# REST RAJAPINNAT MICROSOFT AZURESSA

LAB-AMMATTIKORKEAKOULU
Insinööri AMK
Ohjelmistotekniikka
Syksy 2020
Samu Salminen

**Tiivistelmä**

| Tekijä(t) | Julkaisun laji | Valmistumisaika |
|---|---|---|
| Salminen, Samu | Opinnäytetyö, AMK | Syksy 2020 |
| | Sivumäärä<br>42 | |

Työn nimi

**REST rajapinnat Microsoft Azuressa**

Tutkinto

Tieto- ja viestintätekniikka, insinööri (AMK)


Tiivistelmä

Tavoite oli tehdä kolme REST rajapintaa, joista pääsääntöisesti haetaan tietoja http GET pyynnöllä käyttäen kysely parametrejä ja Basic Authentication otsikko protokollaa rajapintaan palautuvan arkaluontoisen datan tietoturvana. Rajapinnat toteutettiin Azure Functions ja AppService työkaluilla.

Kysely parametrejä käytettiin sql tietokannassa olevan kontrollitaulun rivien kyselyihin ja lopullisen http paluuviestin datan sisältö oli sijoitettuna Azure Datalake kansioihin tekstitiedostoina, josta niiden sisältöä haettiin Datalaken omia luokkia ja kirjastoa käyttämällä. Rajapintoja pystyttiin kutsua vain tietyillä Basic Authentication käyttäjä ja salasana yhdistelmillä, jotka olivat sijoitettuna Azure KeyVault palveluun salaisuuksina.

Rajapintojen toiminnallisuuksiin kuului myös muokata Datalakesta vastaanotettu tekstitiedostonsisältö JSON muotoon, ja palauttaa data Http paluuviestissä. Lisäksi rajapinnat muokkasivat SQL kontrollitaulun rivejä tai lisäsivät niihin tietoa.



Asiasanat

REST API, Azure, Azure Functions, SQL, Datakale, C#

**Abstract**


| Author(s) | Type of publication | Published |
|---|---|---|
| Salminen, Samu | Bachelor's thesis | Autumn 2020 |
| | Number of pages | |
| | 42 | |


Title of publication

**REST API's in Microsoft Azure**

Name of Degree

Bachelor or Information and Communication Technology


Abstract

Goal of the project was to develop three different REST API programs which were mostly used with Http GET requests with query parameters.
Basic authentication header protocol was used to protect sensitive http response data. REST APIs were built and programmed by using tools Azure Functions and App Service.

Query parameters were used to query rows from SQL control table and http response body's data was situated in Azure Datalake's folder as text files. Data was fetched by using Datalake's classes and methods
REST API's could only be called by using authenticated secret credentials secured by Azure KeyVault.

REST API's also had a mandatory feature to transform a text string content to more sophisticated JSON format and return it as a http response message. REST API's also needed to add and modify rows in the SQL control table.



Keywords

REST API, Azure, Azure Functions, SQL, Datalake, C#

SISÄLLYS

# 1 JOHDANTO

Avoimesta datasta puhuttaessa REST rajapinnat ovat konkreettinen Infrastruktuuri, jolla dataa voidaan hyödyntää ja siirtää paikasta toiseen. Nykyaikana yleistyneet erilaiset mobiilisovellukset käyttävät REST rajapintoja lähes poikkeuksetta ulkopuolisiin yhteyksiin, kuten datan hakemiseen. Myös suurin osa moderneista websovelluksista, jotka käyttävät JavaScriptillä toteutettua selainpään logiikkaa tarvitset usein REST rajapintoja saadakseen palvelimilla olevaa dataa käyttöönsä. (Digia 2016. API:t ovat modernin integraatiostrategian ydin, Tero Kivisaari 2016).

Opinnäytetyön tavoite oli toteuttaa REST palvelu, joka tehtiin IT-yritys Enfo Oyj:n asiakkaalle. Enfo Oyj on suuri pohjoismainen IT-palvelutalo, jossa on töissä yli 900 eri IT-alan erikoisasiantuntijaa. Enfon asiakkailleen tarjoamiin palveluihin kuuluu mm. hybridialustat eli konesalit sekä julkinen- ja yksityinen pilvi, datan hallinta ja integraatiot ja dataan pohjautuvat digitaaliset palvelut. Enfon liikevaihto oli vuonna 2019 n. 121,5 miljoonaa euroa. (Wikipedia 2020, Enfo Oyj)

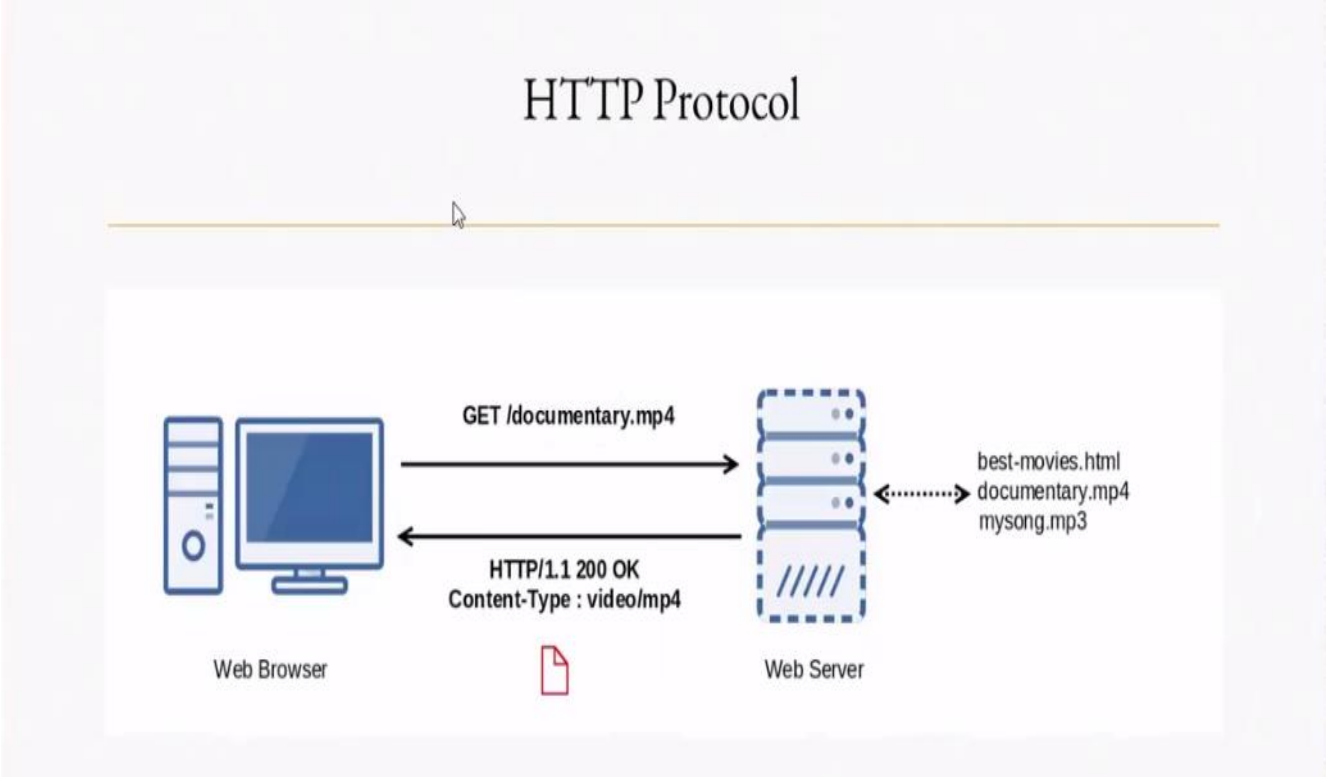
Projektien tavoite oli luoda asiakkaalle nopeaan datan hakemiseen sopiva palvelu, joka soveltuu muiden ohjelmistojen käytettäväksi. Sovelluksen piti pystyä myös hyödyntämään asiakkaan Microsoft Azure Portal ympäristöä ja sen data palveluja kuten SQL ja Datalake. Tavoite oli luoda http protokollaa käyttäen datanhakuun tarkoitettu REST rajapinta ProductCatalog ja sen varmennus rajapinta TokenConfimer Azure ympäristön omilla työkaluilla. Lisäksi asiakas tarvitsi myös ODatareader -nimisen rajapinnan, jolla pystyisi hakemaan tietyn entiteettikokoelman tauluista tietoa xml muodossa.

Tutkimuksen kohteina ovat projektien teoriasisältö, rajapinnoissa käytetyt periaatteet ja työkalut, projektityössä tarvittavat teknologiat ja rajapintojen kehitysosuuden vaiheet. Rajapintojen vaatimukset ja tarpeet vaihtuivat kehityksen aikana, joka lisäsi haasteita ja vaati jatkuvaa päivittämistä tuotantoympäristössä oleville REST rajapinnoille.

# 2 TEKNOLOGIAT TEORIA

## 2.1 HTTP (HyperText Transfer Protocol)

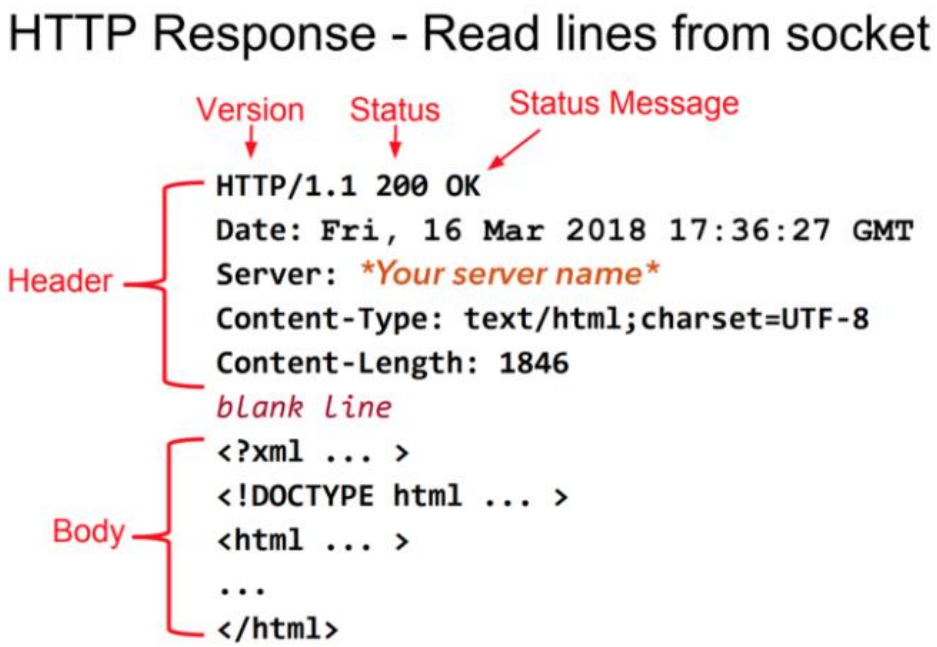
Http koostuu sanoista HyperText Transfer Protocol. Se on sovellustason protokolla, jota web-palvelimet ja selaimet käyttävät kommunikointiinsa. Http-protokolla perustuu asiakas-palvelin malliin (Client-Server model), jossa yhtä pyyntöä kohden annetaan yksi vastaus. Kuvassa 1 on esitelty http protokollan perustoimintaa, jossa web selain pyytää best movies -palvelimelta documentary.mp4 videota käyttöönsä esimerkiksi websivun linkistä. Onnistuneessa http paluuviestissä selain saa käyttöönsä tiedoston documentary.mp4 ja pystyy esimerkiksi näyttämään sen websivuston käyttäjälle. (Web-palvelinohjelmointi Java 2019a)




Kuva 1. Web selaimen ja web palvelimen välinen asiakas-palvelin malli (Medium Corporation 2020a)

Tämä tarkoittaa sitä, että jokainen pyyntö käsitellään erillisenä kokonaisuutena, eikä saman käyttäjän kahta peräkkäistä pyyntöä yhdistetä automaattisesti toisiinsa. Selain siis lähettää viestin palvelimelle, joka palauttaa Http-protokollan mukaisen vastauksen, jossa sen viestit ovat tekstimuotoisia. Viestit koostuvat riveistä, jotka muodostavat otsikon

(header) ja riveistä, jotka muodostavat rungon (body). Kuvassa 2 esimerkkinä http-paluuviestin mahdollinen sisältö ja rakenne selitettynä.




Kuva 2. http paluuviestin otsikon- ja rungon rakenne (Medium Corporation 2020b)

Otsikossa (header) näkyy http-protokollan versio (Version), paluuviestin tila (Status) ja tilanviesti (Status Message), paluuviestin esitysmuoto (esim. text/html;charset=UTF-8), ja paluuviestin pituus (Content-length). Lisäksi muita tietoja voi olla, esimerkiksi päivämäärä (Date) ja palvelin (Server). Runko (body) taas muodostuu paluuviestistä, joka voi olla json, xml tai html -formaatissa.

## 2.2 Http-statuskoodit

Statuskoodit (status code) kuvaavat http-protokollan pyynnön suorituksen onnistumisen tilaa. Statuskoodien avulla palvelin kertoo mahdollisista ongelmista tai tarvittavista lisätoimenpiteistä palvelun kutsujalle. Yleisin statuskoodi on 200, joka kertoo kaiken onnistuneen oikein (Status 200 OK). Http protokollan versiossa 1.1 on viisi pääkategoriaa vastausviesteihin.

1. Informaatioviestit, jotka ovat välillä 100 - 199 ja niistä yleisempiä ovat 100 ”Continue”, 101 ”Switching Protocols” tai 102 ”Processing” (Web-palvelinohjelmointi Java 2019a)

2. Onnistuneet tapahtumat, jotka ovat välillä 200 - 299, kuten 200 "OK”, 201 ”Created” tai 202 ”Accepted”. (Web-palvelinohjelmointi Java 2019a)

3: Tarvittavia lisätoimintoja kuvaavat virheet, jotka ovat välillä 300 - 399 ja voivat olla esim. 301 "Moved Permanently”, 302 ”Found” tai 303 “See Other” (Web-palvelinohjelmointi Java 2019a)

4: Virhe pyynnössä tai erikoistilanne ovat välillä 400 - 499 ja voivat olla esim. 401 "Not Authorized", 403 “Forbidden” tai 404 "Not Found". (Web-palvelinohjelmointi Java 2019a)

5. Virhe palvelimella tai omassa yhteydessä on välillä 500 - 599, joista yleisin 500 "Internal Server Error". Muita palvelin virheitä voi olla mm. 503 ”Service Unavailable” tai 502 ”Bad Gateway”. (Web-palvelinohjelmointi Java 2019a)

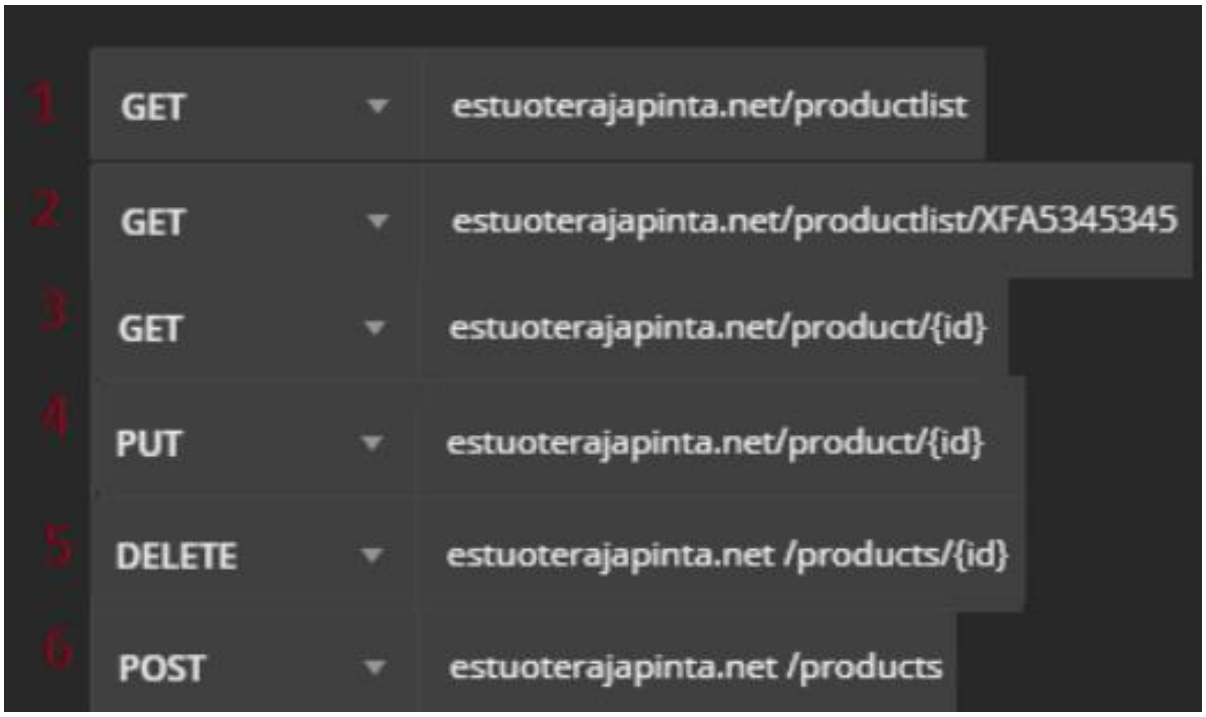
## 2.3 Representational state transfer (REST)

Representational state transfer (REST) on ohjelmointirajapintojen toteuttamiseen tarkoitettu arkkitehtuurimalli. REST-malli määrittelee sovellukset tietoa käsittelevien osien, tietokohteiden, sekä näiden yhteyksien kautta. (Web-palvelinohjelmointi Java 2019b)

Tietoa käsittelevät osat ovat selainohjelmisto ja palvelinohjelmisto. Resurssit ovat sovelluksen käsitteitä ja tietoa, kuten esimerkiksi henkilöt, tuotteet tai myynti ja osto. Eli monenlaisista asioista voi luoda resurssin tarvittaessa. Resurssikokoelmat ovat löydettävissä ja navigoitavissa esimerkiksi Kuvan 3 mukaisella GET pyynnöllä osoitteesta 1. Yksittäisille resursseille määritellään uniikit osoitteet, kuten esimerkiksi kuvan 3 kohdan 2 mukainen GET pyyntö, jossa on resurssin tunnus tai nimi. Resursseihin liittyvällä tiedolla on selkeästi määritelty ja ennalta valittu esitysmuoto, joka on yleensä HTML, JSON tai XML.

(Web-palvelinohjelmointi Java 2019b)

Resursseja ja tietoa käsittelevien osien yhteys perustuu tässäkin tapauksessa tyypillisesti asiakas-palvelin -malliin (kuva 1). Asiakas tekee esimerkiksi GET pyynnön ja palvelin kuuntelee ja käsittelee vastaanottamiaan pyyntöä sekä palauttaa siihen vastauksen (response). (Web-palvelinohjelmointi Java 2019b)




Kuva 3. Esimerkki tuotelistan http pyynnöt ja linkit

### 2.3.1 REST-rajapinnat web-sovelluksissa

Http-protokollan yli käsiteltävillä REST-rajapinnoilla on tyypillisesti seuraavat ominaisuudet: Juuriosoite resurssien käsittelyyn esimerkiksi aiemmin käytetty kuvan 3 kohdan 1 GET pyyntö. Palautusmuodon määrittelevä mediatyyppi HTML, JSON, XML tai joku muu, joka kertoo asiakkaalle, miten resurssiin liittyvää dataa pitää käsitellä. Resursseja voidaan käsitellä http-protokollan eri metodeilla kuten GET, POST, PUT, DELETE. (Web-palvelinohjelmointi Java 2019b)

Esimerkiksi verkkokaupan tuotteiden käsittelyyn ja muokkaamiseen tarkoitettu rajapinta voisi olla esimerkiksi seuraavanlainen: lähetetty GET pyyntö kohdan 1 osoitteeseen palauttaisi verkkokaupan kaikki tuotteet. GET-pyynnöt eivät tarvitse otsikkotietoja http/1.1 protokollan vaatiman ”Host” otsikon lisäksi. Mahdolliset kyselyparametrit lähetetään palvelimelle osana haettavaa osoitetta. Esimerkkinä tuotelistan Json-formaatissa oleva paluuviesti on esitetty kuvassa 4. (Web-palvelinohjelmointi Java 2019b)

```
{
    "Name": "ProductList",
    "RetailCode:": "55465ADF34245X",
    "Products": [{
        "Id": "XFA5345345",
        "Name": "ScooterX2020",
        "Data": [""]
    }, {
        "Id": "YTA5356456",
        "Name": "BicycleX2020",
        "Data": [""]
    }, {
        "Id": "ZFE563Q5345",
        "Name": "MotorcycleZERO2020",
        "Data": [""]
    }, {
        "Id": "FER62654600",
        "Name": "MotorcycleFIV2020",
        "Data": [""]
    }]
}
```


Kuva 4. Tuotelistan JSON -rakenne paluuviestissä

GET pyyntö lähetetään kuvan 3 kohdan 3 osoitteeseen, jossa id on tietyn yksittäisen tuotteen tunniste, joka palauttaa kyseisen tuotteen tiedot. Id voisi olla esimerkiksi satunnaisesti generoitu merkkijono, kuten kuvassa 4 JSON tuotelistan vastauksen ensimmäisen tuotteen Id, eli ”XFA5345345”. (Web-palvelinohjelmointi Java 2019b)

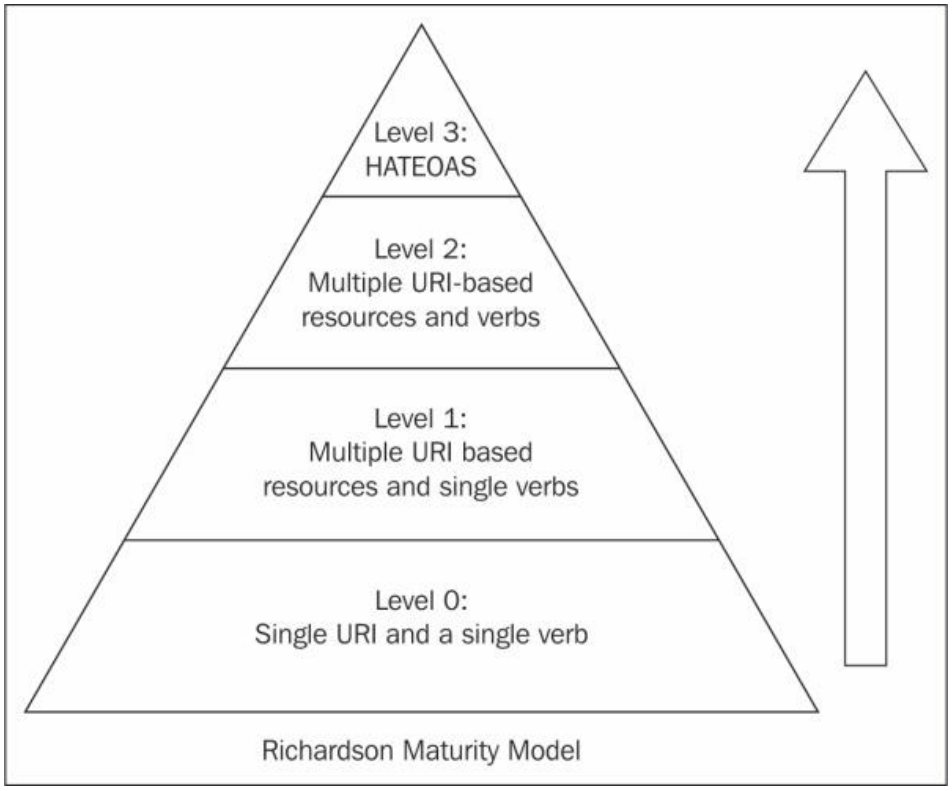
PUT pyyntö kohdan 4 osoitteeseen muokkaa kyseisen tuotteen tietoja. Tuotteen uudet tiedot lähetetään osana pyyntöä. DELETE pyyntö kohdan 5 osoitteeseen poistaa tuotteen tietyllä Id arvolla. (Web-palvelinohjelmointi Java 2019b).

POST pyyntö kohdan 6 osoitteeseen luo uuden tuotteen verkkokauppaan pyynnössä lähetettävän datan pohjalta. POST-pyyntötapaa käytetään yleisesti tiedon lähettämiseen ja lisäämiseen palvelimelle. Käytännön ero POST- ja GET-kyselyn välillä on se, että POST-tyyppisillä pyynnöillä kyselyparametrit liitetään pyynnön runkoon. Rungon sisältö ja koko määritellään otsikko-osiossa. POST-kyselyt mahdollistavat musiikin, kuvien ja videoiden eli multimedioiden lähettämisen palvelimelle/palvelimelta. (Web-palvelinohjelmointi Java 2019b)

Datan muoto on rajapinnan toteuttajan päätettävissä. Yleisimmin käytetty datamuoto on JSON, koska sen käyttäminen selainohjelmistoissa käytettävällä JavaScript koodilla on helpointa. Myös palvelinohjelmistot tukevat olioiden (objects) muuttamista JSON-muotoon. (Web-palvelinohjelmointi Java 2019b)

### 2.3.2 REST-toteutuksen kypsyystasot (RMM)

RMM eli Richardson Maturity Model jaottelee REST-toteutuksen neljään tasoon, joista kukin tarkentaa toteutusta. Richards Maturity Model pyramidi on esitetty kuvassa 5. (Web-palvelinohjelmointi Java 2019b)




Kuva 5. Richardson Maturity model -pyramidi (Rest API Tutorial 2020)

Tason 0 palveluissa HTTP-protokollaa käytetään pääsääntöisesti väylänä vain viestien lähettämiseen ja vastaanottamiseen, joten http-protokollan käyttötapaan ei oteta kantaa. Esimerkkinä voi vaikka olla tarkistuskoodin lähettäminen kuvan 3 kohdan 2 linkillä, joka palauttaisi vastauksen tuotteen löytyvyydestä tuotelistalta yksinkertaisesti löytyi tai ei löytynyt, eli: ”Product found” tai ”Product not found”, mutta ei palauttaisi mitään viestiä itse tuotteesta tai sen tiedoista paluuviestissä. (Web-palvelinohjelmointi Java 2019b)

Tason 1 palvelut käsittelevät palveluita resursseina. Resurssit kuvataan palvelun osoitteena, kuten vaikka aiemmin mainittu kohdan 1 tuotelistalinkki. Linkki sisältää tuotelistan, josta resursseja voidaan hakea myös tunnisteiden, eli id:n perusteella, kuten kohdan 3 linkissä. 0 tasoon verrattuna käytössä on nyt konkreettisia resursseja, jotka voidaan hakea kaikki yhtäaikaisesti tai tehdä yksittäisen resurssin haku id tunnuksen perusteella.

(Web-palvelinohjelmointi Java 2019b)

Tasolla 2 resurssien käsittelyyn käytetään kuvaavia http-pyyntötyyppejä. Resurssin pyyntö tapahtuu GET-metodilla, ja resurssin tilan muokkaaminen PUT tai POST-metodilla. Tämän lisäksi palvelun vastaukset kuvaavat tapahtuneita toimintoja. Esimerkiksi jos palvelu luo resurssin, vastauksen tulee olla statuskoodi 201, eli luotu (created) sekä mahdollisesti jokin viesti, joka viestittää paluussa resurssin luomisen onnistumisesta käyttäjälle: ”New product X54325345 added succesfully”. Oleellista tällä tasolla on pyyntötyyppien erottaminen sen perusteella, että muokkaavatko ne palvelimen dataa vai eivät. GET-metodi hakee valmiina olevan datan, kun taas POST, PUT ja DELETE metodit muokkaavat dataa jollakin tavalla. (Web-palvelinohjelmointi Java 2019b)

Taso 3 (HATEOS) sisältää tasot 1 ja 2, mutta lisää käyttäjälle mahdollisuuden ymmärtää palvelun tarjoama toiminnallisuus palvelimen vastausten perusteella. (Web-palvelinohjelmointi Java 2019b)

HATEOAS (Hypertext As The Engine Of Application State) tarkoitus on helpottaa käyttäjän navigointia rajapinnan resursseissa lähes yhtä vaivattomasti kuin tavallisella websivustolla. Esimerkiksi kun käyttäjä tekee GET pyynnön verkkokaupan kuvan 7 linkistä hän saisi API vastauksessa tuotteen tiedot normaalisti, mutta myös esimerkiksi linkin kyseisen tuotteen kuviin verkkokaupassa. Tavanomainen API vastaus tuotteesta ”XR5353455” näyttäisi mahdollisesti kuvan 6 mukaiselta, Kun taas HATEOS tason 3 hyperlinkillä kuvilla (images) varustettu vastaus on esitelty kuvassa 7. (Web-palvelinohjelmointi Java 2019b)

```
<Product>
    <id> XR5353455</id>
    <name>Microcar-XE</name >
    <data>[....]</data>
</Product>
```

Kuva 6. Tuotteen XML esimerkki (Studytonight 2020. HATEOAS - Important Concept For REST API, tekijän muokkaus)

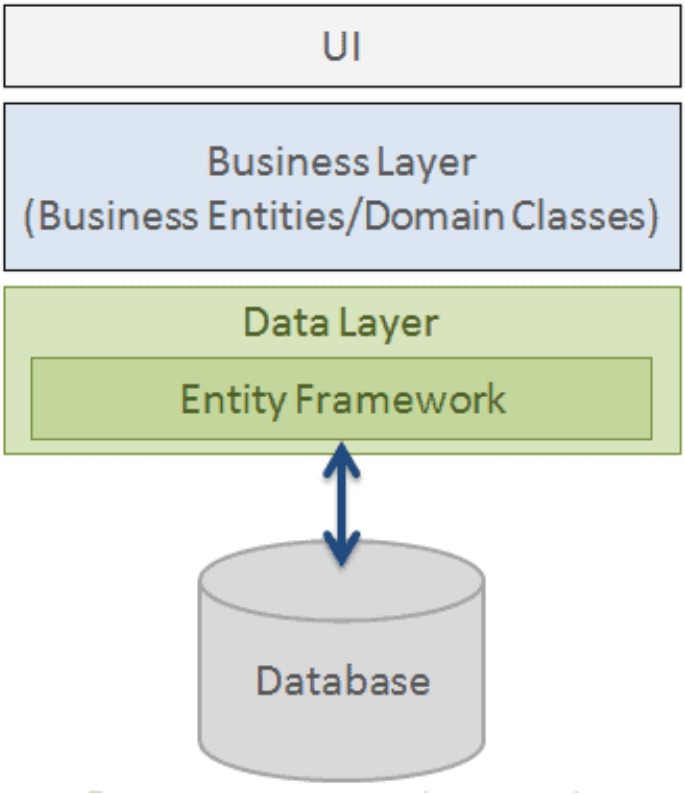
```
<Product>
    <id> XR5353455</id>
    <name>Microcar-XE</name >
    <data>[....]</data>
    <link rel="images" href="/estuoterajapinta.net/productlist/ XR5353455/images">
</Product>
```

Kuva 7. Hyperlinkillä varustettu tuotteen XML esimerkki (Studytonight 2020. HATEOAS - Important Concept For REST API, tekijän muokkaus)

## 2.4 Entity Framework

Entity Framework on Microsoftin tukema avoimen lähdekoodin ORM framework .NET pohjaisille ohjelmistoille. ORM tulee sanoista ”Object-relational mapping”. Se on ohjelmointi tapa, jolla pystytään muuttamaan tietokannan datasta olioiksi ja päinvastoin olio-ohjelmoinnin keinoin. (Microsoft 2020a, tekijän käännös)

Entity Framework antaa kehittäjille mahdollisuuden työstää dataa käyttämällä alustaspesifioituja luokkia ja niiden olioita ilman, että tarvitsee keskittyä tietokannan tauluihin ja sarakkeisiin. Tämä yksinkertaistaa koodia ja vähentää sen määrää, kun tehdään ja/tai ylläpidetään data pohjaisia ohjelmistoja. Kuvasta 8 ilmenee, että Entity Framework sijaitsee data kerroksessa eräänlaisena lisäosana tai apuvälineenä tavanomaisen ohjelmiston arkkitehtuurissa. (Microsoft 2020a, tekijän käännös)




Kuva 8. Entity Framework ohjelmisto arkkitehtuurissa (Microsoft 2020a)

### 2.4.1 Entity Framework ominaisuudet

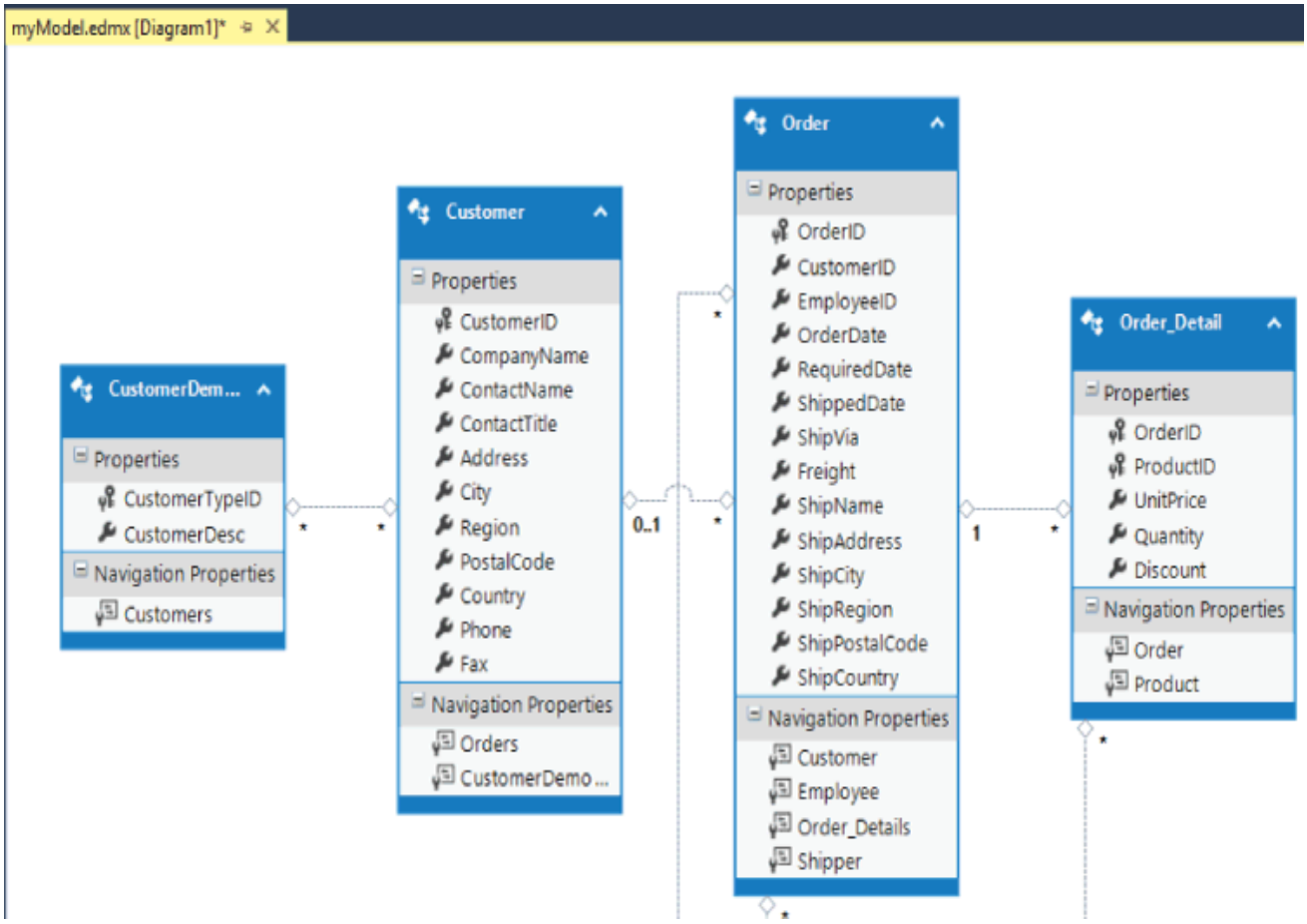
Entity Framework on järjestelmäriippumaton, joten se toimii Windows-, Linux- ja Mac järjestelmissä. Entity Frameworkin kyselyt mahdollistavat LINQ kyselyiden käytön, eli C# tai VB.NET datan hakemiseen tietokannoista. Tietokannan tuottaja kääntää LINQ kyselyt tietokanta spesifeihin kysely kieliin (esim. SQL). Entity Framework mahdollistaa myös normaalin SQL kyselyn tekemistä suoraan tietokantaan. Tallennukset Entity Frameworkissa toimivat siten, että Entity Framework voi kutsua SavesChanges -metodia suorittaakseen INSERT, UPDATE ja DELETE komentoja tietokannalle, mikäli entiteettien sisältö on muuttunut. SaveChanges -metodista on myös asynkroninen versio, SavesChangesAsync.

(Microsoft 2020a, tekijän käännös)

### 2.4.2 Entity Framework käyttötapa

Entity Frameworkissa pystytään hakemaan dataa monista eri tauluista yhtäaikaisesti. Tämän käyttötavan mahdollistaa Entity Frameworkin edmx -tiedostotyyppi. Siinä voidaan nopeasti rakentaa tietokantamalli niistä tauluista, joita halutaan hakea. Kuvassa 9 on esimerkkinä myModel.edmx tiedosto, johon on otettu taulut: CustomerDemographics, Customer, Order ja Order_Detail. Tauluille on myös määritelty relaatiot Entity Frameworkin toimesta.

Entity Framework tarvitsee normaalin SQL kantayhteyden sisältävän merkkijonon (connection string), josta ilmenee palvelin, käyttäjäntiedot ja tietokannannimi. Näillä tiedoilla pystyy valitsemaan haluamansa taulut, joista Entity Framework luo edmx tiedoston automaattisesti.



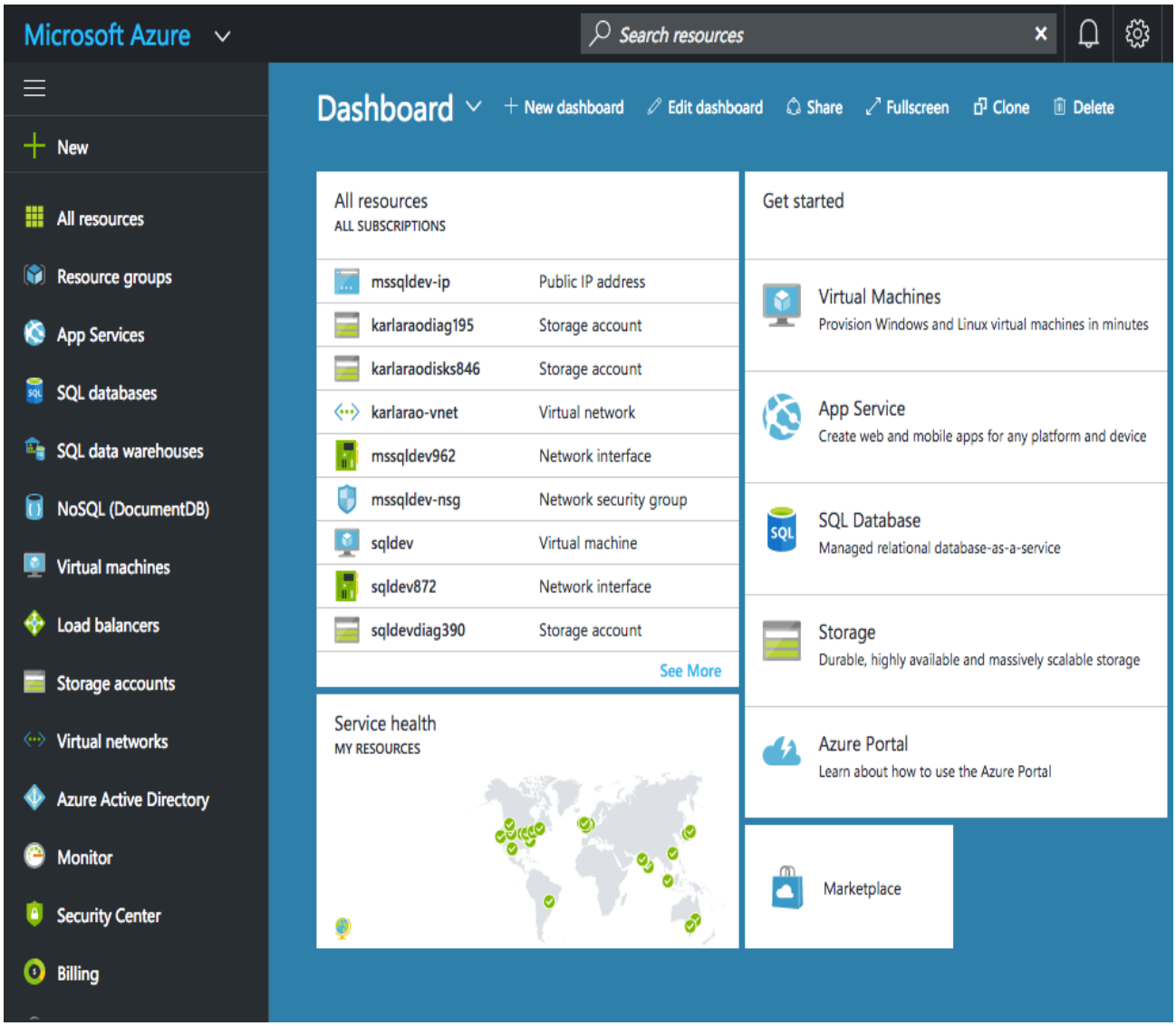

Kuva 9. Entity Framework edmx tiedosto esimerkki

# 3 MICROSOFT AZURE PALVELUT

## 3.1 Azure Portal

Azure Portal on Azure pilviympäristö, joka toimii puhtaasti pilvipalveluna tai sen voi yhdistää osaksi paikallista IT-infrastruktuuria. Azure Portalin palveluihin kuuluu yli 600 eri Azure -palvelua, joihin lukeutuu mm. tietokantoja, tallennustilaa, data-analytiikkaa ja verkkopalveluja. Azure Portalista voi hallita, muokata ja luoda kaikkia näitä palveluita. Azure portal ja sen palvelut tukevat useita eri ohjelmointikieliä, kuten C#, C++ ja Java ja siinä on myös hyvä valikoima Linuxin kanssa yhteensopivia palveluita. Azure Portal käyttöliittymän dashboard on esitettynä kuvassa 10. Yleisiä Azure Portal palveluita ovat mm. SQL, Datalake, AppService, Functions ja DataBricks. (Itewiki 2020)

Azure Portal tarjoaa käyttäjille Azuren ns. serverless -palveluja. Serverless tarkoittaa käytön perusteella maksettua palvelinkapasiteettia Azure Portal pilvipalvelussa. Azuren Serverless -tarjontaan kuuluu kokoelma erilaisia palveluita, joille ominaista on se, että niissä ei itse tarvitse huolehtia palvelimesta, isännöinnistä tai ajoympäristöstä. Lähes kaikki Azuren palvelut, virtuaalikoneita ja muutamaa muuta palvelua lukuun ottamatta, ovat juuri tällaisia rakenteeltaan. Serverless -palveluissa suoritusympäristöstä itse huolehtiminen on poistettu kehittäjältä lähes kokonaan. (Azure functions ja API Apps integraation avuksi, Mika Berglund 2017)

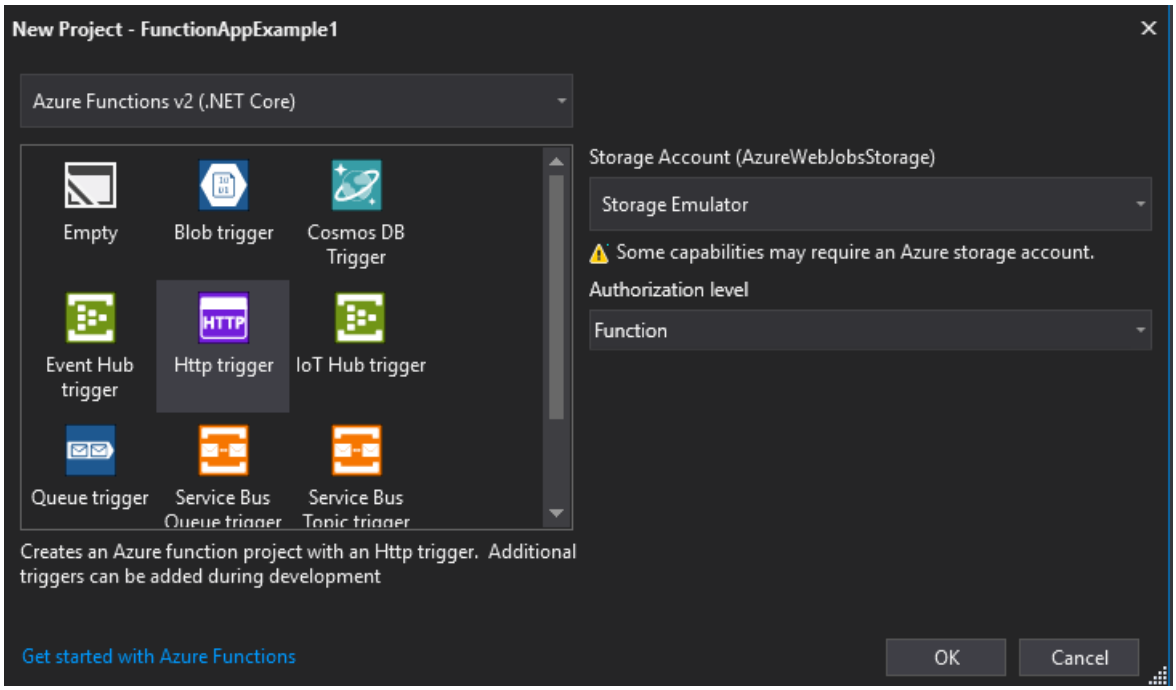



Kuva 10. Azure portal dashboard käyttöliittymä

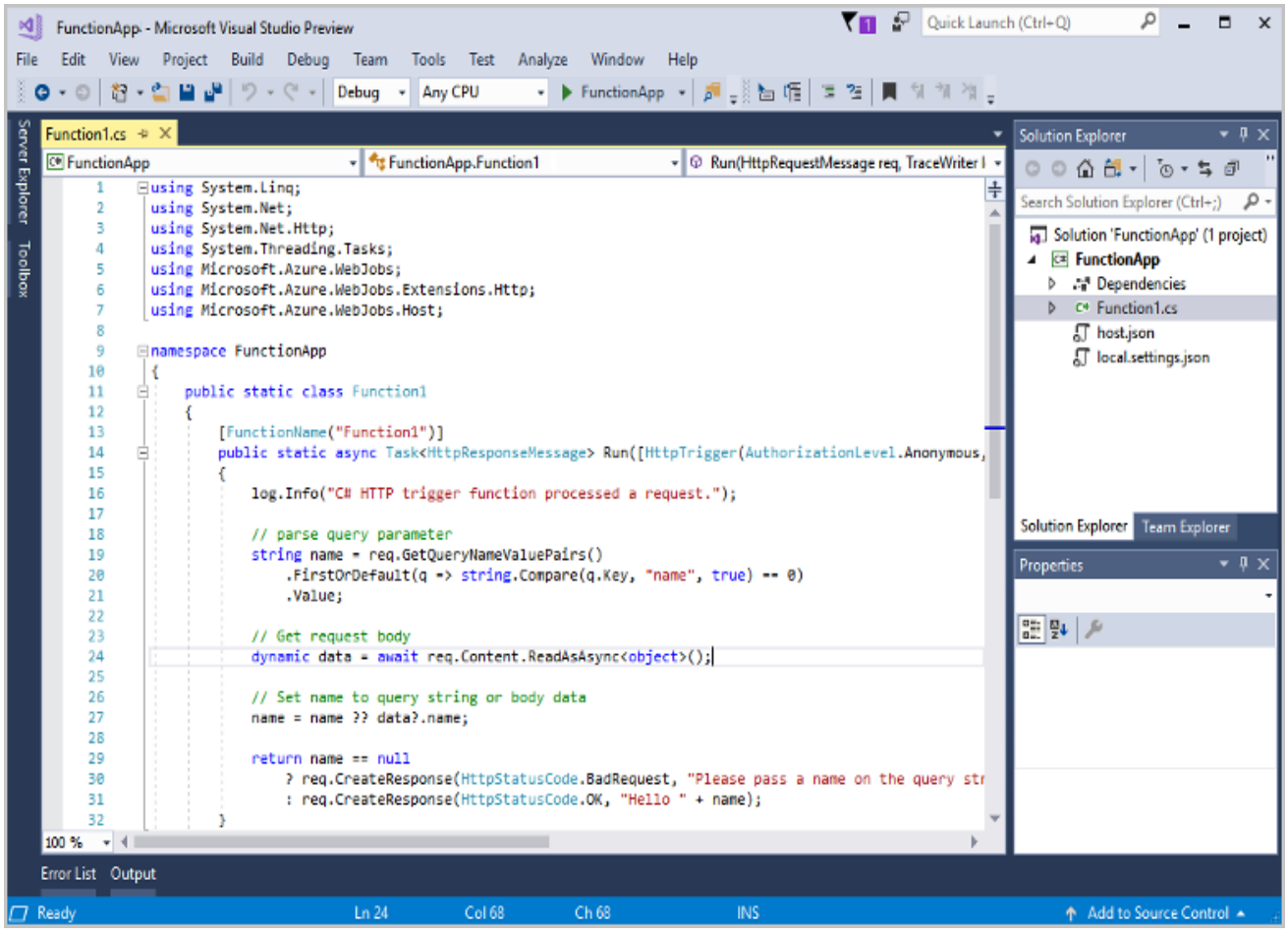
## 3.2 Azure Functions

Functions on Azuren palvelu, joka tarjoaa Microsoft Azure ympäristön oman koodin julkaisuun funktioiden muodossa. Funktioita voidaan sitten kutsua muista sovelluksista. Funktioihin voi esimerkiksi laittaa aikakutsun, eli sitä kutsutaan tietyn ajan välein tai aina tietyssä kellon ajassa. Functions -palvelun erona perinteisempään sovellusten suoritusympäristöön, on se, että sinne julkaistaan vain sovellusten osia, funktioita, eikä kokonaisia sovelluksia. Tämä on hyvä keino saada esimerkiksi samoja ominaisuuksia käyttävät sovellukset pilkottua pienempiin osiin, ja nämä pienemmät osat eli funktiot, tulevat tehokkaammin hyödynnettyä eri sovellusten kesken. (Azure functions ja API Apps integraation avuksi, Mika Berglund 2017)

Azure funktiota voidaan lähteä kehittämään suoraan Visual Studiosta Functions -templaten avulla, jossa voi suoraan valita funktion rakenne ja toiminnallisuus. Funktion toiminnallisuudeksi voidaan valita http trigger, eli http pyynnöllä käynnistettävänä, kuten yleisesti REST rajapinnoissa tai esimerkiksi Blob trigger ja Cosmos DB trigger, jotka käynnistyvät, kun Blob Storage- tai Cosmos DB -palvelussa tapahtuu muutoksia. Lisäksi funktio voidaan laittaa käyntiin kerran tunnissa, päivässä, kuukaudessa tai jopa vuodessa tiettynä ajankohtana itsestään, vaikkapa joka päivä klo 00:00. Yleisimmät vaihtoehdot valmiiksi alustoiksi on esitetty kuvassa 11. Esimerkki yksinkertaisesta Http GET pyynnöllä toimivasta Azure funktiosta on kuvassa 12.

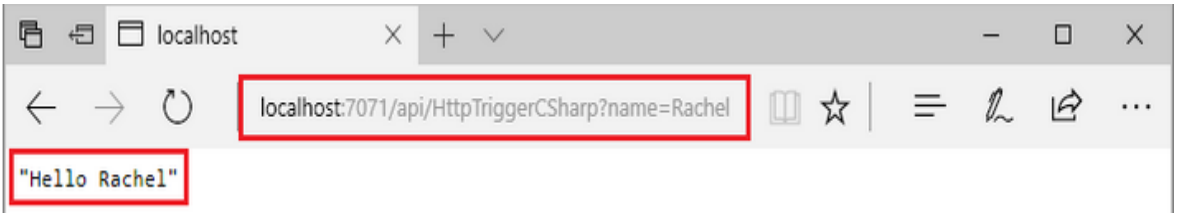



Kuva 11. Visual Studio Azure function new template esimerkki



Kuva 12. Azure funktio koodiesimerkki (Microsoft 2020d)

Kuvan 12 Azure funktiota kutsutaan GET pyynnöllä sen palveluosoitteesta ”name” kysely parametrillä. Rivillä 19 merkkijono muuttujaan ”name” alustetaan käyttäjän lähettämä kyselyparametri ”name”. Seuraavaksi rivillä 24 alustetaan http pyynnön runko (body) dynaamiseen muuttujaan ”data”. Rivillä 29 alkavassa palautuksessa katsotaan, onko muuttuja ”name” null, eli tässä tapauksessa tyhjä. Jos nimi parametri on tyhjä, palautetaan http paluuviestissä käyttäjälle teksti ”Please pass a name on the query string! Mikäli käyttäjä kuitenkin syötti kyselyparametri nimeen (name) merkkijonon, esim. ”Rachel”, niin funktio palauttaa tekstin ”Hello Rachel” ja palautuksen tila olisi 200 OK. Esimerkki funktion kysely parametrin sisältävästä vastauksesta esitetään kuvassa 13.




Kuva 13. Azure funktion paluuviesti selaimessa (Microsoft 2020d)

## 3.3 App Service

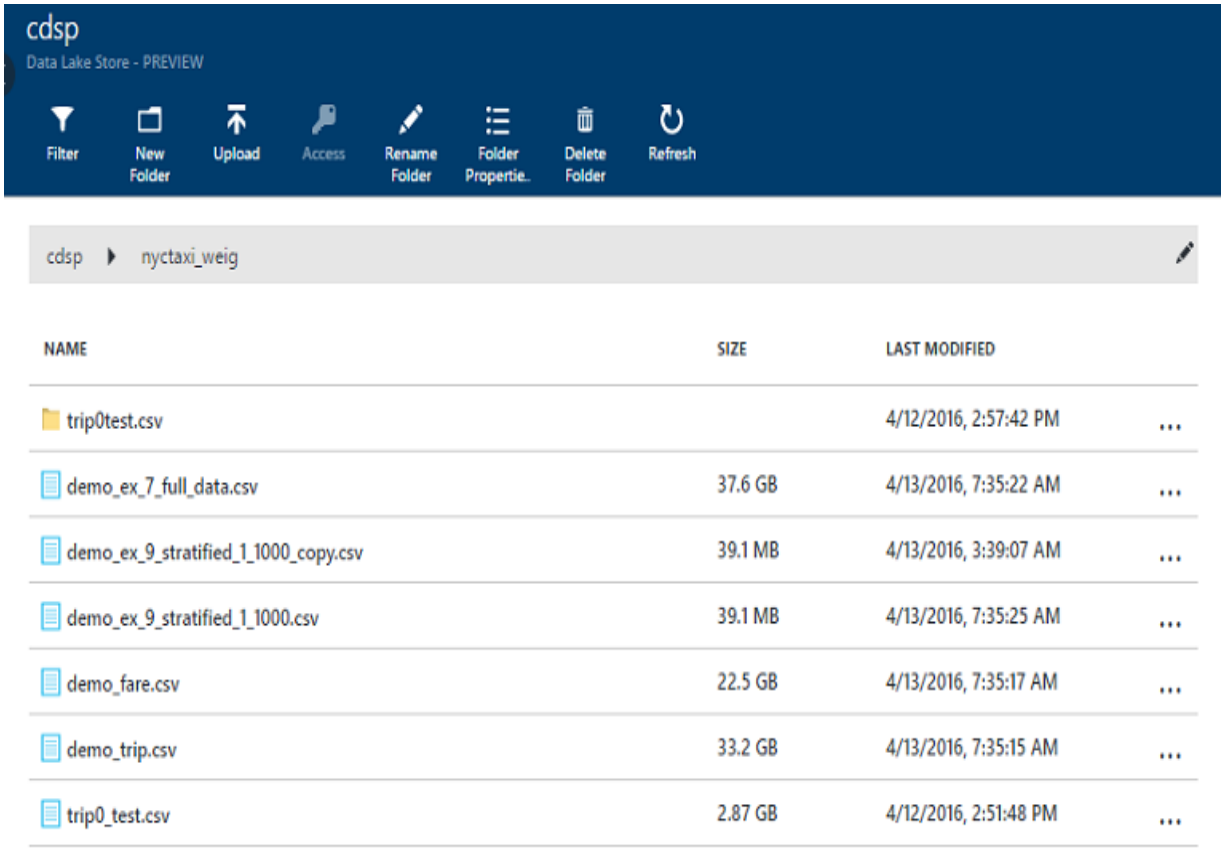
App Service on http protokollaan pohjautuva työkalu, johon voi julkaista omia websivustoja, websovelluksia tai rajapintoja. Tuetut ohjelmointikielet ja teknologiat mm. .NET, .NET Core, Node.js, Java, Python ja PHP. App Service tukee käyttöjärjestelminä Windowsia ja Linuxia. (Microsoft 2020f, tekijän käännös)

App Servicen päätarkoitus on lisätä Azuren ominaisuudet jo valmiisiin sovelluksiin. Hyödyt ovat mm. Azuren tietoturva, automaattinen skaalaus ja automaattinen toimivuuden ja vikojen seuranta. Lisäksi App Servicessä olevat sovellukset voivat nyt käyttää helpommin Azuren muita työkaluja, kirjastoja ja metodeita hyödykseen, kuten Azure Datalake, Azure KeyVault ja Azure SQL. (Microsoft 2020f, tekijän käännös)

## 3.4 Azure Datalake

Azure Datalake on skaalautuva datan tallennusvarasto ja analyyttinen palvelu. Datalakeen voidaan varastoida lähes rajattomasti dataa tiedostojen muodossa eri lähteistä. Palvelu ei sinänsä rajoita käyttäjien määrää, tiedostojen kokoa tai datan määrää, joka on tallennettu Datalakeen, mutta se laskuttaa käytöstä tarvittavan tallennus määrän ja käytettyjen analyyttisten palveluiden mukaan. Datalake tarjoaa myös C# kirjaston nimeltään AdlsClient, jolla käyttäjä voi kirjoittaa omaa koodia ohjelmiinsa mm. datan siirtoa, analysointia ja muita operaatioita varten. (Microsoft 2020b, tekijän käännös)

Azure alusta tarjoaa myös helposti käytettävän Datalake Explorer palvelun, jossa tiedostoja voi katsoa, muokata, lisätä ja poistaa manuaalisesti. Tästä ominaisuudesta on paljon apua, mikäli tarvitsee esimerkiksi nopeasti tarkastaa tietyn tiedoston sisältö tai muokata sieltä jotain. Kuvassa 14 on esimerkki Datalake Explorer käyttöliittymästä ja cdsp -nimisen datalake varaston sisällöstä. (Microsoft 2020b, tekijän käännös)



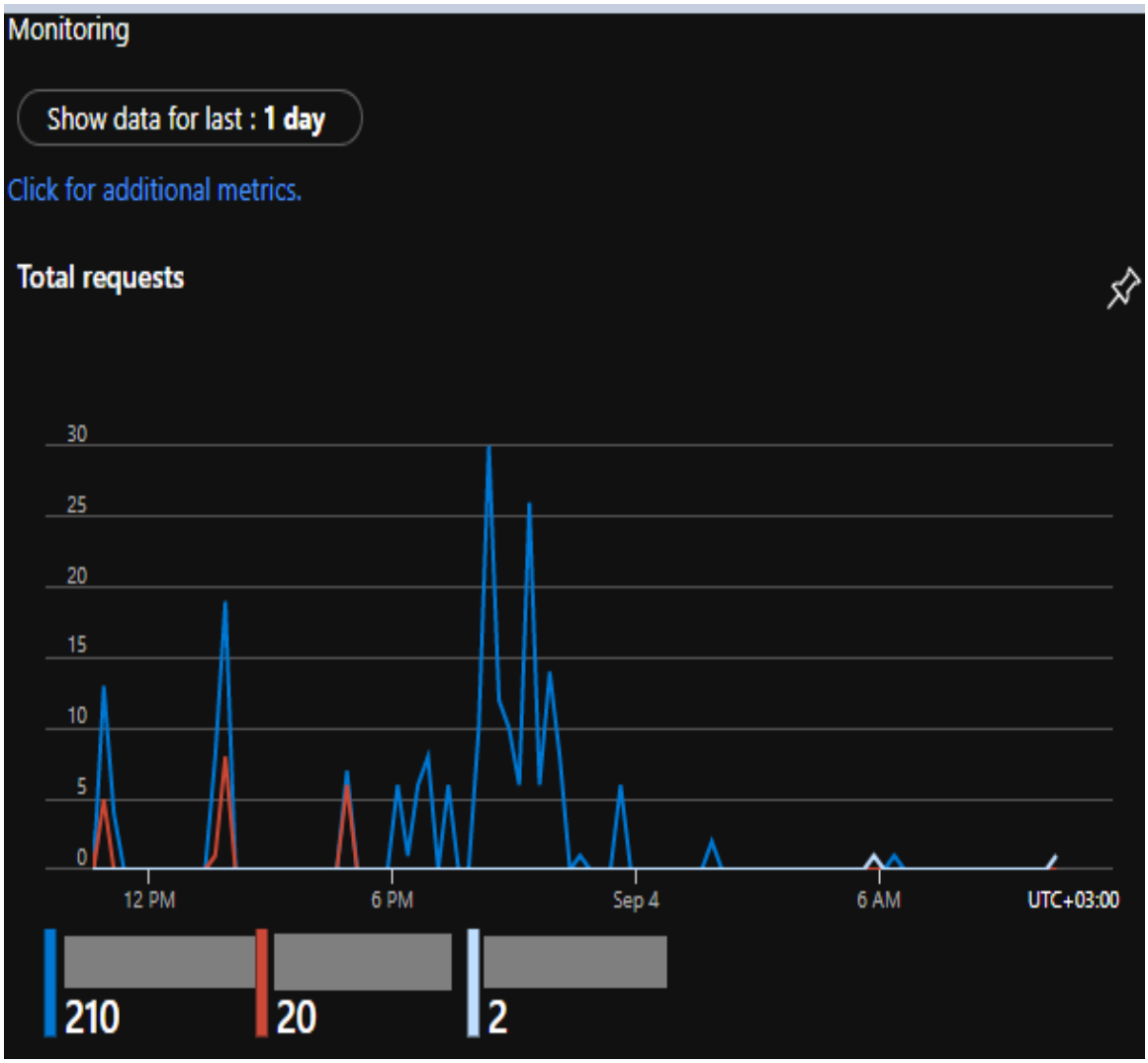

Kuva 14. Azure Datalake Store kansio

## 3.5 Azure KeyVault

KeyVaultia käytetään pääosin suojasäilönä käyttötokeneille, salasanoille, sertifikaateille ja API avaimille, eli yleisesti kaikelle arkaluontoiselle datalle, joita ohjelmat käyttävät tai saattavat sisältää.

KeyVaultiin rakennetun avainten monitoroinnin kanssa on myös helppoa seurata, milloin avaimia ja muita salaisuuksia pystytään käyttämään. KeyVaultissa myös voidaan määrittää, mitkä käyttäjät ja/tai sovellukset pystyvät mitäkin avainta tai muuta salaisuutta käyttämään tai näkemään. (Microsoft 2020c, tekijän käännös)

Azure ympäristön KeyVault monitorointi näyttää erilaisia mittauksia halutulta ajalta siellä sijaitseviin salaisuuksiin (secrets) ja avaimiin (keys). Kuvassa 17 on esimerkki kaikkien ”keys” tai ”secrets” pyyntöjen mittauksesta (Total requests) yhden vuorokauden ajalta.

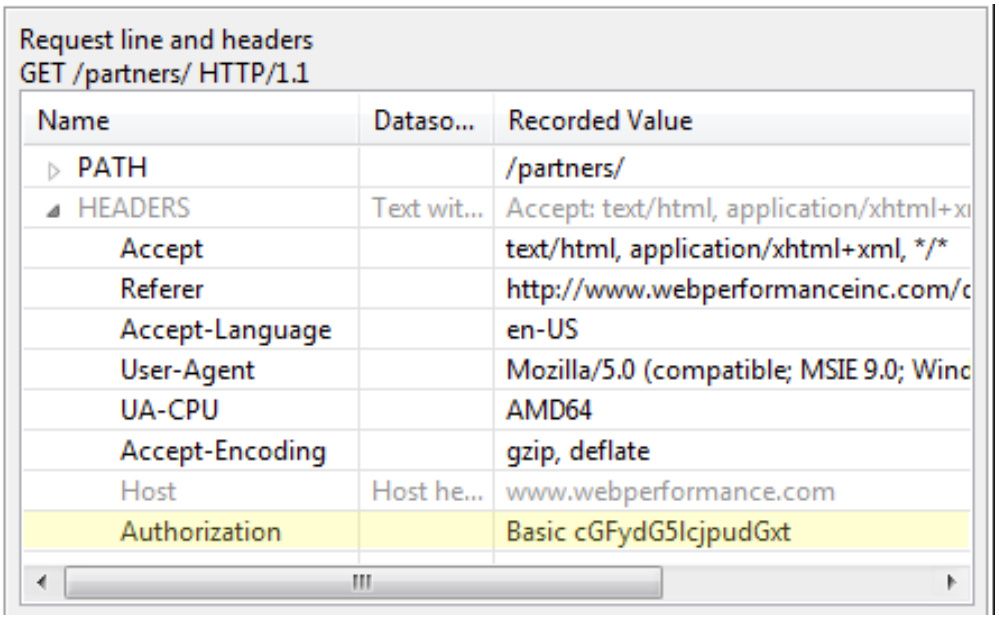



Kuva 17. Azure KeyVault monitorointi ikkuna

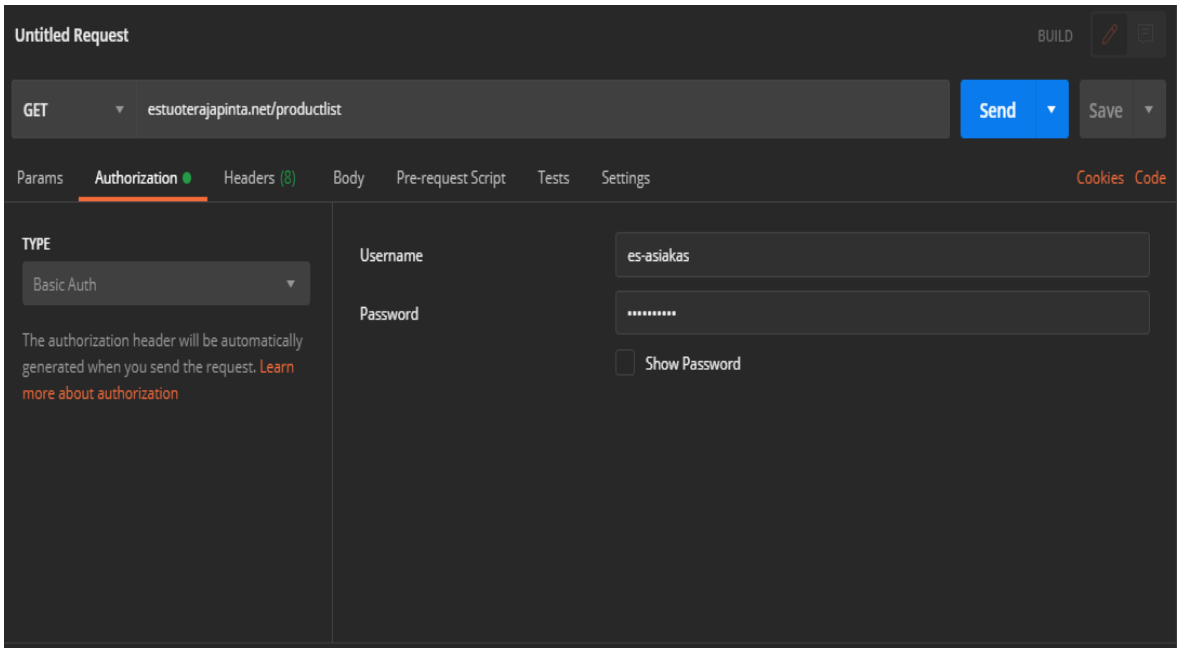
# 4 MUUT KÄYTETYT PALVELUT

## 4.1 Basic Authentication

Basic Authentication on yksinkertainen käyttäjän varmistustapa, joka on rakennettu http protokollaan. Asiakas voi lähettää http pyynnön mukana Authorization otsikon kanssa kirjainyhdistelmän, joka sisältää sanan ”Basic” + välilyönnin, jonka jälkeen kaksi sanaa Base64 muodossa: ”username:password”. Tällöin kutsun vastaanottanut sovellus voi tarkastaa Authorization otsikon sisällön, ja verrata niitä omaan suojaus järjestelmäänsä, kuten Azure KeyVaultiin. Kuvassa 15 on esitetty basic authentication otsikko (header), jossa näkyy base64 salattu käyttäjätunnus keltaisella pohjalla. (SmartBear Software 2020, tekijän käännös)




Kuva 15. Basic Authentication otsikko (Webperformance 2020a)




Kuva 16. Postman client basic authentication välilehti

Otsikon (header) käyttäminen http pyyntöjen tekemiseen tarkoitettujen ohjelmien, kuten Postman, on yksinkertaista sen Authorization välilehden kautta, kuten kuvassa 16 on esitettynä. Clientin Authorization välilehdessä voi syöttää username ja password yhdistelmän kirjautumisikkunan mukaisesti.

## 4.2 .NET kirjastot

Azuren portalin ulkopuoliset työkalut, teknologiat ja kirjastot olivat tärkeä osa REST rajapintojen toiminnallisuutta. Newtonsoft vapaan lähdekoodin (open source) kirjasto ja microsoftin oma System.Data.SqlClient tulivat tarpeelliseksi, kun datan muokkaus ja SQL taulukoiden käyttö olivat pakollisia ominaisuuksia rajapintojen ominaisuuksissa.

### 4.2.1 Newtonsoft

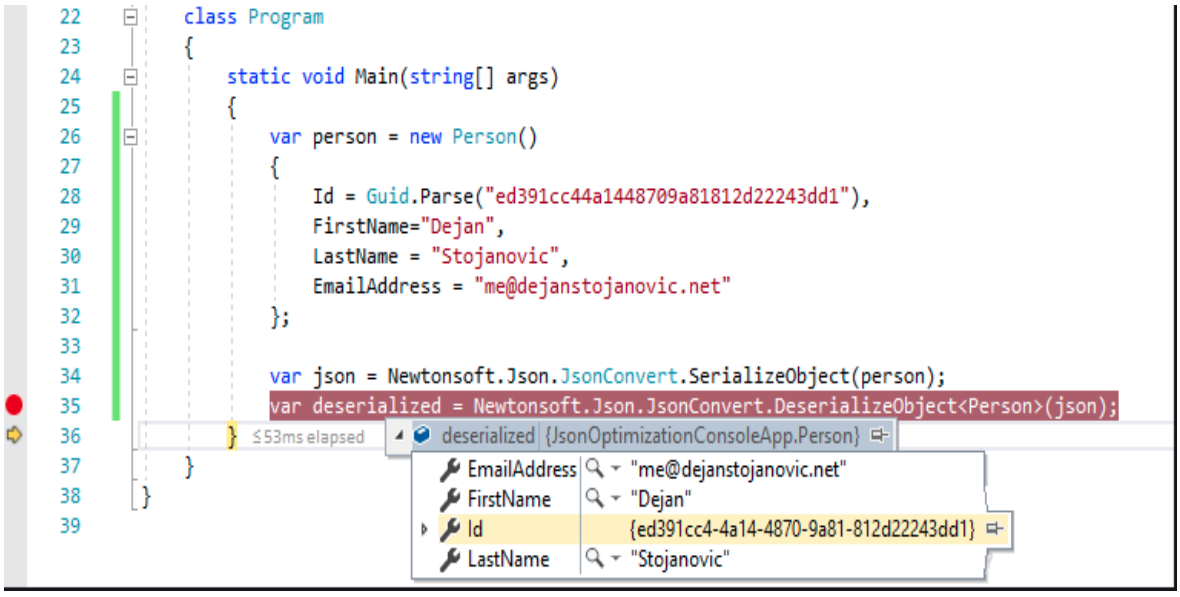
Newtonsoft on .NET kirjasto, jonka omilla luokilla ja metodeilla helpotetaan mm. data tyyppien muunnosta. Näissä projekteissa pääsääntöisesti tarkoitus on muuttaa tavallisen .txt tiedoston sisältö C# merkkijonosta JSON muotoon, jolloin sovellukset saavat tiedoston sisällön käyttöönsä halutussa formaatissa.

JSON formaatti (***J**ava**S**cript **O**bject **N**otation*) on samankaltainen kuin Javascript objekti rakenne, mutta on silti täysin riippumaton Javascriptistä. JSON näyttää esimerkiksi Kuvan

18 mukaiselta. Kuvassa 19 on esimerkki yksinkertaisesta datatyypin muunnoksesta Newtonsoft kirjaston avulla.

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

Kuva 18. JSON rakenne esimerkki




Kuva 19. Newtonsoft Datatyypin muunnos oliosta merkkijonoksi (Stojanovic 2018)

Kuvassa 19 on kehittäjän tekemä Person -luokka, jonka Newtonsoft SerializeObject metodi muuttaa merkkijono muotoon muuttujaan ”json”, kuten kuvassa 20 on esitetty. Tämän jälkeen merkkijono muuttuja ”json” muutetaan takaisin C# objekti Person muotoon

DeserializeObject metodilla muuttujaan ”deserialized”, jonka näkee kuvassa 19 punataustaisessa muuttujassa.

```
"{\"EmailAddress\":\"me@dejanstojanovic.net\",\"FirstName\":\"Dejan\",
\"LastName\":\"Stojanovic\"Id \":\"ed391cc4-4a14-4870-9a81-812d22243dd1\" }";
```

Kuva 20. JSON-lause merkkijono muuttujassa.

### 4.2.2 System.Data.SqlClient luokat ja metodit

SqlClient sekä sen luokat ja metodit mahdollistavat SQL tietokantojen datan hakemisen, lisäämisen, poistamisen ja muokkaamisen .NET ohjelmistoissa. Tärkeitä aliluokkia SqlClient:ssä ovat mm. SqlConnection, SqlCommand ja SqlDataReader. Kuvassa 21 on esitetty komentoluokka ja sen käynnistys metodissa nimeltään CreateCommand.

SqlConnection, eli SQL yhteys on pakollinen luokka, jonka Open -metodin suorittaminen käynnistää yhteyden .NET ohjelmiston ja SQL tietokannan välillä.

```
private static void CreateCommand(string queryString,
    string connectionString)
{
    using (SqlConnection connection = new SqlConnection(
               connectionString))
    {
        SqlCommand command = new SqlCommand(queryString, connection);
        command.Connection.Open();
        command.ExecuteNonQuery();
    }
}
```

Kuva 21. Esimerkki Sql luokan yhteys- ja komentoluokista (Microsoft 2020e)

connectionString on merkkijono muuttuja, joka on erilainen riippuen käytetystä tietokannasta, mutta siihen kuuluvat aina SQL tietokannan osoite (Data Source), tietokannan nimi (catalog) ja tietokantaan oikeuttavat tunnukset, UID (käyttäjä) ja PWD (salasana). Esimerkiksi Active directory connection string merkkijono muuttuja on esimerkiksi kuvassa 22 esitetyn kaltainen.

```
"Data Source=01-testdata-sql-prod.database.windows.net;initial cata-log=TESTPRODUCTS;
Persist Security In-fo=False;UID=TESTUSER@email.com;
PWD=I6g43>Gjr934;Authentication=Active Direc-tory Password;"
```

Kuva 22. Sql yhteyden merkkijonomuuttuja esimerkki

SqlCommand on luokka, jossa on Sql lause tai varastoitu sql proseduuri, joita voi käynnistää eri metodeilla ja sen tehtävänä on suorittaa. SqlDataReader on datanlukija luokka, joka antaa .NET ohjelmistoille selkeän tavan lukea SQL tietokantojen rivejä .NET koodissa.

```
class Class1
{
    0 references | 0 changes | 0 authors, 0 changes
    private static void ReadOrderData(string connectionString)
    {
        string queryString =
            "SELECT OrderID, CustomerID FROM dbo.Orders;";

        using (SqlConnection connection =
                  new SqlConnection(connectionString))
        {
            SqlCommand command =
                new SqlCommand(queryString, connection);
            connection.Open();

            SqlDataReader reader = command.ExecuteReader();

            // Call Read before accessing data.
            while (reader.Read())
            {
                ReadSingleRow((IDataRecord)reader);
            }

            // Call Close when done reading.
            reader.Close();
        }
    }

    1 reference | 0 changes | 0 authors, 0 changes
    private static void ReadSingleRow(IDataRecord record)
    {
        Console.WriteLine(String.Format("{0}, {1}", record[0], record[1]));
    }
}
}
```

Kuva 23. SqlDataReader perustoiminta C#

Kuvan 23 koodi esimerkissä on esitetty kehittäjän luoma yksityinen metodi ReadOrderData, jossa ensin luodaan muuttuja sql tietokannan avaamiseen käyttäen SqlConnection luokan ”connection” muuttuja rivillä 18 ja 19. SqlConnection muuttujan luomiseen tarvitaan myös kuvan 23 kaltainen connectiongString merkkijono muuttuja.

Seuraavaksi luodaan SqlCommand luokan ”command” olio rivillä 21 ja 22. Olion konstruktoriin sisältyy ”connection” yhteys muuttujan lisäksi ”queryString” merkkijonomuuttuja, jossa on sql transact SELECT komento, joka tarkoittaa taulusta rivien valitsemista. Muuttuja on määritelty rivillä 15 ja 16. Tällä komennolla pystytään valitsemaan Orders taulusta rivit OrderId ja CustomerId.

Sql yhteys avataan taas Open metodilla rivillä 23, jonka jälkeen alustetaan SqlDataReader luokan muuttuja ”reader”, joka on SqlCommand luokan ”command” muuttujan suoritus metodillam johon talletetaan ExecuteReader metodin palauttama arvo, kuten on esitetty rivillä 25. ReadSingleRow metodilla luetaan SqlDataReaderin palauttamat taulukon rivit käyttäen while silmukkaa, jonka jälkeen rivit näytetään Visual Studion konsolissa Console.WriteLine -metodilla. While silmukka on esitetty riveillä 28 - 31.

Kuvassa olevan metodin ReadOrderData suorittamisen yhteydessä Visual Studion konsoli näyttäisi jokaisen rivin, joita SqlDataReader palauttaa, yksittäisenä rivinä. Tämä tarkoittaa sitä, että Metodia ReadSingleRow kutsuttaisiin niin monta kertaa, kun taulukossa Orders on rivejä. ReadSingleRow on erikseen määritelty riveillä 38 - 41 omaksi staattiseksi yksityiseksi (private) metodikseen.

SqlDataReader muuttujasta ”reader” voidaan myös ottaa käyttöön sisältö muihin muuttujiin tai palauttaa se myös ReadSingleRow metodista päämetodille ”return” komennolla.

# 5 TYÖOSUUS

## 5.1 Azure Function luominen ja kehitys

Asiakas tarvitsi REST rajapinnan datan hakemiseen, joka palautti http GET pyynnöllä omasta palvelusta dataa, kuten asiakastietoja, tuotetietoja ja mahdollisesti muuta. Tällöin asiakkaan oma ohjelma pystyi kutsumaan palvelun osoitetta ja saada datan käyttöönsä nopeasti muutamassa sekunnissa. Rajapinnan luonteen ja käyttötarkoituksen mukaan sen nimeksi tuli ProductCatalog, eli tuotekatalogi.

Lisäksi vaatimuksena oli toteuttaa varmennusrajapinta, TokenConfirmer, jonka tehtävänä oli varmistaa ProductCatalog-rajapinnalla tehdyt pyynnöt valmistuvat ja siten muokata sekä palauttaa pyynnön lopullinen status: valmistunut, eli ”Completed”.

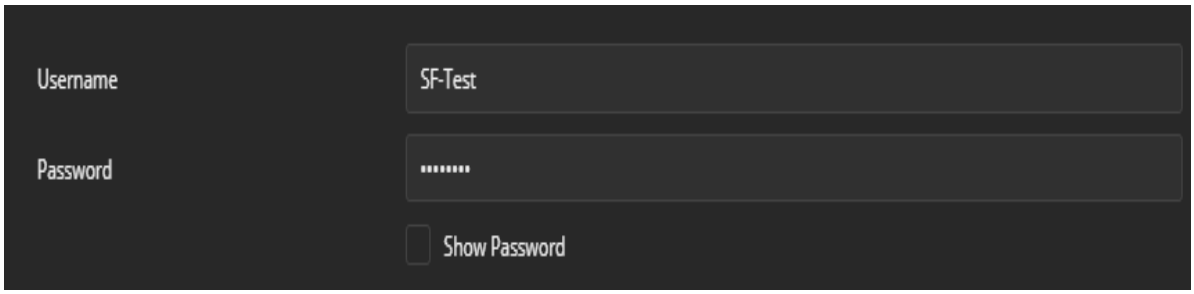
REST rajapinnat päädyttiin kehittämään Azure Functions -työkalulla, koska Functions rajapintojen kehityksen pääsee nopeasti aloittamaan hyvän dokumentaation avulla. Kehityskieleksi tuli C#, koska se oli kehitystyön nopeuden ja helppouden kannalta paras, ja virallinen dokumentaatio on laajin ja monipuolisin C# osuudelle.

ProductCatalog rajapinta oli tarkoitus olla käytännössä asiakkaan sovellusten lisäosaksi, jolla voitiin kutsua haluttujen tiedostojen dataa Azure Dalatakesta ohjelmien omaan käyttöön. TokenConfirmerin tehtäväksi jää lopulta varmistaa ProductCatalogilla tehtyjen kyselyiden lopullinen tila omalla kyselyllään ja parametrillään.

### 5.1.1 Basic Authentication ja kyselyparametrit

Rajapinnan vaatimuksiin kuului pakollinen autentikointi. Basic Authentication sopi projektin tarpeisiin hyvin, koska se oli yksinkertainen toteuttaa. Projektissa se vertasi käyttäjän syöttämät tunnukset http pyyntöihin tarkoitetussa ohjelmassa esim. Postmanissa username ja password, joka tarkoitti sitä, että webselaimella ei voitu kutsua osoitetta ollenkaan.

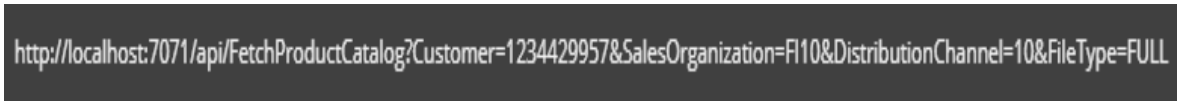
Postman clientillä on Authorization välilehti, johon voi basic authentikointi parametrit syöttää, kuten kuvassa 24.  Tämän jälkeen API sai käyttäjän ja salasanan http otsikon mukana käyttöönsä ja pystyi verrata niitä koodissa myöhemmin mainittavilla metodeilla.




Kuva 24. Postman Authorization välilehti

Aluksi käyttäjien nimet ja salasanat olivat syötettynä erilliseen Azure SQL tauluun, mutta tietoturvariskien takia otettiin käyttöön Azure KeyVault, joka on turvallisempi ja sitä on helpompi valvoa.

ProductCatalog rajapinnan koodissa oli tärkeää huomioida, että minkä asiakkaan dataa haetaan. Tässä rajapinta tarvitsi käyttää kyselyparametrejä, joita tarkastelemalla pystyy kutsumaan tietyn asiakkaan dataan syöttämällä oikeat parametrit API:n Url kutsuun. Tällöin Rajapinta vertaa käyttäjän syöttämiä parametrejä oman kontrollitaulunsa riveihin Customer, SalesOrganization ja DistributionChannel, ja palauttaa myöhemmin data syötteen, mikäli kaikki parametrit täsmäävät taulusta löydettyjen kanssa. Esimerkkinä ProductCatalog API:n kutsun osoitteesta, joka sisältää asiakaskohtaiset kyselyparametrit on esitetty kuvassa 25.




Kuva 25. ProductCatalog esimerkki osoite kyselyparametreillä

ProductCatalogilla rajapinnalla oli neljäntenä kysely parametrinä vielä FileType, joka voi olla Full tai Delta. Full tarkoittaa full-nimikkeellä olevan tiedoston sisällön palauttamista, kun taas Delta palauttaa Delta-nimikkeellä olevan tiedoston, joka on vain tietty osa FULL-tiedostosta. ProductCatalog-rajapinta ei ota kantaa mitä tai paljon esimerkiksi DELTA vastaus sisältää dataa, vaan antaa paluuvastauksessa Datalakesta Delta-nimekkeellä olevan tiedoston datat Full -tiedoston sijasta. Full- ja Delta tiedostoille on Datalakessa oma polkunsa, jota ProductCatalog palvelu osaa vaihtaa riippuen FileType kyselyparametristä.

Rajapintoihin tuli myöhemmin vielä yhteyden testausparametrit, joilla palvelun ylläpitäjä voi nopeasti katsoa, että kaikki rajapinnoissa olevat työkalut toimivat: SQL tietokanta, Datalake, KeyVault ja Azure Functions REST rajapinnan päämetodit ja yhteys.

Ideana on yksinkertaisesti syöttää testausparametrien arvot kohtiin Customer, SalesOrganization ja DistributionChannel sekä käyttää Basic Autentikoinnissa testaukseen tehtyjä tunnuksia. Tähän kyselyyn rajapinnat palauttavat statuskoodin 200 OK tai jonkin virheen, riippuen siitä, mikä palvelu ohjelmaketjussa on alhaalla tai ei jostain syystä toimi.

Mikäli jokin palvelu rajapinnoissa ei toimi tai muu virhe tapahtuu, niin paluuviestinä tulee palvelujen omia virheilmoituksia kuten SqlException, eli sql taulukko peräinen virhe, KeyVaultErrorException, eli KeyVault peräinen virhe tai AdlsException, eli Dataleke peräinen virhe. Myös tavanomaisemmat virheilmoitukset kuten AggregateException tai ArgumentNullException on huomioitu rajapintojen koodissa.

### 5.1.2 Azure KeyVault

Projektissa KeyVaultin pääsääntöinen käyttötarkoitus oli suojata käyttäjien käyttäjätunnuksia ja salasanoja, joita verrattiin koodissa C# KeyVault kirjaston KeyVaultClient avulla käyttäjän Authorization -kohtaan syöttämiin muuttujiin.

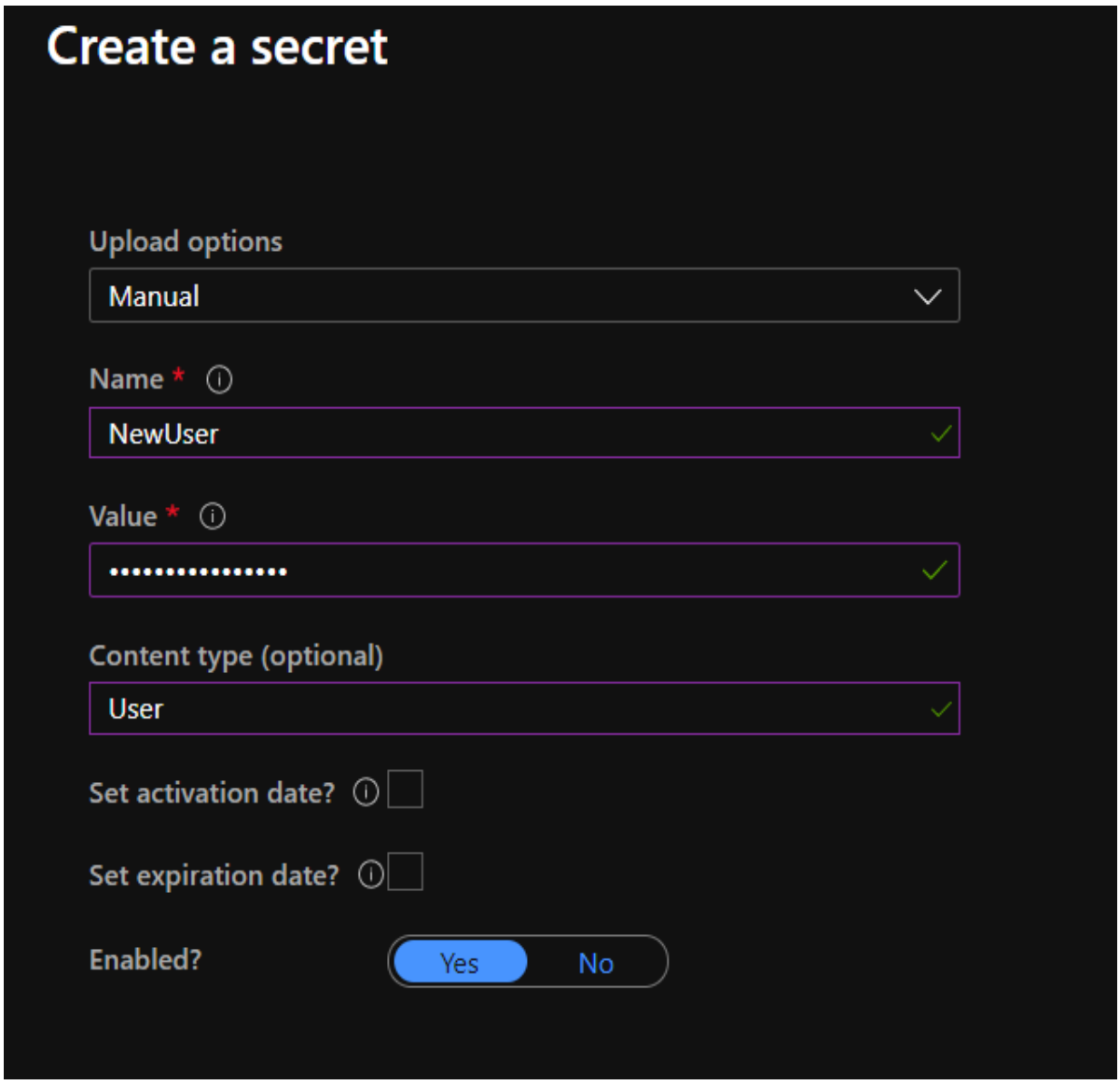
```
try
{
    AzureServiceTokenProvider azureServiceTokenProvider = new AzureServiceTokenProvider();
    var keyVaultClient = new KeyVaultClient(
        new KeyVaultClient.AuthenticationCallback(azureServiceTokenProvider.KeyVaultTokenCallback));

    var secret = Task.Run(async () => await keyVaultClient.GetSecretAsync(
        "[redacted]" + user)).Result;

    if (secret != null && secret.ContentType == "User")
        return secret.Value;
    else
        return "invalid token";
}
catch (KeyVaultErrorException kvex)
{
    return kvex.ToString();
}
catch (AggregateException aex) // korjaa run time errorin väärästä secret nimestä.
{
    return aex.ToString();
}
```

Kuva 26. Azure KeyVault luokkien ja metodien käyttö käyttäjän tunnistamiseen

Kuvan 26 metodissa verrataan käyttäjän syöttämää ”user” nimistä muuttujaa Azure Key-Vault kirjaston avulla KeyVaultissa oleviin varmistettuihin käyttäjiin. Mikäli käyttäjä löytyy, ylläoleva metodi palauttaa ”secret.Value” muuttujan arvon verratakseen sitä myöhemmin käyttäjän syöttämään salasana arvoon.

### 5.1.3 Azure KeyVault käyttäjien lisäys

KeuVaultiin uusien käyttäjien, avaimien tai muiden salaisuuksien lisääminen onnistuu Azure ympäristön kautta. Kuvassa 27 salaisuuden (secret) eli esimerkiksi käyttäjän lisääminen rajapinnan käyttöä varten on yhden kaavion täyttäminen Azure ympäristössä.



Kuva 27. KeyVault uuden salaisuuden luominen

Käyttäjän salaisuuden tiedoiksi riittävät nimi, sekä arvo, jota käytetään. Tässä tapauksessa salasanana ja valinnainen sisällön tyyppi, joka määrittelee salaisuuden nimen omaan käyttäjäksi (User) näissä rajapinnoissa. Tunnukselle voi myös laittaa alkamispäivämäärän (activation date) tai määrittää tunnuksen viimeisen käyttöpäivän (expiration date).

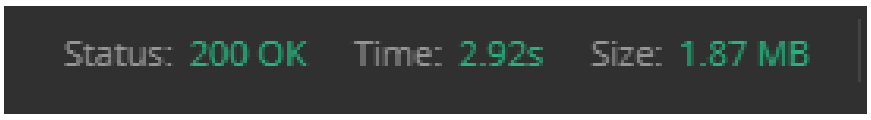
### 5.1.4 Azure SQL kontrollitaulun hallinta

Kontrollitaulussa on asiakkaiden datan parametrit, joita REST rajapinnat tarkastelee, kun kutsu niihin tehdään. Mikäli parametrejä ei ole laitettu tai ne eivät löydy kontrollitaulusta, palauttavat rajapinnat Unauthorized 401 virhettä. Kuvassa 28 on virheen tiedot.

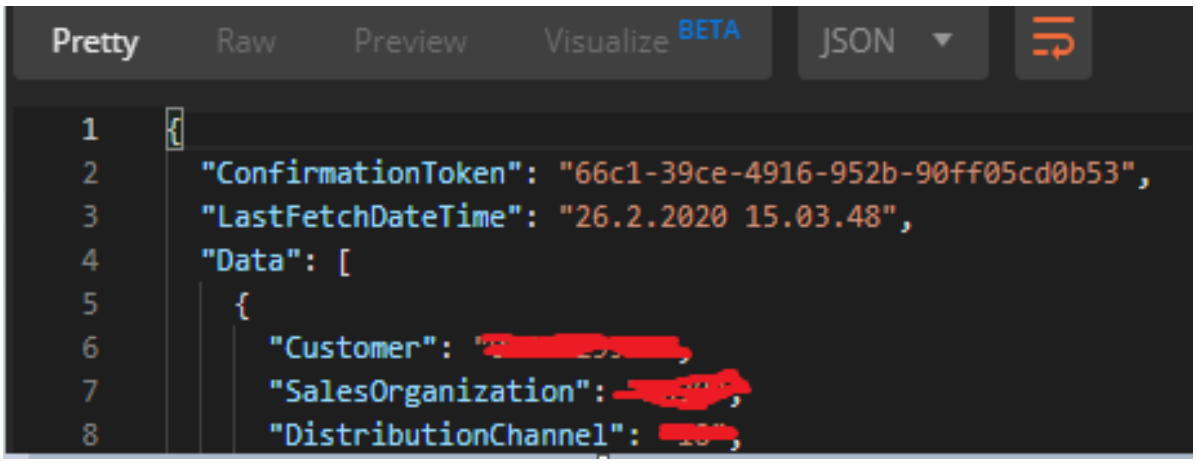

Kuva 28. Unauthorized 401 virhe esimerkki

Ensisijaisesti ProductCatalog rajapinta päivitti onnistuneen kyselyn statuksen ”Fetched” kontrollitaulun riville, jossa syötetyt parametrit vastaisivat käyttäjän syöttämiä kyselyparametrejä. ProductCatalogin paluuviestissä käyttäjä sai myös satunnaisesti luodun varmennustoken arvon (ConfirmationToken), jolla hän pystyi kutsua varmennus rajapintaa TokenConfirmer kyselläkseen lopullisen kyselyn ”Fetched” -statuksella merkitylle datalle.

Esimerkkinä kuvassa 30 on ProductCatalogin rajapinnan onnistuneen kutsun paluuviestistä, joka sisältää asiakkaan datan, jossa on pakollisten parametrien lisäksi myös tuotetietoja, varmennustoken ja viimeksi tehdyn kyselyn aika. Status onnistuneessa kutsussa on 200 OK, vastaus aika ja data sisällön koko, kuten kuvassa 29.



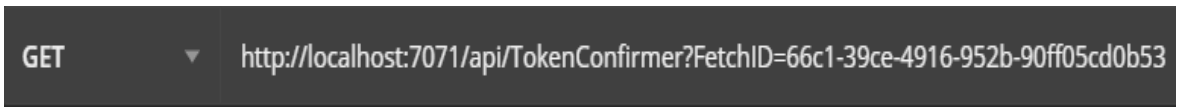

Kuva 29. Onnistunut kutsu status OK esimerkki



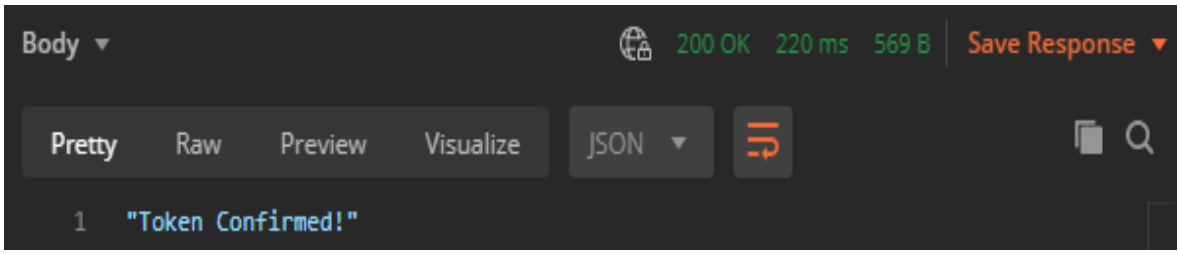

Kuva 30. ProductCatalog rajapinnan onnistunut paluuviesti esimerkki

ProductCatalog päivittää myös Kuvassa 29 olevan ConfirmationToken arvon kysytyn asiakkaan riville kontrollitauluun. Sieltä TokenConfirmer API myöhemmin tarkastaa täsmääkö se jälleen käyttäjän syöttämään kyselyparametriin ”FetchID”.

Esimerkki TokenConfirmer kutsusta ProductCatalog API:n aiemmasta vastauksesta saadulla varmennustoken arvolla (ConfirmationToken) on esitetty kuvassa 31. TokenConfirmer API:n onnistunut status 200 OK vastaus antaa lyhyen confirmed -viestin, kuten kuvassa 32.




Kuva 31. TokenConfirmer rajapinnan GET pyyntö esimerkki FetchID parametrillä



Kuva 32. TokenConfirmer onnistuneen pyynnön paluuviesti

TokenConfirmer päivittää kontrollitauluun haku statuksen tämän jälkeen ”Completed” mikäli käyttäjän syöttämä FetchId, eli token arvo (ConfirmationToken) vastaa kontrollitaulussa ProductCatalogin token arvo kenttään lisäämää arvoa. Tämä tarkoittaa asiakkaan omassa ohjelmistossa sitä, että hakuketju on suoritettu loppuun ja haun tehneellä taholla on tiedossaan kyseisen valmistuneen ajon data ja haun kellonaika.

Rajapintojen piti siis myös päivittää tiettyjä sql taulun rivejä kuten hakupäivä, joka on päivämäärä, jolloin rivin data on haettu. Haun tila, eli status on aluksi tyhjä kenttä, mutta ProductCatalog API:n onnistuneen pyynnön jälkeen tilan status muuttuu ”Fetched” ja lopulta onnistuneen TokenConfirmer pyynnön jälkeen se muuttuu lopulliseen statukseen ”Completed”. ProductCatalog API päivittää myös haetun tiedoston nimen ja varmennustokenin samalla, kun se muuttaa statuksen tilaan ”Fetched”.

### 5.1.5 Azure Datalake

Datalakea hyödynnettiin eritavoilla riippuen käyttäjän syöttämästä FileType parametristä, ProductCatalog haki tiedostoja joko Full- tai Delta outputs -kansiosta ja näytti niiden sisällön. Delta tyyppi oli ensisijainen tapa hakea dataa, mutta joskus saattoi olla tarpeen hakea koko tiedosto. Tämän takia full-haku tehtiin mahdolliseksi ProductCatalog kutsussa.

Datan hakeminen rajapinnoissa toimii niin, että Azure Datalake toimi pääsääntöisenä Output tiedostojen varastona, jonka tiedostoista REST rajapinta kyselee tiedoston sisällön tekstimuodossa käyttöönsä ja palauttaa lopulta käyttäjälle.

Onnistuneessa kutsussa ProductCatalog rajapinnan piti hakea omalla Datalake kirjastollaan datalakesta tietyn kansion sisältä tiedoston sisältö tekstimuodossa käyttöönsä. Tiedostosta tulevaa dataa luetaan .NET Streamreader luokalla, jolla voi lukea tiedostojen sisältöä eri metodeilla. Tiedoston sisältö luettiin C# Streamreaderin ReadLine metodilla eli yksi rivi kerrallaan tai ReadToEnd metodilla, joka kävi läpi kaikki rivit tiedostosta.

Tiedostojen sisältö oli myös nopeaa ladata AdlsClient, Datalaken omalla .NET-kirjastolla sovelluksen käytettäväksi. Kuvassa 33 olevassa esimerkki metodissa haetaan Datalaken output kansiosta tiedoston nimen perusteella kyseisen tiedoston sisältö ja luetaan se Streamreader -kirjaston metodilla ja palautetaan koko sisältö merkkijono formaatissa käytettäväksi myöhemmin.

```
// Create ADLS client object
AdlsClient client = AdlsClient.CreateClient(adlsAccountFQDN, clientCreds);
try
{
    string folder = Environment.GetEnvironmentVariable("OutputFolder");
    string fileName = "/" + folder + "//" + queryFileName; // FILE path in Azure Datalake

    if (queryFileName.Equals("testRun"))
        return "Test run successful!";
    else if (client.CheckExists(fileName))
    {
        //Read file contents
        using (var readStream = new System.IO.StreamReader(client.GetReadStream(fileName)))
        {
            return readStream.ReadToEnd();
        }
    }
    else
        return "File " + queryFileName + " not found!";
}
catch (AdlsException e)
{
    return "AdlsException: " + e.ToString();
}
catch (ArgumentNullException ane)
{
    return "ArgumentNullException: " + ane.ToString();
}
```

Kuva 33. Esimerkki tiedoston sisältö hakumetodista datalake kirjastolla

Tämän jälkeen tiedoston sisältö piti muuttaa JSON muotoon ja näyttää käyttäjän käyttämässä clientissä (esim. Postman tai jokin muu http-pyyntö sovellus).

Datatyyppien muuttaminen tehtiin ProductCatalog rajapintaan omalla koodilla, jotta muuttujien muokkaus ja datan oikeamuotoinen palautus onnistuisi. C# kirjasto Newtonsoft otettiin käyttöön datamuuttujien muokkaamiseen merkkijono muodosta JSON muotoon, jotta niiden palautus onnistuisi.

Projektissa pääsääntöisesti tarkoitus oli muuttaa tavallisen Datalakessa olevan teksti tiedoston sisältö C# merkkijono muuttujan kautta JSON muotoon, jolloin sovellukset saivat tiedoston sisällön käyttöönsä halutussa formaatissa. Kuvassa 34 esimerkki projektissa käytetystä datan muuntamismetodista Newtonsoft kirjastoa käyttäen.

```
1 reference | 0 changes | 0 authors, 0 changes
internal string BuildOutput(string token, string lastFetchDateTime, string data)
{
    // Replace "\r\n" in the Data string with "," for json Deserializing
    data = data.Replace("\r\n", ",");

    // Deserilialize string from the StreamReader to format <List<object>>
    var deString = JsonConvert.DeserializeObject<List<object>>($"[{data}]");

    // Make a new object with token, date and deserialized data string!
    var finalObj = new JsonOutput() { ConfirmationToken = token, LastFetchDateTime = lastFetchDateTime, Data = deString };

    if (finalObj.Data.Count == 0)
    {
        return "No Files found!";
    }
    // Serialize finished object back to json string format:
    var json = JsonConvert.SerializeObject(finalObj, Formatting.Indented);
    return json;
}
```

Kuva 34. ProductCatalog API:n metodi, jolla muunnetaan data JSON muotoon

Kuvassa 34 on BuildOutput -metodi, joka vastaanottaa merkkijono muuttujat: ”token”, ”lastFetchDateTime” ja ”data”. Data on merkkijonomuuttuja, joka on streamReader -metodilla saatu data sisältö Datalake kansiosta haetusta tiedostosta. Mikäli Datan sisällön pituus on 0, niin oliota ei muokata tai palauteta, vaan viesti ”No Files Found” laitetaan käsittelyyn ja siitä palautetaan asiaankuuluva paluuviesti myöhemmin. Data voi olla tyhjä, jos tiedostossa ei ole ollenkaan rivejä tai ne ovat tyhjät tekstistä. Esimerkiksi tyhjä datalista, eli ”Data [ ]”.

Data -muuttuja piti nyt muokata, eli vaihtaa kaikki ”\r\n” kohdat merkkijonossa pilkuiksi (”,”), jotta JSON muotoon muuttaminen onnistuisi. Tämän jälkeen data muutetaan oliolistaksi (object list) muuttujaan ”deString”, josta rakennetaan lopullinen C# oliorakenne, finalObj, josta pitää käydä ilmi ConfirmationToken, eli varmistus token, lastFetchDateTime, eli viimeiseksi tehty kysely ja Data, joka on datan sisältö itsessään. Näillä muokkauksilla data vastaa tarvittavaa loppukäyttäjälle näytettävää JSON paluuviestiä. Seuraavaksi uusi oliorakenne ”finalObj” serialisoidaan takaisin JSON formaattiin merkkijono muuttujaan ”json” ja palautetaan main metodille.

Kuvassa 35 on koodi esimerkki ProductCatalogin StringContent http paluuviestin palautuksesta käyttäjälle main metodista. Kuvassa on ”ret” merkkijono muuttuja, joka on aiemmin muokattu BuildOutput metodilla JSON merkkijonoksi. Nyt StringContent kuvan parametreillä palauttaa sen oikeassa JSON formaatissa vihdoinkin käyttäjälle http paluuviestissä.

```
else if (fileType != null && fileType == "JSON" || fileType == "json")
{
    return new HttpResponseMessage(HttpStatusCode.OK)
    {
        Content = new StringContent(ret, Encoding.UTF8, "application/json")
    };
}
```

Kuva 35. JSON palautus StringContent luokan mukaisesti

### 5.1.6 Erikoisrajapinta ODataReader

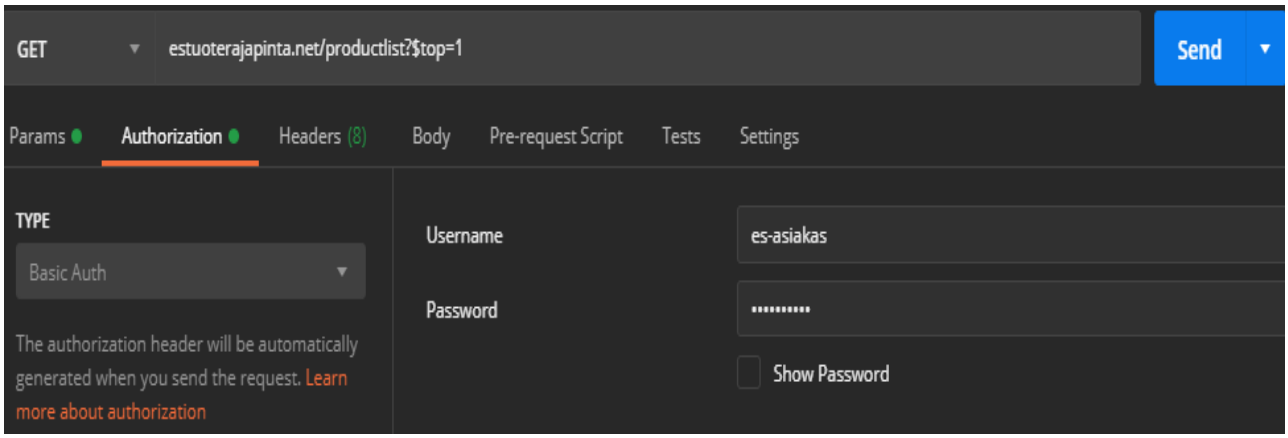
Asiakas tarvitsi myös OData (Open Data Protocol) rajapinnan, jolla haetaan tiettyjen taulujen dataa ja se palautetaan XML formaatissa. Tämä rajapinta julkaistiin Azure AppServiceen ja se tehtiin .NET Entity Framework:llä.

OData rajapinnan tarkoitus on olla alusta, josta asiakas voi nopeasti tietyillä komennoilla hakea dataa monista eri tauluista yhtäaikaisesti. Entity Framework työkalu sopi tähän tarkoitukseen.

Asiakkaan tauluista luotiin Entity Frameworkillä oma edmx -tiedosto ja tehtiin tarvittavat relaatiot taulujen välille. Relaatio tarkoittaa taulujen välistä yhteyttä, jonka avulla voidaan esimerkiksi palauttaa kokonaisien kokoelmien palautus tietokanta sarakkeiden sijaan. Tämän jälkeen palvelu julkaistiin Microsoft Azuressa AppService työkalulla ja se toimii http GET pyynnöllä, aivan kuten aikaisemmatkin rajapinnat.

ODataReader rajapintaa käytetään hieman eri tavalla, kuin aiempia rajapintoja, koska sillä on tarkoitus tehdä tarkempia hakuja tietokantoihin ja entiteettikokoelmiin käyttäen apuna Entity Framework ominaisuuksia. Esimerkkinä kuvassa 36 on ODataReader rajapinnalle tehty rajattu GET pyyntö, joka palauttaa tietyn taulun ensimmäisen rivin XML formaatissa.

Kuten kuvan 36 pyynnöstä huomataan, niin myös ODataReader rajapinta käyttää Basic Authentication -kirjautumista ennen kuin palauttaa mitään dataa http paluuviestissä.




Kuva 36. ODataReader GET pyyntö

# 6 REST RAJAPINTOJEN OMINAISUUDET JA ARKKITEHTUURI

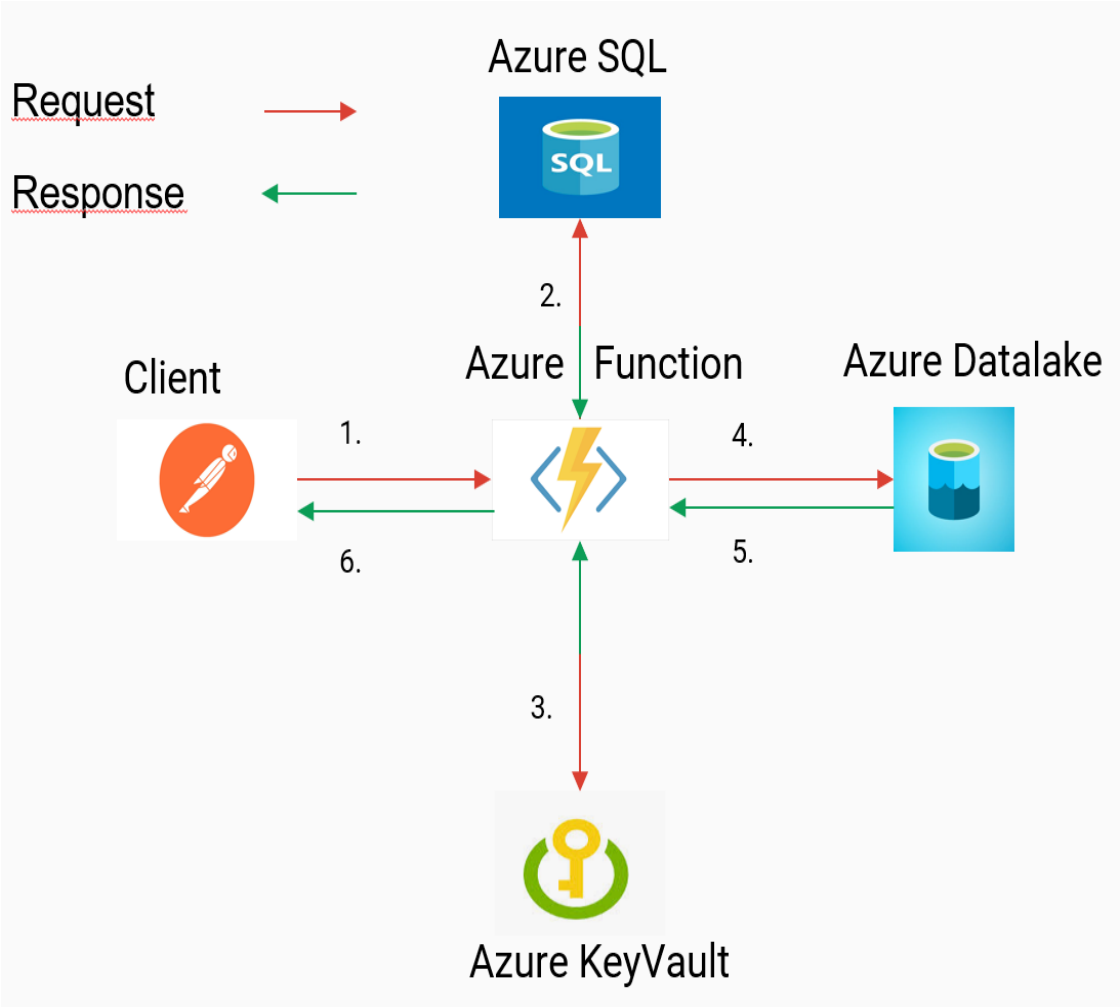
## 6.1 ProductCatalog API yhteenveto

ProductCatalog ominaisuuksiin kuului SQL kontrollitaulun rivien lukeminen ja niiden muokkaus rajapinnan http pyynnön osoitteeseen lisättävien kyselylauseiden perusteella. Tietoturvasyistä käyttäjien tunnistus tapahtuu Basic Authentication otsikkoa vertaamalla Azure KeyVaultissa sijaitseviin varmennettuihin käyttäjiin.

Palvelun päätehtävänä oli tiedostosisällön vastaanottaminen Azure Datalaken Outputs -kansion tiedostoista SQL Kontrollitaulusta saadun asiakkaan mukaan. Eli tarkistetaan paikkaansa pitävät taulun kentät ”Customer”, ”SalesOrganization” ja ”DistributionChannel” SQL kontrollitaulusta. Tiedoston sisällön vastaanottamisen jälkeen kriittiset toimenpiteet ovat seuraavat:

- Datalakesta saadun datan lukeminen Streamreader luokalla merkkijonosta ja siitä muokkaaminen Newtonsoft kirjastolla haluttuun JSON formaattiin.
- Lopuksi datan palauttaminen varmennetulle käyttäjälle http paluuviestissä.

ProductCatalogin arkkitehtuuri on esitetty kuvassa 37.




Kuva 37. ProductCatalog API arkkitehtuuri

## 6.2 ProductCatalog toiminta

Arkkitehtuuri kuvassa 37 on esitetty ProductCatalog REST rajapinnan toimintaa pyyntö- (punainen) ja vastaus (vihreä) nuolilla. Kohdassa 1 aloitetaan Http GET pyynnön lähetyksellä clientiltä ProductCatalog Azure funktion osoitteelle, jossa on pakolliset kyselylauseet lisättynä, eli Customer, SalesOrganization, DistributionChannel ja FileType.

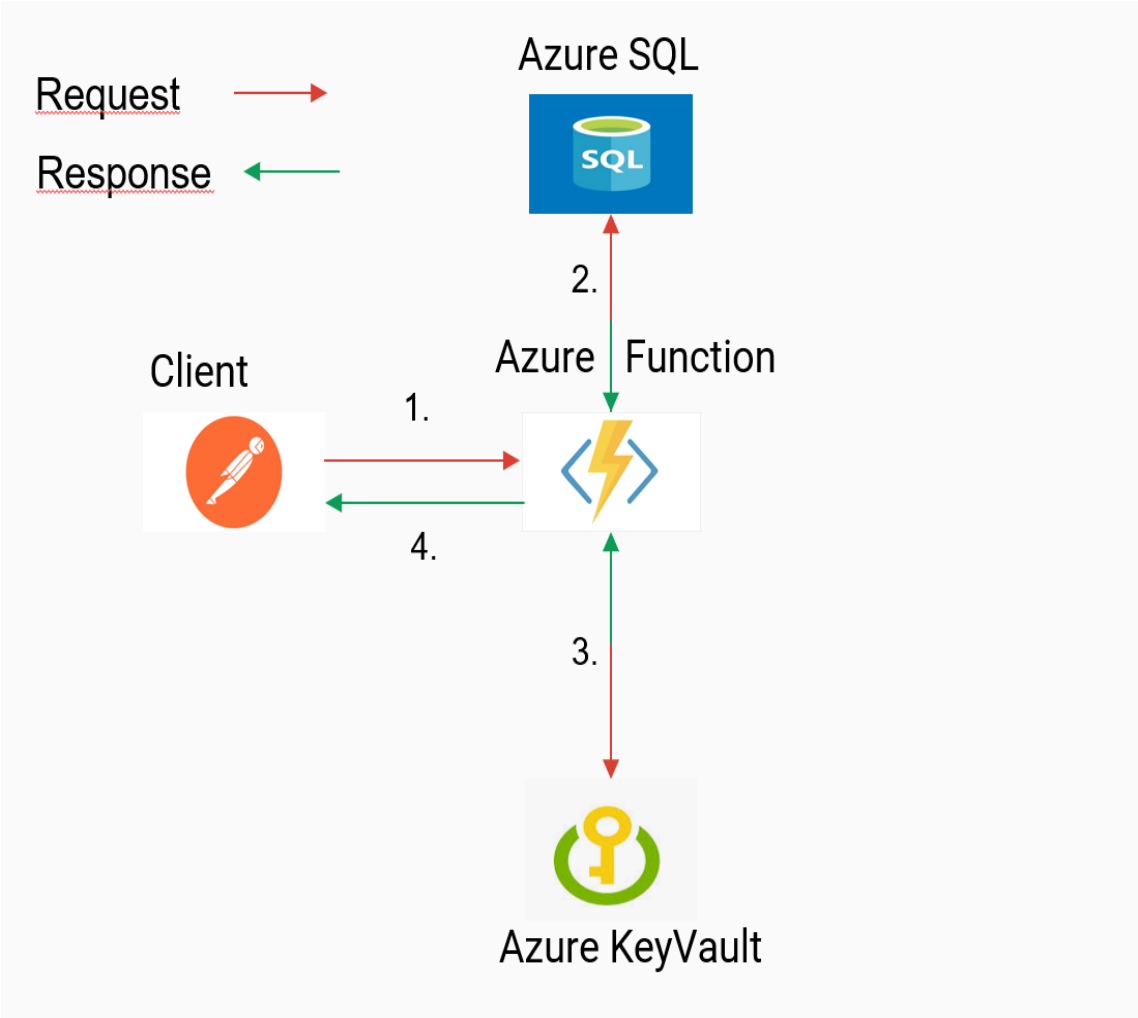
Kohdassa 2 Azure funktio API saa GET pyynnön, jonka jälkeen se kyselee asiakastiedot SQL-kontrollitaulusta käyttäen palvelun osoitteeseen lisätyistä kyselylauseista saadakseen tietyn asiakkaan tiedoston nimen käytettäväksi myöhemmin. Kohdassa 3 Azure funktio tarkistaa GET pyynnön lähettäjän katsomalla clientin authorization välilehteen syötettyä username ja password yhdistelmää, josta vertaa näitä arvoja KeyVaultissa oleviin varmistettuihin käyttäjiin, jonka hyväksymisen jälkeen jatkaa prosessia.

Kohdassa 4 Azure funktio käyttää Azure Datalaken oman kirjaston metodia, jolla se ottaa yhteyttä Outputs kansioon etsien kohdassa kaksi saadulla tiedoston nimellä nyt sitä vastaavaa tiedostoa kansiosta. Kohdassa 5 tiedoston löydettyään funktio saa kyseisen tiedoston koko sisällön datalakesta käyttöönsä Streamreader -metodin paluu (return) muuttujaan. Kohdassa 6 näkyy paluuviesti, joka tarpeellisten merkkijonosta Json-muotoon tehtyjen muokkausten jälkeen palauttaa Delta- tai Full tiedoston sisällön käyttäjälle http paluuviestin rungossa.

## 6.3 TokenConfirmer API yhteenveto

TokenConfirmer ominaisuuksiin kuului SQL kontrollitaulun rivien lukeminen ja niiden muokkaus rajapinnan http pyynnön osoitteeseen lisättävän kyselylauseen ”FetchID” perusteella statuksesta ”Fetched” statukseen ”Completed”. Myös TokenConfirmer API:ssa on sama tietoturva ratkaisu kuin ProductCatalogissa, eli käyttäjien tunnistaminen Basic Authentication otsikkoa vertaamalla Azure KeyVaultissa oleviin käyttäjiin.

Päätehtävänä oli kyselyn statuksen varmistaminen käyttäjälle status 200 OK: ”Token Confirmed!” tai hylkääminen status virheellä 400 BadRequest ja viestillä: ”Missing or incorrect Confirmation token!”. TokenConfirmer API:n arkkitehtuurikuva on esitetty kuvassa 38.




Kuva 38. TokenConfirmer API arkkitehtuuri

## 6.4 TokenConfirmer toiminta

Kuvan 38 kohdassa 1 on Http GET pyynnön lähetys clientiltä TokenConfirmer API Azure funktion palveluosoitteelle, jossa on pakollinen kyselylause FetchId lisättynä. Kohdassa 2 Azure funktio API saa GET pyynnön, jonka jälkeen se kyselee asiakastiedot SQL-kontrollitaulukosta käyttäen palvelun osoitteeseen lisätystä FetchId kyselylauseesta varmistaakseen aikaisemman ProductCatalog kyselyn statuksen.

Kohdassa 3 Azure funktio tarkistaa GET pyynnön lähettäjän katsomalla clientin authorization välilehteen syötettyä username:password yhdistelmää, josta vertaa näitä arvoja KeyVaultissa oleviin varmistettuihin käyttäjiin, jonka hyväksymisen jälkeen jatkaa prosessia. Lopulta kohdassa 4 funktio palauttaa paluuviestin ”Token Confirmed!” ja http statuksen 200 OK käyttäjälle http viestin rungossa, mikäli käyttäjän syöttämä FetchId parametri vastasi kontrollitaulusta haettua asiakkaan riviä.

# 7 YHTEENVETO

Työn tavoite oli luoda REST palvelu, joka täyttäisi asiakkaan tuotekatalogidatan nopean muokkaamisen ja hakemisen tarpeet. Palvelu valmistui aikataulussa, vaikka joitakin ongelmia tulikin matkan varrella. Sql kontrollitaulun rivien muokkaukset ja erinäisten oikeuksien puuttuminen asiakkaan Azure Portal ympäristöön aiheuttivat pieniä ongelmia, mutta ei mitään ylitsepääsemätöntä.

Dataintegraation tekeminen tekstitiedostojen sisällön muuttamisesta JSON muotoon vaati omat ratkaisunsa erillisen .NET kirjaston ja Azure Functionin omien palautus metodien välillä ja loi haasteita JSON formaatin luomisessa ja sen oikein näyttämisessä ja palautuksessa käyttäjälle.

Nykyisin ProductCatalog- ja TokenConfirmer API ovat olleet asiakkaalla tuotantokäytössä jo yli vuoden. Versiota 2.0 ei ole tiedossa ProductCatalog- ja TokenConfirmer rajapinnoista, mutta erääseen vanhempaan REST rajapintaan on tehty ProductCatalogin tapainen merkkijonosta JSON formaattiin muokkaamismetodi. Lisäksi asiakkaan tarve uusille REST rajapinnoille on kasvanut, ja uusia projekteja on tarkoitus aloittaa vielä syksyllä 2020.

Mikäli ProductCatalog ja TokenConfirmer -rajapintojen kehitys aloitettaisiin nyt uudestaan, niin Newtonsoft -kirjaston laajempi opiskelu olisi hyödyksi kehityksen muokkausvaiheessa, jossa ilmeni lopuksi ongelmia tiedostojen ollessa väärässä formaatissa Datalakessa. Lisäksi parempi SQL Transact -kielen tuntemukseen voisi mahdollisesti panostaa aluksi enemmän, jotta kontrollitaulun muokkaus ja lukeminen olisi sujuvampaa .NET ympäristössä.

Azureen REST projekteja tehdessä oppi käyttämään monipuolisesti Azure Portalin työkaluja ja sovelluksia. Lisäksi C# ohjelmistokehitys ja .NET kirjastot tulivat tutummiksi. Lisäksi projektin luomisessa oli osa työskentelystä asiakkaan kanssa keskustelua siitä, millaisiin tarkoituksiin REST rajapinnat tulisivat olemaan ja nekin muuttuivat projektin edetessä ja asiakkaan tarpeiden kasvaessa. Kokonaisvaltaisesti Azuren työkaluilla ja varsinkin Azure Funktiolla on mukava luoda nopeasti REST rajapintoja ja varmasti monia muitakin apuratkaisuja ohjelmistokehityksessä. Oli myös mielenkiintoista todeta, miten hyvin kaikki Azure palvelut toimivat keskenään, lisäksi niiden kehitys- ja käyttäjäkunta on niin laajaa, että internetistä löytyy lähes varmasti ratkaisu tai apuja mahdollisiin pulmatilanteisiin.

# LÄHTEET

Dejan Stojanovic 2018. Reduce traffic by serializing JSON with different alias with Json.NET and C#, (Dejan Stojanovic, 2018). [viitattu 28.8.2020] Saatavissa:

https://dejanstojanovic.net/aspnet/2018/march/reduce-traffic-by-serializing-json-with-different-alias-with-jsonnet-and-c/

Digia 2016. API:t ovat modernin integraatiostrategian ydin, (Tero Kivisaari, 2016).

[viitattu 28.8.2020]. Saatavissa: https://blog.digia.com/rest-api

Enfo Oyj 2020. Palvelumme. [viitattu 15.10.2020]. Saatavissa:

https://www.enfo.fi/palvelumme

Itewiki 2020. Microsoft Azure. [viitattu 23.10.2020]. Saatavissa:

https://www.itewiki.fi/opas/microsoft-azure/

Medium Corporation 2020b. HTTP Server: Everything you need to know to Build a simple HTTP server from scratch, (“Skrew Everything”, 2018). [viitattu 28.8.2020]. Saatavissa:

https://medium.com/from-the-scratch/http-server-what-do-you-need-to-know-to-build-a-simple-http-server-from-scratch-d1ef8945e4fa

Medium Corporation 2020a. Nginx WebServer with basic HTTP Protocol, (Sanjeev Gautam, 2018). [viitattu 28.8.2020]. Saatavissa:

https://medium.com/@me.sanjeev3d/https-medium-com-me-sanjeev3d-nginx-webserver-215b636afcc5

Microsoft 2020a. What is Entity Framework? [viitattu 23.10.2020]. Saatavissa:

https://www.entityframeworktutorial.net/what-is-entityframework.aspx

Microsoft 2020b. Azure Data Lake Storage Gen1 documentation. [viitattu 13.7.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/data-lake-store/

Microsoft 2020c. About Azure Key Vault. [viitattu 13.7.2020]. Saatavissa:

https://docs.microsoft.com/en-us/azure/key-vault/general/overview

Microsoft 2020d. Create your first Azure Function. [viitattu 13.7.2020]. Saatavissa:

https://tutorials.visualstudio.com/first-azure-function/create-function

Microsoft 2020e. SqlConnection Class. [viitattu 13.7.2020]. Saatavissa:

https://docs.microsoft.com/en-us/dotnet/api/system.data.sqlclient.sqlconnection?view=dot-net-plat-ext-3.1

Microsoft 2020f. App Service Overview. [viitattu 13.7.2020]. Saatavissa:

https://docs.microsoft.com/en-us/azure/app-service/overview

REST API Tutorial 2020. Richardson Maturity Model. [viitattu 13.7.2020]. Saatavissa:

https://restfulapi.net/richardson-maturity-model/

Sininen Meteoriitti 2017. Azure functions ja API Apps integraation avuksi, (Mika Berglund, 2017). [viitattu 13.7.2020]. Saatavissa: https://meteoriitti.com/2017/09/13/azure-functions-api-apps/

SmartBear Software 2020. Basic Authentication. [viitattu 13.7.2020]. Saatavissa:

https://swagger.io/docs/specification/2-0/authentication/basic-authentication/

Studytonight 2020. HATEOAS - Important Concept For REST API. [viitattu 28.8.2020].

Saatavissa: https://www.studytonight.com/rest-web-service/hateoas

Web-palvelinohjelmointi Java 2019a. Osa 1, Internetin perusosat. [viitattu 28.8.2020]. Saatavissa: https://web-palvelinohjelmointi-19.mooc.fi/osa-1/1-internetin-perusosat

Web-palvelinohjelmointi Java 2019b. Osa 6, Rajapinnat ja REST. [viitattu 28.8.2020].

Saatavissa: https://web-palvelinohjelmointi-19.mooc.fi/osa-6

Webperformance 2020. How HTTP Authentication works and why load testers should care, (Chris Merril, 2011). [viitattu 13.7.2020]. Saatavissa:

https://www.webperformance.com/load-testing-tools/blog/2011/06/how-http-authentication-works-and-why-load-testers-should-care/

Wikipedia 2020a. Enfo Oyj. [viitattu 15.10.2020]. Saatavissa:

https://fi.wikipedia.org/wiki/Enfo

Wikipedia 2020b. Luettelo http status koodeista. [viitattu 15.10.2020]. Saatavissa:

https://fi.wikipedia.org/wiki/Luettelo_HTTP-tilakoodeista