Elisa Moilanen

FRAMER OSANA SOVELLUSKEHITYSTÄ

Opinnäytetyö

Liiketalouden ammattikorkeakoulututkinto

Tietojenkäsittelyn koulutus

2022

South-Eastern Finland University of Applied Sciences

Tutkintonimike	Liiketalouden ammattikorkeakoulututkinto
Tekijä/Tekijät	Elisa Moilanen
Työn nimi	Framer osana sovelluskehitystä
Toimeksiantaja	Mindhive Oy
Vuosi	2022
Sivut	41 sivua, liitteitä 0 sivua
Työn ohjaaja(t)	Miia Liukkonen, Ville Venäläinen, Ulisses Camargo

TIIVISTELMÄ

Tässä opinnäytetyössä tutkittiin, että voisiko Framerilla tehtyjä prototyyppejä hyödyntää suoraan React-sovelluksessa ja siten nopeuttaa sovelluskehityksen prosessia. Framer työkaluna voisi olla toimeksiantajalle hyvä vaihtoehto siinäkin mielessä, koska sen avulla voidaan helposti suunnitella vuorovaikutteisia käyttöliittymiä. Tätä ominaisuutta toimeksiantaja pitää erittäin tärkeänä.

Tutkimusta tehdessä tutustuttiin ensin käyttöliittymäsuunnittelun teoriaan kirjallisuuden kautta. Toimeksiantajalta saatiin myös asiantuntijahaastattelu, jonka sanoma oli samankaltaista, kuin mitä kirjallisuuslähteet kertoivat. Lopuksi tehtiin vielä produktiivista tutkimusta rakentamalla Frameriin vuorovaikutteisia komponentteja ja testaamalla niiden tuomista React-sovellukseen.

Tutkimuksen tuloksena selvisi, että Framer on vielä kehitysvaiheessa, eikä komponenttien tuominen osaksi React-sovellusta ollut ongelmatonta. Selvisi myös, että vain itse ohjelmoidun komponentin tuominen onnistui molemmilla saatavilla olevista tavoista, joko koodia kopioimalla tai suoraan tuomalla komponentti Framerin import-lauseella. Havaittiin myös, että tuominen onnistui React-sovellukseen, joka käyttää Next.js-frameworkia.

Johtopäätöksenä on se, että Framer ei tällä hetkellä ehkä ole vielä täysin valmis työkalu sovelluskehityksen nopeuttamiseen ilman ohjelmointia. Sen sijaan on kuitenkin mahdollista, että ohjelmistokehittäjät loisivat Frameriin oman kirjaston, jota voisi käyttää kaikissa projekteissa pohjana ja näiden komponenttien koodeja voisi sitten kopioida suoraan React-sovellukseen. Tämän toteuttaminen vaatisi aikaa, joten pikaiseksi ratkaisuksi ohjelmoinnin nopeuttamiseen siitä ei ole, vaan mahdolliset hyödyt tulisivat ajansaatossa, kun omaa komponenttikirjastoa uudelleen käytetään.

Frameriin kohdistuva produktiivinen tutkimus on toteutettu internetistä vapaasti saatavilla olevien lähteiden opastamana. Framerille olisi ollut myös maksullinen opaskirja, josta olisi voinut ehkä saada vielä syvemmin tietoa eri ominaisuuksista. On myös huomattava, että Frameria kehitetään koko ajan, joten tämän tutkimustyön tulokset tulevat todennäköisesti vanhentumaan varsin pian uusien ominaisuuksien tullessa käyttöön. Toimeksiantaja pystyy kuitenkin miettimään jo näiden tietojen perusteella mahdollista seuraavaa askelta Framerin käytössä osana sovelluskehitystä.

Asiasanat: käyttöliittymät, suunnittelu, prototyypit, ohjelmointi

South-Eastern Finland University of Applied Sciences

Degree title	Bachelor of Business Administration
Author (authors)	Elisa Moilanen
Thesis title	Framer as part of application development
Commissioned by	Mindhive Oy
Time	2022
Pages	41 pages, 0 pages of appendices
Supervisor	Miia Liukkonen, Ville Venäläinen, Ulisses Camargo

ABSTRACT

This thesis investigated whether prototypes made with Framer could be imported directly to a React application, which could speed up the application development process. Framer could be a good option for the thesis commissioner as well, as Framer can be used to easily design interactive interfaces. This feature was especially important to the commissioner.

During the study, the theory of user interface design was first introduced through literature. An expert interview was also made with the commissioner and the message of which was similar to that reported in the literature. Finally productive study part was carried out by building interactive components into Framer, and testing their integration into the React application.

The study showed that Framer was still in the development stage, and the integration of the components into the React application was not without problems. It was also found that only the self-programmed component was successfully integrated in both available ways, either by copying the code, or by directly importing the component with the Framer import statement, but only for React application which used the Next.js Framework.

As a result, Framer may not yet be fully ready to accelerate application development without programming. Instead, it is possible for software developers to create their own library in Framer that could be used as a basis for all projects and then the codes for these components could be copied directly into React. Because implementing this would take time, it is no quick way for speeding up programming, and the potential benefits would appear over time, when the developer's own component library is reused.

This productive thesis on Framer was conducted using freely available sources on the Internet to support the work. There is also a paid-for guidebook for Framer which could still provide better information on various features. As Framer is also constantly developed, the results of this study will become obsolete quite soon as new features become available. However, based on this study, the client was already able to consider the next possible step using Framer as part of application development.

Keywords: user interfaces, design, prototypes, programming

SISÄLLYS

1 JOHDANTO

Nykyaikaiset ja tehokkaat työkalut auttavat sovelluskehityksen eri vaiheissa tuoden ajallista- ja rahallista säästöä ohjelmistoyritykselle ja asiakkaalle. Samalla nykyaikaiset työkalut tarjoavat mahdollisuuden toteuttaa ohjelmistosuunnittelua ihmislähtöisesti. Tämän opinnäytetyön aihe on lähtenyt Mindhive Oy:n tarpeista nopeuttaa ja suoraviivaistaa sovellusohjelmointia sekä pienentää käyttöliittymäsuunnittelun (eng. user interface design) sekä käyttökokemussuunnittelun (eng. user experience design) välistä kuilua suhteessa varsinaiseen sovelluksen ohjelmointiin.

Toimeksiantajayritys Mindhive Oy on pieni startup-ohjelmistoyritys, jossa kehitetään sovelluksia asiakaslähtöisesti, lähinnä JavaScript-pohjaista React-kirjastoa sekä Next.js frameworkia hyödyntämällä. Toimeksiantajalle on tärkeää, että sovellus näyttää ja tuntuu käyttäjästä hyvältä. Mindhive Oy pyrkii kehittämään sovelluskehityksen prosessejaan jatkuvasti tehokkaammiksi ja paremmiksi.

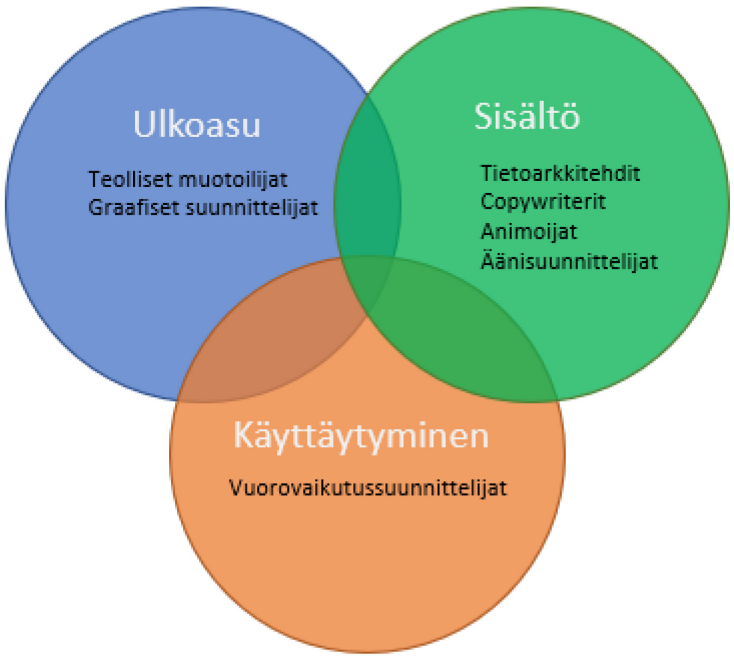
Opinnäytetyössä tutkitaan Framer-työkalua osana käyttöliittymäsuunnittelua ja sovelluskehitystä. Toimeksiantajan ajatuksena on, että Framerilla tehdyt komponentit voitaisiin tuoda suoraan kehitettävän sovelluksen React-komponenteiksi. On myös tärkeää, että tuoduissa komponenteissa olisi vuorovaikutteisuutta ja animaatioita, jotka saavat sovelluksen elämään. Framerin etuna verrattuna toimeksiantajan nykyisin käyttämään Figmaan on se, että se tukee Reactia, joka on toimeksiantajayrityksen pääsääntöisesti käyttämä sovelluskehityksen tekniikka ja joka osaltaan mahdollistaa erilaisten animaatioiden ja vuorovaikutteisten komponenttien ohjelmoimisen nykyaikaisiin sovelluksiin.

Opinnäytetyö on luonteeltaan produktiivinen, ja sen aikana toteutetaan pieniä vuorovaikutteisia komponenttiprototyyppejä Frameria hyödyntäen, jotta voidaan käytännössä todentaa Framerin toimintoja ja ominaisuuksia. Työn tavoitteena on tutkia, tehostaisiko Framer-työkalu toimeksiantajan sovelluskehityksen prosessia suunnitelmasta valmiiksi sovellukseksi ja kannattaisiko toimeksiantajan vaihtaa nykyinen Figma-työkalu Frameriin.

Opinnäytetyössä sivutaan käyttöliittymä- ja käyttökokemussuunnittelua sekä vuorovaikutussuunnittelua tarpeellisilta osin, jotta voidaan ymmärtää Framer-työkalun käyttöön liittyvää kontekstia. Työssä käsitellään myös tarpeellisilta osin peruskäsitteitä sovelluskehityksestä Reactilla. Tähän kontekstiin liittyvät myös tyylit (kutsutaan opinnäytetyössä yleisesti käytetyllä lyhenteellä CSS), Next.js ja Framer motion-kirjasto. Komponentti-sana mainitaan opinnäytetyössä myös usein. Sillä tarkoitetaan yksittäistä pientä käyttöliittymän osaa, kuten esimerkiksi painiketta. Näiden käsitteiden ymmärtäminen on opinnäytetyön kannalta tärkeää, koska nämä liittyvät sekä Frameriin että varsinaisen sovelluksen ohjelmointiin.

2 KÄYTTÖLIITTYMÄN SUUNNITTELU

Cooperin ym. (2014, 23) mukaan käyttöliittymän suunnittelu sisältää kolme toisiinsa liittyvää aihetta (kuva 1). Näitä ovat ulkoasu, sisältö ja käyttäytyminen. Näille kaikille on lisäksi määritelty omat erityiset tietynlaiseen suunnitteluun suuntautuneet ammattinimikkeet. Ulkoasua suunnittelevat graafiset suunnittelijat ja teolliset muotoilija, sisältöä tietoarkkitehdit, copywriterit, animoijat ja äänisuunnittelijat. Käyttäytymisen suunnittelusta puolestaan vastaan vuorovaikutussuunnittelija.



Kuva 1. Käyttöliittymän suunnittelussa kolme toisiinsa liittyvää aihetta

Käyttöliittymän suunnittelussa on eri aikoina erilaisia trendejä. Nykyajan suunnittelutrendeihin kuuluu mm. minimalistinen suunnittelu, joka tarkoittaa sitä, että käyttöliittymä on mahdollisimman pelkistetty. Tämän avulla voidaan poistaa sovelluksesta monimutkaisuutta. Myös pitkät, scrollattavat sivut, parallax-efektit ja liikkuvat kuvat ovat nykyajan trendien mukaisia ominaisuuksia, samoin kirkkaat värit ja sovelluksen aistillinen typografia. Nykyaikaisessa käyttöliittymän suunnittelussa on myös syytä kiinnittää huomiota etenkin mobiililaitteille optimoituun suunnitteluun, sillä näistä laitteista on tullut meille jokaisella arkipäivää. (Kachhawa & Tiwari 2021, 37–39.) Pidän myös itse hyvin tärkeänä sitä, että sovelluksen suunnittelussa on huomioitu mobiilikäyttö, sillä omiin kokemuksiini pohjautuen olen havainnut, että nykyisin lähes kaikki asiat hoidetaan älypuhelimen välityksellä pankkiasioista uutisten selailuun. Näin teen myös itse.

2.1 Käyttöliittymäsuunnittelu

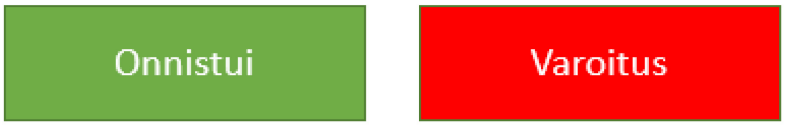
Käyttöliittymäsuunnittelu, eli UI-suunnittelu (eng. user interface design), keskittyy nimensä mukaisesti suunnittelemaan käyttöliittymää, jota käyttäjä käyttää. Käyttöliittymä on laitteen ja käyttäjän välinen työkalu kommunikoida ja olla vuorovaikutuksessa toistensa kanssa. (Kachhawa & Tiwari 2021, 38.)

Käyttöliittymäsuunnittelun tulisi noudattaa tiettyjä sääntöjä ja periaatteita, jotta käyttöliittymää olisi helppoa ja mielekästä käyttää. Näitä periaatteita ovat johdonmukaisuus, hierarkkisuus ja persoonallisuus. Käyttöliittymäsuunnittelu sisältää siis muutakin, kuin värien, kuvien ja fonttien suunnittelua, vaikka ne ovatkin tärkeä osa kokonaisuutta. (Levinson & Belton 2017, 49.)

Johdonmukaisuuden periaatteella tarkoitetaan sitä, että käyttöliittymässä olevat samankaltaiset elementit näyttävät visuaalisesti samalta ja käyttäytyvät käyttäjän näkökulmasta ennalta arvattavasti, eli samalla tavalla kuin sovelluksissa yleensä on totuttu näkemään. (Levinson & Belton 2017, 49–50.) Ihminen on luonnostaan haluton opettelemaan uusia erilaisia käyttöliittymiä. Niinpä käyttöliittymän pitäisi muistuttaa sellaista käyttöliittymää, jota käyttäjä on käyttänyt jo aiemmin, koska silloin uuden oppiminen on huomattavasti helpompaa. (Saariluoma ym. 2010, 61–62.) Myös Kachhawa ja Tiwari (2021, 37–38) mainitsevat, että käyttäjät eivät jaksa käyttää paljon aikaa käyttöliittymän käytön

opetteluun. Mielestäni on hyvä asia, että monet sovellukset muistuttavat toisiaan, jolloin uuden sovelluksen käyttäminen on helppoa jo alusta saakka, mikäli on jo aiemmin oppinut käyttämään toista vastaavanlaista sovellusta.

Hierarkkisuudella tarkoitetaan sitä, että näytöllä olevat visuaaliset elementit ovat hierarkkisessa järjestyksessä, tärkeimmän asian ollessa yleensä ylimmäisenä. Tärkeä elementti on myös isompi kuin vähemmän tärkeä elementti. Myös värit liittyvät hierarkiaan. Yleisesti ottaen punainen on varoitusväri ja vihreä kertoo onnistumisesta (kuva 2). Sovelluksessa käytettävien elementtien tulee sopia toisiinsa, kuten esimerkiksi ikonien koon suhteessa tekstiin. Käytettävien ikonien ja kuvien tulee olla tarkoituksenmukaisia, ja sisältöön tai toimintoihin sopivia. (Levinson & Belton 2017, 50–51.) Jokaisen sovelluksen käyttöliittymässä tulisi myös olla selkeät tekstit (Kachhawa & Tiwari 2021, 38).



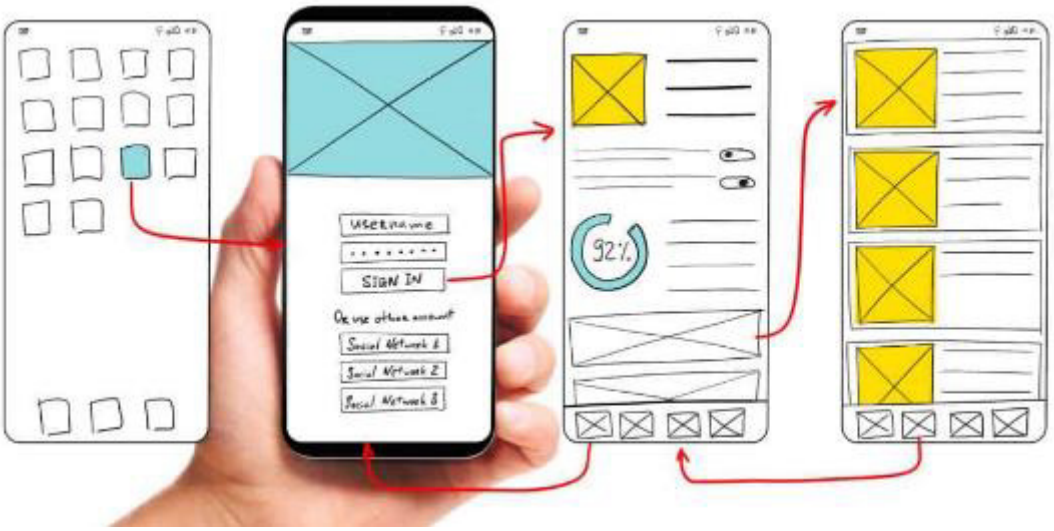
Kuva 2. Viestimiseen yleisesti käytetyt värit

Sovelluksen persoonallisuus on se, joka antaa niin sanotusti sovellukselle äänen, jolla sovellus viestittää käyttäjälle. Sovelluksen persoonalla pyritään erottamaan sovellus kilpailijoista ja luomaan käyttäjälle haluttua tunnetta ja mielikuvia sovelluksesta ja yrityksestä. (Levinson & Belton 2017, 50–51.) Mielestäni sovelluksen persoonallisuus on yksi lähtökohta sovelluksen menestymiselle.

Visuaalisesti käytettävä sovellus koostuu viidestä osasta. Näitä ovat sisällön asettelu, valitut fontit, värimaailma, kuvitukset ja tyylitellyt kontrollielementit, esimerkiksi sovelluksen painikkeet. Näiden elementtien valinnassa ja määrittelyssä täytyy ottaa huomioon aiemmin mainitut jatkuvuus, hierarkia ja persoonallisuus, jotta sovelluksesta saadaan mahdollisimman käytettävä. (Levinson & Belton 2017, 52.)

2.2 Käyttökokemussuunnittelu

Käyttökokemussuunnittelun, eli UX-suunnittelun (eng. user experience design), päämääränä on ymmärtää sovelluksen käyttäjiä, ja sitä kautta suunnitella sovelluksia, joita on helppo ja miellyttävä käyttää. Suunnittelua varten suunnittelijan tulee ymmärtää sovelluksen kohderyhmä, sekä missä ja miten he tyypillisesti käyttävät sovellusta, ja mitkä toimet käyttäjän on tehtävä, jotta sovelluksen käyttämisen tavoite saavutetaan käyttäjälle luonnollisella, miellyttävällä ja tehokkaalla tavalla (kuva 3). (Levinson & Belton 2017, 15.) Suunnittelussa on huomioitava se, että käyttäjän tavoitteet eivät ole sama asia kuin toiminnot. Tavoite on se, johon käyttäjä päätyy toimintojen jälkeen. (Cooper ym. 2014, 14.)



Kuva 3. Käyttökokemussuunnittelu tähtää käyttäjän tavoitteiden saavuttamiseen käyttäjän eri toimintojen tekemisen seurauksena

Sovellukset tulisi lähtökohtaisesti suunnitella keskitason käyttäjille sen sijaan, että niistä tehtäisiin turhan monimutkaisia tai liian yksinkertaisia. Uudella käyttäjällä tulee aina väistämättä olemaan jonkin asteinen oppimiskäyrä sovelluksen käytön opettelussa, mutta käytön opettelu ei saisi olla ylitsepääsemättömän vaikeaa. Toisaalta ei ole hyvä myöskään joutua sellaiseen tilanteeseen, että sovellus kohtelee käyttäjäänsä aina aloittelijana. Molemmat tilanteet saavat käyttäjän lopulta turhautumaan, ja turhautuminen johtaa helposti huonoon käyttökokemukseen. (Cooper ym. 2014, 237.) Itse olen aiheesta täysin samaa mieltä, sillä olen kohdannut joitain sovelluksia, joita on todella hankala käyttää. Sellaista sovellusta ei halua käyttää, jos se ei ole aivan välttämätöntä. Liian helposti käytettäviä sovelluksia ei tule mieleeni ainoatakaan. Sellaiset

sovellukset ovat mielestäni ärsyttäviä, jotka tarjoavat jokaisella käyttökerralla erillisessä ikkunassa opaskierrosta sovelluksen käyttämiseksi. Tästä tulee itselleni mielikuva, että sovellus kohtelee minua aina aloittelijana.

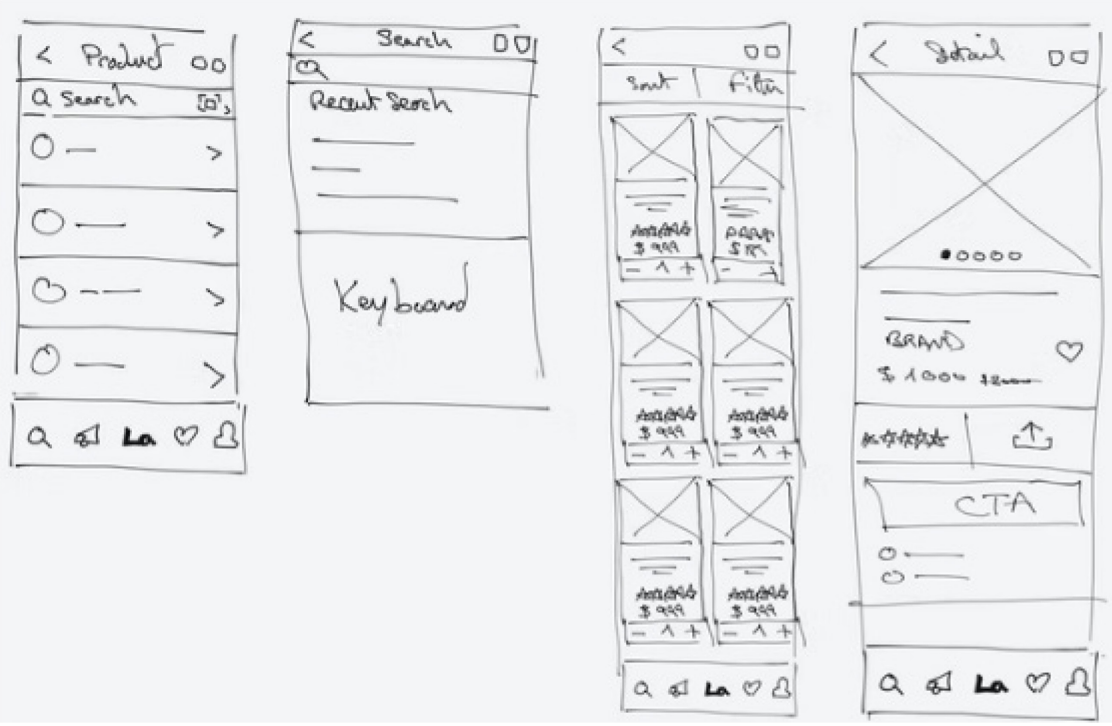
Sovelluksen käytettävyys on erittäin tärkeä termi sovellusten suunnittelussa ja sillä on useita erilaisia kriteerejä. Käytettävyyteen liittyviä kriteerejä ovat mm. tehokkuus, hyödyllisyys, turvallisuus, muistettavuus, käyttäjän sitouttaminen ja tyytyväisyys. Useat kriteereistä ovat käyttäjään liittyviä subjektiivisia tunteita, joita voi olla vaikea mitata. Sen sijaan esimerkiksi tehokkuutta pystyy mittaamaan ajallisesti, eli kuinka nopeasti käyttäjä pääsee tavoitteeseensa. (Kachhawa & Tiwari 2021, 38.)

Käyttökokemussuunnittelu alkaa yleensä käyttäjätutkimuksesta, jonka tarkoituksena on tutkia ja tutustua ihmisten käsityksiin, mieltymyksiin, tarpeisiin ja käyttötapoihin. Toisin sanoen tutkijan tulee siis osata asettua käyttäjän asemaan, jotta hän voisi ymmärtää kuinka käyttäjä haluaisi käyttää sovellusta. Kohdeyleisön ymmärtäminen onkin kriittisessä osassa menestyvän sovelluksen kehittämisprosessia. (Levinson & Belton 2017, 15–16.) Myös Cooper ym. (2014, 22) mainitsevat, että yksi suunnittelijan parhaista työkaluista on empatia. Myös heidän mukaansa suunnittelijan täytyy osata asettaa itsensä käyttäjän asemaan ja yrittää tuntea samoja asioita, kuin käyttäjä tuntisi sovellusta käyttäessään. Suurin virhe suunnittelussa onkin, että suunnittelija unohtaa käyttäjien tarpeet ja tunteet, ja sen sijaan keskittyy vain omiin tunteisiinsa ja ajatuksiinsa.

Käyttäjätutkimusta voidaan esimerkiksi toteuttaa tekemällä kohderyhmälle yksilö- tai ryhmähaastatteluja, tutkimuskyselyjä, käytettävyystestejä tai pyytämällä kohderyhmään kuuluvaa henkilöä pitämään päiväkirjaa omasta käyttäytymisestä ja havainnoista tietyn aikavälin sisällä. Tutkimuksen tekeminen ei siis vaadi erityistä laboratoriota tai erillistä tilaa tutkimuksen tekemiseksi, vaan tutkimusta voi tehdä hyvin monella eri tavalla. Tutkimuksen tekemisen kannalta kaikista tärkeintä on halu kuunnella käyttäjiä ja kohderyhmää. Tutkimuksen tuloksena tunnistetaan kohdeyleisöä ja heitä voidaan jakaa tyypillisiin käyttäjäpersooniin tai käyttäjätyyppeihin. (Levinson & Belton 2017, 16–19.)

Kerättyä tietoa käytetään lopuksi sovelluksen suunnittelussa. Kun käyttäjistä ja ympäristöstä on kerätty tietoa tarpeeksi, voidaan suunnitella käyttäjille soveltuvia käyttöliittymiä, joissa on otettu huomioon myös käyttäjien mahdollisia erityispiirteitä. Tästä esimerkkinä vanhempi väestö, joille suunnitelluissa sovelluksissa tarvitaan mm. isompaa ja hyvin luettavaa fonttia. (Cooper ym. 2014, 132.)

Levinsonin ja Beltonin (2017, 31–19) mukaan käyttökokemussuunnittelussa sovelluksesta olisi hyvä tehdä yksinkertainen rautalankamalli (eng. wireframe) (kuva 4), jossa ei niinkään ole tarkoituksena suunnitella sitä, millaiset esimerkiksi sovelluksen värit ovat vaan enemminkin sitä, kuinka sovellusta käytetään, ja millaisia toimia käyttäjän pitää tehdä tavoitteen saavuttamiseksi. Tämän suunnitelman tekemiseen riittää hyvin myös pelkkä kynä ja paperi.



Kuva 4. Rautalankamallit

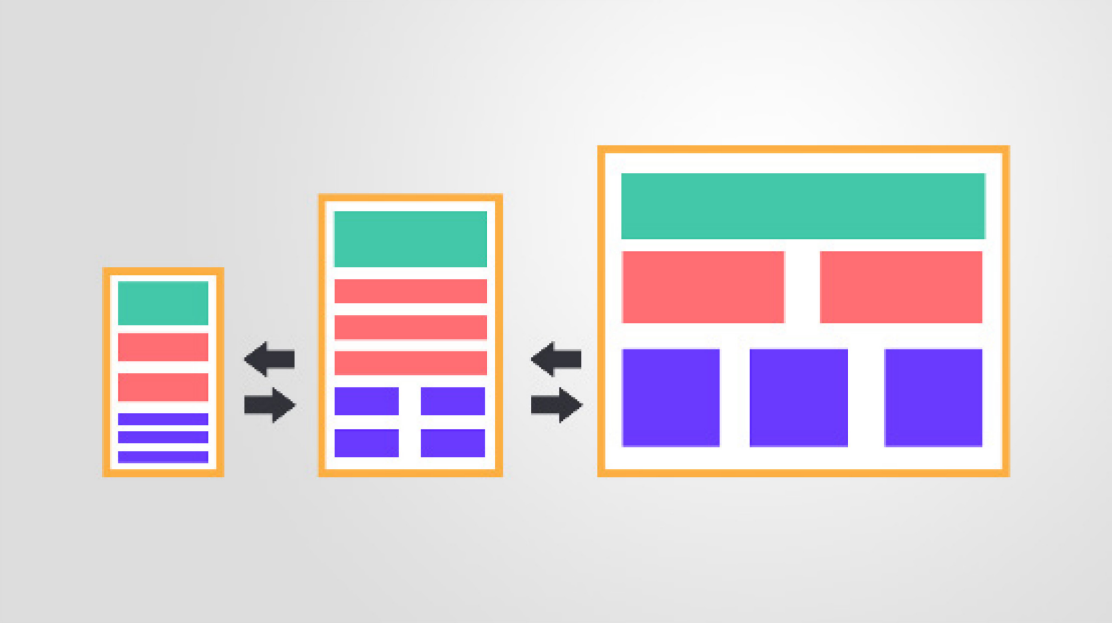
Toisaalta rautalankamallin tekeminen pelkän kynän ja paperin avulla ei välttämättä ole se paras lähestymistapa, ainakaan kovin syvälliseen prototyypitykseen ja suunnitteluun. Käyttökokemussuunnittelija ja sovelluskehittäjä Stefan Domzalski (2021) nostaa esille kuusi erilaista paperille suunnittelun ongelmaa.

Ensimmäisenä ongelmana hän pitää sitä, että paperille suunnittelu on hyvin epätarkkaa, koska suunnittelijalla tulee usein virheitä ja niitä voi olla vaikea korjata paperisuunnitelmissa, tai ainakin se on vähintään hidasta.

Toisena merkittävänä ongelmana Domzalski (2021) näkee sen, että suunnitelmista on vaikea tehdä toistoja, eli iteraatioita. Jos suunnitelmaa täytyy muuttaa, kuten usein suunnitellessa täytyy, suunnittelija joutuu käytännössä piirtämään kaiken uudestaan, eikä samoja komponentteja voi käyttää "leikkaa ja liimaa" -periaatteella, kuten digitaalisessa maailmassa. Tähän liittyy myös kolmas ongelma, joka on se, että suunnitelmaa on vaikea monistaa.

Neljäs ongelma Domzalski (2021) mukaan liittyy siihen, että paperille piirretty malli ei useinkaan näytä yhtä hienolta ja houkuttelevalta, kuin tietokoneella tehty. Tämän vuoksi idea saatetaan jopa hylätä, vaikka se olisi todellisuudessa tarpeellinen ja hyvä. Mutta koska malli ei näytä hyvältä, se koetaan hyödyttömäksi. Tähän viittaa myös Venäläisen (2022) haastattelussa mainitsema kommentti siitä, että jos sovellus näyttää ja tuntuu hyvältä, sen myös ajatellaan toimivan paremmin.

Viides ongelma on skaalautuvuus, joka on oikealla laitteella täysin erilainen, kuin paperille piirrettynä, ja usein paperille piirretty versio ei toimi sellaisenaan digitaalisessa muodossa, eikä eri näyttöpäätteillä (kuva 5).



Kuva 5. Suunnittelu eri näyttöpäätteille on erilaista

Viimeisenä asiana Domzalski (2021) mainitsee, että suunnitelma nykyaikaisessa käyttöliittymäsuunnittelussa viedään joka tapauksessa digitaaliseen muotoon käyttämällä jotain siihen tarkoitettua ohjelmistoa, kuten Adobe Xd:tä, Figmaa tai Frameria.

Vaikka edellä mainitut asiat ovatkin otteita käyttökokemussuunnittelun ammattilaisen Domzalskin blogista, hänen kuvailemansa ongelmat kertovat mielestäni hyvin siitä, miksi suunnittelussa on siirrytty käyttämään prototyypitystä varten suunniteltuja ohjelmia ja mitä etuja digitaalisten suunnittelutyökalujen käyttäminen antaa suunnittelulle.

Asiaa voidaan kuitenkin tarkastella myös hieman eri näkökulmasta, kuten Saariluoma ym. (2010, 120–121) tekevät. Heidän mukaansa on hyvä tehdä useita nopeita luonnosteluja suunnittelun alkuvaiheessa, minkä jälkeen sopivimmasta luonnoksesta tehdään varsinainen prototyyppi. Tämä siksi, että nopeita luonnoksia on helpompi hylätä kuin prototyyppiä, jonka tekemiseen on käytetty enempi aikaa. Ennen kuin luonnostelua voidaan aloittaa tekemään, täytyy suunnittelijalla olla hyvä kokonaiskuva suunniteltavasta sovelluksesta. Nopeiden luonnoksien visualisointiin ei saa myöskään käyttää liikaa aikaa.

Venäläinen (2022) puolestaan on uransa aikana nähnyt, kuinka käyttökokemussuunnittelu on muotoutunut ja muuttuu edelleen. Hänen mukaansa sovelluksesta kannattaa tehdä suoraan vuorovaikutuksellinen prototyyppi, jota asiakas voi katsella sekä kokeilla miltä sen käyttäminen tuntuu. Tällaisen suunnittelun mahdollistavat nykyaikaiset suunnitteluun tarkoitetut sovellukset, kuten Framer.

Itsekin olen sitä mieltä, että nykyaikainen tekniikka on tuonut suunnittelijoille niin paljon mahdollisuuksia, että on parempi tehdä suoraan sellainen suunnitelma, jota asiakkaalle voi helposti esittää, ja jota hän voi itse kokeilla, ja jota helppo ja nopea tarvittaessa muuttaa.

2.3 Vuorovaikutussuunnittelu

Vuorovaikutussuunnittelu (eng. interaction design) on aika uusi suunnittelun alue, johon on viime vuosina alettu kiinnittämään huomiota entistä enemmän,

sillä on huomattu, että vuorovaikutteiset sovellukset menestyvät markkinoilla paremmin. (Cooper ym. 2014, 11.)

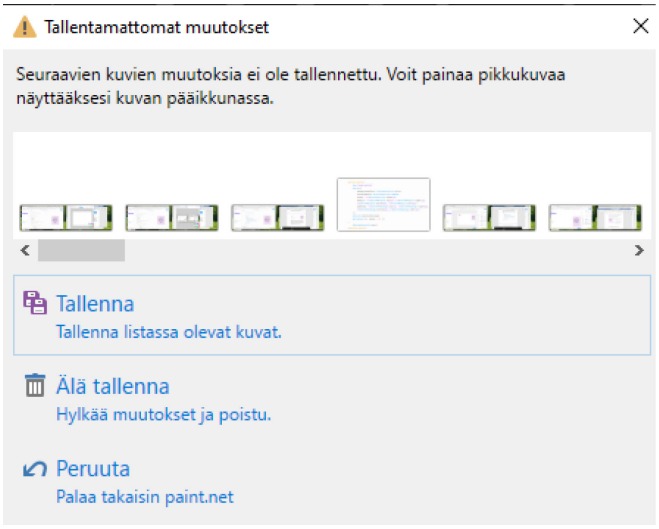
Vuorovaikutussuunnittelu käsittää sovelluksen tavat olla vuorovaikutuksessa käyttäjänsä kanssa. Vuorovaikutussuunnittelussa huomioidaan käyttökokemussuunnittelu ja käyttäjätutkimuksen tulokset. Vuorovaikutussuunnittelua laajennetaan myös visuaaliseen suunnitteluun. (Steane & Yee 2018, 6.) Visuaalinen suunnittelu kannattaakin ottaa mukaan jo suunnittelun varhaisessa vaiheessa, jotta saadaan vuorovaikutusprosessi käyttäjien kanssa pian alkuun. Tämä mahdollistaa myös suunnittelun suunnan määrittelyä käyttäjäpalautteen perusteella. Tämä on tärkeää, jotta käyttöliittymästä tulisi visuaalisesti käytettävä ja näköaistille ergonominen. Onnistuneena lopputuloksena asioiden hahmottaminen on nopeaa, vaivatonta ja virheetöntä. (Saariluoma ym. 2010, 160–161.)

Vuorovaikutussuunnittelussa yksi keskeinen asia on psykologia. Ihmisen ja tietokoneen keskinäiseen vuorovaikutussuhteeseen kuuluu oleellisesti ihmisten käyttäytymisen ja prosessien ymmärtäminen. Näiden asioiden ymmärtäminen auttaa ratkaisemaan monenlaisia suunnitteluun liittyviä ongelmia. Ihmisellä on rajoittunut suorituskapasiteetti, mikä vaikuttaa tarkkaavaisuuteen, muistiin ja aisteihin. Myös tunteet ja motiivit, ihmisten välinen kommunikointi, ihmisen mielensisällöt, kulttuuri, erilaiset ryhmät tai organisaatiot ja yksilön oma persoonallisuus vaikuttavat vuorovaikutukseen ja vuorovaikutussuunnitteluun. (Saariluoma ym. 2010, 61–63.) Myös Kachhawa ja Tiwari (2021, 38) mainitsevat käyttöliittymän käyttäjälle syntyvistä psykologisista tunteista, jotka ovat kuitenkin käyttäjän itsensä subjektiivisia kokemuksia.

Cooper ym. (2014, 168) mainitsevat, että vuorovaikutteisen sovelluksen takana ovat tietyt arvot, joita suunnittelijan tulee seurata. Yksi näistä arvoista on etiikka, joka pitää sisällään sen, että sovellus auttaa käyttäjäänsä eikä tuota harmia. Toinen arvo on tarkoituksenmukaisuus, jonka täyttyessä sovellus auttaa käyttäjäänsä saavuttamaan tavoitteensa ja on myös samalla käytettävä. Sovelluksen tulee myös täyttää sen tilanneen yrityksen tai organisaation tarpeet ja vaatimukset. Lisäksi sovelluksen tulee olla elegantti. Tällä tarkoitetaan sitä, että sovellus on yksinkertainen mutta toimiva, ymmärrettävä ja stimuloi ihmisen kognitiivisia aisteja ja tunteita.

Cooperin ym. (2014, 179) mukaan käyttäjät haluavat, että sovellus käyttäytyy heitä kohtaan kuten miellyttävä ihminen käyttäytyisi. Tämä tarkoittaa esimerkiksi sitä, että sovellus tukee ja opastaa miellyttävällä tavalla käyttäjän tekemään toimia ja sovellus on huomaavainen käyttäjäänsä kohtaan.

Huomaavaisen sovelluksen määrittelyssä Cooper ym. (2014, 182–183, 186) käyttävät useita eri määritelmiä. Heidän mukaansa huomaavainen sovellus muistaa käyttäjänsä ja käyttäjän tekemiä toimia, eikä näin ollen kysele aina samoja asioita uudestaan. Sovelluksen tulisi kysyä vain käyttäjän kannalta tarpeellisia kysymyksiä (kuva 6). Sovelluksen pitäisi kuitenkin käyttää tietynlaista harkintaa siitä, mitä tietoja tallennetaan muistiin automaattisesti ja mitä vain käyttäjän luvalla. Yksi esimerkki tästä on luottokorttitiedot. Tarpeellisten asioiden muistaminen lisää positiivista käyttökokemusta, kunhan ne eivät vaaranna käyttäjän tietoturvaa. Lisäksi sovelluksen tulisi opastaa käyttäjäänsä avuliaasti kertomalla, jos käyttäjän tavoitteeseen pääsemiseksi on vaihtoehtoisia reittejä tai joitain viivytyksiä tai ongelmia valitun reitin varrella. Tästä esimerkkinä dokumentin tulostaminen, jolloin sovelluksen tulisi kertoa, jos jonossa on jo useita dokumentteja ennen kuin käyttäjän haluama dokumentti on mahdollista tulostaa.



Kuva 6. Käyttöliittymä kysyy tarpeellisen kysymyksen käyttäjältä

Näiden lisäksi Cooper ym. (2014, 183) mainitsevat huomaavaisen sovelluksen käyttävän maalaisjärkeä. Tällä tarkoitetaan sitä, että ohjelmisto varoittaa käyttäjää, jos tämä on tekemässä joitain epänormaaleja toimia, tai että esimerkiksi jatkuvasti käytettävät valintapainikkeet sijaitsevat eri paikassa kuin

valintapainikkeet, joita ei käytetä lähes koskaan. Huomaavainen sovellus osaa myös ennakoida käyttäjän tarpeita, jolloin esimerkiksi lennon varaamisen yhteydessä sovellus voi ehdottaa käyttäjälle hotellihuoneita. Mielestäni nykyaikaiset sovellukset ovat varsin huomaavaisia ja osaavat ehdottaa käyttäjälle asioita, joista käyttäjä voisi olla kiinnostunut. Esimerkiksi nykyisin hyvin monessa verkkokaupassa on ehdotuksia tuotteista, joista käyttäjä voisi olla kiinnostunut sen perusteella millaisia tuotteita hän on katsellut kaupassa tai mitä muut käyttäjät ovat tilanneet samalla kun ovat tilanneet saman tuotteen kuin käyttäjällä on ostoskorissa.

Huomaavainen sovellus on myös tunnollinen ja vastuuntuntoinen. Se ymmärtää tehtävää hoitaessaan hoitaa myös tehtäviä päätehtävän ympäriltä, jotta lopputulos olisi käyttäjälle paras mahdollinen. Tästä esimerkkinä päivämäärän tallentaminen tiedoston nimen mukana, jotta käyttäjän on helppoa löytää oikea tiedosto myös seuraavalla kerralla. Sovelluksen tulee myös osata pitää käyttäjä ajan tasalla ja informoida käyttäjää sopivissa määrin, mutta ei kuitenkaan siten, että se kertoo turhaan omista ongelmistaan. Informaation laadun ja määrän tulee olla siis oikeanlaista. Sovelluksen tulisi estää käyttäjiä tekemästä virheitä. Jos sovelluksen itsensä toiminta syystä tai toisesta päätyy virheeseen, pitäisi sovelluksen pystyä toipumaan virheestä ilman, että se aiheuttaa huomattavaa haittaa käyttäjälle tai esimerkiksi tietojen menetystä. (Cooper ym. 2014, 184–187, 190.)

2.4 Luonnokset ja prototyypit

Luonnoksen merkitys käyttöliittymän suunnittelussa on Saariluoman ym. (2010, 120–121) mukaan suuri, koska sen avulla suunnittelijoiden on mahdollista saada määriteltyä paremmin ongelmakohtia. Lisäksi luonnostelu auttaa myös saamaan uusia ideoita. Tärkeintä on, että suunnittelun alkuvaiheessa ei tehdä liian tiukkoja suunnitelmia, vaan on mahdollista tehdä nopeita luonnoksia, jotka toimivat keskustelun ja tarkastelun pohjana. Jos luonnoksista tehdään jo heti aluksi liian tarkkoja, ne voivat muuttaa prosessin suuntaa liian aikaisessa vaiheessa. Heidän mukaansa luonnostelua ei kannata tehdä kovin kauaa yksin, vaan luonnos kannattaa jakaa kaikille suunnitteluprojektiin osallistuville jo alkuvaiheessa, jotta luonnosteluvaiheesta saataisiin paras mahdollinen hyöty. Joka tapauksessa ennen kuin luonnosta voidaan aloittaa

tekemään, täytyy suunnittelijalla olla hyvä kokonaiskuva suunniteltavasta sovelluksesta.

Luonnostelu on usein suunnittelijoiden kokemusten jakamista, eikä luonnos ole sama asia kuin prototyyppi. Luonnos on vain nopea versio ideasta, eikä sen tekemiseen ole käytetty paljoa aikaa. Näin ollen sen hylkääminenkin on helpompaa, mikäli luonnos ei etene jatkokehitykseen. Nopeiden luonnoksien visualisointiin ei saa myöskään käyttää liikaa aikaa. Suunnitteluprosessin aikana nopeista luonnoksista kuitenkin kasvaa pikkuhiljaa yksityiskohtaisempia ja visuaalisempia, jolloin on jo siirrytty suunnittelussa toiseen vaiheeseen. (Saariluoma ym. 2010, 120–121.)

Käyttäjien ottaminen mukaan luonnosteluprosessiin nostaa ideoiden määrää ja saa aikaan uusia ja omaperäisiä ideoita, joita suunnittelija ei välttämättä ole ottanut huomioon ollenkaan. Käyttäjien on myös helpompi kommunikoida suunnittelijoiden kanssa, kun käytetään nopeita luonnoksia. Juuri alkuvaiheen suunnitteluprosessi luonnoksineen on kaikista innovatiivisinta aikaa suunnitteluprosessissa ja käyttäjien suunnitteluun osallistaminen tässä vaiheessa hyödyntää lopullista sovellusta kaikista eniten. Tämä on myös lähtökohta ihmislähtöiseen suunnitteluprosessiin. (Saariluoma ym. 2010, 124–125.)

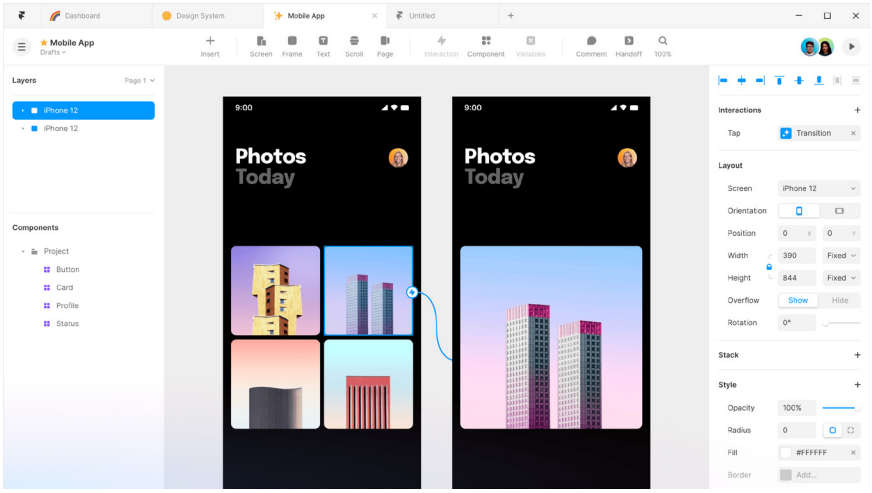
Luonnostelun yksi käytetyimmistä luonnostelutekniikoista on skenaariopohjainen tekniikka, jossa tuotetaan yksittäisiä käyttötapauksia, jotka toimivat pohjana sille, miten suunnittelija ajattelee käyttäjän käyttävän suunniteltavaa sovellusta. (Saariluoma ym. 2010, 126.)

Konsepti on suunnittelun yksi tärkeä käsite, joka ei kuitenkaan ole sama asia kuin skenaario. Konsepti on ikään kuin kokoelma tiedoista, jotka on tulkittu tai kerätty alkuvaiheen käyttäjä- ja muiden tutkimusten ohella yhdeksi kokonaisuudeksi. Konseptisuunnittelu kertoo, miten on mahdollista ratkaista suunnitteluongelmia kerätyn tiedon perusteella, joten tämä on merkittävä osa suunnitteluprosessia. Tässä kohtaa ei kuitenkaan vielä käsitellä yksityiskohtaisesti käyttöön otettavaa teknologiaa. Suunnittelu on luonteeltaan jatkuvasti tarkentuva hahmotusprosessi, joten se sisältää useita iteraatioita ja myös useita erilaisia konsepteja, joiden perusteella voidaan määritellä lopullinen suunnittelulinja. Kun konseptin piirteistä on päästy sopuun, voidaan keskittyä uusien

ongelmien piirteisiin ja niiden ratkaisemiseen iteraatioiden kautta, jolloin suunnitelmasta kehittyy edelleen yksityiskohtaisempi. (Saariluoma ym. 2010, 127.)

Konseptin tulisi kuvata useita eri asioita, kuten käyttäjän tavoitteet ja päämäärät, käyttäjään liittyvät biologiset, psykologiset ja sosiokulttuuriset tekijät sekä konteksti, eli missä ja millaisissa tilanteissa käyttäjä käyttää sovellusta. Konseptisuunnitelmasta pitäisi löytyä myös yleisellä tasolla kuvattuna teknologia, jota käyttäjä käyttää saavuttaakseen päämääränsä. Lisäksi vuorovaikutussuunnittelua korostavassa suunnittelussa kuvataan, kuinka teknologiaa on käytettävä haluttuun päämäärään pääsemiseksi, kuitenkaan menemättä käyttöliittymään liittyviin yksityiskohtiin. (Saariluoma ym. 2010, 128–129.)

Kun alkuvaiheen luonnosteluista ja konseptisuunnittelusta päästään eteenpäin, alkaa seuraava vaihe eli prototyypin tekeminen, jossa konseptisuunnitelma muutetaan todelliseksi. Prototyyppi on luonteeltaan sovelluksen ensiaste, jota testataan ja kokeillaan aktiivisesti. Keskeistä prototyyppien tekemisessä on kommunikaatio asiakkaan kanssa sekä asiakkaan vakuuttaminen. Toiminnallisen prototyypin avulla asiakas ja käyttäjät voivat nähdä, millainen loppusovellus olisi ja miten se käyttäytyy, joten se helpottaa lopullisen sovelluksen käytettävyyden ja toiminnallisuuden kehittämistä edelleen (kuva 7). (Saariluoma ym. 2010, 131.)



Kuva 7. Prototyyppi on sovelluksen esiaste, jota asiakas voi myös kokeilla

Nopea prototyyppien tekeminen luo suunnittelijalle mahdollisuuden kokeilla ideoiden toimivuutta nopeasti. Haasteena on kuitenkin se, että jos luodaan vuorovaikutteisia prototyyppejä, se voi edellyttää ohjelmointia, jolloin

prototyypin tekemiseen menee huomattavasti enemmän aikaa. (Saariluoma ym. 2010, 133.)

3 PROTOTYYPIN RAKENTAMINEN

Tässä osassa päästään opinnäytetyön produktiiviseen osioon ja testataan rakentaa Framerilla erilaisia pieniä komponenttiprototyyppejä. Prototyyppejä testattiin pienillä yksittäisillä komponenteilla kokonaisen käyttöliittymänäkymän sijaan siksi, koska React-ohjelmoinnissa käytetään komponentteja, eikä kokonaisen käyttöliittymäsuunnitelman tekemisestä olisi sinänsä ollut mitään mainittavaa lisähyötyä tutkimustyön kannalta. Komponenttien luominen Frameriin ei ollut opinnäytetyön tekijälle entuudestaan tuttua, joten tämän osion komponenttien tekeminen vaati ennakkoon myös Framerin perusteiden opettelemisen.

Komponentti on erittäin keskeinen käsite React-ohjelmoinnissa. Komponentti voi olla esimerkiksi yksi painike tai jokin muu pienempi tai isompi yksittäinen käyttöliittymän osa, jota voi tarvittaessa myös uudelleen käyttää. Komponentti on itse asiassa vain JavaScript-funktio, joka palauttaa tietynlaiseksi määritellyn React-elementin, joka on itsenäisesti toimiva oma yksikkönsä, kuten esimerkiksi yksi yksittäinen painike on. (React s.a.)

3.1 Toimeksiannon lähtökohdat ja tavoitteet

Lähtökohtia ja opinnäytetyön viitekehyksen hahmottamista varten on haastateltu toimeksiantajayrityksen, Mindhive Oy:n toimitusjohtajaa Ville Venäläistä. Venäläisellä on pitkä, yli 20 vuoden ura it-alalla takana, joten hänellä on paljon omakohtaista- ja muualta hankittua tietoa sovellusten suunnitteluun- ja kehittämiseen liittyen.

Venäläinen (2022) mainitsee haastattelussa lähtökohdiksi kolme keskeistä käyttöliittymäsuunnitteluun liittyvää asiaa. Ensimmäisenä asiana hän nostaa esille sen, että käyttäjät ovat tottuneet käyttämään suuria sovelluksia, joten käyttäjien odotukset ovat jo lähtökohtaisesti sellaiset, että sovelluksen kuuluu näyttää ja tuntua hyvältä. Tästä muodostuu tietynlaisia suunnitteluvelvoitteita, jotta käyttäjien odotukset saadaan täytetyksi. Venäläisen (2022) mukaan on myös tutkitusti todettu, että jos sovellus näyttää hyvältä, niin käyttäjä kokee

sovelluksen myös toimivan paremmin, vaikka todellisuudessa toiminnallisuudet olisivat täysin samat myös vähän huonommalta näyttävässä sovelluksessa. Tämä on hänen mielestään ollut aiemmin jopa vähän vähätelty asia sovelluskehityksessä, vaikka hän mainitsee asian huomioimisen olevan todella tärkeä osa sovelluksen suunnittelua ja lisäarvon tuottamista. Toimeksiantaja pyrkii ottamaan tämän asian aina huomioon sovelluksien suunnittelussa, jotta he voivat tehdä sovelluksia, jotka eivät ainoastaan tee niille suunniteltuja tehtäviänsä, vaan myös näyttävät ja tuntuvat hyvältä.

Jotta sovellus näyttäisi ja tuntuisi hyvältä, niin Venäläisen (2022) mukaan sovellus ei voi enää olla vain staattisten sivujen kokoelma, vaan sen pitää myös hyödyntää ihmisen aisteja ja tapoja hahmottaa. Tästä esimerkkinä ovat käyttöliittymässä tapahtuvat liikkeet ja eleet, joilla käyttöliittymä viestittää käyttäjälle, että se reagoi käyttäjän toimiin, jotain tapahtuu tai käyttäjä voi tehdä joitain toimia. Tämä on myös hänen mukaansa osa käyttäjäkokemusta ja suunnittelun työkalut, kuten Framer, ovat tärkeitä tällaisten asioiden havainnoimiseksi ja suunnittelemiseksi koska Framer ei ole vain käyttöliittymän suunnittelua varten oleva työkalu, vaan sillä voi myös tehdä näyttäviä ja hyvän tuntuisia animaatioita ja eleitä, joista tulee osa ihmisen ja laitteen välistä viestintää.

Venäläinen (2022) korostaa etenkin ihmislähtöistä suunnittelua, jossa huomioidaan ihmisen tapa hahmottaa ja havainnoida asioita sekä kohdataan käyttäjä tunnetasolla. Hänen mukaansa tällaisten suunnitelmien tekeminen pelkästään kooditasolla on työlästä ja aina parhaat animoijat ja käyttöliittymäsuunnittelijat eivät välttämättä ole ohjelmoijia.

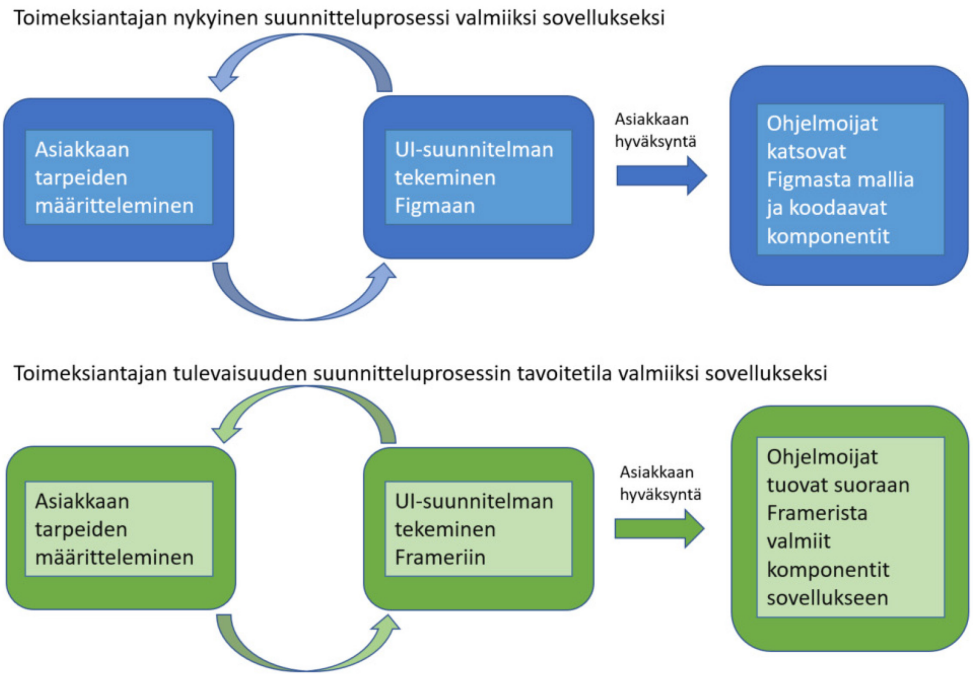
Toinen hänen mainitsemansa lähtökohta on se, että perinteisesti sovellusten suunnittelu ja sovelluskehitys ovat olleet kaksi aika erillistä asiaa ja käyttöliittymäsuunnittelussa osa suunnittelijoista on käyttänyt suunnitteluprosessissaan wireframe-suunnittelua ja erilaisilla blokeilla leikkimistä, jossa halutaan saada fokus pelkästään toiminnallisuuteen tai rakenteeseen, eikä siihen miltä sovellus näyttää. Venäläinen sanoo haastavansa asian, koska tämä lähestymistapa ei ole hänen mielestään käyttäjälähtöistä. Hänen mielestään asiakas ei pysty hahmottamaan rautalankamalleista millainen sovelluksen on tarkoitus loppujen lopuksi olla. Näin ollen Venäläisen mielestä on parempi suunnitella suoraan sovelluksesta sellainen prototyyppi, joka vastaa lopputuotetta ja joka on

myös lähempänä sitä sovellusta ja todellisuutta, jota asiakas on tottunut käyttämään mobiili- ja webbisovelluksissa jo aiemminkin.

Kolmas Venäläisen (2022) mainitsema lähtökohta on Framerin tarjoamat mahdollisuudet sovelluskehityksen ja suunnittelun välisen kuilun kaventamisessa ja hän kertoo itse myös odottavansa sitä, että Framer antaisi toimeksiantajalle mahdollisuuden viedä sovelluskehityksen prosessia askeleen eteenpäin. Kahteen edellä mainittuun lähtökohtaan pohjautuen Framer voisi olla työkalu, joka voisi mahdollistaa asioiden yhdistämisen käyttöliittymäsuunnittelijan/käyttökokemussuunnittelijan ja ohjelmoijien välillä. Tämä voisi olla mahdollista siksi, että Framer käyttää sisäisesti Reactia, joten animoituja, vuorovaikutteisia komponentteja voidaan mahdollisesti hyödyntää ohjelmoitavassa sovelluksessa. Venäläisen (2022) mukaan tämä tarjoaa selkeän mahdollisuuden huomattavasti nopeuttaa sovelluskehitystä, mikäli löydetään oikeat toimintatavat. Tämä tarkoittaisi sitä, että voitaisiin protoilla aika pitkään käyttöliittymän suunnittelussa ilman, että tarvitsee ohjelmoida ja samalla syntyisi koodipohjaa varsinaiselle toteutukselle, jonka käyttäminen suoraan sovelluksessa nopeuttaisi todella paljon sovelluksen ohjelmointia.

Kysyttäessä opinnäytetyön tuloksien mittareista, Venäläinen vastaa kehitysnopeuden syklin nopeutumisen yhdeksi mittariksi. Eli miten nopeasti Framerilla tehdyt, animaatioita ja vuorovaikutteisuutta sisältävät prototyypit saadaan siirrettyä osaksi tuotantokoodia siten, että ominaisuudet säilyvät tai miten paljon saadaan suoraviivaistettua sitä, että käyttöliittymäsuunnitelmat muuttuvat sovelluksen käyttöliittymäksi. Lähinnä kyse on mahdollisesti siitä, miten paljon koodeja voidaan käyttää suoraan valmiiseen sovellukseen.

Toimeksiantaja käyttää tällä hetkellä suunnittelussa Figmaa, joka on Framerin kaltainen käyttöliittymien suunnitteluun tarkoitettu työkalu, mutta se ei tue Reactia. Tämä on tarkoittanut sitä, että suunnitelmissa on käyttöliittymä, joka ohjelmoidaan kokonaan alusta alkaen suunnitelmien mukaisesti varsinaiseen sovellukseen. Venäläisen mielestä ei voida suoraan sanoa mikä on mittari siihen, että onko prosessi nopeampi verrattuna nykyiseen, koska täysin samanlaista vertailua ei voida tehdä (kuva 8). Venäläisen (2022) mielestä voitaisiin ehkä mitata aikaa, joka mahdollisesti säästyy, jos koodit saadaan tuotua Framerista suoraan sovellukseen.



Kuva 8. Toimeksiantajan prosessien nykytila ja tulevaisuuden tavoitetila

Toinen asia on animointi, jota toimeksiantajan mukaan ei ole tehty tarpeeksi yrityksessä, mutta sitä pitäisi tehdä ehdottomasti enemmän. Venäläisen (2022) mukaan animaatioiden tuominen suunnitteluprosessiin on yksi puuttuva pala. Tällöin kaikki siirtymät, eleet ja liikkeet olisivat jo etukäteen suunniteltuja ja mietittyjä. Hänen mielestään tämän tuloksia ei voida tällä hetkellä suoraan mitata, koska sitä ei tavallaan nyt tehdä, mutta pitäisi tehdä.

Kysyttäessä Venäläiseltä, että kuinka tärkeänä hän pitää prototyyppien tekemistä osana sovelluskehitystä, hän vastaa sen olevan erittäin tärkeää. Hänen mukaansa se on paras työkalu, joka hänellä on tähän mennessä tullut vastaan, jolla oman alansa asiantuntija voi puhua sovelluksesta yhteisellä kielellä asiakkaan kanssa. Asiakas voi myös prototyypin avulla tunnistaa asioita, joita pitää huomioida sovellukseen. Prototyyppien avulla sovellus saadaan asiakkaalle todelliseksi ja asiakas voi käyttää prototyyppiä ajattelunsa välineenä. Prototyyppi on myös erittäin keskeinen työkalu sovelluksen kehittämiseksi yhdessä asiakkaan kanssa, joten sen tärkeyttä ei voida Venäläisen (2022) mielestä korostaa liikaa. Asiakasta voidaan prototyypin avulla auttaa miettimään sovellukseen liittyviä asioita ja ongelmia sekä auttaa häntä työstämään digitaalisin keinoin yhdessä suunnittelijoiden ja ohjelmoijien kanssa ratkaisua hänen haasteeseensa.

Seuraava Venäläiseltä kysytty kysymys koski käyttöliittymäsuunnittelun muutoksista hänen uransa aikana. Venäläinen vastaa, että varsinainen käyttöliittymäsuunnittelu on tullut vasta hiljattain, sitä ei ole ollut aikanaan. Hänen mukaansa aikaisemmin on käytetty paljon ns. rautalankamallia. Aiemmin hän on huomannut sellaisen ongelman, että suunnittelijoiden mielestä asiakas kiinnitti huomiota vääriin asioihin, kuten vaikka väreihin. Hän mainitsee, että tällainen lähestymistapa ei ole asiakaslähtöistä ja jos suunnittelu ei lähde asiakkaan tarpeista ja mielipiteistä, niin se on virhe. Suunnittelijan ei pitäisi perustella esimerkiksi käytettyjä värejä omasta näkökulmastaan vaan lähteä liikkeelle aina asiakkaan tarpeista ja lähtökohdista. Mindhive Oy pyrkiikin siihen, että sovellus suunnitellaan valmiin näköiseksi jo alusta alkaen ja jos asiakasta häiritsevät vaikka suunnitellun sovelluksen värit, henki tai jokin muu, lähdetään aina tekemään muutoksia asiakkaan näkökulmasta. Kun nykyaikainen sovellus suunnitellaan nykyaikaisilla tekniikoilla valmista sovellusta vastaavaan muotoon, niin asiakas voi jopa kokeilla prototyyppiä omalla puhelimellaan. Tämä on Venäläisen (2022) mielestä ehdottomasti oikea suunta suunnittelussa, jo ihan myös havainnoinkin kannalta.

Venäläinen (2022) on myös huomannut, että käyttäjäkokemussuunnitteluun liittyvät tunnesuunnittelu (eng. emotional design), ja vuorovaikutussuunnittelu (eng. interaction design), ovat selkeästi tulossa enemmän keskiöön, mikä on hänen mielestään hyvä juttu, koska näiden huomioon ottaminen saa sovelluksen näyttämään ja tuntumaan hyvältä sekä sovellus elää, eikä ole staattinen. ”Tärkeää on huomioida se, miten ihminen havainnoi ja minkälaisiin asioihin silmä pysähtyy tai miten animaatiot ja liikkeet herättävät huomiota”, Venäläinen (2022) sanoo.

3.2 Framer

Framer on työkalu, joka auttaa sovelluksen suunnittelussa ja prototyyppien tekemisessä. Sen avulla sovelluksista voidaan luoda mallinnuksia, jotka ovat toiminnoiltaan hyvin lähellä oikeaa sovellusta. Framerin sisällä voidaan käyttää Reactia erilaisten komponenttien ja animointien ohjelmoimiseen. Framer käyttää sisäisesti itsekin Reactia käyttöliittymässään ja komponenteissaan. (Framer s.a.)

Framer on vielä kohtuullisen nuori työkalu, sillä Framerin ensimmäinen versio, Framer Studio on julkaistu vuonna 2015. Framerilla on palkattuja ohjelmoijia töissä, mutta Framerin ympärille on myös vuosien saatossa rakennettu suunnittelijoille suunnattu yhteisö, joilla on oma ryhmä pikaviestipalvelu Discordissa. (Framer s.a.)

Frameria on kehitetty tiimityötä silmällä pitäen. Yksi tai useampi käyttäjä voi käyttää Framerin hallintapaneelia samaan aikaan ja tehdä suunnitelmia siitä, miltä verkkosivut ja sovellukset voisivat näyttää ja miten eri osat käyttäytyvät käyttäjän tehdessä eri toimintoja, kuten vaikka painaessa painiketta. (Framer s.a.)

Mielestäni työkaluista löytyy oikeastaan kaikki tarvittava valmiina, sillä Framer sisältää tavallisen käyttöliittymän lisäksi myös koodieditorin. Koodieditorin avulla voi tuoda lisäominaisuuksia myös Framerin ulkopuolelta import-lauseiden avulla. Framerilla saa mielestäni lähtökohtaisesti tehtyä varsin nopeasti prototyyppejä ja niitä voidaan myös jakaa suoraan Framerista asiakkaan katseltavaksi ja kokeiltavaksi, jolloin asiakas voi käytännössä jo katsella ja kokeilla sovellusta ennen kuin siihen on koodattu riviäkään koodia itse. Näin ollen myös mahdollisten muutosten tekeminen on nopeaa. Kun suunnitelma on hiottu valmiiksi Framerin puolella, on siitä hyvä pohja alkaa ohjelmoimaan varsinaista sovellusta.

Toimeksiantajan nykyisin käyttämä Figma on hyvin saman tyyppinen työkalu, kuin Framer, mutta Framerissa on React-ohjelmointiin mahdollisuus, joka tekee siitä toimeksiantajan näkökulmasta mielenkiintoisen. Frameriin on myös mahdollista tuoda suunnitelmia Figmasta lisäosaa käyttämällä (Framer s.a.).

3.3 React

React on Facebookin kehittämä JavaScript-pohjainen käyttöliittymien ohjelmointiin tarkoitettu kirjasto. React käyttää erityistä JSX-syntaksia, joka muistuttaa sivun sisällön rakenteen kuvaamiseen käytettävää HTML-merkintäkieltä. Syntaksi on kuitenkin puhtaasti JavaScriptiä. JSX-syntaksia ei ole kuitenkaan pakko käyttää, mutta lähes aina sitä kuitenkin käytetään, kun ohjelmointiin käytetään Reactia. (React s.a.)

3.3.1 React-ohjelmointi

Reactissa voidaan välittää dataa ”propsien” avulla pääkomponentilta lapsikomponentille. Lapsikomponentti on siis komponentti, jota käytetään pääkomponentissa. Sovellus voi vain lukea propsien arvoja, mutta niiden arvoja ei voi suoraan muuttaa lapsikomponentissa. Props voi olla esimerkiksi jokin muuttuja, kuten vaikka painikkeen teksti, joka välitetään ylemmältä tasolta alemmalle tasolle, eli lapsikomponentille, joka puolestaan käsittelee tiedot ja palauttaa painikkeen, jossa on propsin välittämä teksti. Propseja voi välittää useamman ketjussa olevan lapsikomponentin läpi siihen komponenttiin, jossa tietoa tarvitsee käyttää. Pääkomponentista voi lähettää propseina myös funktioita, eli kutsuttavalle koodinpätkälle määriteltyjä nimiä, joita voi kutsua lapsikomponentista käsin toteuttamaan haluttu toiminto pääkomponentissa. (React s.a.)

Komponentin käyttämiä tietoja tallennetaan niin kutsuttujen tilojen (eng. state), avulla ja se pitää muuttujan tilan tallessa komponentissa. Muuttujan tilaa ei saa koskaan yrittää tallentaa samalla tavalla, kuin perinteisessä JavaScriptissä suoraan asettamalla arvoa muuttujaan yhtäsuuruusmerkin avulla, vaan tulee aina käyttää Reactin omaa erityistä tapaa muuttaa tilaa. Tällöin Reactin nykyaikaisessa versiossa muuttujien määrittelyssä käytetään funktiota ”useState” ja määritellään nimi, johon muuttujan arvo tallennetaan sekä funktio, jolla muuttujan tilaa voidaan muuttaa. (React s.a.) Kuvassa 9 on havainnollistettu tätä toimintoa. ”Active” on muuttuja, johon arvo tallentuu. Tässä esimerkissä active saa alkuarvoksi ”boolean”-arvon ”false”, mutta alkuarvo voisi olla ihan hyvin tyhjäkin tai vaikka merkkijono, numero, taulukko tai olio. Lisäksi esimerkissä on määritelty ”setActive” -funktio, jonka avulla kyseisen ”active”-muuttujan tilaa voi muuttaa .

```
const [active, setActive] = useState(false);
```

Kuva 9. Reactin käyttämä tilallinen muuttuja

Tilan muuttaminen on aina asynkroninen tapahtuma, mikä tarkoittaa sitä, että tila ei muutu välittömästi tilan muutoksen hoitavan funktion kutsumisen jälkeen, vaan React osaa odottaa, että kaikki asiaan liittyvät komponentit kutsuvat ensin tilan muutosta tilankäsittelijöissään. Tällä toiminnallisuudella saadaan optimoitua sovelluksen suorituskykyä. (React s.a.)

Jotta kyseistä tilan muuttavaa ”useState”-funktiota voidaan käyttää komponentissa, se täytyy ensin tuoda erikseen jokaisen komponentin saataville käyttämällä import-lausetta sen komponentin sisällä, jossa sitä halutaan käyttää. Import-lauseessa määritellään mitä tuodaan sekä polku lähteen sijainnille (kuva 10). Tällä samalla tavalla voidaan sovellukseen tuoda myös erilaisia ”paketteja”, joilla voi sovellukseen saada lisää toimintoja ilman, että kaikkea tarvitsee ohjelmoida itse alusta alkaen. Myös sovellukseen tehdyt komponentit täytyy tuoda import-lauseella niihin komponentteihin, joissa niitä halutaan käyttää.

```
import { useCallback, useState } from "react"
```

Kuva 10. Reactissa käytettävä import-lause

React osaa päivittää ja uudelleen ”renderöidä” näytöllä olevat komponentit aina kun komponenttiin liittyvä tila muuttuu. Renderöinti on React-ohjelmoinnin yhteydessä käytetty sana, joka tarkoittaa sitä, että sovellus palauttaa ennalta määritellyn, yleensä JSX-syntaksia käyttävän React-komponentin. (React s.a.)

3.3.2 Framer motion

Framer motion on vuonna 2018 valmistunut React-kirjasto, jota voi käyttää React-sovelluksissa ja Framerissa. Motion-kirjasto on avoimen lähdekoodin kirjasto (eng. open source). (Framer s.a.) Tämä tarkoittaa sitä, että kirjastoa saa kuka vain tarkastella, muokata, jakaa ja käyttää vapaasti. Avoimen lähdekoodin tuotteilla voi kuitenkin olla erilaisia lisenssejä (Opensource.com s.a.). Framer motionilla on MIT-lisenssi, mikä tarkoittaa sitä, että kuka vain voi käyttää kirjastoa haluamallaan tavalla, myös kaupalliseen tarkoitukseen. Ainoana ehtona kirjaston käyttämiselle on, että lisenssi- ja tekijänoikeustekstit sisällytetään sovelluksen lähdekoodiin. MIT-lisenssi sallii myös lisensoidun lähdekoodin yksityisen käyttämisen, jolloin sovelluksen lähdekoodia ei tarvitse julkaista julkisesti kaikkien saataville, vaikka siinä olisikin käytetty MIT-lisenssin alaista kirjastoa. (Horn 2018.)

Motion-kirjasto on tehty helpottamaan animaatioiden tekemistä sovellukseen. Sen avulla voi myös tehdä sovellukseen erilaisia käyttäjän kanssa vuorovaikutuksessa olevia eleitä sekä näyttäviä siirtymiä. Framerin suunnittelutyökalulla

tehdyt prototyypit käyttävät motion-kirjastoa taustallaan elävöittämässä sovellusta. (Framer s.a.)

Kun motion-kirjastoa käytetään React-sovelluksissa, tulee kirjasto tuoda ensin kyseisen komponentin käyttöön import-lauseella. Tämän jälkeen kirjaston käyttäminen onnistuu React-komponentissa käyttämällä tavallisen JSX-elementin edessä sanaa ”motion”. Esimerkiksi ”motion.button” tekee painikkeesta motion-kirjastolla toimivan painikkeen, jolloin sille voi syöttää erilaisia motion-kirjaston ennalta määriteltyjä propseja, joilla voi tehdä helposti mm. animaatioita, siirtymiä ja erilaisia efektejä. (Framer s.a.)

3.3.3 Next.js

Next.js on MIT-lisensoitu React framework, joka antaa React-sovellukseen sovelluskehitystä helpottavia ominaisuuksia suoraan ilman lisäasetusten tekemistä. (Next.js s.a.) Framework on siis työkalu, joka tuo valmiita ratkaisuja sovelluskehityksen nopeuttamiseen, eli se luo sovellukselle ikään kuin perustukset ennalta määritellyillä ominaisuuksilla, joista on hyvä aloittaa sovelluksen kehittäminen (Ranjan 2021).

Vaikka sovellus käyttäisi Next.js frameworkia, niin käyttöliittymän ohjelmoinnissa käytetään siltikin Reactia, eli kyseessä ei ole eri ohjelmointikieli. Next.js huolehtii tehokkaasti sovelluksissa yleisesti tarvittavista ominaisuuksista, kuten reitityksistä, kuvien optimoinnista, tiedon hakemisesta ja erilaisista integraatioista. Next.js on optimoitu sovelluksen nopeutta ajatellen, joten se soveltuu käytettäväksi niin isoissa kuin pienissäkin sovelluksissa. Esimerkiksi Ticketmaster, Lego, Marvel, Ferrari ja Vogue ja monet muut isot ja pienet yritykset käyttävät Next.js frameworkia. (Next.js s.a.)

Next.js:n voi ottaa helpoiten käyttöön luomalla suoraan Next.js-sovelluksen perinteisen React-sovelluksen sijasta. Tähän riittää, kun uuden sovelluksen luomisen yhteydessä komentokehoteikkunassa vaihtaa ”create-react-app”-lauseen tilalle ”create-next-app”. Valmis sovelluspohja on pian luotuna ja valmiina ohjelmointia varten. (Next.js s.a.)

3.4 Tyylitiedostot (Cascading Style Sheets, CSS)

Css on kieli, jolla sovellukselle luodaan sen ilme. Sillä määritellään esimerkiksi värit, fontit, välit ja asettelut. Sen avulla voidaan tehdä myös erilaisia animaatioita ja efektejä. Css:n perussyntaksi on aika yksinkertaista. HTML-koodiin lisätään jokin luokan nimi tai id, jota puolestaan käytetään css:n valitsimena. Tälle valitulle luokalle (eng. class), tai id:lle kehittäjä voi määritellä haluamansa ominaisuudet css-syntaksin mukaisesti avain–arvo-pareina. (Mozilla.org s.a.)

Luonnollisesti myös Reactin JSX-syntaksi tukee css:ää. Css:ää voidaan käyttää niin sanotusti ”inline”-tyyleinä suoraan koodissa, jolloin JSX-elementille annetaan halutut tyylit propseina käyttäen “style”-propsia tyylien määrittelyssä. Vaihtoehtoisesti myös JSX:ssä voidaan käyttää luokkia ja id:eitä, jolloin tunnuksen “class” sijasta React käyttää propsia “className”. Lisäksi Reactille on saatavilla lukuisia eri kirjastoja, joiden avuilla voidaan määritellä tyylejä eri tavoilla. (React s.a.)

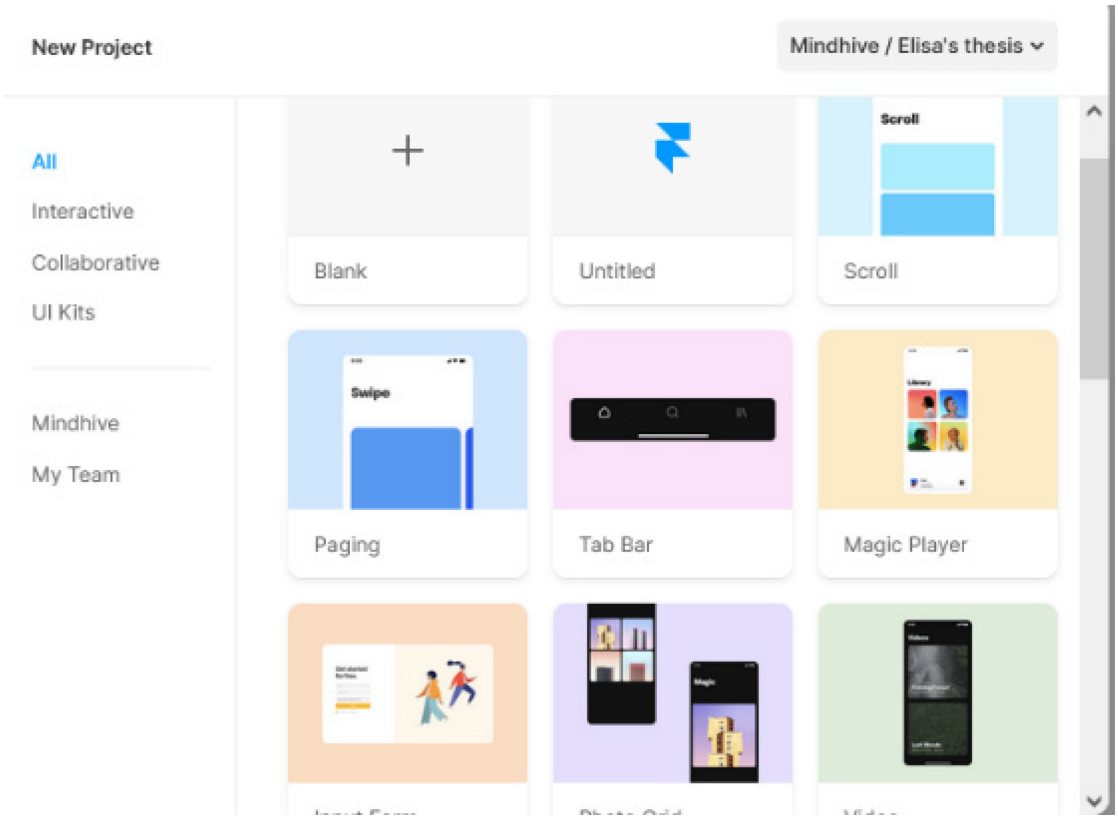
Jokaisella verkkoselaimella on olemassa omat selainkohtaiset perustyylinsä. Jos sovellukseen tai verkkosivuille ei ole määritelty minkäänlaisia css-tyylejä, on näkymä varsin koruton ja tylsä. Css:llä voidaan määritellä tarkalleen miltä sovellus tai verkkosivu näyttää. Tähän on tosin olemassa myös pieniä poikkeuksia, sillä kaikki selaimet eivät välttämättä kykene näyttämään kaikkia tyylejä samalla tavalla kuin toinen selain. (Mozilla.org s.a.)

3.5 Framerin käyttöönotto ja projektin alustus

Framerista on olemassa ilmainen versio sekä maksulliset pro- ja enterprise-versiot, joiden käyttämiseksi täytyy ostaa lisenssi. Ammattimaisessa käytössä Framerin ilmaisella versiolla ei kovinkaan hyvin pärjää, sillä ilmaisversiossa ei voi esimerkiksi luoda yksityisiä projekteja vaan kaikki luodut projektit ovat julkisesti kaikkien nähtävillä. Maksullisessa versiossa voi myös mm. tuoda Frameriin omia fontteja, tehdä tiimille omia mallipohjia sekä yksityisiä komponentteja. Opinnäytetyötä varten käytössä on maksullinen pro-versio. Frameria voi käyttää selainversiona osoitteessa framer.com tai tietokoneelle ladattavan työpöytäsovelluksen kautta. Tätä opinnäytetyötä toteutetaan Framerin selainversiolla.

Alkuvaiheessa Frameriin tulee luoda tili. Opinnäytetyötä tehdessä tilin luontiin valittiin Google-tili. Tämän jälkeen toimeksiantaja lisäsi käyttöoikeudet tiimiinsä ja lähetti sähköpostitse kutsulinkin, jonka kautta pystyi liittymään tiimin jäseneksi ja sai myös pro-ominaisuudet käyttöön, koska toimeksiantajalla oli tähän lisenssi.

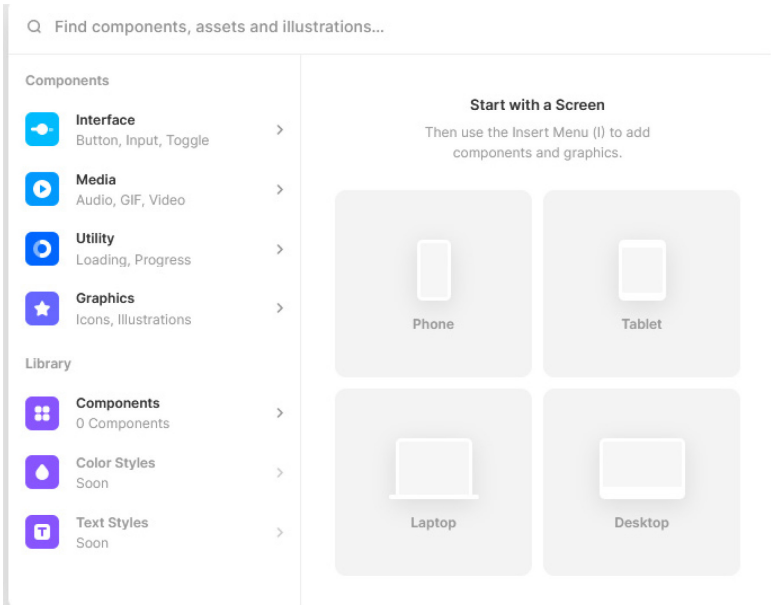
Alkuasetusten jälkeen päästään itse tutkimustyön pariin. Ensin luodaan tiimin näkymässä uusi kansio, jonka alle voi alkaa luomaan projekteja. Projekteja voi luoda joko käyttämällä valmista mallipohjaa tai tehdä kaiken itse alusta alkaen (kuva 11).



Kuva 11. Framerin valmiita mallipohjia projektille

Valmiiden mallien vaihtoehtoja on useita. Tiimit voivat myös luoda itse omia mallipohjia, jotka löytyvät valikosta kohdasta “My Team”. Tässä kohtaa kaikki tehdään itse alusta alkaen, joten valinnaksi tulee “Blank”.

Seuraavaksi avautuu uusi ikkuna, jossa on niin ikään useita eri mahdollisuuksia tehdä valintoja. Ensin tulee valittavaksi mille näyttöpäätteelle prototyyppiä halutaan suunnitella (kuva 12).



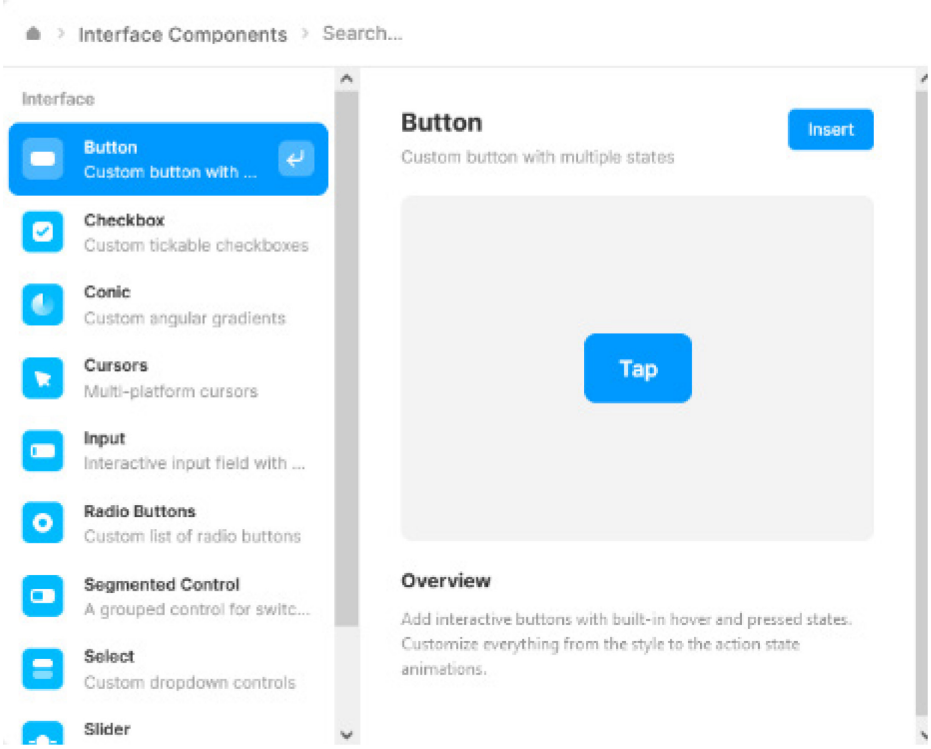
Kuva 12. Rakennettavan prototyypin näytön koon valinta

Valitaan tällä kertaa näytöksi näyttöpääte, eli desktop, jolle aletaan rakentamaan käyttöliittymän prototyyppiä.

3.6 Valmiiden komponenttien muokkaaminen koodin avulla

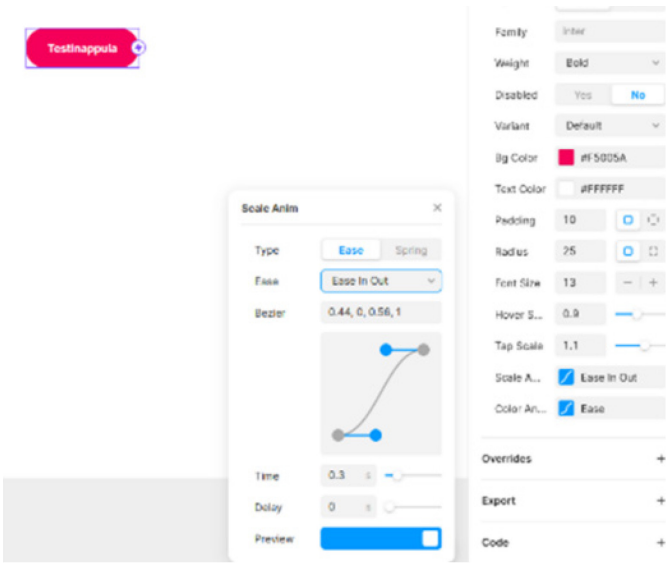
Ikkunan, josta Framerissa voi valita valmiita komponentteja, saa näkyville hallintapaneelin yläpalkissa olevasta “insert” -painikkeesta. Avautuvasta ikkunasta voi valita luodulle näyttöpohjalle lisättäväksi sisältöä, kuten painikkeita, tekstikenttiä tai muita käyttöliittymän komponentteja.

Ensiksi kokeillaan luoda Framerin valmis painikekomponentti lisäämällä valikosta oletuspainike, jota voidaan muokata (kuva 13). Framerin valmis painikekomponentti sisältää jo oletuksena älykkäitä ominaisuuksia, kuten esimerkiksi painikkeen tilan hallinnan ja erilaisia hover-efektejä, jotka tapahtuvat silloin, kun hiiren kursorin vie painikkeen päälle.



Kuva 13. Frameriin voidaan lisätä erilaisia komponentteja kirjastosta. Kuvassa lisätään painike.

Luotua painiketta voi muokata monipuolisesti näytön sivussa olevan hallintapaneelin kautta (kuva 14). Muokkaamista varten ei tarvitse osata ohjelmoida. Muokkaaminen on helppoa ja nopeaa, ja painikkeelle voi helposti tehdä pieniä animaatioita, jotka tapahtuvat silloin, kun hiiren kursorin laittaa painikkeen päälle tai kun painiketta klikkaa.

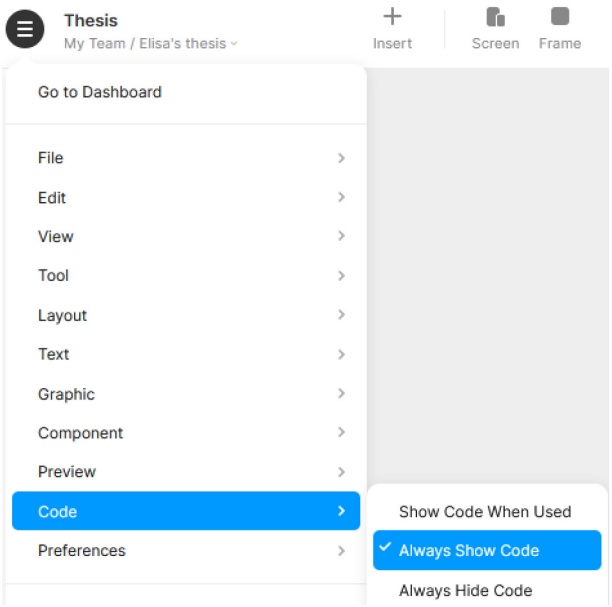


Kuva 14. Framerin hallintapaneelista voi tehdä helposti monenlaisia asetuksia komponenteille

Oikeassa sivupalkissa on myös “Code”-välilehti, josta aukeaa näkymä React-komponenttiin lähetetyistä propseista (kuva 15). Propseja ei kuitenkaan pysty tässä näkymässä muokkaamaan käsin, vaan ne päivittyvät sitä mukaa, kun asetuksia tehdään painikkeelle käyttöliittymän kautta.

Kuva 15. Hallintapaneelin "Code" -välilehdeltä näkee painikkeelle välitetyt propsit

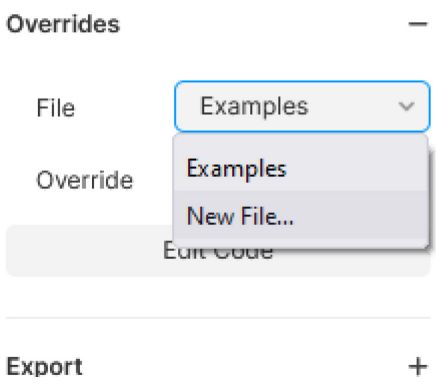
Jos painiketta haluaa muokata koodista käsin, tulee ottaa käyttöön koodiominaisuus. Ominaisuuden saa käyttöön käyttöliittymän vasemman yläkulman painikkeesta, jolloin näkyviin avautuu valikko, josta voi valita milloin koodin muokkausominaisuus on näkyvillä (kuva 16).



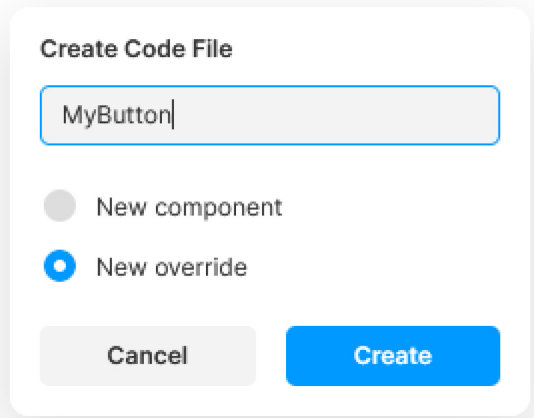
Kuva 16. Framerin koodin muokkaamisen asetukset

Kun muokkausominaisuus otetaan käyttöön, ilmestyy oikeanpuoleiseen hallintapaneeliin välilehti “Overrides”.

Seuraavaksi luodaan uusi tiedosto koodin ylikirjoitusta varten (kuvat 17–18).



Kuva 17. Uuden kooditiedoston luominen



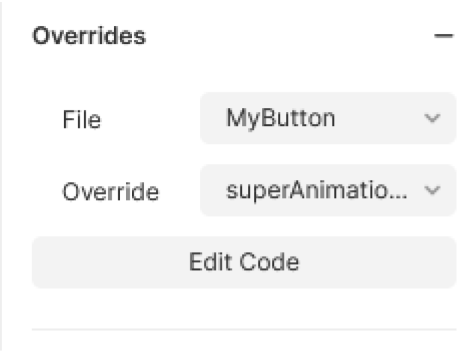
Kuva 18. Kooditiedoston nimeäminen ja koodin käyttötarkoituksen valinta

Tiedoston luomisen jälkeen tiedosto aukeaa Framerin editointitilaan, jossa koodia voi muokata suoraan. Tiedosto sisältää valmiita mallifunktioita, joita voi hyödyntää. Tiedostoon voi tehdä myös omia funktioita, joita voi liittää komponenttiin. Kuhunkin komponenttiin voi kuitenkin liittää vain yhden funktion.

Seuraavaksi tehdään testauksen vuoksi tiedostoon oma funktio, jonka nimeksi annetaan “superAnimations” (kuva 19), ja liitetään sen jälkeen juuri luomamme funktio painikkeeseen (kuva 20).

```
export function superAnimations(Component):
ComponentType {
    return (props) => {
        return (
            <Component
                {...props}
                whileHover={{ scale: 5.5 }}
                whileTap={{ rotate: 90 }}
                drag
                dragConstraints={{
                    top: -250,
                    left: -250,
                    right: 250,
                    bottom: 250,
                }}
            />
        )
    }
}
```

Kuva 19. Oma funktio, jossa toiminnallisuuksia



Kuva 20. Oman funktion liittäminen painikkeeseen hallintapaneelissa

Painike saa nyt propseina tiedot siitä, kuinka sen tulee käyttäytyä, kun kursori viedään sen päälle (whileHover) tai kun sitä klikataan (whileTap). Se saa myös “drag”-ominaisuuden, joka tarkoittaa sitä, että sitä voi raahata ja asetetut dragConstraints-ominaisuudet määrittävät rajat sille, minkä kokoisen alueen sisällä se on mahdollista.

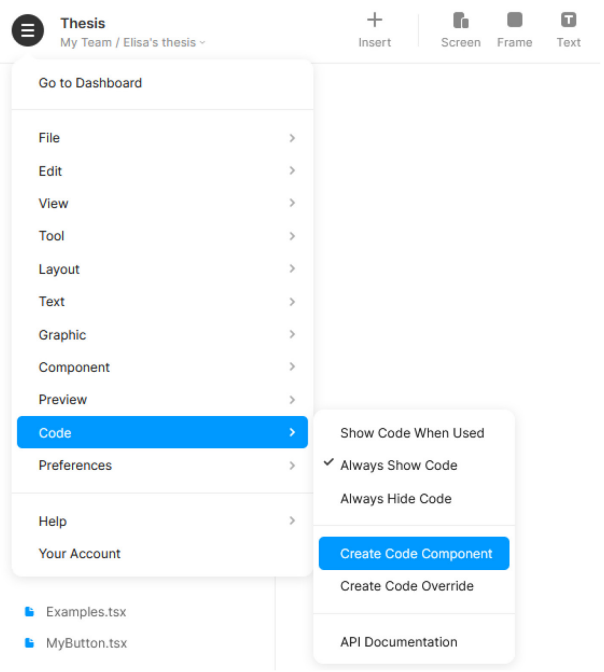
Nyt testipainike skaalautuu isommaksi, kun hiiren osoitin siirtyy sen päälle, sekä kääntyy 90 astetta (kuva 21). Painiketta voi myös raahata näytöllä ja se jää jopa kivasti liukumaan irti päästämisen jälkeen, jolloin liike näyttää sula-valta ja tyylikkäältä.

Kuva 21. Testipainike, jonka päälle on asetettu hiiren kursori

Todettakoon kuitenkin vielä selvyydeksi, että tämän komponentin muokkaaminen on tehty vain ominaisuuksien ja koodin ylikirjoittamisen demonstroimiseksi, eikä painikkeiden hyvään suunnitteluun yleisesti kuulu se, että painike tekee näin paljon temppuja, jotka eivät ole sille tyypillistä käyttäytymistä. Kuten teoriapohjassakin on jo aiemmin todettu, niin painikkeen tämän kaltainen epänormaali käyttäytyminen todennäköisesti johtaisi huonoon käyttökokemukseen.

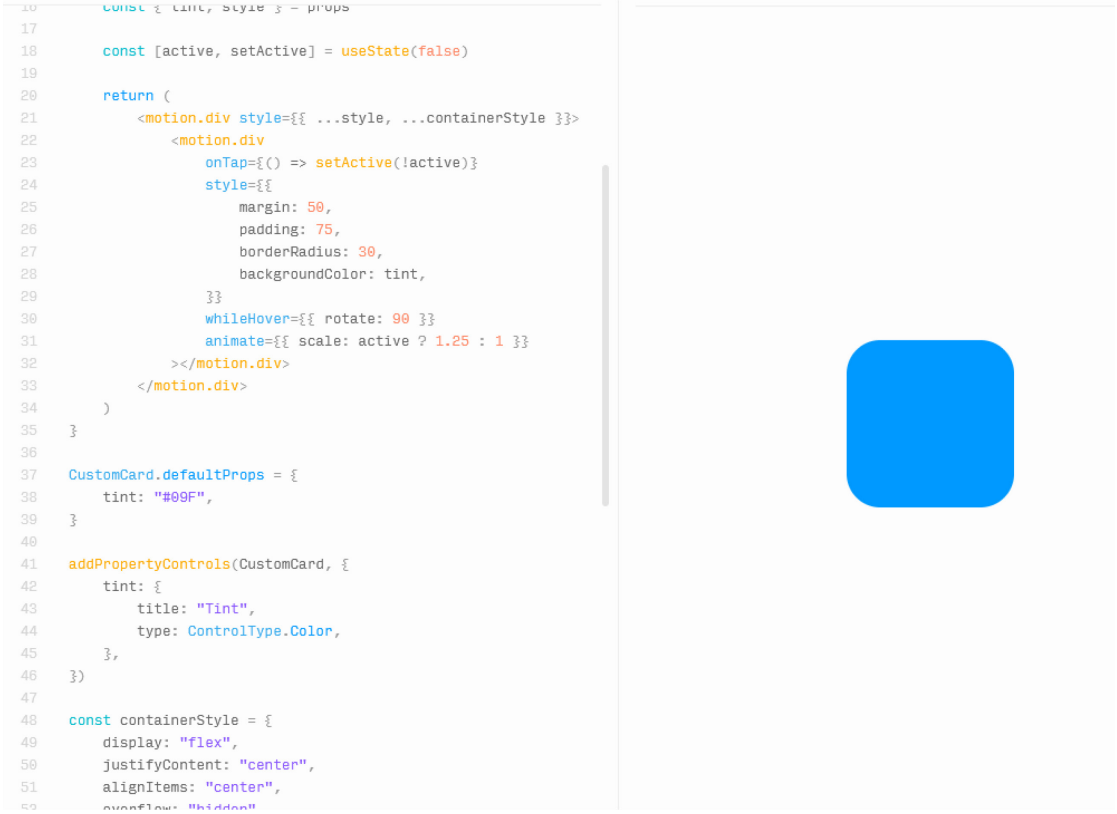
3.7 Komponentin toteutus ohjelmoimalla

Komponentteja voi Framerissa luoda myös itse ohjelmoiden. Seuraavaksi kokeilemme luoda interaktiivista korttikomponenttia itse koodaten. Koodikomponentin voi luoda valikosta valitsemalla “Code” ja avautuvasta alavalikosta “Create Code Component” (kuva 22). Oman komponentin luomiseksi ei tarvitse erillistä koodieditoria, vaan ohjelmointi tapahtuu suoraan Framerin omalla editorilla.



Kuva 22. Oman koodikomponentin luominen

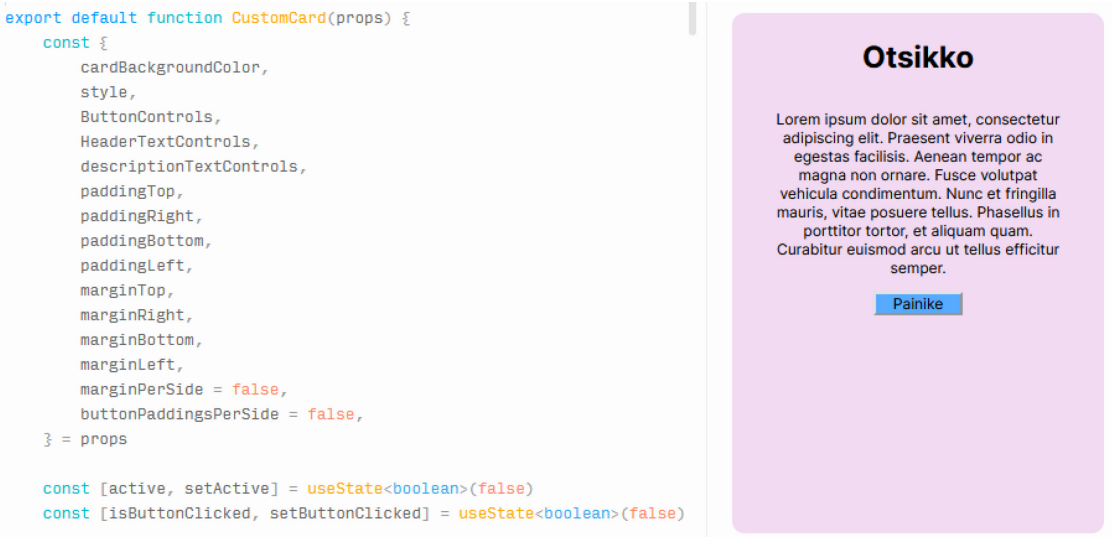
Komponentin nimeämisen jälkeen näyttöön avautuu mallikoodi, jota voi muokata haluamakseen. Mallikoodissa on kulmista pyöristetty neliö, joka pyörähtää 90 astetta, kun hiiri asetetaan päälle. Lisäksi neliö suurentuu, kun sitä klikkaa ja pienentyy taas takaisin, kun sitä klikkaa uudestaan (kuva 23).



Kuva 23. Koodikomponentin lisäämisen jälkeen avautuva mallinäkymä React-koodilla

Myös käyttäjän itsensä luomille komponenteille on mahdollista lisätä erilaisia ominaisuuksien säätömahdollisuuksia hallintapaneeliin. Käyttäjä voi itse määrittää millaisia ominaisuuksia tahtoo komponentilleen välittää. Ominaisuuksien säädöt paneeliin saadaan lisättyä “addPropertyControls”-funktion avulla. Nimet funktion vastaanottamalle objektille voi määritellä itse, mutta “ControlType:n”, eli kontrollin tyypin tulee olla ominaisuudelle tarkoituksenmukainen. (Framer s.a.)

Seuraavaksi ohjelmoidaan itse korttipohja, jossa on otsikko, tekstiä ja painike. Kuvassa 24 on korttipohja oletusasetuksilla. Korttipohjassa itsessään on hover-efekti, joten kun käyttäjä asettaa hiiren kursorin korttipohjan päälle, kortin taustalle ilmestyy harmaalla sumennettua reunustaa, joka tulee kortin yli ja kortista tulee vaikutelma, niin kuin se vähän irtoaisi taustastaan. Tämä mielestäni luo sellaista vuorovaikutteista mielikuvaa käyttäjälle, että kortti kutsuu klikkaamaan itseään. Kun korttia klikkaa, se muuttuu suuremmaksi ja tulee näin ollen ikään kuin paremmin esille ja luettavaksi käyttäjälle. Kun korttia klikkaa uudestaan, se palautuu samankokoiseksi kuin oli aiemmin. Painikkeessa on myös hover-efekti, joka kutsuu käyttäjää klikkaamaan sitä. Painike muuttuu hieman isommaksi, kun hiiren vie sen päälle.



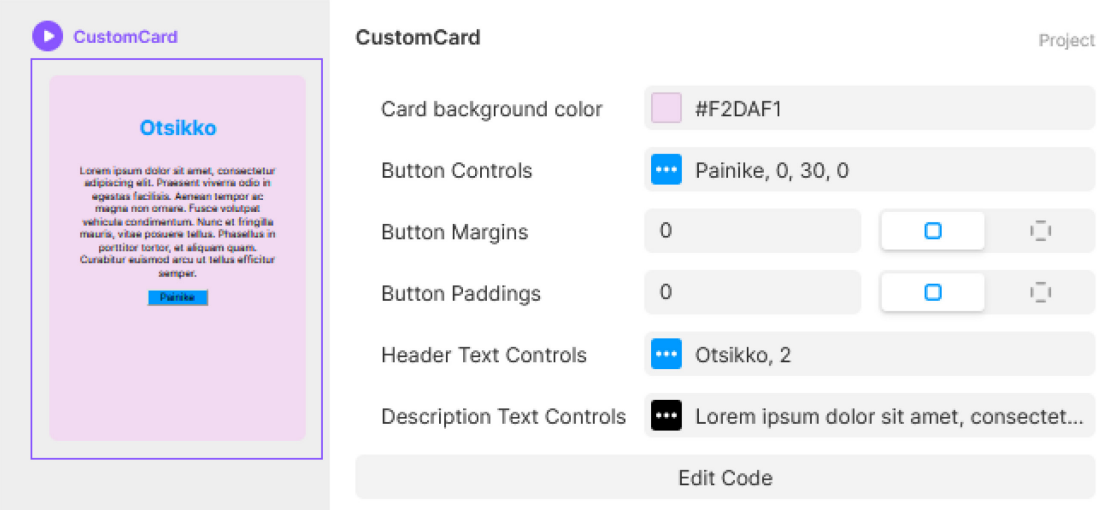
Kuva 24. Vasemmalla puolella asiat, joihin käyttäjä voi vaikuttaa hallintapaneelista käsin. Oikealla puolella kortti oletusasetuksilla.

Käyttäjällä on aika monipuoliset mahdollisuudet vaikuttaa kortin sisältöön ja ulkonäköön. Käyttäjä voi mm. valita kortin taustavärin, otsikon tekstin, kirjasinkoon ja -värin. Sen lisäksi käyttäjä voi asettaa tekstin. Painikkeelle käyttäjä voi asettaa tekstin, ulko- ja sisämarginaalit, kirjasinkoon, värin, pituuden, sekä reunojen pyöristyksen. Kuvassa 25 esimerkki painikkeen koodista, jossa käyttäjän on mahdollista vaikuttaa propseina kaikkiin näihin painikkeen tyyleihin. Nämä propsit löytyvät kohdasta “style” ja sen alapuolella olevista ominaisuuksista, joissa on css-syntaksin mukaisesti kuvattu ensin ominaisuuden nimi, kuten esimerkiksi “width”, eli pituus ja kaksoispisteen jälkeen tulee arvo, jonka käyttäjä voi valita suoraan hallintapaneelista, joten koodaustaitoa tämän komponentin käyttämiseen ei tarvitse enää sen jälkeen, kun komponentti on valmiina. Huomion arvoista on myös, että painikkeelle voi asettaa myös asemerkiksi onClick-funktiokutsun, joka aktivoituu painiketta painettaessa. Käyttäjä voi asettaa painikkeelle hallintapaneelista myös “key:n”- eli avain arvon, jonka perusteella React osaa erotella eri painikkeet toisistaan.

```
<motion.button
    key={ButtonControls.key}
    style={{
        display: "flex",
        justifyContent: "center",
        alignItems: "center",
        backgroundColor: ButtonControls.color,
        borderRadius: ButtonControls.radius,
        width: `${ButtonControls.width}%`,
        margin: `${marginTop}px ${marginRight}px
        ${marginBottom}px ${marginLeft}px`,
        padding: `${paddingTop}px ${paddingRight}px
        ${paddingBottom}px ${paddingLeft}px`,
    }}
    onClick={buttonClicked}
    whileHover={{ scale: 1.2 }}
>
    {ButtonControls.text}
</motion.button>
```

Kuva 25. Esimerkki kortin painikkeen tyylimäärityksistä

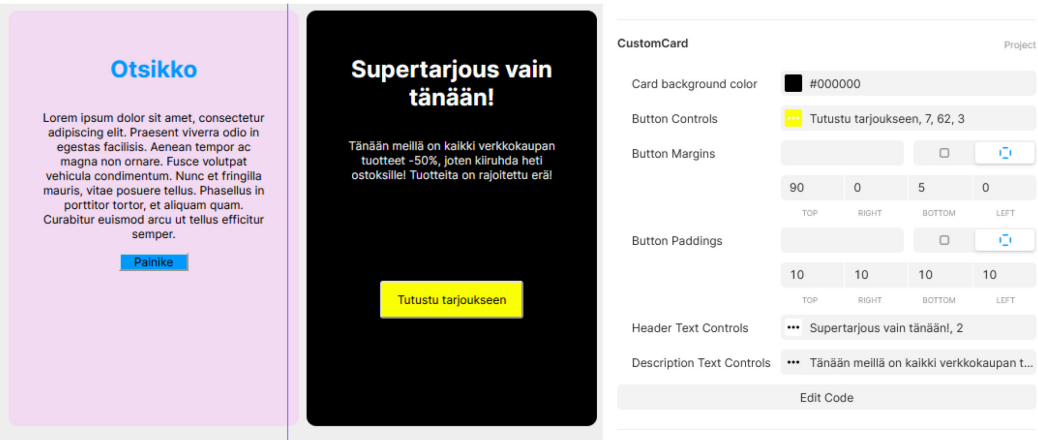
Kun katsotaan luotua korttikomponenttia hallintapaneelin kautta, koodin puolella asetetut muokkausmahdollisuudet ovat nyt näkyvillä (kuva 26). Kuvassa kortille ei ole asetettu hallintapaneelin kautta vielä arvoja, eli kyseessä on kortti oletusarvoilla. Tältä kortti näyttää siis joka kerta, kun projektiin luodaan uusi korttikomponentti.



Kuva 26. Koodin avulla hallintapaneeliin luodut kortin muokkausmahdollisuudet

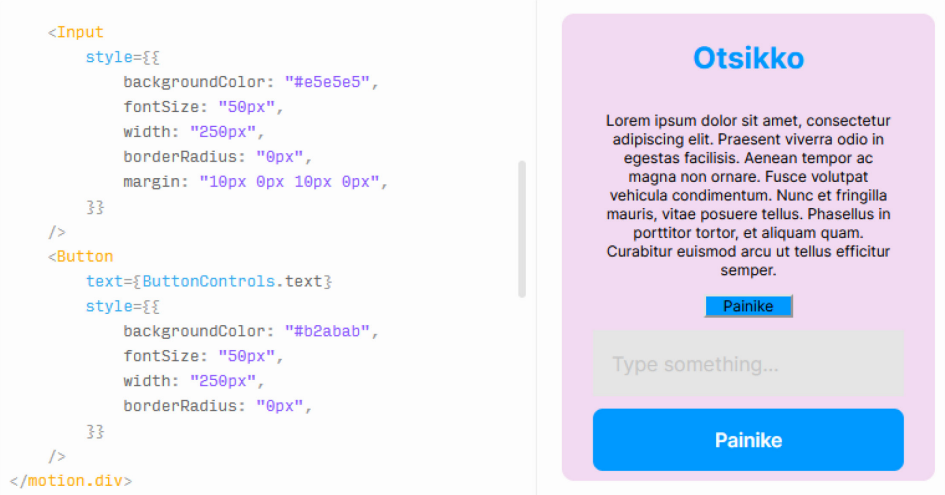
Seuraavaksi tehdään muokkauksia oletuskortille. Suunnittelijan on helppo ja nopea nähdä korttiin tulevat muutokset reaaliajassa muokkauksen

yhteydessä. Kortti näyttää ja tuntuu muokattuna jo aivan erilaiselta kuin oletuspohja, mutta molemmilla korteilla on silti yhteneväisiä interaktiivisia ominaisuuksia, kuten hover-efektit ja skaalautuvuus klikatessa (kuva 27). Nyt projektiin voi luoda korttipohjia haluamansa määrän ja muokata niiden ulkonäköä ja sisältöä haluamakseen, tai vaikka kopioida suoraan jo muokattua korttia ja vaihtaa siihen vain eri tekstikenttien sisällöt.



Kuva 27. Peruskortti ja hallintapaneelin kautta muokattu kortti

Tämän tyyppisessä itse ohjelmoidussa komponentissa voi halutessaan myös käyttää Framerin omia komponentteja. Framerin komponentin tuominen omaan komponenttiin on helppoa. Kun ollaan Framerin koodieditori-näkymässä, ylhäällä olevasta “import”-painikkeesta avautuu ikkuna, josta voi valita haluamansa komponentin. Ikkunan kautta voi valita myös itsetehdyistä komponenteista haluamansa. Tutkimuksen aikana testattiin tuoda valmiit tekstikenttä- ja painikekomponentit ja molemmat tulivat nätisti itse ohjelmoituun korttikomponenttiin (kuva 28).



Kuva 28. Framerin oma tekstikenttä ja painike tuotuna koodikomponenttiin

Hallintapaneelin editoriin ei kuitenkaan tullut import-lauseen avulla lisätyille painike- ja tekstikenttäkomponenteille minkäänlaisia hallintamahdollisuuksia. Tämä tarkoittaa sitä, että myös oman komponentin sisälle tuoduille komponenteille tulee ohjelmoida erikseen muokkausmahdollisuudet hallintapaneeliin.

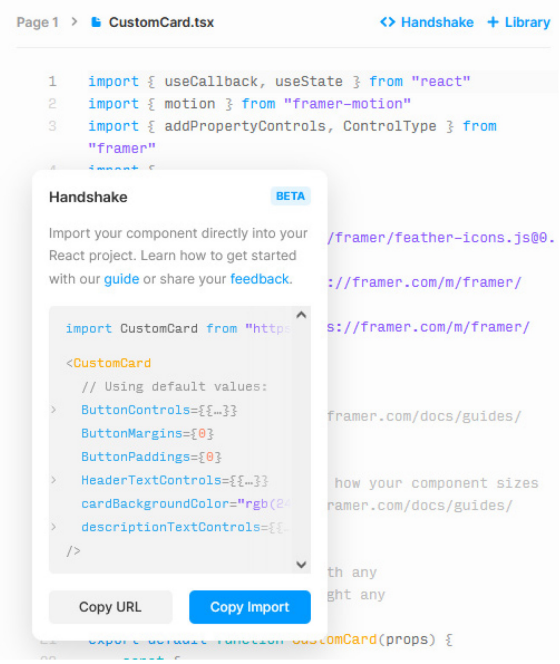
3.8 Framer-komponenttien integroiminen sovellukseen

Seuraavaksi tutkitaan, kuinka Framerin komponentteja saadaan tuotua React-sovellukseen. Frameria kehitetään koko ajan paremmaksi ja React-sovellukseen tuonnin ominaisuus oli vielä tätä opinnäytetyötä tehdessä beta-versio, joten tämä ominaisuus tulee todennäköisesti vielä muuttumaan ajan saatossa.

Framerin dokumentaation mukaan komponentit eivät olisi suoraan vietävissä Reactilla tehtyyn sovellukseen, koska Framerin omat komponentit, kuten Frame, Stack, Scroll ja Page eivät ole yhteensopivia tavallisen React-sovelluksen kanssa, joten jos koodia haluaisi käyttää suoraan sovelluksessa, kyseiset osat tulisi korvata tavallisella HTML-merkintäkielellä. Näin ollen Framerilla luotuja komponentteja kannattaisi Framerin mukaan käyttää siten, että ne tuodaan varsinaiseen sovellukseen hyödyntämällä import-lauseita, jotka luodaan Framerin ”handshake”-toiminnolla. (Framer s.a.)

Kyseinen ”handshake”-painike löytyy Framerin koodieditorin yläpalkista. Kun painiketta klikkaa, avautuu näytölle ikkuna, jossa voi joko kopioida pelkän

komponentin Frameriin osoittavan osoitteen tai vaihtoehtoisesti suoraan Reactissa käytettävän import-lauseen (kuva 29).



Kuva 29. Framerin "handshake"-toiminto komponentin integroimiseksi osaksi React-sovellusta.

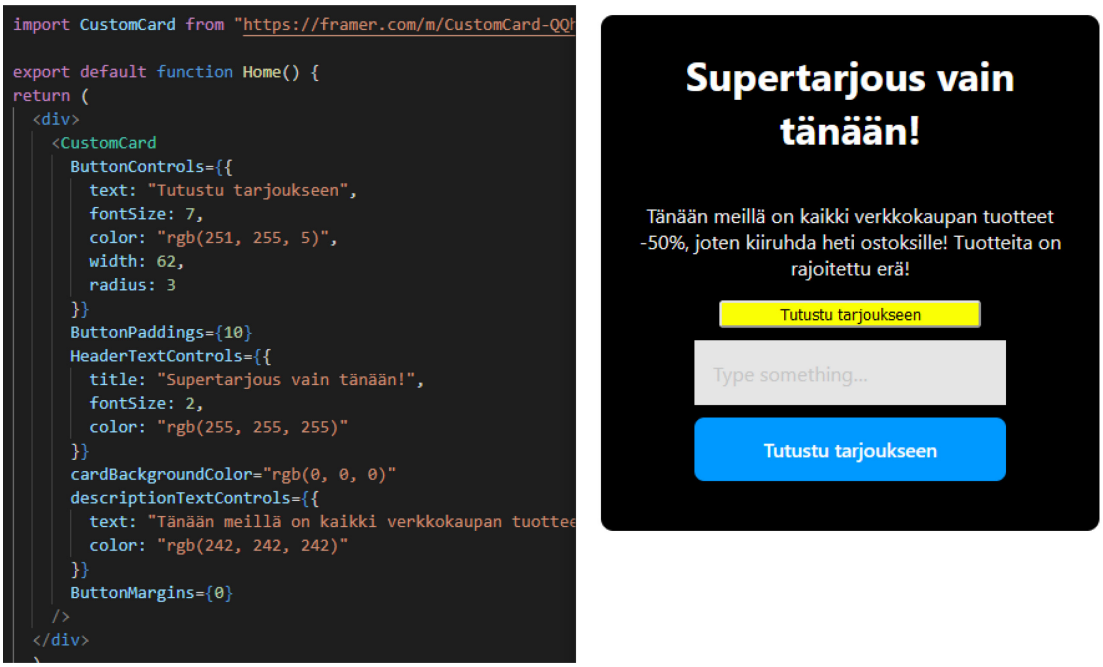
Framerin tutoriaalin mukaan projektiin tulee myös asentaa ”framer”-, ”framer-motion”- ja ”next”-paketit. Ensimmäisenä testattiin tuoda komponenttia ohjeiden mukaisesti Framerista React-sovellukseen, jossa ei ollut käytössä Next.js frameworkia. Virheilmoituksia ei ilmaantunut Framerin pakettien asentamisen jälkeen, mutta sovellus ei myöskään näyttänyt komponenttia, joten sivu oli täysin valkoinen.

Koska Framerin tutoriaalissa on maininta vaatimuksista Next.js -ympäristölle, ja ohjeista jää hieman auki se, että pitäisikö tuonnin onnistua myös perinteiseen React-sovellukseen, niin luotiin myös Next.js-sovellus, johon testattiin komponentin tuontia. Framerin vaatimuksena Next.js-sovellukselle on, että tuonti sallitaan verkko-osoitteesta ja next.config.js-tiedostoon tulee lisätä nämä asetukset.

Next.js-versioon komponentin tuominen onnistui riippuvuuksien asentamisen, ja tarvittavien asetusten tekemisen jälkeen. Komponentin tuomista varten komponentin propsit täytyy kopioida Framerin hallintapaneelista kohdasta

”code”, joka onkin jo esitelty aiemmin tämän opinnäytetyön yhteydessä (kuva 15).

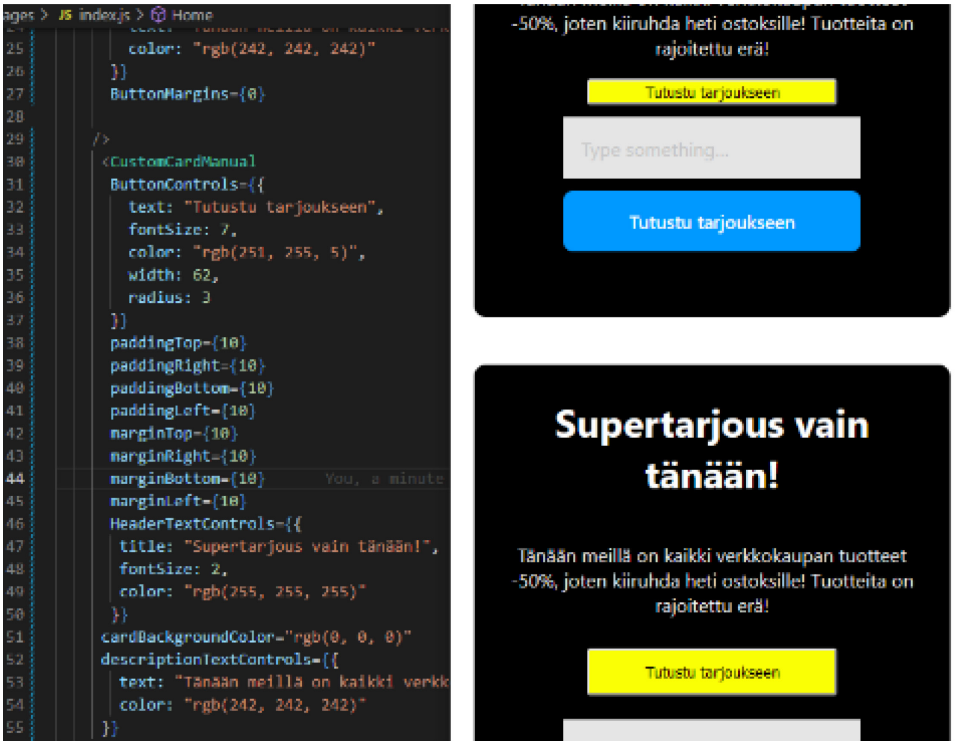
Vaikka tuonti näennäisesti onnistuikin, niin painikkeen marginaalit ja välistykset eivät tulleet mukaan, vaikka nämä näkyvätkin täysin oikein Framerin hallintapaneelissa (kuva 30). Tässä kohtaa vika on todennäköisesti Framerissa, sillä muokkausyrityksetkään eivät auttaneet asiaa. Ominaisuus on todennäköisesti vielä sen verran keskeneräinen, että se ei tue tämän tyyppistä tuontia, kuin mitä kyseisen komponentin painikkeessa on käytetty. Huomion arvoista on myös se, että aina kun Framerissa tekee muutoksia komponenttiin, täytyy Reactin import-lause päivittää. Jokaiselle versiolle generoituu uusi osoite, eikä näin ollen Framerin puolella tehdyt muutokset ole suoraan synkronisoituna sovellukseen.



Kuva 30. Import-lauseella tuotu itse tekemä korttikomponentti ja Framerista kopioidut propsit

Seuraavaksi kokeillaan korttikomponentin tuontia suoraan kopioimalla koodia Framerista React (Next.js)-sovellukseen. Tällöin sovellukseen luodaan käsin uusi komponentti ja kopioidaan Framerin puolelta koodit suoraan luotuun komponenttiin. Virheilmoituksia ei tule, koska tarvittavat paketit on jo asennettu, joten tämän jälkeen komponenttia voi käyttää sovelluksessa samalla tavalla, kuin komponentteja normaalisti käytetään React-sovelluksissa. Komponentti näyttää samalta, kuin suoraan osoitteen kautta Framerista tuotu komponentti.

Kopioimisen etuna näkisin sen, että komponenttia voi nyt muokata suoraan sovelluksessa käsin, kuten esimerkiksi painiketta, joka ei näytä tuotuna samalta, kuin Framerin suunnittelutyökalussa. Kopioidusta komponentista voi myös poistaa ”addPropertyControls”-kutsun, sillä se on Frameria varten, joten React-sovelluksessa sillä ei ole mitään käyttöarvoa. Nyt kun koodin muokkaaminen suoraan sovelluksessa on mahdollista, niin painikkeen saa taas helposti näyttämään samanlaiselta, kuin Framerilla luodussa prototyypissä (kuva 31). Iloksemme voimme myös huomata, että myös animaatiot toimivat samalla tavalla React-sovelluksessa, kuin Framerin suunnitelmassa.



Kuva 31. Oikealla ylhäällä import-lauseella tuotu korttikomponentti, alhaalla koodia kopioimalla tuotu komponentti, jota pystyy helposti muokkaamaan suoraan React-sovelluksessa.

Tavallisessa React-sovelluksessa, jossa ei ollut Next.js frameworkia käytössä, oli myös paljon erilaisia yhteensopivuusongelmia mm. Framerin pakettien kanssa. Näitä ongelmia ei kuitenkaan lähdetty tässä vaiheessa ratkomaan sen enempää, koska toimeksiantaja käyttää Next.js-frameworkia React-sovelluksissaan, niin opinnäytetyön tutkimuksen tavoitteen voidaan katsoa täyttyneen sillä, että juuri Next.js-sovellukseen komponenttien tuominen Framerista on saatu testattua.

Seuraavaksi kokeillaan vielä tuoda kopioimalla sovellukseemme Framerin valmiista pohjasta muokattu komponentti, eli painike, joka tehtiin ensimmäisenä. Kun painiketta yrittää tuoda osoitteen kautta samaan tapaan kuin itse tehtyä komponenttia, sovellus kaatuu virheilmoitukseen. Framer antaa import-lauseeseen mukaan vain painikkeeseen liitetyt funktiokutsut. Näin ollen näyttäisi siltä, että todellisuudessa vain itse tehtyjä komponentteja on mahdollista tuoda sovellukseen. Framerin omien komponenttien koodiin ei pääse myöskään käsiksi Framerin koodieditorista, joten komponenttien koodien kopioiminenkaan suoraan sieltä ei onnistu.

3.9 Tulokset ja analyysi

Framer näyttäisi olevan hyvä työkalu interaktiivisten sovellusten käyttöliittymien suunnitteluun. Perustehtävänsä se hoitaa erinomaisesti ja on hienoa, että peruskomponentteja voi myös muokata koodin avulla haluamakseen, sekä tehdä itse täysin omiakin komponentteja.

Kuitenkaan Framer ei ollut niin joustava ohjelmoijan ja suunnittelijan välisen kuilun kaventamisessa, kuin olisin odottanut. Komponenttien tuominen sovellukseen ei ollut täysin mutkatonta, ja näyttäisi siltä, että komponenttien tuominen osaksi sovelluskoodia olisi järkevintä tehdä suoraan koodia kopioimalla, jolloin komponenttia voi vielä oikeasti hyödyntää ja muokata sovelluksessa itsessään, eikä se ole enää sen jälkeen riippuvainen Framerista.

Koodin suoraan kopioimista mielestäni puoltaa myös oma pohdintani siitä, että voisiko jossain vaiheessa käydä niin, että Framerista tuotu komponentti katoaa ja import-lauseella tuotu komponentti ei enää toimisi. Vai lupaako Framer niiden olevan “ikuisesti” saatavilla ja muuttumattomina. Mielestäni import-lauseiden käyttäminen sisältää ison riskin tässä suhteessa. Olisi ikävää, jos tämä uhkakuva realisoituisi, jolloin oltaisiin aika huonossa tilanteessa. Vielä huonommassa tilanteessa oltaisiin, jos sama tapahtuisi useammankin eri asiakkaan sovelluksessa. Mielestäni on siis parempi, että komponentin koodit ovat turvassa React-sovelluksen sisällä.

Sitten päästäänkin tähän rajoitukseen, että koodeja voi suoraan kopioida vain itse koodatuista komponenteista. Käytännössä tämä tarkoittaisi siis sitä, että

ohjelmoijien tulisi ensin ohjelmoida Frameriin oma komponenttikirjastopohja, jota suunnittelija voi sitten käyttää prototyyppejä tehdessään hallintapaneelin kautta ilman ohjelmointiosaamista. Tämän jälkeen ohjelmoijat voisivat käyttää suoraan näitä komponentteja suunnittelijan asettamilla arvoilla sovelluksessa.

Sinänsä yllä kuvattu lähestymistapa Framerin käyttöön voisi olla ihan varteenotettava. Vaikka oman kirjaston luomiseen menee oma aikansa, niin jatkossa näitä komponentteja voisi hyödyntää useassa eri projektissa erilaisilla asetuksilla, jolloin koodit saataisiin suoraan Framerista. Tämä taas oletettavasti nopeuttaisi ohjelmointityötä siinä kohtaa. Joten lopputulemana en näkisi tällä hetkellä Framerin käyttöön ottamisen tuovan toimeksiantajalle välitöntä ohjelmoinnin prosessin nopeutumista, vaan mahdollisesti vasta ajan kuluessa.
On myös toki oletettavaa, että Framerin toiminnot koodin suoraan viemiseksi React-sovellukseen tulevat vielä kehittymään paljon vuosien varrella. Muutoksia kannattaa pitää silmällä ja perehtyä niihin aika-ajoin. Olisi hienoa, jos Framerista voisi kopioida kerralla kokonaisen sivun verran koodia, jossa kaikki komponentit olisivat samalla tavalla kuin prototyypissä, mutta tästä ollaan valitettavasti vielä aika kaukana tällä hetkellä.

Viimeisenä huomiona vielä sanoisin, että Framerin opuksia lukiessani asiat oli mielestäni myös esitelty varsin pintapuolisesti. Löysin myös Framerin tekemän opaskirjan Framerin käyttämisestä, josta olisi varmasti saanut paljon tietoa, mutta tuo kirja oli vuosimaksullinen, joten en päässyt sitä lukemaan. Ilmainen dokumentaatio on todennäköisesti tehty tarkoituksella hieman pinnalliseksi, jotta käyttäjät ostaisivat kyseisen maksullisen oppaan.

4 YHTEENVETO

Opinnäytetyön prosessi eteni mielestäni järkevällä tavalla. Ensin tutustuin aiheeseen liittyvään teoriaan ja haastattelin toimeksiantajaa. Sen jälkeen alkoi käytännön osuus, jota varten minun tuli opetella käyttämään Frameria, koska en ollut sitä juurikaan käyttänyt aiemmin. Tietoa löytyi Framerin käyttämisen opetteluun löytyi ihan hyvin, etenkin peruskäytöstä.

Asetettuun ongelmaan lähdin hakemaan ratkaisua tekemällä käytännössä Frameriin interaktiivisia komponentteja ja tutkimalla, kuinka niitä saisi vietyä

mahdollisimman muuttumattomina ja käytettävinä Framerista osaksi React-sovellusta. Minut itseasiassa vähän yllätti se, että Framerista komponenttien tuominen ei ollutkaan niin joustavaa, kuin mitä olin Framerin myyntipuheita lukiessani ajatellut. Tässä mielessä tekemäni tutkimuksen tulos oli minulle hieman pettymys.

Oikeastaan ainoaksi potentiaaliseksi vastaukseksi suunnittelijan ja ohjelmoijien välisen kuilun kaventamiseksi jäi tällä hetkellä oman komponenttikirjaston rakentaminen, joka luonnollisesti vie ensin aikaa, mutta mahdollisesti palkitsee jatkossa, kun kerran tehtyjä omia komponentteja voisi käyttää Framerissa useassa eri projektissa. Itsetehtyjen komponenttien koodit voi kopioida suoraan osaksi React-sovellusta, jolloin ne eivät olisi enää riippuvaisia Framerin suunnittelutyökalusta, mutta olisivat silti riippuvaisia toki mm. framer- ja framer motion-kirjastoista.

Tuloksen luotettavuuteen voi vaikuttaa osaltaan se, että mielestäni Framerin dokumentaatio oli ehkä hieman pintapuolista, enkä tilannut maksullista syvempää opaskirjaa, josta olisi saattanut löytyä ratkaisuja kohtaamiini ongelmiin tai vielä parempia ratkaisua esimerkiksi koodien ja komponenttien viemiseksi React-sovellukseen.

En kuitenkaan näe mitään muuta vaihtoehtoa, miten olisin voinut tutkia asetettua ongelmaa toisella tapaa, kuin tekemällä komponentteja itse ja testaamalla erilaisia asioita, samalla tietysti Framerin dokumentaatiota lukien. Kun on asioita kokeillut ja todennut itse, on niitä helpompi ymmärtää. Siinä mielessä en tekisi mitään toisin opinnäytetyön suhteen.

Jatkotutkimusta ajatellen voisi olla hyvä tilata Framerin tarjoama maksullinen opaskirja käyttöön, josta voisi saada syvempiä tietoja Framerilla ohjelmoinnista, sillä tarkempaa tietoa siitä oli aika huonosti saatavilla. Tämä johtuu todennäköisesti siitä, että ominaisuus taitaa olla sen verran uusi, että sillä ei ole vielä niin paljoa käyttäjiä, että erilaisia ohjeita olisi ilmestynyt saataville blogeihin ja erilaisille Framerin ulkopuolisille tutoriaalisivuille.

Opinnäytetyön tekeminen opetti minulle Framerin työkalun käyttämistä sekä avasi uusia maailmoja käyttöliittymän suunnittelusta ja etenkin

vuorovaikutussuunnittelusta, jota pidän nyt itsekin tämän opinnäytetyön tekemisen jälkeen tärkeämpänä, kuin ennen aloitusta. On tärkeää, että sovellus näyttää hyvältä, tietenkin sen lisäksi, että se myös toimii kuten pitää. Opinnäytetyön tekemisen jälkeen olen alkanut pistämään enemmän merkille sitä, miten eri sovellukset ja sen osat ovat vuorovaikutuksessa kanssani, kun käytän niitä ja olen jopa saattanut miettiä, että miten vuorovaikutusta voisi vielä edelleen parantaa.

Toivon että tekemästäni tutkimuksesta on hyötyä kaikille niille, jotka miettivät Framerin mahdollista potentiaalia käyttöliittymäsuunnittelun ja sovelluskehityksen välisen kuilun kaventamissa. Mutta koska Frameria kehitetään edelleen ja koodien vieminen React-sovellukseen on vasta beta-vaiheessa, niin tutkimustulokseni voivat olla hyvinkin vanhentuneita jo vaikka vuoden päästä tai jopa aiemmin, joten lukijan ei tule tuudittautua siihen ajatukseen, että tämän tutkimuksen tulokset olisivat kovin pitkään paikkansapitäviä. Näin ollen kaikkien, jotka miettivät Frameria vaihtoehtona tähän tarkoitukseen, kannattaa tehdä omaa tutkimustyötä ennen sen käyttöönottamista.

Toimeksiantaja taas sai tutkimuksestani ajantasaista hyötyä tässä hetkessä, joten he voivat miettiä mahdollista seuraavaa askelta Framerin komponenttien React-sovellukseen integroimisen suhteen. Eli käytännössä toimeksiantajan pohdittavaksi jäi, aloittaako toimeksiantaja oman komponenttikirjaston rakentamisen, odottaako siihen hetkeen, että Framer on kehitetty joustavammaksi komponenttien tuomisen suhteen vai hylkääkö toimeksiantaja koko ajatuksen.

Esittelin opinnäytetyön tulokset toimeksiantajalle, joka oli tyytyväinen tekemääni tutkimukseen. Toimeksiantajan mielestä oman komponenttikirjaston rakentaminen olisi juuri sellainen ratkaisu, johon todennäköisesti joka tapauksessa pyrittäisiin. On siis paljon mahdollista, että toimeksiantaja alkaa rakentamaan omia komponentteja Frameriin ja ryhtyy näin hyödyntämään Frameria vuorovaikutteisten käyttöliittymien suunnittelutyökaluna sekä nopeuttamaan sovelluskehityksen prosessia tuomalla komponenttien koodeja suoraan kehitettäviin React-sovelluksiin.

LÄHTEET

Cooper, A., Reinmann, R., Cronin, D., Noelssel, C., Csizmadi, J. & LeMoine, D. 2014. About face: The essentials of interaction design. Neljäs painos. Indianapolis, Indiana: John Wiley & Sons.

Framer. s.a. Documentation. WWW-dokumentti. Saatavissa: https://www.framer.com/docs/ [viitattu 5.3.2022].

Horn, N. 2018. The MIT License (MIT). WWW-dokumentti. Saatavissa: https://github.com/framer/motion/blob/main/LICENSE.md [viitattu 12.4.2022].

Kachhawa, H & Tiwari, A. 2021. Usability Study on User Interface and User Experience Design Patterns. PDF-dokumentti. Saatavissa: https://www.wjrr.org/download_data/WJRR1206013.pdf [viitattu 21.4.2022].

Levinson, D & Belton, T. 2017. Build your first web app. New York: Sterling Publishing Co., Inc.

Mozilla.org. s.a. What is CSS?. WWW-dokumentti. Saatavissa: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS [viitattu 12.4.3.2022].

Next.js. s.a. Documentation. WWW-dokumentti. Saatavissa: https://nextjs.org/docs [viitattu 15.4.2022].

Opensource.com. s.a. What is open source?. WWW-dokumentti. Saatavissa: https://opensource.com/resources/what-open-source [viitattu 12.4.2022].

Ranjan, R. 2021. What is a Framework in Programming & Why You Should Use One?. WWW-dokumentti. Saatavissa: https://www.netsolutions.com/insights/what-is-a-framework-in-programming/ [viitattu 16.4.2022].

React. s.a. Documentation. WWW-dokumentti. Saatavissa: https://reactjs.org/docs/getting-started.html [viitattu 20.3.2022].

Saariluoma, P., Kujala, T., Kuuva, S., Kymäläinen, T., Leikas, J., Liikkanen, L. & Oulasvirta, A. 2010. Ihminen ja teknologia. Hyvän vuorovaikutuksen suunnittelu. Helsinki: Otava.

Steane, J & Yee, J. 2018. Interaction design from concept to completion. London, UK; New York, NY, USA: Bloomsbury Visual Arts.

Domzalski, S. 2021. 6 reasons you shouldn’t use paper wireframes. WWW-dokumentti. Saatavissa: https://bootcamp.uxdesign.cc/6-reasons-you-shouldnt-use-paper-wireframes-df47114f192f/ [viitattu 19.3.2022].

Venäläinen, V. 2022. Toimitusjohtaja. Haastattelu 21.3.2022. Mindhive Oy.

KUVALUETTELO