VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Ville Varjus

Tuotekomponentin koon visualisointi DTCA-työkalun laajennettavassa moduulissa

Tekniikka
2024

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Ville Varjus
Opinnäytetyön nimi	Tuotekomponentin koon visualistointi DTCA-työkalun laajennettavassa moduulissa
Vuosi	2024
Kieli	suomi
Sivumäärä	43 + 2 liitettä
Ohjaaja	Mikael Jakas

Opinnäytetyön taustalla on tarve kehittää Wärtsilän sisäisen DTCA-työkalun web-sovellusta vastaamaan paremmin käyttäjien tarpeita materiaalien, komponenttien ja moottorien kokoa koskevien tietojen visualisoinnissa. Tässä työssä suunniteltiin ja toteutettiin uusi ominaisuus, joka mahdollistaa käyttäjille selkeän tavan visualisoida näiden elementtien kokoa niiden painoarvon perusteella.

Työ pohjautuu käyttäjäkeskeisen suunnittelun, ja web-sovellusten kehittämisen periaatteisiin. Keskeiset, ja valitut toteutusmenetelmät web-sovelluksen jatkokehityksessä ovat React, TypeScript, Python ja Django. Työssä hyödynnettiin iteratiivista kehitysmenetelmää, jossa käyttäjäpalautetta kerättiin ja hyödynnettiin jatkuvasti kehitysprosessin aikana.

Keskeiset havainnot osoittavat, että uusi ominaisuus DTCA-työkalussa tarjoaa loppukäyttäjille selkeän ja tehokkaan tavan visualisoida materiaalien kokoa painon perusteella, lisäksi ominaisuus on suunniteltu, toteutettu ja testattu onnistuneesti. Opinnäytetyön aikana perehdyttiin syvemmin sovelluksen jatkokehitykseen. Samalla tehtiin havaintoja uuden ominaisuuden jatkokehitysmahdollisuuksista, haasteista, ja mahdollisuuksista. Lopuksi käytiin läpi toteutettu tuotekomponentin koon visualisointiin tarkoitettu ominaisuus sekä opinnäytetyötä tehdessä havaitut jatkokysymykset ja oivallukset.

Avainsanat React, TypeScript, Python, Django, visualisointi, Web-sovellus, ohjelmointi, ominaisuus

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Ville Varjus
Title	Visualization of Product Component Size in the Expandable Module of DTCA Tool
Year	2024
Language	Finnish
Pages	43 + 2 Appendices
Name of Supervisor	Mikael Jakas

The background of the thesis lies in the need to develop Wärsilä's internal DTCA tool web application to better meet user's needs in visualizing the size of materials, components, and engines. In this thesis, a new feature was designed and implemented, enabling users to clearly visualize the size of these elements based on their weight.

The thesis is based on user-centered design principles and web application development practices. The key and selected implementation methods for the further development of the web application include React, TypeScript, Python and Django. An iterative development approach was employed, where user feedback was continuously collected and utilized throughout the development process.

Key observations indicate that the new feature in the DTCA tool provides end-users with a clear and effective way to visualize the size of materials based on their weight, and the feature has been successfully designed, implemented and tested. During the thesis work, deeper insights were gained into the further development of the large-scale application. Additionally, observations were made regarding the possibilities, challenges, and opportunities for the further development of the new feature. Finally, the implemented feature for visualizing the size of product components was reviewed, along with the follow-up questions and insights gained during the thesis work.

Keywords React, TypeScript, Python, Django, visualization, web application, programming, feature

SISÄLLYS

KUVA- JA TAULUKKOLUETTELO

LIITELUETTELO

TERMIT JA LYHENTEET

DTCA-TYÖKALU	Design To Cost Assistant Tool, web-sovellus
API, RAJAPINTA	Ohjelmointirajapinta (engl. Application programming interface)
FRONTEND	Verkkosivuston tai sovelluksen osa, joka sisältää käyttöliittymän.
BACKEND	Verkkosivuston tai sovelluksen osa, johon kuuluvat esim. tiedostojen käsittelyt. Backend siis käsittelee, siirtää ja vastaanottaa verkkosivuston tai sovelluksen tietoja.
VERKKOKEHYS	Ohjelmointikehityksessä käytettävä työkalu tai ympäristö, joka tarjoaa valmiita rakenteita ja toiminnallisuuksia web-sovellusten suunnitteluun, kehittämiseen ja ylläpitoon.
MVT-MALLI (MODEL-VIEW-TEMPLATE)	Arkkitehtuurityyppi Djangossa, joka sisältää Model-, View- ja Template-komponentit. Nämä komponentit pelaavat yhteen ja käsittelevät logiikan, datan ja esittämisen Backendin puolella.
FUNKTIO	Tietyn toiminnon suorittamiseen tarkoitettu käskysarja, joka palauttaa arvon.

TYPE	Tyyppi TypeScriptissä. Esim. String | Number | Undefined
STRING	Merkkijono
ORM (ENGL. OBJECT-RELATIONAL MAPPING)	Objekti-relaatiokartoitus. Tietojenkäsittelytieteessä käytettävä tekniikka, jolla saadaan muunnettua tietoa relaatiotietokannan taulukoista ohjelmointikielen käyttöön sopiviksi objekteiksi.
RELAATIOMALLI	Relaatiomallissa tietokantaan tallennettava data esitetään järjestettyinä äärellisinä listoina, jotka on ryhmitelty relaatioiksi. Relaatiota vastaava tietokantatermi on taulu.
FEATURE	Toiminto, ominaisuus
UNIT TEST	Yksittäisten komponenttien tai moduulien testaamista erikseen varmistaakseen niiden toiminnan oikein.
INTEGRAATIOTESTAUS	Testaa eri osien yhteistoimintaa ja integraatiota keskenään. Tavoitteena on varmistaa, että eri komponentit toimivat yhdessä odotetulla tavalla.
RENDER-METODI	Tuottaa react-elementtejä, jotka kuvaavat, miten komponentti näytetään käyttäjälle.

REACT-KOMPONENTTI	Itsenäinen ja uudelleenkäytettävä osa, joka voi sisältää HTML-elementtejä, muita komponentteja ja logiikkaa. Kaikilla react-komponenteilla on render-metodi, joka määrittelee sen palauttaman näkymän. Komponentit voivat myös vastaanottaa props-nimisiä ominaisuuksia, jotka välitetään niille niiden vanhemmilta komponenteilta.
JSX-ELEMENTTI	Reactissa käytetty tapa kapseloida useita elementtejä yhden ylemmän tason elementin sisään ilman, että tarvitsee luoda ylimääräistä diviä tai muuta HTML-elementtiä.
FUNKTIONAALINEN KOMPONENTTI	Yksi tapa määritellä komponentteja React-sovelluksessa. TypeScript-funktio, joka palauttaa React-elementtejä, jotka kuvaavat käyttöliittymän osia.

1 JOHDANTO

Logistiikan tehostaminen ja tuotekomponenttien hallinta ovat keskeisiä osia monimutkaisten tuotantoprosessien optimoinnissa. Wärtsilän sisäinen web-sovellus, DTCA-työkalu, on osoittautunut arvokkaaksi sovellukseksi tämänkaltaisessa ympäristössä. Web-sovellus tarjoaa kattavan tavan seurata ja analysoida eri tuotekomponenttien tietoja.

Tämä opinnäytetyö keskittyy tuotekomponenttien koon visualisointiin liittyvään ominaisuuteen, joka integroidaan osaksi DTCA-työkalua. Tavoitteena on kehittää laajennettava ominaisuus, joka mahdollistaa tuotekomponenttien koon visuaalisen hahmottamisen ja tarjoaa loppukäyttäjille helpon tavan ymmärtää eri komponenttien ja materiaalien kokoja. Opinnäytetyössäni käsittelen uuden ominaisuuden suunnittelua, toteutusta ja integrointia osaksi DTCA-työkalua. Tarkastelen myös erilaisia näkökohtia, kuten teknisiä vaatimuksia, käyttäjävaatimuksia ja ominaisuuden potentiaalista laajennettavuutta tulevaisuuden tarpeita varten.

Työ tarjoaa arvokasta tietoa tuotekomponenttien koon visualisoinnista, mikä auttaa teknisiä asiantuntijoita, suunnittelijoita ja logistiikan ammattilaisia ymmärtämään paremmin eri komponenttien ja materiaalien fyysisiä mittoja suhteessa toisiinsa. Tämän tiedon tarve on merkittävä erityisesti niille, jotka ovat vastuussa tuotannon, varastonhallinnan ja logistiikan suunnittelusta. Insinöörit ja suunnittelijat voivat hyödyntää tätä tietoa suunnitellessaan komponentteja ja niiden sijoittelua tehokkaammin, kun taas logistiikkapuolen ammattilaiset voivat käyttää tätä tietoa optimoidakseen eri komponenttien kuljetusjärjestelyjä ja varastointia.

2 LÄHTÖTILANNE JA VAATIMUSMÄÄRITTELY

2.1 Lähtötilanne

Wärtsilän sisäinen DTCA-työkalu on web-sovellus, joka on suunniteltu vastaamaan materiaalien ja komponenttien hallintaa koskeviin tarpeisiin. Sovellus tarjoaa käyttäjille kattavan tavan etsiä, tarkastella, verrata ja analysoida erilaisia materiaaleja ja komponentteja.

Sovellus on kerännyt merkittävästi palautetta käyttäjiltä, erityisesti logistiikan ammattilaisilta ja suunnittelijoilta, jotka ovat keskeisessä roolissa tuotantoprosessin suunnittelussa. Saadun palautteen perusteella on tunnistettu selkeä tarve kehittää sovellukseen uusi ominaisuus, joka keskittyy tuotekomponenttien ja materiaalien koon visualisointiin sen painon perusteella.

2.2 Vaatimusmäärittely

Tähän työhön on tehty vaatimusmäärittely (Taulukko 1.), joka on käyty läpi tuotteen omistajan kanssa. Vaatimusmäärittelyn avulla on rakennettu vahva perusta opinnäytetyön kehitys- ja toteutusvaiheille, varmistaen, että työ vastaa asetettuja odotuksia ja tavoitteita.

Prioriteettitaso

- P1 = Välttämätön

Taulukko 1. Vaatimusmäärittely.

Nr.	Vaatimus	Tärkeys	Kommentit
VM-001	Ominaisuuden tulee integroitua ongelmitta DTCA-web sovellukseen.	P1	Varmistaa moduulin tehokkaan yhteistyön ja tiedonvaihdon nykyisen sovelluksen kanssa.
VM-002	Ominaisuuden tulee tarjota visuaalinen esitys tuotekomponentin koosta.	P1	Käyttäjän tulee nähdä selkeästi koon visualisointi.
VM-003	Toiminto tulee hyödyntämään DTCA:n tarjoamia tietoja materiaaleista ja komponenteista.	P1	Varmistaa tiedon eheyden ja päivittyvyyden moduulin ja DTCA-sovelluksen välillä.
VM-004	Käyttöliittymän tulee olla suunniteltu helppokäyttöiseksi ja käyttäjäystävälliseksi	P1	Käyttäjän tulee pystyä navigoimaan ja käyttämään visualisointiominaisuuksia intuitiivisesti.
VM-005	Uudet visualisointityylit ja tiedonkäsittelyominaisuudet tulee voida lisätä helposti.	P1	Tulevaisuuden kehitystä ajatellen moduulin laajentaminen uusilla toiminnallisuuksilla tulee olla vaivatonta.
VM-006	Ominaisuuden kehitys tapahtuu olemassa olevilla työkaluilla, mitä aikaisemminkin käytetty DTCA-työkalun kehitykseen.	P1	Ei tuoda uusia ohjelmistokehitystyökaluja tai menetelmiä tämän ominaisuuden toteutukseen.

3 TYÖKALUT JA TOTEUTUSMENETELMÄT

Työssäni jatkokehitetään DTCA-työkalua käyttäen Visual Studio Codea kehitysympäristönä. Frontend on rakennettu React-verkkokehyksellä ja ohjelmointikielenä toimii Typescript. Backend on rakennettu Django-verkkokehyksellä ja ohjelmointikielenä toimii Python. Tietokantana käytän PostgreSQL:ää, johon haetaan yhdistetty data koskien materiaaleja ja komponentteja Wärtsilän sisäisistä tietokantavarastoista ETL:n kautta.

Tässä luvussa tutustutaan tarkemmin yllä mainittuihin työkaluihin ja toteutusmenetelmiin. Lisäksi tarkastellaan, miten valitut työkalut ja toteutusmenetelmät tukevat DTCA-työkalun web-sovelluksen jatkokehitystä ja tarjoavat ratkaisuja sen teknisiin haasteisiin. DTCA-sovelluksen tarkempi kuvaus ja tekninen arkkitehtuuri on esitetty salatussa liitteessä (Liite 1).

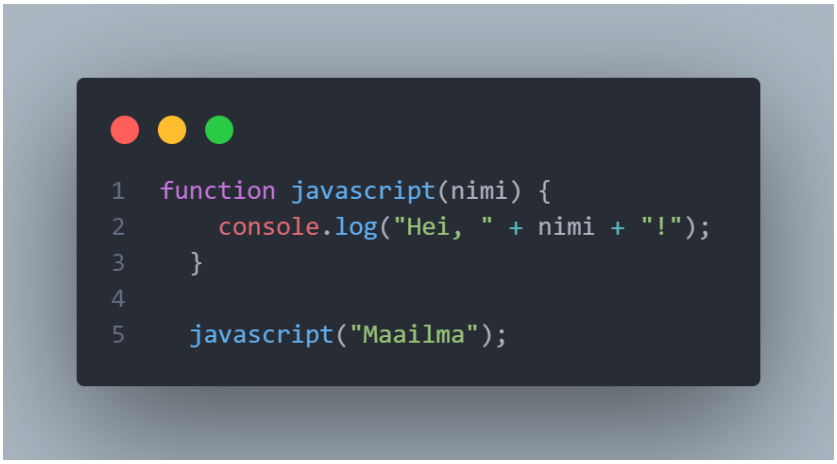
3.1 React

React on Meta:n kehittämä avoimen lähdekoodin JavaScript-kirjasto, joka on suunniteltu helpottamaan käyttöliittymien rakentamista erityisesti palvelinpohjaisiin sovelluksiin [1]. React tekee käyttöliitymien rakentamisen helpoksi pilkkomalla jokaisen sivun paloiksi. Näitä paloja kutsutaan komponenteiksi.

React-komponentti on pala koodia, joka kuvaa jotain tiettyä osaa sivusta. Jokainen komponentti on TypeScript -funktio, joka palauttaa palan koodia, joka puolestaan kuvaa osaa verkkosivusta [2]. Valmiit komponentit renderöidään sivulle hyödyntäen Reactin render metodia [3].

3.2 Typescript

Typescript on Microsoftin kehittämä ja ylläpitämä ohjelmointikieli, joka lisää valinnaisen staattisen tyypityksen JavaScriptiin [4]. Kuvien 1 ja 2 esimerkeissä nähdään yksinkertainen funktio sekä JavaScriptissä että TypeScriptissä. TypeScript-versiossa on lisätty tyypin ”string” muuttujan ”nimi” perään, jotta TypeScript voi tarkistaa, että funktiota kutsuessa annetaan oikeantyyppinen arvo.



Kuva 1. JavaScript-esimerkki.

Kuva 2. TypeScript-esimerkki.

TypeScript-koodi käännetään JavaScriptiksi, joten molemmat esimerkit tuottavat saman tuloksen suorituksen aikana. Staattisen tyypityksen myötä TypeScript helpottaa ja turvaa suuren sovelluksen kehitystä, ettei muuttujille ole mahdollista antaa muiden kun asetetun tyypin arvoa.

3.3 Python

Python on korkean tason ohjelmointikieli, joka tunnetaan selkeästä syntaksistaan sekä monipuolisista käyttömahdollisuuksista. Sen dynaaminen tyyppijärjestelmä ja laaja standardikirjasto tekevät siitä suositun valinnan web-kehityksessä. Pythonin käytettävyys ja yhteensopivuus muiden teknologioiden kanssa ovat tehneet siitä monipuolisen ja laajasti hyväksytyn ohjelmointikielen. [5.]

Koska Python on yksi suosituimmista ohjelmointikielistä, sille on myös kehitetty runsaasti suosittuja kirjastoja, ohjelmistokehyksiä ja rajapintoja, joiden avulla voidaan luoda paljon erilaisia sovelluksia esim. numeerisen laskennan, grafiikan ja web-kehityksen alueilla [6].

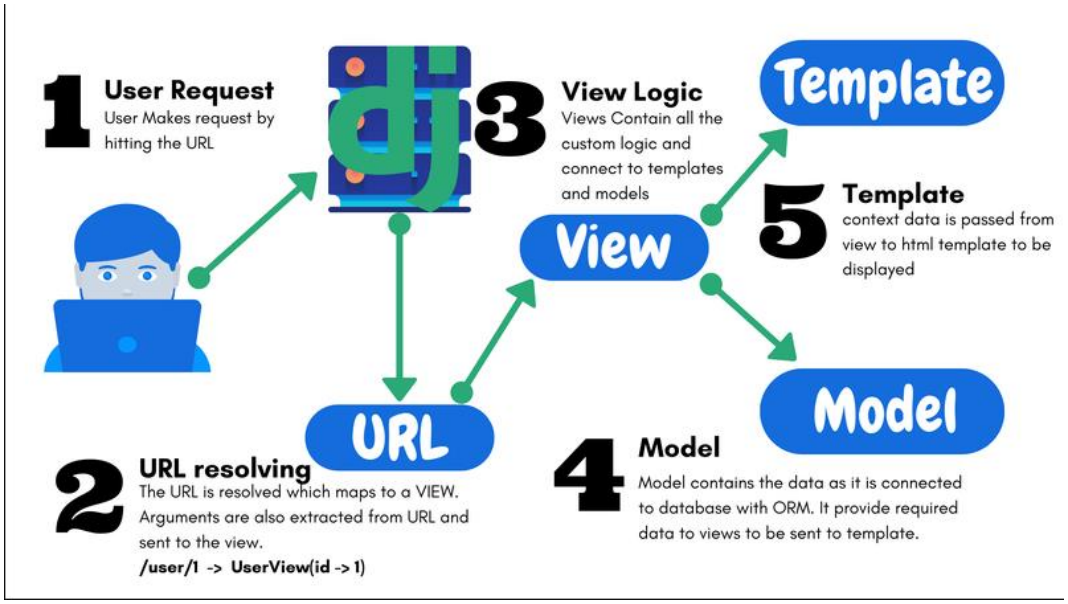
3.4 Django

Django on avoimen lähdekoodin Python-pohjainen verkkokehys, joka noudattaa MVT-mallia. Se on suunniteltu helpottamaan tehokkaiden ja skaalautuvien websovellusten rakentamista [7]. Django tarjoaa valmiit ratkaisut moniin haasteisiin, kuten tietokantayhteydet, käyttäjäautentikaation ja URL-reitityksen.

MVT-mallin ansiosta Django mahdollistaa nopean kehityksen ja keskustelun muiden teknologioiden kuten ETL:n, tietokannan sekä Front-end:in välillä. Djangon MVT-arkkitehtuuri ja dynaaminen tyypitys mahdollistavat, etteivät Djangolla luodun mallin tiedot eroa tietokannasta saadusta datasta. Esitetty data web-sovelluksessa on juuri kuten se on määritelty Djangon datamallin luokassa. Kuvan 3 esimerkin avulla nähdään Djangolla luotu datamalli komponentista ja materiaalista. Nämä mallit sisältävät vain niille luodut muuttujan arvot, joiden avulla tietokannasta haetaan oikea tieto ORM:n avulla [8]. Tämä malli tallennetaan Djangon URL-reititykseen ja luodaan sovelluksen sisäinen rajapinta, jonka kautta saadaan luotu malli näytettyä web-sovelluksessa.

```
1
2  from django.db import models
3
4  #Luodaan mallit
5  class Komponentti(models.Model):
6      nimi = models.CharField(primary_key = True, max_length = 255)
7      paino = models.IntegerField(max_length=255)
8      materiaali = models.ForeignKey(Materiaali, on_delete=models.DO_NOTHING)
9
10     def __str__(self):
11         return self.Komponentti
12
13 class Materiaali(models.Model):
14     nimi = models.CharField(primary_key = True, max_length = 255)
15     komponentti = models.ForeignKey(Komponentti, on_delete = models.DO_NOTHING)
16
17     def __str__(self):
18         return self.Materiaali
```

Kuva 3. Esimerkki Django-datamalleista.



Kuva 4. Django MVT-malli [9].

3.5 Jest

Jest on JavaScript-testikehys, joka on suunniteltu erityisesti React-sovellusten testaamiseen, mutta sitä voidaan käyttää myös muiden JavaScript- ja TypeScript-projektien testaamiseen. Jest tarjoaa yksinkertaisen ja tehokkaan tavan kirjoittaa ja suorittaa testejä, mikä auttaa varmistamaan koodin laadun ja vakaan suorituskyvyn kehityksen aikana. [10.]

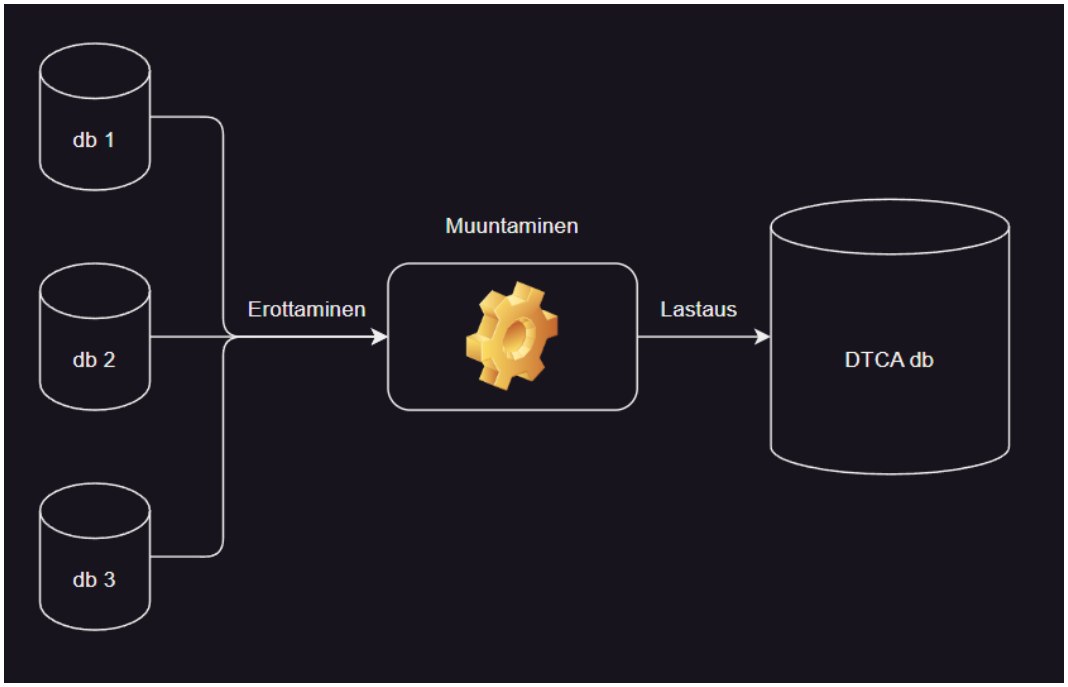
Jest tukee erilaisia testityyppejä, kuten yksikkötestejä (unit test) sekä toiminnallisia testejä. Se tarjoaa monipuolisia ominaisuuksia, kuten automaattisen testien suorituksen, testien rinnakkaisen suorituksen, mock- ja spy-ominaisuudet ja kattavat testiraportit. Työssä hyödynnetään Jest-yksikkötestejä uudessa React-komponentissa. Testit varmistavat komponentin toimivuuden osana DTCA-web-sovellusta.

3.6 ETL

ETL-prosessi (Extract, Transform, Load) on keskeinen osa tietojen hallintaa ja analysointia, jossa dataa kerätään, muunnetaan ja siirretään eri lähteistä yhtenäiseen tietokantavarastoon. ETL:n toimintaperiaatteet perustuvat kolmeen päävaiheeseen:

1. **Erottaminen (Extract):** Tässä vaiheessa tiedot poimitaan eri lähteistä. Tässä tapauksessa eri tietokantavarastoista.
2. **Muuntaminen (Transform):** Saatua dataa käsitellään ja muunnellaan, jotta se vastaa DTCA-tietokannan rakennetta ja vaatimuksia. Esimerkiksi dataa voidaan muokata tai muuttaa sen formaattia DTCA:n tietokantaan sopivammaksi.
3. **Lastaus (Load):** Lopuksi muunnettu data ”lastataan” tietokantaan. Tässä vaiheessa varmistetaan, että ETL:n muunnettu data on eheää ja täyttää kohdetietokannan datamallien rakenteen ja eheyssäännöt. [11.]

Suuressa organisaatiossa data on jaettu useisiin sisäisiin tietovarastoihin, joten ETL:n käyttö on välttämätöntä. ETL:n avulla voidaan kerätä tarvittava data komponenteista ja materiaaleista DTCA-sovelluksen käyttämään omaan tietokantaan Djangolla luotujen datamallien kaltaisiksi, sekä hyödyntää näitä tietoja sovelluksen Front-end-osassa näyttämään oikean tiedot materiaalista. Kuvassa 5 on esitetty ETL-prosessi.



Kuva 5. ETL-toimintaperiaate.

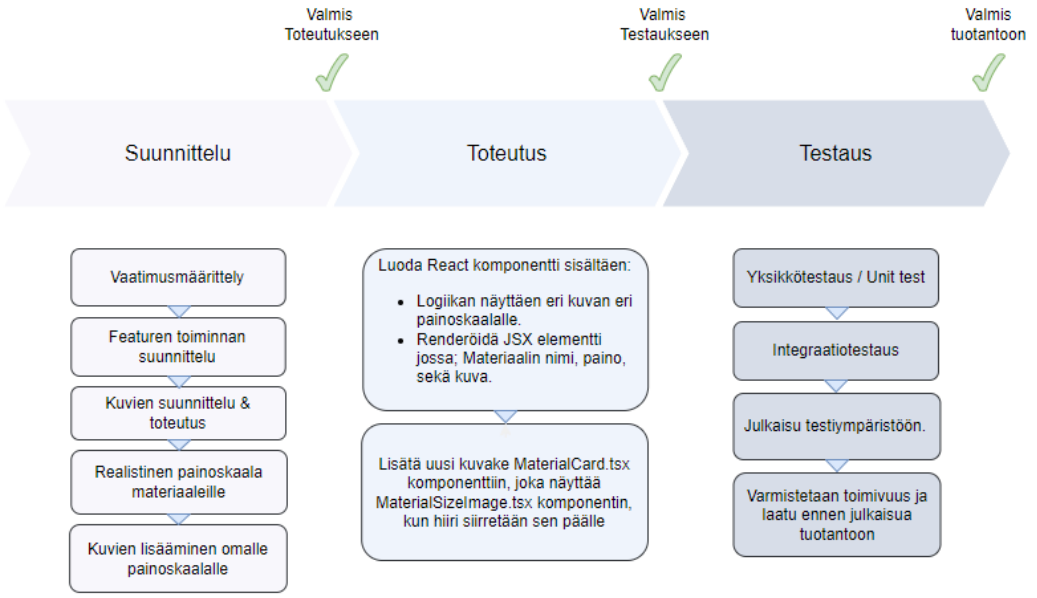
3.7 PostgreSQL

PostgreSQL on avoimen lähdekoodin relaatiotietokantajärjestelmä. Se on suunniteltu tehokkaaseen ja luotettavaan tietokantahallintaan. PostgreSQL tarjoaa monipuoliset ominaisuudet, jotka sopivat sekä pienille että suurille tietokannoille ja sovelluksille. [12.]

PostgreSQL tukee ANSI SQL -standardia ja tarjoaa laajan valikoiman edistyneitä kyselyominaisuuksia, kuten monimutkaisia liitoksia, alikyselyjä, ikkunafunktioita ja tilastollisia toimintoja. Se on myös optimoitu suorituskykyä varten ja sisältää useita optimointimenetelmiä, kuten indeksit ja välimuistituksen. [13.]

4 SUUNNITTELU

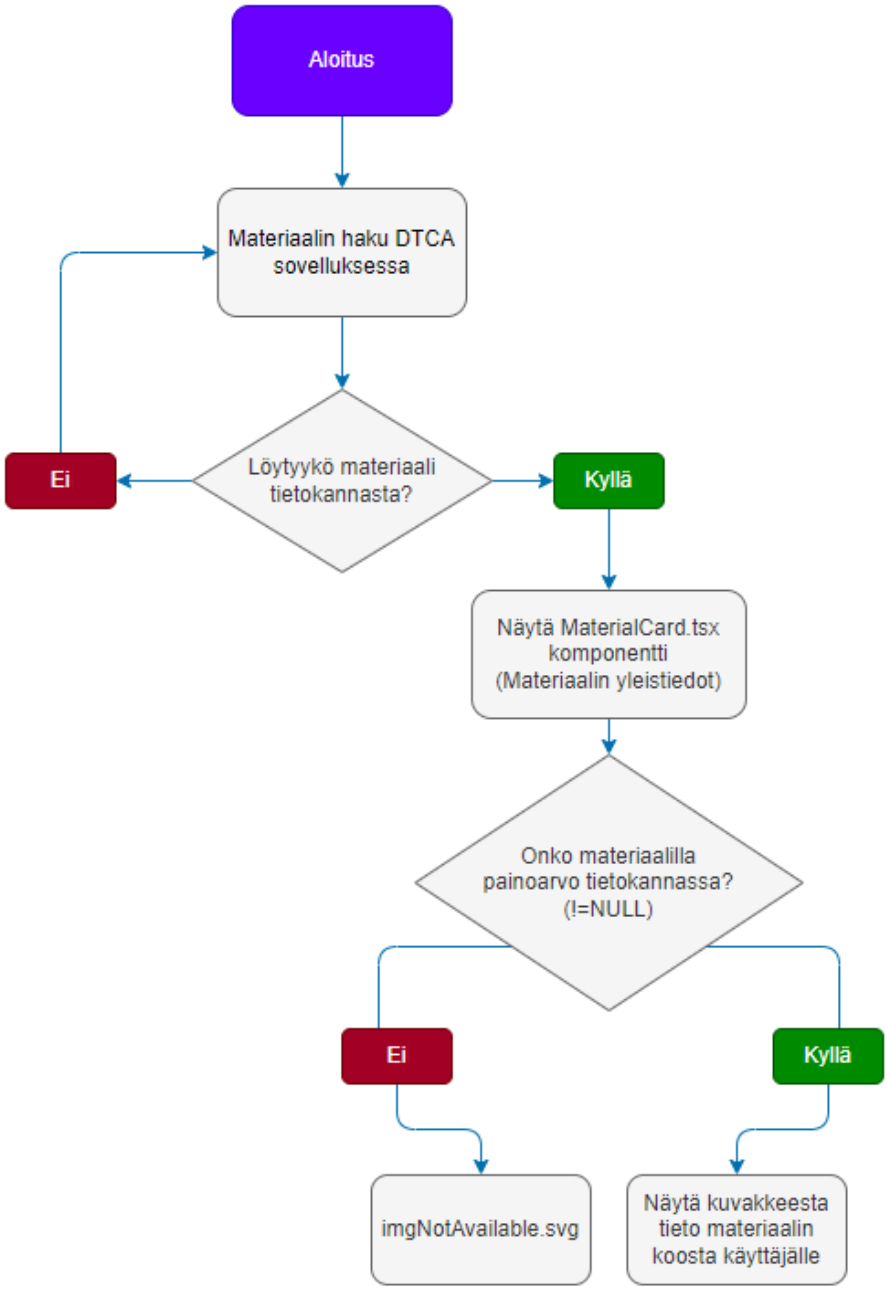
Työ on jaettu kolmeen osa-alueeseen: Suunnittelu, toteutus ja testaus. Suunnitteluvaiheessa määritellään uuden ominaisuuden tavoitteet, vaatimukset ja toiminnallisuudet. Kuvassa 6 nähdään työn eri vaiheet ja mitä ne sisältävät.



Kuva 6. Työvaiheet.

4.1 Ominaisuuden suunnittelu

Työn suunnittelu alkoi tuotteen omistajan vaatimusten pohdinnalla sekä työn tarvittavien ominaisuuksien päättämisellä (Taulukko 1.) Näiden vaatimusten pohjalta syntyi suunnitelma, jonka avulla siirryttiin toteuttamaan toimivaa ominaisuutta sovellukseen. Suunniteltujen ja toteutettujen kuvien tulisi antaa loppukäyttäjälle tieto materiaalin koosta sen painon perusteella. Lisäksi tiedon tulisi oltava helposti saatavilla sovelluksesta. Kuvassa 7 on suunnitelman pohjalta syntynyt ominaisuuden toimintaa kuvaava vuokaavio.



Kuva 7. Ominaisuuden toimintaa esittävä vuokaavio.

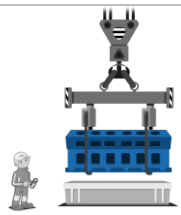
4.2 Kuvien suunnittelu ja toteutus

Kuvien suunnittelun yhteydessä olen tehnyt tiivistä yhteistyötä Wärtsilän suunnittelijoiden kanssa sen varmistamiseksi, että valmiit tuotokset välittävät selkeästi tuotekomponenttien koon visualisoinnin. Suunnittelutyössä otettiin huomioon käyttäjien tarpeet sekä DTCA-sovelluksen helppokäyttöisyys. Kuvat on toteuttanut Pekka Puhakka, Wärtsilän Industrial Designista

Kuvat sijoitetaan painoskaalalle, joka esittää tiettyä painorajaa. Sovelluksen loppukäyttäjät voivat hahmottaa komponenttien koot suhteessa toisiinsa. Skaala on suunniteltu ja hyväksytty tuotteen omistajan kanssa.

Taulukko 2. Painoskaala ja valmiit kuvat.

Paino (kg)	Kuva	Kuvaus
< 1		less_than_one.png
1 - 5		one_to_five.png
5 - 10		five_to_ten.png
10 - 25		ten_to_twentyfive.png

25 - 50		twentyfive_to_fifty.png
50 - 200		fifty_to_twohundred.png
200 - 500		twohundred_fivehundred.png
500 – 2,500		fivehundred_to_twentyfivehun-dred.png
2,500 - 10,000		Twentyfivehundred_to_tent-housand.png
>10,000		over_tenthousand.png
NULL || 0		imgUnavailable.png

5 TOTEUTUS

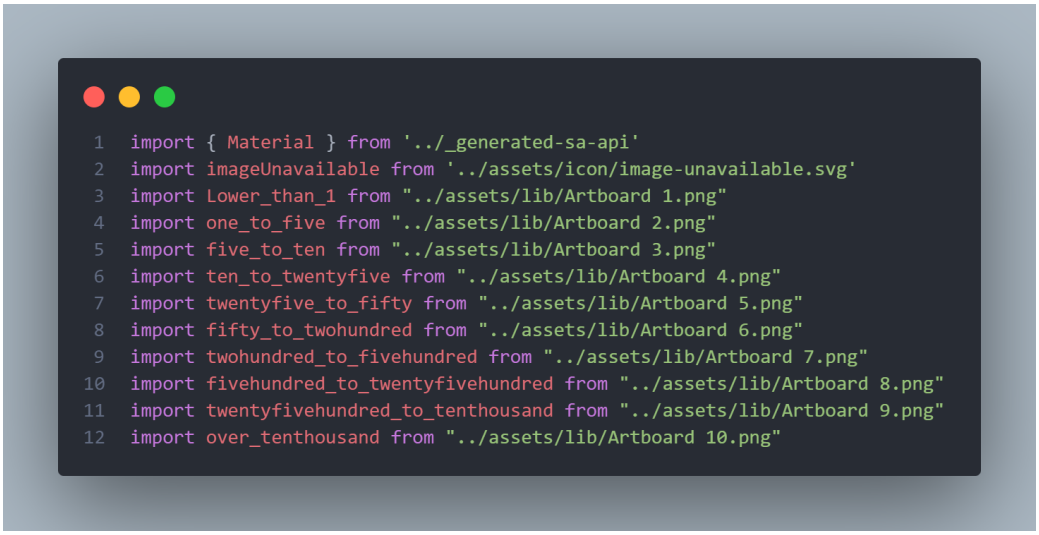
Toteutusvaiheessa keskityin ominaisuuden rakentamiseen suunnittelun pohjalta. Tämä vaihe sisältää keskeisiä askelia, joista ensimmäinen on ohjelmointi. Tämä vaihe vaatii perehtymistä DTCA:ssa käytettäviin teknologioihin, kuten React:iin frontendin ja Djangoon backendin osalta, sekä niiden käyttöön sovelluksen kehittämisessä.

Integraatiovaiheessa pyrin varmistamaan, että kehittämäni ominaisuus toimii yhdessä muun sovelluksen kanssa. Integraatio sisältää ominaisuuden liittämisen olemassa olevaan koodipohjaan ja muutosten tekemisen muissa osissa sovellusta sen varmistamiseksi, että integraatio on onnistunut.

5.1 MaterialSizeImage.tsx komponentti

Uuden ominaisuuden toteuttamisessa keskeinen osa on luoda uusi React-komponentti, joka käsittelee logiikan oikean kuvan liittämisestä oikeaan painoskaalaan sekä renderöi JSX-elementin sisältäen materiaalin nimen, painon ja kuvan. Tämä komponentti vastaa tuotekomponenttien koon visualisoinnista ja käyttäjän näkemästä selkeästä kuvasta sen mukaan, mihin painoskaalaan kyseinen komponentti kuuluu.

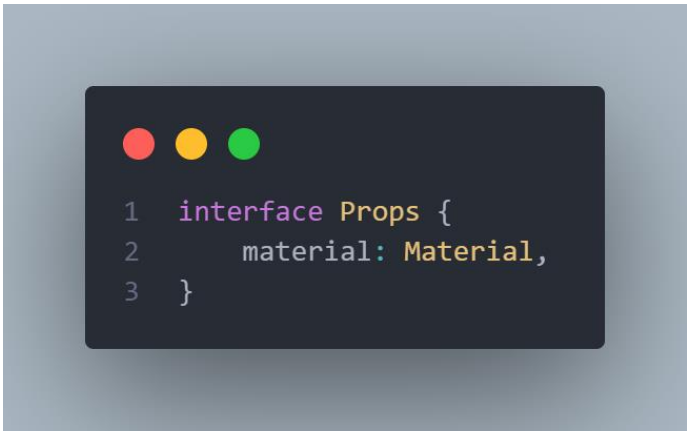
Ensimmäinen asia uuden komponentin luomisessa on importoida, eli tuoda komponenttiin tarvittava tieto muista tiedostoista [14]. Kuvassa 8 tuodaan materiaalin tiedot ja käytettävät kuvat alikansioista komponenttiin.



```
import { Material } from '../_generated-sa-api'
import imageUnavailable from '../assets/icon/image-unavailable.svg'
import Lower_than_1 from "../assets/lib/Artboard 1.png"
import one_to_five from "../assets/lib/Artboard 2.png"
import five_to_ten from "../assets/lib/Artboard 3.png"
import ten_to_twentyfive from "../assets/lib/Artboard 4.png"
import twentyfive_to_fifty from "../assets/lib/Artboard 5.png"
import fifty_to_twohundred from "../assets/lib/Artboard 6.png"
import twohundred_to_fivehundred from "../assets/lib/Artboard 7.png"
import fivehundred_to_twentyfivehundred from "../assets/lib/Artboard 8.png"
import twentyfivehundred_to_tenthousand from "../assets/lib/Artboard 9.png"
import over_tenthousand from "../assets/lib/Artboard 10.png"
```

Kuva 8. MaterialSizeImage.tsx imports.

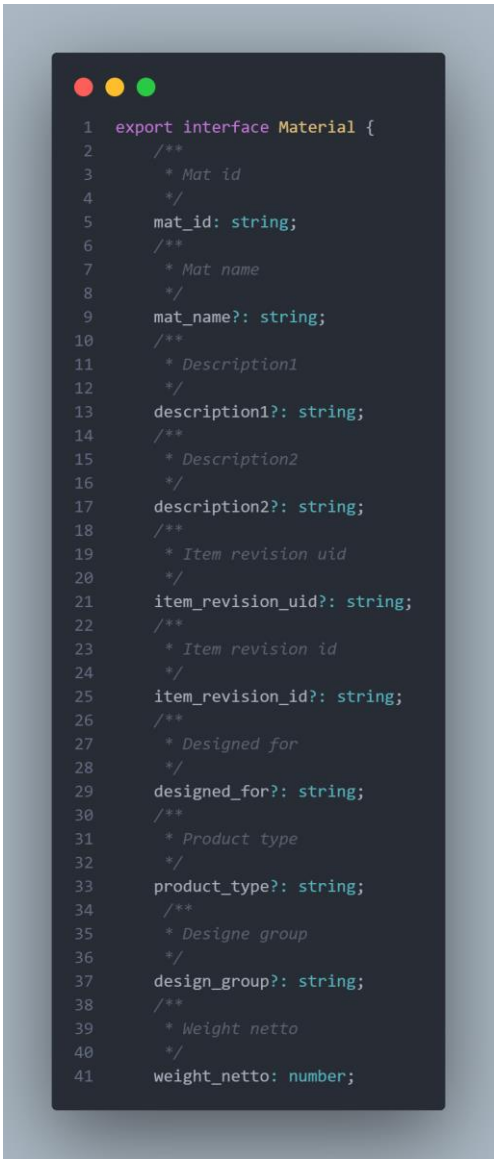
Jotta saadaan materiaalin paino, on tuotava sen rakenne TypeScriptin Interface ominaisuudella kuvan 9 mukaisesti. Interface määrittelee, että komponentti odottaa yhden propsin nimeltään ”material”, joka on tyyppiä ”Material”.



```
interface Props {
    material: Material,
}
```

Kuva 9. Interface Props, material.

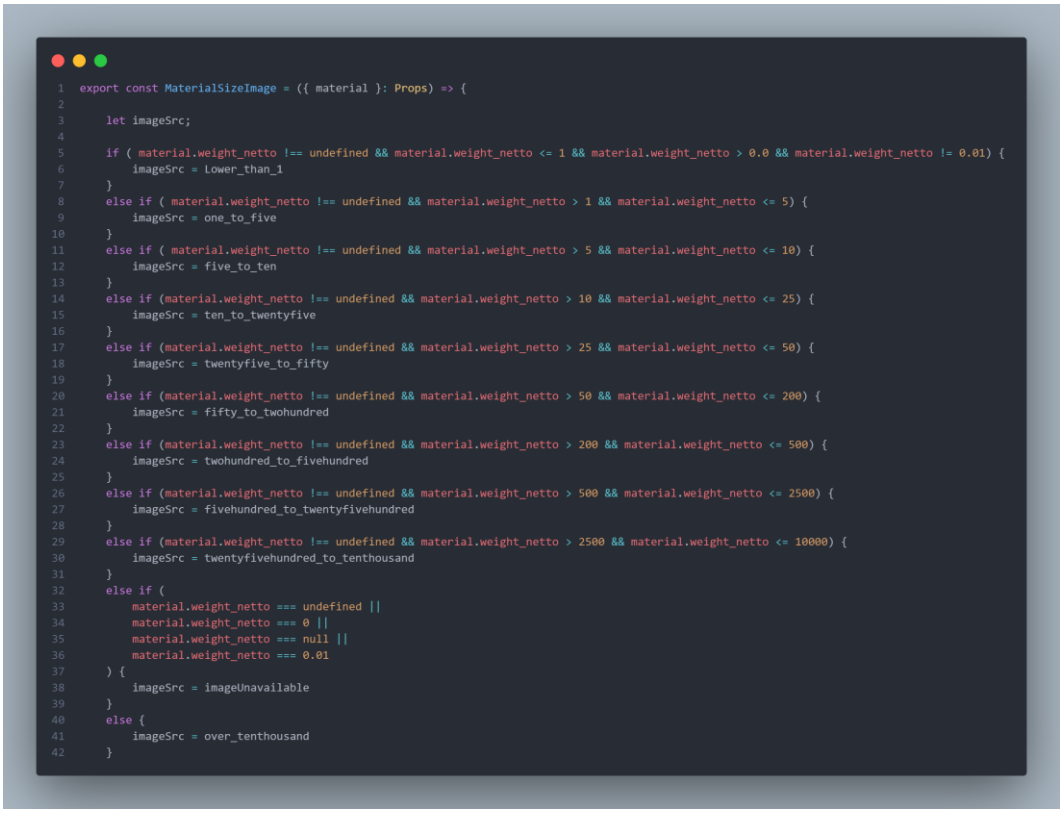
Tiettyjen komponenttien välillä tapahtuva viestintä toteutetaan Reactissa usein propsien avulla. Propsit ovat tapa siirtää tietoa vanhemmilta (parent) komponenteilta lapsikomponenteille. Kun määritellään komponentin propsit, määritellään käytännössä, millaisia tietoja komponentti odottaa ja käsittelee [15]. Tässä tapauksessa käsitellään Material-objektia, jolla on kuvan 10 kaltaisia atribuutteja mm. paino (weight_netto).



```
export interface Material {
    /**
     * Mat id
     */
    mat_id: string;
    /**
     * Mat name
     */
    mat_name?: string;
    /**
     * Description1
     */
    description1?: string;
    /**
     * Description2
     */
    description2?: string;
    /**
     * Item revision uid
     */
    item_revision_uid?: string;
    /**
     * Item revision id
     */
    item_revision_id?: string;
    /**
     * Designed for
     */
    designed_for?: string;
    /**
     * Product type
     */
    product_type?: string;
     /**
     * Designe group
     */
    design_group?: string;
    /**
     * Weight netto
     */
    weight_netto: number;
```

Kuva 10. Material-objekti.

Kuvan 11 esimerkissä toteutetaan logiikka kuvien liittämisestä eri painoskaaloihin. Komponentti hyödyntää ehtolauseita valitsemaan oikean kuvan kunkin tuotekomponentin koon perustuen Taulukko 2 määritykseen. MaterialSizeImage-komponentti on rakennettu käyttäen export const-rakennetta, mikä tarkoittaa, että tätä komponenttia voidaan käyttää yhdistellen sovelluksen muissa osissa [16]. Tämä mahdollistaa modulaarisen lähestymistavan sovelluksen rakentamiseen, jossa eri osat ovat erillisissä tiedostoissa ja niitä voidaan käyttää ja yhdistellä tarpeen mukaan.



```
export const MaterialSizeImage = ({ material }: Props) => {

    let imageSrc;

    if ( material.weight_netto !== undefined && material.weight_netto <= 1 && material.weight_netto > 0.0 && material.weight_netto != 0.01) {
        imageSrc = Lower_than_1
    }
    else if ( material.weight_netto !== undefined && material.weight_netto > 1 && material.weight_netto <= 5) {
        imageSrc = one_to_five
    }
    else if ( material.weight_netto !== undefined && material.weight_netto > 5 && material.weight_netto <= 10) {
        imageSrc = five_to_ten
    }
    else if (material.weight_netto !== undefined && material.weight_netto > 10 && material.weight_netto <= 25) {
        imageSrc = ten_to_twentyfive
    }
    else if (material.weight_netto !== undefined && material.weight_netto > 25 && material.weight_netto <= 50) {
        imageSrc = twentyfive_to_fifty
    }
    else if (material.weight_netto !== undefined && material.weight_netto > 50 && material.weight_netto <= 200) {
        imageSrc = fifty_to_twohundred
    }
    else if (material.weight_netto !== undefined && material.weight_netto > 200 && material.weight_netto <= 500) {
        imageSrc = twohundred_to_fivehundred
    }
    else if (material.weight_netto !== undefined && material.weight_netto > 500 && material.weight_netto <= 2500) {
        imageSrc = fivehundred_to_twentyfivehundred
    }
    else if (material.weight_netto !== undefined && material.weight_netto > 2500 && material.weight_netto <= 10000) {
        imageSrc = twentyfivehundred_to_tenthousand
    }
    else if (
        material.weight_netto === undefined ||
        material.weight_netto === 0 ||
        material.weight_netto === null ||
        material.weight_netto === 0.01
    ) {
        imageSrc = imageUnavailable
    }
    else {
        imageSrc = over_tenthousand
    }
```

Kuva 11. Ehtolauselogiikka kuville materiaalin painon perusteella.

MaterialSizeImage-komponentille annetaan material props, jotta materiaaliobjektin eri arvot ovat saatavilla. Weight_netto muuttuja-arvoa voidaan käyttää ehtolauseissa. **let imageSrc** on muuttuva arvo, joka asetetaan oikean kuvan arvoksi tietyllä painoskaalalla.

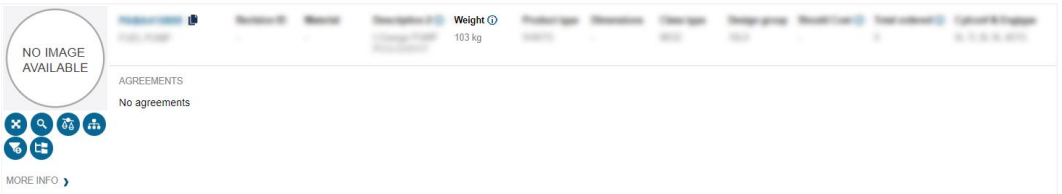
Lopuksi MaterialSizeImage.tsx komponentissa renderöidään Kuvan 12 mukainen JSX-elementti, joka palauttaa tiedon materiaalin nimestä, painosta ja oikeasta kuvasta painoskaalalla [17].

Kuva 12. MaterialSizeImage komponentin JSX-elementti.

JSX-elementin sisällä oleva HTML-tagi **<img>** saa imageSrc-muuttujan src-attribuutin paikalle näyttäen oikean kuvan aikaisemman ehtolauselogiikan myötä. HTML-tageille on annettu className-arvot, jotta näitä kenttiä voidaan myöhemmin muokata CSS:n avulla. Tämä käytäntö mahdollistaa myös sen, että React-komponentit voivat dynaamisesti määrittää luokkia tai luokkien yhdistelmiä riippuen niiden tilasta tai muista tekijöistä. Näin ollen className-attribuuttien käyttö on yleinen tapa hallita elementtien ulkoasua ja tyyliä React-sovelluksissa [18].

5.2 Integraatiovaihe

Integraatiovaiheessa on tarkoituksena lisätä kuvan 13 mukaiseen MaterialCard-komponenttiin uusi kuvake (<i> HTML-tagi). Tämä kuvake tulee sisältämään logiikan, joka renderöi sivulle MaterialSizeImage-komponentin.



Kuva 13. MaterialCard-komponentti DTCA-työkalu sovelluksessa.

5.2.1 ShowImageDetails ja MaterialBasics. Funktionaaliset komponentit MaterialCard-komponentissa

MaterialCard-komponenttiin (liite 2.) rakennettiin uusi funktionaalinen komponentti ”ShowImageDetails”, joka on vastuussa JSX-elementin palauttamisesta sisältäen MaterialSizeImage komponentin. Kuvassa 14 ShowImageDetails funktionaalinen komponentti.

```
1  const ShowImageDetails = (props: {material: Material}) => {
2
3    return (
4          <div className="show-image-wrapper">
5            <div className="show-image-content">
6            <MaterialSizeImage material={props.material}></MaterialSizeImage>
7            </div>
8          </div>
9    )
10 }
```

Kuva 14. ShowImageDetails funktionaalinen komponentti.

Jotta tämä komponentti voidaan renderöidä sivulle dynaamisesti hiiren viemisestä kuvakkeen päälle tapahtuman yhteydessä, täytyi sille luoda logiikka. Kuvan 15 esimerkissä on käytetty React useState-koukkua (hook) määrittämään tilamuuttuja showImageDetails-komponentille. [19.]

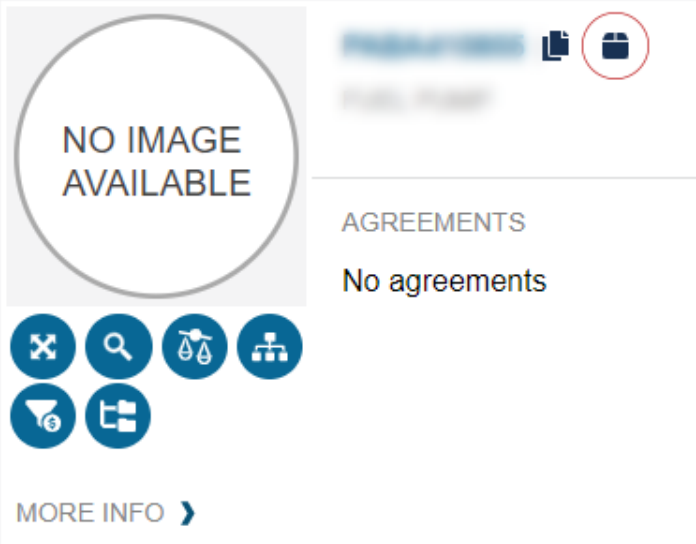
```
export const MaterialBasics = (props: { material: Material; className?: string}) => {

  const [showImageDetails, setShowImageDetails] = useState(false);

  const handleMouseEnter = () => {
    setShowImageDetails(true);
  }

  const handleMouseLeave = () => {
    setShowImageDetails(false);
  }
```

Kuva 15. showImageDetails useState hook.

HandleMouseEnter ja **handleMouseLeave** ovat määritettyjä tapahtumankäsittelijöitä, jotka vastaavat hiiren tulo- ja poistumistapahtumiin komponentin kuvakeen yllä. Tämä arvo voi olla joko epätosi (false) tai tosi (true). Seuraavaksi luotiin uusi kuvake MaterialBasics funktionaaliseen komponenttiin kuvan 16 mukaisesti sekä liitettiin integraation vaiheet yhteen. Kuvassa 17 nähdään valmis kuvake materialCard-komponentissa.

```jsx
return (
  <div className={classnames('material-basics', props.className)}>
    <div className="material-id">
      <div style={{ display: 'flex', flexDirection: 'row' }}>
        <a href={standardReportUrl(materialId)} target="_blank" rel="noopener noreferrer">
          {materialId}
        </a>
        <i
          className="icon-button"
          onClick={() => copyHandler()}
          data-tip={`iconlegend-${UI_STRINGS.copy}`}
        >
          {renderIcon('copy')}
        </i>
        <i
          className="icon-button"
          onMouseOver={handleMouseEnter}
          onMouseLeave={handleMouseLeave}
          data-tip={`iconlegend-${UI_STRINGS.tooltip_material_image}`}
        >
          {renderIcon('material-image')}
        </i>
        {showImageDetails && <ShowImageDetails material={material}/>}
      </div>
    </div>
    <div className="material-name">{material.mat_name}</div>
  </div>
)
```

Kuva 16. MaterialBasics-komponentti JSX-osio MaterialCard-komponentin sisällä.



Kuva 17. Uusi kuvake MaterialCard-komponentissa.

```
<i
  className="icon-button"
  onMouseOver={handleMouseEnter}
  onMouseLeave={handleMouseLeave}
  data-tip={`iconlegend-${UI_STRINGS.tooltip_material_image}`}
>
  {renderIcon('material-image')}
</i>
{showImageDetails && <ShowImageDetails material={material}/>}
```

Kuva 18. MaterialBasics Icon onMouseOver ja onMouseLeave sekä ehtolauseke.

Kuvan 15 esimerkissä useState on alustettu false-arvoksi. Kuvakkeessa hyödynnetään onMouseOver-tapahtumaa (event) muuttamaan useState showImageDetails-arvon todeksi. Kuvan 18 esimerkissä viimeisellä rivillä lisätty ehtolauseke ”**{showImageDetails && <ShowImageDetails material = {material} />}**” renderöi ShowImageDetails-komponentin vain silloin, kuin useState showImageDetails-tila on tosi (true). Tämä mahdollistaa komponentin näyttämisen vain, kun hiiri on kuvakkeen päällä antaen sille oikeanlaisen käyttäytymisen, jota ominaisuudelta haettiin.

5.2.2 Viimeistely

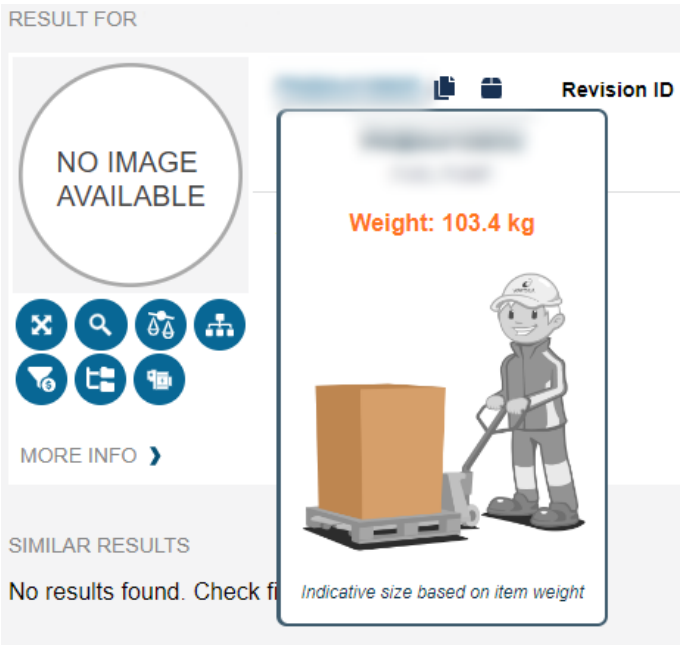
Ominaisuuden viimeistelyyn liittyy MaterialSizeImage ja ShowImageDetails komponenttien HTML-attribuuttien tyylimuutoksia, jotta saadaan haluttu lopputulos komponentin renderöimisestä sivulle. Kuvien 19 ja 20 esimerkkien tyylimuutosten avulla luotiin ikkuna, joka vie vähän tilaa ja näyttää loppukäyttäjille nopeasti sekä vaivattomasti tarvittavan tiedon materiaalin koon visualisoinnista hiiren viemisestä kuvakkeen päälle.

```css
.material_size_disclaimer {
  font-size: 12px;
  font-weight: 400;
  font-style: italic;
  color:#0F334A;
}

.material_title {
  color: #3F5C6E;
  margin-top: 5px;
  font-size: 18px;
  margin-bottom: 0;
}

.material_description {
  font-size: 12px;
  font-weight: 400;
  margin: 0;
  color:#6F808D;
}

.material_size_attributes {
  font-size: 16px;
  font-weight: 600;
  color:#FF7321;
}
```

Kuva 19. MaterialSizeImage HTML-tagi CSS-tyylit.

```
.show-image-wrapper {
  position: absolute;
  width: 225px;
  top: 140px;
  background-color: #3F5C6E;
  text-align: center;
  border-radius: 7px;
  z-index: 100;
}

.show-image-content {
  display: inline-block;
  width: 220px;
  margin: 2px;
  background-color: white;
  border-radius: 5px;
}
```

Kuva 20. ShowImageDetails HTML-tagi CSS-tyylit.



Kuva 21. Ominaisuuden valmis näkymä DTCA-sovelluksessa.

6 TESTAUS

Testausvaiheessa varmistetaan, että **MaterialSizeImage**-komponentti toimii odotetusti eri tilanteissa ja että se noudattaa määriteltyjä toiminnallisuuksia. Testauksen kannalta olen kirjoittanut kolme yksikkötestiä Jest-testikehyksellä.

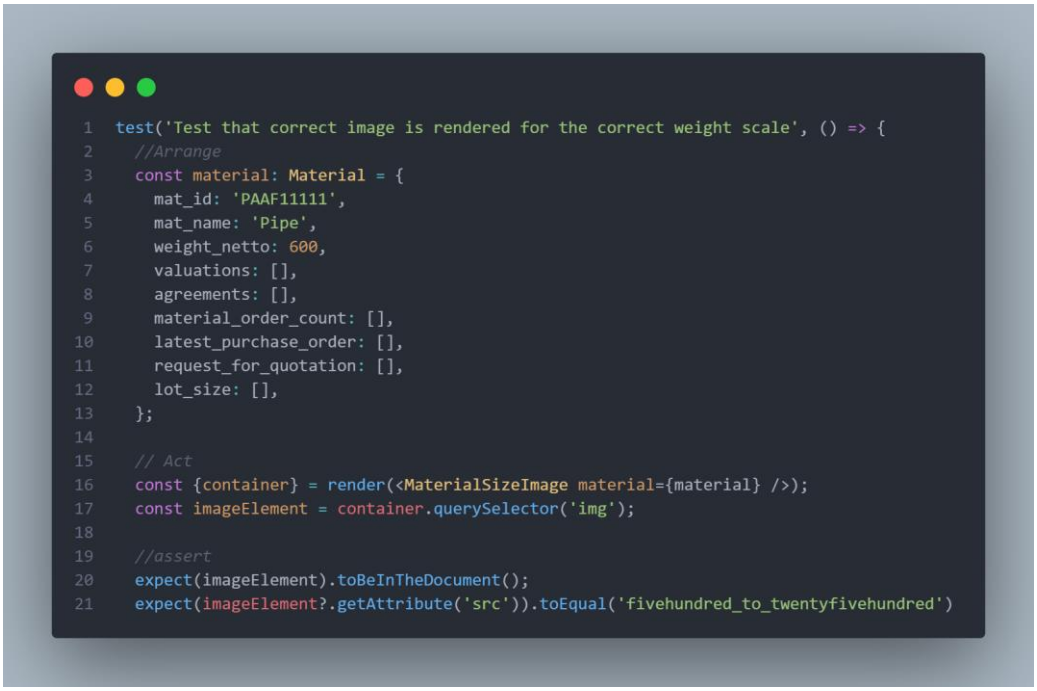
Ensimmäisessä testissä tarkastetaan, että komponentti renderöi materiaalin tiedot oikein kuvan 22 mukaisesti. Toisessa testissä varmistetaan (kuva 23), että kuva renderöidään komponenttiin oikein. Kolmas testi (kuva 24) varmistaa, että oikea kuva renderöidään oikealle skaalalle perustuen materiaalin painotietoihin taulukon 2 mukaisesti.

```
test('Test to include all material info in MaterialSizeImage component', () => {
  //Arrange
  const material: Material = {
    mat_id: 'PAAF11111',
    mat_name: 'Pipe',
    weight_netto: 500,
    valuations: [],
    agreements: [],
    material_order_count: [],
    latest_purchase_order: [],
    request_for_quotation: [],
    lot_size: [],
  };

  // Act
  render(<MaterialSizeImage material={material}/>)
  const mat_id = screen.getByText('PAAF11111');
  const mat_name = screen.getByText('Pipe');
  const weight = screen.getByText('Weight: 500 kg');

  //Assert
  expect(mat_id).toBeInTheDocument();
  expect(mat_name).toBeInTheDocument();
  expect(weight).toBeInTheDocument();
})
```

Kuva 22. Unit test 1. Renderöi materiaalin tiedot.

```
test('Test that image is rendered inside MaterialSizeImage component', () => {
  //Arrange
  const material: Material = {
    mat_id: 'PAAF11111',
    mat_name: 'Pipe',
    weight_netto: 600,
    valuations: [],
    agreements: [],
    material_order_count: [],
    latest_purchase_order: [],
    request_for_quotation: [],
    lot_size: [],
  };
  // Act
  const { getByAltText } = render(<MaterialSizeImage material={material}/>)
  expect(getByAltText('material_size_image')).toBeInTheDocument();
})
```

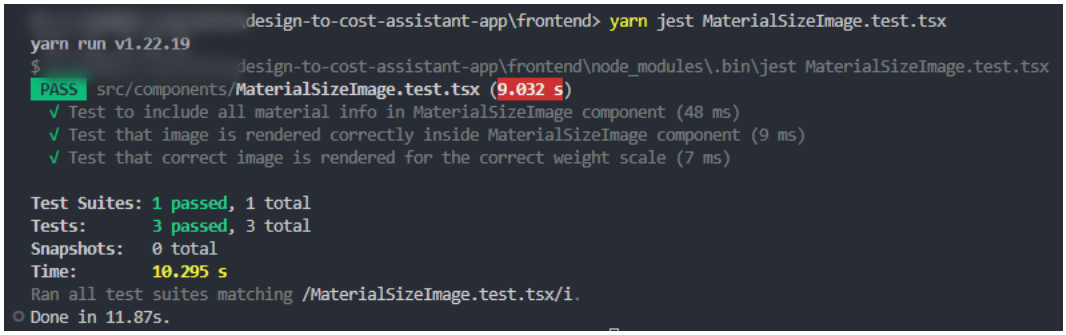
Kuva 23. Unit test 2. Varmistetaan kuvan renderöinti.



```
test('Test that correct image is rendered for the correct weight scale', () => {
  //Arrange
  const material: Material = {
    mat_id: 'PAAF11111',
    mat_name: 'Pipe',
    weight_netto: 600,
    valuations: [],
    agreements: [],
    material_order_count: [],
    latest_purchase_order: [],
    request_for_quotation: [],
    lot_size: [],
  };

  // Act
  const {container} = render(<MaterialSizeImage material={material} />);
  const imageElement = container.querySelector('img');

  //assert
  expect(imageElement).toBeInTheDocument();
  expect(imageElement?.getAttribute('src')).toEqual('fivehundred_to_twentyfivehundred')
```

Kuva 24. Unit test 3. Oikea kuva oikeaan painoskaalaan.

Kuvassa 25 nähdään, että luodut testit menivät läpi ja että ykikkötestaus onnistui. MaterialSizeImage-komponentti toimii odotetusti ja noudattaa määriteltyjä toiminnallisuuksia. Testien läpäisy osoittaa, että komponentti renderöi materiaalin tiedot oikein, kuva renderöidään oikein komponenttiin ja että oikea kuva renderöidään oikealle skaalalle perustuen materiaalin tietoihin sekä komponentin ehtolauselogiikkaan.



Kuva 25. Onnistuneet yksikkötestaukset.

Testauksen vaiheessa uusi ominaisuus julkaistaan testiympäristöön valitulle käyttäjämäärälle testattavaksi. Tämä mahdollistaa käyttäjäpalautteen keräämisen ennen lopullista julkaisua tuotantoon ja auttaa havaitsemaan mahdolliset ongelmat ja parannusmahdollisuudet ennen laajempaa käyttöönottoa. [20.]

7 JATKOKEHITYS

Jatkokehityksenä on suunnitteilla luoda Wärtsilän sisäiseen käyttöön julkinen rajapinta (API), joka liittyy toteuttamaani ominaisuuteen. Tämä API tarjoaisi ohjelmistosuunnittelijoille, kehittäjille sekä suunnittelijoille tavan kommunikoida materiaalin koon visualisointiin liittyvien tietojen kanssa, mikä mahdollistaisi niiden käytön erilaisissa sovelluksissa ja järjestelmissä. Sisäisen julkisen rajapinnan suunnittelu ja toteutus auttavat edistämään yhteistyötä eri tiimien välillä, parantamaan tuottavuutta ja mahdollistamaan ratkaisujen kehittämisen Wärtsilän sisäisessä ekosysteemissä. Lisäksi se tarjoaisi vakaan perustan tuleville kehityksille ja laajennuksille.

Kun materiaaleille saadaan lisätietoja, kuten tarkkoja mittasuhteita, suunnitellaan myös tarkempia kuvia, jotka eivät perustu vain materiaalin painoon. Tämä voi sisältää esimerkiksi kuvia perustuen materiaalin mittoihin, muotoihin tai muihin ominaisuuksiin. Näin sovellus pystyy tarjoamaan entistä monipuolisempaa ja tarkempaa tietoa käyttäjille.

8 YHTEENVETO

Opinnäytetyön tavoitteena oli suunnitella, toteuttaa ja testata Wärtsilän sisäisen DTCA-työkalun Web-sovellukseen uusi ominaisuus, joka tarjoaa loppukäyttäjälle tavan visualisoida materiaalien, komponenttien ja moottorien kokoa perustuen niiden painoarvoon. Jatkokehityksen suunnittelussa otettiin huomioon sovelluksen kokonaisarkkitehtuuri, käyttäjävaatimukset, tekniset rajoitukset sekä resurssit. Suuren sovelluksen jatkokehitys ominaisuuden osalta tarjosi sekä haasteita että oivalluksia.

Oli tärkeää, että ominaisuuden käyttöliittymä on yksinkertainen ja helposti saatavilla. Tässä onnistuttiin noudattamalla suunniteltuja käyttäjävaatimuksia sekä kuuntelemalla loppukäyttäjän tarpeita. Web-sovelluksen jatkokehitys tarjosi arvokkaan mahdollisuuden oppia ja kehittyä ohjelmistokehittäjänä. Ominaisuuden suunnittelu, toteutus ja testaus vaativat ongelmaratkaisutaitoja, teknistä osaamista ja luovuutta, sekä samalla mahdollisuuden syventää ymmärrystä ohjelmistokehityksen käytännöistä ja prosesseista sekä osallistua todelliseen sovelluskehitykseen ja tiimityöhön. Työssä onnistuttiin toteuttamaan tavoiteltu ominaisuus vaatimusten perusteella.

LÄHTEET

[1] React. Viitattu 24.01.2024. https://en.wikipedia.org/wiki/React_(software)

[2] React Komponentti. Viitattu 13.4.2024. https://joinex.fi/react-pahkinankuoressa/

[3] React render method. Viitattu 13.4.2024. w3schools.com/react/react_components.asp

[4] TypeScript. Viitattu 13.4.2024. https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html

[5] Python. Viitattu 31.1.2024. https://fi.wikipedia.org/wiki/Python_(ohjelmointikieli)

[6]. Python kirjastot. Viitattu 13.4.2024. https://www.geeksforgeeks.org/libraries-in-python/

[7] Django. Viitattu 13.4.2024. https://www.w3schools.com/django/django_intro.php

[8] Django ORM datahaku. Viitattu 13.4.2024. https://medium.com/@barnaantekalign/how-django-processes-an-orm-query-f2316c95b

[9] Django MVT-malli. Viitattu 31.1.2024. https://www.dothe-dev.com/blog/amp/what-is-django-used-for/

[10] Jest. Viitattu 13.4.2024. https://www.browserstack.com/guide/react-app-testing-with-jest

[11] What is ETL?. Viitattu 06.02.2024. https://aws.amazon.com/what-is/etl/

[12] PostrgeSQL. Viitattu 13.4.2024. https://en.wikipedia.org/wiki/PostgreSQL

[13] PostgreSQL Benefits. Viitattu 13.04.2024. https://www.ibm.com/topics/postgresql

[14] Importing and Exporting components. Viitattu 13.04.2024. https://react.dev/learn/importing-and-exporting-components

[15] How to pass props to a functional React component using typescript. Viitattu 13.04.2024. https://blog.wordbot.io/tech/how-to-pass-props-to-a-functional-react-component-using-typescript/

[16] Importing and Exporting components. Viitattu 13.04.2024. https://react.dev/learn/importing-and-exporting-components

[17] Introducing JSX. Viitattu 13.04.2024. https://legacy.reactjs.org/docs/introducing-jsx.html

[18] Element, className property. Viitattu 13.04.2024. https://developer.mozilla.org/en-US/docs/Web/API/Element/className

[19] Using the State hook. Viitattu 13.04.2024. https://legacy.reactjs.org/docs/hooks-state.html

[20] Development and test enviroments: Understanding the different types of enviroment. Viitattu 13.04.2024. https://www.unitrends.com/blog/development-test-environments

LIITTEET

LIITE 1. Salassa pidettävä aineisto. DTCA-arkkitehtuuri UML-kaavio

LIITE 2. Salassa pidettävä aineisto. MaterialCard.tsx-komponentti