Ville-Petteri Sassi

KOMPONENTTISUUNNITTELU FRAMERIN JA STORYBOOKIN AVULLA

Opinnäytetyö

Liiketalouden ammattikorkeakoulututkinto

Tietojenkäsittelyn koulutus

2020

Kaakkois-Suomen ammattikorkeakoulu

Kaakkois-Suomen ammattikorkeakoulu

Tekijä/Tekijät	Tutkintonimike	Aika
Ville-Petteri Sassi	Tradenomi (AMK)	Marraskuu 2020
Opinnäytetyön nimi		
Komponenttisuunnittelu Framerin ja Storybookin avulla		45 sivua
Toimeksiantaja		
Mindhive Oy		
Ohjaaja		
Janne Turunen		

Tiivistelmä

Tämän opinnäytetyön aiheena on käyttöliittymäkomponenttien suunnittelu- ja luontiprosessi, jossa sovelletaan TypeScript-ohjelmointikieltä yhdessä React-komponenttikirjaston tarjoamien työkalujen ja komponenttien kanssa. Työn aikana käytetään edellä mainittujen tekniikoiden lisäksi käyttöliittymien prototypointityökalua Frameria, sekä käyttöliittymäkehityksen ja dokumentoinnin työkalua Storybookkia.

Toimeksiantajayritys kehittää itselleen yrityksen omaa komponenttikirjastoa, jonka ideana on nopeuttaa ja helpottaa asiakasprojektien kehitysprosessia käyttäen niiden luonnissa yhteisen kirjaston komponentteja. Kyseinen komponenttikirjasto on osa yrityksen kaavailemaa design systeemiä, johon sisältyy tämän opinnäytetyön aikana läpikäytävät käyttöliittymäsuunnittelu- ja luontiprosessit.

Opinnäytetyön teoriaosioissa kerrotaan Reactin, Framerin ja Storybookin toimintamalleista, luontihistoriasta, niiden tarjoamista toiminnallisuuksista, sekä Reactin tapauksessa sen sisarkirjastoista. Teoriaosuudessa käsitellään myös komponenttipohjaisen kehityksen pääperiaatteet ja sen hyödyt sovelluskehityksessä, sekä lyhyesti design systeemi.

Työssä seurataan demottavan komponentin kehitysprosessi käyttöliittymäsuunnittelusta Framer-sovelluksella, lopulta sen luontivaiheeseen. Projektissa kuvataan ensimmäiseksi demottavan komponentin rakennukseen ja testaukseen tarkoitetun hiekkalaatikkoprojektin luonti ja projektissa käytettävien työkalujen testaaminen. Tämän jälkeen työstä siirrytään demottavan komponentin käyttöliittymäsuunnitteluun, jossa sen luonnin aikana käydään yksityiskohtaisesti läpi prosessin elinkaari ja kerrotaan prosessin aikana tehdyt päätökset, sekä Framer-sovelluksessa käytetyt tekniikat. Käyttöliittymäsuunnitelmien teon jälkeen siirrytään itse komponentin konkreettiseen rakentamiseen hiekkalaatikkoprojektissa.

Projektin aikana tarkoitus on luoda toimiva prototyyppi käyttöliittymäkomponentista ja kirjata onnistunut kehitysprosessi. Tämän seurauksena toimeksiantajayritykselle saadaan todiste konseptista käyttää työn aikana käytettyjä sovelluskehitystekniikoita asiakasprojekteissa. Työn lopputuloksena tullaan saamaan aikaiseksi toimiva käyttöliittymäkomponentti, jonka toimeksiantajayritys ottaa käyttöönsä asiakasprojekteissa.

Asiasanat

ohjelmistokehitys, käyttöliittymät, Framer, Storybook, React

South-Eastern Finland University of Applied Sciences

Author (authors)	Degree	Time
Ville-Petteri Sassi	Bachelor of Business Administration	November 2020

Thesis title

Component design with Framer and Storybook — 45 pages

Commissioned by

Mindhive Oy

Supervisor

Janne Turunen

Abstract

The topic of this thesis was the design and creation process of user interface components in which the TypeScript programming language was applied together with the tools and components provided by the React component library. In addition to the above-mentioned techniques, the user interface prototyping tool Framer, as well as the user interface development and documentation tool Storybook were used during the research process.

The client company had been developing the company's own component library, the idea of which was to speed up and facilitate the development process of customer projects by using the components of a common library in their creation. The component library was part of the design system devised by the company in which the user interface design and creation process was the focus of this work.

In the theory section, the thesis describes the operating models of React, Framer, and Storybook, the history of their creation, the functionalities they offer, and, in the case of React, its sister libraries. The theoretical part also deals with the main principles of component-based development and its benefits in software development, as well as briefly the design system.

The work followed the development process of the demo component, from user interface design with Framer to its final creation phase. The project first describes the creation of a sandbox project for the construction and testing of the demo component and the testing of the tools used in the project. The work then proceeded to the user interface design of the demo component, where during its creation the life cycle of the process was reviewed in detail and the decisions made during the process are explained, as well as the techniques used in the Framer application. After the user interface plans, the thesis moved on to the concrete construction of the component itself in the sandbox project.

During the project, the aim was to create a working prototype of the user interface component, as well as to record a successful development process. As a result, the client company received proof of the concept of using the application development techniques used in the work in customer projects. The end result of the work came to be a functional user interface component which the client company will use in their customer projects.

Keywords

software development, user interfaces, Framer, Storybook, React

SISÄLLYS

1 JOHDANTO

Tässä opinnäytetyössä käydään läpi demottavan komponentin käyttöliittymäsuunnittelu- ja kehitysprosessit. Työssä testataan miten nämä kaksi toimivat yhdessä, kun Framer-sovelluksen avulla React-komponenttikirjaston komponentteja käytetään suoraan käyttöliittymäsuunnittelussa. Työ tehdään mikkeliläiselle Mindhive Oy:lle, joka osana omaa tuotekehitystään kehittää omaa komponenttikirjastoa, jonka sisältämiä komponentteja voidaan käyttää projektien käyttöliittymäsuunnitteluissa ja käyttöliittymien rakentamisessa hyväksi.

Tekstissä käydään ensimmäiseksi läpi teoriaosuus, joka koostuu kolmesta luvusta, joiden sisällä esitellään työn kannalta oleelliset työkalut ja konseptit. Ensimmäinen luku käsittelee komponenttipohjaista kehitystä, jonka periaatetta sovelletaan demottavan komponentin suunnittelussa ja rakentamisessa. Samassa luvussa kerrotaan myös design systeemin ideasta, jonka osa toimeksiantajan kaavailema komponenttikirjasto on.

Komponenttipohjaisesta kehityksestä ja design systeemistä siirrytään sitten työssä käytettävän React-komponenttikirjaston avaamiseen. Luvussa selitetään lyhyesti Reactin luontihistoriasta ja toimintaperiaatteesta. Luvun lopussa käydään vielä läpi demottavan komponentin luontiprosessin aikana käytettävä Storybook-työkalu. Storybookin avulla voidaan työssä luotavassa ns. hiekkalaatikkoprojektissa (sandbox), demottavan komponentin toimintaa ja teemoitusvaihtoehtojen vaikutusta testata, Storybookin tarjoaman visuaalisen käyttöliittymän avulla.

Viimeisessä teoriaa käsittelevässä luvussa käydään läpi demottavan komponentin käyttöliittymäsuunnitteluun käytettävä Framer-sovellus. Luvussa esitellään Framer-sovellus ja sen tarjoamat työkalut. Sovelluksen läpikäynnin jälkeen luvun lopussa selitetään vielä kahdesta vaihtoehtoisesta käyttöliittymäsuunnittelussa käytettävistä sovelluksista.

Teoriaosuuden jälkeen siirrytään käytännön osuuteen, jossa ensimmäiseksi kerrotaan tämän opinnäytetyön tarkoituksesta, sen halutusta tavoitteesta, sekä tarkemmin toimeksiantajayrityksestä Mindhive Oy:stä. Tämän jälkeen

seuraavassa alaluvussa käydään läpi komponentin luontiin tarkoitetun hiekkalaatikkoprojektin luonti, sekä työssä käytettäviin työkaluihin tutustuminen. Projektin luonnista ja työkalujen testauksesta siirrytään sitten demottavan komponentin käyttöliittymäsuunnitteluun Framer-sovelluksessa. Tämän alaluvun aikana selitetään demottavan komponentin rakenteen ja teemoitusvaihtoehtojen luontiprosessien vaiheet. Luvun viimeisessä alaluvussa kerrotaan vielä demottavan komponentin luonnista hiekkalaatikkoprojektissa.

Tavoitteena tällä työllä on luoda toimiva prototyyppi käyttäen yllä mainittuja tekniikoita ja työkaluja, sekä kuvata demottavan komponentin käyttöliittymäsuunnittelu- ja kehitysprosessit. Tämän avulla lopputuloksena saadaan todiste konseptista käyttää Reactin komponentteja käyttöliittymäsuunnittelussa ja yleensäkin koko toimintamallin toimivuudesta.

2 KOMPONENTTIPOHJAINEN KEHITYS JA DESIGN SYSTEM

Tässä luvussa kerrotaan komponenttipohjaisesta kehityksestä käyttöliittymäkehityksessä, jonka pääperiaatetta työssä käytettävän React-komponenttikirjasto käyttää hyväksi. Luvussa käydään läpi myös design system, johon kuuluvat demottavan komponentin suunnittelu- ja luontiprosessi.

2.1 Komponenttipohjainen kehitys

Käyttöliittymien mennessä yhä monimutkaisemmiksi ja käyttäjäkohtaisten kokemusten tarjoavien toiminnallisuuksien lisääntyessä sovellukset, joiden käyttöliittymä on rakennettu yhteen isoon kooditiedostoon alkavat nähdä toiminnassaan hankaluuksia. Pintapuolen (front-end) koodin koko kasvaa, joka sitten hankaloittaa ja hidastaa sen käsittelemistä, sekä tekee siitä hauraamman. Siksipä nykypäivänä käyttöliittymäsuunnittelussa lähestytään käyttöliittymien rakentamista komponenttipohjaisesta näkökulmasta, jonka periaatteen mukaan sovelluksen eri osat pilkotaan moniksi modulaarisiksi komponenteiksi. (Componentdriven s.a.)

Komponenttipohjainen kehitys on siis sovelluskokonaisuuksien rakentamista pala kerrallaan. Prosessi toimii suunnittelemalla ja toteuttamalla komponentteja yksi kerrallaan, jotka voivat olla esimerkiksi käyttäjien sisäänkirjautumis-

näkymiä, sivun otsikkopalkkeja (header), tai sivuvalikkoja. Komponentit yhdistetään sitten suurempiin kokonaisuuksiin, jotka liitetään yhteen luoden sovellusten ja verkkosivujen näkymät. Kun sivuihin liitetään mukaan vielä projektin eri taustakoodit (backend), saadaan aikaiseksi toimiva kokonaisuus. (Componentdriven s.a.)

Komponenttipohjaisen kehityksen ”alhaalta ylös” -toimintaperiaate nopeuttaa sovelluskehitystä, kun sovelluksen tai verkkosivun eri osat on eritelty omiksi komponenteiksi. Bugien paikannus on myös helpompaa, komponenttien ollessa eristyksissä toisistaan, verrattuna sovelluksiin ja verkkosivuihin, joissa niiden osat sijaitsevat samassa kooditiedostossa. Tämä mahdollistaa komponenttien testauksen erilaisissa skenaarioissa, jossa voidaan varmistaa niiden toimivuus. Komponenttien ollessa eristyksissä, voidaan niistä luoda yleispätevämpiä ja sitä kautta hyödyntää uudestaan muualla projektissa. (Componentdriven s.a.)

2.2 Design system

Design system on nykypäivän sovelluskehityksessä hyödyllinen työkalu, jonka avulla voidaan järjestää yrityksen tai organisaation suunnittelun ja kehityksen säännöt ja standardit, teemat, työkalut ja komponentit yhteen pakettiin, jota käytetään hyödyksi lähestyttäessä tulevien projektien kehitystä. Design systeemiin voidaan liittää mukaan myös vähemmän konkreettisia asioita, kuten yrityksen tai organisaation arvoja, ajattelutapoja, sekä yhteiseen kehitykseen liittyviä asioita. (Hacq 2018.)

Komponentit ovat keskeinen osa design systeemin toimintaa ja niiden suunnittelun, rakentamisen ja implementoinnin tulisi noudattaa design systeemissä olevia sääntöjä ja ohjeistuksia. Design systeemin avulla voidaan määrätä yleinen tapa toimia komponenttikehityksessä ja asettaa selviä rajoja yksittäisen komponenttien rakentamiseen, kuten rajata niiden kompleksisuutta, jolloin komponentille pystytään rajaamaan vain yrityksen tai organisaation brändiin sopivaksi katsotut kustomointimahdollisuudet. (Hacq 2018.)

Komponenttikirjaston olemassaolo hyödyntää yrityksen tai organisaation sovelluskehitystä tekemällä johdonmukaisten käyttöliittymäratkaisujen teon nopeammaksi ja helpommaksi, kun käyttöliittymäkehityksessä käytetään valmiiksi suunniteltua teemoitusta, voidaan komponentit implementoida vähemmällä työllä. Vakioitunut suunnittelustandardi tuo myös esille vaikutelman laadusta ja auttaa tiimien välisessä kommunikoinnissa, kun komponenttien nimeämistapa noudattaa samoja sääntöjä. (Baraniak 2019.)

Design systeemejä on monenlaisia ja niiden suunnitteluun liittyy erilaisia kysymyksiä, kuten sen potentiaalinen käyttäjämäärä ja näiden käyttäjien käyttökokemus, design systeemin määräämillä työkaluilla ja tekniikoilla. Design systeemin skaalautuvuuden arvioiminen on myös tärkeää ja se, kuinka moneen projektiin tai alustaan sitä pystytään hyödyntämään, sillä eri projektit saattavat sisältää eri ohjelmointitekniikoita, frameworkkeja tai kohdeprojekti ei ole muuten vain sopiva olemassa oleviin standardeihin. Design systeemin suunnittelussa onkin suotavaa löytää tasapaino tiukan ja löyhän design systeemin väliltä, sillä liian tiukka järjestely saattaa vain vaikeuttaa sovelluskehitys työtä ja liian löyhä design systeemi ei tarjoa tarpeeksi hyötyjä, että sen olemassaolo voitaisiin järkevästi perustella. (Hacq 2018.)

Esimerkkinä design systeemistä voidaan käyttää Googlen kehittämää Material Designia, jonka inspiraationa toimii oikean maailman tekstuurit ja elementit. Material Design sisältää komponentteja, jotka sisältävät omat tilametodinsa, eli komponenttien dataa käsittelevät objektit ja niiden toiminta mukailee Material Designin suunnitteluperiaatteita. Tämän suunnitteluperiaatteen tarkoituksena on luoda typografian, koon, värien ja muiden teemoituselementtien avulla luoda selvä hierarkia, tarkoitus ja käyttäjän immersiota lisäävä kokemus. (Material Design s.a.)

3 REACT JA SEN KOMPONENTIT

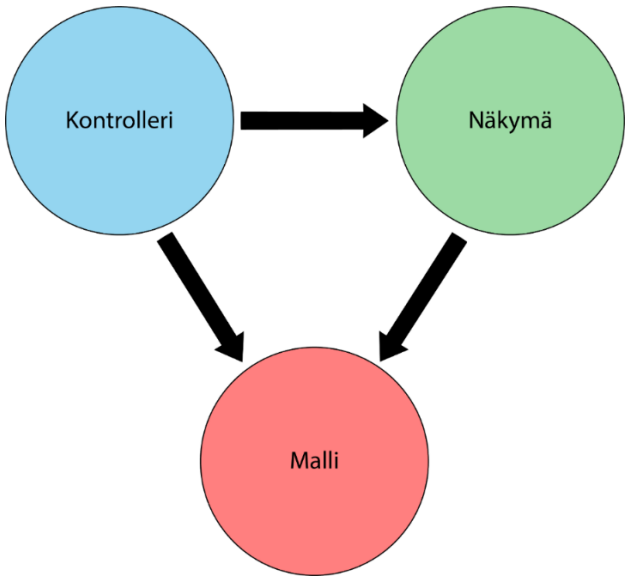
Tässä luvussa kerrotaan opinnäytetyön teossa käytettävästä React-komponenttikirjastosta. Luvussa kerrotaan Reactin luontihistoriasta, sen suunnitteluperiaatteesta, sekä mobiilisovellusten luontiin luodusta React Nativesta. Luvussa käydään läpi myös Reactin eri komponenttityypit ja niiden käyttöperiaat-

teet, sekä Material Design systeemiä hyödyntävät Reactille suunnitellut Material-UI ja React Native-komponenttikirjastot. Viimeisenä luvussa kerrotaan vielä opinnäytetyössä käytetystä Storybook-työkalusta.

3.1 React ja React Native

React, joka tunnetaan myös nimellä ReactJS, on avoimeen lähdekoodiin perustuva JavaScript-komponenttikirjasto, joka on suunniteltu uudelleenkäytettävien pintapuolen (front-end) käyttöliittymien rakentamiseen. Reactin kehityksen aloitti Facebook vuonna 2011 ja julkaisi komponenttikirjaston avoimesti vuonna 2013. Reactin avulla käyttäjä voi luoda HTML-tyylisiä käyttöliittymäkomponentteja sovelluksien tai verkkosivujen käyttöliittymiin. (Krill 2014.)

Krillin (2014) mukaan Reactin toiminta perustuu MVC-mallissa (kuva 1) olevan näkymä (view) osan ympärille. MVC-malli on toimintamalli, jossa sovelluksen tai verkkosivun toiminta on jaettu kolmeen osaan, joista ensimmäinen, malli (model) hoitaa tietojen käsittelyyn liittyvät toiminnot, kuten tietokantakutsut. Toinen MVC-mallin osa, josta React on toimintakokonaisuudessa vastuussa, on näkymä (view), joka hoitaa sovelluksen tai verkkosivun tietojen näkymisen käyttöliittymässä ja myös kaikki muut käyttöliittymän toiminnallisuudet. Viimeinen osa käsittelijä (controller), josta voidaan käyttää myös nimitystä kontrolleri tai ohjain, hoitaa mallien ja näkymien välisen viestinnän. (Balliauw 2009, 23–25.)



Kuva 1. Esimerkki MVC-mallista

Komponenttiensa päivittämiseen React käyttää virtuaalista DOM-mallia, joka muutosten tapahtuessa vertaa muuttuneita ja vanhoja tietoja keskenään, jonka jälkeen vain muutettuihin tietoihin koskevat käyttöliittymäkomponentit päivitetään. Tämä prosessi nopeuttaa käyttöliittymän toimintaa ja auttaa säästämään tehoja. (Honkanen 2017, 13.)

React Native taas on IOS ja Android -laitteille mobiilisovelluksien luontiin suunniteltu framework, toisin kuin React, joka on tarkoitettu verkkopohjaisten sovellusten luontiin. Samaan tapaan kuin Reactissa, sovellukset luodaan yhdistämällä JavaScript tai TypeScript ohjelmointikieltä ja XML-tyylistä merkintäkieltä. React Native muuttaa koodin kohdelaitteelle sopivaan muotoon, kutsumalla natiivia renderöinti-API:a, joka renderöi komponentit aitoina mobiili UI-komponentteina. Koska React Native tekee yhteistyötä kohdelaitteensa renderöinti-API:n kanssa pystyy se käyttämään hyväksi laitteen natiiveja UI-elementtejä, toisin kuin esimerkiksi Ionic ja Cordova, joita käytetään luomaan hybridimobiilisovelluksia, joissa sovelletaan JavaScriptiä, HTML-merkintäkieltä ja CSS:ää. Tämän seurauksena kaikki mobiililaitteen natiivit UI-komponentit joudutaan Ionicin ja Cordovan kanssa kopioimaan manuaalisesti, jonka seurauksena niiden ulkoasu ei välttämättä näytä autenttiselta. React Nativen kyky käyttää mobiililaitteiden natiiveja UI-komponentteja auttaa pitämään ulkoasun myös ajan tasalla ja koska verkkonäkymiä ei tarvitse käyttää, sovelluksen suorituskyky ei myöskään kärsi. (Eisenman 2016.)

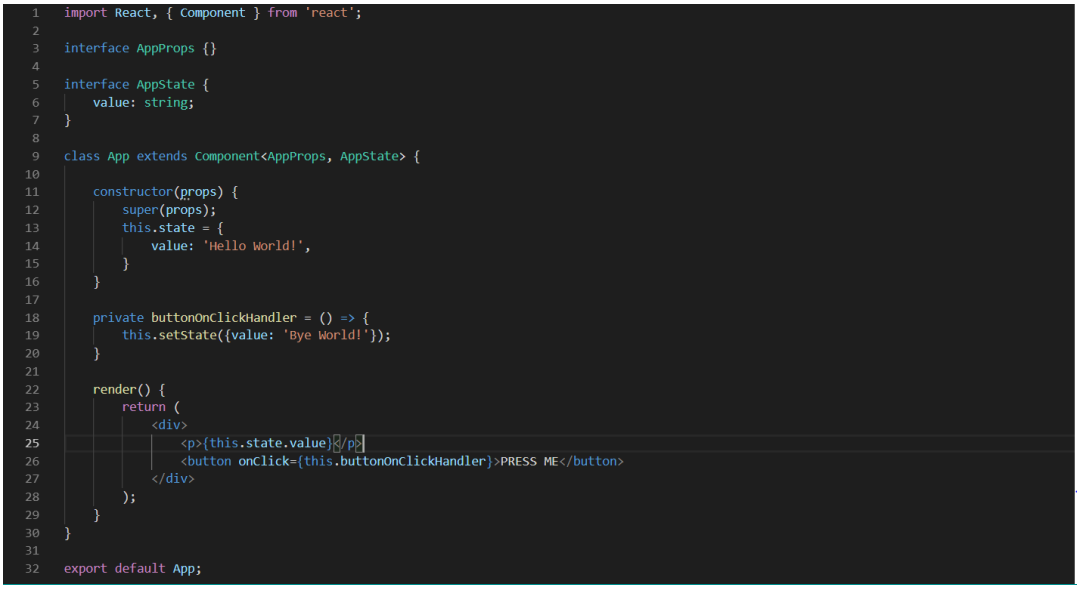
3.2 Reactin komponentit

Reactin toiminta perustuu sen komponenttien väliseen yhteistyöhön, jotka voivat olla joko erilaisia JavaScript-funktioita tai objekteja. React itsessään on Honkasta (2017, 16) lainaten ”yksi iso komponentti, joka sisältää useita lapsikomponentteja”. Komponentit pystyvät lähettämään hierarkiassa alaspäin dataa lapsikomponenteilleen, jotka sitten voivat muokata tätä dataa ja lähettää sen vielä eteenpäin omille lapsikomponenteilleen. Data lähetetään eteenpäin lapsikomponentille annettuna ominaisuusparametrinä (props). (React s.a.)

Reactissa käytettävät komponentit ovat tyypiltään, joko luokkapohjaisia (class) tai funktionaalisia (functional) komponentteja. Komponenttityypit on suunni-

teltu hoitamaan monenlaisen datan esittämiseen, joista ensimmäinen luokkapohjainen komponentti (kuva 2) soveltuu muuttuvan datan renderöintiin ja käsittelemiseen, joka voi olla esimerkiksi verkkosivun hakukenttä, jossa on mukana tulosten suodatustoiminnallisuus. Luokkapohjaiset komponentit ovat ajan tasalla omasta elinkaarestaan (lifecycle) ja verrattuna funktionaalisiin komponentteihin nähdään suuritöisempinä. (Honkanen 2017, 16.)

Monet luokkapohjaiset komponentit voidaan luokitella myös älykkäiksi komponenteiksi (smart components), josta voidaan käyttää myös nimitystä säiliö komponentti (container component). Älykkäät komponentit ovat pääasiassa luokkapohjaisia tai funktionaalisia komponentteja, joilla on kyky hoitaa omat elinkaari- ja tilametodinsa (state) container-funktion avulla. (Arnold 2017.)



```
import React, { Component } from 'react';

interface AppProps {}

interface AppState {
    value: string;
}

class App extends Component<AppProps, AppState> {

    constructor(props) {
        super(props);
        this.state = {
            value: 'Hello World!',
        }
    }

    private buttonOnClickHandler = () => {
        this.setState({value: 'Bye World!'});
    }

    render() {
        return (
            <div>
                <p>{this.state.value}</p>
                <button onClick={this.buttonOnClickHandler}>PRESS ME</button>
            </div>
        );
    }
}

export default App;
```

Kuva 2. Esimerkki älykkäästä luokkapohjaisesta komponentista

Funktiopohjaiset komponentit (kuva 3) ovat toiminnallisuuksiltaan verrattavissa luokkapohjaisiin komponentteihin, ja erona näillä kahdella on vain tavat, joilla ne elinkaari- ja tilametodejansa (state) käyttävät. Alun perin funktionaaliset komponentit olivat luokkapohjaisiin komponentteihin verrattuna yksinkertaisempia. Funktionaaliset komponentit kykenivät tuolloin vain käsittelemään muuttumatonta dataa ja renderöimään sen, eivätkä ne alun perin kyenneet luomaan omia elinkaari- tai tilametodeja (state). Reactin 16.8 versiopäivityksen jälkeen näiden toiminnallisuuksien käyttö kuitenkin mahdollistui useState- ja useEffect-hookkien (koukkujen) avulla. (Jöch 2018.)

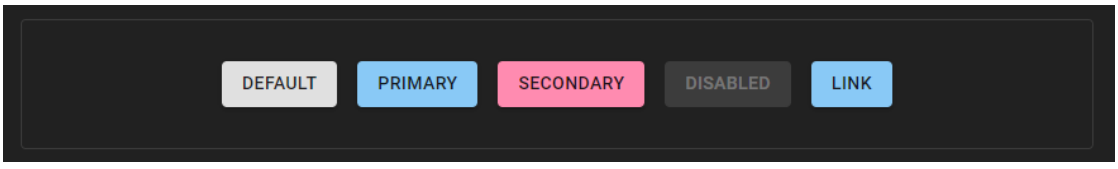
```tsx
import React, { useState } from 'react';

export const App = () => {
    const [value, setValue] = useState<string>('Hello World!');

    const buttonOnClickHandler = () => {
        setValue('Bye World!');
    }

    return (
        <div>
            <p>{value}</p>
            <button onClick={buttonOnClickHandler}>PRESS ME</button>
        </div>
    );
};

export default App;
```

Kuva 3. Esimerkki funktionaalisesta komponentista

Puhtaiden luokkapohjaisten ja funktionaalisten komponenttien lisäksi Reactilla on mahdollista tehdä myös monen muun tyyppisiä komponentteja. Esimerkiksi ns. ”higher-order”-tyyppiset komponentit ovat funktioita, jotka ottavat ominaisuusparametrinä kokonaisen komponentin ja palauttavat sen takaisin uusilla toiminnallisuuksilla varustettuna, kuten vaikka funktio, joka ottaa vastaan objektin, johon on kasattu verkkokaupan ostotapahtuman tiedot ja palauttaa muuttujien mukaan lasketun yhteishinnan. (React s.a.)

Muun tyyppisiä komponentteja ovat myös ns. ”tyhmät” komponentit (dumb components), josta voidaan paremmin käyttää nimitystä esityksellinen komponentti (presentational component). Esityksellinen komponentti on funktionaalinen komponentti, jolla ei ole tilametodeja ja sitä käytetään enemmänkin ominaisuusparametrinä tuodun tiedon esittämiseen käyttöliittymässä. Esimerkkejä esityksellisistä komponenteista ovat esimerkiksi klassinen ”Hello World!” tuloste tai verkkosivun alatunniste. (Arnold 2017.)

Vaikka Reactista löytyy valmiiksi jo suuri määrä käyttökelpoisia komponentteja, voidaan sitä laajentaa vielä lukuisilla eri komponenttikirjastoilla. Näistä yksi on Material-UI, joka sisältää käyttöliittymäsuunnittelussa oleellisia komponentteja, kuten nappeja (kuva 4), tekstikomponentteja ja säiliökomponentteja. (Material-UI s.a.a.)

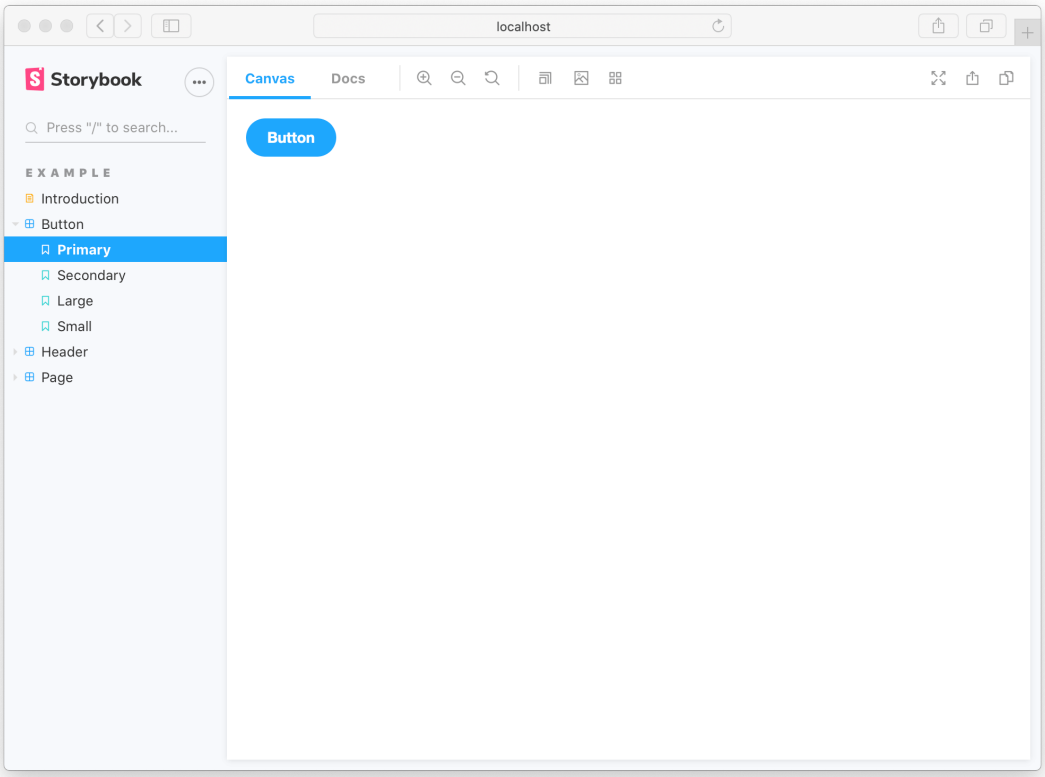


Kuva 4. Esimerkki erilaisista Material-UI napeista (Material-UI s.a.b.)

Samoin kuin Reactille on olemassa Material-UI verkkopohjaisille sovelluksille, on React Nativelle myös olemassa komponenttikirjasto nimeltä React Native Paper. Samalla tavoin, kuin Material-UI, React Native Paper tarjoaa Material Design systeemin periaatteiden mukaan suunniteltuja käyttöliittymäkomponentteja. (React Native Paper s.a.)

3.3 Storybook

Storybook on komponenttikehitykseen tarkoitettu työkalu, joka mahdollistaa irrallisten käyttöliittymäkomponenttien suunnittelun ja dokumentoinnin, vaikuttamatta projektin muuhun koodin. Työkalu tarjoaa oman visuaalisen käyttöliittymän, jolla komponentin toiminnallisuuksia pystytään demoamaan ja jonka avulla päästään näkemään, miltä komponentin ulkoasu näyttää käytännössä (kuva 5). (Storybook s.a.a.)



Kuva 5. Esimerkki Storybookin käyttöliittymästä (Storybook s.a.c.)

Storybookin avulla voidaan luoda komponentista erilaisissa avaintiloissa olevia malleja, joista työkalu käyttää nimitystä story. Näillä avaintiloilla voidaan demota komponentin toiminnallisuuksia ja ulkoasua eri ominaisuusparametrien, kuten esimerkiksi erilaisten värien, fonttien tai animaatioiden kanssa.

Luomalla koodissa sapluunat (templates) komponentin eri avaintiloille ja antamalla niille erilaisia ominaisuusparametrejä, pystytään helposti näkemään miltä komponentti niissä näyttää. Esimerkiksi yksinkertaisen nappikomponentin avaintiloja voivat olla sen eri teemavärit (primary, secondary, jne.), sekä koot (pieni, keskikokoinen, suuri), joista sitten voidaan luoda omat storyt, antamalla sapluunoiden ominaisuusparametreihin avaintiloja vastaavat arvot. (Storybook s.a.b.)

4 FRAMER

Tämän luvun tarkoituksena on kertoa työssä tehtävän komponentin käyttöliittymäsuunnittelussa käytettävästä Framer-prototypointisovelluksesta. Luvussa esitellään sovellus ja sen käyttöliittymä, sekä tutustutaan lisäksi muihin sovelluksen kaltaisiin työkaluihin, jotka tarjoavat samankaltaisia työkaluja projektien käyttöliittymien suunnitteluun.

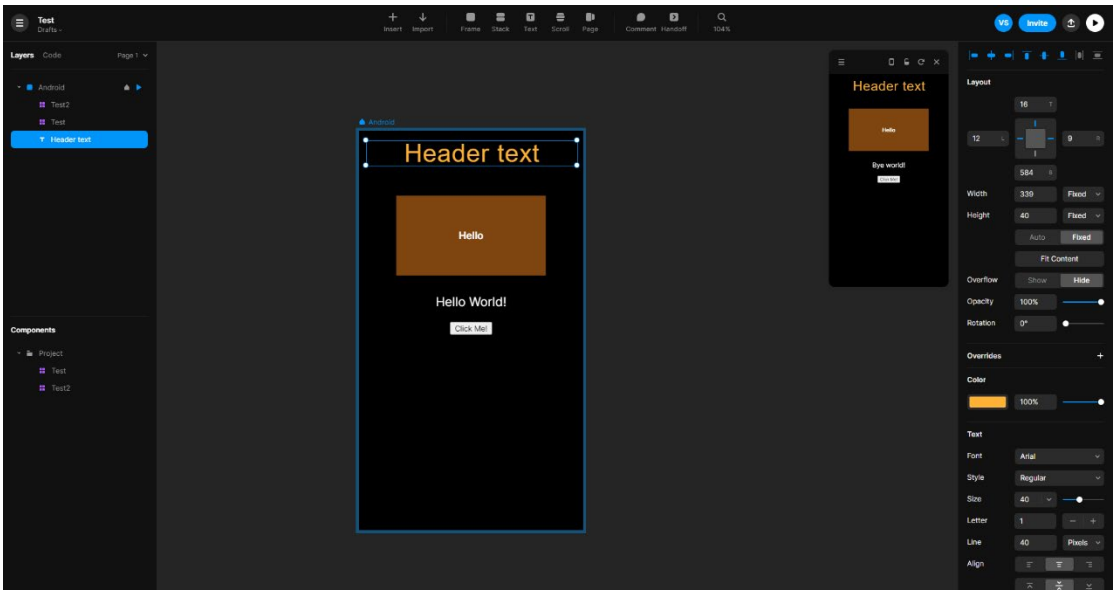
4.1 Sovelluksen esittely

Framer on mobiilisovellusten ja verkkosivujen käyttöliittymäkomponenttien ja näkymien demoamiseen ja prototypointiin tarkoitettu sovellus. Framerin avulla projektitiimi pystyy suunnittelemaan yhdessä toimivia käyttöliittymäkomponentteja, animaatioineen ja tyyleineen. Framer tarjoaa mahdollisuuden muokata komponentteja editorinsa kautta, sekä myös antaa pääsyn komponenttien muokkaukseen koodillisesti. (Framer s.a.a.)

Reactin komponentit ovat toimivia Framerin kanssa ja käyttämällä JSX-merkkikieltä pystytään niiden avulla luomaan toimivia käyttöliittymäkomponentteja ja näkymiä. Framerin kanssa pystytään käyttämään myös TypeScript ohjelmointikieltä mahdollisten koodillisten toiminnallisuuksien tekoon, kuten esimerkiksi napinpainalluksiin reagoivien funktioiden tekemiseen. (Framer API s.a.)

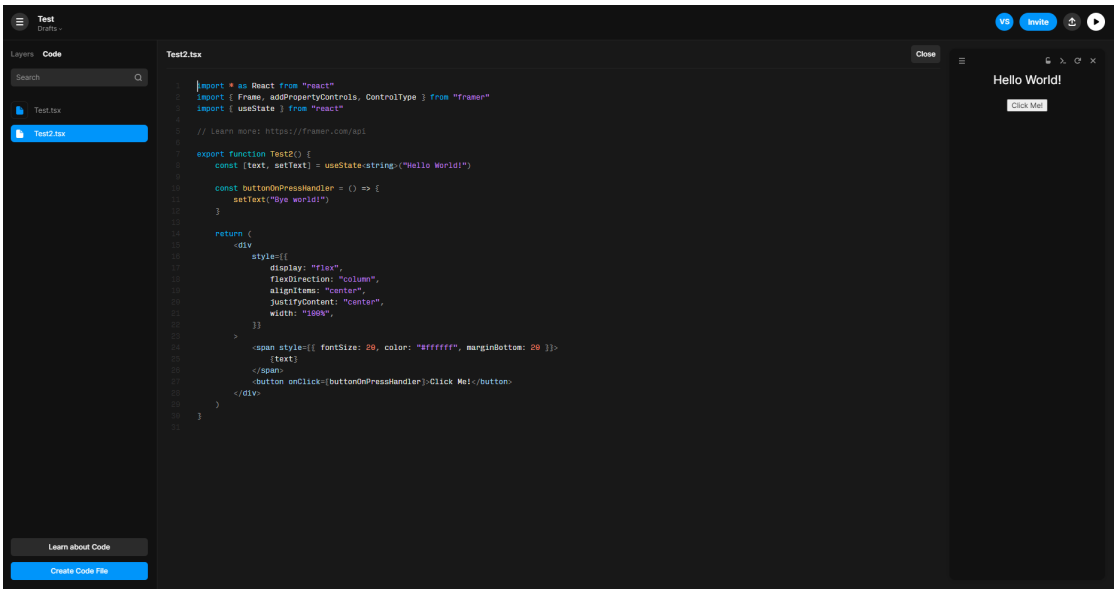
Framer sovellus on käytettävänä suoraan verkkoselaimessa, Framerin omilta sivuilta. Framerin hinnastosivulla (Framer s.a.b.) sovellus on listattu olevan saatavana ilmaiseksi verkosta, mutta sovelluksesta on olemassa myös maksullisia versioita, jotka vaihtelevat yksityisille henkilöille ja pienille projektitiimeille suunnitelluista paketeista, suuremmille yli 20 henkilön suuruisille organisaatioille. Käyttöliittymä sovelluksessa on jaettu kahteen näkymään, joista

ensimmäinen on graafinen näkymä (kuva 6). Graafisessa näkymässä käyttäjä pystyy tekemään eri näkymien ulkoasusuunnittelua, lisäämällä komponentteja, tekstiä ja animaatioita. (Framer s.a.a.)



Kuva 6. Framerin graafinen näkymä

Toinen Framerin käyttöliittymän näkymä on koodillinen näkymä (kuva 7), jossa komponenttien rakennetta pystyy muokkaamaan koodieditorin kautta. Koodieditori toimii tyypillisesti samalla lailla, kuin muut koodieditorit, eli koodia pystytään lisäämään ja muuttamaan, uusia komponentteja pystytään tuomaan osaksi tiedostoa, sekä teemoitusta ja tyylejä voidaan muokata. (Framer s.a.a.)

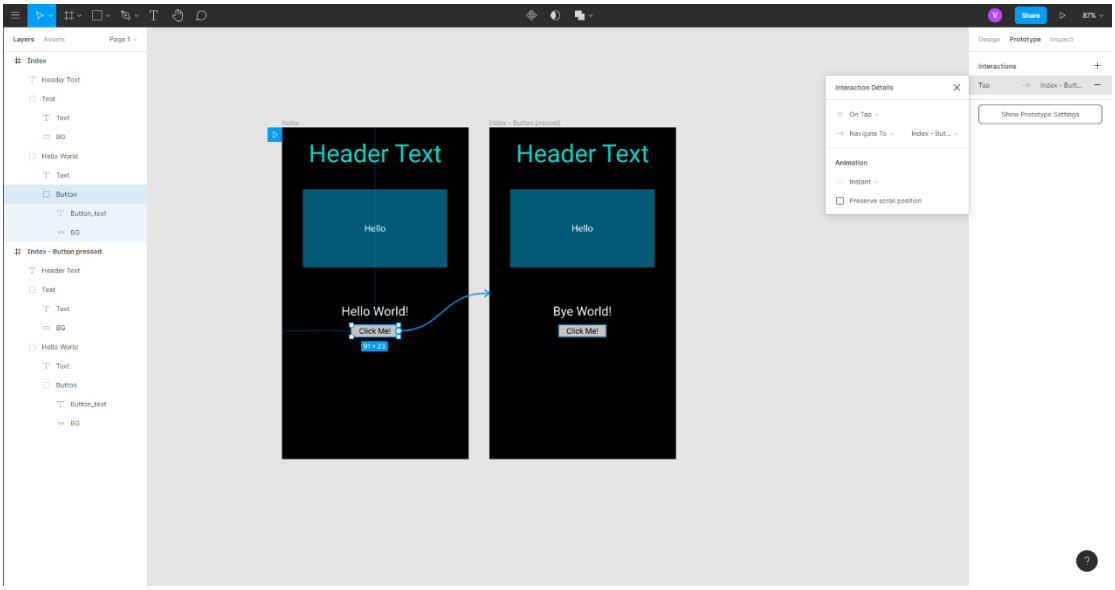


Kuva 7. Framerin koodillinen näkymä

Framerista on olemassa myös Mac OS:lle ladattava versio, joka on nimeltään Framer X. Tietokoneelle ladattava sovellus sisältää Framerin oman sisäisen kauppasovelluksen, josta käyttäjät pystyvät lataamaan projekteihinsa uusia visuaalisia ja toiminnallisia komponentteja. (Framer s.a.c.)

4.2 Muita vastaavia sovelluksia

Framerin lisäksi sovellusten ja verkkosivujen käyttöliittymäsuunnitteluun on olemassa myös monia muita suunnittelu- ja prototypointi-sovelluksia, jotka tarjoavat samankaltaisia työkaluja käyttöliittymäkomponenttien ja näkymien tekoon. Yksi näistä on alalla paljon käytetty Figma (Kuva 8), joka on pääasiassa suunniteltu käytettäväksi suoraan selaimessa, josta on myös olemassa Windows ja Mac OS laitteille ladattava versio. Figman toiminta perustuu ”aina verkossa” (always online) periaatteeseen ja antaa projektitiimin jäsenille mahdollisuuden työskennellä saumattomasti samaan aikaan yhden tai useamman käyttöliittymäsuunnitteluprojektin kanssa. (Bracey 2018.)



Kuva 8. Esimerkki Figman käyttöliittymästä

Toinen vaihtoehtoinen käyttöliittymäsuunnittelussa käytetty ohjelma on Sketch. Mac OS:lle ladattava maksullinen sovellus Sketch on vektoripohjainen piirtotyökalu, joka on paljon käytetty sovellusten ja verkkosivujen UI-komponenttien, näkymien ja ikonien suunnittelussa. Sketchillä ei suoranaisesti pys-

tytä tekemään puhtaita UI-komponentteja, joten sillä valmistetaan enemmänkin graafisia malleja, joiden pohjalta komponentit ja näkymät lopulta rakennetaan. (Purdila 2017.)

Käyttöliittymäsuunnittelu Figmalla ja Sketchillä onnistuu hyvin, mutta ulkoasun lisäksi UI-komponenteilla on useasti myös toiminnallisia vaatimuksia, joiden demoaminen näillä ohjelmilla on vaikeaa tai jopa mahdotonta. Monet yritykset ja organisaatiot ovat tähän mennessä todennäköisimmin ehtineet luoda jonkin kokoisen määrän käyttöliittymä- ja komponenttisuunnitelmia, jotka on mahdollista tuoda osaksi Framerillä luotuun projektiin. Framer sisältää siis toiminnallisuuden, jolla Figmassa tai Sketchissä luodut käyttöliittymäsuunnitelmat pystytään muuttamaan Framerille sopiviksi ja siten tehdä niistä toiminnallisempia, jolloin mahdolliset nappien painallukset, sivusiirtymät ja muut toiminnallisuudet pystytään demoamaan paremmin Framerin omill animaatiotyökaluilla. (Framer s.a.)

5 TOTEUTUS

Tässä luvussa käydään läpi työn toteutusvaiheen eri osat. Alussa esitellään toimeksiantajayritys Mindhive Oy, sekä selvennetään työn tarkoitus ja sen tavoite. Luvussa käytävien työn toteutuksen vaiheiden kertominen alkaa projektin luonnin ja työssä käytettävien työkalujen testauksen läpikäynnillä, josta siirrytään vaiheittain komponentin suunnitteluun ja luontiin.

5.1 Työn tarkoitus, tavoite ja toimeksiantaja

Tarkoitus työllä on demota toimeksiantajayrityksen kaavaileman design systeemin osa, joka tässä tapauksessa on yrityksen oman komponenttikirjaston komponenttien suunnittelu ja luontiprosessi. Komponenttikirjaston komponenttien käyttöliittymäsuunnitelmat on tarkoitus tehdä Framer-sovelluksen avulla. Komponenttien luomiseen taas käytetään omaa erillistä ns. hiekkalaatikkoprojektia (sandbox), jossa ne rakennetaan Reactin, HTML-elementtien ja muiden mahdollisten palikoiden avulla toimiviksi kokonaisuuksiksi. Storybook-työkalua on myös tarkoitus käyttää komponentin ulkoasun ja toiminnallisuuksien testaukseen, työkalun tarjoaman visuaalisen käyttöliittymän avulla.

Työn aikana komponentin suunnittelun ja luontiprosessin läpikäymiseen tullaan käyttämään esimerkkinä kalenterikomponenttia. Kalenterikomponentteja käytetään paljon erilaisissa sovelluksissa, mutta ongelmana niillä on oman kokemuksen perusteella teemoitukseen liittyvien asetusten puute. Siksi päätinkin aiheeksi valita esimerkkinä käytettäväksi komponentiksi kalenterikomponentin, jonka työn aikana suunnittelen omaavan komponentille oleellisten toiminnallisuuksien lisäksi paljon sen ulkoasun määrittteleviä teemoitusvaihtoehtoja.

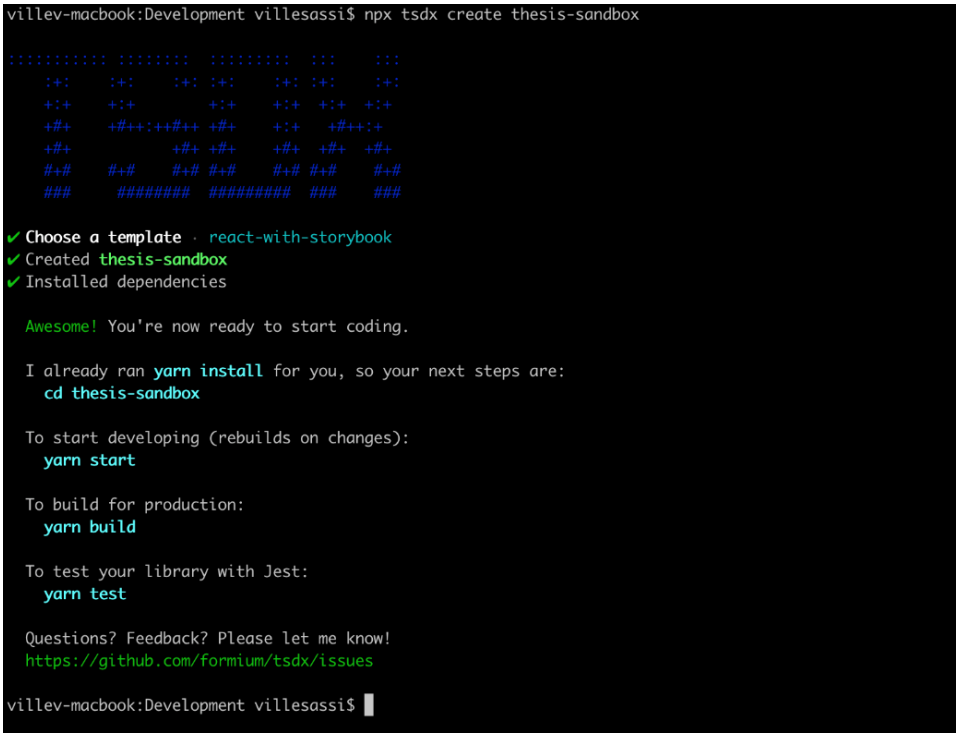
Toimeksiantajana työlle toimii Mindhive Oy, joka on Mikkelissä vuonna 2016 perustettu sovelluskehityspalveluihin keskittynyt yritys. Yrityksen tarkoituksena on auttaa yrityksiä ja yhteisöjä menestymään digitalisaation avulla, jossa käyttäjäkokemus on tärkeä osa sovellusten ja verkkosivujen luontiprosessia. Mindhive panostaa toiminnassaan selkeään ja runsaaseen viestintään projektien aikana, sekä yleisesti ketterän kehityksen mukaisiin toimintamalleihin. (Mindhive Oy s.a.a.)

Mindhive Oy:n tarjoama tuotevalikoima koostuu neljästä palvelutuotteesta, joista ensimmäinen mindleap (Mindhive Oy s.a.b) on suunniteltu ihmislähtöiseksi tavaksi suunnitella asiakkaan visiota palvelumuotoilun, UI-suunnittelun ja data-analyysien avulla, sekä ratkoa yhteistyössä suunnitelmaa vaikeuttavia ongelmia. Mindsolutions -palvelu (Mindhive Oy s.a.c) taas keskittyy pilvipalveluja hyödyntävään sovelluskehitykseen, jossa tarkoituksena on luoda asiakkaan liiketoiminnan kanssa skaalautuva kokonaisuus. Mindpower (Mindhive Oy s.a.d) on resurssipalvelu, jossa yritys tarjoaa alihankintana asiantuntijoitaan asiakasprojekteihin. Lopuksi on mindtrack (Mindhive Oy s.a.e), joka tarjoaa alan korkeakouluopiskelijoille mahdollisuuden päästä harjoittelemaan ammattialaansa oikeiden asiakasprojektien kanssa ja avaa mahdollisia väyliä työelämään.

5.2 Hiekkalaatikkoprojektin luonti ja esikatsaus työkaluihin

Työ aloitettiin ensimmäiseksi luomalla ns. hiekkalaatikkoprojekti (sandbox), jossa komponentteja voitaisiin luoda ja testata vaarantamatta muita projekteja. Projektin luontiin käytettiin hyväksi TSDX:ää (kuva 9), joka on ns. nollakoonpano (zero-configuration) CLI-komento, jonka avulla voidaan luoda valmiita projektipohjia käyttämättä aikaa projektin kehitykselle ja komponenttien

testaukselle oleellisten työkalujen ja lisäosien asentamiseen erikseen. Komennon avulla saatiin luotua React-projekti, jossa kaikki oleelliset työkalut ovat valmiiksi asennettuna, kuten projektin ja sen asennettavien pakettien hallinnasta vastaava Yarn, yksikkötestaukseen käytettävä Jest, koodin ongelmakohtia vainuava ESLint, sekä komponenttien visuaaliseen testaukseen käytettävä Storybook-työkalu.



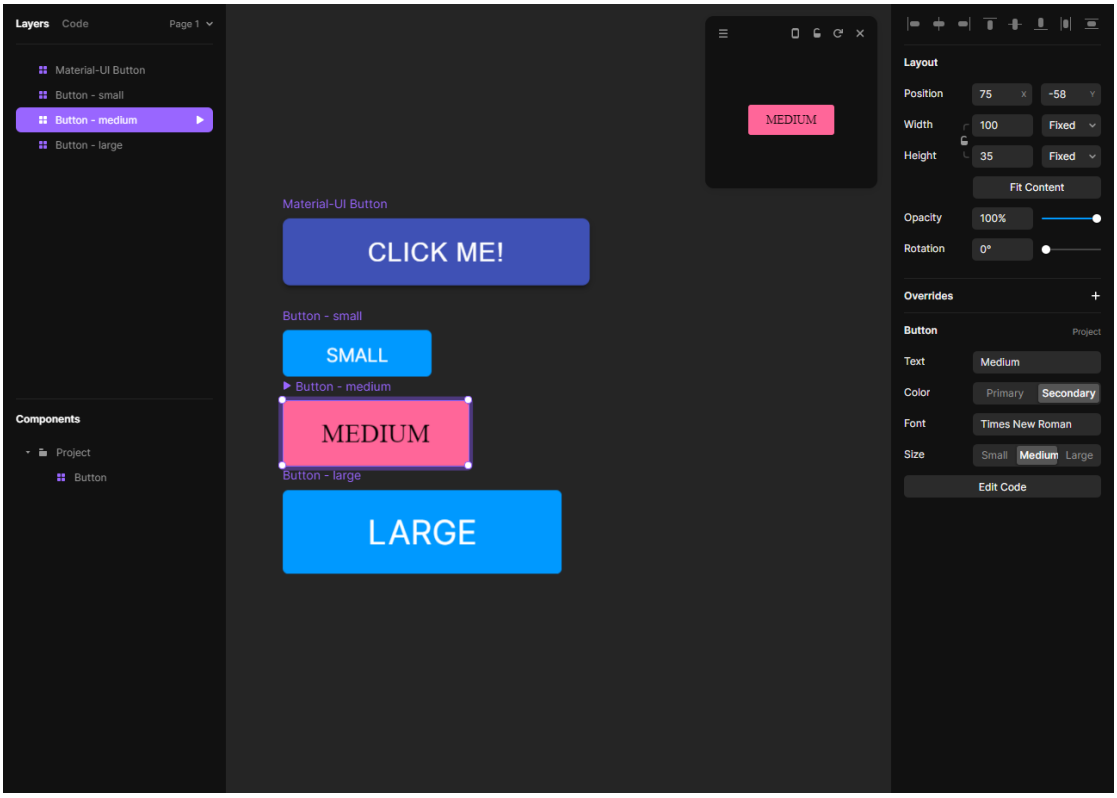
Kuva 9. Komentorivissä käyty TSDX CLI-komento

Kun hiekkalaatikkoprojekti oli valmis, seuraavaksi edessä oli työssä käytettävien työkalujen kokeilu käytännössä. Työkalujen testaukseen päätin demota yksinkertaisen nappikomponentin, tosin koska nappikomponentit ovat helposti saatavina melkein kaikkien komponenttipakettien mukana ja tarjoavat laajalti paljon muokkausmahdollisuuksia, ei demoukseen käyttämääni nappikomponenttia tulla todennäköisesti käyttämään hyväksi tulevaisuudessa. Joten nappikomponentti tuleekin olemaan tarkoitettu puhtaasti työkalujen testaukseen ja työprosessin läpikäyntiin.

Ensimmäinen vaihe työprosessin testauksessa oli komponentin visuaalinen käyttöliittymäsuunnitelma (kuva 10), jonka tekoon käytin Frameria. Nappikom-

ponentin suunnittelussa otin esimerkkiä olemassa olevista nappikomponenteista ja rajasin oman nappikomponenttini mahdolliset kustomointivaihtoehdot neljään eri osaan:

- Teksti (Text), joka on ainoa pakollinen vaihtoehto ja määrittää napissa lukevan tekstin
- Väri (Color), joka määrittää napin, sekä sen sisältämän tekstin värit (primary, secondary)
- Fontti, (Font), joka määrittää napin tekstissä käytettävän fontin (esim. Times New Roman, Calibri, jne.)
- Koko (Size), joka määrittää napin korkeuden, leveyden ja tekstin koon kolmesta mahdollisesta vaihtoehdosta



Kuva 10. Nappikomponentin Framer-suunnitelma

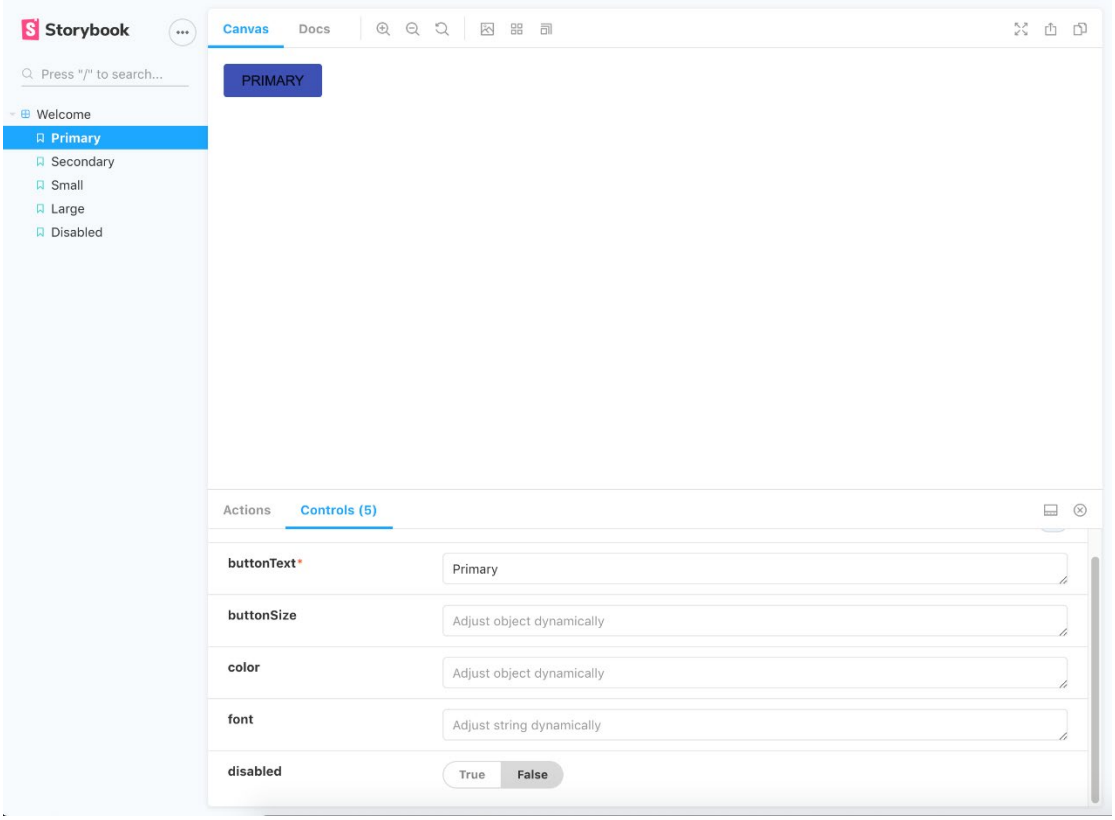
Nappikomponentin käyttöliittymäsuunnitelman valmistuttua, seuraava vaihe oli komponentin luonti projektissa. Komponentti rakennettiin normaalin HTML-napin ympärille, jonka kokoa, kulmia ja väriä muokkaamalla saadaan se näyttämään samanlaiselta, kuin Framerissa tehty nappikomponentti. Napin teksti, väri, koko, fontti, sekä käyttöliittymäsuunnitelmasta puuttuva vaihtoehto napin sammutukselle, josta käytetään englanniksi nimitystä disabled ja napinpainallukseen reagoiva funktio tuodaan ominaisuusparametreinä komponenttiin, jotka sitten käsitellään ja tehdään nappiin niiden mukaiset muutokset.

Monissa projekteissa on olemassa mitä todennäköisimmin oma teemoitus, josta löytyy valmiiksi pää- ja toissijaiset (primary & secondary) värit, sekä myös värit teksteille, taustoille ja muille mahdollisille osille. Yksi tapa luoda teemoitus projektille on käyttää Material-UI:ta, jonka useTheme-funktion avulla päästään käsiksi projektin yleiseen teemaan. Näin saadaan varmistettua, että komponentin värivalinta noudattaa projektin sisäistä teemoitusta, eikä värejä tarvitse manuaalisesti käydä asettamassa. Material-UI:n asennuksen, tuonnin ja teemavärien asettelun jälkeen, komponentin koodi näytti siltä, kuten se kuvassa 11 on esitetty.

```
import React from 'react';
import { useTheme } from '@material-ui/core';

export interface ButtonProps {
    buttonText: string;
    buttonSize?: ButtonSizes;
    color?: ButtonColors;
    font?: string;
    disabled?: boolean;
    onClick: () => void;
}

export enum ButtonSizes {
    'small',
    'medium',
    'large',
}

export enum ButtonColors {
    'primary',
    'secondary',
}

const Button = (props: ButtonProps) => {
    const theme = useTheme();
    const {buttonText, buttonSize, color, font, disabled, onClick} = props;

    const buttonColor = {[ButtonColors.primary]: theme.palette.primary.main, [ButtonColors.secondary]: theme.palette.secondary.main};
    const textColor = {[ButtonColors.primary]: theme.palette.text.primary, [ButtonColors.secondary]: theme.palette.text.secondary};
    const widths = {[ButtonSizes.small]: 80, [ButtonSizes.medium]: 100, [ButtonSizes.large]: 150};
    const heights = {[ButtonSizes.small]: 25, [ButtonSizes.medium]: 34, [ButtonSizes.large]: 45};
    const textSizes = {[ButtonSizes.small]: 10, [ButtonSizes.medium]: 14, [ButtonSizes.large]: 18};

    return <button
    style={{
        backgroundColor: !disabled ? buttonColor[color ?? 0] : theme.palette.grey[300],
        color: !disabled ? textColor[color ?? 0] : theme.palette.grey[600],
        borderWidth: 0,
        borderRadius: 3,
        fontFamily: font ?? 'Arial',
        minWidth: widths[buttonSize ?? 1],
        height:  heights[buttonSize ?? 1],
        fontSize: textSizes[buttonSize ?? 1]
    }}
    disabled={disabled}
    onClick={onClick}>
        {buttonText?.toUpperCase()}
    </button>
}

export default Button;
```

Kuva 11. Nappikomponentin koodi

Komponentin visuaaliseen demoamiseen käytettiin hyväksi Storybookia (kuva 12), jonka avulla napille luotiin 5 mahdollista avaintilannetta, joista kaksi ensimmäistä kuvasivat napin väriteemoituksen muutosta, joko päävärillä tai toissijaisella värillä varustettuna. Seuraavat storyt, kuvasivat nappikomponentin ulkoasua tilanteissa, joissa sen kooksi on valittu, joko pieni tai suuri ja viimeisenä storyna nappi kuvastetaan sammuneena, jossa sen painalluksesta aktivoituva funktio on poissa käytöstä.



Kuva 12. Nappikomponentti Storybookissa

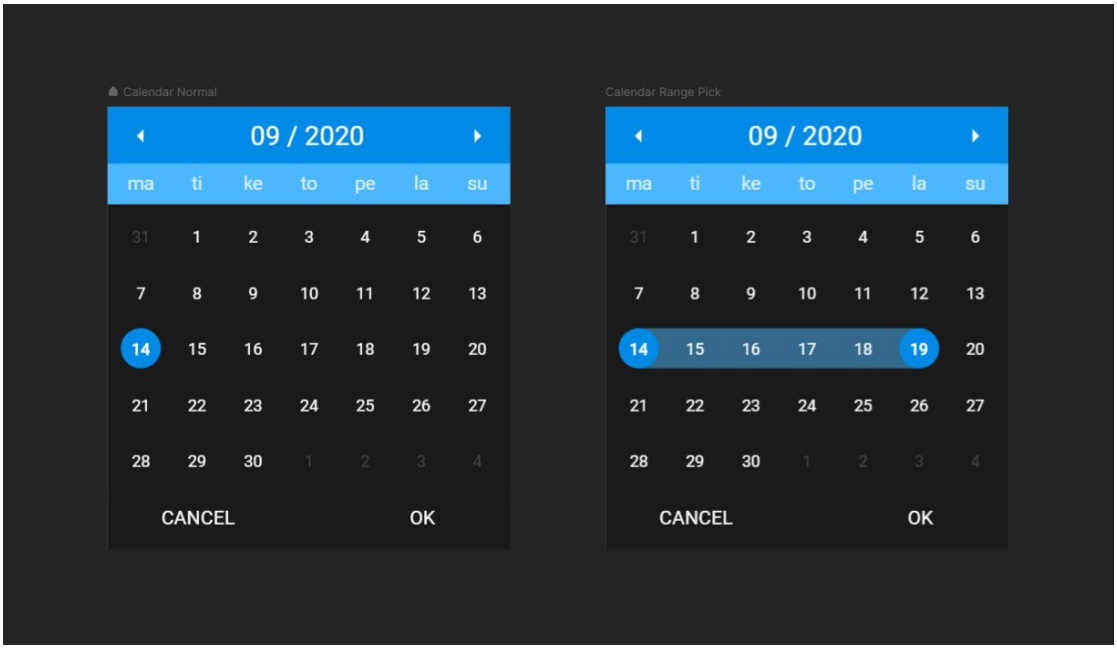
5.3 Komponentin suunnittelu

Kalenterikomponentin teon ensimmäinen vaihe oli luoda sille käyttöliittymäsuunnitelmat Framerin avulla. Kuitenkin ennen suunnittelemisen aloitusta rajasin komponentin tärkeimmät toiminnallisuudet, sekä muut oleelliset ulkoasuun vaikuttavat yksityiskohdat. Kalenterikomponentti tulee rakentumaan kolmesta osasta, joista ensimmäinen projektin päävärii (primary) taustana käyttävä otsikkopalkki (header), sisältää aktiivisen kuukauden ja vuoden näyttävän tekstin, sekä vasemmalla ja oikealla reunoilla sijaitsevat nuolipainikkeet, joiden avulla voi kuukautta vaihtaa edelliseen tai seuraavaan. Otsikkopalkissa alempana osuutena on jokaisen viikonpäivän näyttävä palkki, joka käyttää taustavärinään haaleampaa versiota projektin pääväristä.

Komponentissa keskellä taas sijaitsee itse kalenterivalikko, joka näyttää kerrallaan 35 päivänumeroa, sekä poistaa käytöstä ja tummentaa ne päivät, jotka eivät kuulu valitulle kuukaudelle. Kalenterissa on mahdollisuus valita, joko yksittäinen päivä tai ajanjakso, riippuen komponentille annettavasta asetuksesta.

Valitun päivän taustaväriksi asetetaan projektin pääväri ja ajanjakson valinnassa aloituspäivän ja lopetuspäivän väli korostetaan pääväriä käytävällä osittain läpinäkyvällä päivävalintanappien korkuisella taustalla.

Komponentin kolmantena ja viimeisenä osana on alhaalla olevat napit. Näistä napeista ensimmäinen, peruuta-nappi (cancel) nimensä mukaisesti peruuttaa valinnan ja sulkee kalenterikomponentin. Peruuta-napin viereinen hyväksymisnappi, taas päivän tai ajanjakson valinnan jälkeen, palauttaa valitun päivämäärävalinnan tiedot ja sulkee komponentin. Näistä osista koostuu kalenterikomponentti sitten kokonaisuudessaan, kuten kuvassa 13 on esitetty.

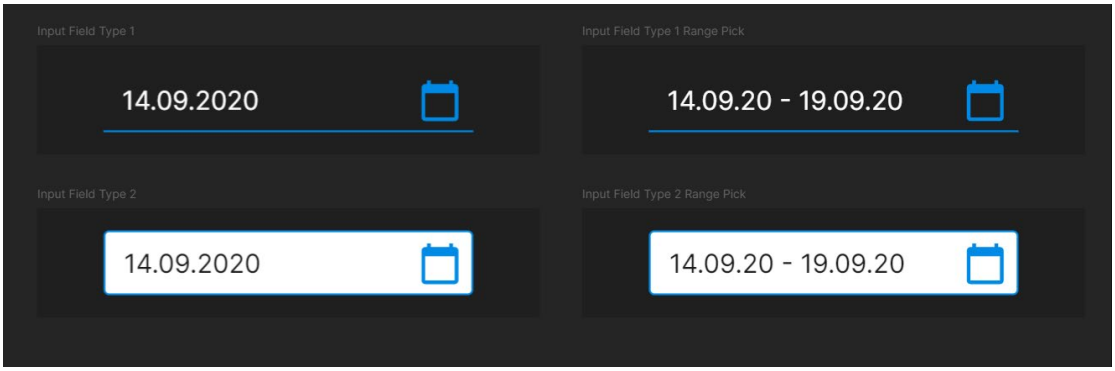


Kuva 13. Kalenterikomponentin ensimmäinen käyttöliittymäsuunnitelma Framerissa

Kalenterikomponentti toimii sovelluksen tai verkkosivun käyttöliittymässä ponnahdusikkunassa. Tämän takia komponentti tarvitsee avukseen jonkinlaisen ”laukaisimen”, jonka kautta kalenterikomponentti voidaan avata ja päästä käsiksi sen palauttamaan dataan. Monia esimerkkejä seuraten päädyin käyttämään tähän syöttökenttää (input field), jota painamalla avataan kalenterikomponentti käyttöliittymään. Syöttökentän avulla pystytään myös kalenterikomponentista tuleva päivä tai ajanjakso näyttämään käyttäjälle.

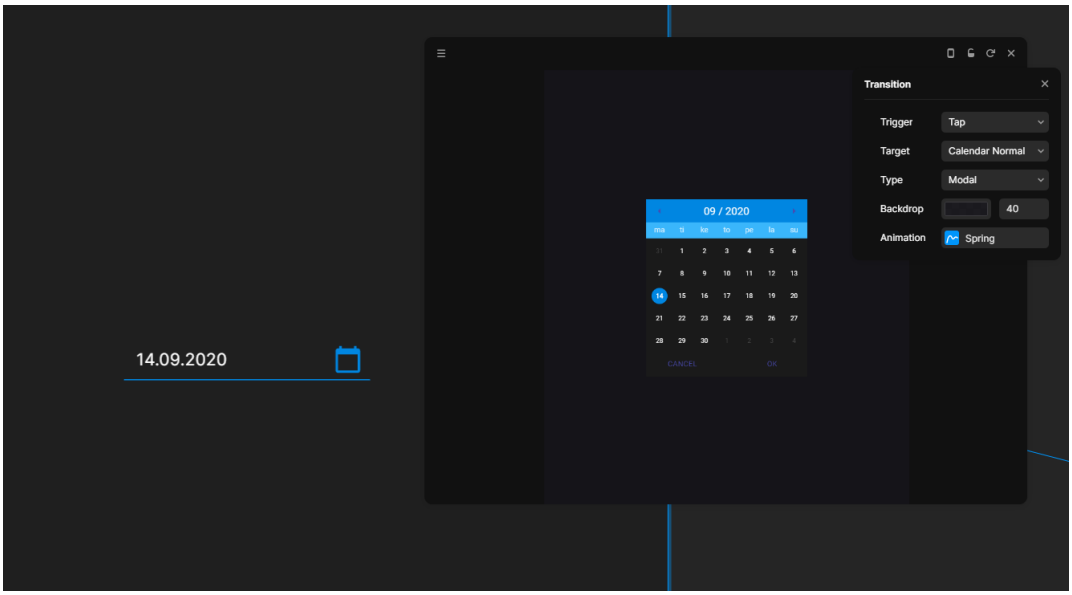
Tein syöttökentistä yhteensä 4 eri variaatiota (kuva 14), joista ensimmäiset, on varustettu kalenteri-ikonilla ja alaviivalla, jotka molemmat käyttävät projektin pääväriä. Komponenteissa tausta on läpinäkyvä ja siinä näkyvä teksti käyttää

värinään projektin yleistä tekstille määrättyä väriä. Toiset syöttökentät käyttävät ikoniensa ja reunojensa värinään myös projektin pääväriä, mutta omaavat pyöristetyt reunat, jotka ympäröivät koko komponentin ja käyttävät valkoista taustaa läpinäkyvän taustan sijaan.



Kuva 14. Syöttökenttien käyttöliittymäsuunnitelma Framerissa

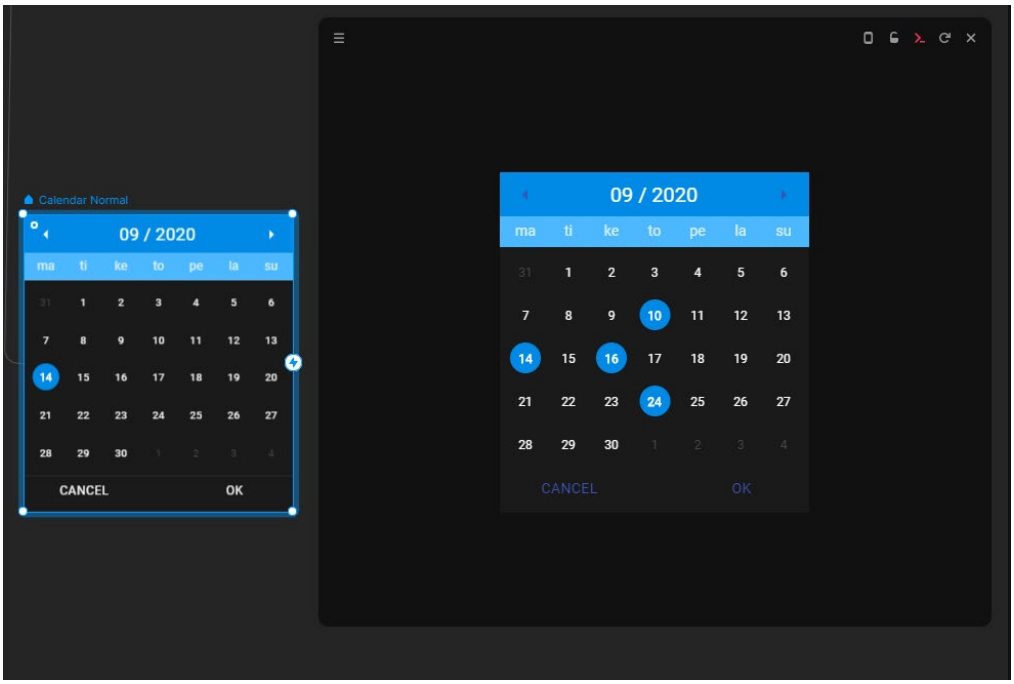
Kuten esimerkkinä kuvassa 15 näkyy, Framerin avulla pystyttiin myös demoamaan osittain syöttökentän ja kalenterikomponentin välinen toiminnallisuus, jossa syöttökenttää klikkaamalla avattaisiin kalenterikomponentti modaalin sisässä. Tämän sain aikaiseksi antamalla yhdelle syöttökentälle vuorovaikutustoiminnallisuuden (interaction), jossa komponenttia painamalla animaationäkymässä kalenterikomponentti avautuu modaalissa syöttökentän päälle.



Kuva 15. Syöttökentän kalenterin avaavan toiminnallisuuden testaus

Kalenterikomponentin Framer-suunnitelmista puuttuu toiminnallisuudet toimivan komponentin demoamiseen (kuva 16), kuten kuukauden vaihto, toimiva

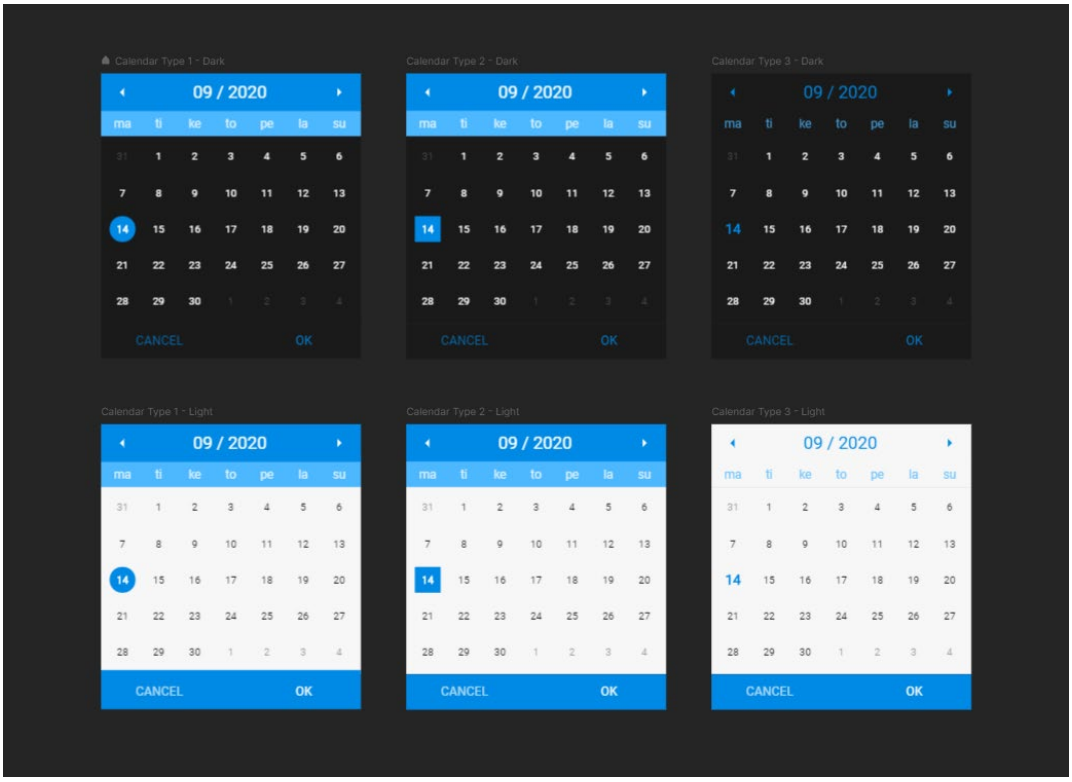
päivän tai ajanjakson valinta, sekä peruutus- ja hyväksymisnapin toiminnallisuudet. Näitä toiminnallisuuksia voidaan tulevaisuudessa hioa tarkemmin, mutta käyttöliittymäsuunnitteluprosessissa keskityn komponentin teemoitukseen, joten en tutustu Framerin animaatiotyökaluihin tämän enempää. Komponentin käyttöliittymäsuunnitelmat ovat näin ollen luotu kuvaamaan staattisten mallien avulla, vain kalenterikomponentin ulkoasua, eri asetuksilla varustettuna.



Kuva 16. Esimerkki kalenterikomponentin demoamisesta Framerissa

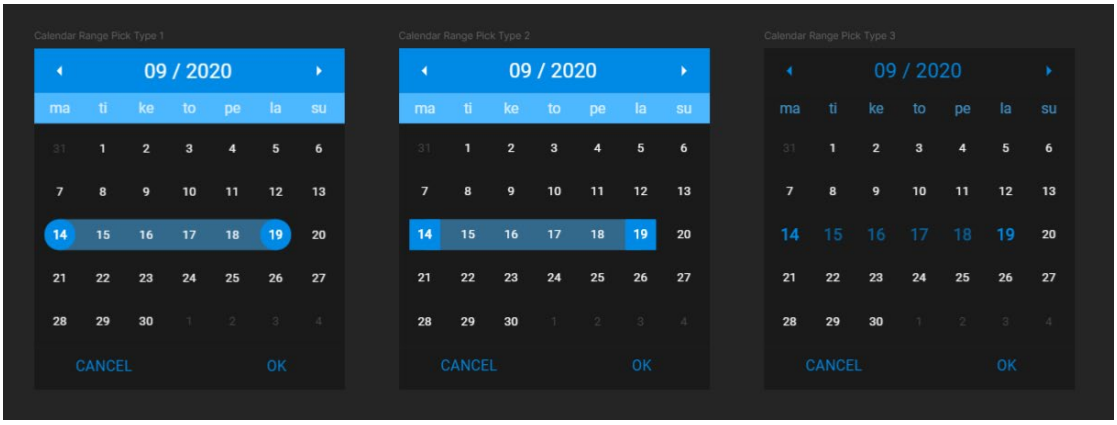
Nyt kun komponentin käyttöliittymäsuunnittelun mallin rakenne oli valmis, päätin viimeistellä sen samaan tapaan, kuin syöttökenttien käyttöliittymäsuunnitelmat. Loin ensimmäiseksi kalenterikomponentista kolme tyyppiä, joista kahdessa ensimmäisessä, otsikkopalkin tausta väritetään annetulla päävärillä ja valittu päivämäärä on korostettu, joko päävärillä väritetyllä ympyrällä tai laatikolla. Kolmas tyyppi kuvaa variaatiota kalenterista, jossa otsikon tausta väritetään taustavärillä, tehden koko komponentin taustasta saman värisen, sekä käyttämällä aikaisemmin otsikon taustassa käytettyä päävärä ja toissijaista väriä nyt sen teksteissä. Nämä kolme käyttävät myös pimeää (dark) moodia, jonka avulla määrätään komponenttien taustaväriksi käytettävän tumman harmaata.

Otin seuraavaksi vielä kopiot näistä kolmesta tyypeistä, jotka noudattavat muuten samoja teemoitusasetuksia, mutta niissä käytetään pimeän (dark) moodin sijasta valoisaa (light) moodia. Valoisan moodin ero pimeään moodin on siinä käytettävä taustan väri, joka tässä tapauksessa vaihdettiin valkoiseksi. Valoisassa moodissa erona on myös alapalkki, jossa peruutus ja hyväksymisnappi sijaitsevat, joidenka tausta väritetään päävärillä. Yleisenä muutoksena edelliseen suunnitelmaan verrattuna, päätin käyttää nappien värinä pimeässä moodissa päväriä valkoisen sijaan, jonka jälkeen saatiin luotua kuvan 17 mukainen lopputulos.



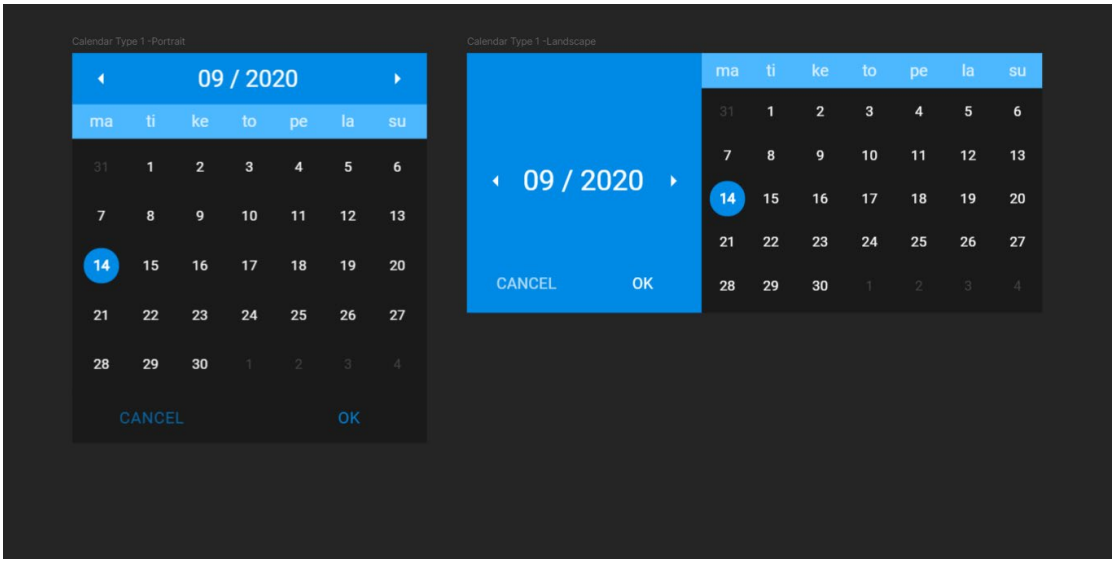
Kuva 17. Kalenterikomponentin viimeistelty käyttöliittymäsuunnitelma

Tehden muutoksia kalenterikomponentin käyttöliittymäsuunnitelmaan, järkeen kävi myös suunnitella päivitetty ajanjakson valinnan esittely (kuva 18) tyypin 2 ja 3 kalenterikomponenteille. Tyypin 2 kalenterikomponenttiin ei tarvinnut tehdä paljon muutoksia tyypin 1 kalenterikomponenttiin verrattuna, johon muokkasin vain aloitus- ja lopetuspäivien korostuksen käyttämään pehmeäkulmaista laatikkoa, ympyrän sijasta. Kolmannen tyypin ajanjakson valinnassa taas en käyttänyt ensimmäisessä ja toisessa tyypissä esiintyvää aloitus- ja lopetuspäivän välissä olevaa väriviivaa, vaan sen sijaan kaikki välipäivät korostetaan haalennetulla päävärillä.



Kuva 18. Päivitetty kalenterikomponentin ajanjakson valinnan käyttöliittymäsuunnitelma

Kalenterikomponentti tulee varmasti käytettäväksi myös mobiilisovelluksissa, jotka yleisesti pystyvät mukauttamaan käyttöliittymänsä portrait ja landscape moodeissa, jotka aktivoituvat käännettäessä mobiililaitteen, joko pysty- tai vaakatasoon. Kalenterikomponentin käyttöliittymä on tehty oletuksena portrait moodiin, mutta landscapeen se ei välttämättä mahdu. Loin kalenterikomponentista siis version, jossa yläpalkin viikonpäivien nimet näyttävä palkki ja kalenteriosa on eritelty alapalkista, sekä kuukausivalinnasta. Kuukausivalinta näkyy tässä landscape versiossa nyt vasemmalla ja sen alapuolella napit peruutukselle ja hyväksynnälle, joiden takana on tällä kertaa pääväriä käyttävä tausta. Loput kalenterikomponentin osat, eli viikonpäivien nimipalkki ja kalenteriosuus taas sijoittuvat oikealle. Näin kalenterikomponentille oli olemassa mallit portrait ja landscape moodeille (kuva 19), joiden välillä mobiilisovelluksessa hypitään, riippuen mobiililaitteen orientaatiosta.

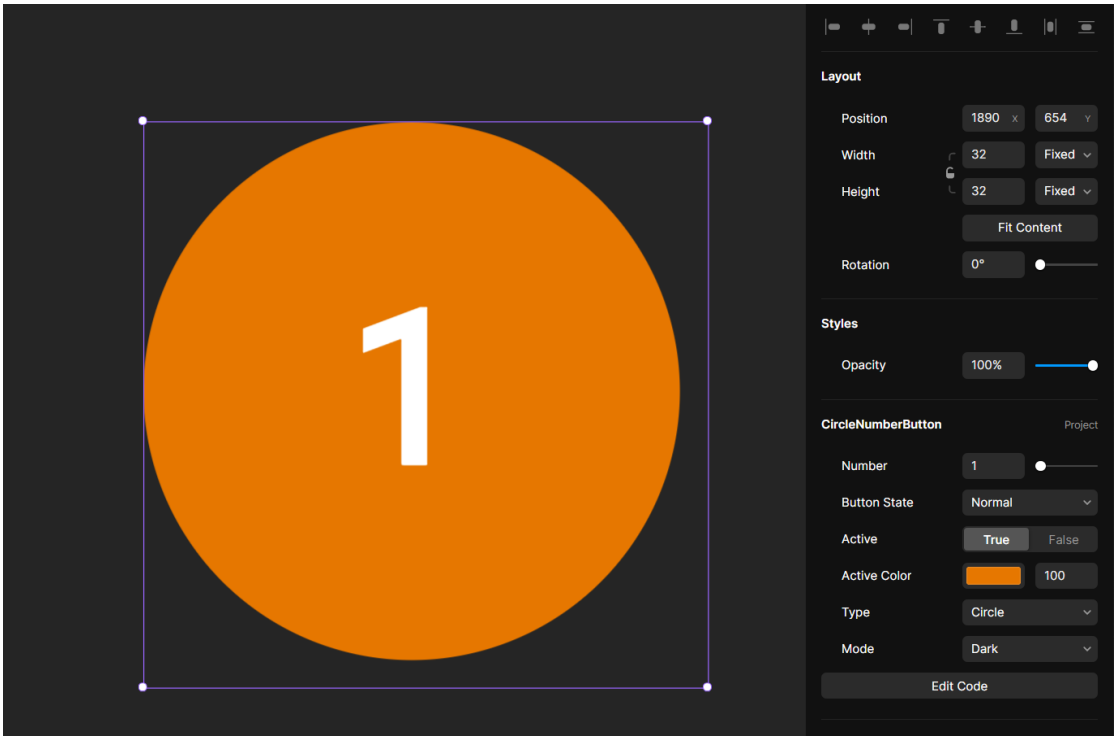


Kuva 19. Esimerkki kalenterikomponentista portrait ja landscape moodissa

Kalenterikomponentin käyttöliittymäsuunnitelmat olivat nyt valmiit ja itse komponentin luonti voitiin aloittaa. Komponentin käyttöliittymäsuunnittelun aikana luotiin muutama koodillinen komponentti, joiden avulla suunnitelmat pystyttiin rakentamaan. Monimutkaisimpana esimerkkinä luoduista komponenteista on kalenteripäivien luontiin käytetty päivämääränappi (kuva 20), johon luotiin yhteensä 6 eri kustomointiasetusta:

- Numero (Number), jossa määritetään näytettävä päivämäärän numero.
- Napin tila (Button state), josta asetetaan nappi toiminnalliseksi tai poistetaan käytöstä
- Aktiivinen (Active), josta voidaan asettaa nappi aktiiviseksi
- Aktiivisuuden väri (Active Color), jossa määrätään aktiivisessa napissa käytettävä väri
- Tyyppi (Type), jonka avulla määrätään, korostetaanko aktiivinen nappi ympyrällä, laatikolla vai väritetyllä tekstillä
- Moodi (Mode), jolla katsotaan, käytetäänkö pimeää vai valoisaa moodia, eli tässä tapauksessa onko teksti valkoista vai mustaa

Näiden asetusten avulla pystyttiin päivämääränappien ulkoasua helposti muuttamaan ja yhdessä muiden käytettyjen komponenttien kanssa, saatiin aikaiseksi malleja, joilla kalenterikomponentin eri tiloja pystyttiin kuvaamaan.



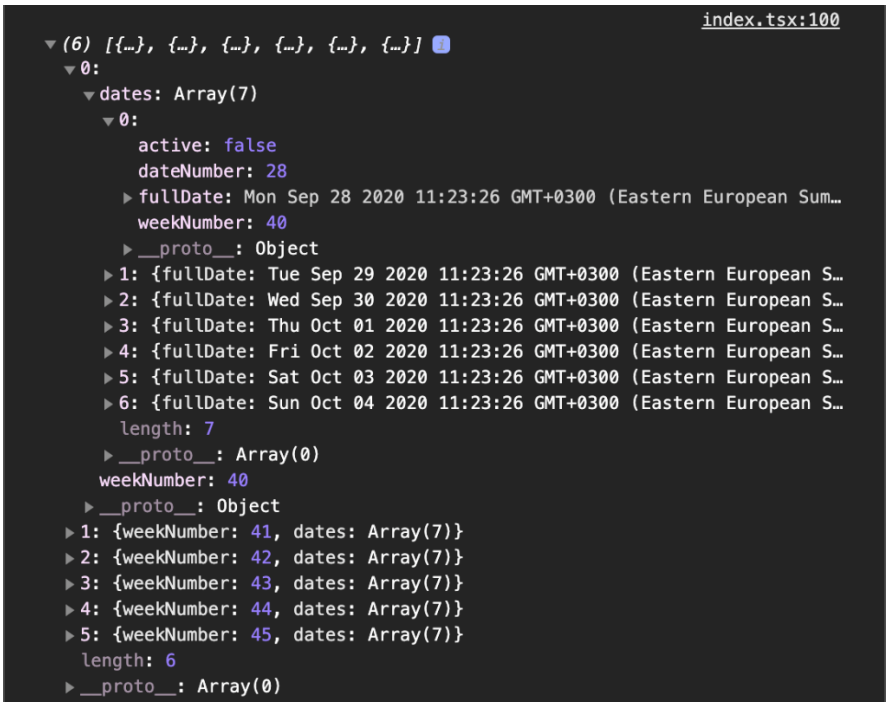
Kuva 20. Kalenterikomponentin päivämäärä napin asetukset

5.4 Komponentin luontiprosessi

Kalenterikomponentin luonnin aloitin ensimmäiseksi luomalla uuden tiedoston hiekkalaatikkoprojektin komponenttikansioon ja rakentamalla yksinkertaisella JSX-koodilla kalenterikomponentin otsikkopalkin, kalenteriosuuden ja alapalkin tyhjinä laatikoina, joiden sisään kunkin osan sisältö myöhemmin laitetaan. Komponentin korkeimpana prioriteettina on tietenkin itse kalenteriosuus, joten ennen muiden toiminnallisuuksien tekoa, lähdin kaavailemaan, miten kalenteripäivät haetaan. Päädyin hakemaan päivät käyttämällä hyväksi JavaScriptin oman Date-objektin toiminnallisuuksia, jonka avulla saadaan haettua näytettävät 35 päivää.

Viikot alkavat tyypillisesti maanantaista ja jos kuukausi alkaa esimerkiksi torstaista halutaan näyttää myös edellisen kuukauden päivät viimeiseen maanantaihin asti. Date-objektissa onkin olemassa toiminto, jolla saadaan valitun päivän päivänumero, joka antaa päivälle numeroarvon 0 ja 6 väliltä, riippuen mikä viikonpäivä on kyseessä. Siispä päivien haku saatiin toimimaan antamalla 35 kierrosta käyvän for-loopin sisään aloituspäivä, joka laitetaan aluksi alkamaan nykyisen kuukauden ensimmäisestä päivästä, josta mennään takaisin vielä nykyisen viikonpäivän päivänumeron verran, jotta saadaan for-loop alkamaan aina maanantaista, oli se nykyinen tai edellinen kuukausi. Joka kierroksella päivää kasvatetaan ja ulkoiseen taulukkoon (array) työnnetään objekti, joka sisältää tiedot senhetkisen päivän numerosta, kokonaisesta päiväobjektista, sekä viikon numerosta, joka haetaan erillisen funktion avulla.

Samalla kun päivät käydään läpi, rakennetaan myös viikoille oma taulukko, johon lisätään uusi rivi, aina kun uusi viikkonumero löydetään. Kun päivät on työnnetty taulukkoonsa, käydään se vielä uudestaan läpi ja jaotellaan päivät viikkotaulukossa vastaaviin viikkoihin, niissä olevan weekNumber-arvon perusteella. Huomasin tässä vaiheessa, että suunnittelemani 7 x 5, kalenterinäkymä ei toimi, jos viikot alkaisivat lauantaista tai sunnuntaista ja kuukaudet olisivat 30 tai 31 päivää pitkiä, koska viimeiset päivät loisivat tällä tyylillä uuden rivin yhdelle tai kahdelle päivälle, joka pistäisi ilkeästi silmään. Päätin siis nostaa kierrosten määrän 42:een, jolloin kalenteriosuudessa näytettäisiin vielä yksi viikko lisää, jonka jälkeen lopputuloksena syntyy kuvan 21 mukainen taulukko.



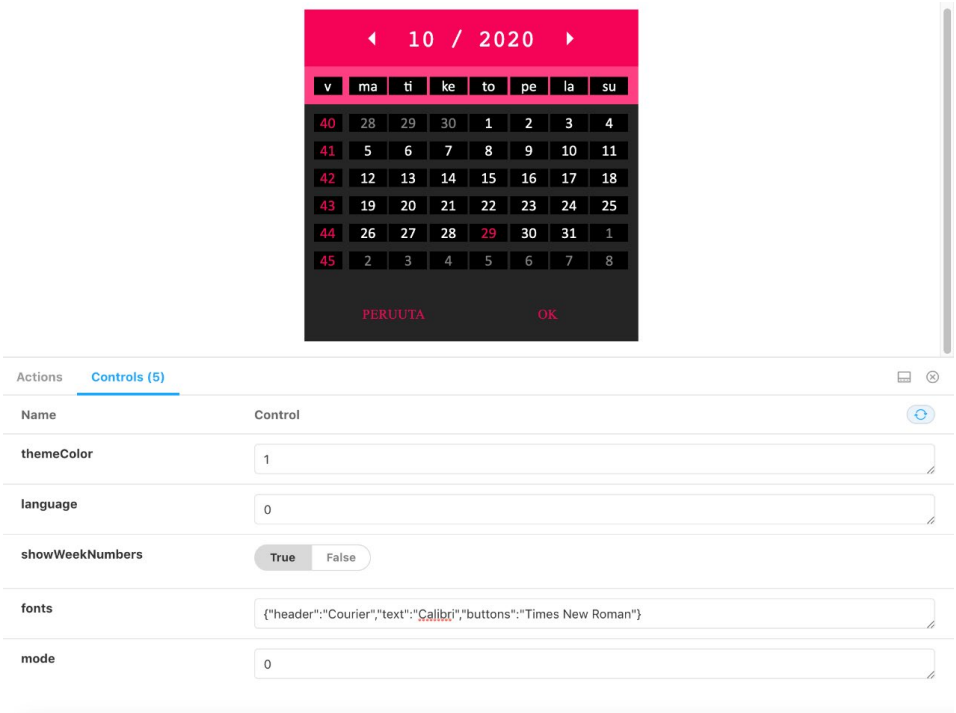
Kuva 21. Viikkotaulukon rakenne selaimen konsoliin tulostettuna

Kalenteripäivien taulukon ollessa valmis, pystyin seuraavaksi siirtymään kalenterikomponentin rakenteen tekoon. Aloitin luonnollisesti komponentin kalenteriosuudesta, jossa käydään viikkotaulukko läpi ja tulostetaan JSX-koodissa päällekkäin jokaisen viikon päivät. Päivien tekstin väri vaihdetaan valkoiseksi tai harmaaksi riippuen siitä kuuluuko kyseinen päivä valitun kuukauden sisään ja värjätään vielä päävärillä, jos kyseinen päivä on sama kuin nykyinen päivä. Viikkorivillä on mahdollisuus näyttää päivien lisäksi myös kyseisen viikon numero, joka näytetään tai piilotetaan riippuen komponentille annetusta boolean-tyyppisestä showWeekNumbers-parametrista.

Kalenteriosuuden ollessa valmis siirryin seuraavaksi otsikkopalkin kimppuun. Ensimmäisenä tein viikonpäivien nimet näyttävän rivin, joka riippuen showWeekNumbers-parametrista, näyttää niiden vieressä vasemmalla myös viikkonumeroiden sarakkeen otsikon. Valitut päivämäärien nimet ja viikkonumero sarakkeen otsikko ovat simppeleitä string-tyyppisiä arvoja, jotka haetaan omasta objektistaan, riippuen kalenterikomponentille annetusta language-parametrista, jolla voidaan komponentissa käytettävä kieli vaihtaa suomen ja englannin välillä. Otsikkopalkissa ylempänä tarvitaan vielä nykyisen kuukauden ja vuoden näyttävä teksti, joka on yksinkertainen tyylitelty teksti, jonne tilametodeiksi tallennetut kuukausi ja vuosi tulostetaan. Nuolipainikkeet on tehty

käyttämällä Material-UI:n ikonipainikkeita (IconButton), joiden sisälle laitoin, napista riippuen vasemmalle tai oikealle osoittavan nuoli-ikonin, jotka jouduin asentamaan erikseen projektiin, kun ne eivät ilmeisesti tulleet TSDX:n mukana.

Viimeisenä komponentin osana oli kalenteriosan alapuolella olevat napit, jotka loin käyttämällä Material-UI:n omia nappikomponentteja. Nappien tekstin laitoin myös määräytymään, otsikon päivämäärien nimien lailla valitun language-parametrin perusteella. Nappien fonttia pystytään myös muuttamaan antamalla string-tyyppinen buttons-arvo, joka luetaan fonts-parametrin sisältä, jossa voidaan antaa eri fontit myös otsikkopalkissa olevalle valitun kuukauden ja vuoden näyttävälle tekstille, sekä komponentissa yleisesti käytettävälle tekstille, jota kalenteriosa ja viikonpäivien nimet näyttävä rivi käyttävät. Käyttöliittymäsuunnitelman mukaisesti kalenterikomponentille pystytään antamaan myös moodi, joka voi olla joko pimeä tai valoisa, mikä vaihtaa komponentin kalenteriosuuden taustavärin ja riippuen annetusta moodista värittää komponentin napit sisältävän alapalkin taustan, joko tummana pimeässä moodissa tai otsikkopalkin tapaan päävärillä valoisassa moodissa. Näin saatiin valmiiksi kalenterikomponentin ensimmäinen versio (kuva 22), joka sisältää asetukset teemoitusvärin vaihdolle, kielelle, fonteille, moodille, sekä vaihtoehdon näyttää tai piilottaa viikkonumerot palkki.



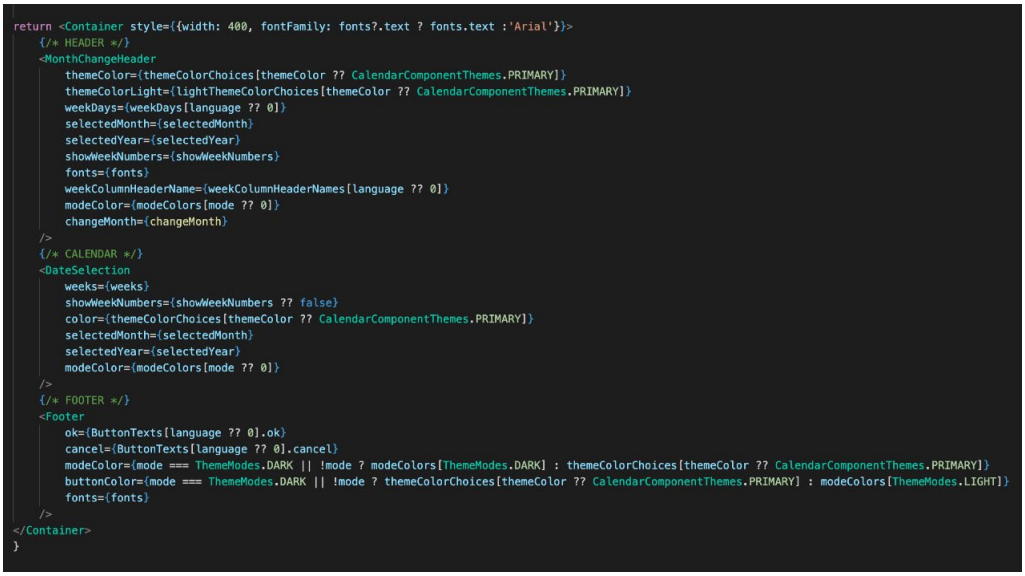
Kuva 22. Kalenterikomponentin ensimmäinen versio Storybookissa

Päivitin ensimmäistä versiota vielä lisäämällä otsikkopalkin nuoli-ikoni napeille toiminnallisuuden vaihtaa valittua kuukautta ja vuotta. Tähän loin toiminnallisuuden, jossa napit painettaessa kutsuvat changeMonth-funktiota (kuva 23), jolle napit antavat niiden suuntaa vastaavan arvon. Nappien antama arvo käydään funktiossa läpi switch-lauseen avulla, jossa riippuen annetusta arvosta valittua kuukautta vähennetään tai kasvatetaan. Rajatilanteissa, siis vähennyksessä valitun kuukauden ollessa ensimmäinen kuukausi tai lisäyksessä sen ollessa viimeinen, muutetaan ne vastaamaan vähennyksessä vuoden viimeistä kuukautta ja päinvastoin lisäyksessä. Rajatilanteissa muutetaan myös valittua vuotta vastaamaan, joko edellistä tai seuraavaa vuotta.

```
/**
 * Updates selected month and year depending on clicked arrow button
 * @param direction
 */
const changeMonth = (direction: MonthChangeDirection) => {
    switch (direction) {
        case MonthChangeDirection.BACK:
            setSelectedMonth(selectedMonth === 0 ? 11 : selectedMonth - 1);
            selectedMonth === 0 && setSelectedYear(selectedYear - 1);
            break;

        case MonthChangeDirection.FORWARD:
            setSelectedMonth(selectedMonth === 11 ? 0 : selectedMonth + 1);
            selectedMonth === 11 && setSelectedYear(selectedYear + 1);
            break;

        default:
            break;
    }
}
```

Kuva 23. Kuukauden vaihto funktio

Ensimmäinen versio kalenterikomponentista sisälsi tässä vaiheessa pitkän määrän koodia, joka vaikeutti kohtalaisesti tekoa. Päätin siis luoda kalenterikomponentin alle components-alikansion, johon loin tiedostot, jokaiselle komponentin kolmelle osalle. Kopioin pääkoodissa olevat osille kuuluvat JSX-osuudet, muuttujat ja liittymät (interface) näihin tiedostoihin. Jouduin muokkaamaan koodeja jonkin verran ja kun nämä komponentit eivät enää olleet suorassa yhteydessä pääkomponentissa olevien funktioiden ja muuttujien kanssa, jouduttiin ne antamaan ominaisuusparametreina komponenteille. Tämän jälkeen kalenterikomponentin ensimmäinen versio toimi taas normaalisti, kun otsikkopalkki, kalenteriosio ja napit sisältävä alapalkki olivat omissa tiedostoissaan ja lopputuloksena saatiin karsittua pääkoodista monta riviä (kuva 24).



```
return <Container style={{width: 400, fontFamily: fonts?.text ? fonts.text :'Arial'}}>
    {/* HEADER */}
    <MonthChangeHeader
        themeColor={themeColorChoices[themeColor ?? CalendarComponentThemes.PRIMARY]}
        themeColorLight={lightThemeColorChoices[themeColor ?? CalendarComponentThemes.PRIMARY]}
        weekDays={weekDays[language ?? 0]}
        selectedMonth={selectedMonth}
        selectedYear={selectedYear}
        showWeekNumbers={showWeekNumbers}
        fonts={fonts}
        weekColumnHeaderName={weekColumnHeaderNames[language ?? 0]}
        modeColor={modeColors[mode ?? 0]}
        changeMonth={changeMonth}
    />
    {/* CALENDAR */}
    <DateSelection
        weeks={weeks}
        showWeekNumbers={showWeekNumbers ?? false}
        color={themeColorChoices[themeColor ?? CalendarComponentThemes.PRIMARY]}
        selectedMonth={selectedMonth}
        selectedYear={selectedYear}
        modeColor={modeColors[mode ?? 0]}
    />
    {/* FOOTER */}
    <Footer
        ok={ButtonTexts[language ?? 0].ok}
        cancel={ButtonTexts[language ?? 0].cancel}
        modeColor={mode === ThemeModes.DARK || !mode ? modeColors[ThemeModes.DARK] : themeColorChoices[themeColor ?? CalendarComponentThemes.PRIMARY]}
        buttonColor={mode === ThemeModes.DARK || !mode ? themeColorChoices[themeColor ?? CalendarComponentThemes.PRIMARY] : modeColors[ThemeModes.LIGHT]}
        fonts={fonts}
    />
</Container>
}
```

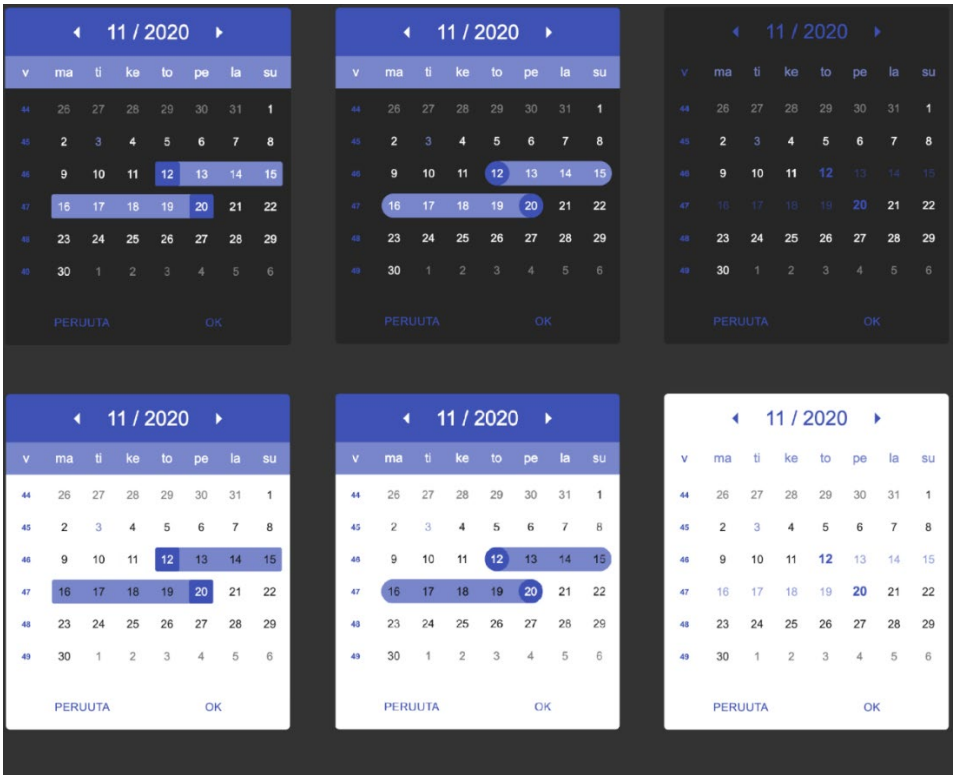
Kuva 24. Kalenterikomponentin JSX osuus

Pohjan ollessa valmis seuraava vaihe oli kalenterikomponentin tyyppi ja moodi vaihtoehtojen teko, sekä kalenterikomponentin kalenteriosassa näytettävien päivänappien komponentin luonti. Aloitin prosessin ensin luomalla erillisen komponentin ja kopioimalla siihen kaikki olemassa olevat näytettävien päivien toiminnallisuudet. Kun komponentti päivänapeille oli luotu, lähdin tekemään toiminnallisuuksia tyyppien vaihtoehdoille, jossa komponentille annetaan yksi kuudesta valittavasta tyypistä. Tyypit loin noudattamalla Framerissa tehtyjen suunnitelmien mukaisesti, joista kaksi ensimmäistä värittävät yläpalkin kuukausivalinnan päävärillä ja sen alemman viikonpäivien nimet näyttävän palkin päävärin vaaleammalla versiolla. Nämä kaksi ensimmäistä tyyppiä ovat kohtalaisen samannäköisiä, mutta eroavat toisistaan käyttämällä valitun päivän reunojen pyöristyksessä eri arvoja, jossa ensimmäinen käyttää vain vähäsen pyöristettyjä reunoja tehden päivävalinnan taustasta pehmeäreunaisen laatikon ja toinen näyttää valinnan taustan pyöreänä pallona. Kolmas tyyppi valinta eroaa kahdesta ensimmäisestä käyttämällä yläpalkin taustavärinä samaa väriä, jota käytetään kalenteriosassa ja napit sisältävässä alapalkissa ja värittää siinä olevat päivien numerot päävärillä, jolla myös kahden edellä mainitun tyypin päivävalinnan taustat väritetään.

Tyypeistä oli nyt olemassa 3 erilaista vaihtoehtoa, joita tein vielä 3 lisää. Nämä olivat kopioita olemassa olevista tyypeistä, mutta terävien reunojen sijasta käyttivät kalenterikomponentissa pehmennettyjä reunoja. Kun olin saanut tehtyä 6 eri vaihtoehtoa tyypeille, siirryin kalenterikomponentin moodien

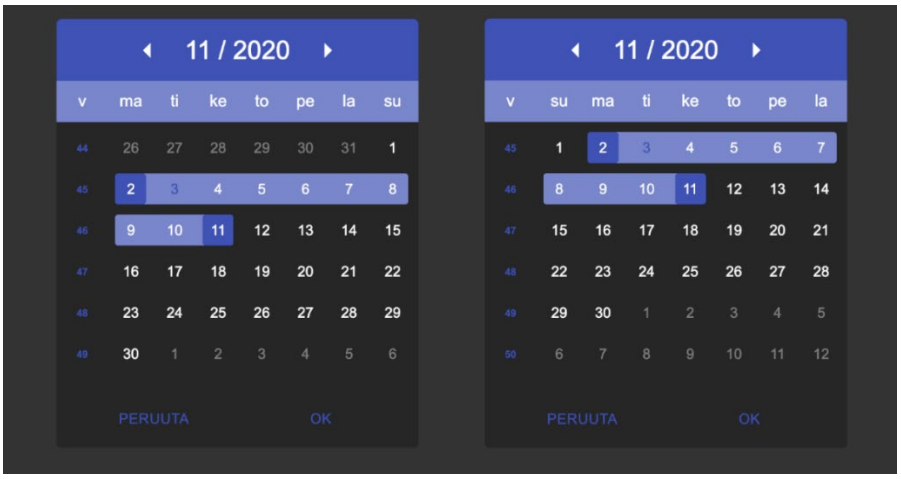
toiminnallisuuksien tekoon. Komponentissa käytettäviä moodeja tein käyttöliittymäsuunnitelmien mukaisesti 2, joista ensimmäinen ja oletuksena käytettävä, pimeä (dark) moodi värittää kalenteriosan ja alapalkin taustan todella tummalla harmaalla, sekä myös yläpalkin, riippuen valitusta tyypistä. Pimeän moodin kaveriksi loin valoisan (light) moodin, joka taas värittää taustat valkoisella. Framerissa tehdyissä suunnitelmissa olin antanut valoisan moodin napit sisältävälle alapalkille antanut taustaväriksi päävärin, mutta päätin kuitenkin käyttää sen sijasta valkoista, jotta valoisan ja pimeän moodien design ei eroaisi liikaa toisistaan.

Kalenterikomponentissa olin suunnitellut olevan olemassa toiminnallisuus myös ajanjakson valinnalle, jolle voidaan antaa vaihtoehtoinen boolean-tyyppisen ominaisuusparametrin. Parametrin arvon ollessa tosi, voidaan kalenterikomponentista valita aloituspäivän lisäksi myös lopetuspäivä, joiden arvot tallennetaan tilamuuttujaan ja niiden väliset päivät korostetaan, joko vaalennetulla päävärillä väritetyllä taustalla, tai värittämällä välipäivien numerot päävärillä ja lisäämällä läpinäkyvyyttä, riippuen valitusta tyypistä. Muutin vielä viikkonumeroiden kokoa pienemmäksi, jotta ne erottuisivat paremmin viikonpäivistä, jonka jälkeen kalenterikomponentista saatiin luotua monenlaisia versioita (kuva 25).



Kuva 25. Esimerkkejä kalenterikomponentin eri tyypeistä ja moodeista

Ajanjakson valinta ei tosin vielä toiminut täysin oikein ja jouduin tekemään muutamia muutoksia koodiin, joissa verrataan painetun päivänapin lähettämää arvoa olemassa oleviin aloituspäivän ja lopetuspäivän arvoihin, jonka jälkeen arvoihin tehdään tarvittavat muutokset. Ajanjakson valinnan toiminnallisuuden korjauksen jälkeen lähdin tekemään toiminnallisuutta viikon alkavan päivän vaihdolle. Tällä hetkellä kalenterissa viikot oli pistetty alkamaan maanantaista, mutta amerikkalaisessa kalenterissa viikot alkavat sunnuntaista, jota komponentti ei tällä hetkellä ota huomioon. Loin siis kalenterikomponentille uuden vaihtoehtoisen parametrin, joka otetaan huomioon viikkotaulukkoa rakentaessa. Kun viikkotaulukoiden toiminnallisuudet oli saatu päivitettyä, muokkasin vielä viikonpäivien nimet näyttävän otsikkopalkin siirtämään sunnuntain ensimmäiseksi, viikkojen alkaessa silloin, jonka jälkeen kalenterikomponentissa pystyttiin valita nyt aloituspäiväksi, joko maanantai tai sunnuntai (kuva 26).



Kuva 26. Kalenterikomponentti eri viikon aloituspäivillä

Kalenterikomponentin toiminnallisuudet alkoivat lähestyä valmista päin, mutta koodin JSX-osuus oli tällä hetkellä todella sekava, kun komponenttien CSS-koodi oli laitettu inline-tyylillä. Inline tyylillä saatiin komponentteihin suoraan upotettua CSS-koodia, jonka takia JSX-koodin koko kasvoi ja sen lukeminen vaikeutui. Tähän ongelmaan asensin projektiin avuksi styled-components-kirjaston, jonka avulla saatiin luotua tyyliteltyjä komponentteja (kuva 27). Näihin tyyliteltyihin komponentteihin saatiin laitettua tarvittavat CSS-koodit ja pystyttiin antamaan niille tarpeen mukaan omia parametreja, kuten värejä, reunojen pehmennystä ja muita arvoja.

```
const FooterContainer = styled(Container).attrs((props : BorderContainerProps) => ({
    backgroundColor: props.backgroundColor,
    borderRadius: props.borderRadius,
}))<BorderContainerProps>`
    background: ${props => props.backgroundColor};
    border-radius: ${props => props.borderRadius};
    width: 100%;
    display: flex;
    flex-direction: row;
    justify-content: space-between;

`;

const StyledButton = styled(Button).attrs((props: ColorAndFontProps) => ({
    color: props.color,
    fontFamily: props?.font ? props?.font : 'Arial',
}))<ColorAndFontProps>`
    color: ${props => props.color};
    font-family: ${props => props.font};
    width: 45%;
    height: 40px;
    margin-bottom: 10px;
    margin-top: 10px;
`;
```

Kuva 27. Napit sisältävän alapalkin komponenttien tyylit

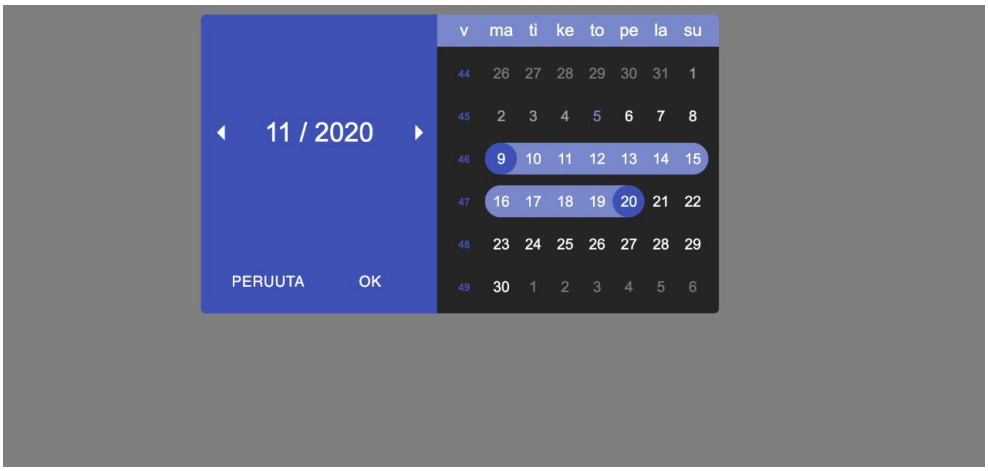
Tyylitellyt komponentit voitiin muutosten jälkeen lisätä koodin JSX-osioon (kuva 28), jossa niille tarvitsi antaa vain olennaiset CSS-tyylit ja ominaisuus-parametrit. Näin koodista saatiin tehtyä luettavampaa.

```
const Footer = (props: FooterProps) => {
    const {ok, cancel, modeColor, buttonColor, fonts, type, close, orientation, selectedDates, useDateRange} = props;
    const borderRadius = type === ThemeTypes.TYPE_3 || type === ThemeTypes.TYPE_4 || type === ThemeTypes.TYPE_6
                        ? orientation && orientation === Orientations.LANDSCAPE
                            ? '0px 0px 0px 5px'
                            : '0px 0px 5px 5px'
                        : '0px';
    const isValid = useDateRange && selectedDates.start && selectedDates.end ||
                    !useDateRange && selectedDates.start;

    return <FooterContainer backgroundColor={modeColor} borderRadius={borderRadius}>
                <StyledButton
                    color={buttonColor}
                    font={fonts?.buttons}
                    onClick={() => close(CalendarCloseOptions.CANCEL)}>
                    {cancel}
                </StyledButton>
                <StyledButton
                    disabled={!isValid}
                    color={buttonColor}
                    font={fonts?.buttons}
                    onClick={() => close(CalendarCloseOptions.OK)}>
                    {ok}
                </StyledButton>
            </FooterContainer>;
}

export default Footer;
```

Kuva 28. Alapalkin siistitty JSX-koodi

Kalenterikomponenttia tullaan käyttämään mitä todennäköisimmin myös mobiililaitteilla ja vaikka se sopii tällä hetkellä hyvin näkymään pystytasossa (portrait), vaakatasossa (landscape) mobiililaitteiden pienet ruudut katkaisevat komponentin, eikä sitä pystytä näyttämään kokonaan. Laitoin siis kalenterikomponentin alkuun tarkistuksen, jossa katsotaan ensin asentamani react-device-detect-kirjaston isMobile-funktion avulla, onko käytetty alusta mobiililaite. Funktion palauttaman arvon ollessa tosi, käynnistetään taustalla pyörivä funktio, jossa päivitetään laitteen orientaation ilmaisevan tilametodin arvoa, riippuen onko laitteen ikkunan nykyinen leveys suurempi kuin sen korkeus.

Tilametodin arvon ilmaistessa vaakatasossa olevaa mobiililaitetta, renderöidään kalenterikomponentista erilainen versio. Tässä landscape-muodossa (kuva 29), kalenterikomponentin viikonpäivien nimet näyttävä palkki irrotetaan yläpalkista ja näytetään oikealle aseteltuna kalenteriosuuden päällä. Vasemmalle asetellaan sitten kuukauden valinta osio yläpalkista, sekä peruutus- ja hyväksymisnapin sisältävä alapalkki. Yläpalkin osion korkeutta kasvatetaan, kunnes se yhdessä alapalkin kanssa vastaa oikealle aseteltujen komponenttien korkeutta. Alapalkin väri laitetaan myös vastaamaan kuukauden näyttävän yläpalkin taustaa ja reunojen pehmennyksiin tehdään tarvittavat muutokset, riippuen valitusta tyypistä.



Kuva 29. Esimerkki kalenterikomponentista landscape-muodossa

Kalenterikomponentti oli nyt kohtalaisen valmis, mutta kokonaisuudesta puuttui vielä sen laukaiseva syöttökenttä. Aluksi testasin kalenterikomponentin ponnahdustoiminnallisuutta luomalla yksinkertaisen napin, jota painaessa boolean-tyyppisen tilametodin arvo muutettiin todeksi. Laitoin seuraavaksi kalenterikomponentin modaalin sisään, joka avataan aina kun edellä mainittu tilametodin arvo on tosi. Tässä vaiheessa kalenterikomponentti oli siirretty omaan kansioon ja erotettu pääkoodista, jonka alle myös kaikki muut komponenttien koodit siirrettiin. Myös kalenterikomponentin alapalkin napeille sain vihdoin tehtyä toiminnallisuuksia, jossa sain ne lähettämään funktion kautta arvon, joka vastaa painettua nappia. Napin painallukset käydään läpi funktiossa ja riippuen annetusta arvosta, valinta joko keskeytetään, sen ollessa peruutusnapin arvo tai valittujen päivien tilametodi päivitetään sen ollessa hyväksymisnapin arvo.

Kun ponnahdustoiminnallisuus oli saatu testattua, loin syöttökentälle myös oman kansion, johon loin sille kooditiedoston. Korvasin pääkoodissa olevan napin syöttökenttäkomponentilla ja toin sille tarvittavat ominaisuusparametrit, kuten käytettävät värit, kalenterikomponentin laukaisevan funktion, valitut päivämäärät, sekä syöttökentän tyypin. Syöttökenttä ei sinänsä ole oikea syöttökenttä, vaan tyylitelty div-komponentti, jonka ulkoasu enemmänkin mukailee tyypillisiä Reactin ja Material-UI:n syöttökenttiä. Syöttökentän taustaväri, reunojen väri ja tyyli, määräytyy sille annetun tyypin mukaan, joita kalenterikomponentissa on yhteensä 6 (kuva 30). Syöttökentässä tulostetaan vielä kalenterikomponentin ollessa suljettuna, valitut päivämäärät, jossa lopetuspäivän ollessa olemassa, näytetään päivämäärien vuosista vain kaksi viimeistä lukua.

Kuva 30. Esimerkki syöttökentän valittavista tyypeistä

Kalenterikomponentista puuttui vielä yksi pieni ominaisuus, maksimi ajanjakson pituuden arvo. Loin siis kalenterikomponentille vielä yhden vaihtoehtoisen parametrin, joka käydään läpi valittujen aloitus- ja lopetuspäivän päivityksen yhteydessä. Tähän loin funktion (kuva 32), joka maksimi ajanjakson pituuden arvon ollessa olemassa, tarkistaa onko aloituspäivän ja lopetuspäivän väli suurempi kuin annettu maksimi arvo.

```
/**
 * check if range between start and end dates are within (optional) max date range
 * @param newSelectedDates
 */
const checkMaxRange = (newSelectedDates: SelectedDates) => {
    if (maxRange) {
        const maxRangeDateStart = newSelectedDates?.start && newSelectedDates?.end ? new Date() : undefined;
        newSelectedDates?.start && newSelectedDates?.end && maxRangeDateStart?.setDate(newSelectedDates.start.getDate() + maxRange);

        if (maxRangeDateStart && newSelectedDates?.end && maxRangeDateStart < newSelectedDates.end) {
            return;
        }
    }
    changeSelectedDates(newSelectedDates);
}
```

Kuva 31. Ajanjakson valinnan pituuden tarkistava funktio

6 PÄÄTÄNTÖ

Kalenterikomponentti oli nyt saatu valmiiksi ja kaikki oleelliset toiminnallisuudet onnistuttiin tekemään ajoissa. Kalenterikomponentissa on nyt teemoitusasetukset siinä käytettävien ulkoasua määrittävien vaihtoehtojen valinnoille, joiden avulla saadaan komponentti sopimaan eri projektien käyttöliittymiin. Komponentin suunnittelu ja luominen onnistui kohtalaisen kivuttomasti, tosin aikarajoitusten ja oman ohjelmointitaitoni takia jouduin sen koodissa välillä käyttämään vähemmän suotavia ratkaisuja. Kalenterikomponentista löytyy varmasti vielä paranneltavaa, kuten koodin laadun ja optimisoinnin parantaminen. Potentiaalia kehitykseen on myös, mutta lopputulos on silti tyydyttävä ja toiminnallinen.

Työn alussa suunnitelmana oli myös kalenterikomponentin implementointi muutamiin olemassa oleviin projekteihin. Mutta koska komponentin suunnittelu ja luomisprosessi kestivät odotettua kauemmin ja aloitettiin opinnäytetyön teon loppuvaiheessa, ei aikaa jäänyt lopulta tarpeeksi, että prosessi olisi voitu käydä tarpeeksi hyvin läpi. Näin ollen komponentin implementointi osaksi projekteja saa jäädä jatkokehityksen ideaksi.

Työn aikana kalenterikomponentin käyttöliittymäsuunnitteluun käytetty Framer ja komponentin visuaaliseen testaukseen käytetty Storybook tulivat jossain määrin tutuiksi. Työkalujen toiminnallisuuksin ei tosin sukellettu syvemmin ja esimerkiksi Framerin tarjoamista toiminnallisuuksista käytettiin vain sen editori- ja koodausnäkymien perustoimintoja. Käyttöliittymäsuunnitelmiin palaaminen ei ole tässä vaiheessa enää tarpeellista, joten sovelluksen tarjoamiin työkaluihin voidaan tutustua tarkemmin tulevissa projekteissa.

Työn lopputuloksena saatiin siis demottua komponentin suunnittelu- ja luomisprosessi, jonka lopputuloksena saatiin toimiva prototyyppi kalenterikomponentista. Tätä kalenterikomponenttia tullaan tulevaisuudessa varmasti hyödyntämään Mindhive Oy:n tulevissa projekteissa. Loppuun haluan sanoa vielä kiitokset lukijalle, toivottavasti tämän opinnäytetyön aihe, sekä sen suunnittelun ja toteutuksen prosessien läpikäynti oli kiinnostavaa luettavaa.

LÄHTEET

Arnold, J. 2017. Dumb Components and Smart Components. WWW-dokumentti. Saatavissa: https://medium.com/@thejasonfile/dumb-components-and-smart-components-e7b33a698d43 [viitattu 18.9.2020].

Balliauw, M. 2009. ASP.NET MVC 1.0 Quickly. E-kirja. Birmingham: Packt Publishing Ltd. Saatavissa: https://kaakkuri.finna.fi/ [viitattu 18.9.2020].

Baraniak, P. 2019. What is a Design System – Everything You Need to Know. WWW-dokumentti. Päivitetty 26.3.2019. Saatavissa: https://uxmisfit.com/2019/03/26/what-is-a-design-system-everything-you-need-to-know/ [viitattu 18.9.2020].

Bracey, K. 2018. What is Figma? WWW-dokumentti. Päivitetty 26.11.2018. Saatavissa: https://webdesign.tutsplus.com/articles/what-is-figma--cms-32272 [viitattu 25.9.2020].

Componentdriven. s.a. Component Driven User Interfaces. WWW-dokumentti. Saatavissa: https://www.componentdriven.org/ [viitattu 11.9.2020].

Eisenman, B. 2016. Learning React Native. E-kirja. Saatavissa: https://github.com/cjitendra12/javascript-ebooks-1/blob/master/%5BLearning%20React%20Native%20Building%20Native%20Mobile%20Apps%20with%20JavaScript%20Kindle%20Edition%20by%20Bonnie%20Eisenman%20-%202016%5D.pdf [viitattu 18.9.2020].

Framer. s.a.a A free prototyping tool for teams. WWW-dokumentti. Saatavissa: https://www.framer.com/ [viitattu 25.9.2020].

Framer. s.a.b Pricing. WWW-dokumentti. Saatavissa: https://www.framer.com/pricing/ [viitattu 25.9.2020].

Framer. s.a.c Framer X is here. WWW-dokumentti. Saatavissa: https://www.framer.com/blog/posts/x-release/ [viitattu 25.9.2020].

Framer API. s.a. Introduction. WWW-dokumentti. Saatavissa: https://www.framer.com/api/ [viitattu 25.9.2020].

Hacq, A. 2018. Everything you need to know about Design Systems. WWW-dokumentti. Päivitetty 22.5.2018. Saatavissa: https://uxdesign.cc/everything-you-need-to-know-about-design-systems-54b109851969 [viitattu 18.9.2020].

Honkanen, J. 2017. ReactJS. Seinäjoen ammattikorkeakoulu. Tietotekniikan koulutusohjelma. Opinnäytetyö. PDF-dokumentti. Saatavissa: https://www.theseus.fi/bitstream/handle/10024/138247/Honkanen_Joni.pdf?sequence=1 [viitattu 4.9.2020].

Jöch, D. 2018. Functional vs Class-Components in React. WWW-dokumentti. Päivitetty 29.3.2019. Saatavissa: https://medium.com/@Zwenza/functional-vs-class-components-in-react-231e3fbd7108 [viitattu 4.9.2020].

Krill, P. 2014. React: Making faster, smoother UIs for data-driven Web apps. WWW-dokumentti. Päivitetty 15.5.2014. Saatavissa: https://www.infoworld.com/article/2608181/react--making-faster--smoother-uis-for-data-driven-web-apps.html [viitattu 18.9.2020].

Material Design. s.a. Introduction. WWW-dokumentti. Saatavissa: https://material.io/design/introduction [viitattu 25.9.2020].

Material-UI. s.a.a Usage. WWW-dokumentti. Saatavissa: https://material-ui.com/getting-started/usage/ [viitattu 18.9.2020].

Mindhive Oy. s.a.a. Ytimessä ihmiset ja tavat toimia yhdessä. WWW-dokumentti. Saatavissa: https://www.mindhive.fi/about [viitattu 9.10.2020].

Mindhive Oy. s.a.b mindleap. WWW-dokumentti. Saatavissa: https://www.mindhive.fi/mindleap [viitattu 9.10.2020].

Mindhive Oy. s.a.c mindsolutions. WWW-dokumentti. Saatavissa: https://www.mindhive.fi/mindsolutions [viitattu 9.10.2020].

Mindhive Oy. s.a.d mindpower. WWW-dokumentti. Saatavissa: https://www.mindhive.fi/mindpower [viitattu 9.10.2020].

Mindhive Oy. s.a.e mindtrack. WWW-dokumentti. Saatavissa: https://www.mindhive.fi/mindtrack [viitattu 9.10.2020].

Purdila, A. 2017. What is Sketch and who is it for? WWW-dokumentti. Päivitetty 7.11.2017. Saatavissa: https://webdesign.tutsplus.com/tutorials/what-is-sketch-and-who-is-it-for--cms-29832 [viitattu 25.9.2020].

React s.a. Components and Props. WWW-dokumentti. Saatavissa: https://reactjs.org/docs/components-and-props.html [viitattu 18.9.2020].

React. s.a. Higher-Order Components. WWW-dokumentti. Saatavissa: https://reactjs.org/docs/higher-order-components.html#gatsby-focus-wrapper [viitattu 18.9.2020].

React Native Paper. s.a. Cross-platform Material Design for React Native. Saatavissa: https://callstack.github.io/react-native-paper/ [viitattu 18.9.2020].

Storybook. s.a.a Introduction to Storybook for React. WWW-dokumentti. Saatavissa: https://storybook.js.org/docs/react/get-started/introduction [viitattu 2.10.2020].

Storybook. s.a.b What's a Story. WWW-dokumentti. Saatavissa: https://storybook.js.org/docs/react/get-started/whats-a-story [viitattu 2.10.2020].

KUVALUETTELO