jamk

Mobiilisovelluksen kehittäminen ja julkaisu laskutuspalvelun yhteyteen React Nativella

Aku Lehtonen

Opinnäytetyö, AMK
Huhtikuu 2021
Tietojenkäsittely ja tietoliikenne
Insinööri (AMK), tieto- ja viestintätekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Lehtonen, Aku

Mobiilisovelluksen kehittäminen ja julkaisu laskutuspalvelun yhteyteen React Nativella

Jyväskylä: Jyväskylän ammattikorkeakoulu. Huhtikuu 2021, 43 sivua.

Tietojenkäsittely ja tietoliikenne. Insinööri (AMK), tieto- ja viestintätekniikka. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Verkkojulkaisulupa myönnetty: kyllä

Tiivistelmä

Nykyaikaisten SaaS-palveluiden yksi keskeisimmistä ominaisuuksista on niiden käytettävyys paikasta, laitteesta tai kellonajasta riippumatta. Pienyrittäjille on saatavilla kattava valikoima laadukkaita ja hinnaltaan kilpailukykyisiä ohjelmistoja päivittäisten talouden ja kirjanpidon tehtävien hoitamiseen. Opinnäytetyön tavoitteena oli toteuttaa helppokäyttöinen, aina mukana kulkeva sekä helposti ylläpidettävä mobiilisovellus verkossa toimivan laskutuspalvelun yhteyteen ja tutustua samalla alustariippumattoman mobiilikehityksen maailmaan ja erityisesti React Nativeen.

Laskux Oy:llä oltiin kiinnostuneita mobiilisovelluksen kehittämisestä. Yrityksen laskutuspalvelu on suunnattu erityisesti niille pienille yrityksille, joissa laskuja luodaan jo tien päällä tai esimerkiksi työmaalla. Tutuilta yrittäjiltä saadun sanallisen palautteen perusteella kirjaukset esimerkiksi suoraan varastosta asiakkaan mukaan annetuista tuotteista saattavat jäädä usein epämääräisiksi tai kokonaan tekemättä. Aina taskussa kulkevan laskutussovelluksen avulla asiakkaan ja tuotteiden tiedot voitaisiin kirjata järjestelmään mahdollisimman vaivattomasti ja palata laskuttamaan myöhemmin.

Projektin alkuvaiheissa suunnitteluun hyödynnettiin miellekarttoja sekä suullista ideointia brainstorming-sessioiden muodossa. Toteutus tehtiin alustariippumattomaan mobiilisovelluskehitykseen tarkoitetulla React Nativella, jonka kanssa hyödynnettiin Redux-kirjastoa tilanhallintaan. Sovelluksen taustajärjestelmänä toimii toimeksiantajan kehittämä API-rajapinta. React Native on Facebookin julkaisema JavaScript-pohjainen kirjasto, joka mahdollistaa sovelluksen kääntämisen iOS- ja Android-alustoille samaa koodipohjaa hyödyntäen. Ennen tuotantoon vientiä sovelluksesta julkaistiin useita beta-versioita pienen testiryhmän kokeiltavaksi, joka tarjosi paljon hyvää näkemystä sovelluksen parantamiseen.

Opinnäytetyön tuloksena saavutettiin julkaisukelpoinen mobiilisovellus, joka vastaa toimeksiantajan tarpeita ja on helposti jatkokehitettävissä myös tulevaisuudessa.

Avainsanat (asiasanat)

React Native, JavaScript, Redux, Mobiilikehitys, Julkaiseminen, Alustariippumattomuus, Laskutusohjelmisto

Muut tiedot (salassa pidettävät liitteet)

jamk

Description

Lehtonen, Aku

Mobile application development and publishing for an invoicing service using React Native

Jyväskylä: JAMK University of Applied Sciences, April 2021, 43 pages.

Information and Communication Technologies. Bachelor's Degree Programme in Information and Communications Technology.

Permission for web publication: Yes

Language of publication: Finnish

Abstract

One of the key features of modern SaaS services is their availability regardless of location, device, or time of day. Today, a comprehensive range of high-quality and competitively priced software is available for small businesses to perform day-to-day financial and accounting tasks. The thesis aimed to get familiar with React-Native and implement an easy-to-use and easy-to-maintain mobile application in connection with an online invoicing service.

Laskux Oy was interested in developing a mobile application because their invoicing service is intended especially for small companies, where invoices could be created already on the road or a construction site, for example. Based on verbal feedback from entrepreneurs, for example, entries for products delivered directly from stock according to the customer may often be vague or non-existent. The customer and product information of the always-in-pocket invoicing application could be entered into the system immediately as effortlessly as possible and return to the invoicing later.

In the early stages of the project, a lot of use of mind maps and verbal brainstorming was used for planning. The implementation was done with React-Native, and the Redux library was utilized for space management. The back end of the application is created by Laskux Oy. React Native is a JavaScript-based library published by Facebook that allows the application to be compiled for iOS and Android platforms using the same code base.

As a result of the thesis, a publishable mobile application was achieved that meets the needs of the client and can be easily further developed in the future.

Keywords/tags (subjects)

React Native, JavaScript, Redux, Mobile Development, Publishing, Cross-platform, Invoicing service

Miscellaneous (Confidential information)

Sisältö

Kuviot

1 Työn lähtökohdat

1.1 Aihe ja toimeksiantaja

Opinnäytetyön aiheeksi valikoitui alustariippumattoman mobiilisovelluksen kehittäminen. Toimeksiantaja halusi, että nykyaikaista laskutussovellusta on pystyttävä käyttämään koska tahansa paikasta riippumatta. Toimeksiantajan palvelu onkin suunnattu nimenomaan yrittäjille, jotka saattavat olla usein tien päällä tai esimerkiksi työmaalla. Tällaisissa tilanteissa on käyttäjän kannalta hyödyllistä, kun tarvittavat laskutusohjelman toiminnot kulkevat aina käden ulottuvilla. Tarvittavat kirjaukset laskutukseen liittyen voisi tehdä sovelluksessa, josta ne tallentuisivat suoraan järjestelmään ja niihin olisi helppo palata laskuttamisen yhteydessä. Laskuttajan työ nopeutuisi ja mahdollisten virheiden tai unohtuneiden kirjausten määrää vähenisi.

Opinnäytetyön toimeksiantajana toimi Laskux Oy. Laskux Oy on henkilöstöluokaltaan 1-4 henkilön suuruinen ohjelmistoyritys, jonka kotipaikka sijaitsee Keski-Suomessa Karstulassa. Yritys on perustettu vuoden 2020 alussa. Yrityksen tämänhetkinen päätoimi keskittyy samannimisen SaaS-mallilla tarjottavan laskutusohjelmiston kehittämiseen ja myyntiin. Palvelu on tarkoitettu etupäässä pienille ja keskisuurille yrityksille, mutta palvelu skaalautuu tarvittaessa myös isommankin yrityksen tarpeisiin. Toimeksiantajan kehittämä ohjelmisto toimii selainpohjaisessa käyttöliittymässä.

1.2 Tavoitteet

Opinnäytetyön tavoitteena oli kehittää mobiilisovellus, joka vastaa toimeksiantajan tarpeita, on teknisestä näkökulmasta toteutettu yleisten hyvien käytänteiden mukaisesti, on helppo ylläpitää ja jatkokehittää tulevaisuudessa. Aiempaa kokemusta mobiilikehityksestä toimeksiantajalta löytyi Ionic frameworkista ja React Nativesta. Opinnäytetyössä toteutettavan mobiilisovelluksen teknologiaksi oli ajateltu React Nativea, mutta tarkoitus oli myös kartoittaa ja vertailla muita vaihtoehtoja sovelluksen toteuttamiseksi ja tehdä valinta vertailun pohjalta. Sovelluksen taustalogiikka toimii toimeksiantajan kehittämän rajapinnan kautta, jonka kanssa sovellus on aktiivisessa vuorovaikutuksessa. Sovellus tulee olemaan saatavilla iOS- ja Android-käyttöjärjestelmille ja ladattavissa sovelluskaupoista.

1.3 Tutkimusmenetelmät

Opinnäytetyö toteutettiin soveltavana tutkimuksena, jossa painotetaan käytännön ongelmien ratkaisemista ja hyödynnetään olemassa olevaa tietoa toteuttamalla jokin käytännön sovellus. Soveltavassa tutkimuksessa pyritään luomaan uusia ratkaisuja, joista on teknistä tai taloudellista hyötyä. Tutkimustuloksia vertaillaan teorian ja käytettyjen menetelmien välillä, jotta saavutettaisiin haluttu lopputulos.

Opinnäytetyössä pyritään vastaamaan seuraaviin tutkimuskysymyksiin:

1. Mikä alustariippumattoman mobiilikehityksen teknologioista soveltuu tarkoitukseen parhaiten?
2. Mitä asioita on hyvä ottaa huomioon mobiilisovelluksen suunnittelussa?
3. Miten sovellus kannattaa arkkitehtuuriltaan toteuttaa, jotta sitä on helppoa myös jatkokehittää?

Tutkimusta tehdessä käytetään työntekijän käytännön ja kokemuksen kautta tullutta tietoa, jota tukee toimeksiantajalta saatava tieto ja ammattitaito yhdistettynä internetistä, kirjoista ja artikkeleista saatavaan tietoon systemaattisesti pohtien ja kriittisesti arvioiden.

2 Tavoitteet ja teknologioiden valinta

2.1 Yleistä

Opinnäytetyössä kartoitettiin vaihtoehtoja käytettävästä teknologiasta ennen varsinaisen suunnittelun ja toteutuksen aloittamista. Sovellukselle asetettiin useita yleisiä vaatimuksia:

- Sovelluksen pitää vastata toiminnoiltaan selainversiota
- Saman koodipohjan pitää olla käännettävissä molemmille alustoille (iOS ja Android)
- Sovelluksen pitää olla tyylillisesti brändin ja selainkäyttöliittymän mukainen
- Sovelluksesta pitää olla saatavilla yhtä aikaa beta- ja tuotantoversio
- Sovellukseen pitää voida kirjautua samoilla tunnuksilla kuin selaimessa

Martinin mukaan budjetti on aina iso kysymys, kun suunnitellaan mobiilisovelluksen kehittämistä. Natiivisovelluksen kehittäminen on usein kallista ja aikaa vievää. Tästä syystä on luotu alustariip-

pumattomat sovelluskehykset. Koska jokaisella yrityksellä on erilaiset tarpeet ja mobiilisovelluskehitystä tarjoavien yritysten palveluntarjonta ja ammattitaito eri teknologioiden välillä vaihtelee, ei ole välttämättömyys, että yhdelle yritykselle sopiva sovelluskehys sopii täydellisesti myös toiselle yritykselle. (Martin 2020.)

Teknologioiden vertailuun valittiin neljä eri sovelluskehystä, joiden avulla on mahdollista kehittää alustariippumattomia mobiilisovelluksia. Vertailun tarkoituksena oli selvittää, mikä tarkoitukseen sopivista teknologioista soveltuisi toimeksiantajan vaatimuksiin parhaiten. Teknologiaa valittaessa kiinnitettiin huomiota seuraaviin asioihin:

- Valmiiden komponenttien määrä ja helppokäyttöisyys
- Saatavilla olevan tiedon ja dokumentaation määrä
- Käytettävä ohjelmointikieli
- Mahdolliset kustannukset
- Oppimiskäyrä
- Teknologian pysyminen ajan tasalla teknologian kehittäjän puolelta
- Sovelluksen kääntäminen sovelluskauppoihin sopivaan muotoon

2.2 Teknologioiden vertailu

2.2.1 React Native

Johtuen JavaScriptin suuresta suosiosta ja laajasta tunnettavuudesta, Martinin mukaan React Native on usein ykkösvalinta kehittäjien keskuudessa. JavaScript on yksi kaikkein käytetyimmistä ja soveltuvimmista ohjelmointikielistä alustariippumattomaan sovelluskehitykseen. Lisäksi React Native yhdistelee kaikkia hyviä ominaisuuksia JavaScriptistä ja Reactista ja on Facebookin kehittämä sekä tukema. Martin arvioi React Nativessa eduksi myös sen tarjoaman mahdollisuuden natiivikomponenttien kirjoittamiseen tarvittaessa käyttöjärjestelmäkohtaisilla natiiveilla ohjelmointikielillä. (Martin 2020.)

Aiempaa osaamista toimeksiantajan puolesta löytyi React Nativen osalta, joka myös vaikutti teknologian valinnassa React Nativen eduksi. React Native on myös yksi käytetyimmistä mobiilikehityksen teknologioista, joten sen hallitseminen hyödyttää varmasti myös tulevaisuutta ajatellen. Laajan käyttäjäkunnan ansiosta React Nativesta on myös saatavilla laajasti erilaista tietoa ja dokumentaatiota.

2.2.2 Ionic

Ionic on yksi vanhimmista alustariippumattomaan sovelluskehitykseen tarkoitetuista sovelluskehyksistä. Muihin vertailtaviin kehyksiin verrattuna Ionic toimii täysin web-teknologioihin (HTML ja CSS) pohjautuen. Ionicin toiminta perustuu WebView-näkymään. Sen avulla saadaan upotettua tavallinen verkkosivunäkymä mobiilisovellukseen. Ionic tukee myös TypeScript-kielellä kehitystä, joka on JavaScriptiä vahvemmin tyypitetty ohjelmointikieli.

Jos suorituskyky on korkea prioriteetti, on järkevämpää valita muu tapa rakentaa sovellus kuin Ionic. Xamarin ja React ovat osoittaneet parempia tuloksia suorituskykynsä puolesta, kun sovelluksia on käännetty natiiviksi. Ionicin avullakin on mahdollista rakentaa korkean suorituskyvyn omaavia sovelluksia, mutta se vaatii kehittäjältä syvempää ymmärrystä sovelluksen optimoinnista kuin käytettäessä esimerkiksi Reactia. (Lobastov 2019.)

2.2.3 Flutter

Flutter on Googlen tuottama teknologia. Flutterin avulla pystytään kehittämään näyttäviä natiivityyppisiä sovelluksia. Flutter sisältää oman renderointimoottorin, käyttövalmiit widgetit, komentorivityökalut, API:t ja paljon muuta, joka helpottaa kehitysprosessia. React Nativeen verrattuna Flutterin suosio on silti yhä kasvuvaiheessa, koska se julkaistiin vasta vuonna 2018. (Martin 2020.)

Tapauksissa, joissa prototyyppiä tarvitaan liikeidean testaamiseen ja käyttöliittymältä vaaditaan hyvää mukautuvuutta, on Flutter erinomainen valinta. Sovellus ei käytä kaikkea puhelimen kapasiteettia ja siinä on kohtuullisen yksinkertainen logiikka. Se tarjoaa yhden koodipohjan, muutosten helppouden ja nopeuden. Joka tapauksessa on hyvä mainita, että tällaisten sovellusten jatkokehittäminen johtaa usein suurempiin kuluihin ja ajankäyttöön verrattuna siihen, että tehtäisiin sovellus erikseen jokaiselle alustalle. (Krol 2019.)

2.2.4 Xamarin

On mahdotonta olla sivuuttamatta Xamarinia, kun puhutaan yleisimmistä lähestymistavoista alustariippumattomaan mobiilikehitykseen. Xamarinin avulla 90 % koodista pystytään hyödyntämään alustariippumattomasti. Xamarin pohjautuu Microsoftin teknologioihin ja sillä on jo yli 1.4 miljoonan kehittäjän yhteisö.

Vaikka hybridin mobiilikehityksen työkalut kehittyvät kovaa vauhtia, kärsivät ne silti puutteista suorituskyvyssä ja yhteensopivuusongelmista natiivien ominaisuuksien kanssa, joita Xamarin tarjoaa. Useimmissa tapauksissa nopeus tuotteen markkinoille saamiseksi on ratkaiseva tekijä. Kun yhä useammat alustariippumattomat teknologiat yhdistyvät ja kehittyvät, on Xamarinin yhä vaikeampi säilyttää asemansa markkinoilla. Tänä päivänä joukko sovelluskehyksiä on jo sivuuttanut Xamarinin tunnettuudessa ja suorituskyvyssä. React Native on ollut ylivoimaisesti suosituin, jota on seurannut Flutter. (Altexsoft 2020.)

2.3 Vertailun tulokset

Vertailun perusteella React Native osoittautui sopivimmaksi toimeksiantajan tarpeisiin. React Nativesta löytyy runsaasti dokumentaatiota sekä laaja kehittäjien yhteisö. Johtuen laajasta käyttäjäkunnasta, React Nativeen löytyy myös runsaasti kolmannen osapuolen kirjastoja, päivityksiä tulee usein ja mahdolliset puutteet sekä viat korjataan useimmiten nopealla aikataululla. Tässä tapauksessa myös aiempi JavaScript kokemus katsottiin React Nativen eduksi, kun taas esimerkiksi Xamarin olisi vaatinut ennen sovelluksen kehittämisen aloittamista enemmän perehtymistä C#-ohjelmointikieleen ja .NET-ympäristöön.

Verrattaessa React Nativea Flutteriin, olisi Flutterin käyttöönotto vaatinut ensin perehtymistä Dart-ohjelmointikieleen. React Native -kehittäjille on myös tulevaisuutta ajatellen useimmiten enemmän kysyntää työmarkkinoilla muihin mobiiliteknologioihin verrattuna. Merkittäviä etuja React Nativessa muihin vertailtuihin teknologioihin olivat myös suorituskyky ja Hot Reload -toiminto (ks. kuvio 1).

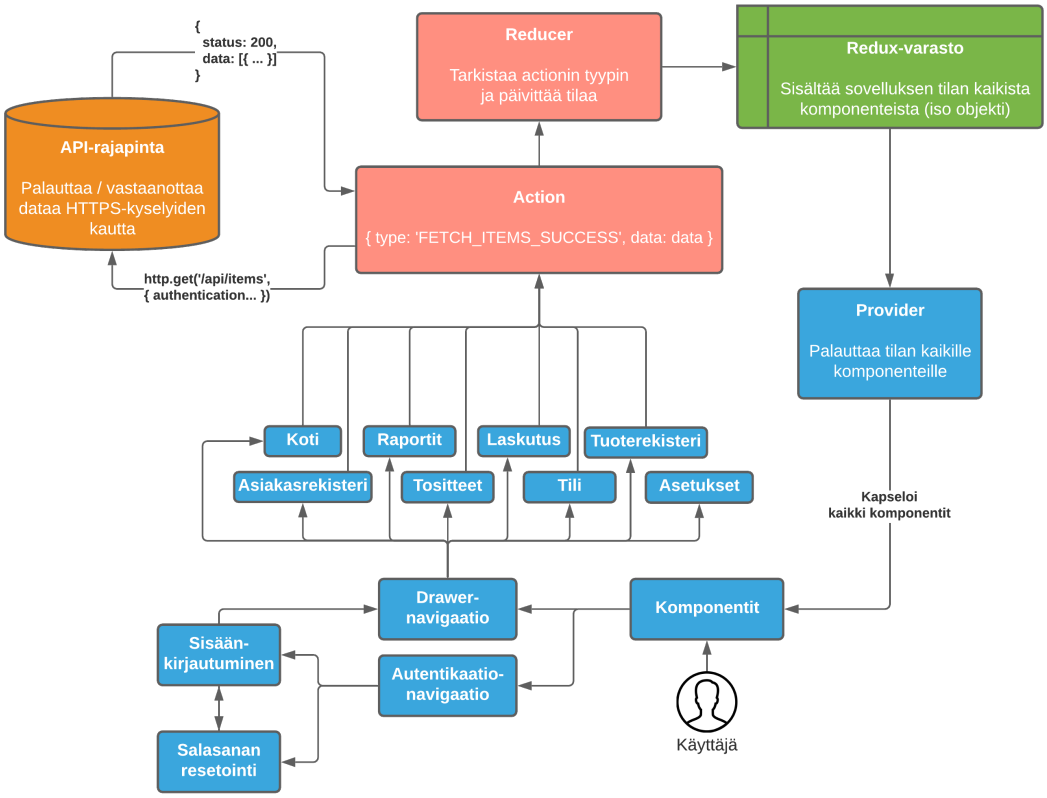
	React Native	Ionic	Xamarin	Flutter
Slogan	A framework for building native apps with React.	One app running on everything.	Complete Binding for the underlying SDKs.	Makes it easy and fast to build beautiful mobile apps.
Kehittäjä	Facebook	Drifty	Microsoft	Google
Kieli	Javascript	TypeScript	C#	Dart
Tehokkuus	Lähellä natiivia	Kohtalainen	Hyvä	Kohtalainen
Käyttöliittymä	Käyttää natiivi UI-komponentteja	WebView (HTML, CSS)	Käyttää natiivi UI-komponentteja	Käyttää natiivi UI-komponentteja
Koodin uudelleenkäytettävyys	90 %	98 %	98 %	50 - 90%
Testaus	Laitteella tai emulaattorilla	Selaimella	Laitteella, emulaattorilla tai pilvipalvelussa	Laitteella tai emulaattorilla
Hot reload	Kyllä	Ei	Ei	Kyllä

Kuvio 1. Alustariippumattomien teknologioiden vertailu

3 Suunnittelu

3.1 Arkkitehtuuri

Sovelluksen rakenne tulee koostumaan eri ruuduista, jotka ovat itsenäisiä React-komponentteja. Sovellukseen tulee drawer-navigointi eli sivusta aukeava valikko. Valikon avulla pystytään liikkumaan eri ruutukomponenttien välillä. Sovelluksen arkkitehtuuri tulee noudattamaan mallia, jossa data noudetaan API-rajapinnasta HTTP-kyselyiden avulla ja tallennetaan paikallisesti Redux-varastoon. Redux mahdollistaa skaalautuvan tilanhallinnan toteuttamisen mobiilisovelluksessa. Redux-varastosta voidaan hakea tietoja ja sen tilaa voidaan päivittää mistä tahansa React-komponentista käsin (ks. kuvio 2).



Kuvio 2. Sovelluksen arkkitehtuurikuvaus

3.2 Käytettävyys

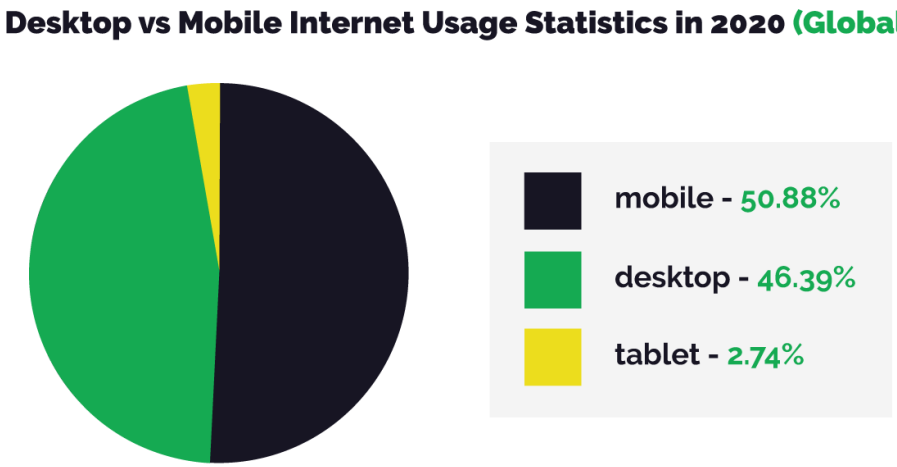
Kun sovellusta aletaan suunnitella, on palvelun selainpohjainen käyttöliittymä kehitetty jo prototyyppiasteelle. Sovellusta lähdetään rakentamaan selainpohjaisen käyttöliittymän pohjalta, jolloin siinä olisi käytettävissä kaikki tai lähestulkoon kaikki ne ominaisuudet, kuin selainpohjaisessa käyttöliittymässä olisi.

Sovellukselle asetettiin heti alussa korkeaksi prioriteetiksi hyvä käytettävyys. Palvelun selainpohjaista käyttöliittymää ei ole suunniteltu suoraan mobiililaitteen ruutuun sopivaksi, joten mobiilikäyttäjät pyritään ohjamaan mobiilisovelluksen käyttöön. Näin pystytään rajaamaan käyttötapaukset tarkemmin ja keskittymään paremmin laitekohtaisten eroavuuksien tunnistamiseen.

Sovellukseen on tarkoitus lisätä myös tuki eri kielien ja maakohtaisten numeroformaattien käyttöön. Käytännössä tämä tarkoittaa sitä, että sovelluksen kieli tullaan valitsemaan laitteeseen vali-

tun kielen mukaan ja numeroformaatti käyttäjän laitteeseen valitun alueen mukaan. Aluksi sovellus tulee tukemaan suomen ja englannin kieltä, sekä suomalasia ja yhdysvaltalaisia numeromuotoja, mutta toteutus on tarkoitus suunnitella niin, että uusia kieliä on jälkeenpäin helppo lisätä tekemättä muutoksia lähdekoodiin.

Käytettävyydeltä vaaditaan myös skaalautuvuutta isommille ruuduille, kuten tablet-laitteille. Kuviossa 3 nähtävän Telemedian julkaiseman kaavion mukaan noin 2,7 % internetin käyttäjistä käyttää tablet-laitetta. Koska React Native käyttää käyttöliittymien tyylittelyyn CSS Flexbox -teknologiaa, onnistuu responsiivisesti toimivien käyttöliittymien toteutus niin mobiili- kuin tablet-laitteiden ruuduille sopivaksi. (Telemedia 2020.)

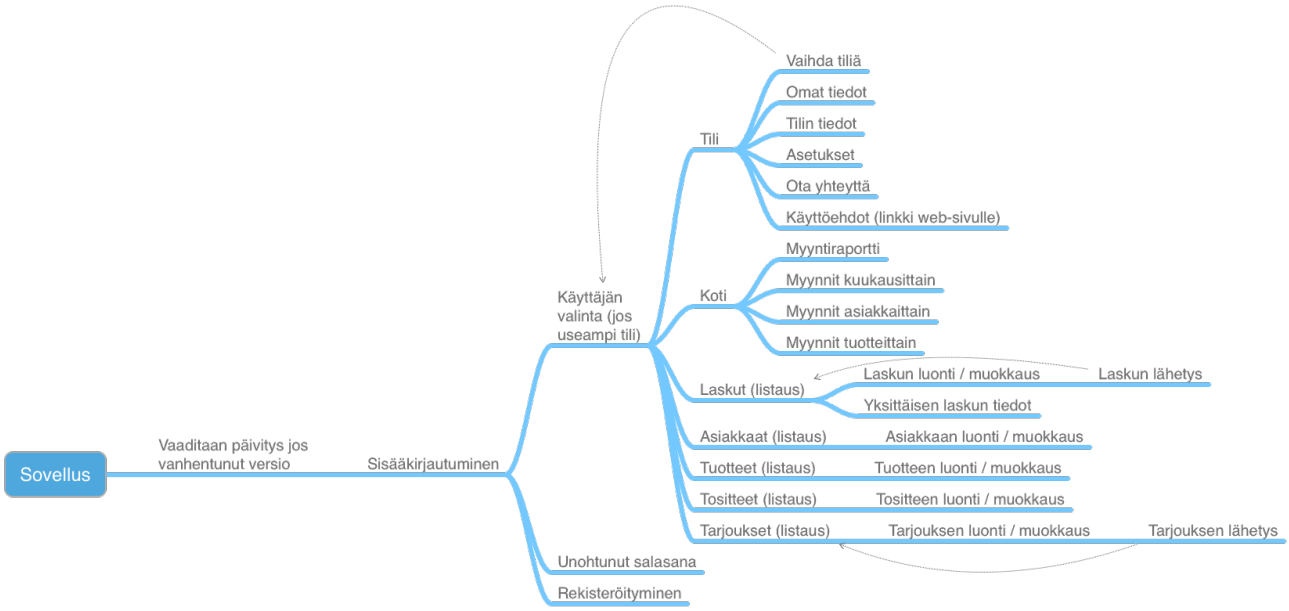


Kuvio 3. Internetin käyttö eri päätelaitteilla (Muokattu, Telemedia 2020)

3.3 Näyttöhierarkia

Näyttöhierarkiaa suunniteltaessa pyritään siihen, että sovelluksen eri toiminnot jaetaan selkeästi omiin ruutuihin ja aliruutuihin. Hyvin suunniteltu näyttöhierarkia parantaa käytettävyyttä ja helpottaa myös sovelluksen tilanhallintaa ja virheiden käsittelyä. Jokaiselle päätoiminnolle luodaan oma pääruutu, josta pääsee etenemään aliruutuihin. Useimmat päätoiminnoista sisältävät ruudut CRUD-mallin mukaan (ks. kuvio 4). CRUD tulee sanoista **c**reate, **r**ead, **u**pdate ja **d**elete. Käytännössä useimmiten pääsivu tulee sisältämään listauksen haetusta datasta, joka sisältää mahdolliset haku- ja suodatustoiminnot. Valittua listan elementtiä kosketettaessa, näytetään valinta saatavilla olevista toiminnoista.

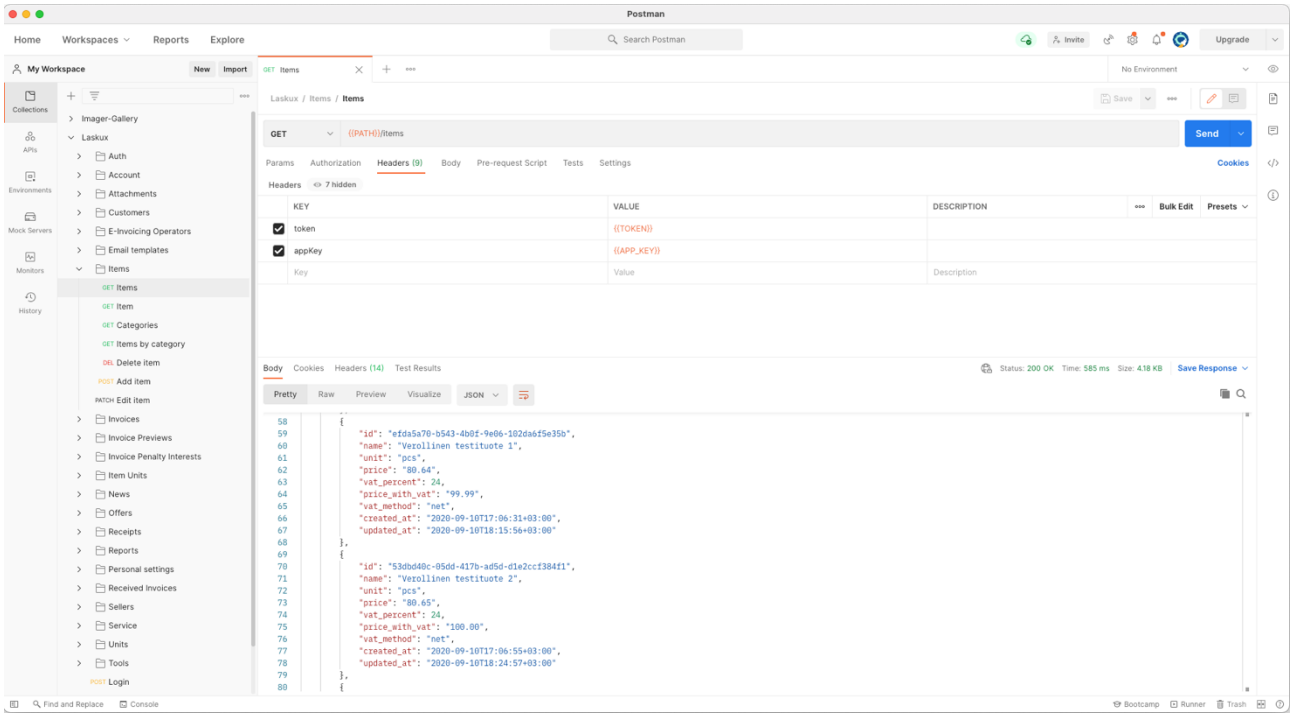
Kun käyttäjä aloittaa sovelluksen käytön, on hänen osattava navigoida eri näkymien välillä ilman erillistä ohjeistusta. Toimiva navigaatio on kuin ajoneuvo, joka kuljettaa käyttäjän sinne mihin halutaan. Toimivan navigaation toteuttaminen on kuitenkin haasteellista mobiililaitteelle johtuen pienestä ruudusta. Epäolennaiset tiedot täytyy pystyä jättämään pois ja priorisoimaan ainoastaan käyttäjän kannalta oleelliset asiat näkyville. Tähän ongelmaan on kehitetty useita erilaisia navigaatiomalleja, mutta siitä huolimatta jokainen näistä sisältää pienempiä tai isompia ongelmia käytettävyydessä. (Babich 2017.)



Kuvio 4. Suunnitelma näyttöhierarkiasta

3.4 API-rajapinta

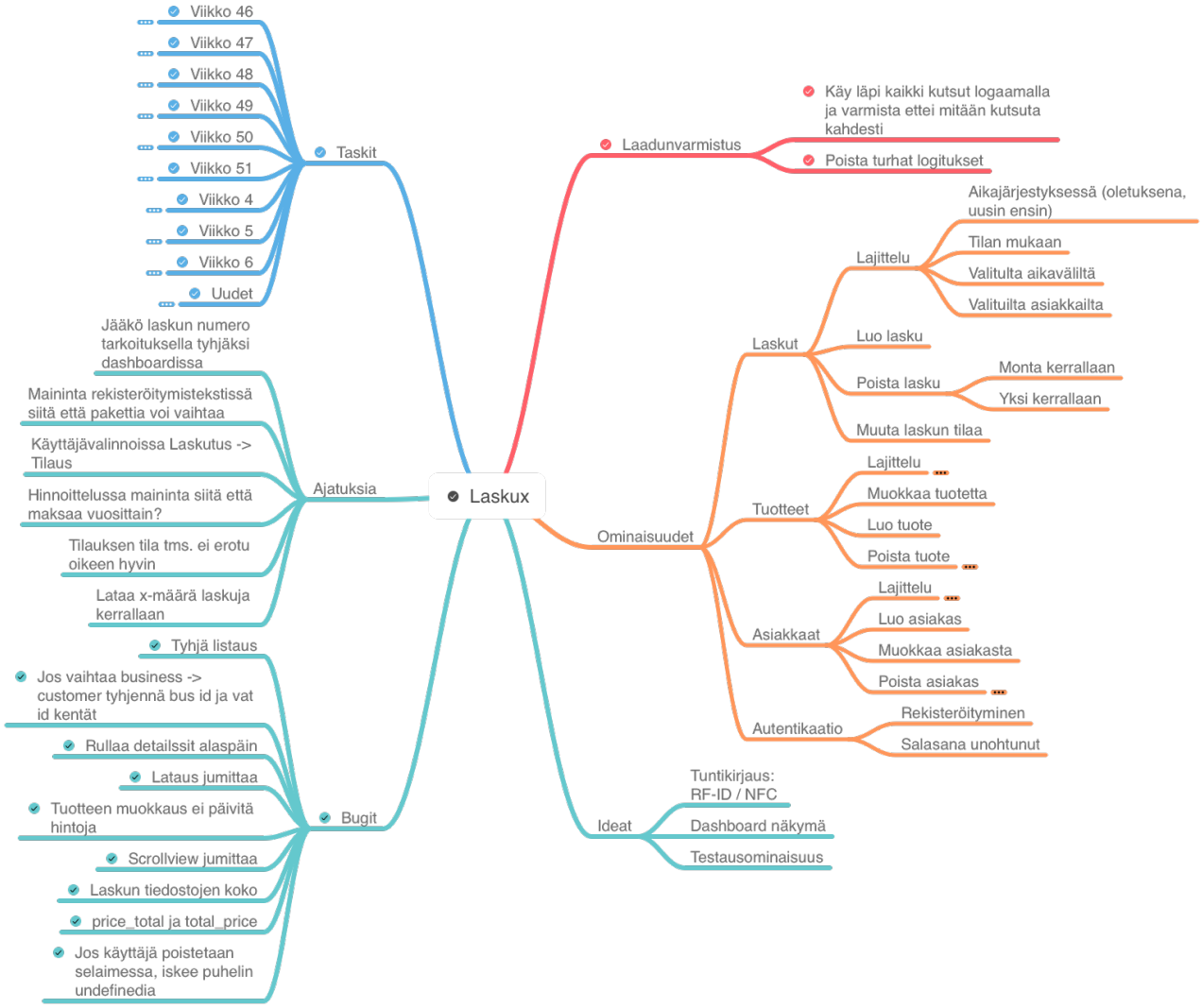
Koska sovellus toimii aktiivisessa vuorovaikutuksessa toimeksiantajan toteuttaman rajapinnan kanssa, käytetään rajapinnan testaamiseen ja sen dokumentointiin paljon työkalua nimeltään Postman. Postman on nimenomaan API-rajapintojen testaamiseen tarkoitettu työkalu, jolla voi lähettää HTTP-kyselyjä haluamillaan tiedoilla mihin tahansa URL-osoitteeseen. Postman mahdollistaa yhteisten työtilojen käytön. Kun rajapinnan kehittäjä tekee muutoksia, voi hän mallintaa kyselyt Postmaniin (ks. kuvio 5), jolloin muut työtilan käyttäjät näkevät muutokset heti. Erityisesti kyseinen työtilaominaisuus tulee helpottamaan mobiilisovelluksen kehittämistä merkittävästi.



Kuvio 5. Postman työtila ja tuotelistaus noudettuna

3.5 Projektinhallinta

Projektin edetessä tullaan käyttämään Jira-tehtävienhallintaohjelmistoa projektitehtävien hallintaan ja Confluence-organisaatiowikiohjelmistoa esimerkiksi HTTP-kyselyiden tarkempaan kuvaamiseen ja yleiseen dokumentaatioon. Projektin alkuvaiheen suunnittelussa hyödynnetään paljon MindNode nimistä miellekartta-työkalua (eng. mind map). Miellekartta on helppokäyttöinen suunnittelun väline etenkin projektin varhaisessa vaiheessa, kun luonnostellaan uusia ominaisuuksia ja ideoita (ks. kuvio 6).



Kuvio 6. Miellekartta suunnittelun apuvälineenä

4 Tekninen toteutus

4.1 Kehitysympäristö

React Native -projektien toteuttamiseen on olemassa kaksi tapaa. Ensimmäinen tapa on käyttää Expo CLI:tä, joka on tarkoitettu vähemmän kokeneille kehittäjille. Expo tarjoaa valmiiksi määritellyn sovelluskehyksen React Native -sovellusten kehittämiseen, kääntämiseen ja julkaisuun. Expo tarjoaa myös verkkoselaimessa toimivan Snack-testaustyökalun React Native -sovelluksen ajamiseen. Vaikka Expo tarjoaa hyödyllisiä työkaluja ja nopeuttaa kehitystä, se asettaa myös joitain rajoituksia. Expo rajoittaa useiden kolmansien osapuolien kirjastojen käyttöä projekteissa. Expoa käytettäessä kehittäjä ei itse pääse hallitsemaan sovelluksen Xcode- ja Android Studio -projekteja, joiden pohjalta sovellus viimein käännetään eri alustoille sopiviksi versioiksi.

Toinen tapa on käyttää React Native CLI:tä. React Native CLI:tä käytettäessä projektin hakemistoon tulee näkyviin lähdekoodin lisäksi myös iOS- ja Android-projektit, jolloin kehittäjä pääsee hallitsemaan paremmin alustakohtaisia projektiasetuksia. Kehitysympäristöksi valittiin React Native CLI, sillä etukäteen oli tiedossa, että opinnäytetyössä toteutettavan sovelluksen kanssa käytetään laitteen natiivirajapintoja sekä alustakohtaisten projektien hallinnasta löytyi aiempaa kokemusta.

React Native CLI -kehitysympäristön pystyttäminen vaati muutamia valmisteluja. Ensin tuli asentaa Node-ajoympäristö. Seuraavaksi asennettiin Android Studio, sekä Xcode ohjelmistot. Android Studion lisäksi tarvittiin Java Development Kit, jotta voitiin kääntää ja ajaa sovellusta Androidilla. Kun kaikki tarvittavat ohjelmistot oli asennettu, voitiin projekti perustaa komennolla.

```
npx react-native init laskux-mobile-app
```

React Native sisältää pikalataus (eng. hot reload) toiminnon. Pikalataus tarkoittaa, että tehtäessä muutoksia koodiin tai muuhun projektin tiedostorakenteeseen, sovellus latautuu välittömästi uudelleen n. 2 sekunnissa, jolloin tehdyt muutokset ovat heti nähtävissä ja testattavissa. Kun projekti oli perustettu, voitiin sovellus käynnistää joko Android- tai iOS-emulaattorilla komennolla.

```
npx react-native run-android --variant=release --appId fi.laskux.mobile
```

4.2 Navigaatio

Kun projekti oli perustettu, aloitettiin sovelluksen rakenteen toteutus lisäämällä sovellukseen navigaatio. Navigaation toteuttamiseen käytettiin kirjastoa nimeltä react-navigation. Sovelluksessa käytettiin sivusta aukeavaa eli niin sanottua drawer-valikkotyyppiä ruutujen välillä navigointiin. Sivusta aukeava valikko säästää tilaa etenkin pienillä ruuduilla verrattuna ruudun alalaidassa sijaitsevaan tab-valikkotyyppiin. Navigaation animaatioiden toiminta vaati lisäksi react-native-gesture-handler nimisen kirjaston. Kirjastot asennettiin komennolla.

```
npm install @react-navigation/native @react-navigation/drawer @react-navigation/stack
```

Navigaation rakenteesta tuli hierarkinen. Ensin luotiin päänavigaatiokomponentti, joka sisältää kaksi alinavigaatiokomponenttia. Toinen alinavigaatiokomponenteista sisältää autentikaatioon tarvittavat ruudut ja toinen itse sovelluksen ruudut. Kun käyttäjä ei ole kirjautunut sisään, näytetään autentikaatio navigointikomponentti ja kun sisäänkirjautuminen on onnistunut, näytetään sovelluksen päänavigaatiokomponentti. Päänavigaatiokomponentti sisältää sovelluksen varsinaiset ruudut. Oikean navigaatiokomponentin näyttäminen esitetään allaolevassa lähdekoodissa.

```
return (
  <NavigationContainer>
    <View style={styles.container}>
      {account.state === RequestStates.SUCCESS ? (
        <AppContainer />
      ) : (
        <AuthenticationNavigator initialRouteName="Login" />
      )}
    </View>
  </NavigationContainer>
);
```

AppContainer-komponentti sisältää sivustavetovalikossa näkyvät pääruudut. Sivuvalikon ruuduille määritettiin tunniste, näytettävä komponentti, nimi ja ikoni alla olevan lähdekoodin mukaisesti.

```
{
  name: 'Invoices',
  component: InvoicesScreen,
  options: {
    drawerIcon: ({focused}) => (
      <NavigatorIconWrapper>
        <DrawerNavigatorIcon focused={focused} name="file1" size={22} />
      </NavigatorIconWrapper>
    ),
    drawerLabel: Translations.INVOICES,
}
```

Sivuvalikkoon tulevat ruudut lisättiin AppStack.Navigator-pääkomponentin sisään yhdessä aliruutujen kanssa. AppStack.Navigatorille voidaan määritellä erilaisia parametrejä, kuten tyyli ja animaatioasetukset sekä valita oletuksena aukeava ruutu alla olevan lähdekoodin mukaisesti.

```
<AppStack.Navigator initialRouteName="HomeTabs">
  <AppStack.Screen name="HomeTabs" component={MainScreens}/>
  <AppStack.Screen
    name="CreateInvoice"
    component={CreateInvoiceScreen}
```

```
    options={{
      title: Translations.CREATE_INVOICE,
      ...getLeftCloseButton(),
    }}
  />
</AppStack.Navigator>
```

Kun kaikki ruudut oli lisätty yhden pääelementin sisälle, onnistui mistä tahansa ruudusta navigoida toiseen ali- tai pääruutuun helposti koodista React-Navigationin sisältämää navigate-funktiota käyttäen alla olevan lähdekoodin mukaisesti.

```
navigation.navigate('CreateCustomer');
```

4.3 Tilanhallinta

Sovelluksen kehityksen edetessä pidemmälle ja koodipohjan kasvaessa ja jakautuessa useisiin eri komponentteihin ja niiden alikomponentteihin, on työlästä ylläpitää sovelluksen eri tiloja aina yksittäisten komponenttien sisällä. Tähän on ratkaisuna erityiset tilanhallintaan suunnitellut kirjastot, joista tunnetuimpana Redux-kirjasto, joka on yksi käytetyimmistä Reactin ja React Nativen kanssa käytetyistä tilanhallintakirjastoista. Reduxissa sovelluksen tila varastoidaan Redux-varastoon. Redux asennettiin projektiin komennolla.

```
npm install redux react-redux
```

Reduxin asentamisen jälkeen luotiin Redux-varasto, joka sisältää halutut reducerit, jotka taas sisältävät aina jonkin etukäteen määritellyn osan tilanhallinnasta, esimerkiksi autentikaation tilan. Sovelluksen juurikomponentti ympäröitiin erityisellä Reduxin sisältämällä Provider-komponentilla, jolle annetaan parametriksi luodut reducerit alla olevan lähdekoodin mukaisesti. Näin mahdollistettiin Redux-varastoon pääsy mistä tahansa sovelluksen osasta.

```
const rootReducer = combineReducers({
  fetchAppVersion: fetchAppVersionReducer,
  login: loginReducer,
  logout: logoutReducer,
  fetchAccount: fetchAccountReducer,
  fetchAccountDetails: fetchAccountDetailsReducer,
  setAccount: setAccountReducer,
```

```
  fetchInvoices: fetchInvoicesReducer,
  fetchAccounts: fetchAccountsReducer,
  fetchInvoice: fetchInvoiceReducer,
});

const store = createStore(rootReducer, applyMiddleware(thunk, errorCatcher,
clearStates));

export default function App() {
  return (
    <View style={styles.container}>
      <Provider store={store}>
          <AppNavigator />
      </Provider>
    </View>
  );
}
```

Reducerit kirjoitettiin omiin tiedostoihinsa. Reducer sisältää aina alkutilan. Esimerkiksi rajapintaan tehtävä kysely tuotteista voidaan kirjoittaa omaan reduceriin, jolloin sen tilaan ja dataan päästään käsiksi sovelluksesta. Reducerille määriteltiin alla olevan lähdekoodin mukaisesti alkutilaksi HTTP-kyselyn tila, mahdollinen rajapinnan palauttama virhe sekä tyhjä taulukko, joka sisältää rajapinnasta palautuvat tuotteet.

```
import {
  FETCH_ITEMS_REQUEST,
  FETCH_ITEMS_REFRESH,
  FETCH_ITEMS_SUCCESS,
  FETCH_ITEMS_FAILURE,
  FETCH_ITEMS_RESET_STATE,
} from '../../actions/items/fetchItems';

import RequestStates from '../../../../constants/States/RequestStates';

const initialState = {
  state: null,
  error: null,
  items: [],
};

const fetchItemsReducer = (state = initialState, {type, data}) => {
  switch (type) {
    case FETCH_ITEMS_REQUEST:
      return {...state, state: RequestStates.LOADING, error: null};
    case FETCH_ITEMS_REFRESH:
      return {...state, state: RequestStates.REFRESHING, error: null};
    case FETCH_ITEMS_SUCCESS:
      return {...state, state: RequestStates.SUCCESS, items: data};
    case FETCH_ITEMS_FAILURE:
      return {...state, state: RequestStates.ERROR, error: data};
    case FETCH_ITEMS_RESET_STATE:
      return initialState;
    default:
      return state;
```

```
  }
};

export default fetchItemsReducer;
```

Tilan muuttaminen tapahtuu lähettämällä reducerille toimintoja (eng. action). Toimintojen sisällä voidaan tehdä esimerkiksi HTTP-kyselyjä API-rajapintaan. Kyselyn edetessä tila ja palautunut data viedään reducerille. Haettaessa tuotteita rajapinnasta, määritellään kyselyn tilaksi ensin ”loading”. Kun rajapinta on palauttanut haetut tuotteet, muutetaan tilaksi ”success” ja lähetetään lista palautuneista tuotteista reducerille. Mikäli rajapinta palauttaa jostain syystä virheen, muutetaan tilaksi ”error” ja palautetaan rajapinnasta palautunut virheviesti reducerille. Alla olevassa lähdekoodissa on esitetty tuotelistauksen noutamiseen käytettävän toiminnon rakenne.

```
export const FETCH_ITEMS_REQUEST = 'FETCH_ITEMS_REQUEST';
function fetchItemsRequest() {
  return {
    type: FETCH_ITEMS_REQUEST,
  };
}

export const FETCH_ITEMS_REFRESH = 'FETCH_ITEMS_REFRESH';
function fetchItemsRefresh() {
  return {
    type: FETCH_ITEMS_REFRESH,
  };
}

export const FETCH_ITEMS_SUCCESS = 'FETCH_ITEMS_SUCCESS';
function fetchItemsSuccess(data) {
  return {
    type: FETCH_ITEMS_SUCCESS,
    data: data,
  };
}

export const FETCH_ITEMS_FAILURE = 'FETCH_ITEMS_FAILURE';
function fetchItemsFailure(error) {
  return {
    type: FETCH_ITEMS_FAILURE,
    data: error,
  };
}

export const FETCH_ITEMS_RESET_STATE = 'FETCH_ITEMS_RESET_STATE';
export function fetchItemsResetState() {
  return {
    type: FETCH_ITEMS_RESET_STATE,
  };
}

export function fetchItems(path, token, isRefreshing = false) {
  return async function (dispatch) {
```

```
    if (isRefreshing) {
      dispatch(fetchItemsRefresh());
    } else {
      dispatch(fetchItemsRequest());
    }
    try {
      const items = await getRequest(path, token);
      dispatch(fetchItemsSuccess(items));
    } catch (error) {
      dispatch(fetchItemsFailure(error));
    }
  };
}
```

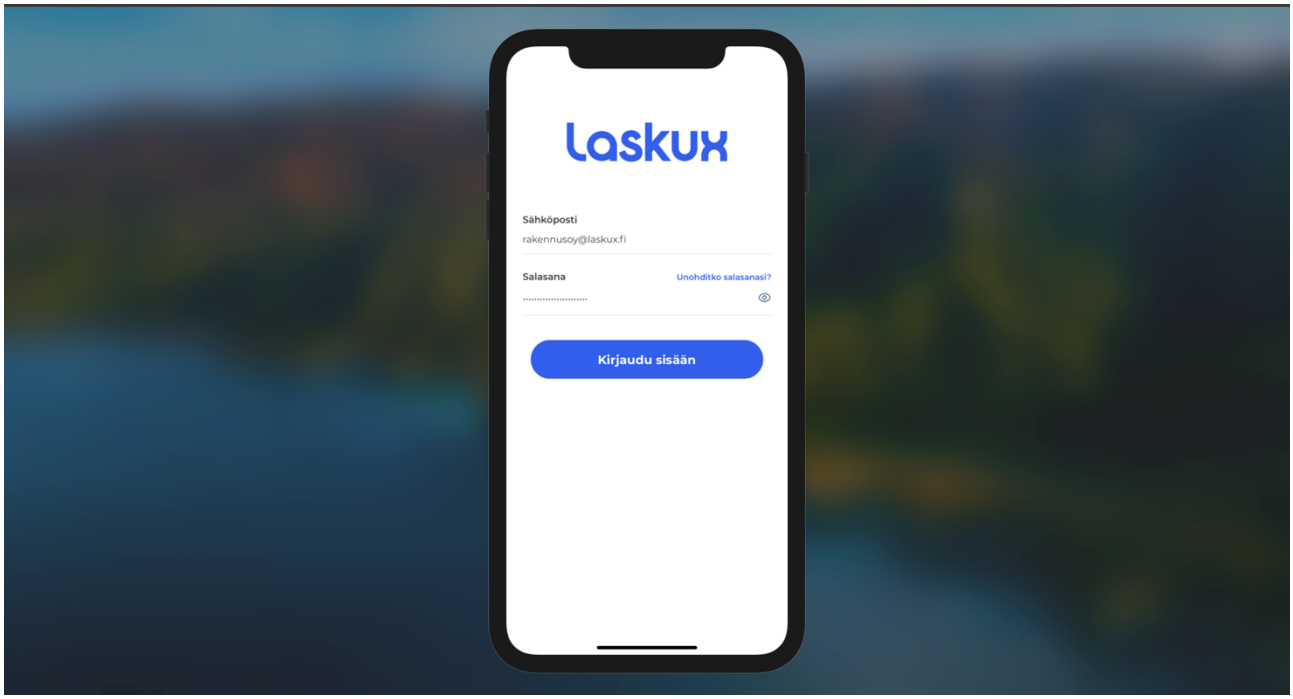
Kun tila tallennettu Redux-varastoon, voidaan mistä tahansa sovelluksen osasta saada tieto, onko tuotteiden haku onnistunut sekä päästä käsiksi haettuihin tuotteisiin. Mikäli kysely ei ole onnistunut, saadaan Redux-varastosta haettua rajapinnasta palautunut virheviesti. Redux-varaston tilan saa haettua esimerkiksi Reduxin useSelector-funktion avulla allaolevan lähdekoodin mukaisesti.

```
const fetchItemsState = useSelector((store) => store.fetchItems);
const items = fetchItemsState.items;
```

4.4 Ominaisuudet

4.4.1 Autentikaatio

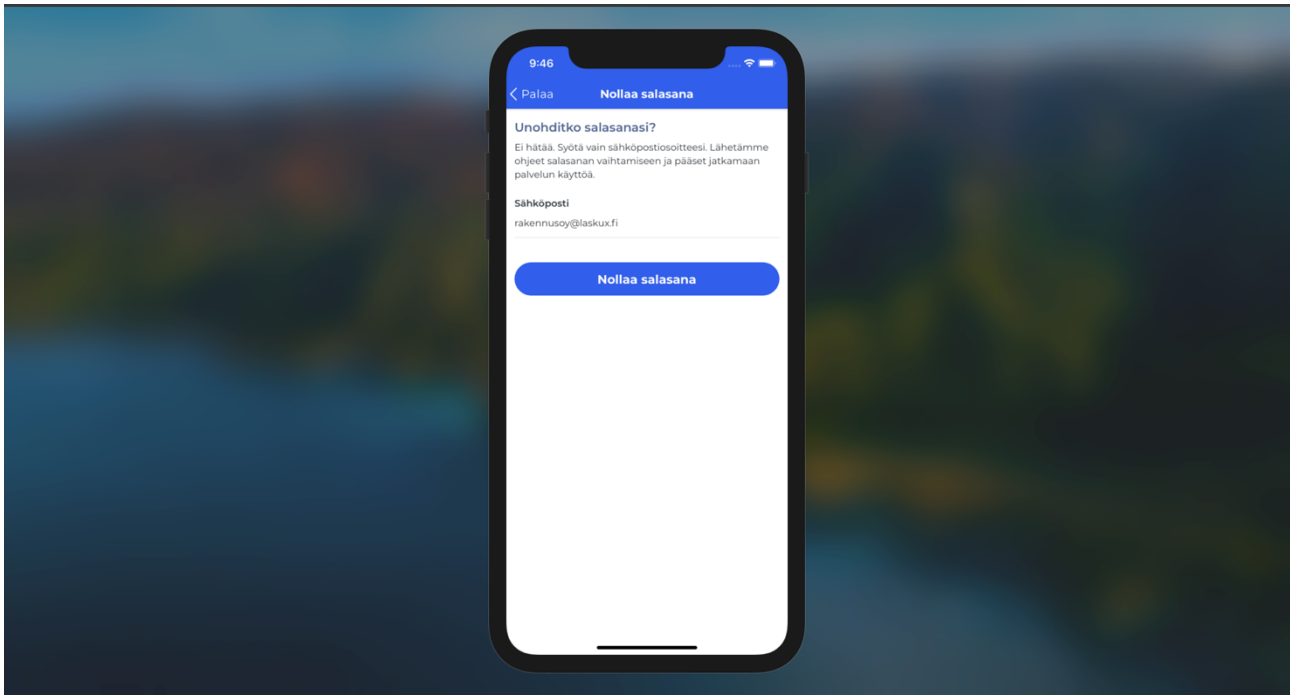
Sovellukseen kirjaudutaan palveluun luotuja tunnuksia käyttäen. Alkuvaiheessa rekisteröitymistä ei toteutettu suoraan sovellukseen. Kirjautumiseen tarvitaan sähköpostiosoite ja salasana (ks. kuvio 7). Kirjautumistiedot lähetetään API-rajapintaan. Kun kirjautuminen onnistuu, käyttäjälle luodaan rajapinnassa tunnistautumiseen käytettävä avain (eng. token). Avain on pitkä sattumanvaraisen merkkijono, joka tallennetaan käyttäjän laitteeseen ja tietokantaan. Avain on toiminnassa, kunnes käyttäjä kirjautuu ulos palvelusta tai avain vanhentuu. Avainta käytetään tunnistautumiseen kun lähetetään muita HTTP-kyselyjä sovelluksesta rajapintaan. Käyttämällä avainta vältytään siltä, ettei käyttäjän tunnuksia kuten sähköpostia tai salasanaa tarvitse lähettää jatkuvasti verkon yli kulkevissa HTTP-kyselyissä, joka saattaisi aiheuttaa tietoturvariskejä.



Kuvio 7. Sisäänkirjautuminen

Sisäänkirjautumisen yhteydessä on myös oma näkymä ja toiminnallisuus salasanan nollaamiseen (ks. kuvio 8). Käytännössä tämä toteutettiin toimimaan alkuvaiheessa niin, että käyttäjä syöttää sähköpostiosoitteen sovelluksen kenttään ja painaa salasanan nollauspainiketta. Sen jälkeen käyttäjälle lähetetään sähköpostiin linkki, jota painamalla hänet ohjataan palvelun verkkosivuille, jossa hän pystyy luomaan uuden salasanan. Tämän jälkeen käyttäjä palaa sovellukseen ja kirjautuu normaalisti uudella salasanallaan.

Salasanan palauttamisesta jäi sovellukseen hyvä jatkokehityksen mahdollisuus. Salasanan palautus voitaisiin toteuttaa mobiilisovellukselle optimaalisemmin, esimerkiksi lähettämällä käyttäjälle tekstiviesti. Toinen vaihtoehto olisi tunnistaa taustajärjestelmässä, onko salasanan palautus pyydetty mobiilisovelluksesta vai selaimesta ja mikäli se on mobiilisovelluksesta, liitettäisiin sähköpostiviestiin erityinen sovelluksen avaava linkki. Sitten uuden salasanan syöttö voitaisiin rakentaa suoraan sovellukseen, joka helpottaisi prosessia käyttäjän näkökulmasta.



Kuvio 8. Salasanan nollaus

Palveluun rakennettiin toiminto, jossa yhden käyttäjätunnuksen alla voi olla useampia käyttäjätilejä. Käytännössä, mikäli käyttäjällä on esimerkiksi kaksi yritystä, pystyy hän hallinnoimaan molempien yritysten laskutusta saman käyttäjätunnuksen alta. Sovelluksessa tämä on huomioitu sisäänkirjautumisen yhteydessä (ks. kuvio 9). Kun käyttäjä on kirjautunut sisään, ennen varsinaiseen sovellukseen ohjaamista, hänet ohjataan listaukseen, josta valitaan käytettävä tili.

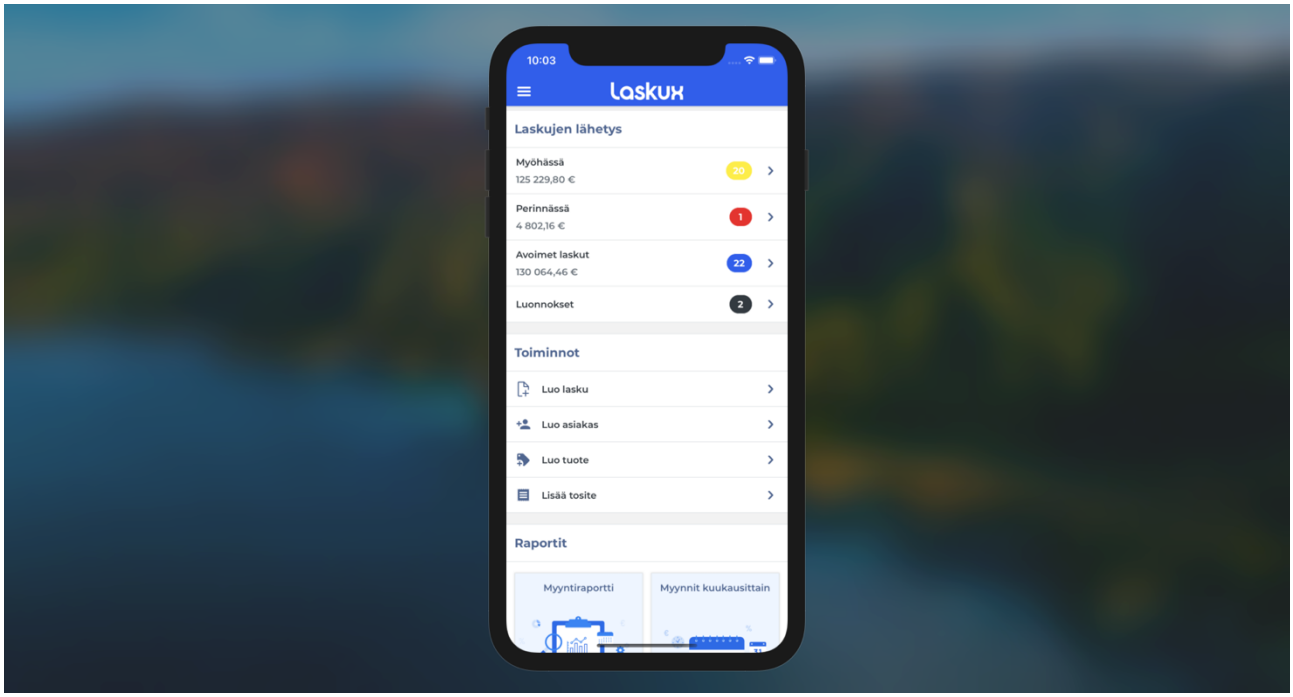
Sisäänkirjautumisen yhteydessä tehtävän käyttäjätilin valinnan lisäksi sovellukseen tehtiin mahdollisuus vaihtaa aktiivista käyttäjätiliä Tili -ruudun kautta. Käytännössä valinta toteutettiin niin, että sisäänkirjautuminen palauttaa kaikki käyttäjätunnukseen liitetyt tilit ja niille yksilölliset avaimet. Kun tili valitaan, tallennetaan valitun tilin avain ja muut yleistiedot Redux-varastoon, josta ne haetaan tehtäessä HTTP-kyselyjä. Mikäli tiliä vaihdetaan, haetaan tilit uudelleen rajapinnasta ja vaihdetaan sovelluksessa aktiiviseksi valittu tili.

Kuvio 9. Käytettävän tilin valinta

4.4.2 Koti

Kun käyttäjä on kirjautunut sisään sovellukseen, ohjataan hänet sovelluksen ”kotiruudulle” (ks. kuvio 10). Kotiruudulta käyttäjä pystyy seuraamaan lähetettyjen laskujen tilannetta, olivatpa ne myöhässä, perinnässä, avoimena tai luonnostilassa olevista laskuista. Teknisesti laskut noudetaan API-rajapinnasta taustalla heti sisäänkirjautumisen yhteydessä ja tallennetaan Redux-varastoon, jolloin ne pystytään käymään läpi jo ennen varsinaisen Laskut-näkymän lataamista.

Kotiruudulta pääsee myös suoraan pikavalintojen kautta siirtymään yleisimpiin toimintoihin, kuten luomaan laskua tai lisäämään uutta asiakasta. Kotiruudun alareunassa sijaitsevat myös erinäiset myyntiraportit. Kotisivusta rakennettiin hyvin pitkälle dynaaminen. Eri valinnat vaihtelevat haetun datan ja käyttäjällä olevan palvelupaketin mukaan. Kotiruudulla näytetään myös tarvittaessa palveluun luotuja ilmoitusviestejä, esimerkiksi tietoa tulevasta päivityksestä.

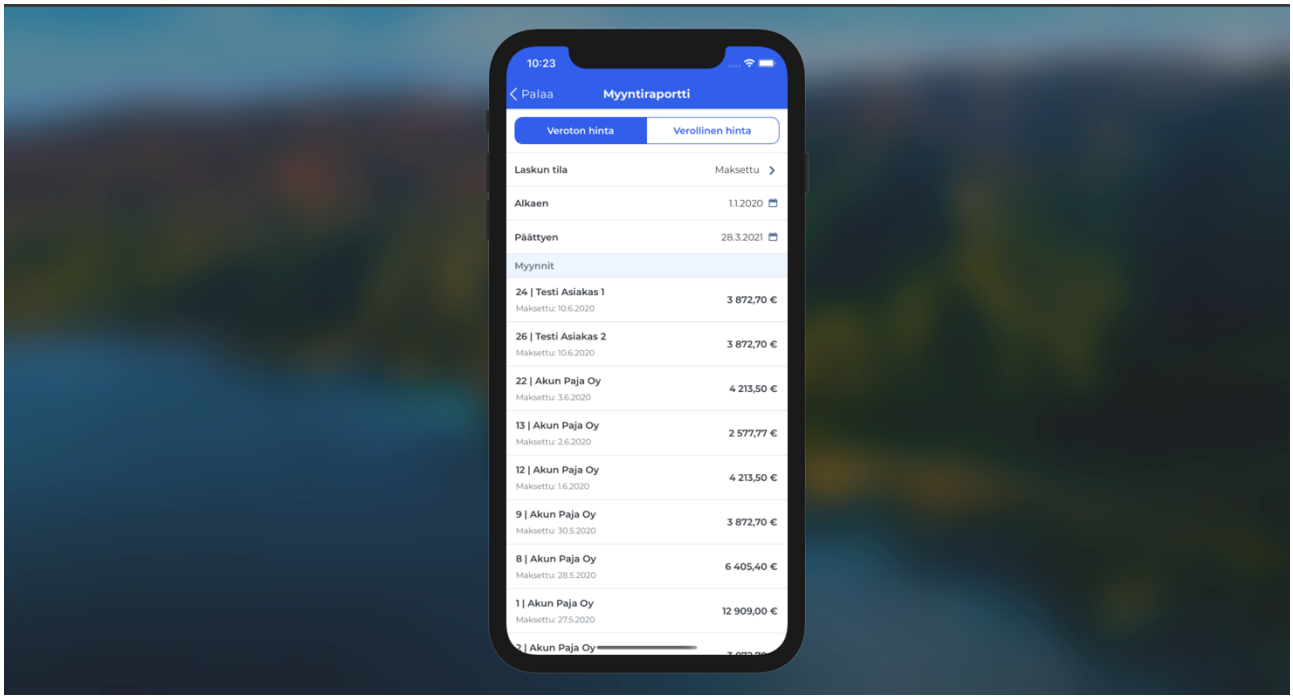


Kuvio 10. Sovelluksen ensimmäinen sivu

4.4.3 Raportit

Sovellukseen toteutettiin näyttäminen neljälle erilaiselle raportille: myyntiraportti, myynnit kuukausittain, myynnit tuotteittain ja myynnit asiakkaittain. Raportit muodostetaan palvelimen päässä, joten varsinaista laskentalogiikkaa sovellukseen ei tarvinnut toteuttaa raportoinnin osalta. Raporttien yhteydessä on eri määrityksiä raporttien hakuun muodostettavaa kyselyä varten.

Raportit saa rajattua haluamiensa päivämäärien sisälle, näytettyä verollisen tai verottoman hinnan mukaan tai rajattua huomioimaan ainoastaan avoimena tai maksettuna olevat laskut (ks. kuvio 11). Päivämäärä valitaan automaattisesti alkamaan vuoden alusta ja päättymään latausheteellä olevaan päivämäärään. Useimmissa tapauksissa em. seikat tekevät sovelluksen käyttämisestä huomattavasti vaivattomampaa.

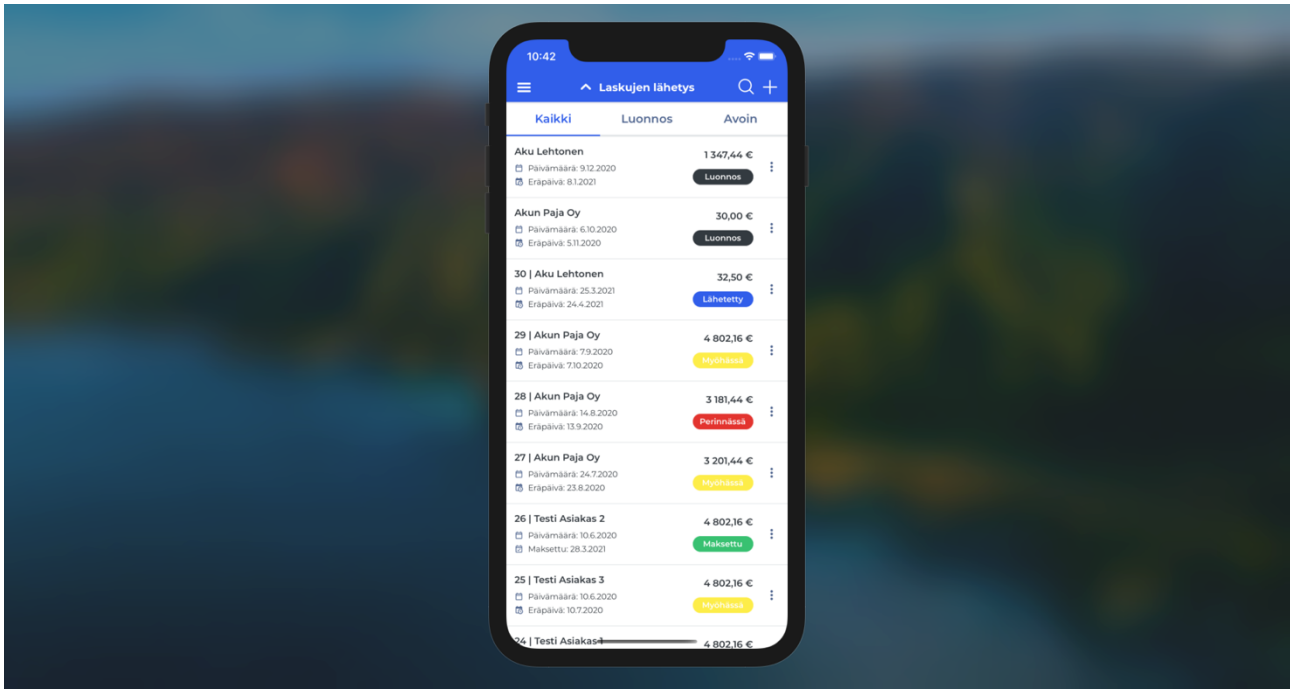


Kuvio 11. Myyntiraportti maksetuista laskuista

4.4.4 Laskutus

Laskutusnäkymässä listataan kaikki palvelussa luodut laskut (ks. kuvio 12) sekä saapuneet ostolaskut, mikäli käyttäjällä on saapuneiden laskujen ominaisuus käytössään. Laskuja saa myös haettua asiakkaan nimen perusteella, laskun numeron perusteella tai laskulta löytyvien tuotteiden nimien perusteella. Laskut saa myös jaettua niin että listassa näytetään joko kaikki, luonnokset, avoimet, maksetut tai arkistoidut laskut.

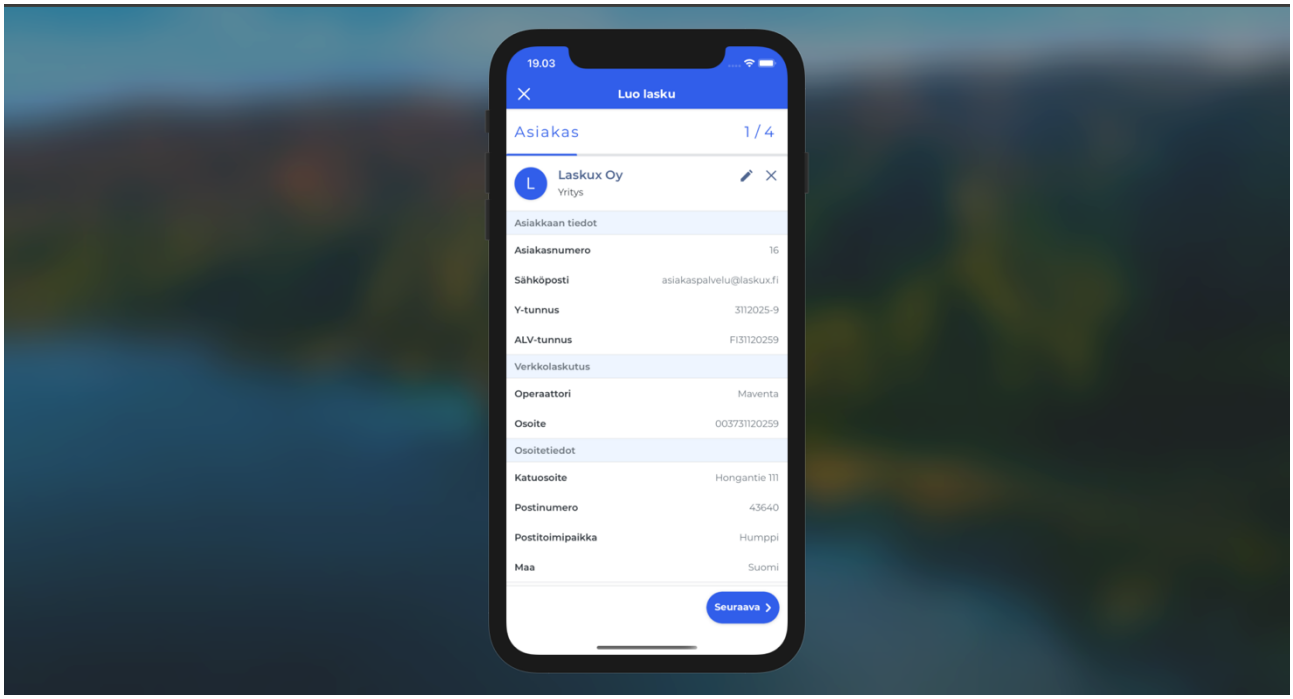
Painettaessa laskua listauksesta, laskuille voi suorittaa erilaisia toimintoja, kuten merkitä laskuja maksetuksi, lähettää laskuja perintään, muokata ja katsella laskuja sekä lähettää hyvityslaskuja. Valinta eri saatavilla olevista toiminnoista rakennettiin myös toimimaan dynaamisesti, eli valinnoissa näytetään vain sellaisia toimintoja, joita valitulle laskulle on mahdollista suorittaa. Esimerkiksi lähetettyä laskua ei ole mahdollista enää muokata, eikä hyvityslaskua hyvittää uudelleen.



Kuvio 12. Laskut-näkymä

Laskun luontiprosessista tehtiin nelivaiheinen. Ensimmäisessä vaiheessa laskulle valitaan asiakas (ks. kuvio 13). Asiakkaan pystyy valitsemaan joko asiakasrekisteristä tai luomaan kokonaan uuden asiakkaan käyttäen normaalia asiakkaan luontiprosessia. Mikäli uusi asiakas luodaan laskutusvaiheessa, asiakas tallennetaan asiakasrekisteriin ja valitaan automaattisesti asiakkaaksi laskulle. Laskua luodessa näkymän yläreunassa näkyy vaiheet 1-4, jolloin käyttäjän on luontevampaa seurata missä vaiheessa prosessia hän on menossa.

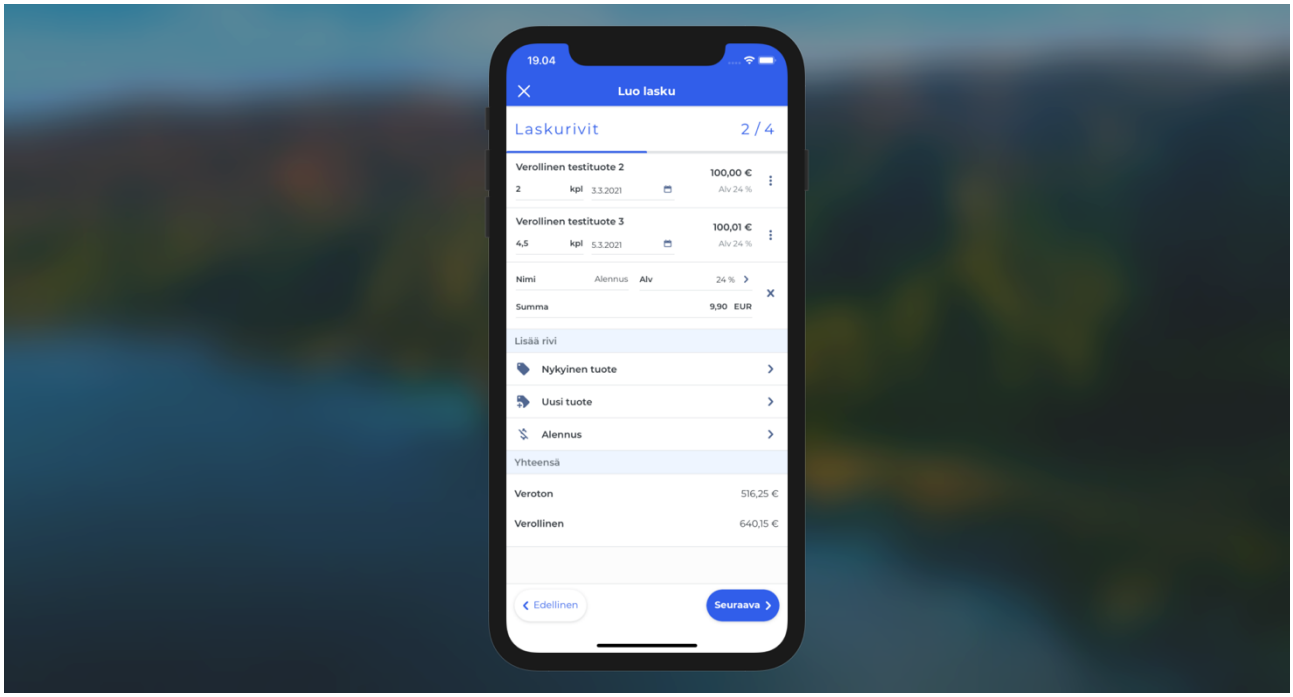
Asiakkaan valintaan toteutettiin myös mahdollisuus tehdä valinta suoraan asiakasrekisterin listauksesta. Käyttäjä voi siis etsiä haluamansa asiakkaan hyödyntäen esimerkiksi asiakasrekisterissä saatavilla olevia suodattimia ja valita laskutettavan asiakkaan suoraan listasta. Tämä on myös hyvä esimerkki pienestä yksityiskohdasta, joka voi tuntua mitättömältä mutta osoittautua joissain tilanteissa erittäin käteväksi lisäominaisuudeksi.



Kuvio 13. Asiakkaan valinta laskulle

Toisessa vaiheessa laskulle valitaan tuoterivit (ks. kuvio 14). Tuoterivit voidaan valita samaan tapaan tuoterekisteristä kuin asiakkaatkin, tai luoda uudet normaalia tuotteen luontiprosessia hyödyntäen, jolloin tuotteet tallennetaan tuoterekisteriin ja lisätään automaattisesti laskun riveiksi. Tuotteelle syötetään aina määrä. Tarvittaessa riville voidaan lisätä myös päivämäärä- ja kuvauskentät, jotka tulevat näkymään laskun tulosteella. Normaalien tuoterivien lisäksi laskulle voidaan lisätä alennusrivejä, jotka voidaan nimetä vapaavalintaisesti, jonka lisäksi niille valitaan sovellettava verokanta ja summa. Tuoterivien alle lasketaan reaaliajassa laskun veroton ja verollinen loppusumma.

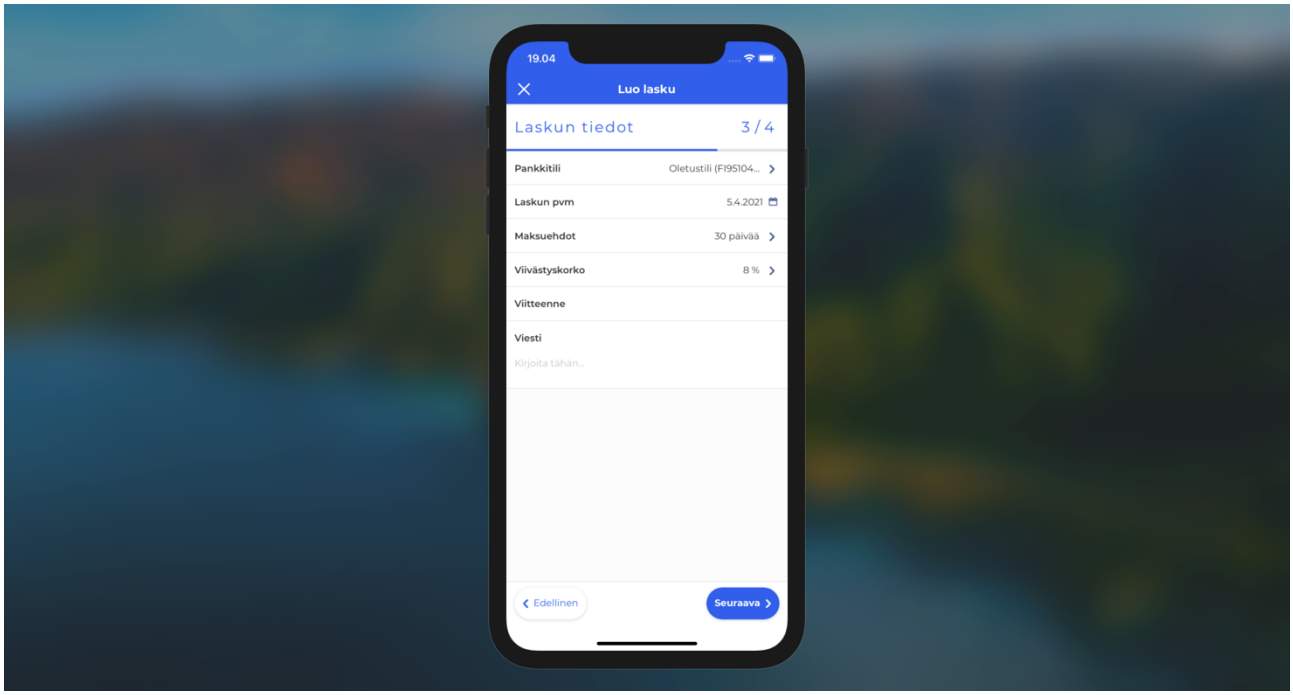
Laskurivien lisäämisessä on lisäksi dynaamisina elementteinä kentät ALV 0 % -verokohtelukoodille ja selitteelle. Kentät ilmestyvät näkyviin, mikäli laskulle valitaan tai luodaan rivi, jonka arvolisäverosprosentti on 0 %. Kenttien täyttäminen myyntilaskuille on lakisääteistä laskutettaessa ilman arvolisäveroa.



Kuvio 14. Tuoterivien lisääminen laskulle

Kolmannessa vaiheessa laskulle syötetään muita laskulle olennaisia tietoja, kuten saajan pankkitili, päivämäärä, maksuehdot ja viivästyskorko (ks. kuvio 15). Sovelluksessa pystytään valitsemaan myös käyttäjäkohtaisista asetuksista oletusarvot esimerkiksi sovellettavista maksuehdoista tai viivästyskorosta, koska useimmiten käyttäjät valitsevat toistuvasti samat arvot. Automaattinen täyttäminen myös helpottaa ja nopeuttaa laskutusprosessia.

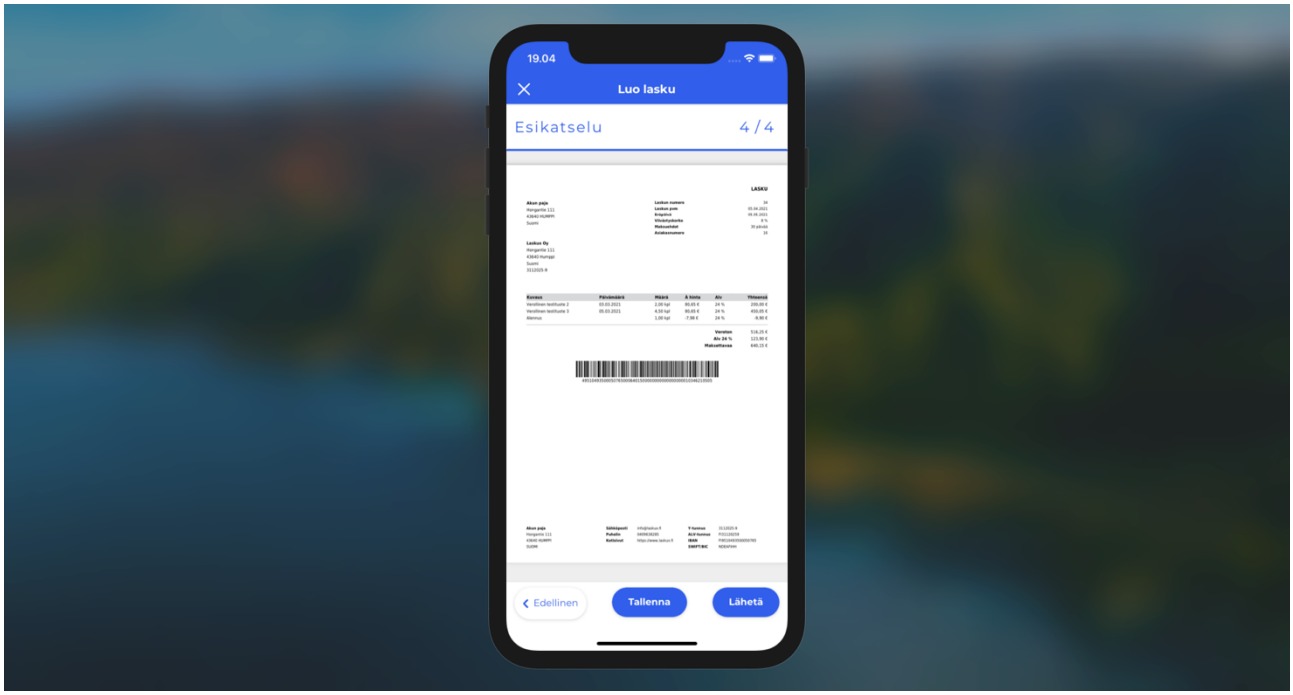
Myös tässä vaiheessa osa kentistä on dynaamisia. Esimerkiksi pankkitilien valinta näytetään vain, mikäli käyttäjä on lisännyt tietoihinsa useamman kuin yhden pankkitilin. Toinen piilossa oleva kenttä on laskun numero. Käyttäjä voi valita palvelun asetuksista laskun numeroinnin manuaaliseksi, mikäli hän ei jostain syystä halua käyttää automaattista laskun numerointia.



Kuvio 15. Laskun tietojen täyttäminen

Neljännessä vaiheessa näkymään liitetään laskun esikatselu (ks. kuvio 16). Aiemmissa vaiheissa syötetyt tiedot lähetetään palvelimelle, palvelin muodostaa tietojen perusteella PDF:n ja palauttaa sen base64-enkoodattuna sovellukseen. PDF:n näyttämiseen on käytetty 3. osapuolen kirjastoa nimeltään react-native-pdf. Kirjasto tarjoaa komponentin, joka voidaan liittää näkymään ja sille syötetään parametrinä base64-enkoodattu merkkijono generoidusta PDF-tiedostosta. Laskun kuvaa pystyy myös liikuttelemaan ja zoomaamaan näkymän sisällä. Neljännessä vaiheessa vaihtuvat myös ruudun alareunassa näkyvät painikkeet. Tässä vaiheessa laskun voi joko tallentaa luonnokseksi, jolloin sitä voi palata muokkaamaan myöhemmin, tai siirtyä lähettämään laskua vastaanottajalle.

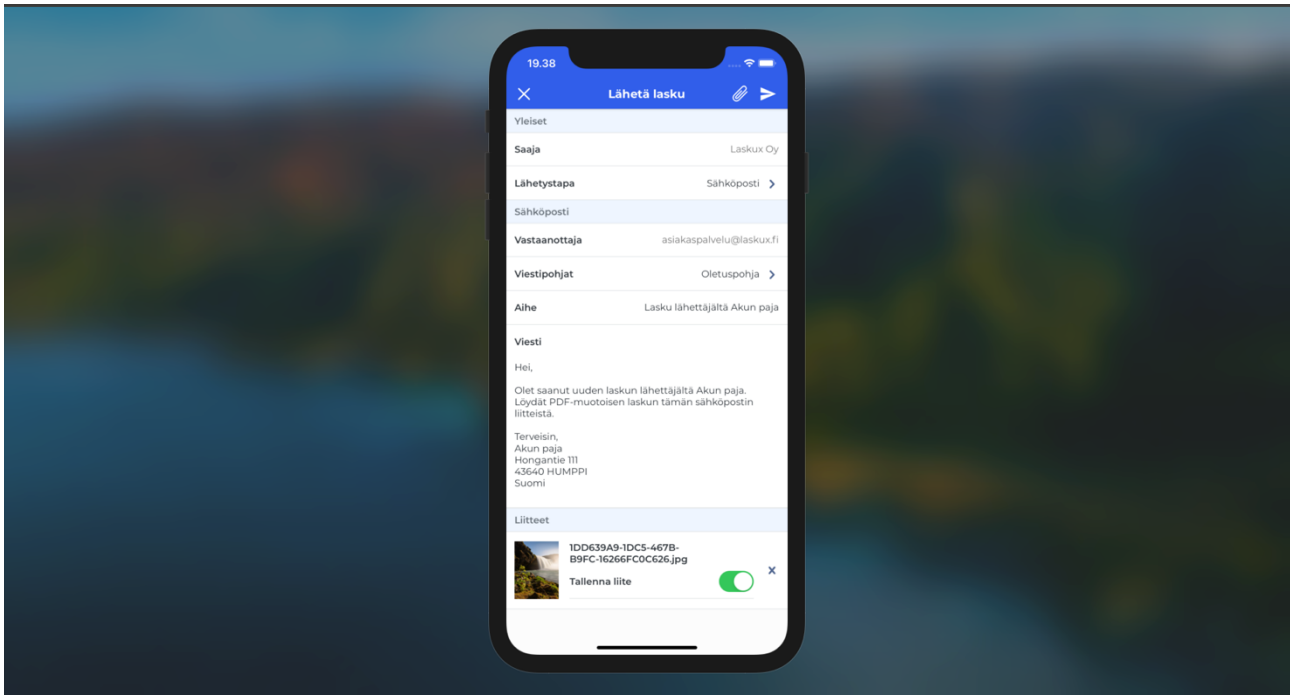
Esikatselua luotaessa lasku lähetetään ensimmäistä kertaa palvelimen päähän. Mikäli laskun tiedoissa on virheitä, palautetaan käyttäjä siihen vaiheeseen, jossa virhe on esiintynyt. Jos käyttäjä on esimerkiksi syöttänyt virheellisen alennussumman laskuriveille, palautetaan hänet takaisin laskun luomisen toiseen vaiheeseen ja näytetään virheilmoitus.



Kuvio 16. Laskun PDF:n esikatselu

Mikäli laskun esikatselusta painetaan Lähetä-painiketta, siirrytään kokonaan uuteen ruutuun. Lähetysruudussa valitaan laskulle lähetystapa (sähköpostilasku, verkkolasku tai paperilasku). Lähetyksen yhteyteen voidaan myös lisätä mahdollisia tiedostoliitteitä joko kamerasta, galleriasta tai aiemmin tallennetuista liitteistä (ks. kuvio 17). Näkymä muuttuu dynaamisesti. Käytettäessä sähköpostia laskun lähetystapana, tulee näkyville myös kentät sähköpostiviestin aiheelle ja viestille. Järjestelmään voidaan lisätä valmiita viestipohjia, jolloin aiheen ja viestin pystyy valitsemaan ja täyttämään automaattisesti, jolloin niitä ei tarvitse kirjoittaa jokaisella kerralla uudelleen. Kun lasku on onnistuneesti lähetetty, palataan takaisin laskujen listausnäkymään.

Laskun lähetyksen yhteydessä hyödynnetään react-native-image-crop-picker -kirjastoa, jonka avulla päästään käsiksi laitteen galleriaan ja kameran käyttöön. Kirjaston avulla voidaan valita kuvia galleriasta tai kuvata kameralla, jonka jälkeen kuvat palautetaan base64-enkoodattuna käytettäväksi koodissa, tai tallennetaan laitteeseen. Kirjaston käyttö helpottaa ja nopeuttaa kehitystyötä reilusti, kun kirjasto huolehtii natiivista sovelluslogiikasta.

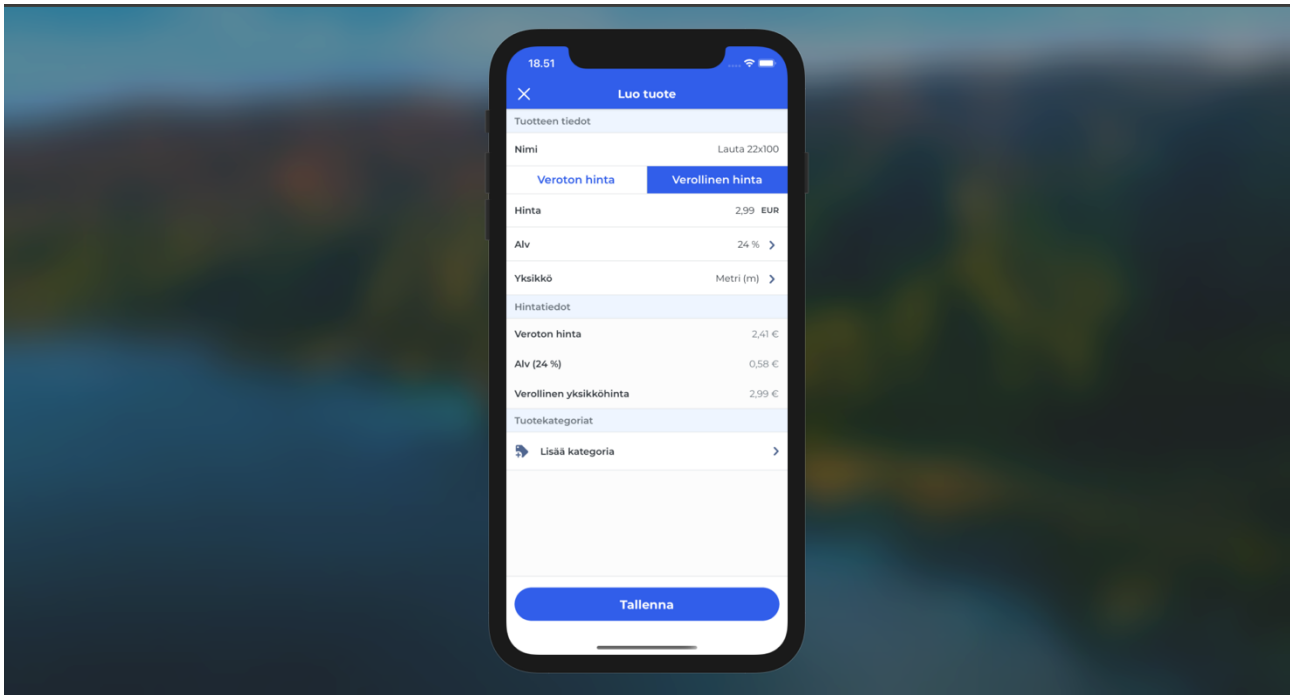


Kuvio 17. Laskun lähetys sähköpostilla mukanaan valokuvaliite

4.4.5 Tuotteet

Tuoterekisteri helpottaa käyttäjää tilanteissa, joissa samaa tuotetta laskutetaan toistuvasti. Tuotteen voi luoda laskutusvaiheessa, tai erillisen tuoterekisterin kautta. Tuotteelle syötetään perustietoina nimi, yksikkö, verokanta ja hinta. Käyttäjä voi valittaa tuotteen luontinäkymässä, syöttääkö hän tuotteelle verollisia vai verottomia hintoja, jolloin käyttäjän ei tarvitse itse laskea hintaa ja laskenta menee varmasti oikein (ks. kuvio 18).

Joissain tilanteissa hinnan laskentatapa vaikuttaa laskennan tulokseen, joten käyttäjän on itse tehtävä valinta hinnan laskentatavasta. Esimerkiksi, kun verollinen hinta 24 % verokannalla lasketaan verottomasta hinnasta 80,65 €, tulee verolliseksi hinnaksi 100,01 €. Taasen vastaavasti veroton hinta 24 % verokannalla lasketaan verollisesta hinnasta 100,00 €, tulee verottomaksi hinnaksi myös 80,65 €. Näin ollen ei voida automaattisesti olettaa laskennan menevän oikein, ilman että käyttäjä tekee itse valintaa hintojen syöttötavasta.

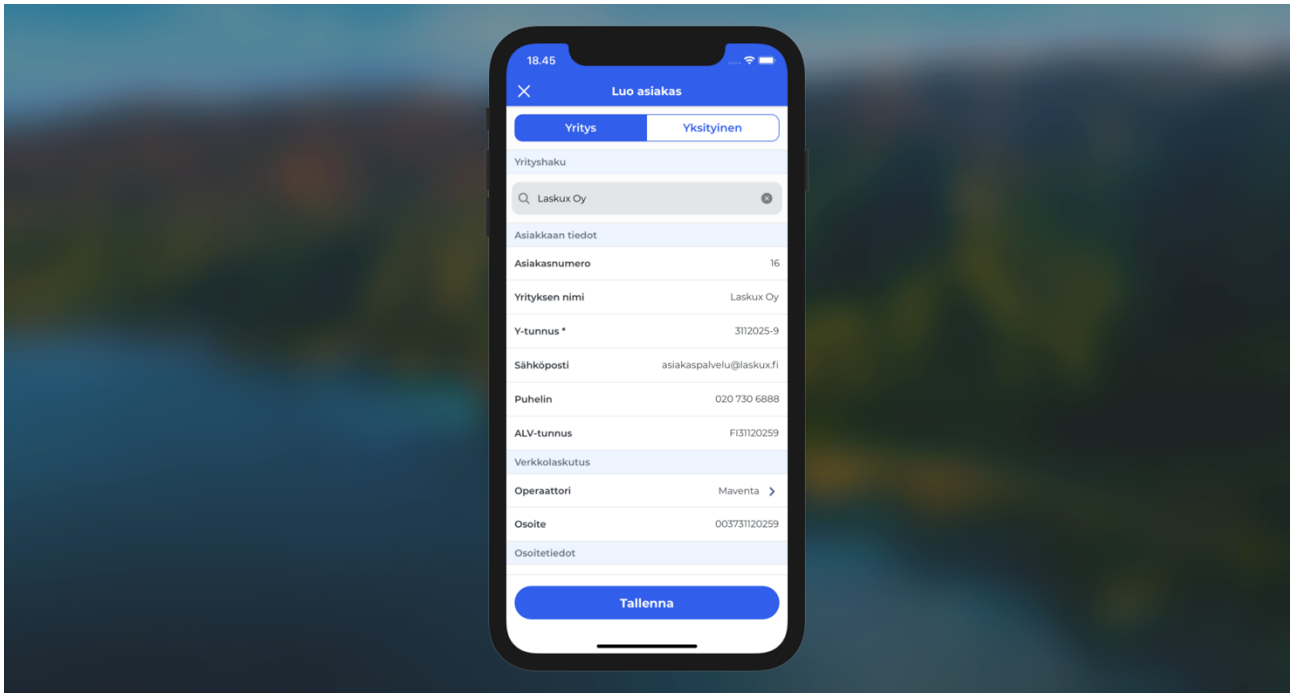


Kuvio 18. Tuotteen luonti

4.4.6 Asiakkaat

Asiakasnäkymässä uuden asiakkaan luonnin helpottamiseksi on tehty hakutoiminto, jonka avulla pystytään tekemään hakuja YTJ (Yritys- ja yhteisötietojärjestelmä) rekisteriin (ks. kuvio 19). Haku toimii joko yrityksen nimellä tai y-tunnuksella. Mikäli yrityksiä löytyy enemmän kuin yksi, näytetään käyttäjälle listaus hakutuloksista ja käyttäjä saa itse valita oikean vaihtoehdon. Kun yrityksen tiedot on löydetty, ne täytetään automaattisesti asiakkaan luontilomakkeelle. Tiedot sisältävät kaikki yrityksen keskeiset tiedot yhteystiedoista osoitetietoihin sekä verkkolaskutustietoihin. Toiminto nopeuttaa uuden asiakkaan luontiprosessia merkittävästi. Se myös poistaa mahdollisuuden näppäilyvirheiltä esimerkiksi verkkolaskutusosoitetta syötettäessä.

Asiakasta luotaessa ruudulla näytetään alkutilanteessa vain muutama kenttä, jotka ovat pakollisia. Asiakkaan pystyy siis luomaan tarvittaessa nopeasti, ilman että käyttäjältä vaaditaan kaikkien tietojen täyttämistä. Tämä ominaisuus voi olla hyödyllinen, kun ottaa huomioon toimeksiantajan toiveet sovelluksen helppokäyttöisyydestä nopeissakin tilanteissa.

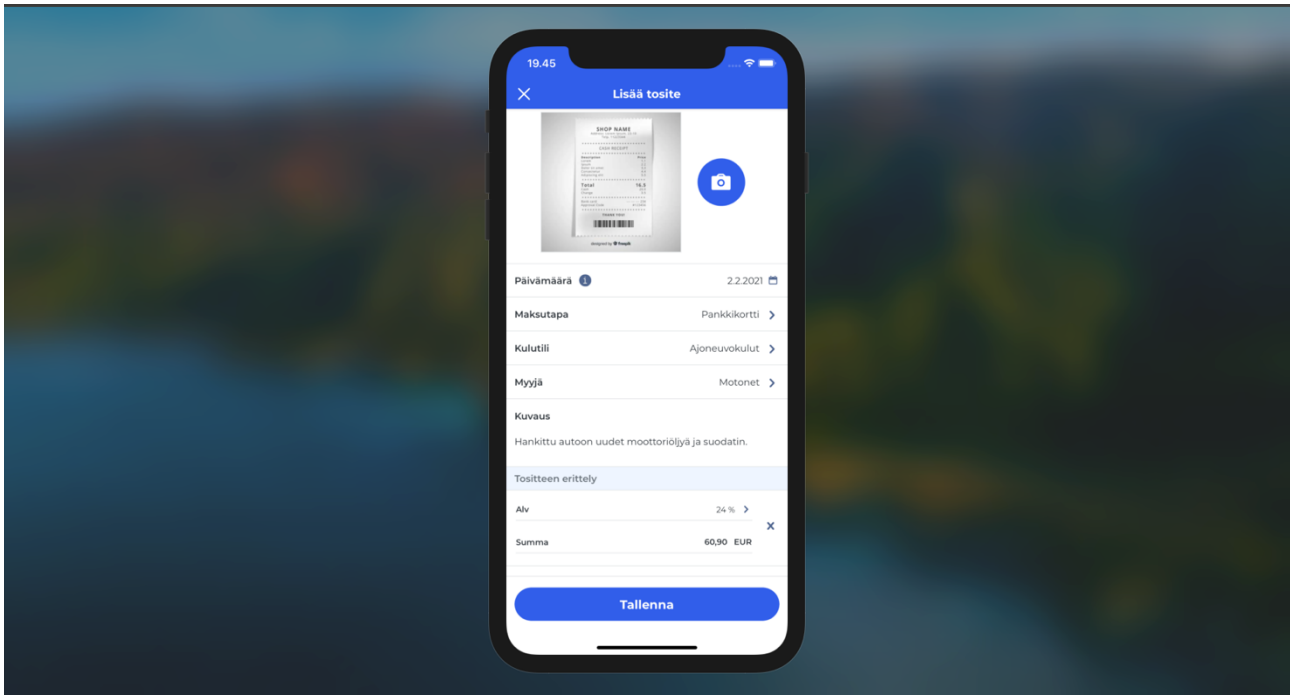


Kuvio 19. Asiakkaan luonti

4.4.7 Tositteet

Tositteiden lisäys on erityisesti mobiilisovellukseen sopiva ominaisuus, koska useimmiten tositteet tallennetaan järjestelmään valokuvan kanssa. Tällöin mobiililaitteen kameraa voidaan hyödyntää kuvan ottamiseen. Tositteiden luontinäkymässä voidaan lisätä tositteeseen liittyviä kuvia joko yksi tai useampi. Tositteelle lisätään päivämäärä, maksutapa, kulutili, myyjä ja kuvaus (ks. kuvio 20). Näiden lisäksi tositteelta merkitään erittely loppusummista. Erittelyyn voidaan valita useita eri verokantoja. Laitteen kameran ja gallerian käyttämiseen on hyödynnetty samaa react-native-image-crop-picker -kirjastoa kuin laskun liitteiden valintaan.

Tositteiden lisäämiseen hyviä jatkokehitysajatuksia voisi olla mahdollinen tositteen kuvan skannaus palvelimen päässä. Myös tämä huomioitiin ominaisuutta toteuttaessa. Kuva lähetetään rajapintaan ennen varsinaisen tositteen tallentamista, jolloin rajapinta palauttaa tositteen kentät, jotka täytetään sovelluksen kenttiin. Eli mikäli skannaus lisätään palvelimen päähän, ei se vaadi sovelluksesta erillisiä toimia.

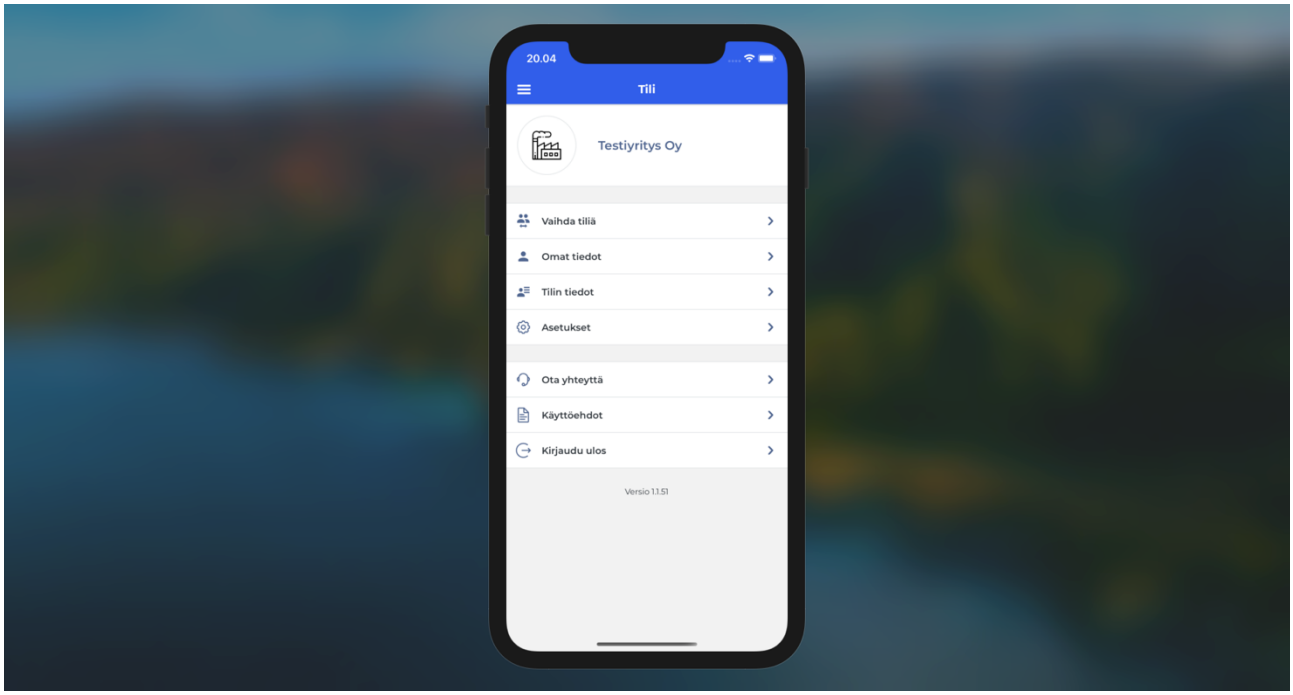


Kuvio 20. Tositteen luonti

4.4.8 Tili ja asetukset

Käyttäjä pystyy hallitsemaan useimpia sovelluksen käytön kannalta tärkeitä palvelun asetuksia Tili-näkymän kautta (ks. kuvio 21). Käyttäjä pystyy vaihtamaan aktiiviseksi valittua käyttäjää, mikäli hänellä on useampia eri käyttäjiä saman käyttäjätilin alla, käytännössä siis muuttamaan kirjautumisen yhteydessä tehtyä valintaa aktiivisesta käyttäjästä. Käyttäjä voi hallita omia tietojaan, vaihtaa salasanan tai muuttaa laskulla näkyviä tilin tietoja. Myös yrityksen logon lisääminen ja vaihtaminen onnistuu tilin tietojen kautta, logo tulostetaan lähtevien laskujen yläkulmaan.

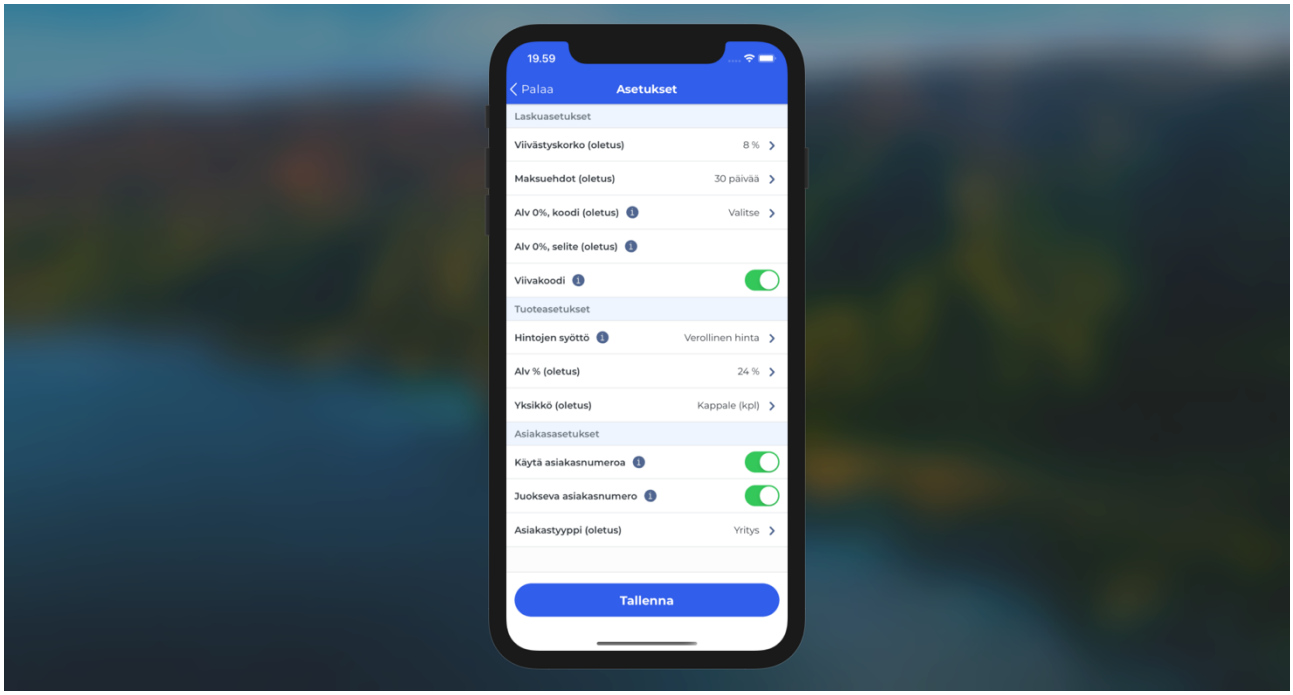
Tili -ruudun kautta käyttäjä voi myös kirjautua ulos sovelluksesta, siirtyä palvelun verkkosivustolle tarkistamaan yleiset käyttöehdot, tarkistaa asennetun sovellusversion. Käyttäjä voi myös olla yhteydessä asiakaspalveluun joko puhelimitse, sähköpostilla tai sovellukseen rakennettua yhteydenottolomaketta hyödyntäen.



Kuvio 21. Oma tili

Asetuksista voidaan hallita useimpia sovelluksen käytettävyyden kannalta olennaisia asetuksia liittyen laskutus-, tuote-, ja asiakasasetuksiin (ks. kuvio 22). Asetuksissa on hyödynnetty useiden kenttien kohdalla kolmannen osapuolen react-native-elements kirjaston tarjoamaa tooltip-komponenttia. Tooltipeillä voidaan liittää kentän nimen yhteyteen koskettamalla ilmestyviä tekstikenttiä, joihin voidaan syöttää ohjeistuksia kenttään liittyen, näin ollen itse kenttien nimet voidaan pitää lyhyinä ja syötettävälle tekstille jää enemmän tilaa.

Asetukset haetaan rajapinnasta ”tilin tiedot"-kyselyllä ja tallennetaan Redux-varastoon jo heti sovelluksen avautuessa. Kun tietoja päivitetään, haetaan tiedot uudelleen ja tallennetaan ne Redux-varastoon. Näin tiedot saadaan pysymään ajan tasalla, koska asetukset vaikuttavat sovelluksen käyttöön useimmissa paikoissa, esimerkiksi kenttien automaattisessa täytössä.

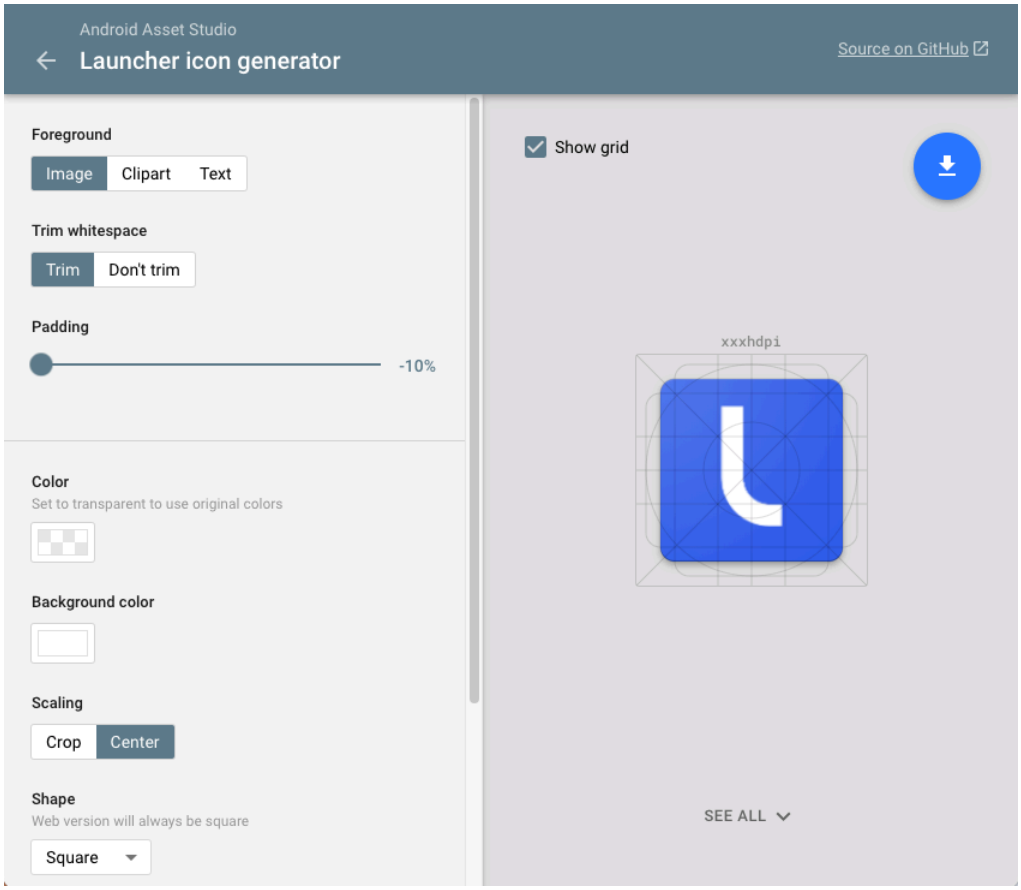


Kuvio 22. Sovelluksen käyttöön liittyviä asetuksia

4.5 Julkaisu

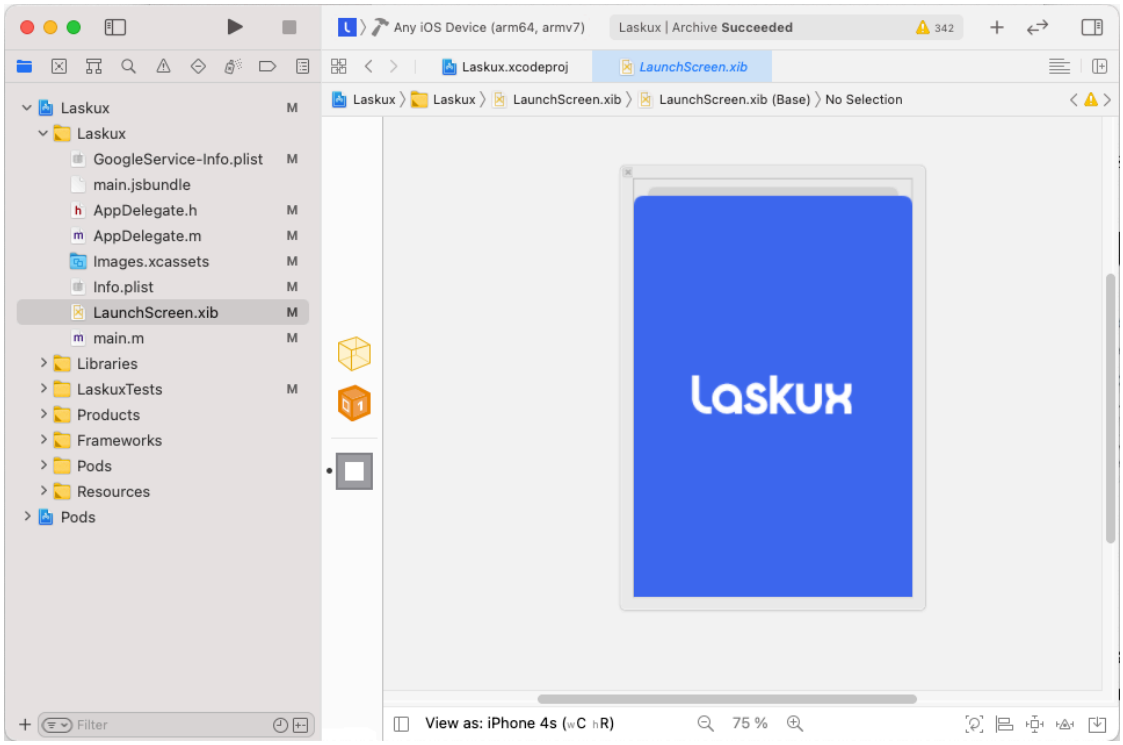
Sovelluksen julkaisu aloitettiin luomalla käyttäjätunnukset alustoille, johon sovellus oli tarkoitus julkaista. Opinnäytetyössä toteutettava sovellus julkaistiin iOS- ja Android-alustoille, joten kehittäjätunnukset tarvittiin App Store- ja Google Play -sovelluskauppoihin. Kun tunnukset oli luotu, alettiin valmistelemaan tuotantokohtaisia julkaisuja.

Sovelluksen julkaisu edellytti ikonin luomista, joka näkyisi laitteen sovellusvalikossa sekä sovelluskaupassa. Ikonien luontiin on olemassa valmiita työkaluja, joille syötetään haluttu kuva ja sen perusteella generoidaan laitekohtaiset ikonit. Hyvä työkalu iOS:lle sopivien ikonien luontiin on Icon Set Creator. Vastaava työkalu Androidille on Android Asset Studio (ks. kuvio 23). Kummatkin olivat ilmaisia käyttää ja tarjosivat hyvät välineet ikonien luontiin.



Kuvio 23. Ikonin luonti Androidille

Ikonin lisäksi sovellukselle tarvittiin aloitusruutu (eng. launch screen), joka olisi näkyvillä aina sovelluksen käynnistyessä ennen kuin sovellus on ehtinyt latautua kokonaan. iOS:lle aloitusruudun pystyi generoimaan suoraan Xcodessa (ks. kuvio 24), jossa hallitaan varsinaista iOS projektia. Androidille aloitusruudun generointi onnistui verkosta löytyvällä Ape Tools työkalulla, joka oli käytettävissä ilmaiseksi. Generoidut kuvakkeet siirrettiin projektin assets-kansioon, josta ne haetaan ja liitetään sovellukseen kääntämisen yhteydessä.



Kuvio 24. Aloitusruudun luonti iOS:lle Xcodessa

Kun ikonit ja aloitusruutu oli lisätty, määritettiin sovellukselle vielä versionumero ja bundle-tunniste. Bundle-tunniste on vapaamuotoinen, mutta yleisten käytänteiden mukaan tunnisteena käytetään usein käänteistä domain nimeämistapaa (eng. reverse domain name notation), jossa tunniste muodostetaan verkkodomainista lukemalla se oikealta vasemmalle. Tässä tapauksessa tunnisteeksi asetettiin domain ja lisättiin siihen sana mobile, eli fi.laskux.mobile. Kun tiedot oli annettu, voitiin kääntää sovelluskauppaan ladattavat versiot. iOS:lle tuotantoon vietävän paketin kääntäminen onnistui suoraan XCodessa, Androidin paketti muodostettiin komennolla.

```
./gradlew bundleRelease
```

5 Tulokset ja pohdinta

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa toimiva ja markkinoille valmis mobiilisovellus, joka täyttäisi sille asetetut vaatimukset. Henkilökohtaisena tavoitteena oli oppia enemmän mobiilisovelluskehityksestä ja ohjelmistokehityksestä yleisesti. Tavoitteena oli myös vertailla eri sovellusriippumattomaan mobiilikehitykseen käytettäviä teknologioita vaihtoehtoina React Nativelle. React Nativen valintaan käytettäväksi teknologiaksi oltiin lopulta tyytyväisiä niin toimeksiantajan kuin työn toteuttajan puolesta. React Nativen kanssa oli helppoa lähteä työskentelemään, etenkin kun omasi aiempaa web-kehitys taustaa. React Nativen käyttämä JSX-syntaksi muistutti rakenteeltaan HTML-merkintäkieltä. Tyylittelyt tehtiin JSX-syntaksin mukaisilla CSS-tyyliohjeilla. Työn aikana syvennettiin osaamista React Nativeen ja kerrytettiin paljon uutta ymmärrystä koko mobiilisovelluksen kehitys- ja julkaisuprosessista. Lisäksi saatiin kartoitettua eri cross-platform teknologioiden hyviä ja huonoja puolia sekä tehtyä vertailua mikä sopii parhaiten tämän tyyppisen sovelluksen toteuttamiseen.

Opinnäytetyön aihetta rajattiin tutkimuskysymyksien avulla ja kaikkiin kysymyksiin saatiin vastattua. Vertailuun valituista teknologioista pystyttiin melko kattavasti perustelemaan, miksi käytettäväksi teknologiaksi valittiin juuri React Native. Samalla kartoitettiin, minkälaisissa tilanteissa muut vertailtavista teknologioista saattaisivat olla sopivampi valinta. Työtä tehdessä opittiin myös se, että suunniteltaessa mobiilisovellusta on syytä huomioida tai vähintäänkin tiedostaa sovelluksen skaalautuminen. Alkuvaiheessa useissa kohtaa on helppoa oikoa ja tehdä ratkaisuja, joita joudutaan melko pian sovelluksen laajentuessa muuttamaan, koska ne ovat joko epäsopivia tai hankalasti ylläpidettävä. Pienissäkin projekteissa kannattaa lähtökohtaisesti ajatella laaja-alaisesti, jolloin vältetään yllätykset, kun vaatimukset sovellukselle alkavat kasvaa.

Kehitystyön päätökseen saattaminen vaati paljon työtä ja useiden asioiden valmiiksi saaminen oli usein yrityksen ja erehdyksen kautta saavutettua. Useammin kuin kerran lähdettiin rakentamaan toteutusta pidemmän kautta, kunnes myöhemmin löytyi parempi tapa tehdä sama asia reilusti vähemmällä määrällä koodia. Refaktorointia eli koodin siivoamista ja organisointia tuli siis tehtyä moneen otteeseen. Yksin sovellusta kehittäessä oli se hyvä puoli, että työtä tehdessä pysyi jatkuvasti tilanteen tasalla sovelluksen eri osista ja hyvin laajaakin refaktorointia oli helppoa toteuttaa tarvittaessa. Yksin koodatessa myös joutui isompaan vastuuseen, koska päätökset teknisistä toteutuksista joutui tekemään itse, mikä oli kuitenkin samalla erittäin opettavaista.

Lopputuloksena työstä syntyi sovellus, joka sisältää kaikki ne keskeiset ominaisuudet, joita modernilta laskutussovellukselta vähintään vaaditaan. Sovelluksessa onnistuu niin laskujen luonti kuin vastaanotto, asiakasrekisterin hallinta, tuoterekisterin hallinta sekä tositteiden lisääminen järjestelmään. Erityisen hyödylliseksi ominaisuudeksi sovelluksessa osoittautui uusien asiakkaiden lisääminen YTJ:stä haettavien tietojen perusteella, joka vähentää merkittävästi käyttäjältä vaadittujen kenttien syöttämistä ja nopeuttaa näin sovelluksen käyttöä. Myös keskeisimmät asetukset löytyvät sovelluksesta, joten käyttäjältä ei aina edellytetä kirjautumista selainversioon, mikäli asetuksia tulee tarve muuttaa. Koska sovellukseen toteutettiin Redux-tilanhallinta, on sovellus huomattavasti skaalautuvampi jatkokehityksen kannalta, kuin se olisi ilman Reduxia. Myös koodin rakenne on pyritty tekemään hyvien käytänteiden mukaisesti, niin että eri sovelluksen osat olisivat mahdollisimman hyvin eroteltu omiksi osikseen, jolloin riippuvuuksia eri koodipohjan osien välillä ei ole päässyt syntymään kovin paljon.

Jatkokehityksen paikkoja jäi ainakin sovelluksen visuaalisessa puolessa ja siinä, kuinka sovellusta voitaisiin kehittää yhä nykyaikaisemman näköiseksi. Visuaalinen ilme onnistuttiin toteuttamaan kuitenkin siihen nähden hyvin, ettei sen suunnittelemiseen osallistunut graafista suunnittelijaa tai muuta erityisesti visuaalisen puolen ammattilaista. Myös joitain yleisiä prosesseja kuten salasanan nollaamista sovelluksessa olisi yhä mahdollista kehittää käyttäjäystävällisemmäksi. Työtä tehdessä opittiin, että toimivan ja käytettävän sovelluksen kehittäminen ei ole helppoa. Sovelluksen kehittäminen koostuu useista yksityiskohdista, joita on mietittävä ja testattava ahkerasti, jotta saavutetaan viimeistellyn tuntuinen lopputulos. Sovellus vastasi kuitenkin toimeksiantajan toiveita odotetusti.

Lähteet

Abramov, D. 2020. Reduxin virallinen verkkosivusto. Nettisivu. Viitattu 25.01.2021. https://redux.js.org/.

Altexsoft. 2020. The Good and The Bad of Xamarin Mobile Development. Artikkeli. Viitattu 20.02.2021. https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native/.

Babich, N. 2020. Basic Patterns for Mobile Navigation. Artikkeli. Viitattu 20.3.2021. https://www.smashingmagazine.com/2017/05/basic-patterns-mobile-navigation/.

Banto, O. 2020. Build Your Mobile App with Ionic 5. Artikkeli. A Medium Corporation. Viitattu 20.2.2021. https://medium.com/weekly-webtips/build-your-mobile-app-with-ionic5-getting-started-c901b8769cd3.

Facebook. 2020. React-Nativen virallinen verkkosivusto. Nettisivu. Viitattu 27.01.2021. https://reactnative.dev/.

Knyga, O. & Hayat, S. 2017. React Native vs Real Native Apps. Artikkeli. Codeburst. Viitattu 24.01.2021. https://codeburst.io/react-native-vs-realnative-apps-ad890986f1f.

Krol, O. 2019. Why (not) Choose Flutter for Mobile App Development. Artikkeli. Viitattu 18.02.2021. https://stfalcon.com/en/blog/post/flutter-mobile-app-development.https://dzone.com/articles/the-good-and-the-bad-of-ionic-mobile-development.

Lobastov, I. 2019. The Good and the Bad of Ionic Mobile Development. Artikkeli. Viitattu 20.02.2021. https://dzone.com/articles/the-good-and-the-bad-of-ionic-mobile-development.

Martin, S. 2020. React Native vs. Flutter vs. Ionic. Artikkeli. A Medium Corporation. Viitattu 19.02.2021. https://medium.com/better-programming/react-native-vs-flutter-vs-ionic46d3350f96ee.

Muchow, J. 2020. Cross-Platform Technologies Compared — React Native and Flutter in 2020. Artikkeli. A Medium Corporation. Viitattu 24.01.2021. https://medium.com/quark-works/cross-platform-technologies-compared-react-native-and-flutter-in-2020-71cd6cb60a58.

React-Navigation. 2020. React-Navigationin virallinen verkkosivusto. Nettisivu. Viitattu 28.3.2021. https://reactnavigation.org/docs/getting-started.

Stewart, S. 2018. Add App Icons and Launch Screen to React Native Apps (iOS & Android). Artikkeli. Viitattu 15.3.2021. https://betterprogramming.pub/react-native-add-app-icons-and-launch-screens-onto-ios-and-android-apps-3bfbc20b7d4c.

Telemedia. 2020. Marked Snapshot: Desktop and mobile internet usage in 2020. Artikkeli. Viitattu 24.3.2021. https://www.telemediaonline.co.uk/market-snapshot-desktop-and-mobile-internet-usage-in-2020/.