OULUN AMMATTIKORKEAKOULU

Jere Kylätie

RAJAPINTOJEN YHDISTÄMINEN YHTENÄISEKSI JÄRJESTELMÄKSI

RAJAPINTOJEN YHDISTÄMINEN YHTENÄISEKSI JÄRJESTELMÄKSI

Jere Kylätie
Opinnäytetyö
Syksy 2020
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, Ohjelmistokehitys

Tekijä: Jere Kylätie
Opinnäytetyön nimi: Rajapintojen yhdistäminen yhtenäiseksi järjestelmäksi.
Työn ohjaaja: Teemu Korpela
Työn valmistumislukukausi ja -vuosi: Syksy 2020 Sivumäärä: 24

Ohjelmistointegraatiolla tarkoitetaan vähintään kahden järjestelmän liittämistä toisiinsa. Tässä opinnäytetyössä kehitettiin Kupari Datan ja Oulun Digin välille järjestelmä, jolla yhdistetään rajapintoja tietokoneiden varastoinnissa ja tiketöinnissä käytettävään ohjelmistoon. Ohjelmisto suunniteltiin helpottamaan yritysten välillä tapahtuvaa työskentelyä, joka koettiin vaikeaksi vanhalla ohjelmalla. Molemmissa organisaatioissa työskentelevillä on käyttöoikeudet ohjelmistoon, jolloin ajantasainen tilanne on näkyvissä molemmissa organisaatioissa yhtä aikaa. Rajapintojen yhdistämisellä saatiin parempi sovellus yritysten käyttöön, joka helpotti varastoinnin ja tiketöinnin seuraamista.

Ohjelmiston suunnittelun jälkeen tehtiin sovelluksen taustajärjestelmä, joka on kirjoitettu Pythonilla ja käyttöliittymä Angulari:lla. Sovellukseen rakennettiin oma taustajärjestelmä, johon on yhdistetty ulkoisten sovellusten rajapinnat.

Ohjelmiston kehittäminen on vielä kesken ja tulee jatkumaan tulevaisuudessa.

Asiasanat: API, relaatiotiotokannat, Python, REST, sovellus

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information technology, Option of software development

---

Author: Jere Kylätie
Title of thesis: Combining interfaces in to a single software
Supervisor: Teemu Korpela
Term and year when the thesis was submitted: Fall 2020
Number of pages: 24

Software integration means, combining at least two system in to one. The purpose of this thesis is to introduce software that has been built for Kupari Data and Oulun Digi. Which is being used to combine interfaces and databases used in inventory and ticketing software. Software was designed to make workflow between two companies easier, which was noticed to be hard with older system. Employees in both companies have rights to use software, so that up to date information is visible to both organisations in real time. Combining interfaces, we can get better software for organization, which makes inventory management and ticketing observation much easier.

After planning of the software backend was built, which is written in Python and user interface is made in angular. Software has standalone backend which is used to combine outside interfaces.

Software development is still ongoing and will continue in the future.

Keywords: API, Backend, Frontend, REST, relational databases, software

SISÄLLYS

1 KÄSITTEITÄ

API	Application programming interface eli ohjelmointirajapinta, jonka kautta eri ohjelmat voivat keskustella keskenään
atws	AutoTask Web Services on Python- kirjasto
ID	Indentifier on yksilöllinen tunniste
JSON	JavaScript Object Notation on avoimen standardin tiedostomuoto
Käyttöliittymä	Käyttäjälle näkyvä osa sovelluksesta, esimerkiksi asennettava sovellus tai nettisivun selaimessa toimiva osuus tai jonkin kirjaston käytettäväksi tarkoitettu rajapinta.
MySQL	Relaatiotietokantaohjelmisto
Rajapinta	Rajapinta mahdollistaa integraation ohjelmistoon, sen avulla voidaan tehdä pyyntöjä ohjelmistolle, josta halutaan noutaa tai tuoda tietoja
REST	Representational State Transfer, malliohjelmisto rajapintojen toteuttamiseen
Taustajärjestelmä	Taustalla toimiva järjestelmä osuus, esimerkiksi palvelimella toimiva sovellus tai jonkin kirjaston sisäinen toteutus tietylle ympäristölle

2 JOHDANTO

Opinnäytetyö on sovellusprojekti, joka on tehty Kupari Data Oy:lle. Projekti on aloitettu syksyllä 2019 ja jatkettu kevääseen 2020. Projektin tarkoituksena on ollut tehdä yhtenäinen sovellus, jossa yhdistetään useamman eri järjestelmän työkalut yhteen paikkaan. Tämän tarkoituksena on vähentää kirjautumisia sekä nopeuttaa ja helpottaa työntekijöiden työskentelyä.

Sovelluksen tarkoitus on yhdistää AutoTask-, F-Secure- ja NinjaRMM-sovellukset. Sovellukseen on luotu oma REST-rajapinta. Ohjelmalla saadaan yrityksen varastot helposti näkyviin ja töiden tiketöinnistä saadaan helpompaa ja nopeampaa. Varastonäkymiä voidaan näyttää räätälöidysti ulkoisille yrityksille tarpeiden mukaan. Ohjelmalla on oma tietokanta, jonne saadaan helposti tallennettua tiedot eri laitteista ja asiakkaista, mikä helpottaa laitteiden ylläpitoa, sekä saadaan tulostettua raportteja eri laitteiden tilasta.

Varastointinäkymä on testikäytössä, mutta tiketöinti on vielä kesken. Kehitystyö jatkuu ja sovellusta tullaan vielä muokkaamaan, kun testikäytön ja siitä saatavien palautteiden yhteenveto saadaan tehtyä.

3 KUPARI DATA

Kupari Data on perustettu 1980-luvun alussa, jolloin tapahtui voimakasta tietotekniikan käytön lisääntymistä. Yrityksen liiketoiminta perustuu laitehuoltoon ja yritys seuraakin tarkasti ICT-alan kehittymistä, jotta se pystyy vastaamaan haasteisiin, joita palveluiden käyttäjät tarvitsevat. Kupari Data tuottaa kaikki palvelut Suomessa, suomalaisin voimin. Henkilökunta on osaavaa, alan asiantuntijoita, joilla on alalta pitkä kokemus.

Kupari Data toimii kolmessa toimipisteessä Espoossa, Oulussa ja Joensuussa. Palvelut joita Kupari Data tarjoaa ovat

- tuotteiden myynti- ja elinkaaripalvelut; asiakkaille pyritään löytämään tuotteita, joiden saanti ja malli voidaan vakioida, jolloin asiakkaille saadaan huomattavia kustannussäästöjä
- laitteiden huoltopalvelut; takuuhuollot sekä laitteiden asennus ja tukipalvelut eri tilanteisiin
- tulostuksen hallintapalvelut; erilaiset tulostuksen hallintaratkaisut asiakkaille
- tukipalvelut; laitteisiin ja ohjelmistoihin kohdistuvat tukitarpeet
- konesali- ja pilvipalvelut; konesalipalveluissa hallitaan asiakkaan palvelimia asiakkaan omassa konesalissa tai virtualisoimalla ne Kupari Datan konesaleihin. Konesalipalvelussa toimitetaan palvelinten lisäksi mm. levykapasiteettia ja varmistuspalveluja. Pilvipalvelut tuotetaan kotimaassa sijaitsevista konesaleista, jolloin etuna on nopeat ratkaisut ongelmatilanteissa sekä tietoturva-asiat
- tietoturvapalvelut; tietoturvapalveluita yhteistyössä F-Securen kanssa
- verkkokauppa; yritysasiakkaille rakennettu verkkokauppa hankintoja varten

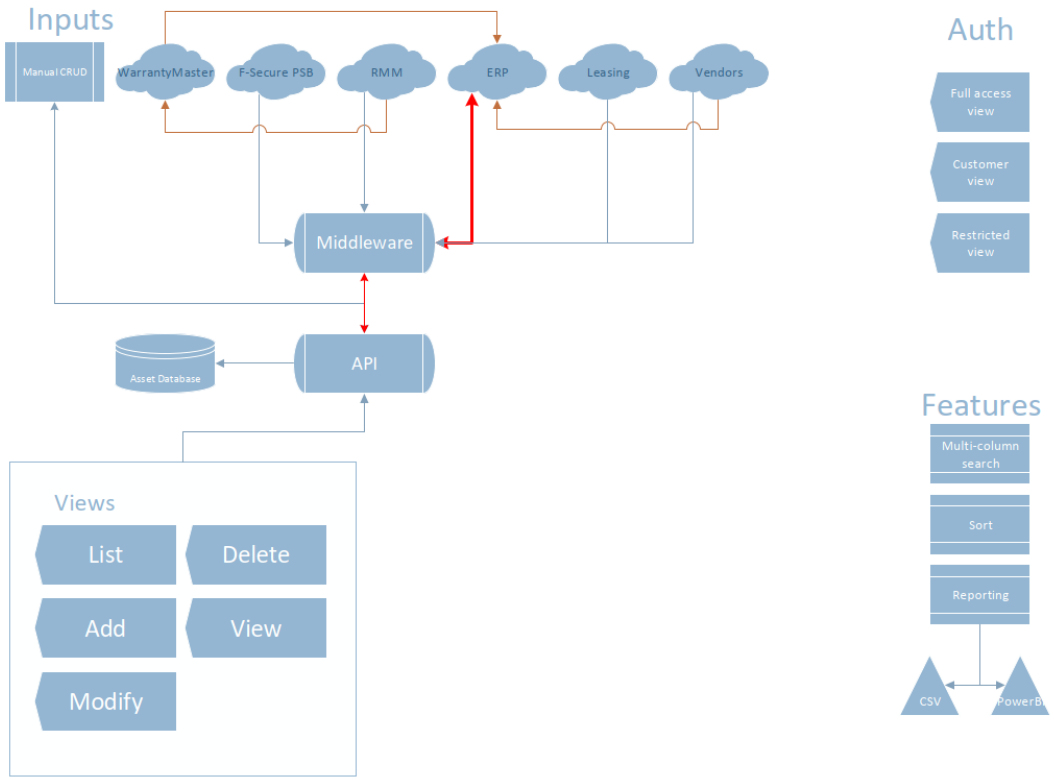
Kupari Datan periaate on kartoittaa asiakkaan tarpeet ja seurata palveluiden toimivuutta sekä läpikäydä säännöllisesti mahdollisia kehitystarpeita. (1.)

4 PROJEKTIN SUUNNITTELU

Projektia aloitettiin suunnittelemaan, kun Kupari Data otti käyttöön AutoTask-järjestelmän. AutoTaskin oma käyttöliittymä on sekava ja hankala käyttöinen. Tarkoituksena on saada yksinkertaisempi käyttöliittymä, jossa voidaan tehdä samat asiat kuin AutoTaskissa, yksinkertaisemmin ja vähemmillä klikkauksilla.

Suunnittelussa tuli myös ilmi mahdollisuus lisätä muita järjestelmiä samaan sovellukseen. Sovelluksen on tarkoitus toimia yhdellä kirjautumisella. Kirjautuminen tapahtuu O365-tunnuksella, jonka kautta voidaan myös rajoittaa eri käyttäjien oikeuksia ja niitä ominaisuuksia, joita ohjelmasta näkyy käyttäjälle.

Alla olevassa kaaviossa (kuva 1) on suunnitelma kokonaisesta sovelluksesta. Sen ominaisuuksista ja miten eri rajapintojen pitäisi keskustella keskenään ja tietokannan kanssa.



KUVA 1. Hallintasuunnitelma ohjelmasta.

5 TAUSTAJÄRJESTELMÄ

Sovelluksen taustajärjestelmä on tehty alusta alkaen. Tarkoituksena saada oma toimiva REST-rajapinta, jossa on toimivat kyselyt eri ulkopuolisiin sovelluksiin. Suurin osuus taustajärjestelmästä koostuu AutoTask API:n kyselyistä ja sen kirjastoista.

5.1 Python Flask

Flask on Python-kirjasto, jolla on helppo luoda web-sovelluksen REST-rajapinta. Flask mahdollistaa helpon tavan saada yhteys käyttöliittymään ja tehdä POST- ja GET-kyselyitä.

Sovelluksessa on käytetty eri Python-kirjastoja rajapintojen yhdistämiseen.

- Python AutoTask Web Services (atws)
- NinjaRMM API (NinjaRMM API)

Näistä suurin kirjasto on atws, jossa myös oli ongelmia. Kirjastoa joutui muokkaamaan ja korjaamaan virheitä kirjaston koodissa. (2.)

5.2 AutoTask

AutoTask on järjestelmä, joka mahdollistaa yrityksen töiden organisoinnin, valvonnan ja tiketöinnin. AutoTaskissa on paljon ominaisuuksia, esimerkiksi varastointi, asiakastiedot, asiakkaiden laskutus, tuotteiden lisäys varastoon ja aktiivisten töiden seuranta. AutoTaskin omassa käyttöliittymässä on ongelmana sen monimutkaisuus. Yksinkertaisen asian tekemiseen pitää mennä mutkan kautta, joka vaatii useampia klikkauksia. AutoTaskin tuominen kokonaan ulkoiseen rajapintaan nopeuttaa työntekoa ja tehokkuutta. (3.)

4.2.1 API

AutoTaskilla on oma API eli ohjelmistorajapinta, jonka avulla sen voi yhdistää omaan rajapintaan. Tähän on olemassa Pythonille tehty kirjasto (atws). Tämä kirjasto mahdollisti yhteyden ottamisen

ja kirjautumisen AutoTaskiin. Tällä hetkellä on sovellukseen luotu varastonäkymä, työtiketin luonti ja tikettien katselu, tuotteiden lisäys tikettiin ja tiketin tulostus. (4.)

Tulostettava tiketti on räätälöity työntekijöiden haluamaksi. Vain tarpeelliset työhön liittyvät tiedot näkyvät tulosteessa. AutoTaskin omassa tulosteessa oli ongelmana turhat tiedot.

4.2.2 Kyselyt

Kuvassa 2 on muutama esimerkki koodista, jossa on tehty kyselyfunktiot. Nämä kyselyt hakevat datan AutoTaskin omasta rajapinnasta. Esimerkiksi AllTicketQuery hakee AutoTaskista kaikki tiketit, joissa id on suurempi kuin 7683. Kun taas funktio NewTicketQuery hakee uudet tiketit, joiden id on suurempi kuin 7683, ja tarkistaa tiketin status-tilan, jos se on New.

```
def AllTicketQuery():
    AllTicketquery = atws.Query('Ticket')
    AllTicketquery.WHERE('id',AllTicketquery.GreaterThan,7683)
    return AllTicketquery

def getTicketSearch(haku):
    query = atws.Query('Ticket')
    query.WHERE('TicketNumber',query.Contains,haku)
    query.OR('Title',query.Contains,haku)
    return query

def newticketQuery():
    query = atws.Query('Ticket')
    query.WHERE('id',query.GreaterThan,7683)
    #query.open_bracket('AND')
    query.AND('Status',query.Equals,at.picklist['Ticket']['Status']['New'])
    return query

def completeTicketQuery():
    query = atws.Query('Ticket')
    query.WHERE('id',query.GreaterThan,7683)
    #query.open_bracket('AND')
    query.AND('Status',query.Equals,at.picklist['Ticket']['Status']['Complete'])
    return query

def inProgressticketQuery():
    query = atws.Query('Ticket')
    query.WHERE('id',query.GreaterThan,7683)
    #query.open_bracket('AND')
    query.AND('Status',query.Equals,at.picklist['Ticket']['Status']['In Progress'])
    return query
```

KUVA 2. Kyselyfunktio.

4.2.3 Funktion kutsuminen

Kyvassa 3 on route-GET-metodi, jolla saadaan rajapinnasta haluttu data route-osoitteesta. Funktiossa kutsutaan NewTicketQuery, joka suorittaa haun AutoTaskin rajapinnasta. Saatu data

käsitellään ja muutetaan luettavaan muotoon. Funktio palauttaa datan JSON-muotoon, jota on helppo käsitellä käyttöliittymän puolella.

```
382 @bp.route('/tickets/new', methods=['GET'])
383 def getTicketsNew():
384
385     responseList=[]
386     tickets = at.query(newticketQuery())
387     #test = at.picklist['Ticket'].lookup('Status')
388     for ticket in tickets:
389         response={}
390             # ContactID could be null as the contact might not be added yet
391         response['TicketNumber'] = ticket.TicketNumber
392         response['id'] = ticket.id
393         response['Title'] = ticket.Title
394         response['AccountID'] = ticket.AccountID
395         response['Priority'] = ticket.Priority
396         response['TicketType'] = at.picklist['Ticket']['TicketType'].reverse_lookup(ticket.TicketType)
397         try:
398             response['ContactID'] = ticket.ContactID
399         except:
400             response['ContactID'] = ''
401         try:
402             response['Status'] =  at.picklist['Ticket']['Status'].reverse_lookup(ticket.Status)
403         except:
404             response['Status'] = ''
405         try:
406             response['DueDateTime'] = ticket.DueDateTime
407         except:
408             response['DueDateTime'] = ''
409         try:
410             response['IssueType'] = at.picklist['Ticket']['IssueType'].reverse_lookup(ticket.IssueType)
411         except:
412             response['IssueType'] = ''
413         try:
414             response['SubIssueType']= at.picklist['Ticket']['SubIssueType'].reverse_lookup(ticket.SubIssueType)
415         except:
416             response['SubIssueType'] = ''
417         finally:
418             responseList.append(response)
419     ##print(test)
420     return jsonify(responseList)
421
```

KUVA 3. Tikettien haku.

4.2.4 Picklist

AutoTaskissa monet kentät ovat kannassa vakioarvoltaan id:nä. Niiden kenttien tietojen hakemiseen joutuu käyttämään atws-kirjaston picklist-funktiota. Alla olevassa kuvassa 4. Haetaan IssueType-kentän arvo muuttujaan picklistillä. Reverse lookup-funktio hakee kentän teksti arvon eikä tulosta pelkkää vakio arvoa.

```
response['IssueType'] = at.picklist['Ticket']['IssueType'].reverse_lookup(ticket.IssueType)
```

KUVA 4. Picklistin käyttö.

4.2.5 Tiketin hakeminen

Alla olevassa kuvassa 5. on tiketinhakufunktio GetTicketSearch. Funktiolla haetaan tikettiä AutoTaskin järjestelmästä tiketti numerolla tai tiketin nimellä. Funktio saa arvon käyttöliittymästä ja käy ne tiketit läpi, jotka sisältävät täsmäävän arvon.

```
32  def getTicketSearch(haku):
33      query = atws.Query('Ticket')
34      query.WHERE('TicketNumber',query.Contains,haku)
35      query.OR('Title',query.Contains,haku)
36      return query
37
```

KUVA 5. Tiketinhakufunktio.

Käyttöliittymässä on vaihtoehtona rajata tikettien näkyvyyttä tiketin tilan mukaan tai hakea tikettiä Tiketin numerolla tai otsikolla. Alla olevassa kuvassa (kuva 6) on koodinäyte, jossa backend saa arvon hausta ja SeachQuery tarkistaa täsmääkö arvo mihinkään rajaukseen. Jos SearchQuery on tiketti numero tai -otsikko se käyttää yllä olevan kuvan funktiota ja palauttaa yksittäisen tiketin kaikki tiedot.

```
184         SearchQuery = request.args.get('query', default='')
185         multi_dict2 = request.args
186         testiDefault = ""
187         #for key in multi_dict2:
188             #print(multi_dict2.get(key))
189             #print(multi_dict2.getlist(key))
190
191         #print("Searching for" + SearchQuery)
192         if SearchQuery:
193             if SearchQuery == 'In Progress':
194                 #print(' if inprogress')
195                 tickets = at.query(inProgressticketQuery())
196
197             elif SearchQuery == 'AllTickets':
198                 #print(' if all')
199                 tickets = at.query(AllTicketQuery())
200
201             elif SearchQuery == 'Complete':
202                 #print(' if complete')
203                 tickets = at.query(completeTicketQuery())
204
205             elif SearchQuery == 'New':
206                 #print(' if new')
207                 tickets = at.query(newticketQuery())
208
209             else:
210                 #print('elif')
211                 tickets = at.query(getTicketSearch(SearchQuery))
212         else:
213             #print('elsesessä')
214             tickets = at.query(AllTicketQuery())
215
```

KUVA 6. Hakufunktio.

Jos hakuun on valittu esimerkiksi InProgress tiketin tilaksi, niin ohjelma kutsuu InProgressticketQuery-funktiota (kuva 7). Funktio tarkistaa tiketit ja etsii picklistillä tiketin, jonka status -tila on In Progress. Ja palauttaa tiketin kaikki arvot. Tämä sama menetelmä on myös kaikilla tilan mukaan haettavilla tiketeillä.

```
52  def inProgressticketQuery():
53      query = atws.Query('Ticket')
54      query.WHERE('id',query.GreaterThan,7683)
55      #query.open_bracket('AND')
56      query.AND('Status',query.Equals,at.picklist['Ticket']['Status']['In Progress'])
57      return query
```

KUVA 7. InProgress-funktio

Tikettien tiedot eivät sisällä suoraan asiakkaiden tietoja. Asiakkaisiin ja asiakastileihin tiketin tiedoissa viitataan vain id:llä. Asiakkaat ja tilitiedot haetaan erillisessä kyselyssä id:n perusteella ja saadaan teksti muodossa tarvittavat kentät (kuva 8), kun luetaan tikettejä.

```
461 @bp.route('/contact/<id>', methods=['GET'])
462 def getSingleContact(id):
463     ContactQuery = request.args.get('query', default='')
464     multi_dict2 = request.args
465     responseContact=[]
466 
467     if ContactQuery:
468         query=atws.Query('Contact')
469         query.WHERE('ContactID',query.Equals,ContactQuery)
470         contacts=at.query(query).fetch_all()
471     else:
472         contacts = at.query(getContactInfo())
473     for contact in contacts:
474         findContactResp = {}
475         findContactResp = contact.id
476         if str(id) == str(findContactResp):
477             break
478     if str(id) in str(findContactResp):
479         Conresponse={}
480         try:
481             Conresponse['Name']= contact.FirstName + " " + contact.LastName
482             Conresponse['id'] = contact.id
483             Conresponse['FirstName'] = contact.FirstName
484             Conresponse['LastName'] = contact.LastName
485             Conresponse['Phone'] = contact.Phone
486             Conresponse['EMailAddress'] = contact.EMailAddress
487             Conresponse['AccountID'] = contact.AccountID
488             Conresponse['AddressLine'] = contact.AddressLine
489             Conresponse['AddressLine1'] = contact.AddressLine1
490             Conresponse['City'] = contact.City
491             Conresponse['AdditionalAddressInformation'] = contact.AdditionalAddressInformation
492             Conresponse['ZipCode'] = contact.ZipCode
493         except:
494             Conresponse['Name']= ''
495             Conresponse['id'] = ''
496             Conresponse['FirstName'] = ''
497             Conresponse['LastName'] = ''
498             Conresponse['Phone'] = ''
499             Conresponse['EMailAddress'] = ''
500             Conresponse['AccountID'] = ''
501             Conresponse['AddressLine'] = ''
502             Conresponse['AddressLine1'] = ''
503             Conresponse['City'] = contact.City
504             Conresponse['AdditionalAddressInformation'] = ''
505             Conresponse['ZipCode'] = ''
506         return jsonify(Conresponse)
```

KUVA 8. Kontaktien haku ja käsittely

4.2.6 Inventaario

AutoTaskissa inventaarionäkymät menevät hankalasti. Ensin pitää hakea inventaarion sijainti id:llä, jonka kautta saadaan valittua minkä varaston tietoja. Halutaan saada näkyviin. Alla olevassa esimerkissä kuvan 9 esimerkissä on käytetty filtteröintiä, jolla helpotetaan varastojen näkymää kumppani yritykselle.

```
@bp.route('/inventorylocations', methods=['GET'])
def ListInventoryLocations():
    ##print(request.headers)

    filters = request.args.get('query', default='')
    InventoryLocationList = []
    InvLocResponse={}
    if filters:
        InvLocResponse['id'] = 2
        InvLocResponse['LocationName'] = "Oulun Digi"
        InventoryLocationList.append(InvLocResponse)
    else:
        invLocations = atws.Query('InventoryLocation')
        invLocations.WHERE('id',invLocations.GreaterThan,0)
        locationList = at.query(invLocations)
        for invLocation in locationList:
            InvLocResponse={}
            try:
                InvLocResponse['id'] = invLocation.id
                InvLocResponse['LocationName'] = invLocation.LocationName
            except:
                InvLocResponse['id'] = ''
                InvLocResponse['LocationName'] = ''
            finally:
                InventoryLocationList.append(InvLocResponse)
    return jsonify(InventoryLocationList)
```

KUVA 9. Inventaariosijainnin haku.

Varaston sijainnin tiedon saamisen jälkeen voidaan hakea eri tuotteet, joita kyseisen varastosijainnin alla on. AutoTaskin puolella tuotteet ja varasto ovat eri paikoissa, joten on jouduttu yhdistämään tuotteiden haku ja InventoryItem-haku (kuva 10), ennen kuin tulostetaan koko varastolista. Varaston tulostus menee samalla tavalla kuten aikaisemmassa kuvassa (kuva 3), jossa näkyy datan käsittely, mikä tulostetaan JSON-muotoon.

```python
@bp.route('/inventoryitem', methods=['GET'])
def ListInventoryItems():
    invetoryQuery = request.args.get('query', default='')
    multi_dict2 = request.args

    responseAllinventory = []
    responseAllProduct = []

    if invetoryQuery:
        InventoryItems=atws.Query('InventoryItem')
        InventoryItems.WHERE('InventoryLocationID',InventoryItems.Equals,invetoryQuery)
        InventoryItems=at.query(InventoryItems).fetch_all()

        allProducts=atws.Query('Product')
        allProducts.WHERE('id',allProducts.GreaterThan,0)
        allProducts=at.query(allProducts).fetch_all()

    else:
        InventoryItems=atws.Query('InventoryItem')
        InventoryItems.WHERE('InventoryLocationID',InventoryItems.GreaterThan,0)
        InventoryItems=at.query(InventoryItems).fetch_all()

        allProducts=atws.Query('Product')
        allProducts.WHERE('id',allProducts.GreaterThan,0)
        allProducts=at.query(allProducts).fetch_all()

    for InvItem in InventoryItems:
        InvItemResponse={}
        for productItem in allProducts:
            productResponse={}
            try:
                productResponse['id'] = productItem.id
                productResponse['Name'] = productItem.Name
                productResponse['ProductCategory'] = at.picklist['Product']['ProductCategory'].reverse_lookup(productItem.ProductCategory)
            except:
                productResponse['id'] = ''
                productResponse['Name'] = ''
                productResponse['ProductCategory'] = ''
            responseAllProduct.append(productResponse)
```

KUVA 10. Inventaariotuotteiden haku.

4.2.7 Uusien tikettien luonti

Uuden tiketin luominen tapahtuu flaskin POST-metodilla, Kun Käyttöliittymä lähettää kenttiä vastaavan JSON-datan. Taustajärjestelmän alla oleva koodi käsittelee datan ja tallentaa sen muuttujiin (kuva 11), jonka jälkeen data lähetetään funktiolle CreateNewTicket (kuva 12).

```python
@bp.route('/tickets', methods=['GET','POST'])
def getAllTickets():
    if request.method == 'POST':

        TicketTitle = request.json['Title']
        TicketAccID = request.json['AccountID']
        TicketDescription = request.json['Description']
        TicketContactID = request.json['ContactID']
        TicketPrio = request.json['Priority']
        TicketIssueType = request.json['IssueType']
        TicketSubIssueType = request.json['SubIssueType']
        TicketType = request.json['TicketType']
        TicketCategory = request.json['TicketCategory']
        TicketQueueID = request.json['QueueID']


        tUDFServiceType = request.json['ServiceType']
        tUDFValtNum = request.json['ValittajanTyoNum']
        tUDFpaaMtNum = request.json['PaamiehentyoNum']
        tUDFPRodNUm = request.json['ProductNumber']
        tUDFProName = request.json['ProductName']
        tUDFSerialNum = request.json['SerialNumber']

        ##print(TicketSubIssueType)

        #print(tUDFProName,tUDFSerialNum)

        newTicketResp =  CreateNewTicket(TicketTitle,TicketAccID,TicketDescription,TicketContactID,TicketPrio, \
            TicketIssueType,TicketType,TicketSubIssueType,tUDFPRodNUm,TicketCategory,tUDFSerialNum,tUDFProName, \
            TicketQueueID,tUDFServiceType,tUDFValtNum,tUDFpaaMtNum)

        return  jsonify(newTicketResp)
```

KUVA 11. Uuden tiketin POST.

CreateNewTicket ottaa saamansa arvot ja lisää datan AutoTaskin haluamaan muotoon ja oikeisiin kenttiin, jonka jälkeen data lähetetään AutoTaskin omaan rajapintaan.

```
69  def CreateNewTicket(tTitle,tAccID,tDesc,tConID,tPrio,tIssueType,tType, \
70      tsubIssue,UDFProdNUM,tCatgor,tSerNum,tProdName,tQueueID,tservType,tvalnum,tpaatnum):
71
72      NewTicketID = 0
73
74      NewTicket = at.new('Ticket')
75      NewTicket.Title = tTitle
76      NewTicket.AccountID = tAccID
77      NewTicket.Description = tDesc
78      NewTicket.ContactID = tConID
79      NewTicket.DueDateTime = datetime.now()
80      NewTicket.TicketType = at.picklist['Ticket']['TicketType'][tType]
81      NewTicket.IssueType = at.picklist['Ticket']['IssueType'][tIssueType]
82      NewTicket.Priority = at.picklist['Ticket']['Priority'][tPrio]
83      NewTicket.TicketCategory = at.picklist['Ticket']['TicketCategory'][tCatgor]
84      NewTicket.SubIssueType = at.picklist['Ticket']['SubIssueType'][tIssueType][tsubIssue]
85      NewTicket.QueueID = at.picklist['Ticket']['QueueID'][tQueueID]
86
87      new_udf1 = NewTicket.set_udf('Product Number',UDFProdNUM)
88      new_udf2= NewTicket.set_udf('Product Name',tProdName)
89      new_udf3= NewTicket.set_udf('Serial Number (non-asset)',tSerNum)
90      new_udf4= NewTicket.set_udf('Service Type',tservType)
91      new_udf5= NewTicket.set_udf('Välittäjän Työnumero',tvalnum)
92      new_udf6= NewTicket.set_udf('Päämiehen työnumero',tpaatnum)
93
94      NewTicket.Status = at.picklist['Ticket']['Status']['New']
95      ##print(tProdName,tSerNum)
96      #NewTicket.create()
97
98      new_ticket = at.create(NewTicket).fetch_one()
99      ##print(new_ticket.id)
100     NewTicketID = new_ticket.id
101     return NewTicketID
```

KUVA 12. Uuden tiketin input-tietojen käsittely.

Tikettien päivittäminen tapahtuu samalla tavalla. Erona on vain, että päivittäessä käytetään Update-funktiota (kuva 13) CreateNewTicket-funktion sijaan.

```
144         updatedTicket.update()
```

KUVA13. Update-funktion kutsuminen.

5.3 NINJARMM

NinjaRMM on seurantasovellus asiakkaiden laitteille. Sovelluksella voidaan tehdä kyselyitä ja seurata laitteiden tilaa ja saada hälytyksiä mahdollisista ongelmista. NinjaRMM-rajapinnan yhdistäminen järjestelmään on vielä hieman keskeneräinen, eikä se ole ollut iso prioriteetti verrattuna AutoTaskin tärkeyteen. Tällä hetkellä sovelluksessa on mahdollista listata asiakkaat NinjaRMM:n kautta (kuva 14). (5.)

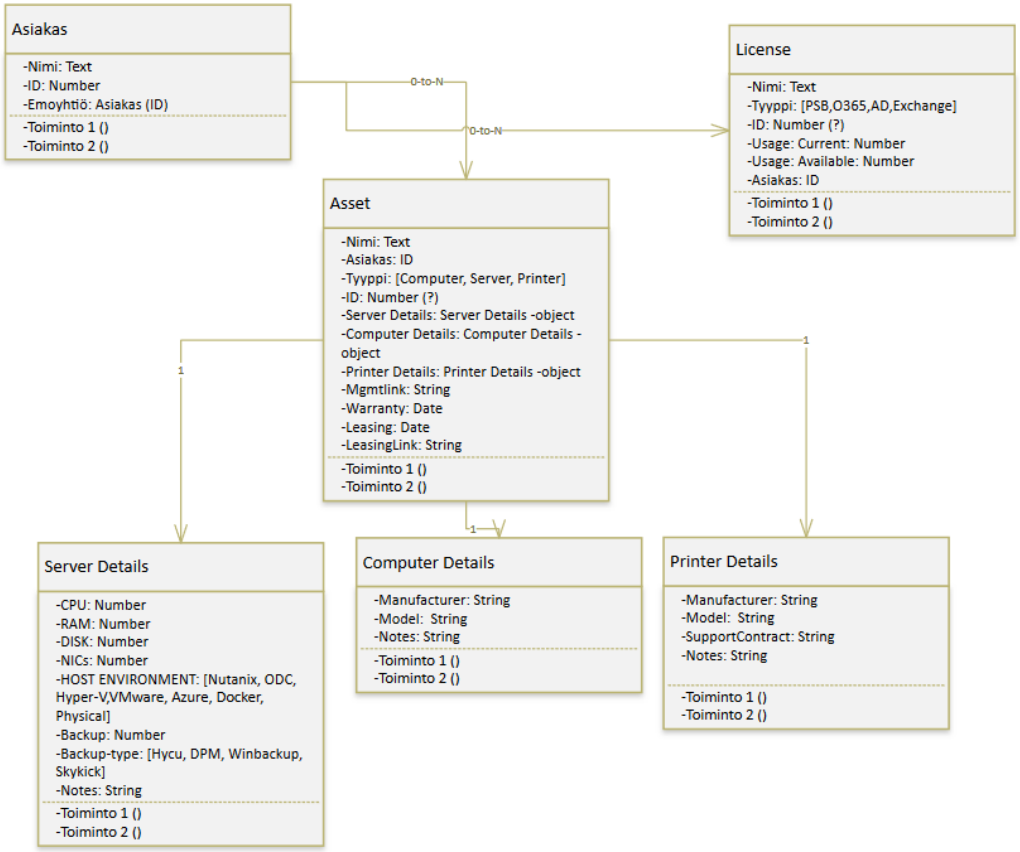
```python
from flask import current_app, Blueprint, request, jsonify
from marshmallow import Schema, fields
import ninjarmm_api

bp = Blueprint('ninjarmm', __name__, url_prefix='/ninjarmm')

# Do we need to initialize napi all the time? We could be getting a lot of these requests?
ninjauser=current_app.config['NINJARMM_USER']
ninjakey=current_app.config['NINJARMM_KEY']

@bp.route('/customers', methods=['GET'])
def getCustomers():
    # Passwords etc need to fetched from ENV safely
    napi = ninjarmm_api.client(ninjauser, ninjakey, iseu=True,debug=True)
    customers = napi.get_customers()
    return jsonify(customers)

@bp.route('/alerts', methods=['GET'])
```

KUVA 14. NinjaRMM yhteys.

6 TIETOKANTA

Ohjelman tietokanta on suunniteltu ja kirjoitettu MySQL-kielellä ja on rakennettu valmiiksi, mutta ohjelman pienen keskeneräisyyden vuoksi tietokantaa ei tällä hetkellä käytetä. Tietokannan tarkoituksena on saada tallennettua raportti, josta saa tietoa laitteista ja laitteiden omistavista asiakkaista.

Tietokantaan on tarkoitus saada tiedot eri järjestelmistä. Esimerkiksi asiakastiedot kerätään AutoTaskin kautta ja laitetiedot NinjaRMM:n kautta (kuva 15).



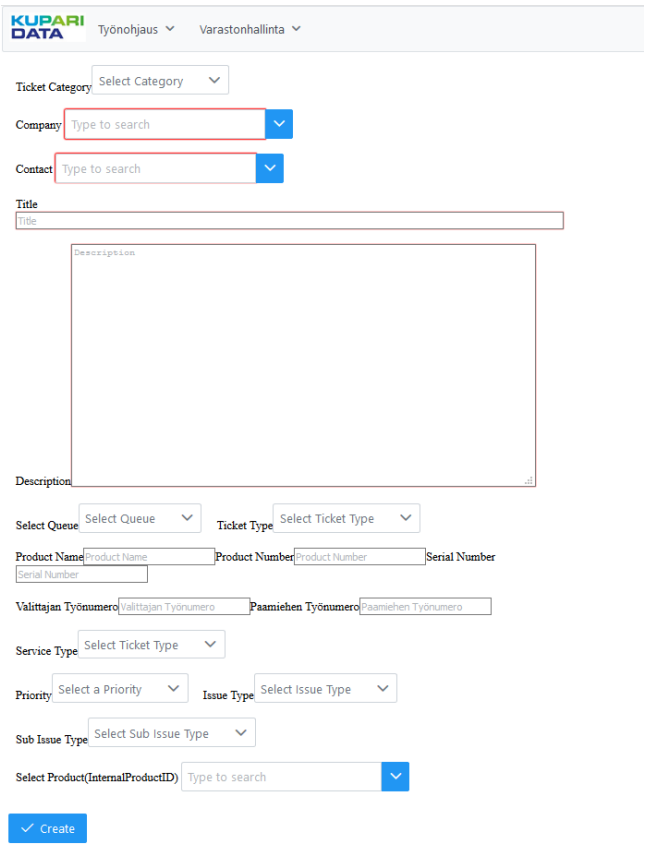
KUVA 15. ER-kaavio tietokannan tauluista.

7 KÄYTTÖLIITTYMÄ

Sovelluksen käyttöliittymä on tehty Angular 6 -ohjelmointikieltä käyttäen. Sovelluksen käyttöliittymä on vielä hieman keskeneräinen ja vaatii eri komponenttien sijoittelua ja järjestämistä selkeämmäksi. Käyttöliittymässä on paljon keskeneräistä kohtia, joita pitää muokata. Ja muutamia ohjelmavirheitä, joita ei ajan loppumisen vuoksi ole ehditty korjaamaan. Tällä hetkellä varsinaisessa käytössä järjestelmästä on AutoTaskin varastointi, joka on käytössä kahdella eri yrityksellä. Varastonäkymässä kumppani näkee reaaliajassa, mitä heidän tuotteitaan löytyy Kupari Datan hyllystä. (6.)

7.1 Tikettinäkymä

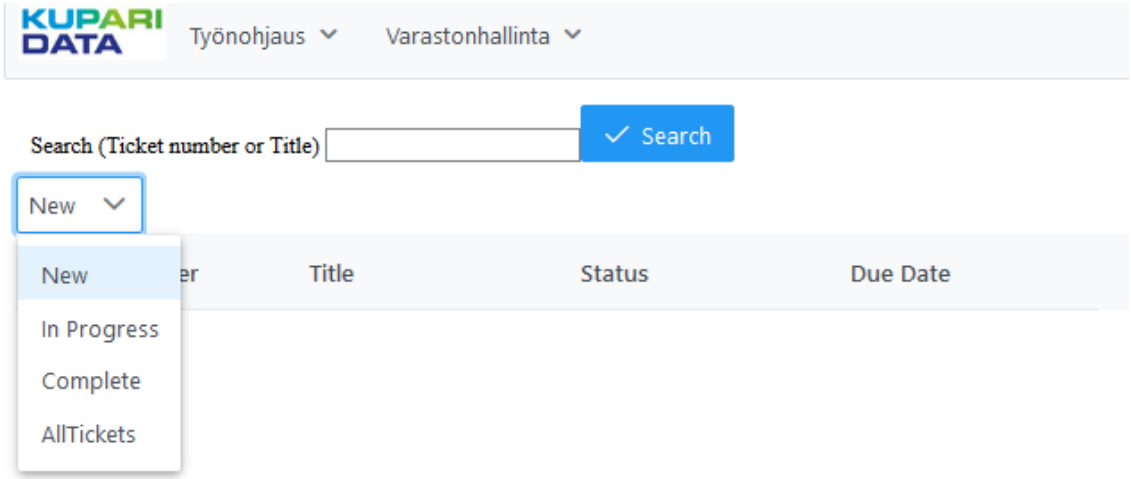
Käyttöliittymän puolelta voi tällä hetkellä luoda uuden huoltotiketin ja tarkastella vanhoja tikettejä. Alla olevasta kuvasta (kuva 16) näkyy käyttöliittymän keskeneräisyys, mutta tiketin luonti on kuitenkin toimiva ominaisuus. Tikettiin voi lisätä varastosta tuotteita, joita voidaan laskuttaa asiakkaalta.



KUVA 16. Uuden tiketin luonti.

7.2 Haku

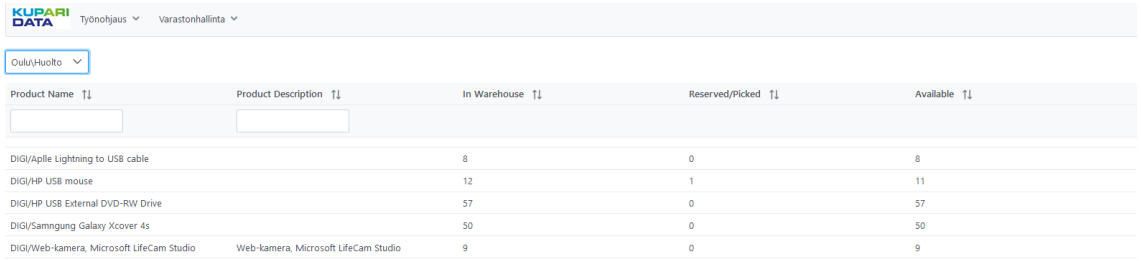
Alla olevassa kuvassa (kuva 17) näkyy hakukenttä ja tiketin tilan valinta, jolla saa tilan mukaan listattua tiketit. Yksittäisten tikettien hakeminen onnistuu Search-kentän kautta joko tiketin numerolla tai nimellä.



KUVA 17. Tikettien haku

7.3 Inventaarionäkymä

Inventaarionäkymässä näkee kaikki tuotteet ja palvelut, joita yritys tarjoaa. Lisäksi siinä näkee paljonko tuotteita on tällä hetkellä hyllyssä ja onko tuotteita varattuna. Tätä näkymää käyttää tällä hetkellä kaksi eri yritystä, Kupari Data ja Oulun Digi, joka on Kupari Datan kumppani. Oulun Digi käyttää varastonäkymää omien tuotteidensa seurantaan. Vasemmassa yläreunasta alasvetovalikosta valitaan, mitä varastoa halutaan katsoa. Tämän jälkeen kyseinen inventaariosijainti aukeaa alas listana (kuva 18).



KUVA 18. Inventory- näkymä.

8 YHTEENVETO

Projektissa on saatu toimiva sovellus Kupari Datan ja Oulun Digin käyttöön. Ohjelman varastointinäkymä on käytössä molemmilla organisaatioilla tällä hetkellä. Kehitystyö jatkuu tiketöinnin ja muiden rajapintojen osalta.

Oulun Digi on ollut tyytyväinen tähän asti käytössä oleviin ominaisuuksiin. Sovelluksessa on vielä paranneltavaa ja tekemistä käyttöliittymässä ja taustajärjestelmässä. Osasyynä ohjelman ominaisuuksien puutteeseen on henkilöstön resurssien puute ja vuoden 2020 aikana tulleet koronaepidemian aiheuttamat lomautukset. Testikäytöstä odotellaan parannusehdotuksia ja palautetta.

Sovelluskehitystä tullaan jatkamaan ja parantamaan tulevaisuudessa. Kun sovelluskehitys saadaan päätökseen, on ohjelman tarkoitus olla yksi paikka, jossa Kupari Data voi tehdä työn hallintaa ja organisointia keskitetysti.

LÄHTEET

1. Kupari Data Oy 2020. https://www.kuparidata.fi. Hakupäivä 13.11.2020.

2. Python Flask. 2010. Flask Documentation 1.1x. Saatavissa: https://flask.palletsprojects.com/en/1.1.x/. Hakupäivä 2.12.2020

3. Welcome to Python AutoTask Web Services 's documentation! 2016. Python AutoTask Web Services. Saatavissa: https://atws.readthedocs.io/index.html.Hakupäivä 13.11.2020.

4. AutoTask APIs. 2020. Datto Developer Help. Saatavissa: https://www.autotask.net/help/developerhelp/Content/AdminSetup/2ExtensionsIntegrations/APIs/APIs.htm. Hakupäivä 2.12.2020

5. NinjaRMM. Developers API. Saatavissa: https://www.NinjaRMM.com/dev-api/. Hakupäivä 13.11.2020.

6. Components. 2020. Angular Material. Saatavissa: https://material.angular.io/components/categories. Hakupäivä 14.11.2020.