OULUN AMMATTIKORKEAKOULU

Juri Haataja

ERISTETTY KÄYTTÖLIITTYMÄN KEHITYS

Käyttöliittymän kehitys, testaaminen sekä dokumentointi eristetyssä ympäristössä

ERISTETTY KÄYTTÖLIITTYMÄN KEHITYS

Käyttöliittymän kehitys, testaaminen sekä dokumentointi eristetyssä ympäristössä

Juri Haataja
Opinnäytetyö
2023 Syksy
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä(t): Juri Haataja
Opinnäytetyön nimi: Eristetty käyttöliittymän kehitys
Työn ohjaaja(t): Jouni Juntunen
Työn valmistumislukukausi ja -vuosi: Syksy 2023 Sivumäärä: 29

Tämän opinnäytetyön tavoitteena oli ottaa käyttöön sekä vahvistaa visuaalisen regressiotestauksen toimivuus käyttäen React- sekä Storybook-ohjelmistokoodikirjastoja sekä Chromatic-pilvipalvelua. Idea aiheen tutkimiseen tuli omasta kiinnostuksesta käyttöliittymäkehitykseen, koettuani ettei pelkkä käyttöliittymän ohjelmistokoodipohjainen testaaminen tuntunut riittävältä ratkaisulta oikeanlaisen ulkoasun varmistamiseksi.

Visuaalisen regressiotestauksen suorittaminen Storybookilla vaatii syvällistä ymmärrystä ohjelmistokoodikirjaston dokumentoinnista, joka oli helposti saatavilla Storybookin virallisilta verkkosivuilta. Lisäksi erilaiset blogipostaukset, jotka käsittelevät visuaalista regressiotestausta Storybookilla, tarjosivat oleellista tietoa käytännön kokemuksista ja parhaista käytännöistä. Yhdistelemällä Storybookin virallisen dokumentaation tarkkuuden ja blogipostauksissa jaetuista käytännön vinkeistä saatavan käytännön näkemyksen, visuaalisen regressiotestauksen toteuttaminen Storybookilla oli perusteellinen ja luotettava prosessi.

Lopputuloksena syntynyt visuaalisen regressiotestauksen putkisto (engl. pipeline) osoittautui toimivaksi ratkaisuksi varmistamaan käyttöliittymäkomponenttien ulkoasun ja visuaalisen eheyden ohjelmistokoodimuutosten välillä. Itse Storybook-kirjaston käyttöönotto tarinoineen, lukuisine laajennuksineen ja määrittelyineen osoittautui jopa yllättävän työlääksi, mihin toivoisinkin yksinkertaistamista esimerkiksi tarinoiden luomisen automatisoinnin muodossa.

Asiasanat: React, Storybook, Chromatic, regressiotestaus, käyttöliittymäkehitys

ABSTRACT

Oulu University of Applied Sciences
Computer Science Degree Program, Software Development

Author(s): Juri Haataja
Title of thesis: Isolated user interface development
Supervisor(s): Senior Lecturer Jouni Juntunen
Term and year when the thesis was submitted: Autumn 2023 Number of pages: 29

This thesis' objective was to implement and verify the use of visual regression testing using React- and Storybook -software libraries as well as Chromatic cloud service. The idea for the research came from my own interest in user interface development, when I personally felt that verifying only the user flow while creating user interfaces wasn't enough, and I wanted to add visual testing as well.

Running visual regression testing with Storybook requires deep diving into the software library's documentation, which was easily available through their official website. In addition to the official documentation, I also read some blog posts that made the topic easier to understand for me. By going through official and unofficial sources, setting up and using Storybook's visual regression testing ended up being a solid tool and reliable process.

The result is a pipeline consisting of documenting user interfaces with Storybook and handling visual regression testing with Storybook's Chromatic cloud service. Downside of this pipeline is added complexity with Storybook's overhead to creating interfaces, which I hope can be partly, if not fully, automated in the future.

Keywords: React, Storybook, Chromatic, regression testing

SISÄLLYS

1 JOHDANTO

Käyttöliittymä suunnitellaan yleensä kuvanmuokkaustyökaluilla tai suoraan ohjelmistokoodissa, mutta oli lähestymistapa kumpi tahansa, suunnitteluprosessi on silti monesti haasteellinen. Kuvanmuokkaustyökaluilla kuten Figma, Adobe XD tai Gimp, on usein vaikea hahmottaa interaktiivisuuden ja dynaamisuuden elementtejä, puhumattakaan kyseisten työkalujen korkeasta oppimiskäyrästä etenkin pelkkään ohjelmointiin tottuneille. Toisaalta ohjelmistokoodilla suunnittelussa kehityssyklit voivat kestää odotettua pidempään, jotta saadaan koottua toimiva prototyyppi. Toimivaa prototyyppiä varten on lisäksi varten voitu tehdä käyttöliittymätestejä.

Vuoden 2023 viimeisellä vuosineljänneksellä törmäsin internetissä avoimen lähdekoodin Storybook-kehitysympäristöön, jonka lupauksena on tarjota ympäristö, jossa käyttöliittymäkomponentteja voidaan kehittää, dokumentoida sekä testata erikseen eristetyssä ympäristössä. Tämä mahdollistaa nopean ja iteratiivisen suunnittelun, kun komponenttien tiloja voidaan määritellä reaaliajassa, muutosten heijastuessa selaimessa näytettävään käyttöliittymään.

Tässä opinnäytetyössä tutkin Storybookin soveltuvuutta käyttöliittymän luomisessa sekä testaamisessa, keskittyen erityisesti siihen, miten se tukee komponenttien dokumentointia sekä visuaalista regressiotestausta. Tavoitteenani on arvioida Storybookin hyötyjä ja haasteita sekä selvittää sen roolia kokonaisvaltaisessa ohjelmistotestauksessa. Opinnäytetyössä ei käydä läpi varsinaista käyttöliittymän suunnittelua, kuten elementtien oikeanlaista ryhmittelyä, asettelua ja värisävyjen käyttöä, jotta fokus pysyy valitussa aihealueessa sekä sivumäärä kohtuullisena.

2 ERISTETYN KÄYTTÖLIITTYMÄN KEHITYS

Storybookin ydinidea, eli eristetty käyttöliittymän (engl. isolated user interface development) kehitys, ei itsessään ole mikään uusi teknologia, vaan enemmänkin konsepti tai lähestymistapa ohjelmistokehityksessä. Sen on mahdollistanut modernit komponenttipohjaiset käyttöliittymäkehitysrajapinnat (engl. component-based UI development frameworks), tarinapohjainen kehitys sekä visuaalinen testaaminen, minkä avulla sovelluksia voidaan kehittää entistä tehokkaammin riippumatta projektin muista osa-alueista. (Storybook a.)

2.1 Komponenttilähtöinen kehitys

Modernissa sovelluskehityksessä käyttöliittymä luodaan yksittäisistä komponenteista (engl. component-driven development, CDD), minkä tarkoituksena on tehdä komponenteista mahdollisimman riippumattomia toisistaan (ComponentDriven).

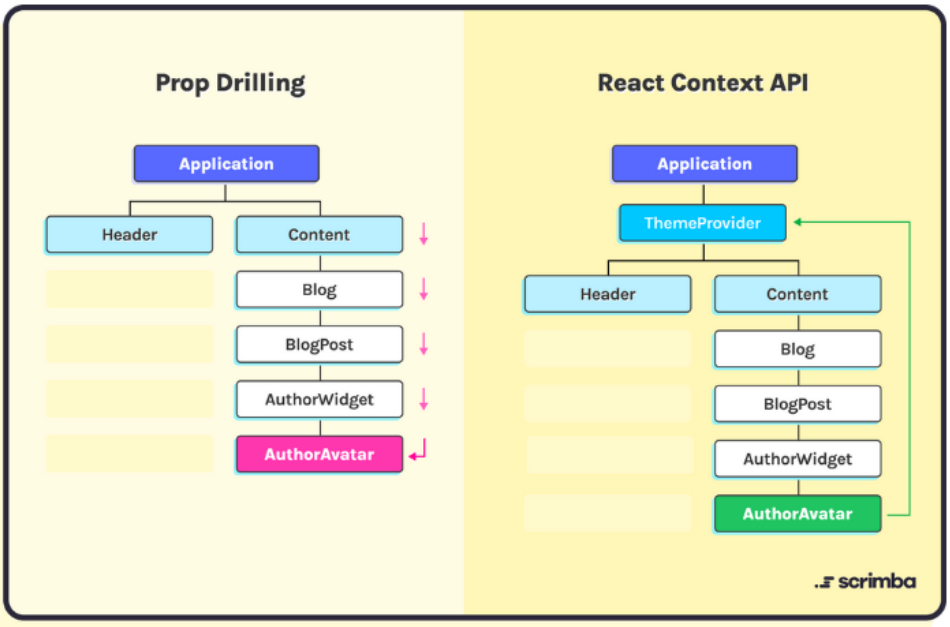
```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Taylor" />,
  mountNode
);
```

Kuva 1. Esimerkkikomponentti Reactissa (Buna 2017).

Ensimmäisiin ja samalla tämän opinnäytetyön kirjoitushetkellä suosituimpiin CDD:tä tukeviin kehyksiin kuuluvat muun muassa Angular sekä React, joista ensimmäinen on Googlen vuonna 2010 julkaiseman JavaScript-pohjaisen AngularJS-kirjaston seuraaja, ja jälkimmäinen vuonna 2013 Metan alla operoivan Facebookin julkaisema (Ivanovs 2023). Komponenttipohjaisella kehityksellä voidaan vähentää toistuvan ohjelmistokoodin määrää käyttämällä olemassa olevia komponentteja, mikä tapahtuu esimerkiksi Reactissa välittämällä niihin ominaisuuksia (engl. props) joko suoraan yläkomponentista (engl. parent component) tai erilaisista tilanhallintaan (engl. state

manager) erikoistuneista komponenteista, joista käytetään ainakin Reactin tapauksessa nimitystä konteksti (Parker 2020).



Kuva 2. "Props Drilling" eli tilan välittäminen syvästi sisäkkäiseen alikomponenttiin sen yläkomponenttien välityksellä, sekä "React Context API" eli konteksti-rajapinta, johon komponentti voi olla suoraan yhteydessä sen tarjoaman tilan saamiseksi (Abramowski 2023).

Edellä mainitut komponenttilähtöisen kehityksen tuomat edut auttavat isompien kokonaisuuksien jakamista pienempiin osiin, sillä ne rajaavat muun muassa kehittäjien vastuualueita, jolloin varsinaiseen toteutukseen ja testaamiseen jää enemmän aikaa. Tämän seurauksena ominaisuuksien toimittamisesta (engl. feature delivery) tulee varmempaa, kun komponentteja on ehditty testaamaan niin oletettua kuin myös rajatapauksiakin simuloivaa dataa vasten.

2.2 Tarinapohjainen kehitys

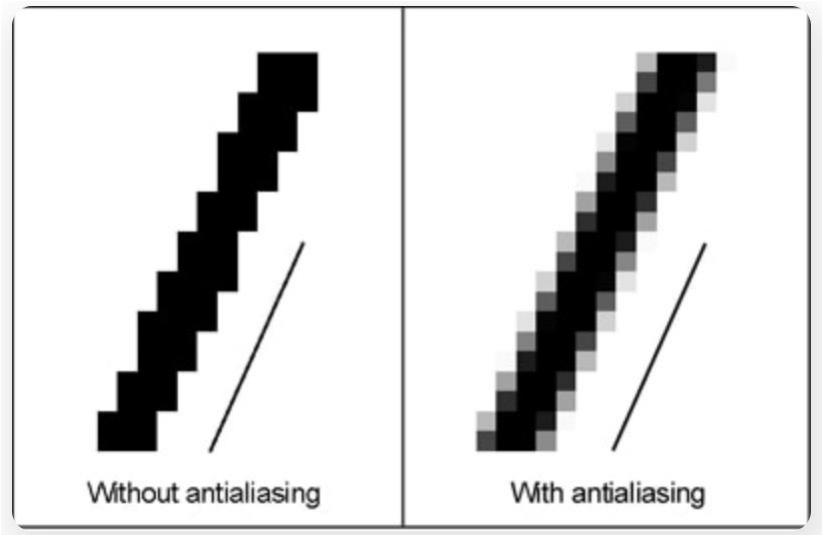
Eristetyssä käyttöliittymän kehittämisessä käytetään komponenttipohjaisia tarinoita, mitkä perustuvat käyttöliittymäkomponenttien kehittämiseen erillisinä ja itsenäisinä osina. Jokainen komponentti toimii omana tarinanaan, kuvaten erilaisia tiloja, käyttötapauksia tai näkymiä, mikä helpottaa niin komponenttien kehittämistä, testaamista kuin dokumentointia. Komponenttipohjainen tarinankerronta edistää selkeää ja hajautettua lähestymistapaa käyttöliittymäkomponenttien kehitykseen, mikä on erityisen hyödyllistä suurissa ja monimutkaisissa ohjelmistoprojekteissa. (Rehkopf.)

2.3 Visuaalinen testaus

Käyttöliittymää voidaan testata ohjelmistokooditasolla muun muassa yksikkö-, regressio- ja toiminnallisuustesteillä, mutta näistä mikään ei varmista käyttöliittymän oikeanlaista piirtymistä laitteen ruudulle. Kun halutaan varmistaa, että sovelluksen visuaalinen ulosanti vastaa olemassa olevien prototyyppien pohjalta ohjelmallisesti luotua käyttöliittymää, käytetään testaamiseen visuaalista testaamista, mikä tapahtuu yleensä vertaamalla kuvakaappauksia keskenään pikselitasolla.

Kuva 3. Visuaalinen regressiotestaus React Native Owl -kirjastolla (Padalkar 2023).

Vaikka kuvakaappausvertailussa sovelluksen halutaankin näyttävän täysin samalta kuin käyttöliittymän prototyypit, voivat visuaaliset regressiotestit epäonnistua, mikäli verrattavien kuvakaappausten välillä on vaihdettu esimerkiksi käyttöjärjestelmää tai selainta. Tämä johtuu monesti siitä, että eri käyttöjärjestelmät ja selaimet voivat käyttää eri fonttityylejä sekä reunojen pehmentämistä (engl. anti-aliasing), mitkä muuttavat kuvan pikseleitä.



Kuva 4. Reunanpehmennyksen vaikutus kuvaan. Vasemmalla ennen ja oikealla jälkeen reunanpehmennyksen (Moore 2023).

Tästä syystä monet kuvakaappausvertailuun omistautuneet kirjastot mahdollistavat raja-arvon muuttamisen, milloin pienet epätäydellisyydet kuvakaappausten välillä voidaan hyväksyä.

Niin natiivia kuin verkkosovellustakin voidaan testata visuaalisesti regressioiden varalta, joskin verkkosovellus on huomattavasti vaikeampi testata, johtuen suuremmasta määrästä aiemmin mainittuja muuttujia, mitkä vaikuttavat sovelluksen ulkonäköön selaimessa.

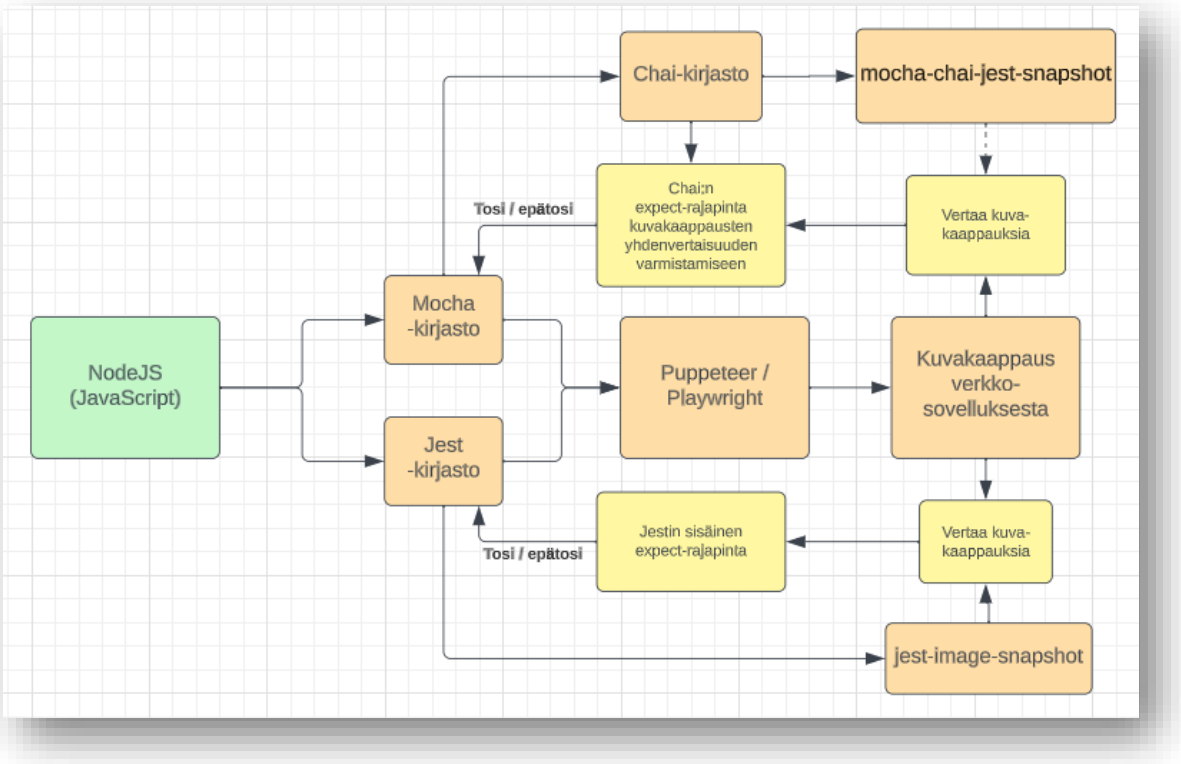
(Xu 2022)

3 KÄYTTÖLIITTYMÄN TOTEUTUS ERISTETYSSÄ YMPÄRISTÖSSÄ

Storybook on työkalu, joka mahdollistaa käyttöliittymäkomponenttien eristetyn kehityksen, dokumentoinnin sekä testaamisen. Se sai alkunsa vuonna 2016 Chromatic-tiimin kehittämänä, ja tarjoaa käyttöliittymäkomponenteille kehitysympäristön, jossa niitä voidaan tarkastella sekä testata eri tiloissa ja konteksteissa.
Storybookin ekosysteemiin kuuluu useita lisäosia (engl. addon) ja integraatioita eri teknologioiden, kuten React- Angular sekä Vue-kirjastojen kanssa. Testaustarkoituksiin Storybook tukee Jest-testauskehystä sekä Chromatic-pilvipalvelua visuaalista regressiotestausta varten, mitkä tekevät siitä tehokkaan työkalun käyttöliittymäkomponenttien kehittämiseen. Storybookin kasvavan suosion syynä ovat sen aiemmin mainittu ekosysteemi, avoin lähdekoodi sekä laaja yhteisö, jotka yhdessä tukevat nykyaikaista selkeää sekä hajautettua ohjelmistokehitystapaa. (Chromatic b.)

3.1 Visuaalinen testaaminen ennen Storybookia

Käyttöliittymäkehitys koostuu yleensä visuaalisesta prototyyppien luontityökalusta ja yhdestä tai useammasta ohjelmistopohjaisesta testauskirjastosta sekä -kehyksestä. Ensimmäisestä työkalusta näkyvimmin markkinoituja ja mitä luultavammin tästä syystä suosituimpia ovat Sketch, Adobe XD sekä Figma (Stevens 2023) mitkä tukevat useampaa samanaikaista käyttäjää, jolloin graafiset suunnittelijat, ohjelmistokehittäjät sekä mahdolliset asiakkaat voivat muokata sekä kommentoida dokumentteja yhteistyössä (Figma). Käyttöliittymän luonti on tapahtunut edellä mainittujen dokumenttien pohjalta integroidussa kehitysympäristössä (engl. Integrated Development Environment, IDE) (AWS) sekä sen testaaminen ohjelmistokielen tai -kehyksen omilla ohjelmistokirjastoilla (tästä eteenpäin käytetään nimitystä ”kirjasto”). Esimerkkeinä testauskirjastoista JavaScript-ohjelmointikielellä ovat Mocha sekä Jest (Unadkat 2023), joiden kenties suurimpana erona on niiden tarjoamat testausrajapinnat (engl. testing API); Jestistä poiketen Mocha ei tarjoa testausrajapintoja sen keskittyessä olemaan pelkkä testauskirjastoja tukeva alusta testien organisointiin sekä suorittamiseen. Jest puolestaan tarjoaa kaikki oleelliset testausrajapinnat sisäänrakennettuna, ollen näin kaikki yhdessä -ratkaisu ohjelmistotestaamiseen.



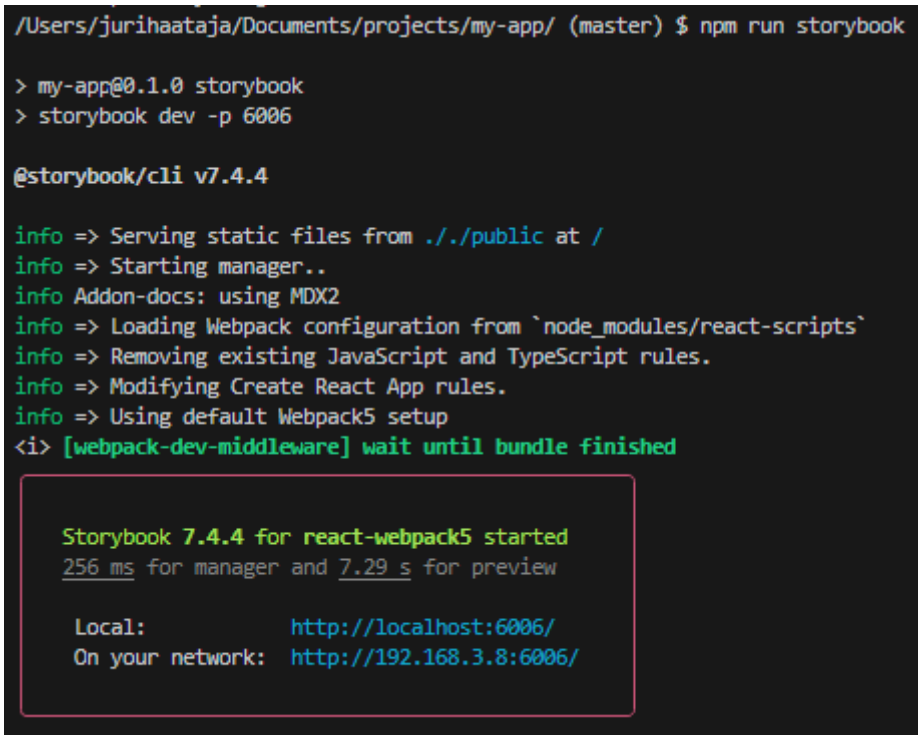
Kuva 5. Mochan ja Jestin vaiheet selaimesta otettujen kuvakaappausten vertaamisessa.

Käyttöliittymän suunnittelun sekä ohjelmistopohjaisen testaamisen lisäksi on myös suositeltavaa varmistaa käyttöliittymäkomponenttien halutunlainen piirtäminen (engl. rendering) laitteen ruudulle, jotta voidaan olla varmoja, ettei vuorovaikutteisista elementeistä, kuten painikkeista, tule käyttäjälle vaikeita tai jopa mahdottomia painaa. Visuaaliseen testaamiseen voidaan JavaScriptissä käyttää avuksi esimerkiksi Puppeteer- tai Playwright-kirjastoja, joilla voidaan simuloida käyttäjän suorittamia toimintoja kuten lomakkeiden täyttämistä ja painikkeiden painamista sekä ottaa kuvakaappauksia verkkosovelluksesta, joita voidaan sitten verrata niin Mochassa (Beleztky 2018) kuin Jestissä (Dsoudza-Sania 2023).

3.2 Storybookin asentaminen

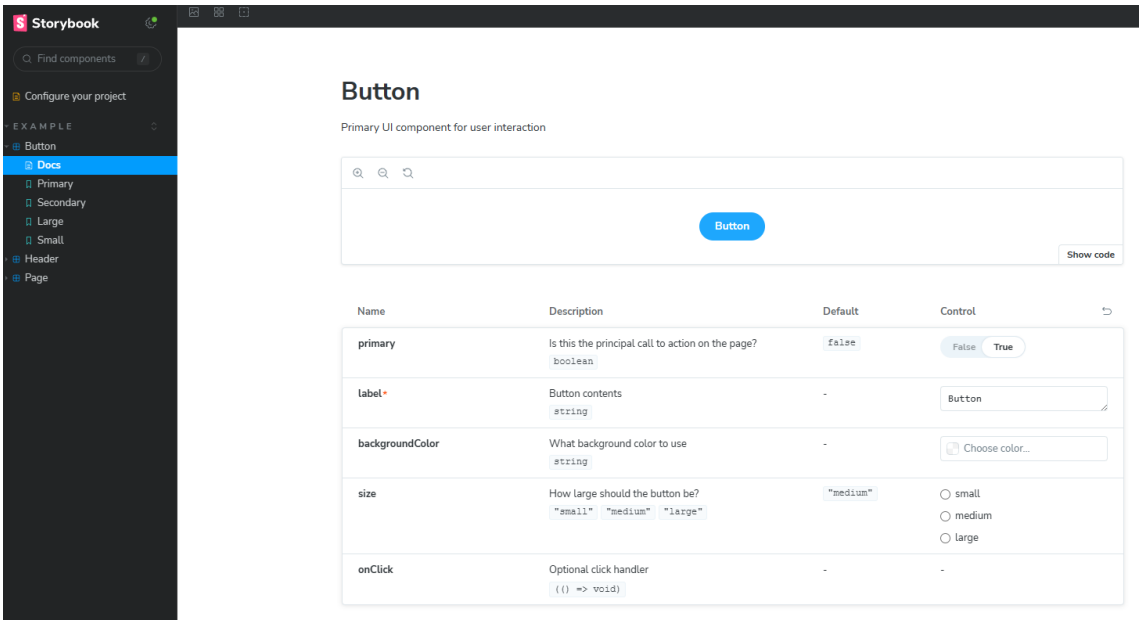
Storybookin asennuksen esivaatimuksiin kuuluu olemassa oleva komponenttipohjaisen kehityskehyksen sovellus, jonka toteutukseen tässä opinnäytetyössä valittiin verkkosovelluspohjainen ReactJS. Syy valintaan oli allekirjoittaneen aiempi kokemus kyseisen kehyksen parissa, sekä sen tarjoama nopeampi kehitys sekä testaaminen verrattuna esimerkiksi

natiiveihin mobiilisovelluksiin pohjautuvaan React Nativeen. Uusi, tyhjä sovellus ReactJS:llä luodaan kirjoitushetkellä komennolla *npx create-react-app my-app --template cra-template-pwa-typescript*, mikä luo yksinkertaisen PWA (lyh. Progressive Web Application) -sovelluspohjan, käyttäen oletuksena TypeScript-ohjelmointikieltä. (Lieschke 2021.) ReactJS-sovelluksen luomisen jälkeen voidaan asentaa Storybook komennolla *npx storybook@latest init*, mikä käyttää kirjoitushetkellä Storybookin versiota 7.4.4. ja varmistaa, mikäli koodianalyysityökalu ESLint halutaan asentaa Storybookin ohessa. (Storybook f.)



Kuva 6. Storybookin asennus ja käynnistäminen Unix-maisella käyttöjärjestelmällä.

Storybook luo asennuksen aikana *.storybook* sekä *stories* -kansiot, joista ensimmäinen sisältää pääasetustiedoston (engl. main configuration file) nimeltä *main.ts* ja tarinapiirtäjätiedoston (engl. story renderer file) nimeltä *preview.ts* (Storybook e). *Stories*-kansion alta löytyy työkalun selainnäkymässä käyttämät staattiset tiedostot, kuten kuvat ja verkkosivut, sekä esittelynä luotu painikekomponentti dokumentointeineen, joiden tarkoituksena on luoda yhdessä pohja kehittäjille Storybookin opetteluun.



Kuva 7. Painikekomponentin dokumentointisivu näyttää komponentin ensisijaisen variaation sekä komponentin pakolliset ja valinnaiset syötettävät ominaisuudet.

3.3 Lisäosien asentaminen

Storybookilla on omien verkkosivujensa mukaan sadoittain lisäosia (engl. addon), joita voidaan asentaa NPM (lyh. Node Package Manager) (NPM) -pakettien tapaan joko globaalisti eli yhteen sijaintiin, josta kaikki projektit samalla laitteella näkevät ne, tai lokaalisti eli projektikohtaisesti (Copes 2018). Storybookin laajennettavuuden vuoksi se vaatii useiden lisäosien toimivuuden takaamiseksi manuaalisen määrittelyn (engl. configuration) niin esiasetuksille (engl. preset) kuin perinteisemmille lisäosille, lataus- sekä rekisteröintivaiheiden pysyessä identtisinä molemmille. Esimerkiksi SCSS (engl. Sassy CSS), mikä on ominaisuuksiltaan monipuolisempi sekä syntaksiltaan selkeämpi kuin perinteisempi CSS (engl. Cascading Style Sheets), vaatii esiasetuksena määrittelyn perinteisten JavaScript-pohjaisten kehysten kuten Reactin sekä Angularin tapaan. Sen tapahtuu määrittelemällä konfiguraatiotiedostoon (engl. configuration file) tekstimalli, jolla tunnistaa SCSS-tiedostot muista projektin tiedostoista nimipohjaisella suodatuksella.
(Storybook h.)

```
// .storybook/main.ts

// Replace your-framework with the framework you are using (e.g., react-webpack5, vue3-vite)
import type { StorybookConfig } from '@storybook/your-framework';

const config: StorybookConfig = {
  // Replace your-framework with the framework you are using (e.g., react-webpack5, vue3-vite)
  framework: '@storybook/your-framework',
  stories: ['../src/**/*.mdx', '../src/**/*.stories.@(js|jsx|mjs|ts|tsx)'],
  addons: [
    {
      name: '@storybook/preset-scss',
      options: {
        cssLoaderOptions: {
          modules: { localIdentName: '[name]__[local]--[hash:base64:5]' },
        },
      },
    },
  ],
};

export default config;
```

Kuva 8. SCSS-esiasetuksen määrittely Storybookin main-tiedostossa (Storybook d).

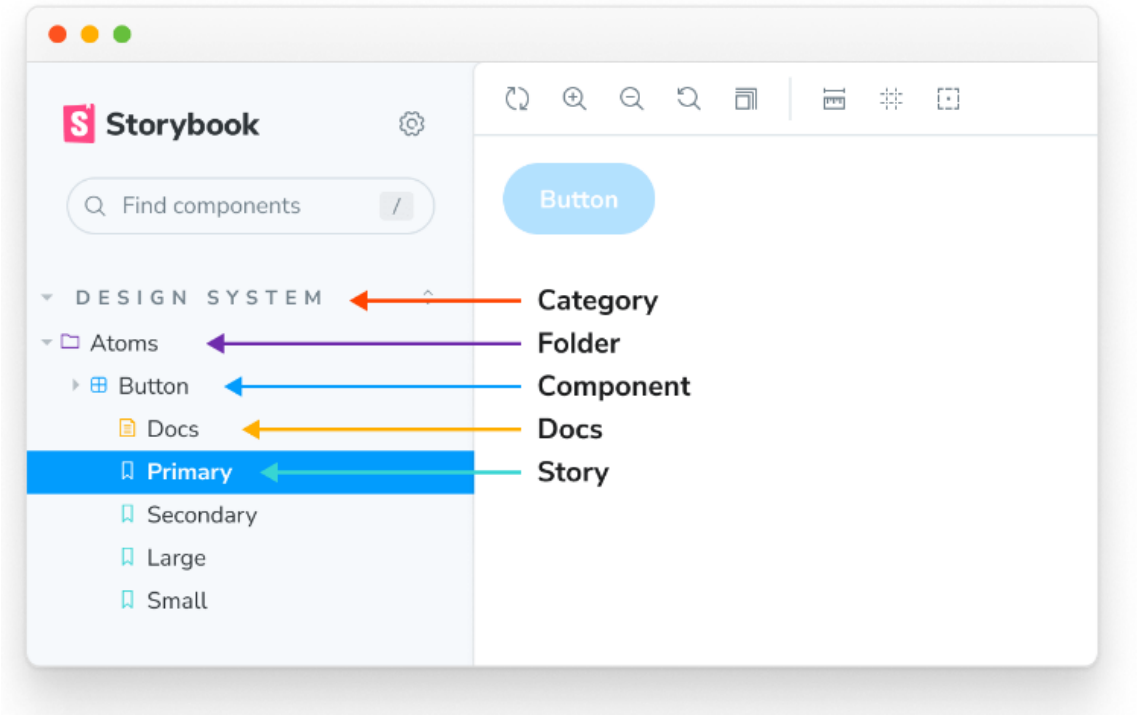
Toisena esimerkkinä on sovelluksen ulkoasua parantava Material UI -kirjasto (viitataan jatkossa nimell MUI), johon löytyy asennuskomento Storybookin laajennussivustolta sekä vaiheittainen dokumentointi sen käyttöönottoon. Se tarjoaa tyyliteltyjä elementtejä kuten painikkeita ja kytkimiä sekä efektejä kuten *ripple* (suom. aaltoilu) ja *skeleton* (tunnetaan myös nimellä *Facebook Shimmer*), joiden tarkoituksena on helpottaa käyttäjäystävällisen käyttöliittymän rakentamista.

Kuva 9. Ripple-efekti muodostaa lyhytkestoisen aaltoefektin alkaen kursorin kohdasta elementin sisällä. Kursorin painalluskohta merkattu ruksilla kuvassa.

MUI käyttää animaatoihin Emotion-tyylitysmoottoria (engl. styling engine), mikä asennetaan elementtien ohella kirjaston dokumentoimalla komennolla *npm install @mui/material @emotion/react @emotion/styled.* (Mui). Emotion-moottorin tarkoituksena on mahdollistaa samanlaisen syntaksin käyttö tyylien luomiseen eri ohjelmointikielillä sekä kehyksillä, samalla helpottaen visuaalisesti yhtenäisen käyttöliittymän tekemistä sen uudelleenkäytettävillä tyyleillä (Emotion).

3.4 Komponenttitarina

Storybookissa jokaiselle komponentille luodaan nimipohjainen tarina kuvastamaan sen käyttötarkoitusta, mikä nopeuttaa sen toiminnallisuuden ymmärtämistä etenkin kehittäjillä, joille kyseinen komponentti ei ole ennestään tuttu. Selkeä komponentin dokumentointi vähentää riskiä, että ohjelmistokoodikantaan luotaisiin lähes tai täysin identtinen komponentti sen seurauksena, ettei olemassa olevaa komponenttia löydetty loogisilla hakutermeillä.

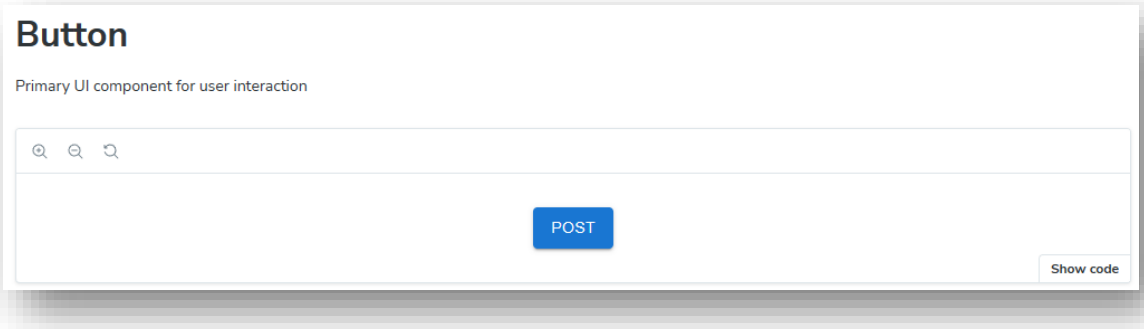


Kuva 10. Yksinkertaistettu esimerkki painikkeen eri tarinoista: ensisijainen, toissijainen, iso sekä pieni (Storybook c).

Storybookin tiedostohierarkia rakentuu kirjaston automaattisesti luodusta ”suunnittelujärjestelmä” (engl. design system) kategoriasta, jonka alle voidaan lisätä kansioita esimerkiksi atomimallisuunnittelulla (engl. atomic design). Edellä mainittu suunnittelumalli perustuu käyttöliittymän jakamisen viiteen eri kategoriaan: atomit, molekyylit, organismit, mallipohjat (engl. templates) sekä sivut (Guzman, 2019).

Mikäli koetaan, ettei pelkkä komponentin nimi riitä avaamaan sen käyttötarkoitusta riittävästi, voidaan hyödyntää Storybookin AutoDocs-ominaisuutta, mikä luo Storybookiin dokumentoinnin

komponentille sen ohjelmistokoodin pohjalta, sisältäen sen ominaisuudet, funktiot sekä muille kehittäjille suunnatut kommentit.



Kuva 11. Komponenttitason dokumentointi automaattisella AutoDocs-automaattidokumentoinnilla (Storybook g).

Komponentin otsikko saadaan sen nimestä, joka on annettu käytettäväksi muulle ohjelmistokoodikokonaisuudelle export-avainsanalla (esim. Button), aliotsikon tullessa sen yläpuolella sijaitsevasta kommenttiosiosta (engl. comment-block).

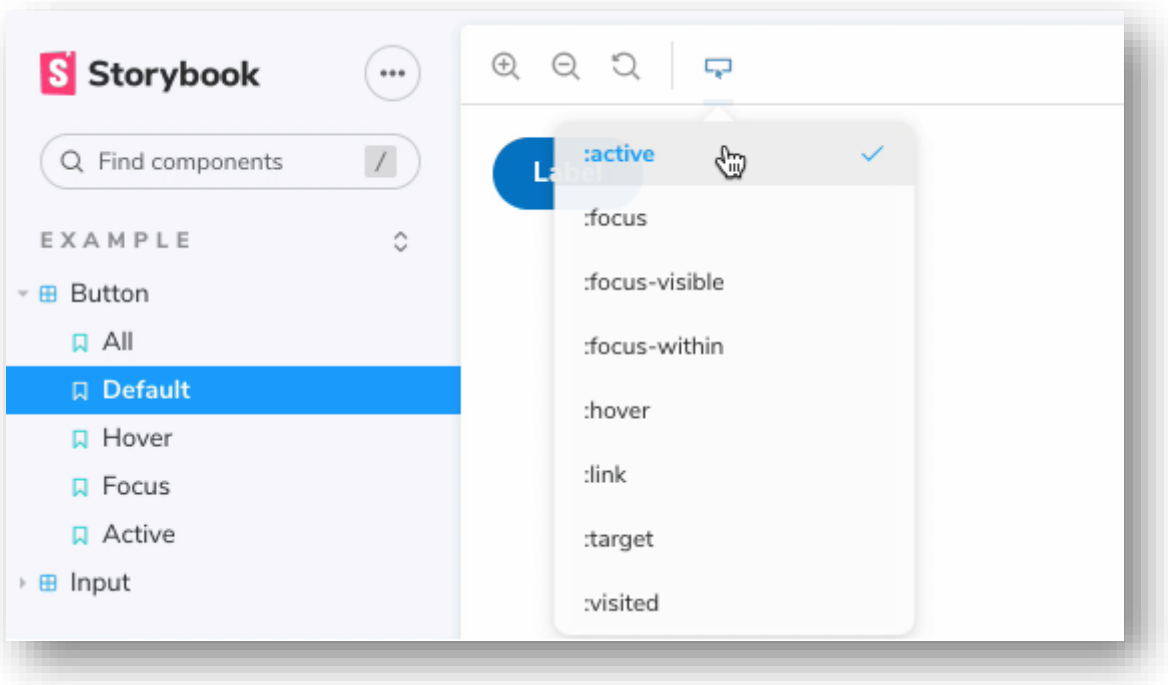
```tsx
import React from 'react';
import MuiButton from '@mui/material/Button';
import './button.css';

interface ButtonProps {
  /**
   * Is this the principal call to action on the page?
   */
  primary?: boolean;
  /**
   * What background color to use
   */
  backgroundColor?: string;
  /**
   * How large should the button be?
   */
  size?: 'small' | 'medium' | 'large';
  /**
   * https://mui.com/material-ui/react-button/#colors
   */
  color?: 'inherit' | 'primary' | 'secondary' | 'success' | 'error' | 'info' | 'warning';
  /**
   * Button contents
   */
  label: string;
  /**
   * https://mui.com/material-ui/api/button/#Button-prop-variant
   */
  variant?: "contained" | "outlined" | "text";
  /**
   * Optional click handler
   */
  onClick?: () => void;
}

/**
 * Primary UI component for user interaction
 */
export const Button = ({
  primary = false,
  size = 'medium',
  backgroundColor,
  label,
  variant="contained",
  color="primary",
  ...props
}: ButtonProps) => {
  return (
    <MuiButton
      size={size}
      color={color}
      style={{ backgroundColor }}
      variant={variant}
      {...props}
    >
      {label}
    </MuiButton>
  );
};
```

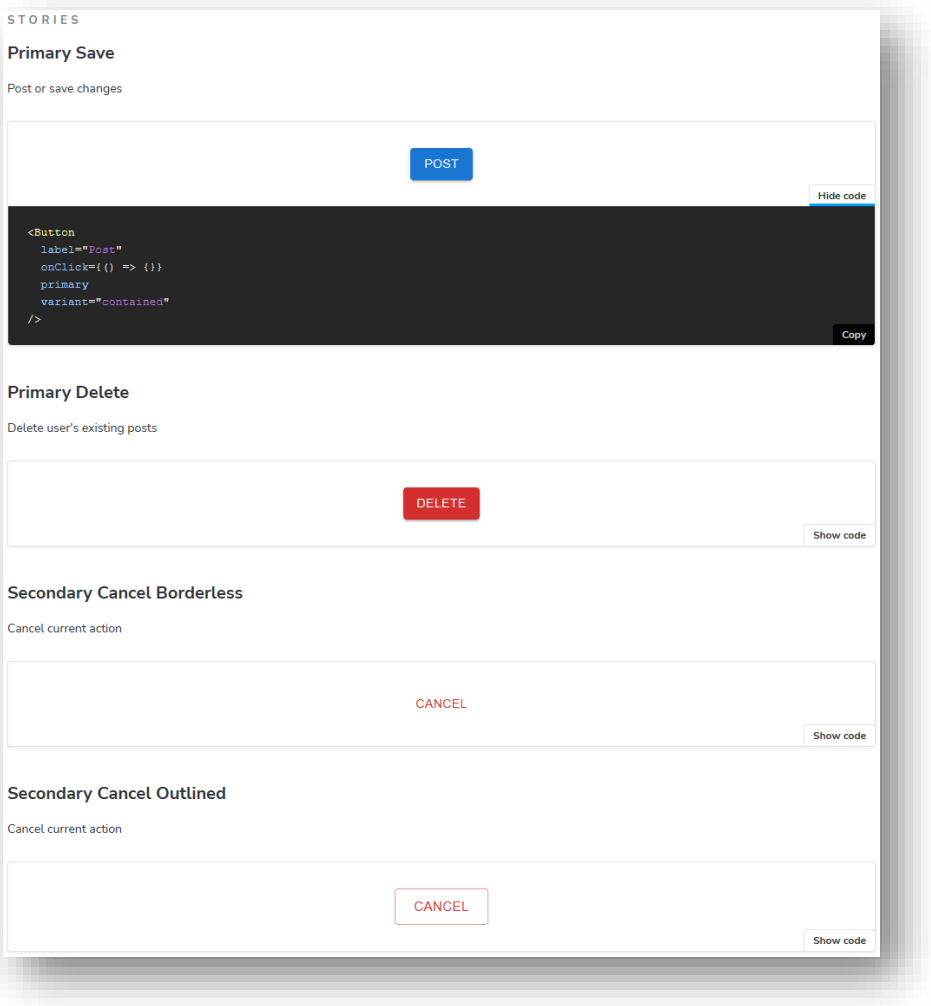
Kuva 12. Kustomoitu painikekomponentti käyttäen MUI-kirjasto painike-elementtiä.

Tekstidokumentoinnin alta löytyy itse komponentti, joka on piirrettynä omassa ruudussaan ja jonka kanssa voidaan komponentin tyypin mukaan olla vuorovaikutuksessa. Vuorovaikutteisten komponenttien testaamiseen Storybookilta löytyy lisäosia, kuten esimerkiksi alla olevassa kuvakaappauksessa näkyvä Pseudo States -lisäosa (Storybook j). Sen avulla voidaan aktivoida yksi tai useampi pseudo-tila komponentissa, ja simuloida muun muassa hiiren kursorin siirtymistä elementin päälle tai sen painamista kursorilla. Pseudo-tilojen tai -luokkien tarkoituksena on mahdollistaa interaktiivisten sekä käyttäjäystävällisten käyttöliittymien luominen, antaen visuaalista palautetta käyttäjälle heidän suorittamastaan toiminnosta.



Kuva 13. Pseudo States -lisäosan tarjoama dropdown-valikko, millä voidaan simuloida pseudo-tiloja komponentissa (Storybook b).

Pseudo-tilojen lisäksi komponenttien ulkoasua voidaan muokata niiden argumenttien kautta, esimerkkeinä MUI-kirjastolla luodulla painikkeella voidaan tavallisen painikkeen koon, etiketin (engl. label), tausta- sekä tekstivärin lisäksi muokata myös varianttia, millä korostetaan painikkeen toiminnallisuutta käyttöliittymässä.

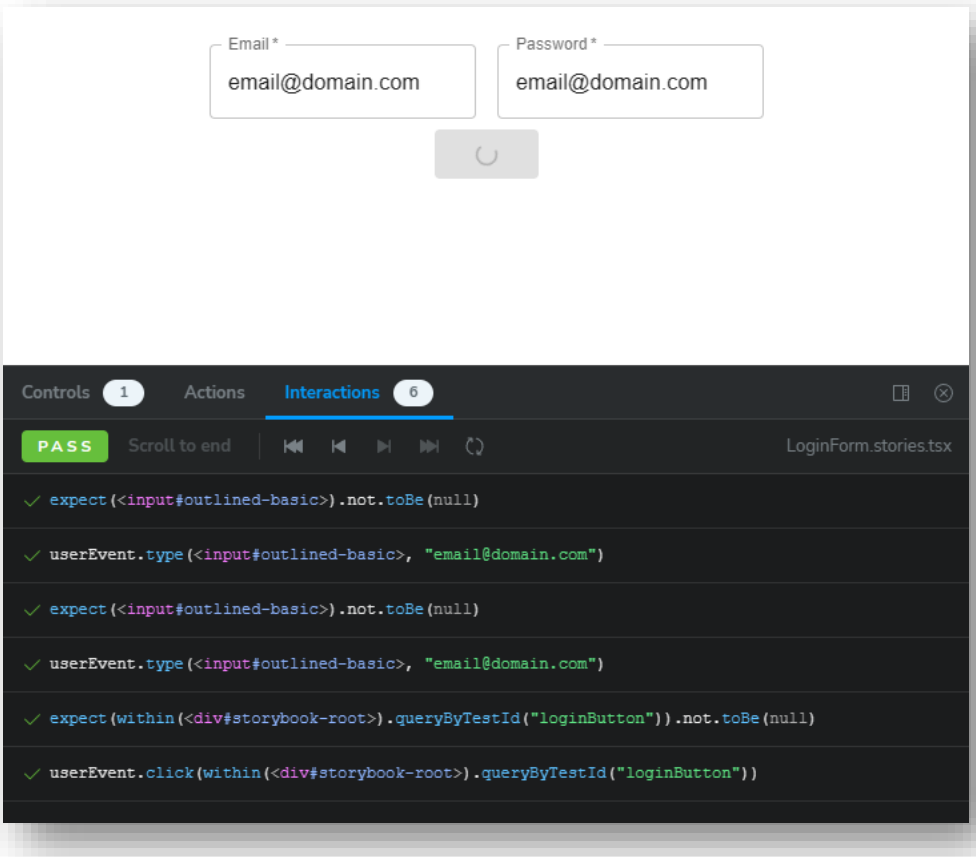


Kuva 14. Storybookin ohjelmistokoodissa painikkeelle luotu dokumentointi visualisoituna.

Jokainen variantti voidaan esitellä allekkain omana erillisenä tarinana, mikä sisältää halutessa tarkemman tekstipohjaisen kuvailun sekä helposti yhdellä painalluksella kopioitavan ohjelmistokoodipätkän sen implementointia varten ohjelmistoprojektiin.

3.5 Käyttöliittymän testiautomaatio

Jatkuvan integraation (engl. CI, Continuous Integration) ja jatkuvan toimituksen (engl. CD, Continuous Delivery) yhteydessä käyttöliittymätestit eli UI-testit sopeutuvat saumattomasti putkeen, automatisoiden testausprosessin ja varmistaen, että uusi ohjelmistokoodi ei häiritse olemassa olevaa toiminnallisuutta. Käyttöliittymän automaatiotesti eroaa manuaalisesta testaamisesta siten, että automatisointi on simuloitu ohjelmistokoodilla, kun taas manuaalinen testaaminen suoritetaan käyttäjän toimesta. Käyttöliittymän testaus voidaan automatisoida Storybookissa asentamalla sitä tukevia kirjastoja, kuten testing-library, addon-interactions sekä Jest, joista kaksi ensimmäistä ovat Storybookin omia kirjastoja, kun taas jälkimmäinen on Facebookin React-kehykselle kehittämä testauskirjasto. Edellä mainittujen kirjastojen käyttöönottoon löytyy vaiheittainen dokumentointi Storybookin verkkosivuilta, mitkä neuvovat niin asennuksessa kuin määrittelyssä (Storybook i).



Kuva 15. Automatisoitu kirjautumistesti.

Käyttöliittymän automaatiotestaus Storybookilla on verrattavissa muihin JavaScript-testikehyksiin sekä -kirjastoihin, syntaksin pysyessä lähes samanlaisena, mikä laskee kynnystä olemassa olevien testien siirtämiselle sekä uusien luomiselle Storybookiin. Lisänä Storybookissa on testien vaiheittainen läpikäyminen, mikä mahdollistaa vaiheiden välillä siirtymisen käyttöliittymän mediaohjauspainikkeiden avulla, helpottaen mahdollisten ongelmien havaitsemista sekä täten nopeuttaen niiden korjaamista.

```
export const FilledForm: Story = {
    play: async ({ canvasElement }) => {
        const canvas = within(canvasElement);

        const emailInputElement: HTMLElement | null = canvas.getByTestId('email').querySelector("input")
        expect(emailInputElement).not.toBe(null)
        if (emailInputElement) await userEvent.type(emailInputElement, "email@domain.com")

        const passwordInputElement: HTMLElement | null = canvas.getByTestId('password').querySelector("input")
        expect(passwordInputElement).not.toBe(null)
        if (passwordInputElement) await userEvent.type(passwordInputElement, "email@domain.com")

        const loginButtonElement: HTMLElement | null = canvas.queryByTestId('loginButton')
        expect(loginButtonElement).not.toBe(null)
        if (loginButtonElement) await userEvent.click(loginButtonElement)
    },
};
```

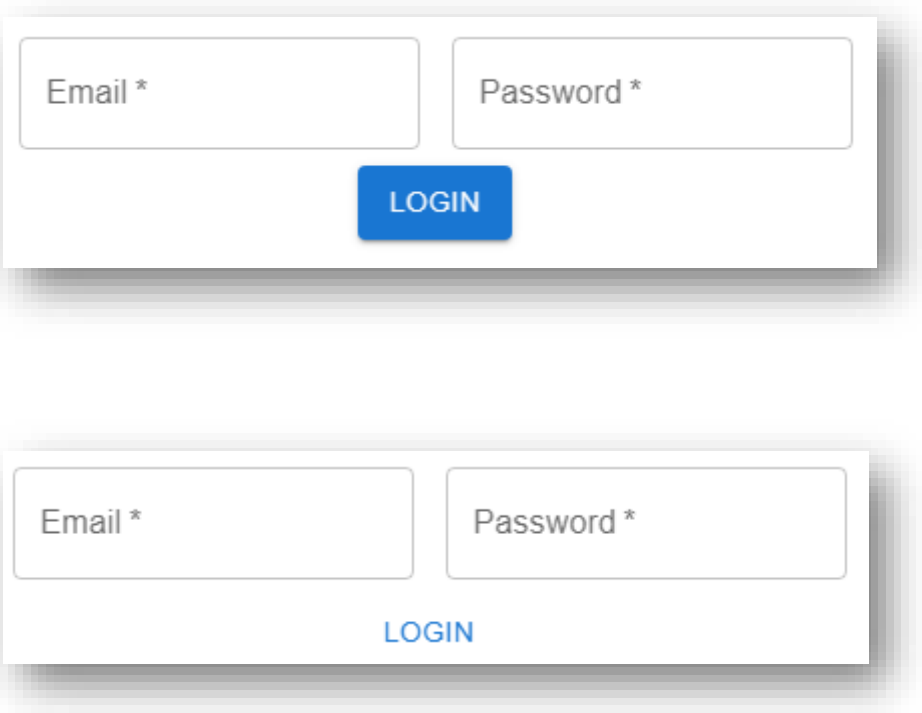
Kuva 16. Automatisoidun kirjautumistestin ohjelmistokoodi käyttäen Storybookin testing-library-kirjastoa sekä Jestin expect-rajapintaa.

Halutessa voidaan myös yhdistää tarinakomponentteja sekä ajaa automatisoidut käyttöliittymätestit niitä vasten yhden testin sisällä, mistä voi olla hyötyä etenkin, kun testataan käyttäjäpolkua (engl. user flow) ja halutaan varmistaa tietyn käyttäjätoimenpiteen (engl. user action) vievän käyttäjän oikeaan komponenttiin käyttöliittymässä.

3.6 Visuaalinen regressiotestaus

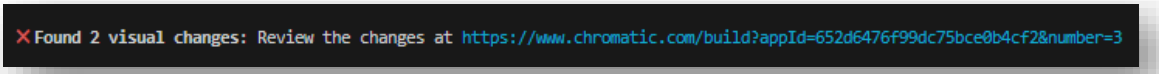
Visuaalinen testaaminen Storybookissa tapahtuu kyseisen kirjaston luoneen kehitystiimin Chromatic-pilvipalvelulla, millä verrataan kuvakaappauksia keskenään (Chromatic a). Chromatic otetaan käyttöön rekisteröitymällä palveluun joko sähköpostilla ja salasanalla tai eri versionhallinta-alustojen kirjautumistunnuksilla, joista tunnetuimpia lienevät GitHub sekä GitLab. Asennus tapahtuu edellisten kappaleiden tapaan NPM-ohjelmistopaketinhallinnalla käyttäen komentoa *npm*

install –save-dev chromatic, minkä lisäksi Chromatic vaatii tietoturvallisuuteen liittyvän merkin (engl. token) käyttöä ensimmäisellä käyttökerralla. Kun merkki lähetetään pilvipalvelimelle käyttäen komentoa *npx chromatic --project-token=xxxxx* ja paikalliset Storybook tarinamuutokset on tallennettu versionhallintaan, tarinat lähetetään synkronoimista varten pilvipalvelimelle.
Opinnäytetyössä testattiin Chromaticin visuaalisten muutosten havaitsemista, eli visuaalista testaamista, tehdään komponentille ulkoasumuutos, kuten alla olevassa kuvakaappauksessa on esitetty, ja synkronoidaan paikalliset Storybook muutokset uudelleen pilvipalvelimelle.



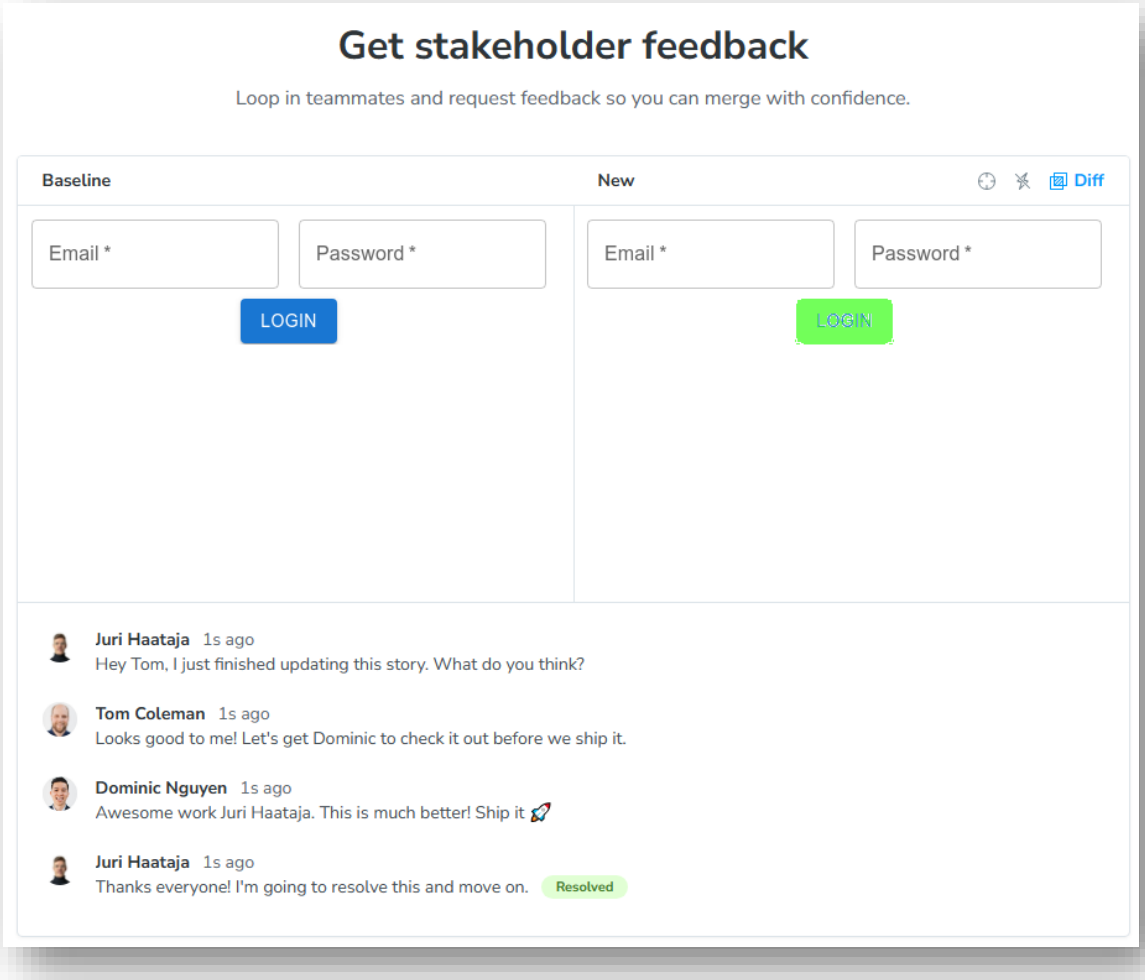
Kuva 17. Kuvakaappaukset kirjautumislomakkeesta ennen ja jälkeen muutoksen.

Chromatic tallentaa kuvakaappaukset komponenteista ja niiden eri tiloista, muodostaen niistä ”perustan” (engl. baseline) eli referenssinäytteen, jonka avulla tulevia muutoksia verrataan. Kun uudet ohjelmistokoodimuutokset ladataan Chromaticiin komentorivikomennolla, komponenteista otetaan uudet kuvakaappaukset, joita palvelu sitten vertaa aiemmin tallennettuun perustaan käyttäen kuvantunnistusta sekä algoritmejä.



Kuva 18. Komentorivin ilmoitus, kun kirjautumislomakkeen painikkeen ulkoasua muokattiin ja verrattiin aiemmin Chromaticiin synkronoitua vasten.

Chromatic palauttaa komennon vastauksena visuaalisen testauksen tuloksen sekä linkin, kuten yllä olevassa kuvassa, jossa on havaittu visuaalinen muutos kirjautumispainikkeessa. Vastauksessa oleva linkki avaa kyseessä olevan käyttöliittymäkomponentin katselmoinnin selaimeen alla olevan kuvakaappauksen tapaan, jossa kehittäjät voivat keskustella sekä hyväksyä tai hylätä muutokset, riippuen halutusta lopputuloksesta.



Kuva 19. Chromatic havaitsee eroavaisuuden kirjautumispainikkeissa, kun alkuperäisen sininen täyteväri otettiin pois käytöstä.

Lyhyesti tiivistettynä Chromaticia käytetään visuaalisena vastineena pelkälle ohjelmistokoodin versionhallille, jolloin muut kehittäjät voivat seurata yksittäisten komponenttien kehitystä niin ohjelmistokoodi- kuin visuaalisella tasolla.

4 POHDINTA

Storybook osoittautui tehokkaaksi työkaluksi käyttöliittymäkomponenttien ulkoasun ja visuaalisen ilmeen eheyden varmistamisessa ohjelmistokoodimuutosten aikana, sillä sen avulla voidaan eristää yksittäiset komponentit sekä tarkastella niiden visuaalisia muutoksia helposti ja tarkasti.

Käyttöliittymäkomponenttien dokumentointi Storybookissa tarjoaa kattavan kuvauksen kunkin komponentin odotetusta ulkoasusta ja toiminnasta, mikä on ensiarvoisen tärkeää visuaalisen regressiotestauksen onnistumisen kannalta, sillä sen avulla voidaan nopeasti vertailla muutoksia dokumentoituihin tiloihin ja varmistaa, että mikään ei ole mennyt huomaamatta ohi.
Storybookin snapshot-toiminnallisuus mahdollistaa visuaalisten tilanteiden tallentamisen ja vertaamisen, mikä antaa mahdollisuuden havaita jopa pienetkin visuaaliset muutokset komponenttien ulkoasussa ja reagoida niihin nopeasti. Automatisoidut visuaalisen regressiotestauksen työkalut, kuten Chromatic, integroituvat saumattomasti Storybookiin tarjoten tehokkaan tavan tunnistaa ja seurata visuaalisia poikkeamia.

Pelkkää ilotulitusta Storybookin käyttö ei kuitenkaan ollut, etenkään sen käyttöönotto. Jokaista komponenttia varten täytyi luoda oma, pelkästään Storybookia varten tarkoitettu tiedosto, jonka kautta itse komponentti sekä sille luodut tarinat ja erilaiset muokkausvaihtoehdot saatiin selainnäkymään. Myös joidenkin ohjelmistokirjastojen kanssa oli ongelmia, kuten aiemmin tässä opinnäytetyössä mainitun aalto-efektin, minkä korjaaminen olisi vaatinut syvällisempää tietämystä JavaScript-kehyksistä. Tästä syystä päädyin asentamaan koko MUI-kirjastokokoelman, jonka mukana kyseinen efekti sekä muutama muukin itselleni vähemmän tarpeellinen kirjasto tulivat.
Kaiken kaikkiaan Storybook osoittautui olennaiseksi työkaluksi visuaalisen eheyden ylläpitämiseksi ohjelmistokoodimuutosten välillä, ja sen avulla voidaan varmistaa, että käyttöliittymä pysyy visuaalisesti johdonmukaisena jokaisessa kehitysvaiheessa.

LÄHTEET

Abramowski, Nicole 2023. How to use React Context API and avoid prop drilling. Scrimba. Hakupäivä 23.9.2023. https://scrimba.com/articles/react-context-api/.

AWS. What is An IDE (Integrated Development Environment)? Hakupäivä 16.9.2023. https://aws.amazon.com/what-is/ide/.

Beletsky, Alexander 2018. How to Use the Power of Jest's Snapshot Testing Without Using Jest. Medium. Hakupäivä 11.9.2023. https://medium.com/blogfoster-engineering/how-to-use-the-power-of-jests-snapshot-testing-without-using-jest-eff3239154e5.

Buna, Samer 2017. React Interview Question: What gets rendered in the browser, a component or an element? FreeCodeCamp. Hakupäivä 16.9.2023. https://www.freecodecamp.org/news/react-interview-question-what-gets-rendered-in-the-browser-a-component-or-an-element-1b3eac777c85/.

Chromatic a. Introduction to Chromatic. Hakupäivä 16.10.2023. https://www.chromatic.com/docs/.

Chromatic b. We help developers build Uis with components. Hakupäivä 13.11.2023. https://www.chromatic.com/company/about.

ComponentDriven. Component Driven User Interfaces. Hakupäivä 14.9.2023. https://www.componentdriven.org/.

Copes, Flavio 2018. NPM GLOBAL OR LOCAL PACKAGES. Hakupäivä 10.10.2023. https://flaviocopes.com/npm-packages-local-global/.

Dsoudza-Sania 2023. Test for visual regression with Jest-image-snapshot. Dev. Hakupäivä 11.9.2023. https://dev.to/saniadsouza/test-for-visual-regression-with-jest-image-snapshot-4i54.

Emotion. Introduction. Hakupäivä 11.10.2023. https://emotion.sh/docs/introduction.

Figma. Hakupäivä 11.9.2023. https://www.figma.com/.

Guzman, Kris 2019. Atomic Design for Developers: Project Structure. Medium. Hakupäivä 28.9.2023. https://betterprogramming.pub/atomic-design-for-developers-part-1-b41e547a555c.

Ivanovs, Alex 2023. The Most Popular Front-end Frameworks in 2023. StackDiary. Hakupäivä 14.9.2023. https://stackdiary.com/front-end-frameworks/.

Lieschke, Simon 2021. Making a Progressive Web App. Create React App. Hakupäivä 24.9.2023. https://create-react-app.dev/docs/making-a-progressive-web-app/.

Makwana, Brijen 2022. Difference between Hot Reloading and Live Reloading in React Native. GeeksForGeeks. Hakupäivä 22.9.2023. https://www.geeksforgeeks.org/difference-between-hot-reloading-and-live-reloading-in-react-native/.

Moore, Joseph 2023. What Is The Best Anti Aliasing Mode?. DisplayNinja. Hakupäivä 21.9.2023. https://www.displayninja.com/best-anti-aliasing-mode/.

Mui. Material UI - Overview . Hakupäivä 24.9.2023. https://mui.com/material-ui/getting-started/.

NPM. About npm. Hakupäivä 10.10.2023. https://docs.npmjs.com/about-npm.

Padalkar, Rahul 2023. Setting up visual regression testing with React Native Owl. LogRocket. Hakupäivä 17.9.2.2023. https://blog.logrocket.com/visual-regression-testing-with-react-native-owl/.

Parker, Chris 2020. How to Pass Data between React Components. PlurarSight. Hakupäivä 14.9.2023. https://www.pluralsight.com/guides/how-to-pass-data-between-react-components.

Rehkopf, Max. User stories with examples and a template. Atlassian. Hakupäivä 16.9.2023. https://www.atlassian.com/agile/project-management/user-stories.

Stevens, Emily 2023. The 10 best user interface (UI) design tools to try in 2023. UX Design Institute. Hakupäivä 14.11.2023. https://www.uxdesigninstitute.com/blog/user-interface-ui-design-tools/.

Storybook a. Hakupäivä 13.11.2023. https://storybook.js.org/.

Storybook b. Pseudo States. Hakupäivä 12.10.2023. https://storybook.js.org/addons/@hover/storybook-addon-pseudo-states/.

Storybook c. Naming components and hierarchy. Hakupäivä 16.9.2023. https://storybook.js.org/docs/react/writing-stories/naming-components-and-hierarchy.

Storybook d. Install addons. Hakupäivä 10.10.2023. https://storybook.js.org/docs/react/addons/install-addons/

Storybook e. Configure Storybook. Hakupäivä 24.9.2023. https://storybook.js.org/docs/react/configure/overview.

Storybook f. Install Storybook. Hakupäivä 24.9.2023. https://storybook.js.org/docs/react/get-started/install/.

Storybook g. How to document components. Hakupäivä 28.9.2023. https://storybook.js.org/docs/react/writing-docs/introduction.

Storybook h. Install addons. Hakupäivä 10.10.2023. https://storybook.js.org/docs/react/addons/install-addons/.

Storybook i. Interaction tests. Hakupäivä 15.10.2023. https://storybook.js.org/docs/react/writing-tests/interaction-testing.

Storybook j. Pseudo States. Hakupäivä 12.10.2023. https://storybook.js.org/addons/@hover/storybook-addon-pseudo-states/.

Unadkat, Jash 2023. Top JavaScript Testing Frameworks. BrowserStack. Hakupäivä 11.9.2023. https://www.browserstack.com/guide/top-javascript-testing-frameworks.

Xu, David 2022. Everything you need to know about Visual Regression Testing in 2022. Medium. Hakupäivä 17.9.2023. https://david-x.medium.com/the-state-of-visual-regression-testing-in-2022-5de10ffe8f6f.