OULUN AMMATTIKORKEAKOULU

Kimmo Raappana

AUTOMAATIOJÄRJESTELMÄ MEKANIIKKAMALLIEN MUUNTAMISEKSI 3D-MALLEIKSI

AUTOMAATIOJÄRJESTELMÄ MEKANIIKKAMALLIEN MUUNTAMISEKSI 3D-MALLEIKSI

Kimmo Raappana
Opinnäytetyö
Kevät 2021
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä: Kimmo Raappana
Opinnäytetyön nimi: Automaatiojärjestelmä mekaniikkamallien muuntamiseksi 3D-malleiksi
Työn ohjaaja: Jukka Jauhiainen
Työn valmistumislukukausi ja -vuosi: Kevät 2021 Sivumäärä: 27

Työn tilaajayritys oli Nokia. Opinnäytetyössä toteutettiin tilaajayritykselle automatisoitu järjestelmä, joka hakee CAD-mallin tuotehallintajärjestelmästä, käsittelee mallin, tallentaa sen palvelimelle ja tarjoaa rajapinnan mallin lataamiseen. Järjestelmä on osa isompaa mikropalveluarkkitehtuuria ja koostuu kahdesta loogisesta kokonaisuudesta. Toinen osa on palvelu, joka suorittaa CAD-mallin haun, lataamisen ja käsittelyn. Toinen osa koostuu REST-rajapinnasta, jonka avulla käsitellyt mallit tallennetaan palvelimelle ja noudetaan sieltä.

Ennen järjestelmän toteuttamista CAD-mallien etsiminen tuotehallintajärjestelmästä, mallin käsittely ja sen asiakasohjelmistoon tuominen on ollut manuaalisesti suoritettava prosessi. Työn tarkoitus oli toteuttaa skaalautuva ratkaisu tuotehallintajärjestelmän ja asiakasohjelmistojen välille.

Työn lopputuloksena on ensimmäinen versio automaatiojärjestelmästä, joka tarjoaa palvelun tuotehallintajärjestelmässä olevien CAD-mallien tuomiseen asiakasohjelmistoon. Järjestelmän kehitys jatkuu opinnäytetyön jälkeen ja sen ominaisuuksia laajennetaan ja vakautta parannetaan testikäyttöönoton ohessa.

Asiasanat: REST, automaatio, 3D-mallit, CAD-mallit

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Software Development

---

Author: Kimmo Raappana
Title of thesis: Automatic Conversion of CAD models into 3D models
Supervisor: Jukka Jauhiainen
Term and year when the thesis was submitted: Spring, 2021
Number of pages: 27

The goal of the thesis work was to build an automation pipeline that provides a scalable solution for bringing CAD models from a product lifecycle management system into client software. The system is a part of a larger microservice architecture and consists of two parts: a service responsible for fetching, converting and optimizing a CAD model and a REST application programming interface that provides endpoints for uploading and downloading the processed models into and from a server.

Before the thesis work, finding CAD models from the product lifecycle management system, processing them and bringing them into client software has been manual work and the need for the thesis work came from the need to automate this process.

During the thesis work a first iteration of a pipeline providing a service for bringing CAD models from a product lifecycle management system into client software was built. The development of the system will continue beyond the thesis work. Its features will be expanded and stability improved side by side with test usage of the system.

3D-models, automation, REST, CAD

SISÄLLYS

SANASTO

.NET	Microsoftin kehittämä ohjelmistokomponenttikirjasto
C#	Microsoftin kehittämä ohjelmointikieli
CAD	Computer Aided Design, tietokoneavusteinen suunnittelu
Django	Pythoniin pohjautuva avoimen lähdekoodin web-ohjelmistokehys
FBX	Autodeskin kehittämä 3D-tiedostotyyppi
glTF	Khronos Groupin kehittämä tiedostotyyppi 3D-malleille. Käyttää kahta mahdollista tiedostopäätettä, GLTF ja GLB
Javascript	Pääasiassa web-ympäristöissä käytetty korkean tason ohjelmointikieli
JSON	Javascript Object Notation, yksinkertainen avoimen standardin tiedosto-muoto tiedonvälitykseen.
Mikropalvelut	Arkkitehtuurimalli, jossa ohjelmisto jaetaan itsenäisesti toimiviin loogisiin kokonaisuuksiin eli mikropalveluihin
OBJ	Wavefront Techologies -yrityksen kehittämä 3D-tiedostotyyppi
Ohjelmistokehys	Kokoelma ohjelmistotuotteita, jotka muodostavat rungon sen päälle kehi-tettävästä tietokoneohjelmasta
PLM	Product Lifecycle Management, tuotteen elinkaaren hallinta
PLM-järjestelmä	Tuotteen elinkaaren hallintaan tarkoitettu järjestelmä, tuotehallintajärjes-telmä
Python	Monipuolinen korkean tason ohjelmointikieli
REST	Representational State Transfer, arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen

1 JOHDANTO

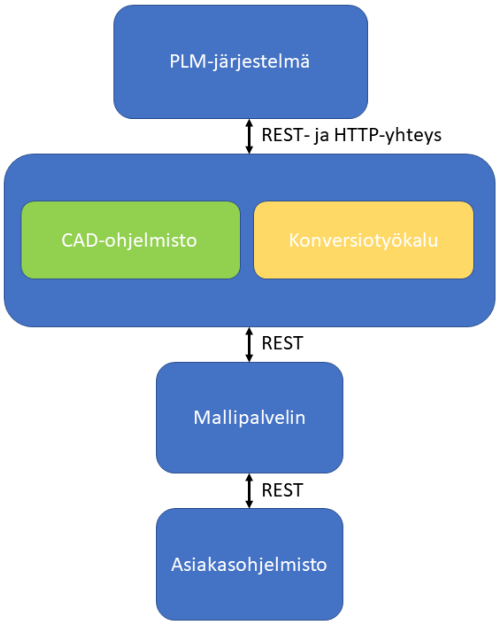
Työn tarkoituksena oli toteuttaa järjestelmä, joka on osa isompaa mikropalveluarkkitehtuuria. Opinnäytetyön toteutus sisältää kaksi tähän mikropalveluarkkitehtuuriin kuuluvaa mikropalvelua. Ensimmäisen mikropalvelun tehtävä on automatisoida CAD-mallin etsiminen tilaajayrityksen käyttämästä tuotehallintajärjestelmästä ja sen kääntäminen 3D-malliksi, jota yleisimmät 3D-moottorit tukevat. Toinen palvelu tarjoaa rajapinnan 3D-mallien tallentamiseen palvelimelle ja noutamiseen palvelimelta. Vaikka palvelut ovat kaksi loogista kokonaisuutta, muodostavat ne yhdessä automaatiojärjestelmän.

Toteutetun automaatiojärjestelmän kaltaiselle systeemille on ollut pitkään tarve mikropalveluarkkitehtuuriin, sillä ennen opinnäytetyötä prosessi on ollut manuaalista ja aikaa vievää työtä joka ei ole skaalautuva ratkaisu. Tuotehallintajärjestelmän sisälle pääsy vaatii erilliset oikeudet, mikä hidastaa entisestään manuaalista mallien etsintää, ja sen käyttöliittymä on monimutkainen ja voi aiheuttaa vaaratilanteen, jossa kouluttamaton käyttäjä vahingossa poistaa tai muokkaa järjestelmässä olevia tietoja. CAD-mallien haun automaatio eliminoi nämä vaaratilanteet. Ajankohtaisuutta tuotteiden 3D-mallien tuomiseen asiakasohjelmistoon myös lisää opinnäytetyön toteutuksen aikana vallinnut Covid-19-pandemia, joka luo tarpeen korvata fyysinen radiotuotteiden prototyyppien lähetys ja tuoteesittelytilaisuuksiin matkustaminen virtuaalisilla prototyypeillä ja tapaamisilla.

2 JÄRJESTELMÄN YLEISKUVA

2.1 Järjestelmän arkkitehtuuri

Toteutettu automaatiojärjestelmä on osa isompaa mikropalveluarkkitehtuuria, joka koostuu monista palveluista. Vaikka toteutettu järjestelmä on itsessään yksi kokonaisuus, myös se koostuu kahdesta pienemmistä itsenäisestä palvelusta. Järjestelmän alkupäässä on palvelin, joka hakee PLM-järjestelmästä CAD-mallin ja kääntää sen 3D-malliksi. Toinen osa on mallipalvelin, joka tarjoaa REST-rajapinnat mallien palvelimelle lähettämiseen ja sieltä lataamiseen. CAD-mallien haun ja käännön suorittava palvelin ja konversiotyökalu sijaitsevat samassa järjestelmässä, joka kommunikoi mallipalvelimen kanssa REST-rajapinnan kautta. Arkkitehtuuri on kuvattu kuvassa 1.

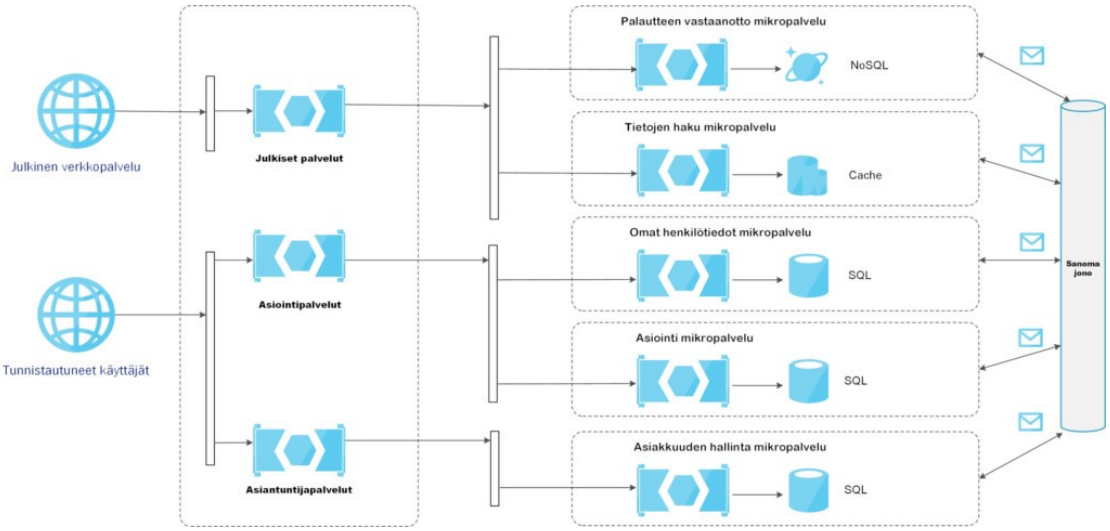


KUVA 1. Järjestelmäarkkitehtuurin yleiskuva

2.2 Mikropalveluarkkitehtuuri

Mikropalveluarkkitehtuuri on ohjelmistosuunnittelussa käytetty arkkitehtuurimalli, jossa ohjelmisto jaetaan itsenäisesti toimiviin loogisiin kokonaisuuksiin eli mikropalveluihin. Yleinen jakotapa ohjelmistokokonaisuuden prosesseille on hajauttaa ne pilvipalveluihin tai eri palvelimille. (1.) Tässä opinnäytetyössä ja isommassa kokonaisuudessa, jonka osa se on, eri palvelut ovat hajautettu omille palvelimilleen.

Mikropalveluarkkitehtuurin etuna on sen modulaarisuus, joka mahdollistaa järjestelmän helpomman testauksen, paremman suorituskyvyn ja joustavuutta siinä, että yhden mikropalvelun voi korvata uudella ilman tarvetta muokata muuta järjestelmää. Myös yksittäisen palvelun käyttökatkon kohdalla mikropalveluarkkitehtuurin ansiosta muun järjestelmän käyttö ei ole estynyt katkon aikana. Koska mikropalveluarkkitehtuurissa palvelut ovat itsenäisiä loogisia kokonaisuuksia, mahdollistaa se myös järjestelmän eri osien yhtäaikaisen kehittämisen eri tiimien toimesta. Kuvassa 2 on esimerkki mikropalveluarkkitehtuurirakenteesta. Kuvan rakenteessa nettisivun toiminto on jaettu palveluihin, jotka on edelleen jaettu mikropalveluihin. (2.)

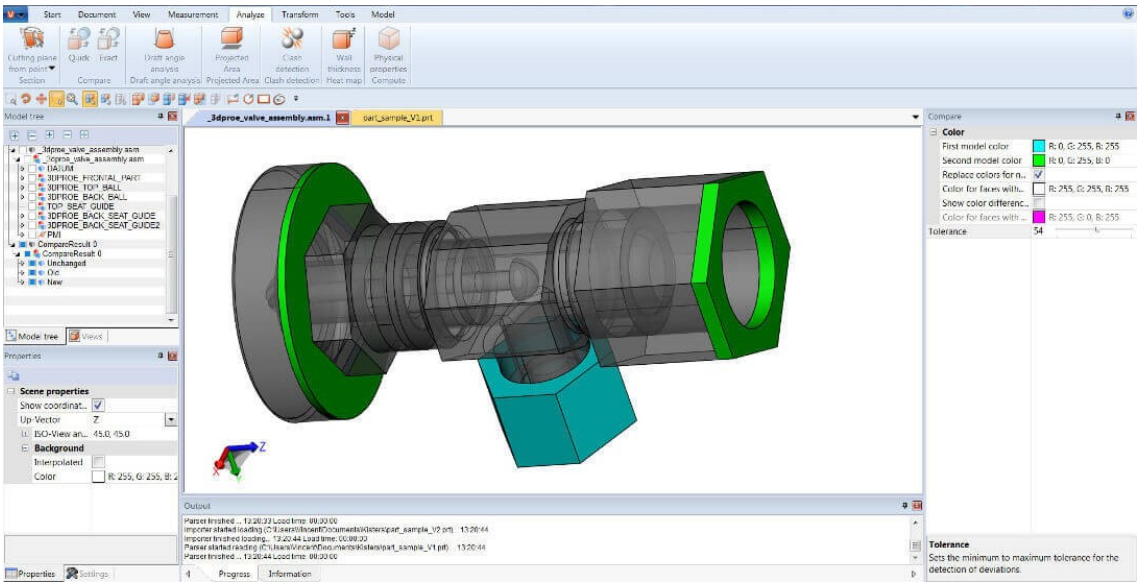


KUVA 2. Esimerkki mikropalveluarkkitehtuurista (2)

2.3 PLM-järjestelmä, CAD-ohjelmisto ja konversiotyökalu

PLM-järjestelmä (Product Lifecycle Management) on tilaajayrityksen käyttämä tuotehallintajärjestelmä, joka on tarkoitettu tuotteiden elinkaaren hallintaan. Käytetyn PLM-järjestelmän palvelimelle on tallennettu CAD-mallit tilaajayrityksen tuotteista. PLM-järjestelmät ovat kehitetty keräämään ja hallitsemaan tuotteeseen liittyviä tietoja ja suunnitteluprosesseja (3).

CAD (Computer-Aided Design), eli tietokoneavusteinen suunnittelu, on korvannut manuaalisen paperille suunnittelun insinöörintyössä. CAD-ohjelmistot auttavat niiden käyttäjiä suunnittelemaan tuotteita kahdessa ja kolmessa ulottuvuudessa (4). Kuvassa 3 nähdään tyypillinen näkymä manipuloitavasta mallista CAD-sovelluksessa.



KUVA 3. Tyypillinen näkymä CAD-sovelluksessa (5)

Tässä opinnäytetyössä CAD-ohjelmisto on osa automatisoitavaa järjestelmää, mutta sitä ei käytetä CAD-mallien manipuloimiseen, vaan se toimii rajapintana PLM-järjestelmän palvelimelle, joka sisältää haettavat CAD-mallit. Tästä johtuen opinnäytetyössä ei perehdytä CAD-ohjelmistoon ja sen käyttöön CAD-työkaluna. CAD-sovelluksella mallien hakua PLM-järjestelmästä ja niiden syöttämistä konversiotyökalulle on automatisoitu kolmannen osapuolen Python-kirjastolla, jolla hallitaan CAD-sovelluksen automatisointiin tarkoitettua avoimen lähdekoodin työkalua. Työkalu toimii lähettämällä CAD-sovellukselle JSON-muotoisia komentoja. Vaihtoehtoinen tapa automatisoida CAD-sovellusta, johon perehdyin toteutusta suunnitellessa, on Tcl-ohjelmointikieli, mutta tässä opinnäytetyössä käytettiin kolmannen osapuolen avoimen lähdekoodin työkalua, sillä sen kanssa voitiin käyttää mitä tahansa ohjelmointikieltä, välttäen tarpeen opetella marginaalista kieltä, kuten Tcl. Ohjelmointikielen vapaudessa on myös se etu, että automatisointiin voidaan käyttää Python-ohjelmointikieltä, jota käytetään myös järjestelmän muissa osioissa.

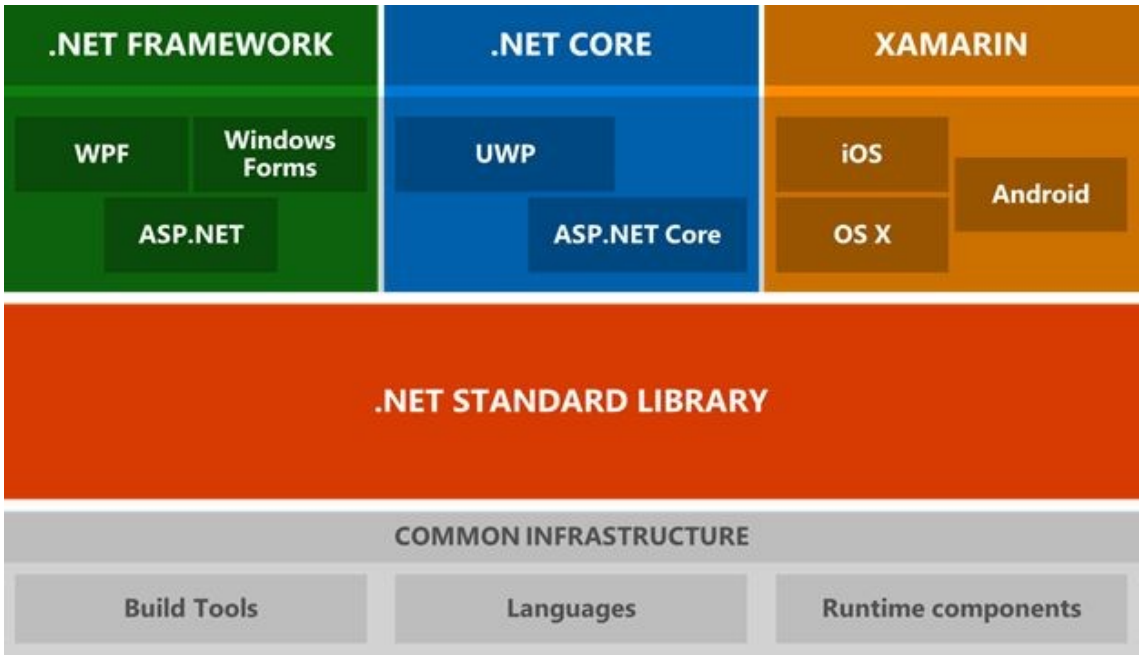
CAD-mallien tiedostotyypit ovat spesifisiä CAD-ohjelmistoille, joten mallit eivät ole käytettävissä asiakasohjelmistoihin kuuluvissa 3D-moottoreissa. Tästä johtuen automaatiojärjestelmään kuuluu konversiotyökalu, joka kääntää CAD-mallin 3D-malliksi. Käytetty konversiotyökalu on CAD-mallien kääntämiseen 3D-malleiksi ja niiden optimointiin tarkoitettu ohjelmisto. Konversiotyökalu ja CAD-ohjelmisto sijaitsevat samalla palvelimella. Palvelin on Django-ohjelmistokehystä käyttäen toteutettu REST-rajapinnan tarjoava palvelin. Tämän REST-rajapinnan päätarkoitus on tarjota metodi, joka käynnistää CAD-mallin haun tuotehallintajärjestelmästä. Django on Python-ohjelmointikieleen perustuva web-rajapintojen rakentamiseen tarkoitettu kirjasto (6).

2.4 Mallipalvelin

Mallipalvelin on palvelin, jonne käännetyt 3D-mallit lähetetään. Palvelin tarjoaa REST-rajapinnan mallien siirtämiseen palvelimelle, mallien lataamiseen palvelimelta, mallien poistamiseen palvelimelta, mallien tietojen hakuun ja mallien tietojen päivittämiseen. Palvelimen ohjelmoinnissa on käytetty C#-ohjelmointikieltä ja .NET Core -kehysympäristöä. Palvelimelle on myös ylläpitäjien käyttöliittymä, jonka kautta sinne voi lähettää manuaalisesti 3D-malleja, tarkastella niiden tietoja ja päivittää niiden tietoja. Käyttöliittymän kautta hallitaan myös 3D-mallien näkyvyyttä asiakasohjelmistoille. Ennen kuin käyttöliittymästä käydään hyväksymässä mallit julkiseen käyttöön, mallit eivät ole näkyvissä asiakasohjelmistolle. Ylläpitäjien käyttöliittymä on toteutettu Javascript-ohjelmointikieltä ja React-ohjelmistokehystä käyttäen.

2.4.1 .NET Core

.NET Core on Microsoftin kehittämä ohjelmistokomponenttikirjasto. Kirjasto sisältää avoimen lähdekoodin, on ilmainen ja sitä voidaan ajaa Windows-, macOS- ja Linux-käyttöjärjestelmillä (7). .NET Core -ohjelmistokehystä voidaan käyttää rakentamaan monenlaisia ohjelmistoja, kuten mobiilisovelluksia, web-sovelluksia ja IoT-sovelluksia. .NET Core tukee myös useita ohjelmointikieliä, joista tärkeimmät ovat C#, F# ja Visual Basic. (8.) .Net Core on osa Microsoftin .NET ekosysteemiä, joka on esitelty kuvassa 4 .NET Core on seuraaja .NET Framework -ohjelmistokehykselle. .NET Core -ohjelmistokehyksen etuja sen edeltäjään ovat mm. parempi suorituskyky, parempi tuki mikropalveluille ja usean käyttöjärjestelmän tuki. (9.)

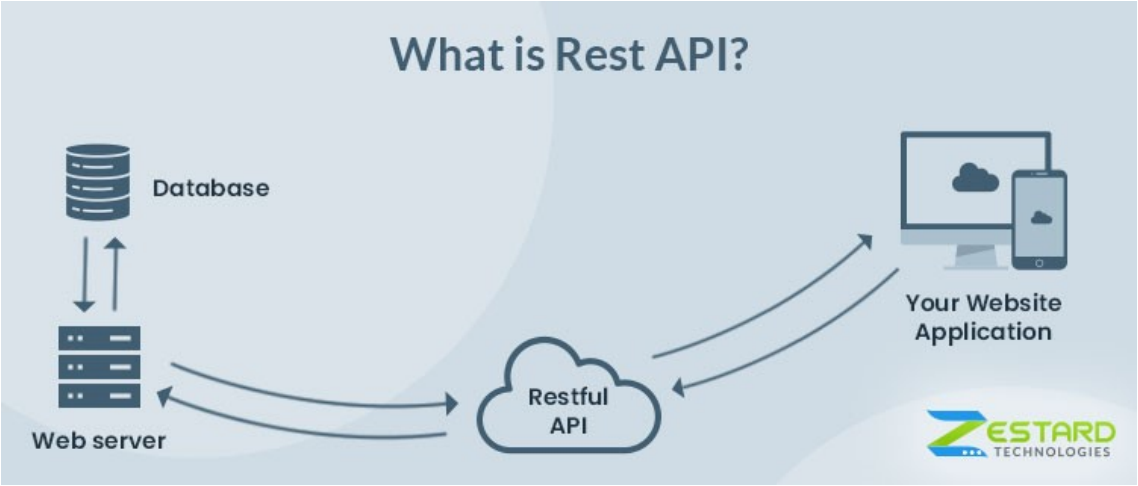


KUVA 4. .Net arkkitehtuuri (10)

ASP.NET Core on .NET Core -ohjelmistokehykseen sisältyvä web-kehysympäristö (kuva 4). Niin kuin .NET Core, myös ASP.NET sisältää avoimen lähdekoodin ja tukee Windows-, macOS- ja Linux -käyttöjärjestelmiä (11).

2.4.2 REST-rajapinta

REST-rajapinta (Representational State Transfer) on verkkopalveluihin tarkoitettu ohjelmistoarkkitehtuurillinen tyyli. REST-rajapinnan tärkeimmät osuudet korkealla tasolla käsiteltynä ovat resurssit ja niiden metodit. Resurssi REST-rajapinnassa on informaatiorakenne jostain asiasta. Mikä tahansa tieto, joka voidaan nimetä, voi olla resurssi. (12.) Esimerkkinä tässä opinnäytetyössä käytetty REST-resurssi on ImportModelResource, jossa määritellään palvelimelle ladattavan 3D-mallin tietue. Tähän tietueeseen kuuluvaa dataa on mm. mallin nimi ja aika, jolloin sitä on viimeksi muokattu. Kuvassa 5 nähdään yleinen REST-rajapinnan periaate.

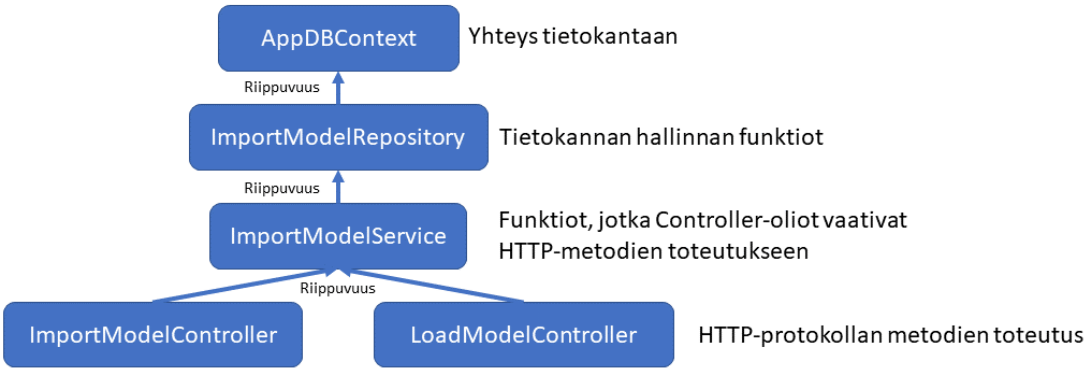


KUVA 5. REST-rajapinnan toimintaperiaate (9)

Resurssien metodit ovat REST-rajapinnan päätepisteitä. Tässä opinnäytetyössä resurssien metodit ovat toteutettu HTTP-protokollan metodeilla. HTTP-protokollaan metodeja ovat GET, POST, PUT, PATCH, HEAD ja OPTIONS (13). REST-rajapintaan sisältyy myös juuriosoite resurssien käyttöön, tässä työssä esimerkiksi /ImportModelResource, ja resurssien esitysmuodon määrittelevä mediatyyppi, joka on tässä työssä JSON (14). HTTP-protokollan GET-metodilla noudetaan dataa, POST-metodilla lisätään uutta dataa, PUT-metodilla päivitetään olemassa olevaa dataa, PATCH-metodilla päivitetään olemassa olevan datan osajoukkoa ja DELETE-metodilla poistetaan dataa. HEAD-metodilla voidaan hakea resursseja, mutta se palauttaa vain otsikkotiedot. OPTIONS-metodilla voidaan hakea käytettävissä olevat HTTP-metodit (13). Näistä PATCH-metodia, HEAD-metodia ja OPTIONS-metodia ei ole käytetty tässä työssä.

3 MALLIPALVELIMEN TOTEUTUS

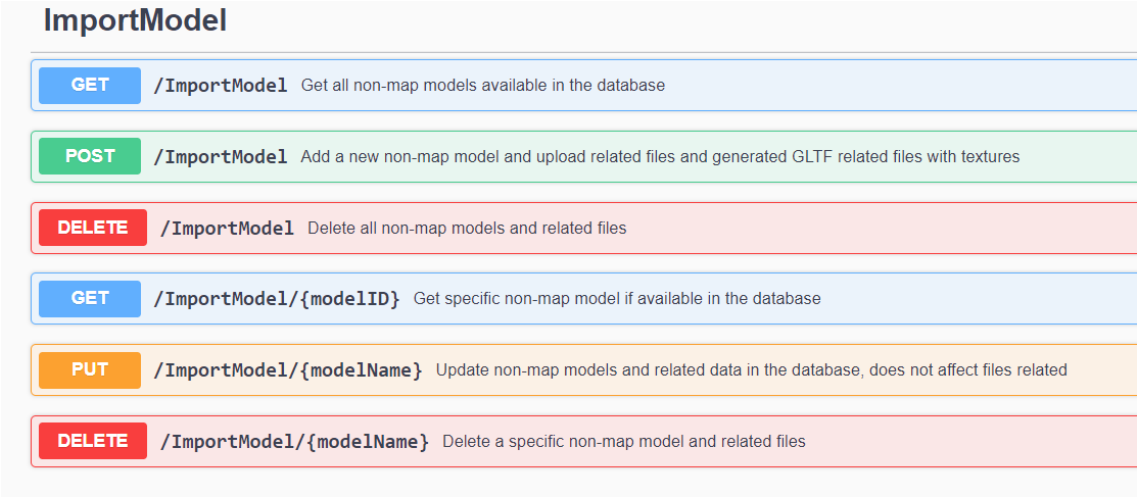
Työn toteutus alkoi järjestelmän loppupäästä, mallipalvelimesta, ja 3D-mallien lataamisesta sinne manuaalisesti. Perustoteutus palvelimelle oli jo entuudestaan valmiina ja opinnäytetyön tehtävänä oli laajentaa olemassa olevaa palvelinta tukemaan geneerisiä 3D-malleja. Palvelimen REST-rajapinta on toteutettu käyttäen .NET Core ja ASP.NET -ohjelmistokehyksiä. REST-rajapinnan toteutus noudattaa dependency injection -suunnittelumallia, jossa luodaan riippuvuuksien ketju injektoimalla instanssi oliosta toiseen. Kuvassa 6 kuvataan tätä riippuvuusketjua ImportModel -resurssille. Palvelimen manuaalista hallinnointia varten tarkoitettu käyttöliittymä on toteutettu käyttäen React Admin -kehysympäristöä.



KUVA 6. REST -rajapinnan ImportModel-resurssin toteutuksen rakenne

3.1 REST-rajapinta

REST-rajapinta toteutettiin .NET Core -ohjelmistokehitystä ja C#-ohjelmointieltä käyttäen. REST-rajapintaan on myös implementoitu Swagger-käyttöliittymä. Swagger-käyttöliittymä on REST-rajapintojen kuvaamiseen tarkoitettu työkalu, jolla voidaan mm. visualisoida rajapinnan resurssit ja niiden metodit (15). Kuvassa 7 nähdään työssä käytettyjä HTTP-metodeja visualisoituna Swagger-käyttöliittymässä.

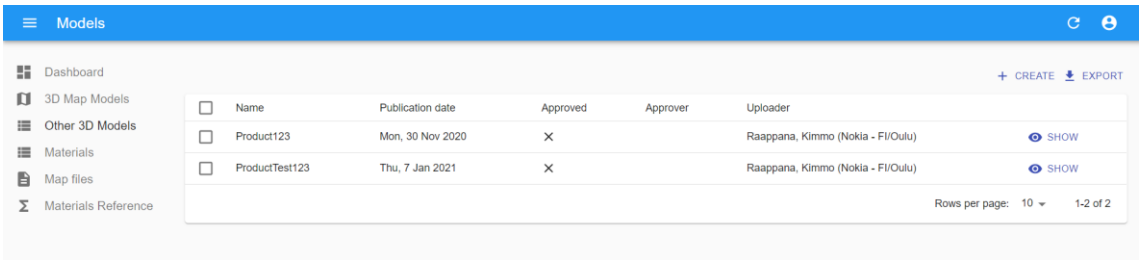


KUVA 7. Mallin HTTP-metodeja Swagger-käyttöliittymässä

Toteutettu REST-rajapinta sisältää kaksi Controller-luokan oliota, ImportModelController ja LoadModelController. ImportModelController-päätepiste vastaa Mallien lähettämisestä palvelimelle, datan noutamisesta sieltä, datan poistamisesta ja päivittämisestä. ImportModelController-olion HTTP-metodit ovat nähtävissä kuvassa 7. LoadModelController sisältää vain yhden HTTP-metodin, GET-pyynnön mallin lataamiseen palvelimelta. GET-pyynnölle annetaan parametrina tuotteen nimi ja tiedostotyyppi. Palvelimelle 3D-mallin ladatessa generoidaan mallille GLTF- ja GLB-muotoiset 3D-mallit, jos niitä ei ladatulle mallille valmiiksi ole. Tuetut tiedostotyypit mallipalvelimella ovat OBJ, GLTF, GLB ja FBX.

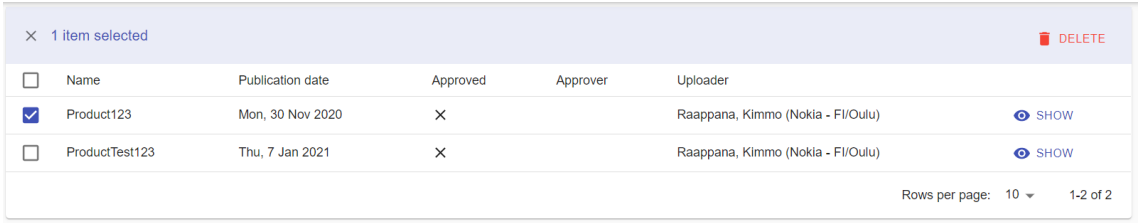
3.2 Mallipalvelimen ylläpitäjien käyttöliittymä ja toiminta

Kuten mallipalvelimelle, myös sen ylläpitäjien käyttöliittymälle oli perustoteutus jo valmiina. Tehtävänä oli laajentaa käyttöliittymä tukemaan geneerisiä 3D-malleja. Kuvassa 8 nähdään näkymä ylläpitäjien käyttöliittymästä. Tässä opinnäytetyössä toteutettiin kuvan vasemmassa reunassa näkyvä ”Other 3D models” -osio. Ylläpitäjien käyttöliittymä on toteutettu käyttäen React Admin -ohjelmistokehystä. Listanäkymään haetaan palvelimella olevat tuotteen GET-pyynnöllä ImportModelController-päätepistettä käyttäen.



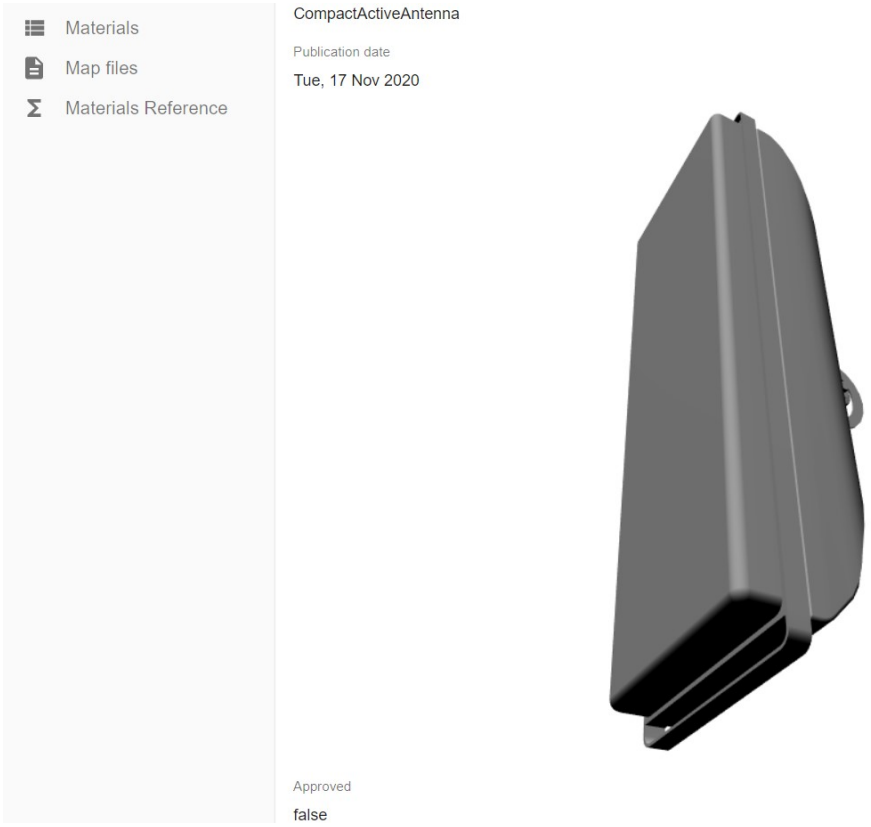
KUVA 8. Listanäkymä ylläpitäjien käyttöliittymässä

Ylläpitäjien käyttöliittymästä voidaan hallita mallipalvelimella olevaa dataa. Listanäkymästä voidaan valita yksi tai useampi tuote, jolloin tuotteen poistamisen vaihtoehto paljastuu. Tämä nähdään kuvassa 9.



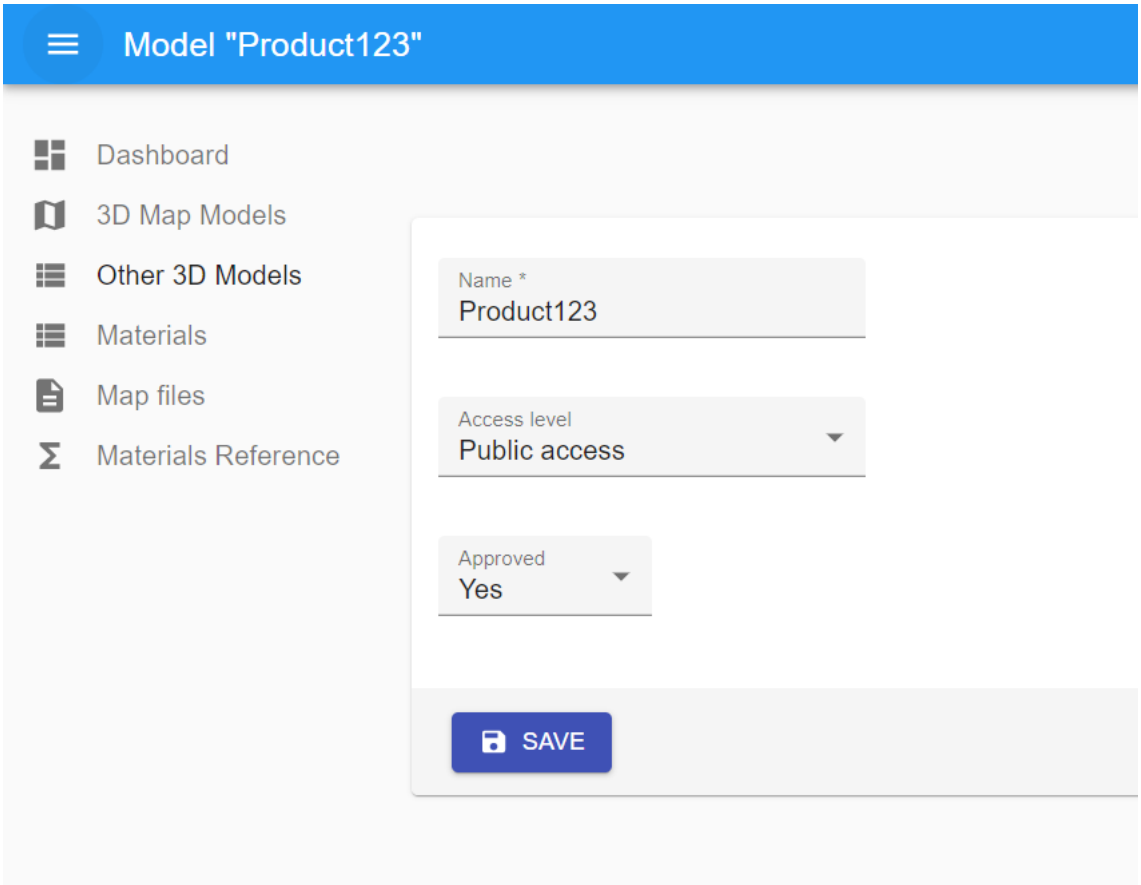
KUVA 9. Mallin poistamisen valinta

Ylläpitäjien käyttöliittymä sisältää myös yksittäisen tuotteen 3D-mallin katselun, joka nähdään kuvassa 10, ja yksittäisen tuotteen näkyvyyden muokkaamisen, nähtävillä kuvassa 11.



KUVA 10. Näkymä palvelimelle ladatusta 3D-mallista ylläpitäjien käyttöliittymässä

Kuvassa 11 näkyvän Access level -kentän avulla voidaan hallita näkyvyyttä vain todennetuille käyttäjille tai julkisesti kaikille käyttäjille. Approved-kentän avulla pidetään kirjaa siitä, että tuotteen 3D-malli on tarkastettu ja hyväksytty näytettäväksi asiakasohjelmistossa.



KUVA 11. Tuotteen näkyvyyden muokkaus

Kaikissa ylläpitäjien käyttöliittymän toiminnoissa on sama periaate. Mallipalvelimelle lähetetään toimintoa vastaava HTTP-pyyntö, jonka avulla kutsutaan controller-olion vastaavaa metodia. Controller-olion sisällä kutsutaan resurssin Service-luokkaa, joka sisältää funktiot HTTP-metodien toteuttamiseen. Näihin kuuluvat tuotteiden listaaminen, etsiminen ID:n ja nimen perusteella sekä tallentamisen, päivittämisen ja poistamisen funktiot. Esimerkiksi PUT-pyyntöä suorittaessa tarkastetaan käyttäen resurssin Service-luokkaa, onko mallia entuudestaan olemassa palvelimella. Kuvassa 12 nähdään ImportModelController-olion PUT-metodissa ImportModelService-olion käyttöä. Funktiossa etsitään ImportModelService-luokan avulla malliresurssi ModelId-kentän perusteella ja tarkastetaan ehtolauseessa, onko kenttää vastaavaa mallia palvelimella. Koska kyseessä on PUT-metodi ja se hoitaa datan päivittämisen palvelimella, palauttaa funktio virheilmoituksen, jos haettua mallia ei löydy. `_importModelService.FindAsync(model.ModelId)` -metodi kutsuu edelleen ImportModelService-olion sisällä vastaavan ImportModel-resurssin Repository-luokkaa. Repository-luokka muokkaa AppDBContext-

luokassa sijaitsevia DbSet-olioita, joiden avulla itse tietokantaan tehdään kyselyitä. Tämä riippuvuuksien ketju esiteltiin jo aiemmin kuvassa 6. Kuvassa 12 nähdään myös mallin näkyvyyden tarkastaminen. Kuvan ehtolauseessa tarkastetaan sekä tuotteen näkyvyystaso että käyttäjän oikeudet. Jos malli on määrätty näkyvyydeltään salaiseksi ja käyttäjällä ei ole ylläpitäjän oikeuksia toiminto estetään.

```
var modelResource = await _importModelService.FindAsync(model.ModelId);
if (modelResource == null)
{
    return NotFound("Model not found");
}

// Check authorization access
if ((model.AccessLevel == "secret" || model.AccessLevel == null) &&
    (!User.IsInRole("NokiaGlobeTeam") || !User.IsInRole("MapModelServerAdmins")))
{
    return Unauthorized();

}
```

KUVA 12. Service-luokan käyttö ja käyttöoikeuksien tarkastamainen ImportModelController-luokan PUT-metodissa

4 PLM-JÄRJESTELMÄN AUTOMAATION TOTEUTUS

Haasteena PLM-järjestelmän automaatiossa oli sekä CAD-sovelluksen että tuotehallintajärjestelmän vanhentuneiden ja epäselvien käyttöliittymien käytön opiskelu. Aiheesta oli myös haasteellista tehdä tiedonhakua, sekä hakutuloksien vähyyden että sovelluksien huonon dokumentaation vuoksi. Automaation kehittämistä myös hidasti vaadittavien lisenssien ja lupien hankinta sovelluksia varten. Itse automaatio on toteutettu kolmannen osapuolen Python-kirjastoa ja PLM-järjestelmän valmiiksi olemassa olevaa REST-rajapintaa käyttäen. REST-rajapinnasta on mahdollista saada vain tekstimuotoista dataa järjestelmässä olevista CAD-malleista, joten yksistään se ei ollut riittävä mallien lataamisen toteuttamiseen, vaan väliin tarvittiin CAD-sovellus rajapinnaksi.

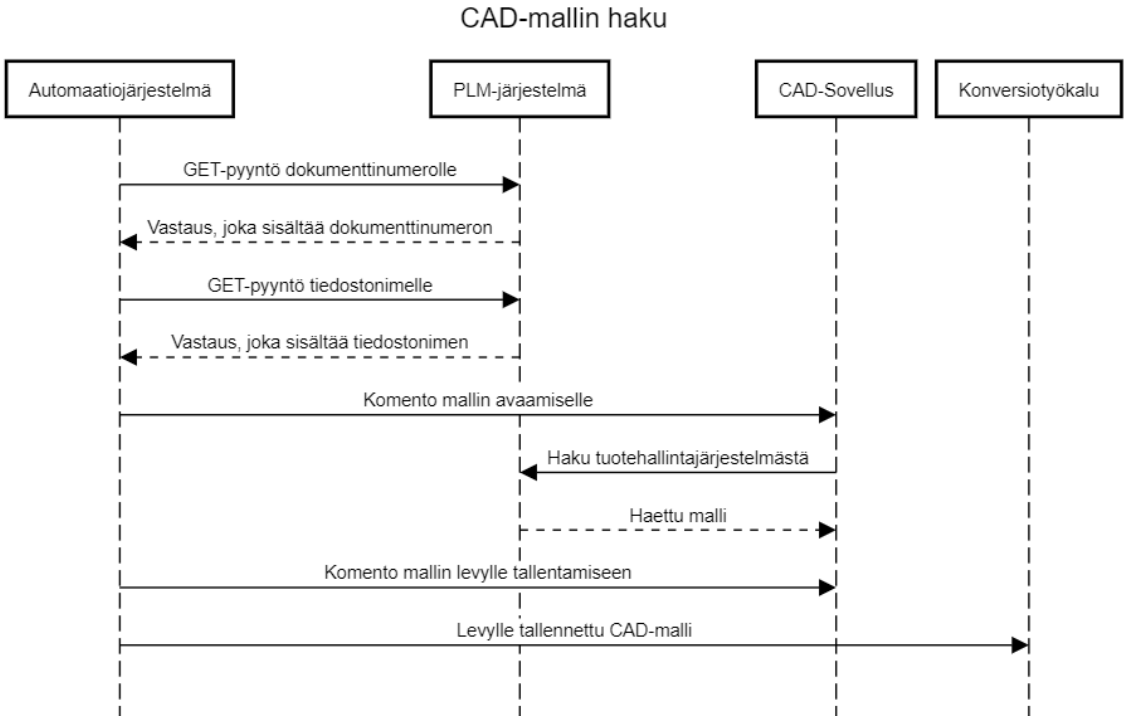
4.1 CAD-sovellus ja PLM-järjestelmä

CAD-sovelluksen ja PLM-järjestelmän automaatio alkoi tutkimustyöllä, jossa selvitettiin, millä teknologioilla se voidaan toteuttaa. Ensimmäinen löytynyt vaihtoehto oli Tcl-ohjelmointikieli (Tool Command Language), joka on yleinen kieli CAD-automatisoinnissa (16; 17). Työ päädyttiin toteuttamaan käyttäen Python-ohjelmointikieleen pohjautuvaa kirjastoa. Kirjastossa oli paljon etuja Tcl-kieleen, kuten kirjaston käytön yksinkertaisuus ja mahdollisuus ohjelmoida automaatio Python-kielellä, jota myös konversiotyökalu käyttää.

CAD-mallin haun suorittava komentosarja alkaa vastaanottamalla haettavan CAD-mallin PLM-järjestelmänumeron ja tekemällä GET-pyynnön PLM-järjestelmän REST-rajapintaan, josta numeron perustella haetaan dokumenttinumero. Tuote- ja dokumenttinumero ovat PLM-järjestelmässä metadataa, joka sisältyy CAD-malleihin. Dokumenttinumeron perusteella tehdään toinen GET-pyyntö REST-rajapinnan eri päätepisteeseen, josta saadaan CAD-mallin tiedostonimi. Kun PLM-järjestelmänumeron perusteella on löydetty tiedostonimi, seuraavaksi muodostetaan yhteys CAD-ohjelmiston hallintaan tarkoitettuun paikalliseen palvelimeen ja käynnistetään CAD-ohjelmisto komentosarjaa käyttäen. Paikallinen palvelin sisältyy kolmannen osapuolen CAD-ohjelmiston automaatioon tarkoitettuun kirjastoon.

CAD-ohjelmiston käynnistyttyä muodostetaan yhteys sen ja PLM-järjestelmän palvelimen välille. Kun yhteys on muodostettu ja todennettu, avataan haluttu malli tiedostonimen perusteella PLM-

järjestelmästä CAD-sovelluksessa. Koska mallin hakeminen PLM-järjestelmästä ja konversiotyökalun prosessit tapahtuvat samalla järjestelmällä, haettu CAD-malli voidaan tallentaa lokaalisti kovalevylle ja syöttää eteenpäin konversiotyökalulle. Lopuksi CAD-sovellus ja yhteydet suljetaan. Tämä CAD-mallin haun komentosarja on kuvattu sekvenssidiagrammina kuvassa 13.



KUVA 13. CAD-mallin haun sekvenssidiagrammi

4.2 Django REST -rajapinta

Django REST -rajapinnan päätehtävä on tarjota POST-metodi, joka käynnistää CAD-mallin haku- ja konversioprosessin syötetylle tuotteelle. POST-pyyntöön syötetään mukaan tuotekoodi ja PLM-järjestelmänumero. PLM-järjestelmänumeron perusteella käynnistetään haku CAD-mallille PLM-järjestelmästä. Jos haku, tallennus ja konversio ovat onnistuneet, tallennetaan Django-palvelimelle tietue, joka sisältää tuotekoodin, tuotehallintajärjestelmänumeron ja tallennetun mallin nimen. Haettujen mallien tiedot tallennetaan Django-palvelimelle, mikä helpottaa virheenseurantaa tilanteessa, jossa järjestelmän jossain vaiheessa esiintyy virhetilanteita.

Kuvassa 14 on komentosarja, joka käynnistää CAD-Mallin haun, kun POST-pyyntö suoritetaan. `CADFetcherProduction`-komentosarjan `start_fetching_from_PLM`-funktioon, joka sisältää kuvassa 13 esitellyn sekvenssin, syötetään parametriksi PLM-järjestelmänumero. Haun ja mallin tallennuksen onnistuessa funktio palauttaa tallennetun tiedostonimen,

joka syötetään mukaan POST-pyyntöön palvelimelle tiedon tallentamista varten. Siinä tapauksessa, missä CAD-mallin käsittelyssä tapahtuu virhe, funktio palauttaa totuusarvomuuttujan arvolla epätosi, jonka avulla voidaan ehtolauseessa joko tehdä edellä kuvattu toimenpide tai palauttaa HTTP-vastaus, jossa ilmoitetaan epäonnistumisesta.

```
class ProductViewSet(viewsets.ModelViewSet):
    queryset = Product.objects.all().order_by('product_code')
    serializer_class = ProductSerializer

    def create(self, request, *args, **kwargs):

        # Returns bool if fails, string for filename is succeeds
        success = CADFetcherProduction.start_fetching_from_PLM(request.data['plm_code'])

        if success is False:
            return HttpResponse("Fetch failed")
        else:
            request_data = request.data.copy()
            request_data['filepath'] = success
            request._full_data = request_data
            print("request_data", request.data)
            return super().create(request, *args, **kwargs)
```

KUVA 14. CAD-mallin haku POST-pyyntöä tehdessä

4.3 Konversiotyökalu

Konversiotyökalu sijaitsee samalla palvelimella, kuin CAD-sovellus ja Django REST -rajapinta. Konversiotyökalun tehtävänä on vastaanottaa PLM-järjestelmästä haettu CAD-malli ja muuntaa se 3D-moottoreille soveltuvaan muotoon. Konversio CAD-mallista 3D-malliksi on vaadittava osa järjestelmää, sillä CAD-ohjelmistossa käytetyt tiedostotyypit eivät ole yhteensopivia asiakasohjelmistoissa käytettävien 3D-moottoreiden kanssa.

Yksi konversiotyökalun päätehtävistä on suorittaa optimointia 3D-malleille. Koska CAD-mallit on tarkoitettu tuotteiden fyysistä valmistusta ja koneistusta varten, ne sisältävät hyvin paljon osia ja yksityiskohtia, joita ei tarvitse tai saa käyttää asiakasohjelmistoissa. Suoraan PLM-järjestelmästä tuodut CAD-mallit saattavat sisältää jopa miljoonia kolmioita ja satoja materiaaleja. Tämä tarkoittaa sitä, että ilman optimointia mallit ovat hyvin raskaita renderöidä 3D-moottoreilla. Myös raaoissa CAD-malleissa sisällä oleva elektroniikka ja mallinnus ovat asiakasohjelmiston kannalta ei-haluttua dataa. Poistamalla konversiotyökalulla nämä tuotteen sisäiset mallinnukset vähennetään tietoturvariskejä ja mallista saadaan kevyempi karsimalla kolmioita. Jos yrityksen 3D-malleja sisältävä asiakasohjelmisto joutuu hyökkäyksen kohteeksi ja sen sisältämät mallit vuotavat, tällä sisäisen

mallinnuksen karsimisella minimoidaan haitta suojelemalla tuotteen sisäisiä ja funktionaalisia komponentteja. Toinen optimointi, joka mallien kolmiomäärälle voidaan tehdä konversiotyökalulla, on sen ulkoisen mallinnuksen kolmioiden karsiminen. Raakana CAD-mallien ulkoiset pinnat ovat hyvin tarkasti mallinnettuja ja niistä voidaankin monessa tapauksessa karsia iso määrä kolmioita ilman, että ihmissilmä huomaa eroa mallinnuksen laadussa. Kolmas optimointitapa, joka konversiotyökalulla suoritetaan, on materiaalien karsiminen. Koska tuotteen sisäisen mallinnuksen poistossa karsitaan kokonaisia sisäisiä osia CAD-mallista, voidaan konversiotyökalulla automaattisesti poistaa käyttämättömät materiaalit mallista.

5 JATKOKEHITYS

Järjestelmää kehitetään käyttäen ketteriä menetelmiä, joten työn spesifikaatio elää asiakastarpeiden ja palautteen myötä. Jo työn suorituksen aikana nousi esille uusia asiakastarpeita mallipalvelimen toimintaan, kuten manuaalinen mallien hyväksyntä ja mallin lataajan nimen tallentaminen tietokantaan. Opinnäytetyön lopputuloksena on ensimmäinen iteraatio automaatiojärjestelmästä. Jatkokehityksen mahdollisia aiheita ovat eri laatutasojen tukeminen samalle mallille ja materiaalituki. Mallipalvelin voisi sisältää tuotteelle useamman laatutason mallin saman tietueen alla. Esimerkiksi mallilla voisi olla matala-, keski- ja korkeatasoiset laadut ja asiakasohjelmistossa voidaan määrittää tarpeen mukaan, mitä näistä käytetään. Tässä ensimmäisessä toteutuksessa mallipalvelimelle ladattavat mallit eivät vielä sisällä materiaaleja. Tämä johtaa siihen, että asiakasohjelmistoissa tuotteen eri osilla ei ole omia värityksiä ja pintamäärityksiä, vaan ne ovat ns. kipsimalleja. Materiaalituki itse mallipalvelimelle on helppo toteuttaa, mutta ongelmat nousevat esiin siinä, mistä materiaalimääritykset tulevat. PLM-järjestelmässä olevat CAD-mallit sisältävät jollain tasolla materiaalimäärityksiä, mutta nämä materiaalit eivät ole visuaaliselta representaatioltaan sellaiset kuin niiden täytyisi olla asiakasohjelmistoissa. Yksi esimerkkiratkaisu materiaaleille olisi materiaalikirjasto, johon määritellään standardisoidut materiaalit. Nämä materiaalit vastaisivat CAD-malliin määriteltyjä materiaaleja, mutta niiden visuaalinen representaatio olisi asiakasohjelmistojen tarpeiden mukaista. Materiaalikirjaston materiaalit asetettaisiin tuotteelle PLM-järjestelmän materiaalien metadatan mukaan jossain vaiheessa järjestelmää.

6 POHDINTA

Työn tavoitteena oli toteuttaa automaatiojärjestelmä, jonka avulla voidaan tuoda CAD-malleja yrityksen PLM-järjestelmästä asiakasohjelmistoihin. Järjestelmä lataa CAD-mallin PLM-järjestelmästä, kääntää sen 3D-malliksi ja lataa mallin palvelimelle. Palvelin tarjoaa REST-rajapinnan mallien tuomiseen asiakasohjelmistoihin. Jo ennen työn toimiantoa oli selvä, että opinnäytetyön aikana viimeisteltyä versiosta automaatiojärjestelmästä ei ehdi toteuttaa ja tavoitteena olikin ensimmäisen toimivan testiversion kehitys. Työn lopputuloksena on ensimmäinen versio automaatiojärjestelmästä, joka vastaa tilaajayrityksen minimitarpeita, mutta vaatii vielä jatkokehitystä. Työn aikana haasteita tuli vastaan sen laajuudesta, lukuisista ohjelmointikehyksistä ja monien eri teknologioiden opettelusta. Koska työn voi jakaa kahteen itsenäiseen loogiseen mikropalvelukokonaisuuteen, olisi jo vain toisen näistä toteutus ollut opinnäytetyölle sopiva laajuus. Tämä johti siihen, että en ehtinyt työn aikana perehtyä syvällisesti yksittäisiin käytettyihin teknologioihin ja niiden oppiminen jäi sille tasolle, millä tarvittavan implementaation projektiin pystyi toteuttamaan. Työn laajuudessa oli kuitenkin se hyvä puoli, että sain kuvan, miltä laajempi, monista mikropalveluista koostuva palvelu voi näyttää kokonaisuutena.

LÄHTEET

1. Tirkkonen, Janne 2018. Mikropalvelut selkokielellä – Kenelle ne sopivat? Alfame. Saatavissa: https://www.alfame.com/blog/mikropalvelut-selkokielella-kenelle-ne-sopivat. Hakupäivä 28.12.2020.

2. Hyvärinen, Jukka 2019. Mikropalveluarkkitehtuuri – milloin ja milloin ei? Digital Illustrated. Saatavissa: https://digitalillustrated.com/mikropalveluarkkitehtuuri-milloin-ja-milloin-ei/. Hakupäivä 30.12.2020.

3. Tuotteen elinkaaren hallinta. Wikipedia 2018. Saatavissa: https://fi.wikipedia.org/wiki/Tuotteen_elinkaaren_hallinta. Hakupäivä 7.1.2021.

4. Bernstein, Larry 2020. What is Computer-Aided Design (CAD) and Why It's Important. Procore Technologies. Saatavissa: https://www.procore.com/jobsite/what-is-computer-aided-design-cad-and-why-its-important/. Hakupäivä 31.12.2020.

5. The simplest and the fastest solution to compare CAD model. CAD Interop, 2015. Saatavissa: https://www.cadinterop.com/en/your-needs/analyze-cad-models/the-simplest-and-the-fastest-solution-to-compare-cad-model.html. Hakupäivä 31.12.2020.

6. Django REST Framework. Saatavissa: https://www.django-rest-framework.org/. Hakupäivä 11.1.2021.

7. .NET. Microsoft. Saatavissa: https://dotnet.microsoft.com/download. Hakupäivä 5.11.2020.

8. ASP.NET Core Tutorials. .NET Core. Tutorials Teacher. Saatavissa: https://www.tutorialsteacher.com/core/dotnet-core. Hakupäivä 5.11.2020.

9. Vatwani, Ritesh 2020. What is REST API? Zestard Technologies. Saatavissa: https://www.zestard.com/blog/rest-api-benefits/. Hakupäivä 31.12.2020.

10. Chand, Mahesh 2020. Difference Between .NET and .NET Core. C# Corner. Saatavissa: https://www.c-sharpcorner.com/article/difference-between-net-framework-and-net-core/. Hakupäivä 5.1.2021.

11. What is ASP.NET Core? Microsoft. Saatavissa: https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet-core. Hakupäivä 5.11.2020.

12. What is REST. REST API Tutorial. Saatavissa: https://restfulapi.net. Hakupäivä 20.10.2020.

13. Jansen, Geert 2011. Essay On RESTful API. Saatavissa: https://restful-api-design.readthedocs.io/en/latest/methods.html. Hakupäivä 1.11.2020.

14. Web-palvelinohjelmointi 2019 Java. Osa 6, Rajapinnat ja REST. Agile Education Research -tutkimusryhmä, Helsingin yliopisto. Saatavissa: https://web-palvelinohjelmointi-19.mooc.fi/osa-6/4-rajapinnat-ja-rest. Hakupäivä 1.11.2020.

15. What is OpenAPI? SmartBear Software. Saatavissa: https://swagger.io/docs/specification/about. Hakupäivä: 4.12.2020.

16. Who uses Tcl. Tcler's Wiki, 2018. Saatavissa: https://wiki.tcl-lang.org/page/Who+Uses+Tcl. Hakupäivä 2.12.2020.

17. EDA / CAD. Tcl Developer Xchange. Saatavissa: http://www.tcl.tk/customers/success/edacad.tml. Hakupäivä 2.12.2020.