LAPIN AMK
Lapland University of Applied Sciences

STORYBOOK NYKYAIKAISEN OHJELMISTOKEHITYKSEN TYÖKALUNA

Nuotio Mira

Opinnäytetyö

Tietojenkäsittelyn koulutus
Tradenomi (AMK)

2025

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Tietojenkäsittelyn koulutus
Tradenomi (AMK)

Tekijä	Mira Nuotio	**Vuosi**	2025
Ohjaaja(t)	Johanna Vuokila		
Toimeksiantaja	Mindhive Oy		
Työn nimi	Storybook nykyaikaisen ohjelmistokehityksen työkaluna		
Sivumäärä	39 + 2		

Tämän opinnäytetyön aiheena oli rakentaa Mindhive Oy:n MAINIO-tuotteen käyttöliittymäkehitystä tukeva Storybook-pohjainen komponenttikirjasto ja yhtenäinen design system. Työn tavoitteena oli parantaa tuotteen visuaalista ja teknistä yhtenäisyyttä sekä luoda pohja jatkokehitykselle ja ylläpidolle. Tutkimuskysymykset kohdistuivat siihen, miten Storybook voidaan liittää osaksi modernia monorepo-ympäristöä ja millä tavoin sen avulla voidaan tukea käyttöliittymäkomponenttien suunnittelua, dokumentointia ja testausta.

Tietoperusta käsitteli design systemien ja komponenttipohjaisen kehityksen keskeisiä käsitteitä sekä Storybookin tarjoamia toiminnallisuuksia. Tutkimusmenetelminä käytettiin toiminnallista kehittämistyötä ja laadullista lähestymistapaa. Aineisto hankittiin kirjallisista lähteistä , dokumentaatioiden analysoinnin ja kehitysprosessin aikana tehtyjen havaintojen avulla. Lisäksi toimeksiantajan kanssa käydyt keskustelut ja palautteet ohjasivat ratkaisujen valintoja ja työn etenemistä.

Työn tuloksena rakennettiin Storybook-ympäristö, johon luotiin käyttöliittymäkomponenttien kirjasto. Dokumentointia ja testausta tuettiin valituilla lisäosilla. Työn tuloksena syntynyt design system toimii MAINIO-tuotteen kehityksen apuna ja tukee jatkossa kehittäjien yhteistyötä, dokumentaation selkeyttä ja saavutettavuuden huomioimista.

Avainsanat	design system, Storybook, käyttöliittymäkehitys, komponentti
Muita tietoja	Työhön liittyi toimeksiantajalle toimitettu käyttöliittymäkomponenttien kirjasto

LAPIN AMK
Lapland University of Applied Sciences

Abstract of the thesis

Business Information Technology
Bachelor of Business Administration

Author	Mira Nuotio	**Year**	2025
Supervisor(s)	Johanna Vuokila		
Commissioned by	Mindhive Oy		
Title	Storybook, tool for modern software development		
Number of pages	39 + 2		

The topic of this thesis deals with creation of a Storybook-based component library and a unified design system to support the user interface development of Mindhive Oy's MAINIO product. The objective of the thesis was to improve the product's visual and technical consistency, as well as to establish a foundation for further development and maintenance. The research questions focused on how Storybook can be integrated into a modern monorepo environment and how it can be utilized to support the design, documentation, and testing of user interface components.

The theoretical framework addressed the key concepts of design systems and component-based development, as well as the functionalities offered by Storybook. The research methods applied were functional development work and a qualitative approach. The data were collected from literary sources, analysis of documentation, and observations made during the development process. In addition, discussions with and feedback from the commissioning company guided the design decisions and the progress of the work.

As a result of the thesis, a Storybook environment was created, including a component library for user interface development. Documentation and testing were supported with selected addons. The resulting design system serves as a tool for the further development of the MAINIO product and will continue to support developer collaboration, clarity of documentation, and consideration of accessibility.

Keywords	design system, Storybook, user interface development, component
Additional information	The thesis included the delivery of a user interface component library to the commissioning company

SISÄLLYS

1 JOHDANTO

Tämä opinnäytetyö on kehittämispainotteinen toiminnallinen opinnäytetyö, joka toteutetaan toimeksiantajayritys Mindhive Oy:lle. Toiminnallisessa opinnäytetyössä tuotetaan konkreettinen ratkaisu (Vilkka 2021), joka tässä opinnäytetyössä on rakennettu Storybook-työkalu. Työn tavoitteena on rakentaa MAINIO-tuotteen käyttöliittymäkehitystä tukeva Storybook-pohjainen design system, joka tarjoaa selkeän rakenteen komponenttien suunnittelulle, dokumentoinnille ja käytölle.

Opinnäytetyön tutkimusmenetelmänä hyödynnetään kehittämistyöhön soveltuvaa laadullista lähestymistapaa. Laadullinen tutkimus soveltuu tähän opinnäytetyöhön, sillä tavoitteena on ymmärtää ja parantaa kehittäjäkokemusta kerätyn aineiston perusteella. (Vilkka 2021.)

Työn teoriaosuuden tietolähteinä toimivat aiheeseen liittyvät kirjalliset lähteet, aiheeseen liittyvien ohjeistusten ja dokumentaation analysointi sekä työelämälähtöistä havainnointi kehittämisprosessin aikana. Alan kirjalliset lähteet soveltuvat tämän opinnäytetyön teorian lähteeksi, sillä sen avulla voidaan esittää aikaisempaa tutkimusta valitusta aiheesta, ja näin teoriaosuus saa perustellun pohjan.

Lisäksi toimeksiantajan kanssa käydyt keskustelut ja palautteet ohjaavat kehittämistyön suuntaa ja tukevat työn arviointia käytännön näkökulmasta. Työn empiirisessä osuudessa dokumentoidaan suunnittelun, toteutuksen ja ratkaisujen eri vaiheet osana kokonaisuutta.

1.1 Tutkimuskysymykset

Päätutkimuskysymyksenä tällä opinnäytetyöllä on selvittää, miten Storybook-työkalua voidaan hyödyntää selkeän design systemin rakentamisessa. Tavoitteena on tarjota toimeksiantajalle selkeä ja dokumentoitu rakenne komponenttien hallintaan, joka tukee käyttöliittymän yhtenäistä kehitystä ja ylläpitoa. Kysymyksellä pyritään vastaamaan siihen, millaisia konkreettisia hyötyjä Storybook tuo kehittäjäkokemukseen, esimerkiksi käyttöliittymäkomponenttien testattavuuden, visuaalisen esittämisen ja dokumentoinnin osalta. Alatutkimuskysymyksinä toimivat seuraavat kysymykset:

• Mitä hyötyä Storybookista on käyttöliittymäkehityksessä?

• Miten Design System tukee käyttöliittymäkehityksen yhtenäisyyttä?

Alatutkimuskysymysten avulla jäsennellään päätutkimuskysymystä tarkemmin. Näiden kysymysten kautta analysoidaan sekä käytännön toteutuksen että teorian näkökulmasta, miten design system voidaan rakentaa Storybook-ympäristössä. Lisäksi pyritään tunnistamaan parhaat käytännöt, joita toimeksiantajayritys voi hyödyntää jatkossa.

Toimeksiannon lähtökohtana on Mindhive Oy:n tarve kehittää yhtenäinen Storybook-pohjainen komponenttikirjasto MAINIO-tuotteelle. Komponenttikirjaston avulla yrityksen MAINIO dashboard -käyttöliittymä noudattaa yhtenäistä design systemiä. Komponentilla tarkoitetaan tässä opinnäytetyössä esimerkiksi käyttöliittymän painiketta, syötekenttää tai muuta rajattua toiminnallista osaa, jota voidaan hyödyntää toistuvasti sovelluksen eri näkymissä (Heineman & Councill, 2001). MAINIO dashboard on yrityksille suunnattu hallintapaneeli, joka tarjoaa näkymän tekoälyagenttien käyttöön ja hallintaan. Tekoälyagentilla tarkoitetaan ohjelmistoa, joka toimii itsenäisesti. Se pystyy havainnoimaan ympäristöä, tekemään päätöksiä ja toteuttamaan tehtäviä tavoitteiden saavuttamiseksi. Varhaisimmat agentit perustuivat ennalta määrättyihin sääntöihin, mutta nykyiset järjestelmät hyödyntävät oppimista ja pystyvät yhdistämään esimerkiksi päättelyn, suunnittelun ja vuorovaikutuksen. (Qu ym. 2025.)

Tavoitteena on parantaa kehittäjäkokemusta, nopeuttaa käyttöliittymäkehitystä ja helpottaa ylläpitoa. Opinnäytetyössä suunnitellaan ja toteutetaan design systemin perusta ja määritellään tekninen viitekehys Storybook-ympäristölle. Näihin nojaten toteutetaan käyttöliittymäkomponenttien kirjasto, joka dokumentoidaan esimerkeillä, saavutettavuustiedoilla sekä käyttövinkeillä. Lisäksi tässä opinnäytetyössä määritellään komponenttien ja tyylien hierarkia sekä luodaan ratkaisu design tokenien hallintaan. Työ tukee MAINIO-tuotteen kehittämistä ja toimii pohjana sen visuaaliselle ja tekniselle yhtenäisyydelle. Työ toteutetaan React-ympäristössä ja käyttöön otetaan Storybook-työkalu, jonka avulla komponentteja voidaan kehittää, testata ja dokumentoida irrallaan varsinaisesta sovelluksesta.

Työn taustalla on ajatus komponenttilähtöisestä ohjelmistokehityksestä, jossa käyttöliittymä rakennetaan modulaarisesti uudelleen käytettävistä osista. Tämä lähestymistapa mahdollistaa järjestelmällisen suunnittelun ja tarjoaa kehittäjille paremman näkyvyyden olemassa oleviin komponentteihin ja niiden käyttötapoihin. Opinnäytetyö kytkeytyy suoraan käytännön kehitystyöhön ja komponenttikirjasto otetaan käyttöön osaksi yrityksen ohjelmistokehitysprosessia.

1.2 Mindhive Oy

Toimeksiantajayrityksenä toimii mikkeliläinen Mindhive Oy, joka tarjoaa asiakkailleen digitaalisia ratkaisuja liiketoiminnan kehittämiseen. Yritys on erikoistunut pilvipohjaisiin mobiili- ja web-sovelluksiin, jotka suunnitellaan yksilöllisesti kunkin asiakkaan tarpeisiin. Mindhiven toiminta kattaa koko asiakkaan digitalisaatioprosessin aina liiketoiminnan uudelleenajattelusta ja palvelumuotoilusta itse tekniseen toteutukseen saakka. Yrityksen erityisosaamista ovat tekoälyn hyödyntäminen ja dataohjautuvat ratkaisut, joilla pyritään tehostamaan asiakkaiden toimintaa ja päätöksentekoa. (Mindhive 2025.)

Asiakasprojektien valmistumisen jälkeen Mindhive toimii myös pilvipalvelukumppanina, vastaten sovellusten ylläpidosta ja jatkokehityksestä. Opinnäytetyön toimeksianto liittyy Mindhiven kehittämään MAINIO-nimiseen tekoälyassistenttiin, jota yritykset voivat hyödyntää osana omaa toimintaansa. MAINIOn käyttöä varten on kehitetty hallintapaneeli eli dashboard, jonka kautta käyttäjät voivat hallita ja seurata tekoälyagenttien toimintaa. (Mindhive 2025.) Kehitystyön tavoitteena on luoda tälle käyttöliittymälle selkeä sekä Storybook-pohjainen komponenttikirjasto, jonka avulla parannetaan kehittäjäkokemusta ja tuetaan yhtenäistä käyttöliittymäsuunnittelua.

2 KOMPONENTTILÄHTÖINEN KEHITYS

Ohjelmistokehityksessä on etsitty tapoja rakentaa sovelluksia niin, että ne olisivat paitsi laadukkaita myös helpommin ylläpidettäviä ja laajennettavia. Yksi merkittävimmistä lähestymistavoista tähän on ollut komponenttilähtöinen kehitys (Component-Based Development, CBD). Sen keskeinen ajatus on yksinkertainen: miksi kirjoittaa sama asia alusta aina uudelleen, jos sen voi rakentaa kerran ja käyttää uudelleen? (Heineman & Councill 2021.)

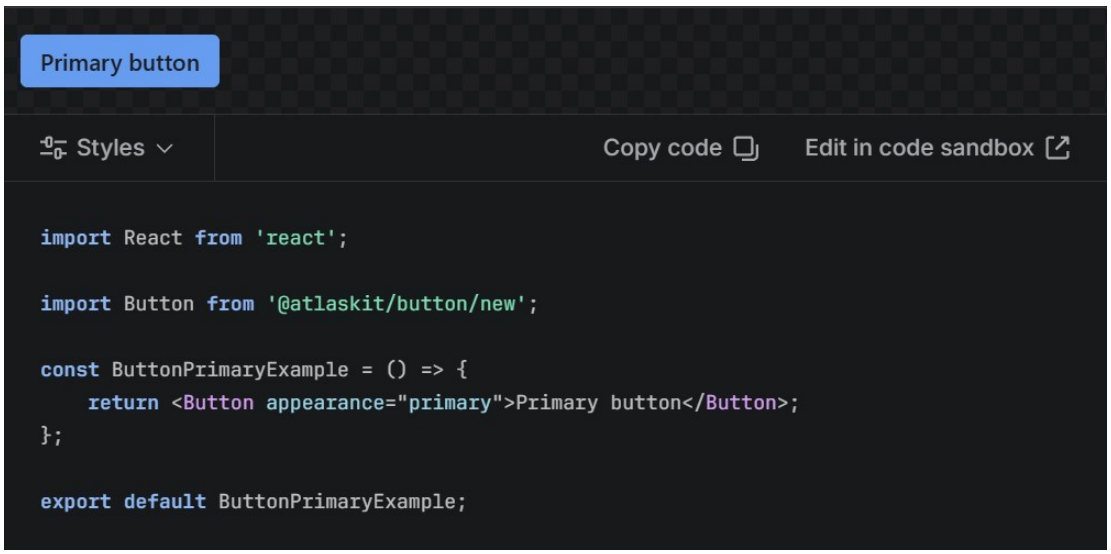
Komponenttilähtöinen kehitys pohjautuu uudelleenkäytettävyyteen. Ohjelmisto ei synny yhtenä suurena kokonaisuutena, vaan se jaetaan pienempiin osiin, komponentteihin, joilla on selkeä rajapinta ja määritelty tehtävä. Näitä osia voidaan yhdistellä, vaihtaa tai käyttää uudelleen eri projekteissa. (Heineman & Councill 2001.)

Komponenttilähtöisen kehityksen periaatteita tukee myös atomic design tapa rakentaa käyttöliittymä pienistä uudelleen käytettävistä osista suuremmiksi kokonaisuuksiksi (Frost 2016). Tässä opinnäytetyössä atomic design -ajattelu valittiin ohjaavaksi viitekehykseksi, sillä se tarjoaa työtä tukevan mallin design systemin rakentamiselle.

2.1 Komponentti ohjelmistokehityksessä

Komponentti voidaan ymmärtää ohjelmiston rakennuspalikkana. Se on itsenäinen kokonaisuus, joka suorittaa tietyn tehtävän ja tarjoaa sen muille ohjelmiston osille selkeän rajapinnan kautta. Esimerkiksi kirjautumisnäkymä, maksutoiminnallisuus tai graafinen käyttöliittymäelementti, kuten esimerkiksi painike, voi olla komponentti. Komponentin vahvuus on siinä, että se voidaan erottaa muusta ohjelmasta ja liittää uusiin järjestelmiin ilman, että sitä tarvitsee kirjoittaa kokonaan uudestaan. (Heineman & Councill 2001.)

Kuvio 1 havainnollistaa käyttöliittymäkomponentin toteutuksen. Kyseessä on painike-komponentti, joka määritellään omana erillisenä kokonaisuutena ja voidaan uudelleen käyttää eri yhteyksissä.



Kuvake 1. Painike-komponentin esimerkkitoteutus uudelleen käytettävästä käyttöliittymäelementistä.

2.2 Komponenttilähtöisen kehityksen hyödyt ja haasteet

Komponenttilähtöisen kehityksen vahvuudet liittyvät erityisesti ohjelmistojen ylläpidettävyyteen ja tehokkuuteen. Koska komponentit voidaan ottaa käyttöön useammassa sovelluksessa, kehitysaika lyhenee ja kustannuksia säästyy. Samalla sovellusten luotettavuus kasvaa, kun käytetään kertaalleen testattuja ja toimiviksi todettuja osia. Lisäksi komponenttien eriyttäminen tukee rinnakkaista kehitystä, kun eri tiimit voivat työstää eri osia samaan aikaan ilman, että ne häiritsevät toistensa työtä. (Heineman & Councill 2001.) Hyödyt eivät kuitenkaan tule ilman ongelmia. Yksi suurimmista haasteista on integraatio: miten erilaiset komponentit saadaan toimimaan yhteen? Jokainen komponentti on ehkä laadittu hieman eri tavalla, eri kielillä tai eri ympäristöissä. Tämä luo riippuvuuksia ja vaatii huolellista suunnittelua. Lisäksi komponenttien laatu voi vaihdella, etenkin jos osa niistä on hankittu ulkopuolisilta toimittajilta. (Heineman & Councill 2001.)

Kun ohjelmistot alkoivat rakentua yhä enemmän erillisistä, toistuvasti käytettävistä osista, syntyi tarve hallita tätä kokonaisuutta paremmin. Komponenttilähtöinen ajattelu onkin luonut perustan myös käyttöliittymäkehityksen design systemeille: aivan kuten sovellukset koostuvat teknisistä komponenteista, myös käyttöliittymät voidaan rakentaa selkeästi määritellyistä ja uudelleenkäytettävistä käyttöliittymäpalikoista. Näin kehitystyö pysyy yhtenäisenä ja hallittuna, vaikka mukana olisi suuri joukko kehittäjiä. (Heineman & Councill 2001.)

2.3 Atomic Design

Atomic design on metodologia, jonka tarkoituksena on luoda käyttöliittymiä järjestelmällisesti pienistä osista kokonaisuuksiksi. Sen lähtökohtana on, että käyttöliittymät eivät muodostu yksittäisistä sivuista, vaan ne rakentuvat pienistä, uudelleenkäytettävistä komponenteista, joita yhdistämällä saavutetaan yhtenäinen ja skaalautuva järjestelmä. Ajatus perustuu analogiaan kemiasta, jossa atomit yhdistyvät molekyyleiksi ja edelleen organismeiksi muodostaen suurempia kokonaisuuksia. (Frost 2016.)

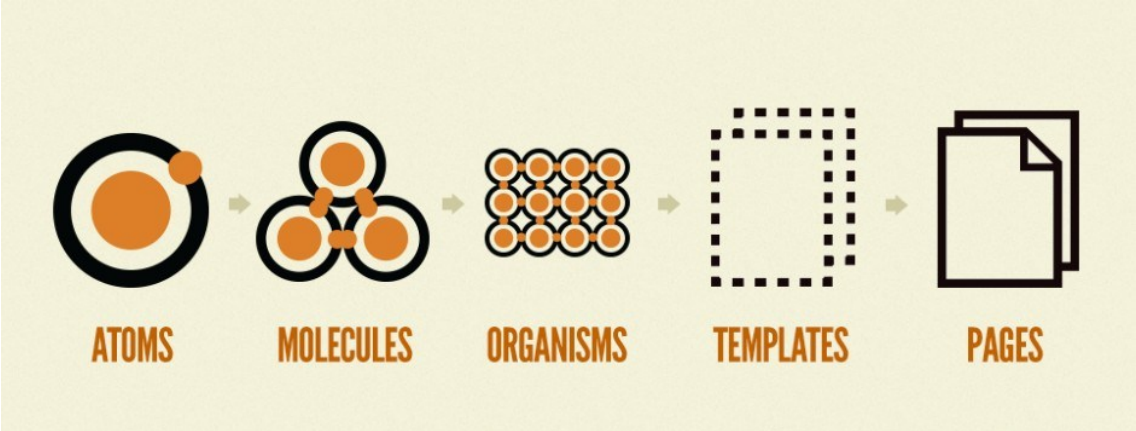
Ensimmäisenä tasona atomic designissa ovat atomit, jotka ovat käyttöliittymän pienimpiä elementtejä. Atomeja ovat esimerkiksi painikkeet, syötekentät, typografiset elementit ja värimuuttujat. Atomit ovat usein abstrakteja, eikä niistä ole yksinään käytännön hyötyä, mutta ne toimivat perustana, josta muut tasot rakentuvat. (Frost 2016.)

Seuraava taso muodostuu molekyyleistä, joissa useampi atomi yhdistyy pieneksi kokonaisuudeksi. Molekyyleillä on jo selkeä käyttötarkoitus, ja ne toteuttavat yksinkertaisen toiminnon. Esimerkkinä voidaan mainita hakukenttä, joka muodostuu esimerkiksi syötekentästä ja painikkeesta. Molekyylit toimivat design systemin perustuksena ja tukevat uudelleenkäytettävyyttä. (Frost 2016.)

Organismit rakentuvat useista molekyyleistä ja muodostavat käyttöliittymän selkeämpiä ja itsenäisiä osia. Esimerkkinä tästä on sivun navigaatiopalkki, joka koostuu esimeriksi logosta, hakutoiminnosta ja valikosta. Organismien avulla käyttöliittymä alkaa hahmottua konkreettisesti, ja niiden avulla voidaan kuvata käyttöliittymän suurempia kokonaisuuksia. (Frost 2016.)

Organismeista siirrytään mallitasolle. Mallit kokoavat eri osat yhteen rakenteelliseksi kokonaisuudeksi, joka toimii sivupohjana. Mallien avulla voidaan havainnollistaa käyttöliittymän rakennetta ja asettelua, vaikka sisältö olisi vielä paikkamerkkien muodossa. Näin ne muodostavat yhteyden yksittäisten komponenttien ja valmiiden sivujen välille. (Frost 2016.)

Viimeisenä tasona ovat sivut, joissa mallien rakenne täydennetään todellisella sisällöllä. Sivut esittävät käyttöliittymän lopullisessa muodossaan ja toimivat tärkeänä vaiheena käytettävyyden ja kokonaisuuden arvioinnissa. Niiden avulla testataan design systemin toimivuutta todellisessa kontekstissa ja varmistetaan, että komponentit tukevat käyttötilanteita tarkoituksenmukaisesti (Frost 2016). Kuviossa 2 esitetään atomic design tasot.



Kuvio 2. Atomic design (Frost 2016)

3 DESIGN SYSTEM

Design systemille ei ole yksiselitteistä ja yleisesti hyväksyttyä määritelmää, ja käsitettä käytetään eri tavoin eri yhteyksissä. Tässä opinnäytetyössä Design systemillä tarkoitetaan kokonaisuutta, joka koostuu toisiinsa liittyvistä käyttöliittymän toistuvista ja uudelleen käytettävistä elementeistä ja yhteisistä toimintatavoista. (Kholmatova 2017.)

Design system kokoaa yhteen käyttöliittymän visuaaliset ja toiminnalliset osat sekä niitä ohjaavat periaatteet yhtenäiseksi kokonaisuudeksi. Kokonaisuuden tavoitteena on varmistaa käyttöliittymien johdonmukaisuutta sekä tehostaa kehitystyötä. Design system voi sisältää muun muassa komponenttikirjaston, design tokenit, typografia- ja värimääritykset sekä saavutettavuusohjeistukset (IxDF 2025). Design token on nimetty arvo, jolla hallitaan käyttöliittymän visuaalista tyyliä, esimerkiksi linkin fontin väritystä. Saavutettavuusohjeistukset varmistavat, että käyttöliittymä täyttää esteettömyyden vaatimukset (WCAG 2018).

Ajatus ei ole täysin uusi. Jo vuonna 1968 Douglas McIlroy esitti ensimmäisen kerran idean ohjelmistojen kokoamisesta valmiiden komponenttien avulla. Hän toi esille, että ohjelmistokehitystä tulisi lähestyä kuten teollista tuotantoa: rakentamalla vakioituja osia, joita voidaan yhdistellä eri tarkoituksiin. McIlroy oli mukana kehittämässä myös Unix-työkaluja, jotka ilmensivät tätä periaatetta. (Naur & Randell 1969.)

Varsinaisen modernin komponenttikäsitteen määritteli Brad Cox 1980-luvulla. Hän kehitti yhdessä Tom Loven kanssa Objective-C-ohjelmointikielen, jonka taustalla oli ajatus ohjelmistojen rakentamisesta uudelleenkäytettävistä osista. Myöhemmin suuret yritykset, kuten Apple, ottivat tämän käyttöönsä (Cox 1986).

Komponentti voidaan määritellä yksinkertaisimmillaan ohjelmiston tunnistettavaksi osaksi, jolla on oma tehtävänsä. Margaret Rouse kuvaa komponentin olevan yksikkö, joka tarjoaa tietyn toiminnon tai toimintojen joukon, ja jota voidaan hyödyntää muiden komponenttien kanssa (Bigelow 2024). Käytännössä tämä voi tarkoittaa esimerkiksi käyttöliittymän painiketta, lomakekenttää tai toisaalta myös rajapinnan määrittelyä. Tässä opinnäytetyössä komponentteja tarkastellaan nimenomaan käyttöliittymän osina.

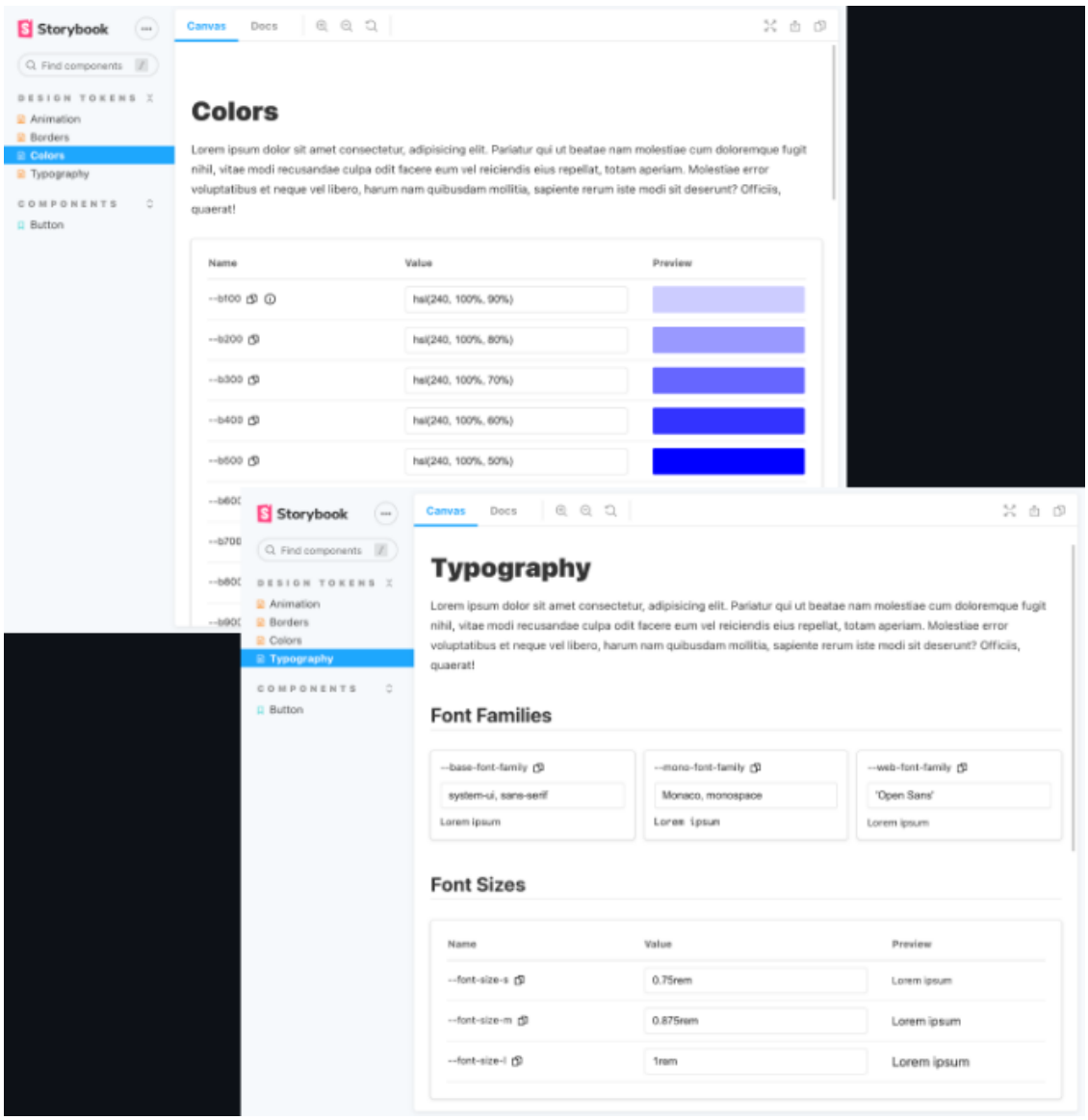
Selainpohjaisessa kehityksessä komponentteihin liittyvä ajattelu alkoi yleistyä jo ennen 2010-lukua modularisoinnin kautta. Modularisoinnilla tarkoitetaan ohjelmiston jakamista pienempiin kokonaisuuksiin, joita voidaan muokata, ylläpitää ja hyödyntää uudelleen eri yhteyksissä. Esimerkkejä tästä ovat esimerkiksi tyylikirjastojen grid-järjestelmät, joissa käyttöliittymän tila jaetaan eri elementeille. (De La Cuarda 2016.)

Kaksi keskeistä käsitettä komponenttilähtöisessä kehityksessä ovat koheesio ja kytkentä. Koheesiolla tarkoitetaan sitä, kuinka hyvin komponentti keskittyy yhteen toiminnallisuuteen. Korkea koheesio tekee komponenteista ymmärrettäviä ja uudelleenkäytettäviä. Kytkentä puolestaan kuvaa riippuvuuksia eri komponenttien välillä: mitä löyhempi kytkentä, sitä helpommin komponentteja voidaan muokata ja käyttää eri yhteyksissä. (GeeksforGeeks 2019.)

3.1 Design tokens

Design tokenit ovat käyttöliittymän visuaalisia ja toiminallisia ominaisuuksia kuvaavia, uudelleen käytettäviä ja nimettyjä määrityksiä, jotka muodostavat yhdenmukaisen lähteen käyttöliittymän tyyliratkaisuille. Tyypillisiä design tokeneita ovat esimerkiksi värit ja fontit. (Google 2025.)

Design tokenit toimivat välineenä käyttöliittymän visuaalisten ominaisuuksien, kuten värien ja typografian, yhdenmukaistamiseen ja hahmottamiselle. Kun jokainen väri- ja fonttivariaatio määritellään omaksi design tokeniksi, saadaan selkeä kokonaiskuva siitä, millaisia erilaisia elementtejä käyttöliittymässä on käytössä. Visuaalisessa muodossa esitettyjen design tokenien tarkastelu voi paljastaa huomattavan määrän päällekkäisiä tai tarpeettomia vaihtoehtoja, mikä luo mahdollisuuden yhtenäistämiseen ja käytettävien elementtien määrän rajaamiseen ja yhdenmukaistamiseen. Koska design tokenit muodostavat single truth lähteen, voidaan muutokset tehdä keskitetysti ja ne päivittyvät koko järjestelmään. Tämä nopeuttaa kehitystyötä ja varmistaa brändi-ilmeen johdonmukaisuuden eri tuotteissa ja alustoilla. (Frost 2016.) Design token esimerkki esitelty kuviossa 3.



Kuvio 3. Design tokens (GitHub 2021)

3.2 Komponenttikirjasto

Komponenttikirjasto on kokoelma käyttöliittymän valmiita ja toistuvasti käytettäviä elementtejä, kuten painikkeita, valikkoja, lomakekenttiä tai kuvakkeita. Sen on tarkoitus toimia kehittäjille ja suunnittelijoille keskitetty lähde, kirjasto, josta löytyvät kaikki tarvittavat komponentit selkeine kuvauksineen ja käyttöohjeineen. Yhdestä paikasta saatavilla olevat ja tarkasti määritellyt komponentit nopeuttavat kehitystyötä ja vähentävät virheitä, ja näin lisää tehokkuutta toteutustyöhön. Yhtenäinen komponenttikirjasto myös ylläpitää yhtenäistä visuaalista ilmettä eri tuotteissa ja alustoissa. (UPXin 2023.)

Jokainen komponentti kuvataan kirjastossa erikseen. Tyypillisesti komponenttikirjastossa jokaisesta elementin kuvauksessa on sen nimi, käyttötarkoitus, mahdolliset muokattavat ominaisuudet, kuten väri, koko ja muoto, sekä eri käyttötilat ja niiden erilaiset visuaaliset variaatiot. Usein mukana on myös valmiita koodiesimerkkejä, jotka helpottavat komponentin integrointia erilaisiin käyttöliittymä- ja ohjelmistokehysratkaisuihin (Kawamoto 2024).

Kuvio 4. Komponenttikirjaston hakemisto, joka noudattaa atomic design -periaatetta

3.3 Kehityskäytännöt

Yhtenäinen design system voi merkittävästi parantaa kehittäjien työn tehokkuutta ja laatua. Visuaalisten ja teknisten ratkaisujen ollessa koottuna yhteen paikkaan, kehittäjät voivat hyödyntää valmiita komponentteja ja ohjeistuksia ilman tarvetta toistaa samoja, mahdollisesti työläitä, ratkaisuja jokaisessa projektissa. Tämä vähentää turhaa variaatiota, nopeuttaa kehitystyötä ja vähentää virheitä, myös uusien tiimin jäsenten perehdytys nopeutuu. Keskeisessä roolissa ovat komponenttien selkeä ja kattava dokumentaatio. Lisäksi huolellinen suunnittelu ehkäisee ristiriitoja olemassa olevien järjestelmien kanssa ja tukee skaalautuvuutta, mikä on tärkeää laajoissa tai useita tiimejä kattavissa projekteissa. (Maza 2020.)

Design system toimii työkaluna ohjelmistokehityksen laadun ja tehokkuuden parantamisessa. Se tarjoaa yhtenäisen viitekehyksen, jonka avulla tiimit voivat hyödyntää valmiiksi määriteltyjä ja testattuja käyttöliittymäkomponentteja ilman tarvetta rakentaa samoja ratkaisuja toistuvasti. Tämä ei ainoastaan nopeuta kehitysprosessia, vaan myös vähentää virheiden riskiä ja helpottaa järjestelmän ylläpitoa. Kun komponentit ovat ennustettavia ja dokumentaatio kattavaa, uusiin projekteihin perehtyminen nopeutuu ja yhteistyö suunnittelijoiden ja kehittäjien välillä sujuu ilman jatkuvaa tulkintaa. Lisäksi teknisen velan hallinta helpottuu, koska muutokset ja korjaukset voidaan toteuttaa keskitetysti ja päivittää kaikkiin komponentin käyttöpaikkoihin samanaikaisesti. (Kostiuchenko 2025.)

4 STORYBOOK

Storybookin ensimmäinen versio julkaistiin huhtikuussa 2016, kun sen kehitti sri lankalainen startup-yritys Kadira. Syyskuussa 2016 julkaistu versio 2.0 toi mukanaan laajennettavuuden ja maksullisen palvelun. Kadiran toiminnan päätyttyä saman joulukuussa projektin kehitys pysähtyi. Keväällä 2017 yhteisön kehittäjät ottivat projektin haltuunsa, ja toukokuussa 2017 julkaistiin Storybook 3.0, joka oli ensimmäinen täysin yhteisön ylläpitämä versio. (Shilman 2017.)

Storybook on avoimen lähdekoodin kehitystyökalu, jota käytetään käyttöliittymäkomponenttien ja käyttöliittymänäkymien rakentamiseen eristettynä muusta sovelluksesta. Sen avulla voidaan kehittää ja testata käyttöliittymän kaikkia eri tiloja ilman, että koko varsinaista sovellusta tarvitsee käynnistää. Tämä helpottaa komponenttien uudelleenkäyttöä, ylläpitoa sekä dokumentointia. (Storybook 2025c.)

Kehittäjä voi tallentaa yksittäiset komponenttien variaatiot storyiksi, jotka kuvaavat komponenttien ulkoasua ja toiminnallisuutta tietyssä tilassa. Storyt muodostavat interaktiivisen hakemiston, josta kehittäjä löytää olemassa olevat valmiit komponentit ja niiden eri versiot. Tarvittaessa komponenttien yhteyteen voidaan liittää kehittäjälle ohjeita, käyttötapauksia sekä esimerkkikoodeja. Storybook tukee myös lisäosia (addoneja), joiden avulla voidaan esimerkiksi testata saavutettavuutta, eri näyttökokoja tai suorittaa visuaalisia vertailutestejä. (Storybook 2025f.)

Storybookia käytetään laajasi sekä kehityksen että dokumentoinnin työkaluna, ja sen avulla voidaan julkaista komponenttikirjastoja tiimin sisäiseen ja myös julkiseen käyttöön. Storybook toimii useiden käyttöliittymien kehittämiseen tarkoitettujen ohjelmistokehysten, kuten Reactin, Vuen ja Angularin kanssa. (Storybook 2025c.) Tässä opinnäytetyössä työ on toteutettu React-kirjastoa käyttäen.

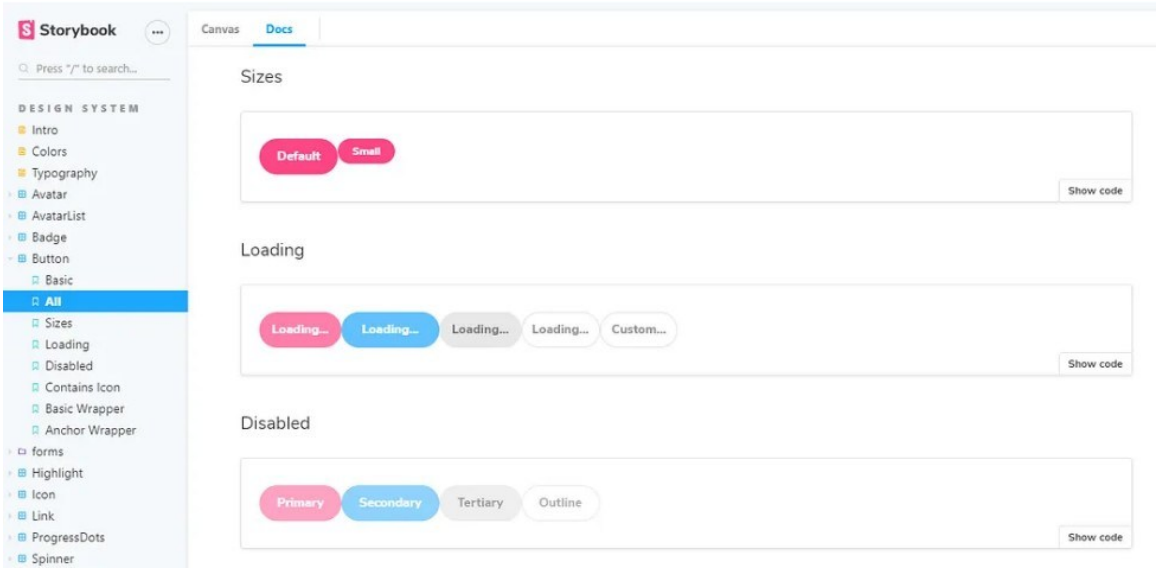
4.1 Toimintaperiaatteet

Storybookin idea perustuu siihen, että käyttöliittymäkomponentteja voidaan kehittää eristetyssä ympäristössä. Käyttöliittymäkomponentti tarkoittaa yksittäistä käyttöliittymän osaa, kuten esimerkiksi painiketta, valikkoa tai tekstikenttää, jota

voidaan käyttää uudelleen sovelluksen eri osissa. Eristetty ympäristöllä tarkoitetaan sitä, kun komponenttia kehitetään ja testataan irrallaan muusta varsinaisesta sovelluksesta. Tämä vähentää riippuvuuksia ja tekee mahdollisten virheiden tunnistamisesta helpompaa. (Storybook 2025b.)

Keskeisenä Storybook-periaatteena on tarinat, eli komponentin stories. Tarina kuvaa yksittäisen käyttöliittymäkomponentin tietyn tilan. Tilalla tarkoitetaan sitä, miten komponentti käyttäytyy tai miltä se näyttää tietyissä käyttötilanteissa. Esimerkiksi painikkeelle voidaan määrittää erilliset tarinat esimerkiksi kuvaamaan komponenttia sen pääasiallisessa tilassa, niin kutsutussa hover-tilassa, jossa hiiri viedään painikkeen päälle tai tilassa, jossa painike on poissa käytöstä. (Storybook 2025a.)

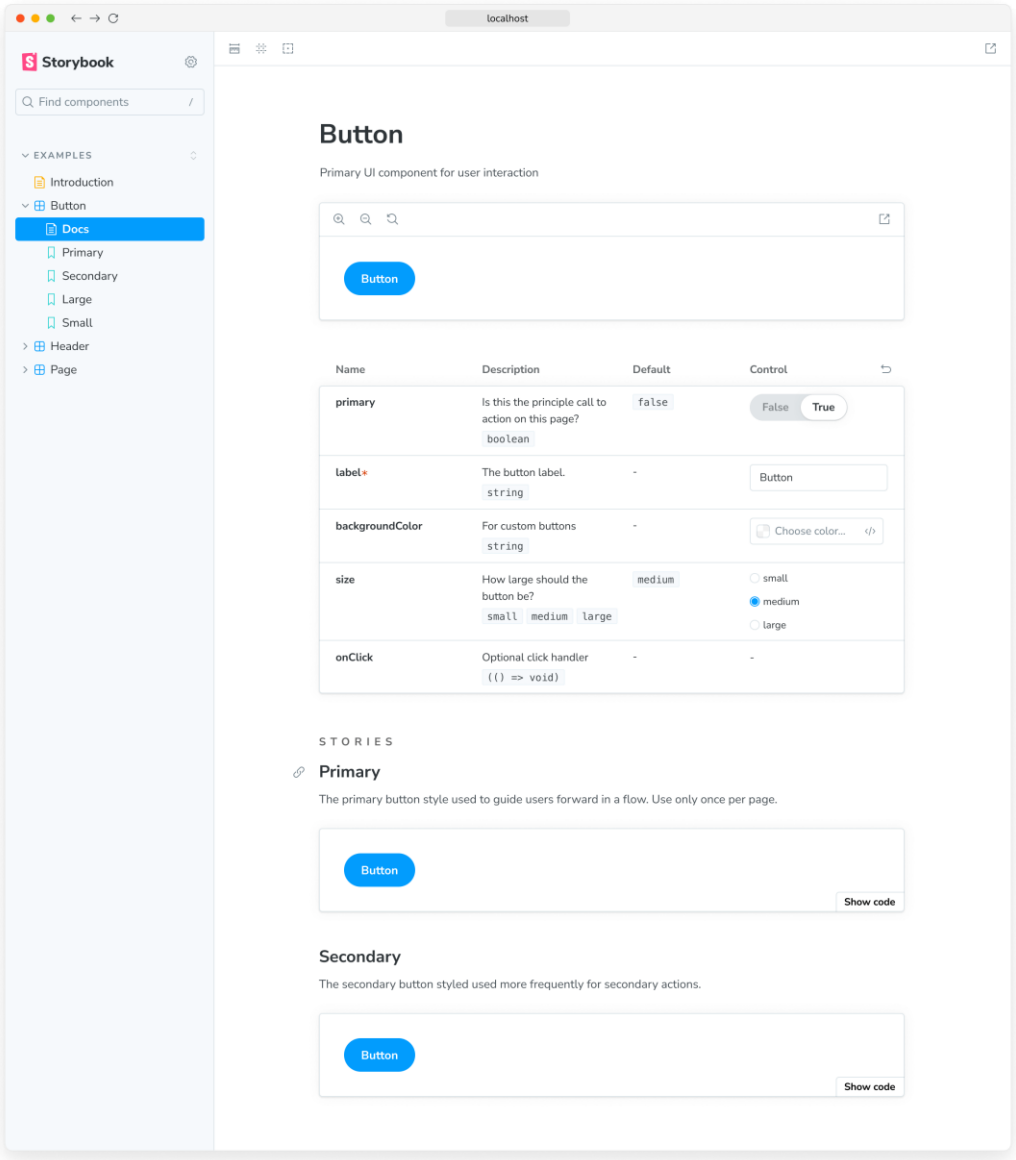
Storyjen avulla kehittäjä voi rakentaa interaktiivisen kirjaston, joka sisältää kaikki projektin käyttöliittymäkomponentit ja niiden eri tilat. Hakemisto toimii dokumentaationa, josta kehittäjätiimin jäsenet kuten esimerkiksi suunnittelijat ja testaajat voivat nähdä, mitä komponentteja on käytettävissä ja millaisia muunnelmia niistä on olemassa. Lisäksi storyihin voidaan liittää komponenteille ohjeita ja esimerkkikoodeja, mikä helpottaa komponenttien käyttöönottoa muissa projekteissa. Näin Storybook toimii sekä kehitystyökaluna että dokumentaation välineenä. (Storybook 2025c.)



Kuvio 5. Komponenttikirjastossa painikekomponentti sen eri tiloissa

4.2 Tarinat

Storybookin toiminnan keskiössä ovat tarinat. Tarina kuvaa käyttöliittymäkomponentin yhden tilan ja määrittelee, miltä komponentti näyttää sekä miten se käyttäytyy kyseisessä tilanteessa. Tilan käsite voi liittyä esimerkiksi komponentin eri käyttötilanteisiin, kuten painikkeen oletustilaan, tilaan, jossa hiiri on sen päällä, tai tilanteeseen, jossa painike ei ole käytettävissä. Jokainen tila dokumentoidaan erilliseksi tarinatiedostoksi, mikä mahdollistaa komponentin järjestelmällisen tarkastelun ja testaamisen. (Storybook 2025a.)



Kuvio 6. Painike-komponentin docs-näkymä, jossa esitetään komponentin dokumentaatio, kontrollipaneeli sekä tarinat

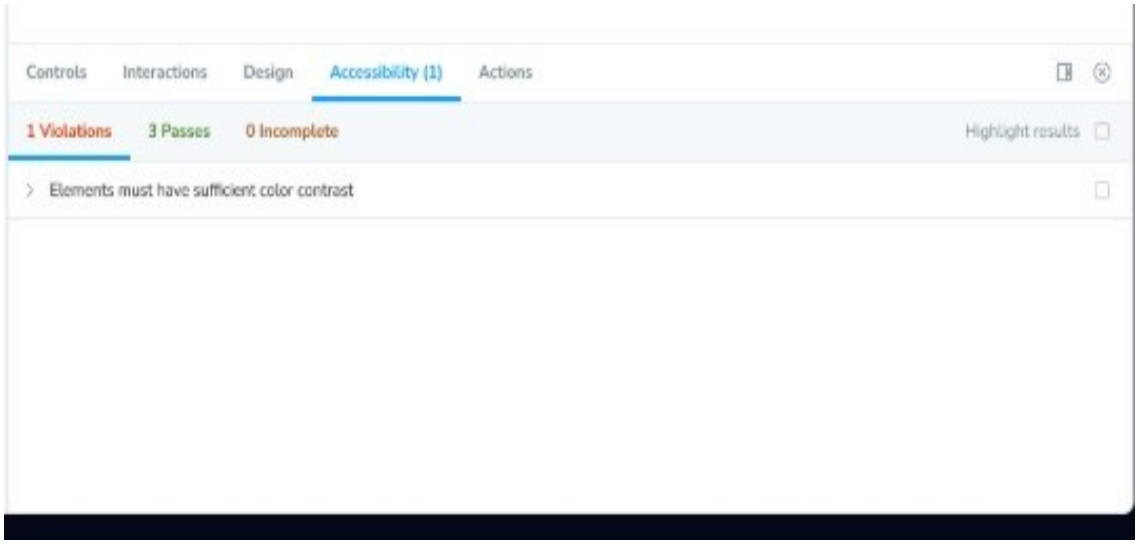
Tarinat muodostavat interaktiivisen kokoelman, josta kehittäjät, suunnittelijat ja testaajat voivat etsiä ja hyödyntää valmiita komponentteja. Tämä hakemisto toimii sekä dokumentaationa että kehityksen apuvälineenä. Tarinoihin voidaan liittää myös esimerkkikoodeja, ohjeita ja käyttötapauksia, mikä tukee komponenttien ymmärtämistä ja uudelleenkäyttöä. (Storybook 2025d.)

Tarinoiden määrittely tapahtuu ohjelmointikoodin avulla. Kehittäjä luo komponentille erillisen tarinatiedoston, jossa jokainen tila määritellään omana kokonaisuutenaan. Esimerkiksi painikkeelle voidaan määritellä erilliset tarinat perusversion, eri värivariaatioiden sekä käyttötilojen esittämiseksi. Näin yksittäisestä komponentista muodostuu dokumentoitu kokonaisuus, jota voidaan käyttää sekä kehitystyössä että testaamisessa. (Storybook 2025d.)

Tarinoiden käyttö tukee ohjelmistokehityksessä samoja periaatteita kuin komponenttilähtöinen kehitys laajemmin. Kun jokainen tila on dokumentoitu ja testattavissa erillään muusta sovelluksesta, voidaan vähentää riippuvuuksia ja parantaa ylläpidettävyyttä. Lisäksi se helpottaa yhteistyötä tiimin sisällä, sillä kaikki osapuolet voivat tarkastella komponentteja ilman, että koko sovellus täytyy käynnistää. (Storybook 2025a.)

4.3 Lisäosat

Storybookin perustoiminnallisuus on yksinkertainen, se tarjoaa ympäristön komponenttien kehittämiseen ja dokumentointiin. Tarpeen mukaan sitä voidaan laajentaa lisäosilla eli addoneilla. Lisäosat voivat olla joko Storybookin kehittäjien tai ulkopuolisten tekijöiden tuottamia, ja niiden avulla voidaan ottaa käyttöön uusia työtapoja ja ominaisuuksia. Esimerkkejä yleisesti käytetyistä lisäosista ovat esimerkiksi Actions, jolla voidaan seurata komponenttien tapahtumia, Controls, jolla voidaan muuttaa komponentin ominaisuuksia käyttöliittymässä, sekä Viewport, jonka avulla voidaan testata eri näyttökokoja. (Storybook 2025e.)



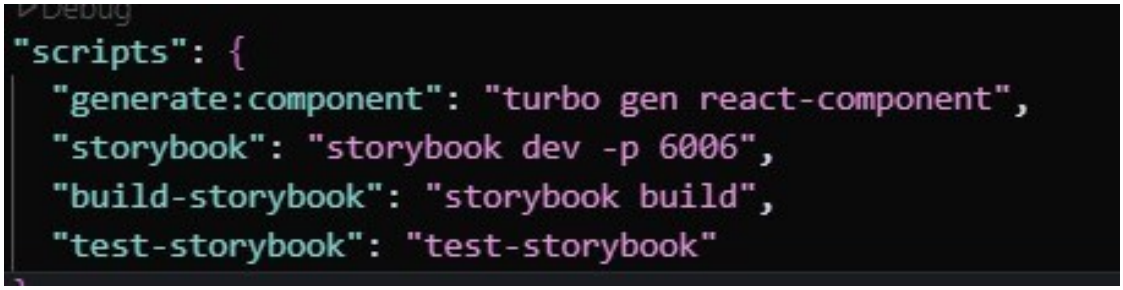
Kuvio 7. Storybook Accessibility-lisäosa tarkistaa komponentin saavutettavuutta, kuten kontrastivaatimusten täyttymistä

Lisäosat laajentavat Storybookin käyttömahdollisuuksia erilaisissa tilanteissa. Esimerkiksi komponenttikirjaston rakentaminen helpottuu, kun lisäosien avulla voidaan testata ja havainnollistaa komponenttien eri tiloja. Toisena esimerkkinä on järjestelmän suunnittelu, jossa halutaan varmistaa useiden komponenttien yhteensopivuus ja uudelleenkäytettävyys. Lisäksi lisäosat tukevat visuaalista testausta, saavutettavuuden arviointia sekä mahdollisuutta jakaa Storybook helposti muiden tiimien ja sidosryhmien kanssa. Näin ne tukevat myös yhteistyötä ja kommentointia projektien eri vaiheissa. (Gift 2019.)

Tässä opinnäytetyössä käytetyt lisäosat esitellään tarkemmin luvussa 5, jossa käsitellään toteutettua projektia.

5 STORYBOOK MAINIO-PROJEKTISSA

Tässä luvussa käydään läpi, kuinka Storybook-työkalu otettiin käyttöön MAINIO-projektissa ja miten sen avulla rakennettiin pohja design systemille. Luvussa esitellään vaiheitttain ympäristön asennus, esimerkkikomponenttien luonti ja dokumentointi sekä lisäosien valinta ja asennus kehitystyön tukeksi. Lisäksi tuodaan esiin, millaisia käytännön ratkaisuja ja haasteita projektissa kohdattiin, ja kuinka ne ratkaistiin. Tarkoituksena on kuvata kokonaisuus konkreettisesti: miten työkalu saatiin liitettyä osaksi React ja Vite-ympäristöä, joka on JavaScript-kehitystyökalu (Vite 2025), sekä millaisia komponentteja sen avulla luotiin ja millä tavoin design tokenit, otettiin käyttöön yhtenäisen visuaalisen ilmeen varmistamiseksi. Myös lisäosien sekä testaus, asennus ja käyttö esitellään. Samalla havainnollistetaan työn tuloksia esimerkkikoodien ja kuvien avulla.



Kuvio 8. Projektin package.json sisältää Storybookin asennussisällön

5.1 Storybookin asennus ja rakenne

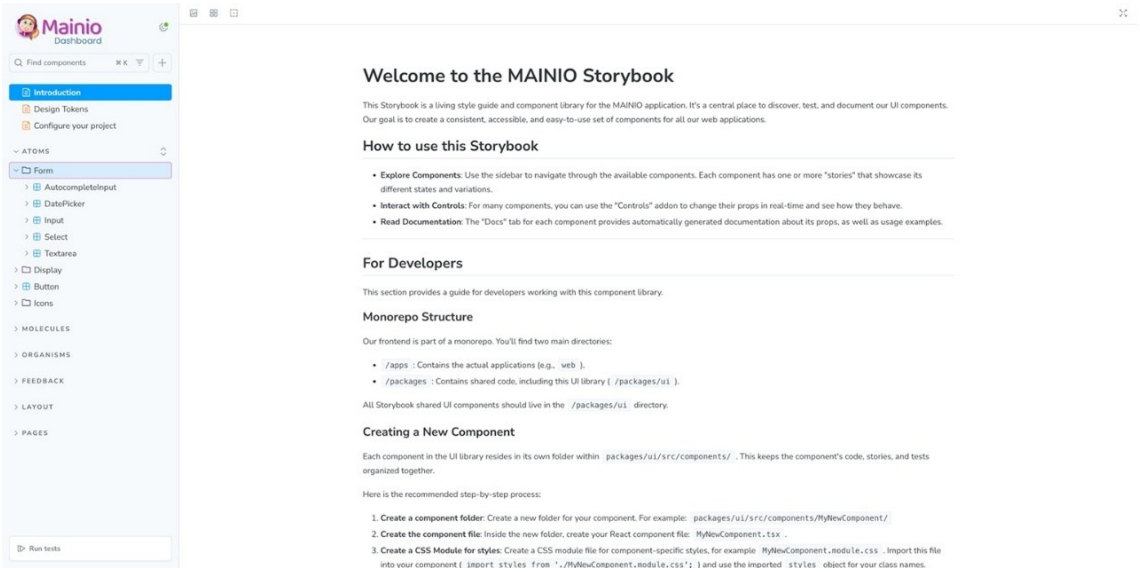
MAINIO-projektin koodi toteutettiin turborepon avulla monorepossa, mikä tarjosi modernin ja hallitun tavan organisoida useiden sovellusten ja jaettujen kirjastojen kokonaisuutta. Monorepo tarkoittaa sitä, että useiden eri sovellusten ja kirjastojen koodi säilytetään samassa kansiossa yhden projektin alla (Monorepo.tools 2025), turborepo taas on työkalu, joka nopeuttaa ja helpottaa monorepon käyttöä ajamalla tehtäviä rinnakkain (Vercel). Monorepo jakautui kahteen pääkansioon: apps sisälsi varsinaiset sovellukset, kuten Next.js-pohjaisen front-endin sekä Pythonilla rakennetun taustapalvelun, kun taas packages sisälsi jaettavat kirjastot. Storybookiin liittyvä työ tehtiin nimenomaan packages/ui-hakemistossa, joka toimi erillisenä, uudelleenkäytettävänä käyttöliittymäkomponenttien kirjastona. Tämä rakenne eriytti selkeästi käyttöliittymäkomponenttien kehityksen muusta projektista.

Käytännön toteutuksessa packages/ui oli jaettu lähdekoodihakemistoon (src), jossa komponentit, niiden tyylit ja dokumentaatio sijaitsivat yhtenäisinä kokonaisuuksina. Jokaisella komponentilla oli oma hakemistonsa, jonne oli koottu sen toiminnallisuus, tyylitiedostot, Storybook-tarinat kumpaa käytän tarinat vai storyt? ja vientitiedosto.

Projektin tekninen ympäristö oli rakennettu modernien työkalujen ja teknologioiden varaan. Front-end-sovellus toteutettiin Next.js-kehyksen versiolla 15 ja käyttöliittymäkirjasto Reactin versiolla 19. Koko front-endin kehityksessä käytettiin TypeScriptiä, mikä toi varmuutta tyypityksiin ja vähensi virheitä. Storybook liitettiin projektiin Vite-rakennustyökalun avulla, mikä mahdollisti nopean kehityspalvelimen käytön. Monorepon hallintaan hyödynnettiin Turborepoa ja Yarn workspaceseja, joka tarkoittaa, että Yarn hallitsee ja yhdistää riippuvuudet keskitetysti yhdellä asennuksella, mikä helpotti riippuvuuksien hallintaa ja yhtenäisten kehitystyökalujen käyttämistä koko projektissa.

Storybookin asetukset sijaitsivat omassa .storybook-hakemistossaan packages/ui-kansiossa. Tärkeimpiä tiedostoja olivat main.ts, joka määritteli ladattavat tarinat ja käytettävät lisäosat, sekä preview.tsx, jossa kaikki tarinat käärittiin Material-UI:n teemantarjoajan sisään yhtenäisen ulkoasun varmistamiseksi. Lisäksi manager.ts mahdollisti Storybook käyttöliittymän brändäyksen MAINIO-tuotteen mukaiseksi (kuvio 9). Storybookin käyttöliittymä mukautettiin vastaamaan projektin brändiä manager.ts-tiedoston avulla. Dokumentaatiossa hyödynnettiin MDX-tiedostoja, joiden avulla komponenttien esittelyjä ja käyttöohjeita voitiin kirjoittaa vapaamuotoisemmin (kuvio 10).

Kuvio 9. .storybook-hakemiston keskeiset tiedostot manager.ts, main.ts ja preview.tsx



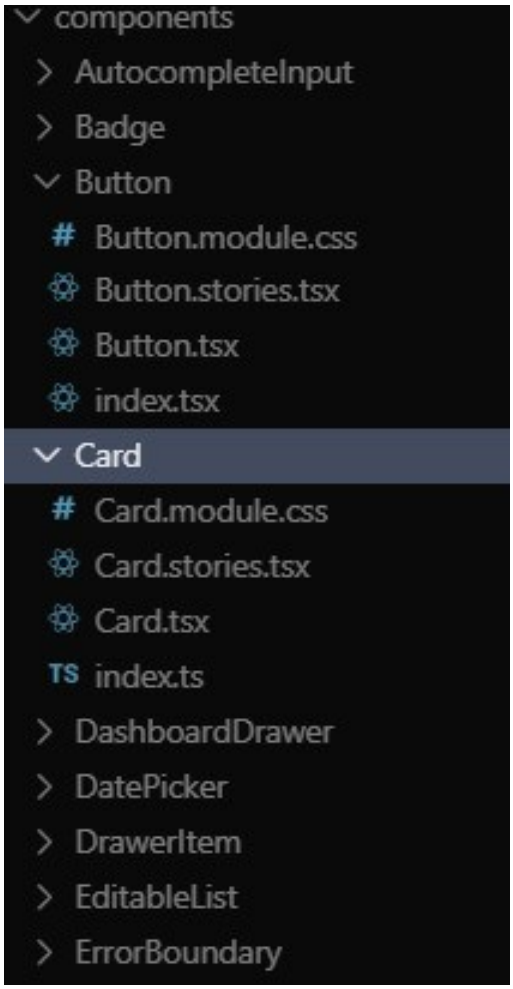
Kuvio 10. MDX-tiedosto Storybook-ympäristössä

5.2 Käyttöliittymäkirjaston rakentaminen ja komponenttien dokumentointi

Kun Storybook oli saatu asennettua ja konfiguroitua osaksi projektin monorepo-rakennetta, seuraava vaihe oli varsinaisen käyttöliittymäkirjaston luominen. Projektissa tämä toteutettiin packages/ui-hakemistoon, johon koottiin kaikki yhteiset käyttöliittymäkomponentit. Ratkaisun etuna oli se, että komponentit oli selkeästi

eriytetty omaksi kokonaisuudekseen ja niitä voitiin hyödyntää toistuvasti eri käyttöliittymän osissa. Tämä noudatti toimeksiannon tavoitetta rakentaa selkeä ja yhtenäinen design system, jota voidaan jatkossa helposti laajentaa ja ylläpitää.

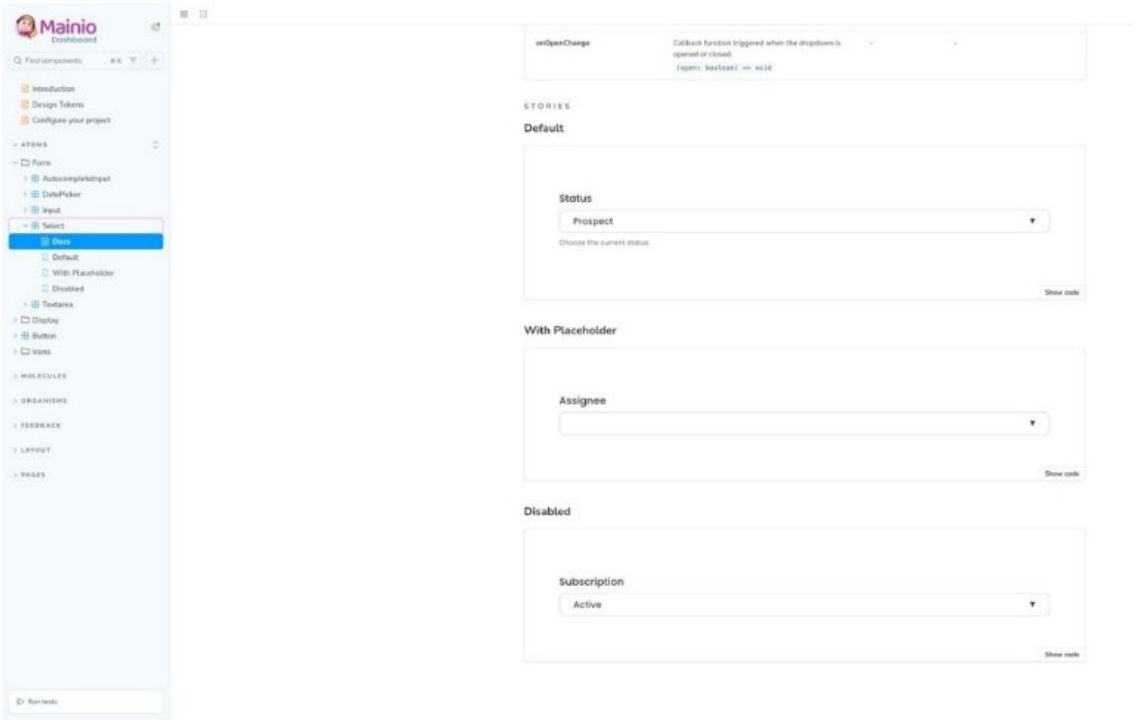
Käyttöliittymäkirjaston perusrakenne muodostui siten, että jokainen komponentti sai oman alikansionsa components-hakemiston sisällä (kuvio 11). Kansion sisältönä oli vähintään kolme tiedostoa: varsinainen komponentti (esimerkiksi Button.tsx), sen tarinat (Button.stories.tsx) sekä tyyliasetukset. Lisäksi dokumentoinnin ja testauksen tarpeisiin voitiin lisätä erillisiä tiedostoja. Tämä kansiorakenne teki selväksi, mitä kuhunkin komponenttiin kuului ja helpotti niiden ylläpitoa.



Kuvio 11. Komponentin kansiorakenne

Komponenttien dokumentoinnissa Storybook osoittautui toimeksiannon kannalta keskeiseksi työkaluksi. Jokaiselle komponentille luotiin tarinat, joiden avulla pystyttiin havainnollistamaan sen eri tilat ja variaatiot. Tämä ei palvellut ainoastaan dokumentointia, vaan myös kehitysvaiheessa tapahtuvaa testausta. Esimerkiksi

painikkeelle määriteltiin tarinoita (kuvio), joissa se esitettiin eri väreillä, kokoeroilla sekä ikoneilla varustettuna. Näin kehittäjät pystyivät näkemään yhdellä silmäyksellä, miltä komponentti näytti eri tilanteissa (kuvio 12).



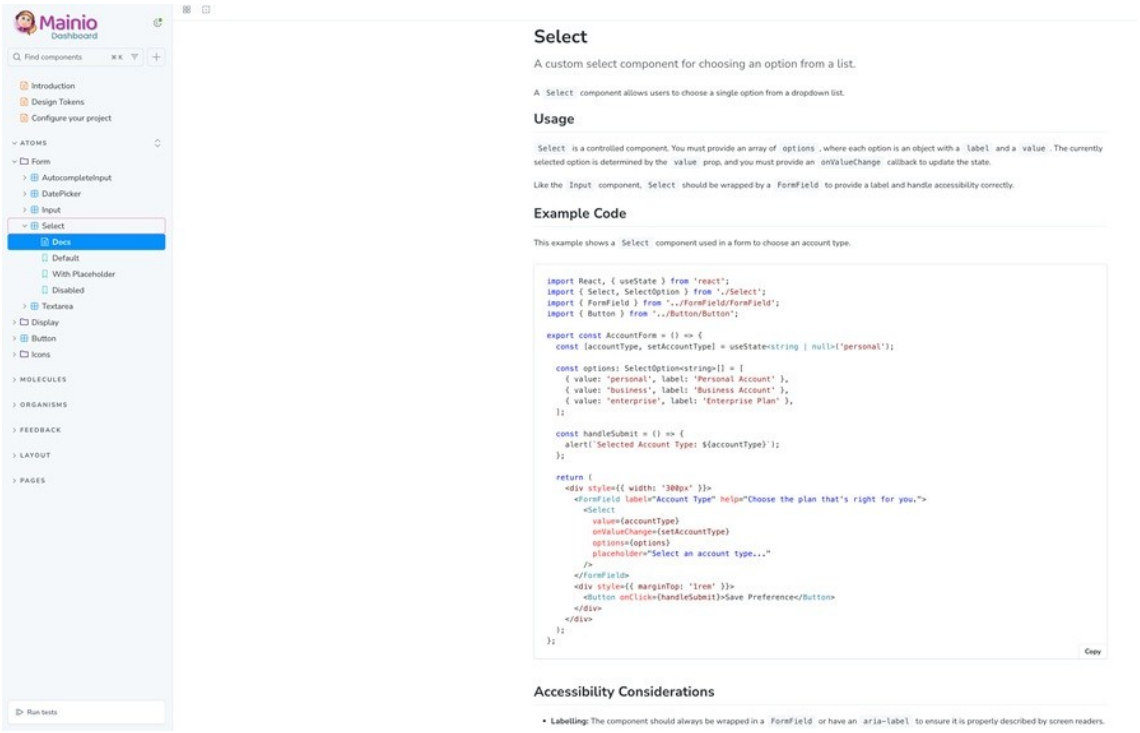
Kuvio 12. Tarinat select-komponentille Storybook-näkymässä

Koodiesimerkissä on määritelty useita painike-komponentin tarinoita, jotka kuvaavat sen eri variaatioita. Jokaiselle tarinalle annetaan args-parametrit, kuten variant ja children, joiden avulla komponentin ulkoasu ja sisältö määritellään. Näin muodostuvat Primary, Secondary ja Danger -painikkeet sekä erillinen Disabled-tila. Lisäksi koodissa on toteutettu InteractionTest, jossa Storybookin play-funktiolla simuloidaan käyttäjän vuorovaikutusta komponentin kanssa. Testissä haetaan painike käyttöliittymästä ja suoritetaan klikkaus, jonka onnistuminen varmistetaan odottamalla onClick-funktion käynnistymistä (kuvio 13).

```tsx
// Button-komponentin Storybook-tarinat osoittavat eri tilat ja variaatiot
export const Primary: Story = {
  args: {
    variant: 'primary',
    children: 'Primary Button',
  },
};

export const Secondary: Story = {
  args: {
    variant: 'secondary',
    children: 'Secondary Button',
  },
};

export const Danger: Story = {
  args: {
    variant: 'danger',
    children: 'Danger Button',
  },
};

export const Disabled: Story = {
  args: {
    children: 'Disabled Button',
    disabled: true,
  },
  render: (args: ButtonProps) => (
    <div style={{ display: 'flex', flexDirection: 'column', gap: '10px' }}>
      <Button {...args} variant="primary">Primary Disabled</Button>
      <Button {...args} variant="secondary">Secondary Disabled</Button>
      <Button {...args} variant="danger">Danger Disabled</Button>
    </div>
  )
};

export const InteractionTest: Story = {
  args: {
    children: 'Click Me',
    onClick: fn(),
  },
  play: async ({ args, canvasElement }) => {
    const canvas = within(canvasElement);
    const button = await canvas.getByRole('button', { name: /Click Me/i });
    await userEvent.click(button);
    await expect(args.onClick).toHaveBeenCalled();
  },
};
```

Kuvio 13. Storybook-tarinat osoittavat painike-komponentin kaikki eri variaatiot: primary, secondary, danger, disabled sekä interaktiivisuustestit

Storybookin docs-välilehti tarjosi automaattisesti muodostetun dokumentaation, joka täydentyi komponentin prop-tyypeillä ja selityksillä. Tämä vastasi toimeksiannon vaatimusta siitä, että luodusta design systemistä tulisi mahdollisimman selkeä ja käytännöllinen myös muille kuin alkuperäisille kehittäjille (kuvio 15).



Kuvio 15. Storybook-näkymä select-komponentin Docs-tiedostosta

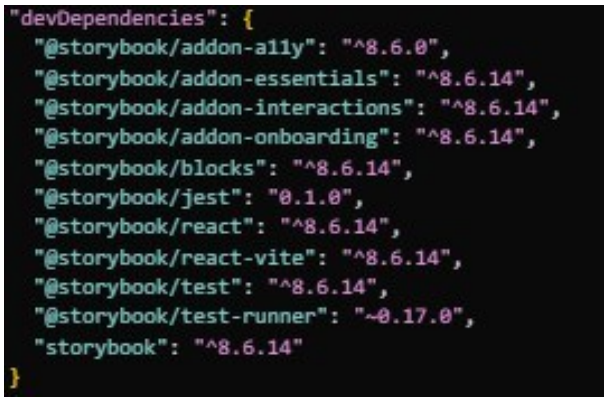
Komponenttien toteutuksessa edettiin atomic design -ajattelun mukaisesti. Ensin rakennettiin pienimmät uudelleenkäytettävät elementit, kuten painikkeet ja syötekentät, joista muodostettiin suurempia kokonaisuuksia esimerkiksi lomakkeiden ja välilehtien käyttöön. Tämä vaiheittainen eteneminen mahdollisti sen, että myöhemmissä kehitysvaiheissa voitiin hyödyntää jo valmiiksi testattuja ja yhtenäisellä tavalla dokumentoituja palikoita. Ratkaisulla vähennetään päällekkäistä työtä ja luotiin perusta käyttöliittymäkirjastolle.

Käyttöliittymäkirjaston rakentamisen yhteydessä otettiin käyttöön myös design tokenit, joilla varmistettiin käyttöliittymän yhtenäinen ilme. Tokenit määriteltiin erilliseen tokens.css-tiedostoon, josta ne voitiin hyödyntää kaikissa komponenteissa. Tämä sisälsi esimerkiksi värit, typografian ja tilakohtaiset määrittelyt. Ratkaisu helpottaa kirjaston ylläpitoa, sillä esimerkiksi värimaailmaa voitiin muuttaa keskitetysti vain muokkaamalla yhtä tiedostoa.

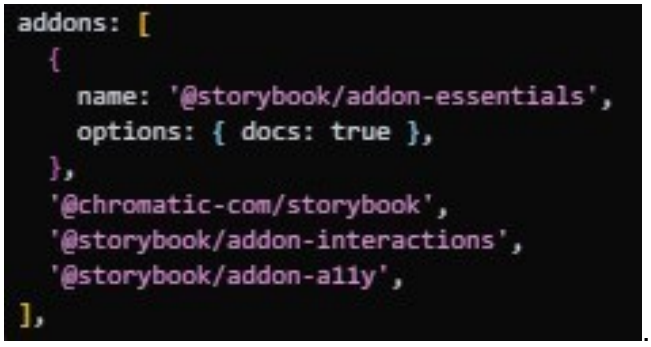
5.3 Lisäosat

Storybookin perusominaisuuksia laajennettiin MAINIO-projektissa useilla lisäosilla, jotka tukivat komponenttikirjaston kehitystä ja dokumentointia toimeksiannon vaatimusten mukaisesti. Ilman lisäosia Storybook olisi toiminut vain komponenttien esittelyalustana, mutta niiden avulla siitä muodostui käytännöllinen ja monipuolinen työkalu kehittäjille.

Lisäosien asennus tehtiin Yarnin avulla. Jokainen lisäosa lisättiin komennolla yarn add -D @storybook/addon-nimi, minkä jälkeen ne otettiin käyttöön .storybook/main.ts-tiedostossa lisäämällä ne addons-taulukkoon. Joidenkin lisäosien kohdalla törmättiin yhteensopivuusongelmiin, sillä kaikki paketit eivät tukeneet MAINIO-projektissa käytössä ollutta Storybookin versiota. Tämä näkyi muun muassa, virheilmoituksina kehityspalvelinta käynnistäessä. Ongelmat ratkaistiin kokeilemalla eri versioyhdistelmiä ja toimivat versiot package.json-tiedostoon (kuvio 16). Tämä vaihe oli tärkeä, jotta projektin ympäristö pysyi toimivana, vakaana ja toistettavissa jatkossa.



Kuvio 15. Storybook-lisäosien asennus ja konfiguraatio package.json:issa



Kuvio 16. Storybook-lisäosien aktivointi .storybook/main.ts -tiedostossa

Dokumentoinnin kannalta keskeisin lisäosa oli @storybook/addon-docs (kuvio 16). Sen avulla jokaiselle komponentille muodostui oma docs-sivu, jossa näkyivät komponentin propsit automaattisesti taulukkomuodossa sekä esimerkkikäyttö (kuvio 17). Koska projektissa käytettiin TypeScriptiä, lisäosa pystyi lukemaan tyypitykset suoraan lähdekoodista ja liittämään ne dokumentaatioon. Tämä vähensi manuaalista työtä ja varmisti, että dokumentaatio pysyi ajan tasalla aina, kun komponenttia muutettiin.

```
const meta: Meta<typeof Button> = {
  title: 'Atoms/Button',
  component: Button,
  tags: ['autodocs'], // Enables automatic documentation generation
  // ... muut asetukset
};
```

Kuvio 17. Automaattinen dokumentaatiogenerointi autodocs: true -asetuksella

Kehitystyötä tuki myös storybook/addon-controls (kuvio 18), jonka avulla komponenttien propseja voitiin muokata suoraan käyttöliittymässä. Controls-lisäosa lisäsi Storybookin sivupaneeliin lomakekentät kaikille dokumentoiduille ominaisuuksille. Käytännössä tämä tarkoitti sitä, että esimerkiksi painikkeen väri, koko tai teksti voitiin vaihtaa lennosta ilman, että lähdekoodia muokattiin. Tämä nopeutti huomattavasti eri variaatioiden kokeilua ja teki kirjastosta hyödyllisen myös suunnittelun tueksi.

```
argTypes: {
  variant: {
    control: 'select',
    options: ['primary', 'secondary', 'danger'],
    description: 'The visual style of the button.',
    table: {
      defaultValue: { summary: 'primary' },
    },
  },
  disabled: {
    control: 'boolean',
    description: 'Whether the button is interactive or not.',
    table: {
      defaultValue: { summary: 'false' },
    },
  },
}
```

Kuvio 18. Controls-lisäosan konfiguraatio argTypes-määrittelyssä

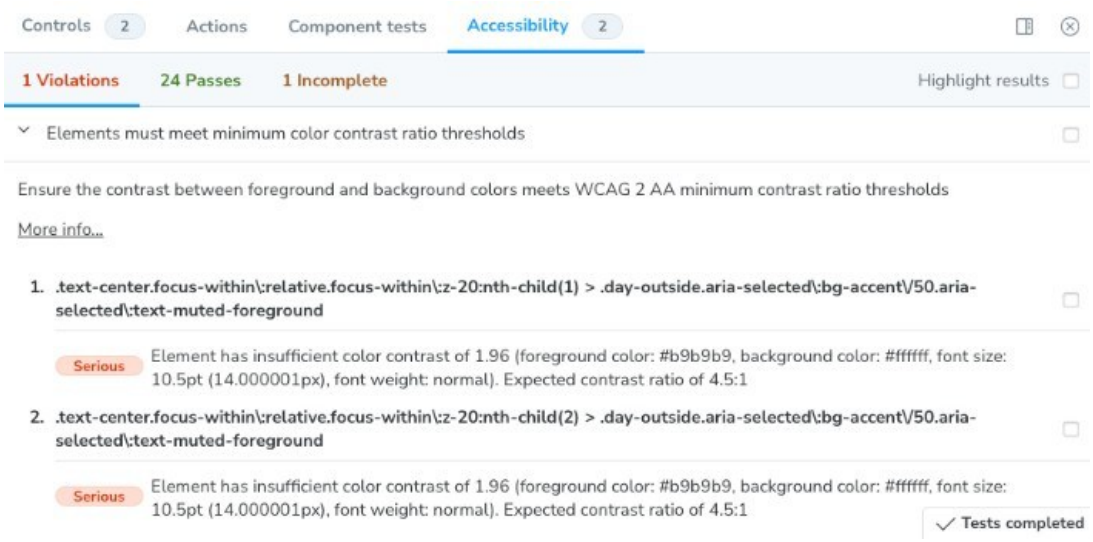
Interaktiivisten komponenttien testaamiseen otettiin käyttöön @storybook/addon-actions (kuvio 19). Sen avulla voitiin tarkastella, mitä tapahtumia komponentti tuotti. Esimerkiksi painikkeen klikkaus kirjautui Actions-paneeliin, jossa näkyi tapahtuman nimi ja sen mukana välitetyt parametrit. Tämä helpotti huomattavasti tapahtumien toiminnan todentamista ilman, että selaimen konsolia täytyi seurata. Kuvaksi sopii Action-paneelin näkymä, jossa painikkeen klikkaus näkyy kirjauksena.

```
export const InteractionTest: Story = {
  args: {
    children: 'Click Me',
    onClick: fn(), // Actions-lisäosa kirjaa tapahtuman
  },
  play: async ({ args, canvasElement }) => {
    const canvas = within(canvasElement);
    const button = await canvas.getByRole('button', { name: /Click Me/i });
    await userEvent.click(button);
    await expect(args.onClick).toHaveBeenCalled();
  },
};
```

Kuvio 19. Actions-lisäosan käyttö tapahtumien kirjaamiseen

Käyttöliittymän responsiivisuuden varmistamiseen hyödynnettiin @storybook/addon-viewport. Lisäosa lisäsi Storybookin työkalupalkin pudotusvalikon, josta näkymä voitiin vaihtaa eri laitekokoihin. Näin komponenttien toimivuutta voitiin tarkastella suoraan Storybookissa mobiili-, tabletti- ja työpöytänäkymissä ilman erillistä selaimen kehittäjätilaa.

Saavutettavuuden huomioimiseksi asennettiin @storybook/addon-a11y lisäosa (kuvio 20). Lisäosa rakensi automaattisen raportin, jossa esitettiin esimerkiksi kontrastivirheet, puuttuvat alt-tekstit tai virheelliset aria-atribuutit. Tämä teki saavutettavuuden tarkistamisesta helposti osan kehitysprosessia ja mahdollisti virheiden korjaamisen jo varhaisessa vaiheessa.



Kuvio 20. A11y lisäosa

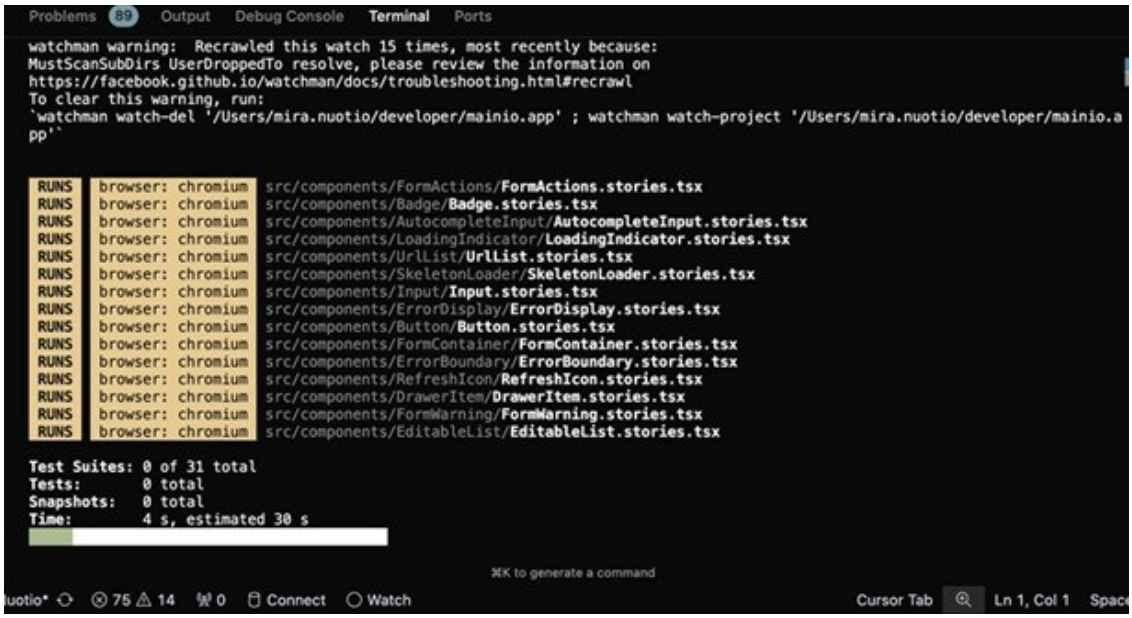
Visuaalisen ja teknisen yhtenäisyyden varmistamiseen käytettiin storybook-design-token -lisäosaa. Se linkitettiin projektin tokens.css-tiedostoon, johon oli määritelty värit, typografia ja muut visuaaliset perusmäärittelyt. Lisäosa muodosti Storybookiin oman näkymän, jossa nämä tokenit esitettiin selkeästi ja havainnollistettiin esimerkiksi väripaleteilla. Tämä ratkaisu varmisti, että komponentit noudattivat yhtenäistä visuaalista linjaa ja muutokset näkyivät keskitetysti koko kirjastossa. Projektin värimäärittelyt näkyvät visuaalisina lohkoina Design Token -paneelissa (kuvio 21).

Kuvio 21. Design tokens Storybook-näkymä

Kokonaisuutena lisäosat tekivät Storybookista MAINIO-projektissa monipuolisen työkalun, joka tuki sekä kehittäjiä että toimeksiantajaa. Ne nopeuttivat komponenttien testaamista, selkeyttivät dokumentaatiota, varmistivat saavutettavuuden ja pitivät visuaalisen ilmeen yhtenäisenä. Yhdessä nämä ominaisuudet muodostivat perustan toimeksiannon edellyttämälle design systemille ja helpottivat sen käyttöönottoa tulevissa kehitysvaiheissa.

5.4 Komponenttien testaaminen Storybookissa

MAINIO-projektissa testattavuuden varmistamiseksi otettiin käyttöön Storybookin oma Test Runner, joka hyödyntää taustalla Playwright-teknologiaa. Sen avulla jokainen tarina voidaan suorittaa selaimessa, ja komponenttien renderöityminen voitiin varmistaa automaattisesti. Test Runner ajaa niin sanottuja smoke testejä (kuvio 22), joissa tarkastetaan, että komponentti renderöityy ilman virheitä.



Kuva 22. Storybook Test Runner käynnissä. Tarinoiden testiajo konsolilokissa

Kuviossa 22 on esimerkki smoke testien suorittamisesta terminaalissa komennolla npm run test-storybook. Terminaalinäkymässä Storybook Test Runner käynnistää Playwright Chromium-moottorilla ja renderöi tarinan yksi kerrallaan. Loki listaa parhaillaan ajettavat tiedostot, kertoo ajon tilastot mm. ajon kesto, sekä kertoo kuinka monta testiä on tehty ja testin yhteenvedon.

```
module.exports = {
  async play({ page, story }) {
    // Test Runner suorittaa play-funktiot automaattisesti
    // ja mahdollistaa CI/CD-ympäristössä testauksen
  },
};
```

Kuvio 23. Storybook Test Runner -konfiguraatio .storybook/test-runner.js:ssä

Test Runnerin lisäksi projektiin asennettiin storybook/test -kirjasto (kuvio 24). Se tarjoaa Jest- ja Vitest-yhteensopivan expect-rajapinnan sekä Testing Library -apuja, joita voidaan hyödyntää tarinoiden play-funktioissa interaktioiden ja assertoinnin kirjoittamiseen. Näin komponenttien toiminnallisuuksia voidaan testata suoraan niiden tarinoiden yhteydessä.

```
import { fn, userEvent, within } from '@storybook/test';
import { expect } from '@storybook/jest';

export const InteractionTest: Story = {
  args: {
    children: 'Click Me',
    onClick: fn(),
  },
  play: async ({ args, canvasElement }) => {
    const canvas = within(canvasElement);
    const button = await canvas.getByRole('button', { name: /Click Me/i });
    await userEvent.click(button);
    await expect(args.onClick).toHaveBeenCalled();
  },
};
```

Kuvio 24. Storybook/test -kirjasto ja Testing Library -apujen käyttö play-funktioissa

Saavutettavuuden osalta projektiin lisättiin @storybook/addon-a11y, joka suorittaa axe-core-kirjastoon pohjautuvia tarkistuksia. Lisäosa raportoi mahdolliset saavutettavuusongelmat, kuten kontrastivirheet tai puuttuvat ARIA-attribuutit, ja antaa kehittäjälle ehdotuksia korjaustoimenpiteistä (kuvio 26).

Lisäksi testauksen tueksi otettiin käyttöön Mock Service Worker (MSW) yhdessä sen Storybook-lisäosan kanssa (kuvio 25). Näiden avulla voitiin mockata API-kutsuja tarinoissa, jolloin komponentteja pystyttiin testaamaan realistisesti ilman riippuvuutta taustapalveluista.

```
"devDependencies": {
  "msw": "^2.10.2",
  "msw-storybook-addon": "^2.0.5",
  "@storybook/addon-interactions": "^8.6.14",
  "@storybook/test": "^8.6.14"
}
```

Kuvio 25. MSW-lisäosan konfiguraatio package.json:ssa

```
// A11y-lisäosa suorittaa automaattisesti axe-core-tarkistukset
// ja raportoi mahdolliset saavutettavuusongelmat

// Esimerkki A11y-parametrien määrittelystä tarinassa
export const AccessibleButton: Story = {
  args: {
    children: 'Accessible Button',
    'aria-label': 'Submit form',
  },
  parameters: {
    a11y: {
      // A11y-lisäosa testaa automaattisesti
      // - kontrastisuhteet
      // - ARIA-attribuutit
      // - näppäimistönavigaatio
      // - screen reader -yhteensopivuus
    },
  },
};
```

Kuvio 26. A11y-lisäosan automaattinen saavutettavuustestaus

Projektin riippuvuuksiin kuuluu myös Chromatic, joka mahdollistaa visuaalisen regressiotestauksen vertaamalla komponenttien renderöitymistä eri versioiden välillä. Tätä ominaisuutta ei kuitenkaan ehditty tässä opinnäytetyössä ottamaan käyttöön, mutta tästä jätettiin huomio projektin dokumentaatioon. Näiden määritysten jälkeen MAINIO-projektiin saatiin käyttöön Storybook-ympäristö, joka tukee sekä komponenttien toiminnallisuuden että saavutettavuuden testaamista ja tarjoaa pohjan myös visuaalisen testauksen hyödyntämiselle tulevaisuudessa.

5.5 Esimerkkikomponentti

Tässä luvussa esitellään esimerkkikomponentti, joka havainnollistaa, kuinka komponenttilähtöinen kehitys toteutettiin MAINIO-projektissa Storybookin avulla.

Esimerkkikomponenttina toimii Button-komponentti, eli painike, joka on osa käyttöliittymän perusrakennetta ja jota hyödynnetään useissa eri näkymissä (kuvio 27). Komponentti on rakennettu erillisenä kokonaisuutena siten, että se ei ole riippuvainen muista komponenteista, ja sen eri tilat on määritelty Storybookissa omiksi tarinoikseen.

```tsx
import React from 'react';
import './Button.module.css';

export interface ButtonProps {
  variant?: 'primary' | 'secondary' | 'danger';
  disabled?: boolean;
  children: React.ReactNode;
  onClick?: () => void;
  className?: string;
}

export const Button: React.FC<ButtonProps> = ({
  variant = 'primary',
  disabled = false,
  children,
  onClick,
  className,
}) => {
  return (
    <button
      className={`button ${variant} ${disabled ? 'disabled' : ''} ${className || ''}`}
      disabled={disabled}
      onClick={onClick}
    >
      {children}
    </button>
  );
};
```

Kuvio 27. Button-komponentin perusrakenne ja TypeScript-tyypitys

Komponentille määriteltiin useita tiloja, kuten perustila, hover-tila ja disabled-tila (kuvio 28). Jokainen tila kuvattiin erikseen tarinana, jolloin niiden toiminta ja ulkoasu voitiin testata ja dokumentoida itsenäisesti. Storybookin avulla komponentti saatiin esitettyä käyttöliittymästä irrallaan, mutta kuitenkin samoilla ominaisuuksilla, joita se hyödyntää varsinaisessa sovelluksessa.

```tsx
import type { Meta, StoryObj } from '@storybook/react';
import { Button, ButtonProps } from './Button';

const meta: Meta<typeof Button> = {
  title: 'Atoms/Button',
  component: Button,
  tags: ['autodocs'],
  argTypes: {
    variant: {
      control: 'select',
      options: ['primary', 'secondary', 'danger'],
      description: 'The visual style of the button.',
    },
    disabled: {
      control: 'boolean',
      description: 'Whether the button is interactive or not.',
    },
  },
  args: {
    children: 'Button Text',
    disabled: false,
    variant: 'primary',
  },
};

export default meta;
type Story = StoryObj<typeof Button>;

// Perustila
export const Primary: Story = {
  args: {
    variant: 'primary',
    children: 'Primary Button',
  },
};
 // Hover-tila (CSS-pseudoklassi)
 export const HoverState: Story = {
   args: {
     variant: 'primary',
     children: 'Hover Me',
   },
   parameters: {
     pseudo: { hover: true },
   },
 };

 // Disabled-tila
 export const Disabled: Story = {
   args: {
     children: 'Disabled Button',
     disabled: true,
   },
   render: (args: ButtonProps) => (
     <div style={{ display: 'flex', flexDirection: 'column', gap: '10px' }}>
       <Button {...args} variant="primary">Primary Disabled</Button>
       <Button {...args} variant="secondary">Secondary Disabled</Button>
       <Button {...args} variant="danger">Danger Disabled</Button>
     </div>
   )
 };
```

Kuvio 28. Button-komponentin Storybook-tarinat eri tiloille

Storybookin avulla voidaan myös yhdistää testaus ja dokumentointi. Edellä kuvatun komponentin play-funktioon (kuvio 29) on mahdollista liittää yksinkertainen testi, joka varmistaa, että painikkeen teksti renderöityy oikein ja että painike reagoi odotetusti klikkaustapahtumaan. Tämä lisää testattavuutta ja tukee komponenttien laadunvarmistusta kehityksen aikana.

```
import { fn, userEvent, within } from '@storybook/test';
import { expect } from '@storybook/jest';

export const InteractionTest: Story = {
  args: {
    children: 'Click Me',
    onClick: fn(),
  },
  play: async ({ args, canvasElement }) => {
    const canvas = within(canvasElement);

    // Testataan että painike renderöityy oikein
    const button = await canvas.getByRole('button', { name: /Click Me/i });
    expect(button).toBeInTheDocument();
    expect(button).toHaveTextContent('Click Me');

    // Testataan klikkaus
    await userEvent.click(button);
    await expect(args.onClick).toHaveBeenCalled();

    // Testataan että painike on käytettävissä
    expect(button).not.toBeDisabled();
  },
};
```

Kuvio 29. Play-funktio testauksen ja dokumentoinnin yhdistämiseksi

6 POHDINTA

Tämän opinnäytetyön päätavoitteena oli selvittää, miten Storybook-työkalua voidaan hyödyntää selkeän ja laajennettavan design systemin rakentamisessa. Työn tuloksena syntyi dokumentoitu käyttöliittymäkomponenttien kirjasto, joka tukee MAINIO-tuotteen kehitystyötä Mindhive Oy:ssä. Ratkaisun keskeisiä hyötyjä olivat komponenttien systemaattinen hallinta, dokumentoinnin automatisoituminen sekä design tokenien avulla saavutettu visuaalinen yhtenäisyys. Atomic design -ajattelu tarjosi lisäksi rakenteen, jonka avulla komponenttikirjasto voitiin rakentaa vaiheittain pienistä elementeistä kokonaisuuksiksi. Tulokset osoittivat, että valittu lähestymistapa tuki toimeksiannon tavoitteita: lopputulos on hyödynnettävissä myös jatkokehityksessä ja ylläpidossa.

Työ hyödytti useita osapuolia. Toimeksiantajalle se tarjosi pohjan yhtenäiselle design systemille, mikä nopeuttaa käyttöliittymäkehitystä ja vähentää virheitä tulevaisuudessa. Omalla kohdallani projekti tarjosi merkittävän oppimisprosessin: ilman aiempaa kokemusta Storybookista, monorepo-arkkitehtuurista tai design systemeistä sain konkreettisen mahdollisuuden syventää osaamistani. Työ tuottaa arvoa myös laajemmin, sillä komponenttikirjaston dokumentoinnin ja yhtenäisten käytäntöjen avulla kehitystyö helpottuu muidenkin kehittäjien ja suunnittelijoiden näkökulmasta.

Toteutuksen aikana kohdatut haasteet liittyivät erityisesti uusien työkalujen ja lisäosien käyttöönottoon, kuten yhteensopimattomiin versioihin. Nämä tilanteet ratkesivat kuitenkin kokeilujen ja dokumentaation avulla, mikä vahvisti ongelmanratkaisukykyä ja toi ymmärrystä työkalujen rajoituksista. Projektin luonne kehittämispainotteisena työnä edellytti jatkuvaa oppimista ja reflektointia, mikä oli linjassa työn tavoitteiden kanssa.

Eettisyyden ja luotettavuuden näkökulmasta työ toteutettiin toimeksiantajan kanssa sovitun mukaisesti ja työn tulokset ovat läpinäkyvästi dokumentoituja. Hyödynnettävyys on selkeä: syntynyt komponenttikirjasto voidaan ottaa käyttöön osana MAINIO-tuotteen jatkokehitystä ja sen avulla voidaan varmistaa visuaalinen ja tekninen yhdenmukaisuus. Jatkotutkimusaiheiksi voidaan nostaa esimerkiksi design systemin laajentaminen entistä kattavammaksi.

LÄHTEET

Bigelow S. 2024. What are components? Viitattu 7.8.2025 https://www.techtarget.com/whatis/definition/component

Cox, B. J. 1986. Object-Oriented Programming: An Evolutionary Approach. Reading, Mas: Addison-Wesley.

De La Cuarda, P. 2016 Frontend Modularization Principles. Viitattu 11.8.2025 https://medium.com/@pedrodelacuadra/frontend-modularization-principles

Google 2025. Design tokens. Viitattu 8.8.2025 https://m3.material.io/foundations/design-tokens/overview.

Frost, B. 2016. Atomic Design. Viitattu 4.8.2025 https://bradfrost.com/blog/post/atomic-web-design/.

GeeksforGeeks 2019. Cohesion and Coupling. Viitattu 8.8.2025 https://www.geeksforgeeks.org/software-engineering/software-engineering-differences-between-coupling-and-cohesion/.

Gift, E. 2019. Improving Developer Experience with Storybook Addons. Medium. Viitattu 8.8.2025 https://medium.com/@lauragift21/improving-developer-experience-with-storybook-addons-7e8a92cbb4bb.

GitHub 2021. Storybook Design Token. Viitattu 5.8.2025 https://github.com/UX-and-I/storybook-design-token/blob/master/docs/teaser.png.

Kawamoto, D. 2024. 18 Best Design System Examples. Viitattu 11.8.2025 https://builtin.com/articles/design-system

Heineman, G. T., & Councill, W. T. 2001. Component-Based Software Engineering: Putting the Pieces Together. Boston: Addison-Wesley.

IxDf 2025. What are Design Systems? Interaction Design Foundation. Viitattu 11.8.2025 https://www.interaction-design.org/literature/topics/design-systems.

Kholmatova, A. 2017. Design systems : practical guide to creating design languages for digital products. Freiburg: Smashing Media AG.

Kostiuchenko, O 2025. Why Developers Love Good Design Systems. Viitattu 5.8.2025 https://phenomenonstudio.com/blog/why-developers-love-good-design-systems-and-how-to-get-buy-in.

Maza, L 2020. Developer experience in design systems. Viitattu 1.8.2025 https://medium.com/@larimaza/dx-developer-experience-in-design-systems-21ec53037c7d.

Mindhive Oy 2025. Viitattu 28.5.2025 https://www.mindhive.fi/.

Naur, P., & Randell, B. 1969. Software Engineering: Report on a Conference Sponsored by the NATO Science Committee.

Monorepo.tools 2025. Understanding Monorepos. Viitattu 11.8.2025 https://monorepo.tools/.

Qu, X., Damoah, A., Sherwood, J., Liu, P., Jin, C. S., Chen, L., Shen, M., Aleisa, N., Hou, Z., Zhang, C., Gao, L., Li, Y., Yang, Q., Wang, Q., & De S., Cristabelle. 2025. A Comprehensive Review of AI Agents: Transforming Possibilities in Technology and Beyond. Viitattu 8.8.2025 https://arxiv.org/abs/2508.11957.

Shilman, M. 2017. The Storybook Story: From UI development startup to open collective. Viitattu 15.8.2025 https://storybook.js.org/blog/the-storybook-story/.

Storybook 2025a. What is Story? Viitattu 1.8.2025 https://storybook.js.org/docs/get-started/whats-a-story.

Storybook 2025b. Get started with Storybook. Viitattu 1.8.2025 https://storybook.js.org/docs.

Storybook 2025c. Why Storybook? Viitattu 2.8.2025 https://storybook.js.org/docs/writing-stories.

Storybook 2025d. How to write stories. Viitattu 2.8.2025 https://storybook.js.org/docs/writing-stories.

Storybook 2025e. Esssentials. Viitattu 10.8.2025 https://storybook.js.org/docs/writing-stories.

Storybook 2025f. Storybook Addons. Viitattu 1.8.2025 https://storybook.js.org/docs/configure/user-interface/storybook-addons.

Turborepo 2025. Introduction. Viitattu 10.8.2025 https://turbo.build/repo/docs.

Vilkka, H. 2021. Näin onnistut opinnäytetyössä: Ratkaisut tutkimuksen umpikujiin. Jyväskylä: PS-kustannus.

Vite 2025. The Build Tool for the Web Viitattu 4.8.2025 https://vite.dev/.

WCAG 2018. Web Content Accessibility Guidelines. Viitattu 15.8.2025 https://www.w3.org/TR/WCAG21.