OULUN AMMATTIKORKEAKOULU

Joonas Kinnunen

RAJAPINNAN KEHITTÄMINEN .NET-OHJELMISTOJEN VÄLILLE

RAJAPINNAN KEHITTÄMINEN .NET-OHJELMISTOJEN VÄLILLE

Joonas Kinnunen
Opinnäytetyö
Kevät 2024
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä: Joonas Kinnunen
Opinnäytetyön nimi: Rajapinnan kehittäminen .NET-ohjelmistojen välille
Työn ohjaaja: Janne Kumpuoja
Työn valmistumislukukausi ja -vuosi: kevät 2024 Sivumäärä: 29 + 1 liite

---

Opinnäytetyö tehtiin toimeksiantona Pinja Digital Oy:lle. Opinnäytetyössä luotiin uusi rajapinta Pinjan kehittämien Timber- ja Forest by Pinja -toiminnanohjausjärjestelmien välille. Timber by Pinja on sahojen ja puunjalostuslaitosten toiminnanohjausjärjestelmä ja Forest by Pinja on puunhankinnan, korjuun ja logistiikan hallinnoimiseen kehitetty toiminnanohjausjärjestelmä. Tarve rajapinnalle tuli asiakkaalta, jolla oli tarve synkronoida Timber by Pinjan sivutuotevaraston ja Forest by Pinjan välivarastojen varastosaldot.

Opinnäytetyön teoriaosuudessa keskitytään verkkorajapintoihin, niiden käyttötarkoituksiin ja yleisimpiin rajapintatyyppeihin. Toiminnanohjausjärjestelmät on ohjelmoitu Microsoftin .NET-kehitysympäristössä ja myös uusi verkkorajapinta rakennettiin käyttäen .NET-kehitysympäristön työkaluja.

Uutta rajapintaa käytetään asiakkaan palvelimella toimivan ajastetun taustapalvelun kautta. Palvelu hakee ajastetusti rajapinnan kautta Timber by Pinja -ohjelmistoon syntyneet uudet sivutuotteet ja niiden määrät ja lisää ne Forest by Pinjassa määriteltyyn välivarastoon. Palvelu hakee myös Forest by Pinjan välivarastoista lähteneet kuljetusmääräykset, jotka vähentävät varastomäärää ja lähettää vähennettävät määrät Timber by Pinjaan. Rajapinta saatiin valmiiksi ja käyttöön asiakkaalle.

Avainsanat: ERP, toiminnanohjausjärjestelmä, C#, .NET, rajapinta, ohjelmointi

ABSTRACT

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

---

Author: Joonas Kinnunen
Title of thesis: Interface Development Between .NET Softwares
Supervisor: Janne Kumpuoja
Term and year when the thesis was submitted: spring 2024
Number of pages: 29 + 1 appendix

The thesis was done as an assignment for Pinja Digital Oy. In the thesis, a new interface was created between the Timber by Pinja and Forest by Pinja ERP systems developed by Pinja. Timber by Pinja is an enterprise resource planning system for sawmills and wood processing plants, and Forest by Pinja is an enterprise resource planning system developed for wood procurement, harvesting and logistics management. The need for the interface came from a customer who needed to synchronize the inventory balances of Timber by Pinja's warehouse and Forest by Pinja's warehouses.

The theory part of the thesis focuses on software interfaces, their uses and the most common interface types. The ERP systems are programmed in Microsoft's .NET development environment and the new web interface was also built using the tools of the .NET development environment.

The new interface is used via a scheduled background service running on the customer's server. The service fetches the new by-products created in the Timber by Pinja software and their quantities via the interface at a timed basis and adds them to the intermediate storage defined in Forest by Pinja. Service also retrieves transport orders from Forest by Pinja's intermediate warehouses, which reduce the stock quantity, and sends the reduced quantities to Timber by Pinja. The interface was completed and ready for use by the customer.

Keywords: ERP, enterprise resource planning system, C#, .Net, interface, programming

SISÄLLYS

1 JOHDANTO

Opinnäytetyö tehtiin toimeksiantona Pinja Digital Oy:n metsäteollisuuden tarpeisiin kehittämiin toiminnanohjausjärjestelmiin. Pinja on teollisuuden SaaS-palveluihin ja tekoälyyn erikoistunut yhtiö, jolla on asiakkaita yli 30 maassa.

Työn tavoitteena on toteuttaa Pinjan Forest- ja Timber by Pinja -toiminnanohjausjärjestelmien välille rajapinta, jonka avulla on mahdollista siirtää tietoa ohjelmistojen välillä muuttuneista varastomääristä. Timber by Pinja on sahojen ja puunjalostuslaitosten toiminnanohjausjärjestelmä ja Forest by Pinja on puunhankinnan, korjuun ja logistiikan hallinnoimiseen kehitetty toiminnanohjausjärjestelmä.

Uutta rajapintaa käytetään asiakkaan palvelimella toimivan ajastetun taustapalvelun kautta. Palvelu hakee ajastetusti rajapinnan kautta Timber by Pinja -ohjelmistoon syntyneet uudet sivutuotteet ja niiden määrät ja lisää ne Forest by Pinjassa määriteltyyn välivarastoon. Palvelu hakee myös Forest by Pinjan välivarastoista lähteneet kuljetusmääräykset, jotka vähentävät varastomäärää ja lähettää vähennettävät määrät Timber by Pinjaan. Rajapinnan tarkempi toteutus käydään läpi opinnäytetyön Rajapinnan toteutus -osiossa.

Toiminnanohjausjärjestelmät, joiden välille rajapinta rakennetaan, on molemmat kehitetty .NET-kehitysalustalla ja myös rajapinnan toteutus tehdään .NET-ympäristössä. Ohjelmointi tapahtuu Microsoftin Visual Studio -ohjelmistolla.

2 VERKKORAJAPINNAT

2.1 Rajapinnat ohjelmistokehityksessä

API tai ohjelmointirajapinta on joukko sääntöjä tai protokollia, joiden avulla ohjelmistosovellukset voivat kommunikoida keskenään tietojen, ominaisuuksien ja toimintojen vaihtamiseksi. Rajapinnat yksinkertaistavat sovelluskehitystä antamalla kehittäjille mahdollisuuden integroida tietoja, palveluita ja ominaisuuksia muista sovelluksista sen sijaan, että kehittäisivät niitä tyhjästä. Rajapinta toimii välittäjänä asiakasohjelmiston ja palvelimen välillä ja kapseloi palvelimen logiikan siten, että asiakasohjelmisto voi hyödyntää palvelimen tarjoamia toimintoja ilman, että sen tarvitsee ymmärtää palvelimen sisäistä toteutusta tai logiikkaa. (1.)

Kun asiakas lähettää pyynnön rajapinnan kautta, rajapinta käsittelee pyynnön, suorittaa tarvittavan logiikan palvelimen puolella, esimerkiksi tietokantahaun ja palauttaa tuloksen asiakkaalle esimerkiksi JSON- tai XML-muodossa. Tämä prosessi piilottaa palvelimen yksityiskohdat asiakkaalta, jolloin sovelluksen kehittäjä voi keskittyä sovelluksen logiikan rakentamiseen ilman tarvetta ymmärtää palvelimen sisäistä toimintaa.

Koska rajapintaa käyttävän kehittäjän ei tarvitse tietää, eikä hän voi tietää rajapinnan takana olevan palvelimen toiminnasta, rajapinnan dokumentointi on tärkeä osa rajapinnan rakentamista. Hyvä dokumentaatio auttaa kehittäjää ja kehittäjä voi tarkistaa dokumentaatiosta, miten rajapintaa käytetään, millaisia kutsuja rajapintaan voi lähettää ja minkälaisia vastauksia rajapinta palauttaa.

Rajapinnat ovat nykyisin olennainen osa ohjelmistoja. Yksi tärkeä rajapintatyyppi on internetin yli toimivat rajapinnat (Web API). Verkkorajapintojen avulla ohjelmistot voivat vaihtaa tietoa keskenään verkon välityksellä. Esimerkiksi säätietoja tarjoava yritys voi tarjota rajapinnan, jota kutsumalla ohjelmoija voi pyytää säätietoja, jotka hän voi näyttää omassa ohjelmassaan. Suosittu Google Maps -karttasovellus on hyvä esimerkki sovelluksesta, joka käyttää lukuisia verkkorajapintoja esimerkiksi julkisen liikenteen aikataulujen ja reittien hakemiseen.

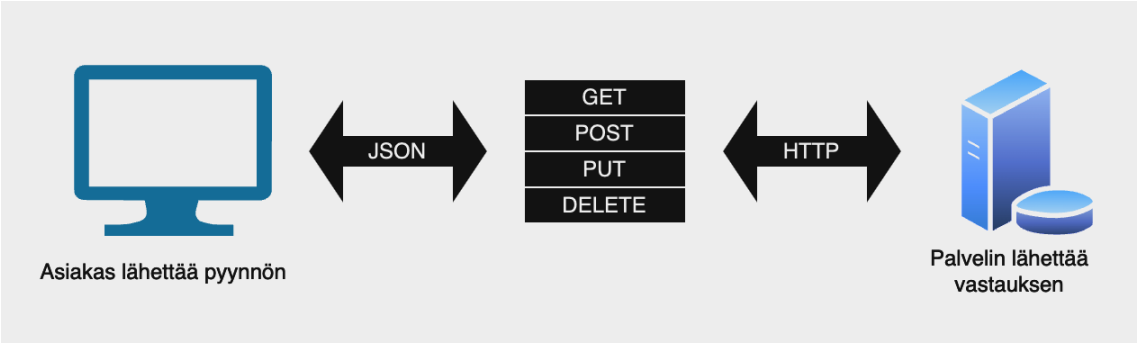
Verkkorajapinta voi olla avoin tai sen käyttö voi vaatia API-avaimen tai muun tunnistautumisen. Tunnistautumista tarvitaan erityisesti, kun rajapinta käsittelee arkaluontoisia, asiakaskohtaisia tietoja. Tunnistautumisella rajapinnan tarjoaja voi myös rajoittaa rajapinnan käyttöä tai veloittaa rajapinnan käytöstä.

Verkkorajapinnan tekniseen toteuttamiseen on olemassa useita eri tapoja. Rajapintatyyppien käyttö vaihtelee sovelluskehityksen tarpeiden ja tietoturvastandardien mukaan.

2.2 REST

Verkkorajapintatyypeistä käytetyin on REST (Representational State Transfer). REST on arkkitehtuurimalli, joka hyödyntää HTTP-protokollan standardeja operaatioita, kuten GET, POST, PUT ja DELETE resurssien hallintaan verkon yli. Se on suunniteltu olemaan kevyt, joustava ja helposti käytettävä, mikä tekee siitä suositun valinnan web-palveluiden ja mobiilisovellusten kehittämiseen. REST perustuu resurssien tilojen tai edustusten siirtämiseen asiakkaan ja palvelimen välillä, missä jokainen resurssi on yksilöitävissä URI:lla (Uniform Resource Identifier). REST-arkkitehtuurissa kommunikointi on tilatonta, mikä tarkoittaa, että jokainen pyyntö asiakkaalta palvelimelle sisältää kaiken tarvittavan tiedon pyynnön käsittelyyn, eikä palvelimen tarvitse säilyttää asiakkaan tilatietoja pyyntöjen välillä. (2.)

Kuvassa 1 kuvataan REST-arkkitehtuurimalli. Asiakas lähettää pyynnön HTTP-metodia käyttäen ja palvelin vastaa pyyntöön. Tieto liikkuu asiakkaan ja palvelimen välillä JSON-muodossa.



KUVA 1. REST-arkkitehtuuri

HTTP-protokollan standardioperaatiot lyhyesti:

- POST: Luo uuden resurssin palvelimelle.

- GET: Hakee resurssin tai resurssien tiedot palvelimelta.
- PUT: Päivittää olemassa olevan resurssin kokonaan uusilla tiedoilla.
- DELETE: Poistaa resurssin palvelimelta.

Muita vähemmän käytettyjä operaatioita ovat: PATCH, HEAD, OPTIONS, TRACE ja CONNECT.

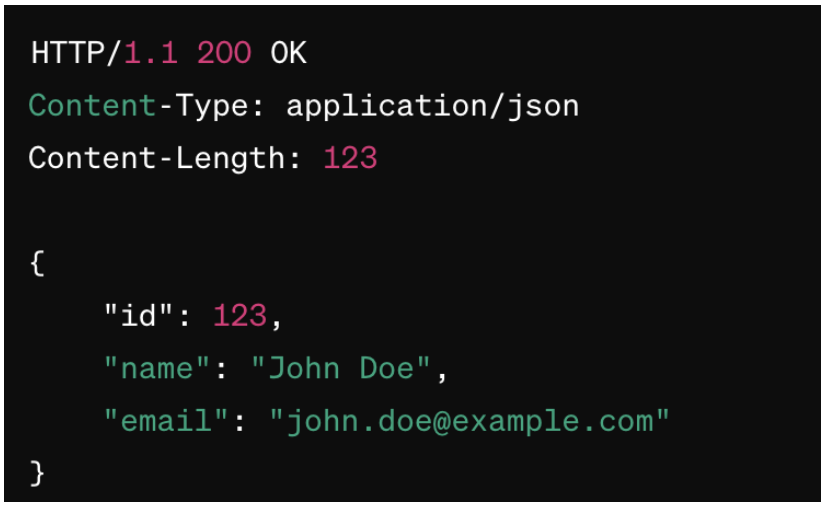
HTTP-protokollassa määritellään myös standardoidut vastauskoodit, joilla palvelin ilmoittaa asiakkaalle vastauksena pyyntöön pyynnön tilan. Vastauskoodit vaihtelevat pyynnön mukaan ja nämä koodit kertovat pyynnön onnistumisesta, virheistä tai muista tilanteista. Vastauskoodi palautetaan vastauksen otsaketiedoissa. Opinnäytetyön liitteenä (liite 1) on taulukko tyypillisimmistä vastauskoodeista.

Kuvassa 2 on tyypillinen GET-tyyppinen rajapintakutsu. Pyyntö kertoo palvelimelle, että asiakasohjelma haluaa hakea tietoja käyttäjästä, jonka ID on 123, ja odottaa vastauksen JSON-muodossa. Authorization-otsikossa välitetään tunnistautumistieto, jolla palvelin voi varmentaa, että kutsujalla on oikeus hakea käyttäjän tietoja.

```
GET /users/123 HTTP/1.1
Host: api.example.com
Accept: application/json
Authorization: Bearer your_access_token_here
```

KUVA 2. HTTP GET -rajapintakutsu

Kuvassa 3 palvelin palauttaa vastauksen koodilla 200, joka kertoo, että pyynnön käsittely onnistui. Vastauksen mukana palautetaan pyydetyn käyttäjän tiedot JSON-muodossa.



KUVA 3. HTTP-vastaus koodilla 200 OK

Esimerkiksi verkkokauppaa selatessa, kun käyttäjä avaa tuotelistauksen, verkkosivu voi lähettää REST-rajapinnan kautta palvelimelle GET-tyyppisen pyynnön polkuun */api/products*. Palvelin palauttaa vastauksen otsaketiedoissa koodin 200 merkkinä onnistuneesta pyynnöstä ja viestiosiossa JSON muotoisen tuotelistauksen, joka sisältää tuotteiden tiedot, hinnat ja muut tarpeelliset tiedot ja tuotelistaus näytetään verkkosivulla käyttäjälle. Tuotelistaus sisältää vain tuotelistauksessa tarvittavat tiedot. Kun käyttäjä klikkaa tuotelistauksesta tuotetta, näytetään verkkosivulla tuotteen tarkemmat tiedot, kuten pidempi tuotekuvaus. Yksittäisen tuotteen tietojen hakua varten palvelimelle lähetetään GET-pyyntö polkuun */api/products/123*.

Kun käyttäjä täyttää tietonsa verkkokauppaan ja lähettää rekisteröintilomakkeen, verkkosivu lähettää lomakkeen tiedot, kuten esimerkiksi sähköpostiosoitteen, salasanan ja postiosoitteen POST-tyyppisen pyynnön mukana palvelimen polkuun */api/users*. Palvelin tallentaa vastaanottamansa tiedot tietokantaan ja lähettää vastauksena tiedon rekisteröinnin onnistumisesta. Jos vastaus onnistui, vastauksen otsakkeessa palautetaan koodi 201 (Created). Jos pyyntö on virheellinen, eikä rekisteröinti onnistunut, vastauksessa palautetaan koodi 400 (Bad Request). Jos vastauskoodi on 201, käyttäjälle näytetään ilmoitus siitä, että rekisteröinti onnistui. Muussa tapauksessa näytetään virheilmoitus.

Jos käyttäjä haluaa päivittää aiemmin rekisteröinnin yhteydessä antamiaan tietoja, kuten postiosoitetta, voi hän kirjautua verkkokauppaan ja syöttää päivitetyt tiedot. Päivitetyt tiedot lähetetään HTTP PUT-tyyppisenä pyyntönä polkuun *api/users/123*. Pyynnön mukana lähetetään käyttäjän

tunnistautumistiedot, joilla palvelin pystyy varmentamaan, että käyttäjällä on oikeus päivittää tietoja. Vastaavasti kuin rekisteröinnin yhteydessä, palvelin palauttaa myös PUT-pyyntöön vastauksena tiedon tietojen päivityksen onnistumisesta. Onnistuneesta päivityksestä palautetaan koodi 200 (OK). Jos pyynnössä yritetään päivittää tietoja, joihin käyttäjällä ei ole päivitysoikeutta, palautetaan koodi 403 (Forbidden).

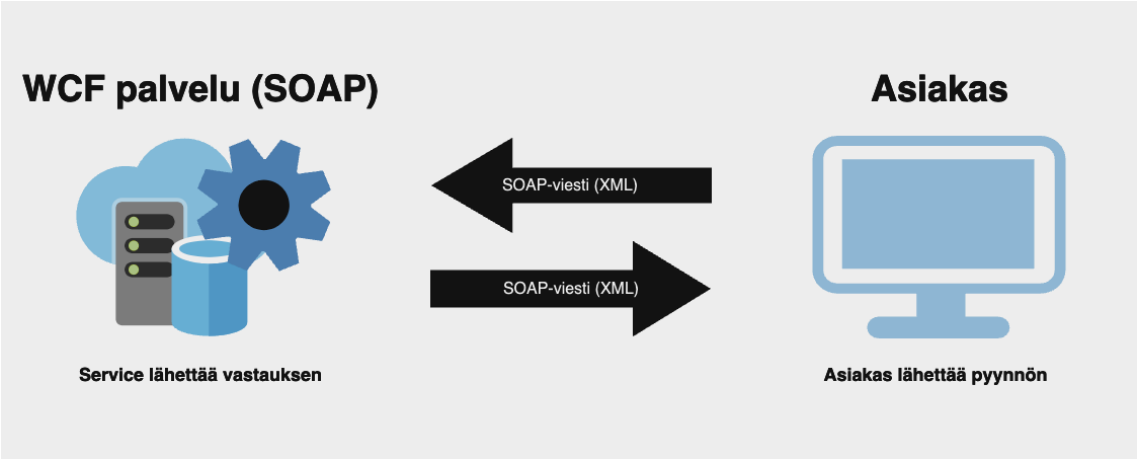
Kun käyttäjä haluaa poistaa käyttäjätilinsä kokonaan ja painaa verkkokaupan asiakastilillään Poista-painiketta, lähetetään palvelimelle DELETE-tyyppinen pyyntö polkuun *api/users/123*. Palvelin poistaa käyttäjän tiedot tietokannasta ja lähettää vastauksena tiedon poiston onnistumisesta. Onnistuneesta poistosta palautetaan koodi 200 (OK) ja esimerkiksi jos poistettavaa käyttäjää ei löydy, palautetaan koodi 404 (Not Found).

2.3 SOAP

SOAP (Simple Object Access Protocol) on protokolla, jonka kautta voidaan vaihtaa viestejä tietokoneverkoissa käyttäen XML-muotoa. SOAP tukee monimutkaisempaa viestintäkuviota ja tarjoaa korkeamman turvallisuustason verrattuna RESTiin. SOAP-rajapinnat ovat usein hyvä valinta yrityssovelluksissa, jotka vaativat tiukkaa tietoturvaa ja transaktioiden eheyttä.

SOAPin tärkeimpiä eroja verrattuna RESTiin on se, että SOAP-viesteissä käytetään ainoastaan XML-muotoa, kun taas REST tukee myös esimerkiksi JSON- ja HTML-muotoja. REST tukee ainoastaan HTTP-siirtoprotokollaa, kun taas SOAP tukee lisäksi myös esimerkiksi FTP-, TCP- ja SMTP-protokollia.

Kuvassa 4 on kuvattu SOAP-rajapinta, jossa asiakas lähettää pyynnön palvelimelle ja palvelin vastaa pyyntöön. Asiakas ja palvelin keskustelevat SOAP-viesteillä.

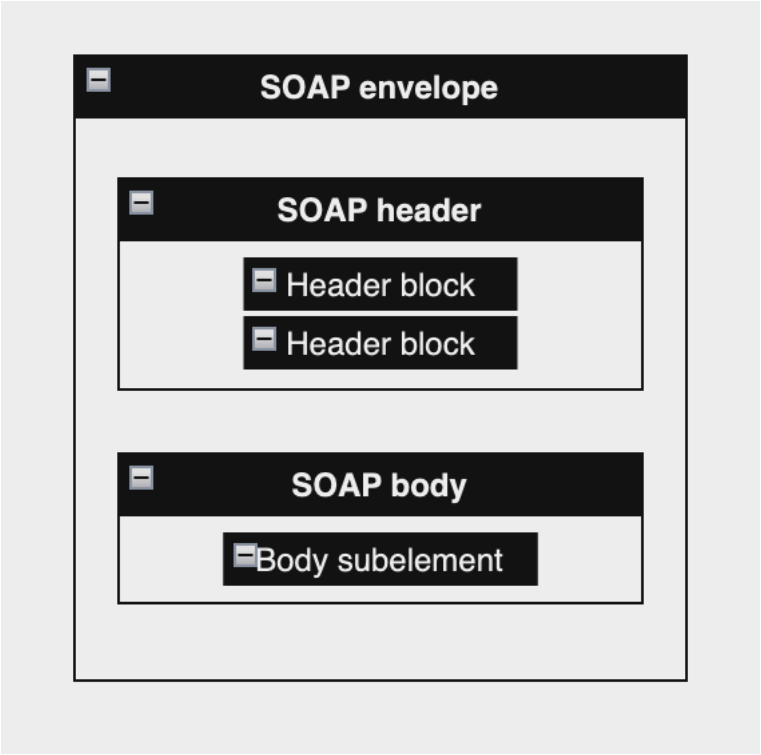


KUVA 4. WCF SOAP -rajapinta

SOAP-rajapinnassa liikkuvan XML-sanoman rakenne ja tietotyypit on tarkasti määritelty XML-skeemassa. Skeeman avulla sanoman tiedot ja rakenne on helppo varmentaa, mikä vähentää mahdollisten virheiden riskiä viestinnässä. Skeemassa voi olla määriteltynä sisäänrakennettujen datatyyppien, kuten esimerkiksi integer, float ja date lisäksi myös käyttäjän määrittelemiä omia tietotyyppejä.

SOAP-viestin XML:n rakenne koostuu envelope-, header-, body- ja fault-elementeistä. Viestin rakenne on kuvattu kuvassa 5.

Envelope on viestin pääelementti, joka sisältää pakollisen body-elementin ja valinnaisen header-elementin. Body-elementti sisältää varsinaisen sanoman, eli viestin joka vastaanottajalle välitetään. Valinnainen header-elementti voi sisältää useita alitietueita, jotka voivat sisältää esimerkiksi digitaalisen allekirjoituksen salasanalla suojattuun palveluun tai muita sovelluskohtaisia tietoja. Fault on valinnainen body-elementin alielementti, jota käytetään virheiden raportoinnissa. (3.)



KUVA 5. SOAP XML-sanoman rakenne

Tässä opinnäytetyössä rajapinta rakennettiin käyttämällä WCF-palveluja, jotka käyttävät SOAP-protokollaa. .NET ympäristöön kuuluva Windows Communication Foundation (WCF) on ohjelmistokehys, joka mahdollistaa eri sovellusten välisen kommunikaation erityisesti Windows-pohjaisissa järjestelmissä. WCF:n avulla voidaan luoda palvelukeskeiseen arkkitehtuuriin perustuvia sovelluksia. WCF-palveluiden avulla voidaan lähettää viestejä palvelun ja sitä käyttävän asiakkaan välillä.

.NET-ympäristössä viittaus WCF-palveluun on helppo lisätä esimerkiksi Windows Forms -ohjelmaan suoraan Visual Studio -kehitysympäristöstä. Visual Studio hakee palvelun vaatiman sanoman rakenteen ja tietotyypit. Kun viittaus palveluun on lisätty ohjelmaan, palvelua on helppo kutsua suoraan ohjelman koodista. Kuvan 6 esimerkkikoodissa kutsutaan Windows Forms -ohjelman koodissa MessagingClient-nimistä WCF-palvelua. Lähettävä viesti ja vastaus käyttää palvelussa määriteltyjä Message- ja MessagingClientResponse-luokkia.

```
// Create a client
MessagingClient client = new MessagingClient();

// Call the client
Message message = new Message("Hello", DateTime.Now());
MessagingClientResponse result = client.SendMessage(message);
Console.WriteLine(result.message);
```

KUVA 6. WCF-palvelun kutsu

2.4 GraphQL

GraphQL on alun perin Facebookin kehittämä, nykyisin avoimen lähdekoodin kyselykieli. GraphQL eroaa esimerkiksi RESTistä siten, että REST on resurssipohjainen ja jokaisella resurssilla on oma identifioiva osoite. Restissä osoitteille tehdään pyyntöjä käyttämällä HTTP-metodeja (GET, POST, PUT, DELETE) ja niiden toiminta määrittyy käytetyn HTTP-metodin avulla. GraphQL:ssä kaikki kyselyt lähetetään tyypillisesti POST-metodilla ja kohdistetaan samaan URL-osoitteeseen.

GraphQL:ssä määritellään tarkasti, mitä tietoja halutaan hakea tai muuttaa. Koska palvelin palauttaa vain pyydetyt tiedot, turha datan siirto vähenee ja ohjelman suorituskyky voi parantua. Jos haettavia tietoja on paljon, asiakasohjelman pitäisi tehdä useita kyselyitä REST-rajapinnan eri polkuihin, mutta GraphQL:ssä voidaan hakea kaikki tarvittavat tiedot kerralla. Toisaalta REST-rajapintaan tehty kutsu voi palauttaa myös tietoja, joita asiakasohjelma ei tarvitse. Kääntöpuolena GraphQL:n tarkasti rajatuissa kyselyissä on se, että kyselyistä voi tulla todella monimutkaisia ja pitkiä. Vaikka GraphQL:n yhtenä tärkeimmistä ajatuksista on vähentää turhaa datan siirtoa, huonosti ja epätarkasti tehty kysely voi johtaa pahimmillaan valtavaan datan määrään, joka lisää tiedonsiirtokuluja ja kuormittaa palvelinta. Huonosti suunniteltu GraphQL-rajapinta voi myös aiheuttaa turvallisuusriskejä. Jos käyttäjäoikeudet datan hakuun on huonosti rajattu, hyökkääjä voi kuormittaa palvelinta tekemällä rajapintaan mahdollisimman laajoja kyselyitä, mikä voi johtaa palvelimen kaatumiseen.

GraphQL:n yleistymistä on hidastanut se, että se vaatii esimerkiksi RESTiä monimutkaisemman palvelinpuolen toteutuksen. Lisäksi sen oppiminen vie yleensä enemmän aikaa ja vaatii enemmän opiskelua RESTiin verrattuna.

Alla olevassa kuvassa 7 on esimerkki kyselystä, jossa haetaan henkilön tietoja GraphQL:n kautta. Kyselyssä haetaan käyttäjän tiedot ID:llä 123. Kyselyssä on määritelty, että vastauksena halutaan henkilön id, nimi, ikä ja sähköpostiosoite. Kyselyssä query määrittää, että tehdään tietojen haku. GraphQL:ssä tietojen muuttaminen tai lisääminen tehdään mutaatioiden avulla ja kyselyssä query korvattaisiin sanalla mutation.

```
query {
  person(id: "123") {
    id
    name
    age
    email
  }
}
```

KUVA 7. GraphQL-rajapinnan kysely

Käyttäjällä voi olla pyydettyjen tietojen lisäksi myös monia muita tietoja, mutta palvelin palauttaa vastauksen, joka sisältää vain pyydetyt tiedot. Vastaus onnistuneeseen pyyntöön sisältää aina tieto-olion, joka sisältää vastauksena lähetetyt tiedot. GraphQL-rajapinta palauttaa aina vastauksena HTTP-koodin 200 OK, vaikka kysely sisältäisi virheitä. Jos kyselyssä on virheitä, vastauksena palautetaan rajapinnan toteutuksesta riippuen tieto-olion sijaan tai lisäksi virheolio, joka sisältää luettelon kyselyn virheistä. Kuvassa 8 on esimerkki vastauksesta onnistuneeseen kyselyyn.

```
{
  "data": {
    "person": {
      "id": "123",
      "name": "John Doe",
      "age": 34,
      "email": "johndoe@example.com"
    }
  }
}
```

KUVA 8. GraphQL-rajapinnan vastaus

3 TYÖKALUT JA TEKNIIKAT WINDOWS-YMPÄRISTÖSSÄ

Rajapinta toteutettiin .NET-ympäristössä kehitettyjen toiminnanohjausjärjestelmien välille, joten myös rajapinnan kehitys oli luonnollista tehdä .NET-kehitysalustan työkaluilla. .NET on alun perin Microsoftin kehittämä ohjelmistokehityskehys, joka tukee useita ohjelmointikieliä, kuten C# ja Visual Basic. .NET on nykyisin avoimen lähdekoodin kirjasto ja sen koodia ylläpitää Microsoftin lisäksi vapaaehtoisten yhteisö. .NET-kirjaston avulla on mahdollista kehittää esimerkiksi Windows-, mobiili-, web- ja konsoliohjelmistoja. Kirjastoa käytetään ohjelmistojen kehityksessä laajasti ympäri maailmaa ja se sisältää useita eri työkaluja ja tekniikoita erityyppisten ohjelmistojen rakentamiseen.

Ohjelmointi tehtiin Microsoftin Visual Studio ohjelmistolla ja ohjelmointikielenä oli pääasiassa C# ja pieniä osia koodista toteutettiin VB.Net-kielellä. Rajapinnan varsinainen toteutus tehtiin WCF-palveluun, joka on yhteydessä asiakkaan tietokantaan.

Visual Studio -kehitysympäristö sisältää monia ohjelmointia helpottavia työkaluja, kuten koodin täydennystyökalu IntelliSensen. Visual Studioon on saatavilla myös runsaasti lisäosia. Esimerkkinä ohjelmoinnissa käytettiin apuna GitHub Copilot -tekoälyohjelmoijaa, joka oli lisätty lisäosana Visual Studioon.

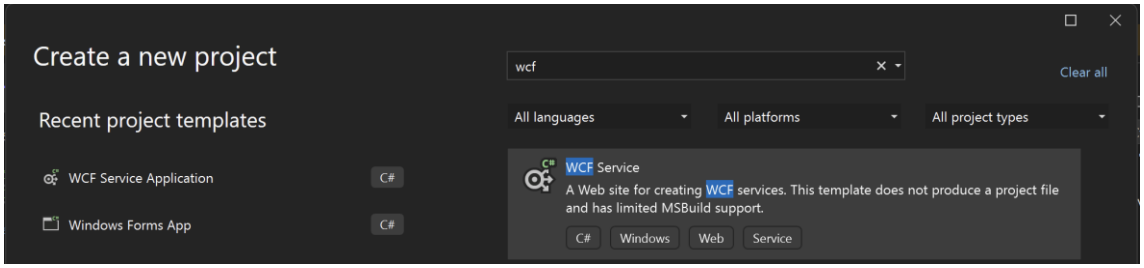
Rajapintaa käytetään asiakkaan palvelimella toimivan palvelun (Windows Service Application) avulla, joka ottaa yhteyttä WCF-palveluun. Palvelu toimii käyttöjärjestelmän taustalla, eikä vaadi käyttäjän sisäänkirjautumista.

Tietokannat toimivat SQL Serverillä ja niiden hallintaan käytettiin SQL Server Management Studio (SSMS) -ohjelmistoa. SSMS-ohjelmiston avulla eri palvelimilla sijaitsevien tietokantojen hallinta yhdellä ohjelmistolla oli helppoa.

SQL Server Management Studio (SSMS) on integroitu ympäristö minkä tahansa SQL-infrastruktuurin hallintaan. SSMS tarjoaa yhden kattavan apuohjelman, joka yhdistää laajan joukon graafisia työkaluja moniin monipuolisiin skriptityökaluihin ja tarjoaa pääsyn SQL Serveriin kaiken tasoisille kehittäjille ja tietokannan ylläpitäjille. (4.)

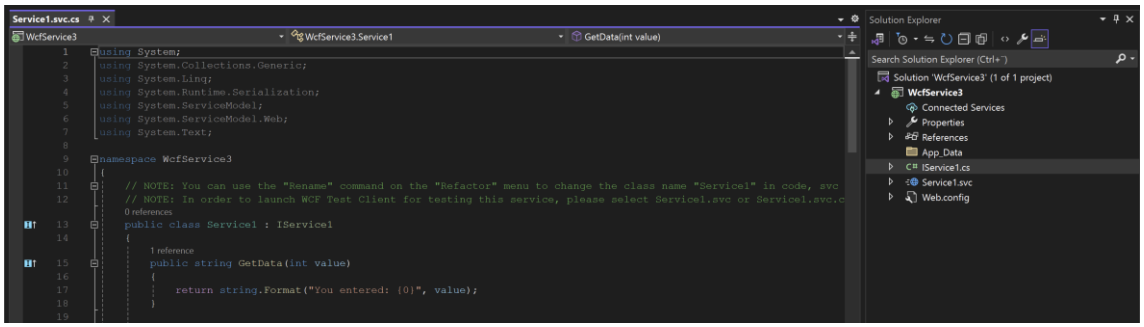
3.1 WCF-palvelu

Uusi WCF-palvelu on helppo perustaa Visual Studio-kehitysympäristössä. Uusi palvelu perustetaan valitsemalla valikosta uusi projekti ja tämän jälkeen WCF-palvelun tyyppi (kuva 9).



KUVA 9. WCF-projektin luonti

Visual Studio luo automaattisesti WCF-palvelun käynnistämiseen tarvittavat tiedostot. Projektipohjassa on valmiina esimerkit palvelusopimuksen, operaatioiden ja tietosopimuksen käytöstä. Kuvassa 10 on Visual Studion luoma esimerkkipalvelu, joka sisältää GetData()-operaation. Sopimuksista ja operaatioista kerrotaan lisää alempana.



KUVA 10. WCF esimerkkiprojekti

Uudelle palvelulle on määriteltävä palvelusopimus (Service Contract), mikä kuvaa, mitä toimintoja palvelu tarjoaa asiakkaille, eli mitä operaatioita (Operation Contracts) voidaan kutsua ja millaisia viestejä (Data Contracts) nämä operaatiot käyttävät. Alla olevassa esimerkkikuvassa (kuva 11) on määritelty IMath niminen palvelusopimus, jolla on Add()- ja Multiply()-operaatiot. Operaatiot toteutetaan MathService-luokassa.

```C#
// Define the IMath contract.
[ServiceContract]
public interface IMath
{
    [OperationContract]
    double Add(double A, double B);

    [OperationContract]
    double Multiply (double A, double B);
}

// Implement the IMath contract in the MathService class.
public class MathService : IMath
{
    public double Add (double A, double B) { return A + B; }
    public double Multiply (double A, double B) { return A * B; }
}
```

KUVA 11. WCF-palvelusopimus

Tietosopimus on muodollinen sopimus palvelun ja asiakkaan välillä, jossa kuvataan abstraktisti vaihdettavat tiedot, eli kommunikoidakseen asiakkaan ja palvelun ei tarvitse jakaa samoja tyyppejä, vain samat datasopimukset. Datasopimus määrittelee tarkasti kullekin parametrille tai palautustyypille, mitä dataa sarjoitetaan (muutetaan XML:ksi) vaihdettavaksi (5).

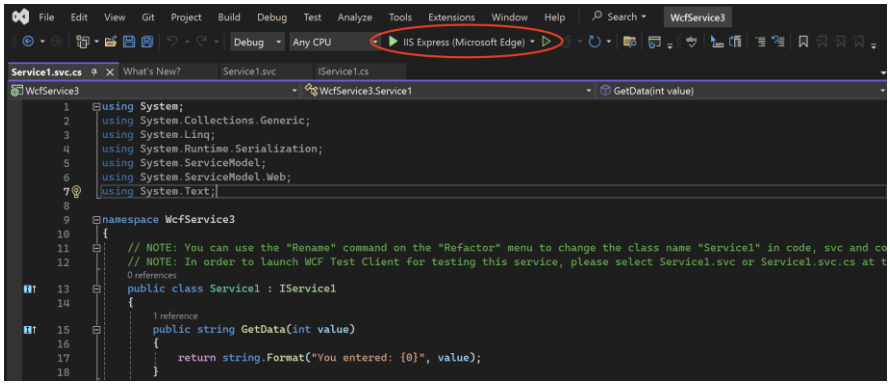
WCF-palvelu osaa muuttaa .NET-primitiivityypit, kuten esimerkiksi kokonaisluvut ja liukuluvut XML-muotoon ilman datasopimusta, mutta itsemääritellyille tyypeille on määriteltävä datasopimukset. Kuvassa 12 on esimerkki itse määritellystä datasopimuksesta.

```C#
namespace MyTypes
{
    [DataContract]
    public class PurchaseOrder
    {
        private int poId_value;

        // Apply the DataMemberAttribute to the property.
        [DataMember]
        public int PurchaseOrderId
        {

            get { return poId_value; }
            set { poId_value = value; }
        }
    }
}
```

KUVA 12. WCF-datasopimus

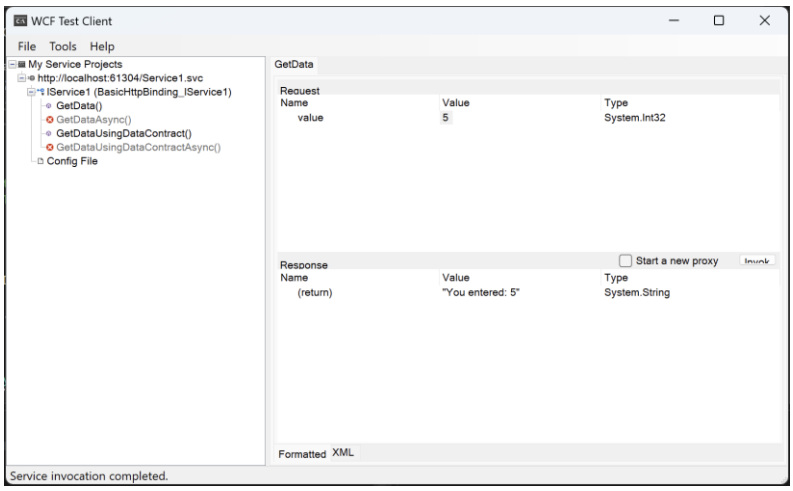
Uusi palvelu halutaan yleensä verkkoon, jotta sen asiakkailla on pääsy siihen. Tyypillisesti WCF-palvelu toimii esimerkiksi IIS-palvelimella. Internet Information Services (IIS) Windows® Serverille on joustava, turvallinen ja hallittava Web-palvelin kaiken web-palvelun isännöintiin. Median suoratoistosta verkkosovelluksiin, IIS:n skaalautuva ja avoin arkkitehtuuri on valmis käsittelemään vaativimpiakin tehtäviä. (6.)

WCF-palvelu voidaan käynnistää myös paikallisesti suoraan Visual Studiosta IIS Expressin avulla painamalla käynnistä-painiketta yläpalkissa (kuva 13). Tästä on hyötyä kehitystyön aikana, koska palvelua on helppo kutsua ja testata paikallisessa verkossa.



KUVA 13. WCF-palvelun käynnistäminen

WCF Test Client näyttää ikkunan, josta selviää palvelun osoite ja muita hyödyllisiä tietoja. Palvelun operaatioita on mahdollista testata suoraan ikkunassa syöttämällä pyyntö operaatiolle. Kuvassa 14 on annettu GetData()-operaatiolle arvo 5 ja palvelu vastasi ”You entered 5”.



KUVA 14. WCF-palvelun testaaminen

3.2 Windows-palvelusovellus

Microsoft Windows -palvelut, jotka tunnettiin aiemmin nimellä NT-palvelut, ovat palveluita, joilla on mahdollista luoda pitkäkestoisia suoritettavia sovelluksia, jotka toimivat omissa Windows-istunnoissaan. Palvelut voidaan käynnistää automaattisesti, kun tietokone käynnistyy, ne voidaan keskeyttää ja käynnistää uudelleen, eivätkä ne näytä käyttöliittymää. Koska palvelut toimivat taustalla, eivätkä häiritse tietokoneen käyttäjää, ne sopivat hyvin käytettäviksi palvelimella tai muulla tietokoneella, jossa tarvitaan pitkäkestoisia taustatoimintoja. (7.)

Windows-palvelusovelluksia käytetään usein esimerkiksi tiedostojen varmuuskopioinnissa. Palvelu voi varmuuskopioida tietokoneen tiedostot ajastetusti etäpalvelimelle. Palvelusovellus voi myös toimia eräänlaisena järjestelmänvalvojana tarkkaillen järjestelmän suorituskykyä ja keräten lokitietoja. Palvelu voi automaattisesti ilmoittaa järjestelmänvalvojille, jos se havaitsee epätavallista toimintaa tai virheitä järjestelmän lokitiedoissa.

Tässä opinnäytetyössä palvelusovellusta käytetään kutsumaan funktioita, jotka puolestaan kutsuvat WCF-palvelua. Funktiot synkronoivat varastosaldot kahden ohjelmiston välillä. Palvelusovellus sopii hyvin tähän tarkoitukseen, koska synkronoinnin pitää tapahtua ajastetusti ja ilman käyttäjän toimenpiteitä.

Windows-palvelusovelluksia on mahdollista kehittää esimerkiksi Visual Studio-kehitysympäristössä. Kehitys poikkeaa monien muiden sovellusten kehityksestä esimerkiksi siten, että virheenjäljitys ei suoraan toimi kuten yleensä. Sovelluksesta on ensin luotava suoritettava tiedosto, joka täytyy asentaa. Asentamisen jälkeen palvelu on vielä käynnistettävä, jonka jälkeen palvelu voidaan liittää virheenjäljitykseen Visual Studiosta.

4 RAJAPINNAN TOTEUTUS

Rajapinnan kehittämisen lähtökohtana oli Pinja Digital Oy:n asiakkaan tarve synkronoida asiakkaan käyttämien, Pinjan kehittämien toiminnanohjausjärjestelmien varastosaldot. Asiakkaalla on käytössään Pinjan kehittämät Timber- ja Forest by Pinja -ohjelmistot. Timber by Pinja on sahojen ja puunjalostuslaitosten toiminnanohjausjärjestelmä ja Forest by Pinja on puunhankinnan, korjuun ja logistiikan hallinnoimiseen kehitetty toiminnanohjausjärjestelmä. Timber by Pinja ohjelmiston sivutuotevarastoon syntyy sahalla sivutuotteena esimerkiksi purua ja muita sivutuotteita, joiden varastolisäykset asiakas haluaa synkronoida Forest by Pinjan operatiiviseen välivarastoon. Vastaavasti Forest by Pinjan välivarastosta ajetut kuormat vähentävät varastosaldoa ja näiden varastosaldon vähennysten haluttiin vähentävän myös Timber by Pinjan sivutuotevaraston varastomääriä. Varastosaldojen muutokset synkronoidaan automaattisesti ilman ohjelmien käyttäjän toimenpiteitä.

4.1 Suunnittelu

Suunnittelun pohjana oli asiakkaan kanssa laadittu vaatimusmäärittely dokumentti, jossa määriteltiin uudelle toiminnallisuudelle keskeiset vaatimukset ja ominaisuudet. Toteutettavassa rajapinnassa tieto liikkuu kahteen suuntaan Forest- ja Timber by Pinja -ohjelmistojen välillä. Rajapinnan tekninen toteutus tehdään WCF-palveluihin, jotka ovat yhteydessä asiakkaan Forest- ja Timber by Pinja -ohjelmistoihin. Rajapintaan lähetettävä pyyntö voi olla joko varastotapahtumien haku tai varastotapahtuman lisäys.

Varastotapahtumien hakupyynnössä on oltava mukana aikaleima, josta lähtien varaston tapahtumat haetaan. Timber by Pinjan vastaussanomassa on oltava ainakin varastonumero, puutavaralaji ja aikaleima. Uudet varastotapahtumat kohdistetaan Forest by Pinjan varastoon puutavaralajin ja varastonumeron perusteella. Vastaavasti kun Forest by Pinjasta lähetetään pyyntö varastotapahtuman lisäyksestä, sanomassa on välitettävä aikaleima, puutavaralaji ja varastonumero.

Kaikki rajapinnan kautta kulkevat pyynnöt ja vastaukset on tallennettava tietokantaan (lokitus), jotta esimerkiksi mahdolliset virhetilanteet ovat jälkikäteen helpommin jäljitettävissä. Lokituksessa on oltava mukana myös aikaleima.

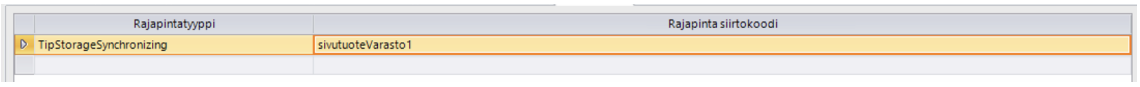
Forest by Pinjaan lisätään konfigurointityökalut, joiden avulla asiakas voi määrittää linkitykset Forest by Pinjan ja Timber by Pinjan välillä synkronoitavien varastojen ja puutavaralajien välille. Varastotapahtumat haetaan ja lähetetään automaattisesti ja ajastetusti asiakkaan palvelimelle lisättävän palvelun (Windows Service Application) avulla. Palvelu tallentaa edellisen varastotapahtumien haku- ja lisäysajan tietokantaan. Palvelu käyttää aikaleimaa varastotapahtumien haussa, jolloin tapahtumat haetaan vain aikaleiman jälkeiseltä ajalta.

Toiminnon toteuttaminen vaati ohjelmointia kolmeen eri WCF-palveluun. Asiakkaan palvelimella olevia Forest by Pinja ja Timber by Pinjan tietokantoja vasten toimii omat WCF-palvelut, jotka keskustelevat Pinjan palvelimella olevan WCF-palvelun kautta. Lisäksi varsinainen varastosiirtojen toimintalogiikka ohjelmoitiin Forest By Pinja-ohjelmistoon, jonne lisättyjä funktioita kutsuttiin asiakkaan palvelimella toimivasta Windows-palvelusta.

4.2 Konfigurointi

Forest by Pinjan ja Timber by Pinjan varastot on määritelty itsenäisesti kummassakin ohjelmistossa. Myös puutavaralajit ovat molemmissa ohjelmistoissa käyttäjän itse määriteltävissä. Varastomäärien siirtoa varten tarvittiin konfigurointityökalut, joilla käyttäjä voi määritellä linkitykset Timber by Pinjan ja Forest by Pinjan varastojen ja puutavaralajien välille.

Forest by Pinjan puutavaralajirekisteriin lisättiin puutavaralajin tietojen taakse mahdollisuus lisätä puutavaralajia vastaava Timber by Pinjan puutavaralajikoodi ja varastopaikkarekisteriin varastoa vastaava Timber by Pinjan varaston varastonumero. Lisäksi varastoille lisättiin valinta, jolla varastosynkronointi kytketään päälle tai pois. Varastomäärien muutosten noudossa ja lähetyksissä käytetään määriteltyjä puutavaralajien koodeja ja varastonumeroita yhdistämään varastot ja puutavaralajit ohjelmistojen välillä. Kuvassa 15 on esimerkki varastonumeron lisäämisestä.



KUVA 15. Varaston rajapintasiirtokoodin määrittäminen

Forest by Pinjan tietokannassa oli valmiina taulu puutavaralajien linkitystä varten. Tietokantaan lisättiin vastaava taulu varastopaikoille. Lisäksi tietokannan varastotauluun lisättiin bittityyppinen kenttä, johon tallennetaan tieto siitä, onko varastosynkronointi kytketty päälle varastolle.

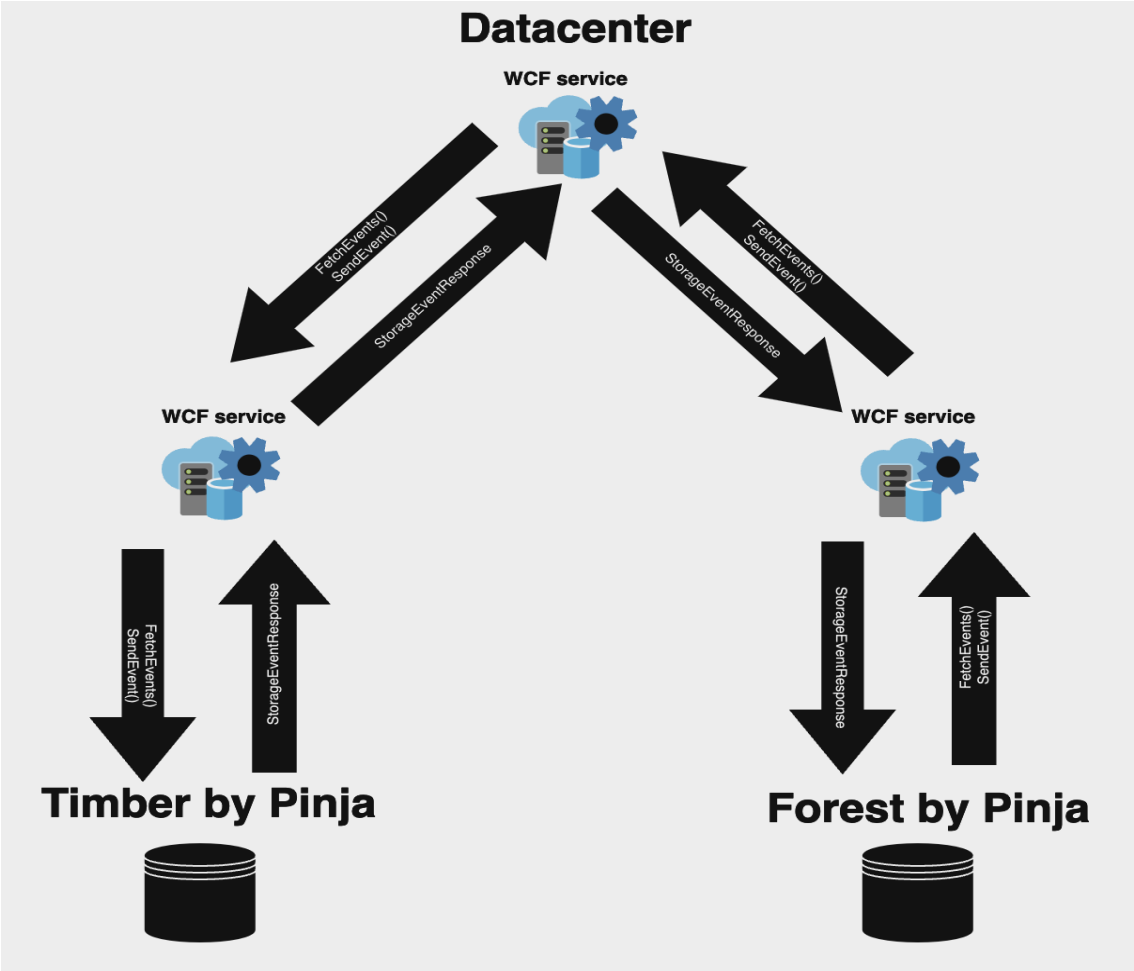
Ohjelmistojen välillä siirrettävien, linkitettyjen puutavaralajien varastomäärien yksiköt voivat vaihdella. Esimerkiksi Timber by Pinjassa puutavaralajin yksikkö voi olla irtokuutiometri (i-m³) ja linkitetyn Forest by Pinjan puutavaralajin yksikkö kiintokuutiometri (m³). Määrät siirretään rajapinnassa kiintokuutiometreinä, joten käyttäjän oli määriteltävä Timber by Pinjan puutavaralajien yksiköille muuntokertoimet, joilla määrä muunnetaan kiintokuutioiksi. Puutavaralajeilla oli valmiiksi olemassa mahdollisuus määritellä muuntokertoimet eri yksiköiden välillä.

4.3 Rajapinta

Ohjelmistot keskustelevat keskenään rajapintojen kautta, jotka toteutettiin kolmeen eri WCF-palveluun. Rajapinnan lisääminen kuhunkin palveluun tapahtui teknisesti samalla tavalla. Kuhunkin palveluun lisättiin kaksi toimintoa, toinen varastotapahtumien haulle ja toinen lähetykselle.

Forest by Pinjan ja Datacenterin WCF-palveluihin lisättiin viittaukset toiseen kutsuttavaan WCF-palveluun. Kun Visual Studiossa lisättiin viittaus toiseen WCF-palveluun, tuotiin tarvittavat tietotyypit ja operaatiot automaattisesti ohjelmaan. Kun tietotyypit ja operaatiot on tuotu, palvelua oli helppo kutsua ohjelman koodista. Kutsuvan asiakkaan autentikoinnin jälkeen sanoma välitetään eteenpäin seuraavaan palveluun. Timber by Pinjan WCF-palveluun toteutettiin varsinaiset tietokantakyselyt, joilla sivutuotevaraston tapahtumat haetaan ja uudet varastolisäykset lisätään.

Asiakkaan WCF-palvelu lähettää kutsun edelleen Pinjan palvelimella olevaan datacenterin WCF-palveluun. Datacenterin palvelu autentikoi kutsuvan asiakkaan ja ottaa yhteyttä datacenterin tietokantaan tallennetussa osoitteessa toimivaan asiakkaan Timber by Pinjan WC-palveluun. Timber by Pinjan palvelu vastaa pyyntöön lähettämällä vastaussanoman, joka sisältää pyynnöstä riippuen joko pyynnössä määritellyn varaston varastotapahtumat tai kuittauksen onnistuneesta varastotapahtuman lisäyksestä. Ohjelmistojen rakenne on kuvattu kuvassa 16.

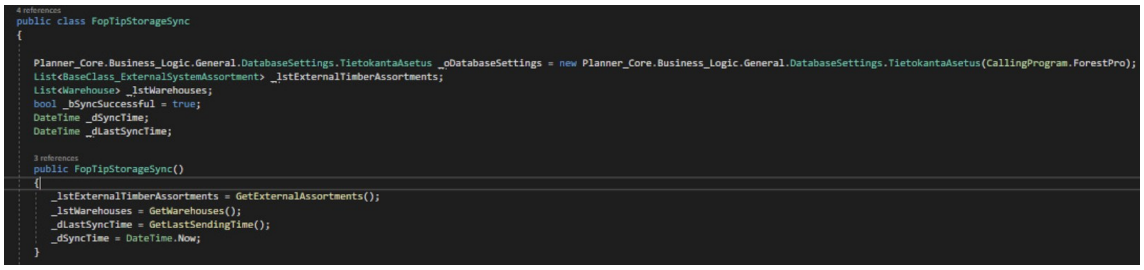


KUVA 16. Ohjelmistojen rakenne

4.4 Varastomäärien muutosten nouto ja lähetys

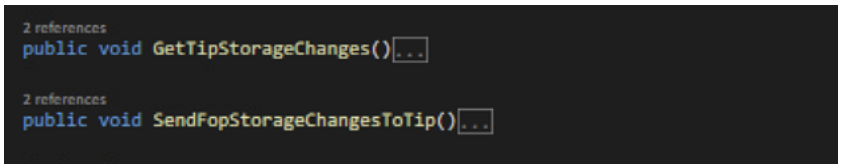
Rajapintojen toteutuksen jälkeen Forest by Pinjaan lisättiin viittaus WCF-palveluun. Visual Studio toi viittauksen lisäyksen yhteydessä palvelun kutsussa tarvittavat tyypit ohjelmaan. Forest by Pinjaan lisättiin uusi FopTipStorageSync-luokka, johon lisättiin varastosiirrossa tarvittavat toiminnallisuudet. Luokkaan lisättiin viittaus Forest by Pinjan WCF-palveluun.

Luokan alustuksessa haetaan siirtoa varten tarvittavat perustiedot, kuten edellinen synkronointiaika, synkronoitavat varastot, puutavaralajit ja niiden linkitykset. Tiedot haetaan luokkaan lisätyissä omissa funktioissaan (kuva 17).



KUVA 17. FopTipStorageSync-luokka

Varsinaista varastojen synkronointia varten lisättiin kaksi funktiota (kuva 18), GetTipStorageChanges() ja SendFopStorageChangesToTip(). Funktiot lähettävät pyynnön asiakkaan Forest by Pinjan-tietokantaan yhteydessä olevaan WCF-palveluun.



KUVA 18. Funktiot FopTipStorageSync-luokassa

Funktiossa GetSideProductChanges() haetaan Timber by Pinjan sivutuotevarastoon syntyneet varastolisäykset. Sivutuotevarastoon syntyy varastosaldon lisäyksiä esimerkiksi sahan sivutuotteena.

Forest by Pinjan tietokantaa vasten toimivasta Windows-palvelusovelluksesta lähetetään WCF-palveluiden kautta Timber by Pinjaan XML-muotoinen pyyntö, joka sisältää aikaleiman, jonka jälkeiseltä ajalta varastomäärien muutokset noudetaan. Aikaleima haetaan Forest by Pinjan tietokannasta, johon se tallennetaan jokaisen onnistuneen varastomäärien muutosten noudon jälkeen. Timber by Pinja palauttaa XML-sanoman, joka sisältää varastomäärien muutokset varastoittain ja puutavaralajeittain. Sanomassa on mukana Timber by Pinjan puutavaralajien koodit ja varastonumerot, joille on määritelty linkitykset Forest by Pinjaan (ks. 3.1 konfigurointi). Lisäksi sanoma sisältää myös varastomuutoksen tietokanta ID:n Timber by Pinjassa.

Varastomäärien muutokset synkronoidaan Forest by Pinjan varastoon määriteltyjen linkitysten avulla. Käytännössä Forest by Pinjan tietokantatauluun, josta varastomäärät haetaan, lisätään rivi, joka sisältää tiedon varastomäärän muutoksesta. Lisäksi tietokannan lokitustauluun lisätään rivi, joka sisältää Timber by Pinjasta tulleen varastomuutostietueen tietokanta ID:n ja Forest by Pinjaan

lisätyn varastomuutoksen tietokantatietueen ID:n. Lokituksen avulla varastomuutokset ohjelmien välillä voidaan todentaa jälkikäteen.

Funktiossa SendSideProductChanges() lähetetään Forest by Pinjan välivaraston varastosaldon vähennykset TImber by Pinjaan. Välivarastoista ajetaan kuormia eri määräpaikkoihin ja nämä kuormat vähentävät välivarastojen määrää. Ajetusta kuormasta tallentuu tietokantaan suoriteilmoitus.

Windowsin palvelu hakee Forest by Pinjan tietokannasta varastot, joille on kytketty synkronointi päälle ja tämän jälkeen varastoihin kohdistuneet suoriteilmoitukset. Suoriteilmoitusten varastoille ja puutavaralajeille haetaan konfiguroinnissa määritellyt Timber by Pinjan varastonumerot ja puutavaralajit. Tiedoista muodostetaan sanoma, joka lähetetään WCF-palveluiden kautta Timber by Pinjaan. Timber by Pinja välivarastoon lisätään sanomassa määritellylle sivutuotevarastolle ja puutavaralajille tietue, joka vähentää sivutuotevaraston määrää.

Timber by Pinja lähettää vastauksena sanoman, joka sisältää tiedon lähetyksen onnistumisesta ja tietokantaan tallennetun uuden tietueen ID:n. Forest by Pinjan tietokannan lokitustauluun lisätään rivi, johon tallennetaan lähetetyn suoriteilmoituksen tietokanta tietueen ID ja vastauksena saatu Timber by Pinjan tietueen tietokanta-ID.

Asiakkaan palvelimelle on asennettu Windows-palvelusovellus, joka toimii Windowsin taustalla ja suorittaa erilaisia taustatehtäviä, esimerkiksi tietojen lähetyksiä ja päivityksiä. Palvelu kutsuu Forest by Pinjan luokkia ja funktioita, jotka lisätään palveluun .dll-tiedostoina. Palvelu on yhteydessä Forest by Pinjan tietokantaan, joten sillä on pääsy samoihin tietoihin. Palvelu hakee suoritettavat tehtävät ja suoritusaikataulut tietokannasta ja tehtävien aikataulujen hallinta tapahtuu Forest by Pinjassa.

Uudelle tehtävälle ajettiin tietokantaan uusi tehtävätyyppi TipSideProductSynchronizing, jota Widows-palvelu käyttää. Palvelun koodiin lisättiin kutsu uudelle tehtävätyypille. Tehtävä kutsuu kahta, aiemmin Forest by Pinjaan lisättyä funktiota, GetSideProductStorageChanges() ja SendSideProductStorageChanges().

5 YHTEENVETO

Opinnäytetyön tavoitteena oli luoda asiakkaalle automaattinen varastomäärien synkronointi Pinjan toiminnanohjausjärjestelmien välille. Tavoitteeseen päästiin ja toiminto saatiin asiakkaalle käyttöön.

Projekti oli opettavainen oman ohjelmointiosaamisen kannalta ja opetti myös hallitsemaan isoja ohjelmistokokonaisuuksia ja niiden välisiä yhteyksiä. Projektiin kuului oleellisena osana metsäalan ymmärrys ja sen soveltaminen ohjelmointiin. Opin projektin aikana lisää metsäalasta ja sen toiminnasta.

Projekti kesti odotettua pidempään ja muutamia asioita jouduttiin miettimään uudelleen kehityksen aikana. Esimerkiksi puutavaralajien yksikkömuunnoksia ei huomioitu alkuperäisessä määrittelyssä ja niitä ei osattu ottaa aluksi huomioon, osittain omasta metsäalan kokemattomuudesta johtuen.

Rajapinnan testaaminen ja kehittäminen oli melko työlästä, koska se vaati käytännössä neljän ohjelmiston yhtäaikaista käynnissä oloa. Myös ohjelmistojen konfigurointi toimimaan vei aikaa, koska jokainen ohjelmisto tarvitsi oman testiympäristön ja ohjelmistojen välisiin yhteyksiin oli määriteltävä osoitteet ja tunnukset testausta varten.

Virheiden hallinta oli olennainen osa projektia. Koska tietoa liikuteltiin monen ohjelman välillä, oli tärkeää pystyä jäljittämään missä mahdollinen virhe tapahtui ja mikä sen aiheutti. Virheiden mahdollisimman aikainen havaitseminen vähentää kehitystyöhön kuluvaa aikaa ja parantaa myös ohjelmiston laatua.

Rajapinnat ja ohjelmistot toimivat kaikki .NET-kehitysympäristössä, joten käytetyt työkalut ja tekniikat sopivat hyvin yhteen ja sopivat toiminnon kehittämiseen. Opin paljon uutta käytetyistä tekniikoista, kuten Windows Communication Foundationista (WCF) ja .NET-kehitysympäristön työkaluista yleisesti. Kehitystyön aikana tuli vastaan monia pieniä ongelmia ja pulmatilanteita, joita ratkomalla varmuus omaan osaamiseen .NET-työkalujen käytöstä kasvoi. Vaikka WCF ei ole tämän opinnäytetyön kirjoitushetkellä uutta tekniikkaa, on sen osaamisesta varmasti hyötyä tulevaisuudessakin esimerkiksi olemassa olevien WCF-palveluiden ylläpidossa. Opintojen aikana .NET ei noussut esille millään tavalla, joten opintoihin nähden kaikki ympäristöstä opittu oli uutta.

Kehityskohteena jatkoa varten, dokumentoinnin projektista olisi voinut tehdä paremmin. Dokumentointia olisi kannattanut tehdä jo projektin aikana, mikä olisi helpottanut myös kehitystä, koska valittuja ohjelmointiratkaisuja olisi voinut tarkistaa dokumentaatiosta. Hyvä dokumentaatio auttaa myös muita kehittäjiä, jotka mahdollisesti myöhemmin tekevät muutoksia koodiin.

Uskon että projektista oli minulle paljon hyötyä tulevaisuutta varten. Sain arvokasta kokemusta .NET-kehityksestä ja ohjelmoinnista. Pystyn varmasti jatkossa hyödyntämään oppeja tulevissa projekteissa.

LÄHTEET

1. IBM 2024. What is an API? Hakupäivä 16.3.2024. https://www.ibm.com/topics/api.

2. Fielding, Roy, Thomas 2000. Architectural Styles and the Design of Network-based Software Architectures. Kalifornian yliopisto. Väitöskirja. Hakupäivä 28.4.2024. https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

3. IBM 2024. The structure of a SOAP message. Hakupäivä 27.2.2024. https://www.ibm.com/docs/en/integration-bus/10.1?topic=soap-structure-message.

4. Microsoft 2024. What is SQL Server Management Studio (SSMS). Hakupäivä 17.3.2024. https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms

5. Microsoft 2024. Using Data Contracts. Hakupäivä 13.3.2024. https://learn.microsoft.com/en-us/dotnet/framework/wcf/feature-details/using-data-contracts

6. Microsoft IIS. A flexible & easy-to-manage web server. Hakupäivä 18.3.2024. https://www.iis.net/

7. Microsoft 2024. Introduction to Windows Service Applications. Hakupäivä 13.3.2024. https://learn.microsoft.com/en-us/dotnet/framework/windows-services/introduction-to-windows-service-applications

8. Mozilla 2024. HTTP response status codes. Hakupäivä 28.4.2024. https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

TYYPILLISIMMÄT HTTP-VASTAUSKOODIT LIITE 1

HTTP-metodi	Statuskoodi	Selitys
GET	200 OK	Pyynnön käsittely onnistui ja vastaus sisältää pyydetyn resurssin.
GET	403 Forbidden	Asiakkaalla ei ole oikeutta hakea resurssia.
GET	404 Not Found	Pyydettyä resurssia ei löytynyt.
POST	201 Created	Uusi resurssi on onnistuneesti luotu palvelimelle.
POST	400 Bad Request	Asiakkaan lähettämässä pyynnössä on virhe eikä palvelin pysty käsittelemään sitä.
POST	401 Unauthorized	Pyynnössä ei ole mukana tarvittavia tunnistautumistietoja.
POST	403 Forbidden	Asiakkaalla ei ole oikeutta luoda resurssia.
PUT	200 OK	Resurssi on päivitetty onnistuneesti.
PUT	400 Bad Request	Pyyntö on virheellinen.
PUT	401 Unauthorized	Pyynnössä ei ole mukana tarvittavia tunnistautumistietoja.
PUT	403 Forbidden	Asiakkaalla ei ole oikeutta päivittää resurssia.
PUT	404 Not Found	Päivitettävää resurssia ei löydy.
DELETE	200 OK	Resurssi on poistettu onnistuneesti.
DELETE	401 Unauthorized	Pyynnössä ei ole mukana tarvittavia tunnistautumistietoja.
DELETE	403 Forbidden	Asiakkaalla ei ole oikeutta poistaa resurssia.
DELETE	404 Not Found	Poistettavaa resurssia ei löydy.

LIITE 1. Tyypillisimmät http-vastauskoodit (8)