin, kun pyrin tekemään vanhassa esiin tulleita ongelmia paremmin. Hyödynsin myös BecomeChadissa hyviksi toteamiani menetelmiä, kuten automaattisen tallennuksen funktiota.

Kun sain idean toteuttaa riimittelysovelluksen, en tiennyt koulun kursseja lukuunottamatta mobiilisovellusten kokonaisvaltaisesta valmistamisesta juuri mitään. Aikaisemmat projektini olivat kaikki liittyneet web-ohjelmointiin, joten tämä projekti oli tietyllä tapaa hyppy tuntemattomaan. Mielestäni kuitenkin taidot ja varsinkin asennoituminen uuden oppimiseen ja luotto omiin kykyihin paranevat parhaiten, kun astuu kunnolla ulos mukavuusalueelta. En siis ajattele, että Play-kaupasta löytyvä sovellus tai itse sovelluksen idea tai toimivuus ovat tärkeimpiä asioita, joita minulle jäi tästä projektista käteen. Huomasin nimittäin sovelluksen ongelmia yksitellen ratkaistessani muutoksen ajattelutavassani. En enää ajattele minkään yksittäisen asian olevan mahdoton tai liian vaikea opittavaksi. Kaikki tavoitteet ovat saavutettavissa, ja niiden tiellä seisoo vain yksi asia, joka on työn määrä. Jos on valmis tekemään tuon työn sen määrästä riippumatta, saavuttaa lopulta kaikki itselleen asettamat tavoitteet.

LÄHTEET

Agile Alliance. 2022. The Agile Manifesto. WWW-dokumentti. Saatavissa: https://www.agilealliance.org/agile101/the-agile-manifesto/ [viitattu 18.11.2022]

Apple. Enrollment. WWW-dokumentti. Saatavissa: https://developer.apple.com/support/enrollment/ [viitattu 12.1.2023]

Aston, B. 2015. A brief history of JavaScript. Medium. Verkkolehti. Saatavissa: https://medium.com/@_benaston/lesson-1a-the-history-of-javascript-8c1ce3bffb17 [viitattu 28.11.2022]

Babich, N. 2021. What's White Space Design? 5 Real Examples. WWW-dokumentti. Saatavissa: https://xd.adobe.com/ideas/principles/web-design/what-is-white-space-in-design/ [viitattu 26.1.2023]

Expo. 2022a. App credentials explained. WWW-dokumentti. Saatavissa: https://docs.expo.dev/app-signing/app-credentials/ [viitattu 18.11.2022]

Expo. 2022b. EAS Build. WWW-dokumentti. Saatavissa: https://docs.expo.dev/build/introduction/ [viitattu 18.11.2022]

Fisher, M. 2022. What is wireframing? WWW-dokumentti. Saatavissa: https://www.experienceux.co.uk/faqs/what-is-wireframing/ [viitattu 14.12.2022]

Fitts` Law. 2022. Ineraction Design Foundation. WWW-dokumentti. Saatavissa: https://www.interaction-design.org/literature/topics/fitts-law [viitattu 18.11.2022]

Hu, C., Dash, P. 2021. Dark mode may not save your phone's battery life as much as you think, but there are a few silver linings. Purdue University. WWW-dokumentti. Saatavissa: https://www.purdue.edu/newsroom/releases/2021/Q3/dark-mode-may-not-save-your-phones-battery-life-as-much-as-you-think,-but-there-are-a-few-silver-linings.html [viitattu 2.12.2022]

iStock. 2022. The ultimate guide to hex colors. WWW-dokumentti. Saatavissa: https://marketing.istockphoto.com/blog/hex-colors-guide/ [viitattu 14.12.2022]

Kangsoo, K., Erickson, A., Lambert, A., Bruder, G., Welch, G. 2019. Effects of Dark Mode on Visual Fatigue and Acuity in Optical See-Through Head-Mounted Displays. University of Central Florida. WWW-dokumentti. Saatavissa: https://www.researchgate.net/publication/336569145_Effects_of_Dark_Mode_on_Visual_Fatigue_and_Acuity_in_Optical_See-Through_Head-Mounted_Displays [viitattu 2.12.2022]

Kotimaisten kielten keskus. Kotimaisten kielten keskuksen nykysuomen sanalista. 2007. XML-tiedosto. Saatavissa: https://kaino.kotus.fi/sanat/nykysuomi/ [viitattu 11.12.2022]

MDN. 2022a. Working with JSON. WWW-dokumentti. Saatavissa: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON [viitattu 11.12.2022]

MDN. 2022b. XML introduction. WWW-dokumentti. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction [viitattu 11.12.2022]

Mohan, M. 2019. How React works under the hood. WWW-dokumentti. Saatavissa: https://www.freecodecamp.org/news/react-under-the-hood/ [viitattu 28.11.2022]

Möllenhoss, S. 2021. What is an AAB file for Android and how is it different from APK? WWW-dokumentti. Saatavissa: https://www.nextpit.com/what-is-aab-file-android [viitattu 10.1.2023]

Node.js. 2011. What is npm? WWW-dokumentti. Saatavissa: https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/ [viitattu 2.01.2023]

Petersen, K., Wohlin, C. & Baca, D. 2009. The Waterfall Model in Large-Scale Development. Berlin: Springer.