Tampereen ammattikorkeakoulu

Web API:n implementointi

Vladislav Melnitsuk

OPINNÄYTETYÖ
Toukokuu 2021

Tieto- ja viestintätekniikka,
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikan tutkinto-ohjelma
Ohjelmistotekniikka

MELNITSUK, VLADISLAV:
Azure Web API:n implementointi

Opinnäytetyö 21 sivua,
Toukokuu 2021

Opinnäytetyön tavoitteena oli muuttaa ruokalistojen ja ravintoloiden ylläpitäminen CMS-järjestelmästä API pohjaiseksi rajapinnaksi. Työssä tarkastellaan eri toimitsijoiden tarjoamia pilvipalveluita, valitun pilvipalvelun kanssa yhteensopivia tietokanta mahdollisuuksia ja työhön soveltuvia API teknologioita.

Työn tarkoituksena oli rakentaa helposti skaalautuva rajapinta, jonka monitoroiminen ja ylläpitäminen on rahallisesti mahdollisimman edullista. Vaatimuksien perusteella päätettiin hyödyntää markkinoilla olevia pilvipalveluita, sillä ne ratkaisivat skaalautuvuuden ja monitoroimisen ongelmat pientä kuukausikorvausta vastaan.

Rajapinta päädyttiin suorittamaan puhtaasti Microsoftin tarjoamilla palveluilla ja teknologioilla. Rajapinta koostuu kahdesta erillisestä osasta, jotka ovat pilvipalvelu ja palvelussa suoritettava API järjestelmä. Pilvipalveluksi valittiin Microsoftin Azure. Azuren tarjoama API management service, siihen helposti liitettävä Cosmos DB tietokanta ja Microsoftin mittava dokumentaatio vastaavanlaisesta API järjestelmästä lopulta päihittivät kilpailijat Google Cloudin ja Amazon Web Services. Azuressa suoritettava Web API ohjelmoitiin ASP.NET C# ohjelmointikielellä.

Avainsanat: Web API, Microsoft Azure, C#

ABSTRACT

Tampere University of Applied Sciences
Degree in ICT Engineering
Software Engineering

MELNITSUK, VLADISLAV:
Implementing Web API with Azure

Bachelor's thesis 21 pages,
May 2021

The object of this bachelor's thesis was to transfer previously used CMS based menu management system into a separate API service. This thesis goes through how the used cloud service and API program within were selected and implemented.

Implemented API service had to be easily scalable in terms of traffic and future APIs as well as had on option to be easily and extensively monitored. Allocated timeframe and requirements narrowed used technologies to a pre implemented cloud service with sizeable features. Cloud services which were considered included Amazon Web Services, Google Cloud and Microsoft Azure.

Azure came up on top because of its immense documentation regarding similar projects, integrated support for NoSQL database Cosmos DB and API management service which solved scalability and monitoring requirements. API logic program running within Azure was programmed with ASP .NET Web API framework and C# language using REST API architecture.

Key words: API, Microsoft Azure, C#

SISÄLLYS

1 JOHDANTO 6
2 TEORIA 7
2.1 Pilvipalvelut 7
2.1.1 Mikä on pilvipalvelu? 7
2.1.2 Pilvipalveluiden tyypit 7
2.1.3 Hyödyt ja haitat 8
2.2 API 8
2.2.1 Mikä on API? 8
2.2.2 API tyypit 9
3 TEKNOLOGIOIDEN VALINTA 10
3.1 Pilvipalvelu 10
3.2 Tietokanta 11
3.3 API 11
3.4 Versionhallinta 12
4 IMPLEMENTOINTI 13
4.1 Tietomallin suunnittelu 13
4.2 ASP.NET Web API 14
4.2.1 API:n tekeminen 14
4.2.2 Autentikointi 15
4.3 Azure App Service 16
4.3.1 Azure API management 16
4.4 Versionhallinta 18
5 LOPPUPOHDINTA 20
LÄHTEET 21

ERITYISSANASTO

IaaS	pilvipalvelun muoto, Infrastructure as a Service
Saas	pilvipalvelun muoto, Software as a Service
PaaS	pilvipalvelun muoto, Platform as a Service
API	Ohjelmointirajapinta, Application Programming Interface
REST	arkkitehtuurimalli, Representational State Transfer
RPC	protokolla, Remote Procedure Call
XML	merkintäkieli, Extensible Markup Language
JSON	tiedostomuoto, JavaScript Object Notation
CPU	tietokoneen suoritin, Central Processing Unit
RAM	tietokoneen hajasaantimuisti, Random access memory
SQL	standardoitu kyselykieli, Structured Query Language
CMS	sisällönhallinta järjestelmä, Content Management System
HTTP	hypertekstin siirtoprotokolla, Hypertext Transfer Protocol
AD	Microsoft Windows käyttäjätietokanta, Active Directory

1 JOHDANTO

Opinnäytetyö käsittelee ruokalistojen API-palvelun suunnittelua, erilaisten teknologioiden pohtimista ja valittujen teknologioiden implementointia. Työtä edeltävä palvelunmuoto todettiin olevan liian vaivalloinen ylläpitämisen ja järjestelmän jaettavuuden kannalta. Toteutetun järjestelmän toivottiin olevan täysin erillään entisestä toteutuksesta, hyödyntävän skaalautuvaa pilvipalvelua ja toteutuksen ylläpitäminen mahdollisimman edullista.

Opinnäytetyö käsittelee hieman pilvipalveluiden ja API teoriaa. Teoria osuudessa avataan lukijalle palvipalvelun määritelmä, kolmea eri palveluiden tyyppiä ja mitkä ovat niiden tuomat hyödyt ja haitat. APIt osuus avaa hieman, mikä on API ja mitkä ovat eri vaihtoehdot API:n luomiseen arkkitehtuuriselta kannalta.

Pohjustuksen jälkeen tutkitaan työn suorittamista teknologioiden valinnasta implementointiin. Teknologioiden valinnassa pohditaan käytettyä pilvipalvelua ja siihen sopivia tietokantaratkaisuja. Implementoinnissa dokumentoidaan työprosessi ja aukaistaan lukijalle hieman, miten työn eri teknologiat keskustelevat ja ovat liitoksissa toisiinsa sekä kuinka ne on tehty.

Loppupohdinta tarkastelee vaadittujen kriteerien täyttämistä ja kuinka työ on suoriutunut tähän mennessä menujen hallinnoimisesta. Pohdinnassa otetaan myös kantaa työn mahdollisiin parannuksiin, mikäli työ tehtäisiin toiseen kertaan.

2 TEORIA

2.1 Pilvipalvelut

2.1.1 Mikä on pilvipalvelu?

Pilvipalvelulla tarkoitetaan, että käyttäjän laitteella suoritettavan ohjelman tai järjestelmän sijaan, toiminto suoritetaan internetin välityksellä muualla. Suoritettava toiminto yleensä tarjotaan käyttäjälle palveluna, jonka tarkoituksena on helpottaa, nopeuttaa tai mahdollistaa toimintoja, jotka olivat ennen liian hankalia. Palvelusta useimmiten veloitetaan joko kuukausittaista tai käytetyn prosessointi kapasiteetin mukaista hintaa.

Pilvipalvelu yksinkertaisimmillaan voi olla kokoelma henkilökohtaisia tietokoneita, mutta useimmiten palvelut suoritetaan suurissa konesaleissa. Hyviä esimerkkejä tunnetuista palveluista ovat sähköposti- ja pankkijärjestelmät.

2.1.2 Pilvipalveluiden tyypit

Pilvipalvelut voidaan jakaa kolmeen eri tyyppiin niiden tarjottavan palvelun mukaan: IaaS (Infrastructure as a Service), SaaS (Software as a Service) ja PaaS (Platform as a Service). IaaS:n tapauksessa käyttäjä ostaa pääsyn jonkinlaiselle prosessointi yksikölle. Yksiköllä voidaan esim. tallentaa tiedostoja tai isännöidä verkkosivustoa.

SaaS puolestaan tarkoittaa, että kokonaisen yksikön sijaan käyttäjälle tarjotaan suoritettavaa sovellusta. Web sähköpostijärjestelmät ovat SaaS palveluita, koska samalla yksiköllä voidaan suorittaa useampaa järjestelmää, mutta käyttäjää veloitetaan vain hänen sovelluksestansa.

PaaS palvelu on yhdistelmä IaaS ja SaaS palveluista. Käyttäjä maksaa sekä käytettävästä laitteistosta että siinä suoritettavasta sovelluksesta. Näitä ovat esim. ecommerce palveluiden tarjoajat. Käyttäjä rakentaa heidän tarjoamallaan sovelluksella verkkokaupan ja isännöi sovelluksen heidän servereillään.

2.1.3 Hyödyt ja haitat

Pilvipalvelun hyödyt ovat riippuvaisia käyttäjästä ja tarjottavasta palvelusta, mutta yhdistävä tekijä kaikilla on helppous ja palvelun tarjoama kätevyys. Palvelut myös voivat nopeuttaa ja laskea konseptoinnin kuluja. Yhtiöiden ja yksilöiden ei tarvitse opetella tai pystyttää kompleksisia järjestelmiä saavuttaakseen visionsa. Paas palvelun tapauksessa käyttäjän ei tarvitse huolehtia laitteiston päivittämisestä, kuluista tai siihen tarvittavasta työvoimasta. Palvelun hintaa voidaan muokata riippuen käytöstä ja tarjoajaa voidaan vaihtaa hyvinkin nopealla aikataululla.

Pilvipalvelu ei kuitenkaan ole paras vaihtoehto, mikäli tiedetään sovelluksen elinkaaren olevan mittava tai käytettävyysasteen tasainen. Riippuen järjestelmän kompleksisuudesta uuden kehittäminen voi olla huomattavasti edullisempaa, kuin palvelun käyttäminen. Vaativien sovelluksien tapauksessa pilvipalvelut voivat hidastua tai epäonnistua tarjoamaan korkeita käyttöasteita, mikäli hintaluokassa sitä ei ole huomioitu.

Palveluiden käyttäjät ovat myös riippuvaisia tarjoajasta. Tarjoaja voi lopettaa palvelun tarjoamisen tai altistua tietomurroille. Ja tärkein haitta on tietenkin, että käyttäjä ei voi hyödyntää palvelua, mikäli hänellä ei ole internet yhteyttä.

2.2 API

2.2.1 Mikä on API?

API on akronyymi Application Programming Interface:sta ja tarkoittaa ohjelmaa, jonka avulla vähintään kaksi ohjelmaa voivat keskustella keskenään. Yksinkertaisuudessaan API toimii kahden järjestelmän sanansaattajana ja tulkkina. API määrittelee yhteiset käytettävät kutsut, datan ja formatoinnin.

2.2.2 API tyypit

Kaikki modernit APIt käyttävät HTTP-protocollaa käytettäviin kutsuihin. Nämä APIt ovat nimeltään Web API. Web APIt käyvät keskustelua web selaimen kautta ja voivat olla suljettuja, avoimia tai kumpaakin. Avoimet APIt ovat kaikkien käytössä tai niiden käyttö vaatii minimaalisia käyttöoikeuksia. Nämä ovat usein tarkoitettu avoimesti käytettävän datan hyödyntämiseen. Suljetut APIt ovat vahvasti salattuja ja vaativat aina jonkinlaisen autentikoimisen.

API protokollat määrittelevät API-kutsujen data tyypit ja komennot. Protokollat ovat riippuvaisia käytettävästä API arkkitehtuurista. REST arkkitehtuurissa API:n tarkoituksena on eriyttää käyttöliittymä data kerroksesta mahdollistaen täten näiden erillään kehittämisen. Arkkitehtuurissa ei tallenneta kontekstiin asiakasohjelman tietoja ja ohjelmat voivat tallentaa kutsuja välimuistiin.

RPC tyyppinen API arkkitehtuuri on etäkäytön kutsuprotokolla. XML-RPC käyttää XML-koodausta kutsuissa, kun taas JSON-RPC käyttää JSON-koodausta. Molempien protokollien kutsu voi sisältää useita parametreja ja odottaa yhtä tulosta. RPC tyyppisessä API:ssa kutsutaan metodeja toisin kuin REST:ssa, jossa vaihdetaan dokumentteja keskenään. RPC:ssä URI paljastaa serverin, mutta ei sisällä parametreja. REST arkkitehtuurissa parametrit sisällytetään kutsun URI:ssa.

3 TEKNOLOGIOIDEN VALINTA

3.1 Pilvipalvelu

Toteutuneen projektin toivottiin olevan skaalautuva ja käytettävyysajan mahdollisimman korkea ja luotettava. Täysin räätälöidyn palvelinklusterin tekeminen käytössä olevien palvelimille ei ollut rahallisesti eikä ajallisesti järkevää. Joten oli hyödynnettävä jonkinlaista valmista IaaS- ja SaaS-palvelua. Vaihtoehtoina olivat markkinoiden kolme suurinta palveluntarjoajaa: Microsoftin Azure, Amazonin AWS tai Googlen Cloud.

Palveluista AWS (Amazon Web Services) on markkinoiden vanhin ja osuudeltaan suurin. AWS:n laajaan tarjontaan kuuluu mm. tietojenkäsittelyä, tietokantojen ylläpito ja sisällön tuottamis- palveluja. AWS:llä on opinnäytetyön kirjoittamisen ajankohtana myös eniten palvelinkeskuksia ja hinnat instanssia kohden 69 $ kuukaudessa. Vertailun vuoksi instanssi on asetettu palvelin, jossa on 2 virtuaalista CPU:ta ja 8 GB RAM:a. Palvelu vaikutti paperilla parhaimmalta mahdolliselta vaihtoehdolta.

Googlen Cloud on palveluista ”pienin” ja ”halvin”. Yksi instanssi Googlen Cloudia olisi maksanut vain 52 $. Googlen tarjoamat palvelut ovat tosin suunnattu enemmän luonnollisien kielien prosessointiin ja koneoppimiseen. Työn datan hallinnan tarkoitukseen kehnoin vaihtoehto kolmesta. Työn toteutuksen aikana Googlella oli myös vähiten konesaleja tarjolla.

Microsoftin Azure on kaikista kolmesta kallein. Yhden instanssin hinta on hieman yli Amazonin 70 $ kuukaudelta. Azure on vaihtoehdoista myös ainut, joka oli alusta asti suunniteltu sovelluksien ylläpitämiseen ja rakentamiseen hyödyntämällä Microsoftin laajaa keskuspalvein verkostoa. Azuren blob storage, API Management studio, alhaisin kysynnän mukainen hinnoittelu ja tietenkin tekijän palvelimen tuttavallisuus vaikuttivat palvelun valintaan.

3.2 Tietokanta

Pilvipalvelun valinnan jälkeen pohdittiin ja tutkittiin Azuren kanssa yhteensopivia tietokanta palveluita. Valitun teknologian oli kyettävä skaalautumaan, helppo ylläpitää, turvallinen ja nopea. Kannan oli myös oltava ei relationaalinen ruokalistojen dokumentoinnin takia. Useamman ruokalistan olisi kyettävä hyödyntämään samoja reseptejä ja allergeenejä.

Tietokantamallin oli kyettävä ns. kasvaa tarpeen tullen joko lisäämällä muuttujia tai irtonaisia dokumentteja uusiksi tauluiksi. Nämä vaatimukset rajoittivat tietokannan teknologian hyödyntävän NoSQL mallia. Yhdistämällä kaikki vaatimukset päädyttiin Microsoftin tuotteeseen Azure Cosmos DB. Nimestäkin päätellen voidaan todeta tuotteen toimivan moitteetta Azuren kanssa.

3.3 API

Tietokannasta poiketen itse API:n teknologia ei ollut riippuvainen aiemmin valituista teknologioista. API:n oli kuitenkin oltava WEB API sillä Azuren tarjoama App Service oli täydellinen osuma API:n vaatimuksiin. App Servicen avulla API:n tietoturva, autentikointi ja versiointi hoidetaan suoraan Azuressa. Itse API:lle on ohjelmoitava vain uusia pyyntöjä ja logiikan muutokset.

Web API on REST arkkitehtuuria hyödyntävä applikaatio. Kuten nimestä voidaan päätellä, Web API:a tavoitellaan HTTP protokollan kautta web palvelimen avulla. Palvelin tutkii pyyntöä, määrittelee protokollan avulla, mitä käyttäjä haluaa ja lähettää vastauksen paketoituna takaisin. Esim. HTTP GET protokollan tapauksessa palvelin vain palauttaa ja vain lukee tietoa.

Web API toteutettiin ASP.NET alustalla C# ohjelmointikieltä käyttäen. ASP.NET HTTP protokollien kartoittaminen on tehty Wep API:a ajatellen ja tukee laajasti käytettyjä JSON, XML formaatteja.

3.4 Versionhallinta

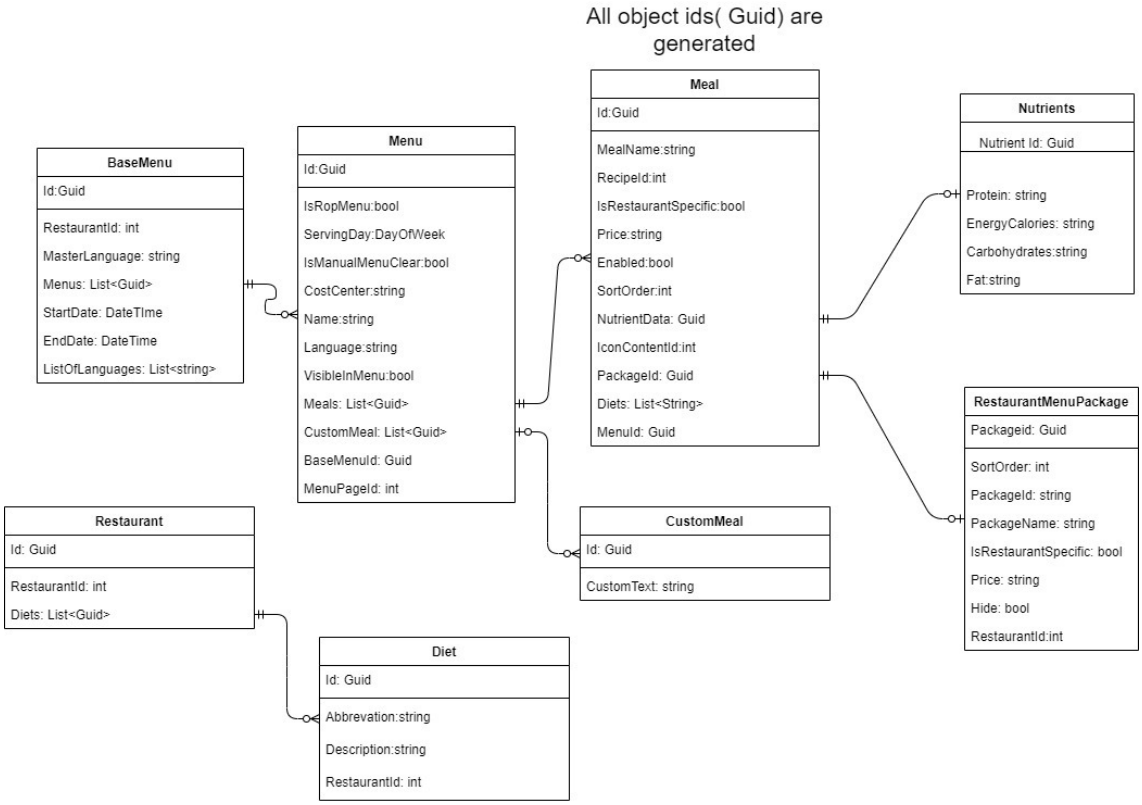
Versionhallinnan haluttiin olevan mahdollisimman luotettava ja automaattinen. Välttyäkseen käyttäjävirheiltä hallinnassa käytettiin useampaa teknologiaa. Itse projektin koodipohja eli .NET API:n koodi säilytettiin GitHubissa. API projektin rakentaminen ja paketoiminen tehtiin automaattisesti Jenkins- järjestelmän avulla uuden version saapuessa. Paketoitu versio lähetettiin Octopus Deploy- sovellukseen, jossa tarvittaessa oikea versio lähetettiin Azuren Web App Serviceen manuaalisesti.

4 IMPLEMENTOINTI

4.1 Tietomallin suunnittelu

Projektia edeltävä CMS:n käytössä oleva tietokantamalli haluttiin tehdä uudestaan helpottaakseen uusien ruokalistojen luomista ja muokkaamista. Uuden tietomallin oli kuitenkin säilytettävä jonkinlainen yhtenäisyys vanhaan malliin, jotta CMS luokkien käytettävyys säilyisi.

Tarkoituksena oli tehdä malli, jonka pohjalta ymmärrettäisiin CMS luokkakaavio ja hyödyntäisi NoSQL -mallia mahdollisimman paljon (Kaavio 1).



KAAVIO 1. Tietomallin 2. iteraatio

Tietomallia suunnitellessa tarkoituksena oli hyödyntää ravintoloiden ruokalistojen rytmitystä. Ravintolat uusivat ruokalistansa viikoittain, jota mallin oli heijastettava. Mallissa on yksi viikoittainen BaseMenu, joka sisältää listan viikon menujen id:tä. BaseMenua voidaan ajatella ravintolana, jolla on viikon menut suunniteltuna. Menut ovat esim. lounas tai päivällinen. Lounaalla voi olla useampi tarjoiltava ateria,

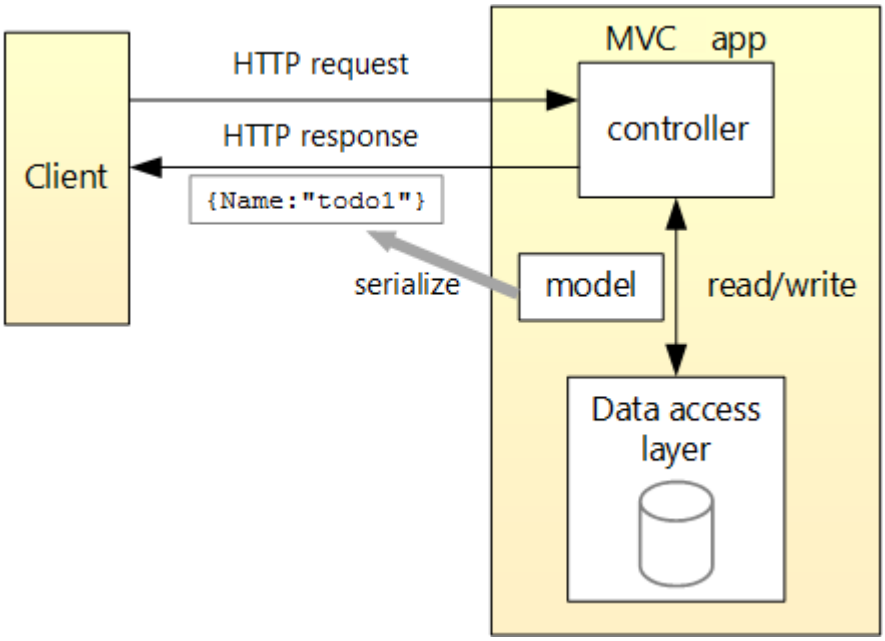
kuten salaatti tai päivänkeitto. Näitä kuvastetaan meal dokumentilla. Jokainen ateria sisältää tiedon ravinnearvoista, allergeeneista (diet- taulukko) ja hintatiedoista. Hintatiedot oli eriytettävä ateriasta, sillä useampi ateria voi olla samanhintainen samassa ravintolassa.

Tietomalli eli koko projektin ajan ja sitä iteroitiin useampaan otteeseen uusia tarpeita huomattaessa.

4.2 ASP.NET Web API

4.2.1 API:n tekeminen

Projektia ohjelmointiin käyttäen Visual Studiota ja .NET SDK 5.0 tai uudempaa versiota. Projekti käytti ASP.NET Core Web API templaattia. Web API toiminta kostuu 3 osasta: Controller, Model ja Data layer (Kuva 1). Applikaation kanssa keskustelu käydään vain kontrollereiden kautta.



KUVA 1. Web API design

APIen toteuttaminen alkoi tietomallin siirtämisellä model luokkiin. Luokkien on oltava täsmälleen tietokannassa olevien mallien mukainen välttyäkseen kääntämisen virheilta tai varoituksilta.

Mallien valmistuessa voidaan aloittaa itse logiikan implementointi eli kontrollereiden ohjelmointi. Applikaatio tarvitsee ainakin yhden kontrollerin, mutta kyseisessä Web API:ssa tehtiin kaksi kutakin tarvetta varten.

Ensimmäinen kontrolleri oli vastuussa GET toiminnoista. Ohjelmakoodi 1. voidaan havaita, että syöttämällä apin perus URL:n jälkeen ” "getbasemenu/{menuId}" aaltosulkujen sisään tarvittavat tiedot palauttaa API täsmäävän viikoittaisen ruokalistan.

```
//getbasemenu
[HttpGet("getbasemenu/{menuId}")]
0 references | paulaka, 311 days ago | 2 authors, 5 changes
public async Task<ActionResult<BaseMenu>> GetBaseMenu(Guid menuId)
{
    var baseMenu = await _menuDataService.GetItem<BaseMenuAndMenus>(menuId);
    if (baseMenu == null)
    {
        return NotFound();
    }

    return Ok(baseMenu.Data.BaseMenu);
}
```

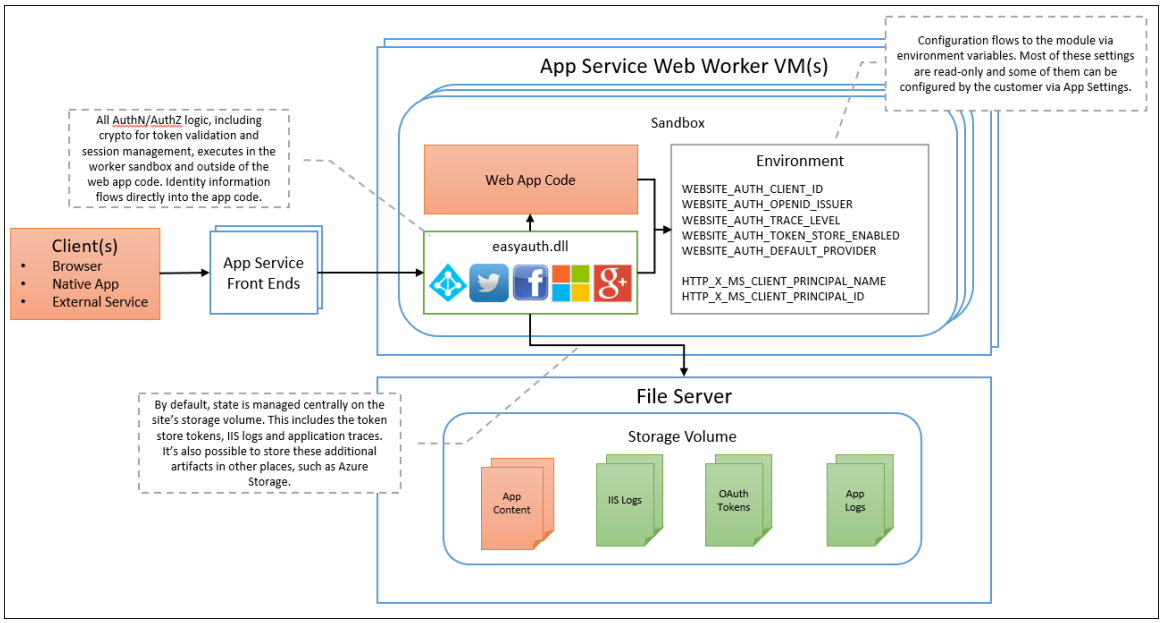
OHJELMAKOODI 1. GET basemenu API

Palautettu arvo on formatoitu JSON:ksi(Kuva 3) ja riippuen tuloksesta status on HTTP 404(Not Found) tai HTTP 200(OK). Toinen kontrolleri on vastuussa metodeista DELETE, POST ja PUT. Tietoja muokkaavien metodien implementointi tapahtui samalla tavalla. Huomioiden, että saman url:n omistavat API:t voidaan pakottaa toteuttamaan eri toimintoja hyödyntämällä eri HTTP protokollia.

4.2.2 Autentikointi

Jotta APIa ei voi käyttää kuka tahansa on siihen lisättävä jonkinlainen käyttäjän identiteetin tarkistamisen metodi. Projektissa käytettiin Azure App Servicen sisäänrakennettua ”Easy Auth” toimintoa.

Autentikointi moduulia suoritetaan samassa hiekkalaatikossa ohjelmakoodin kanssa.ja se toimii uudelleenohjaamalla liikenteen HTTPS:n kautta riippumatta App Servicen asetuksista. Mikäli haluamaton taho saa pääsyn Azuren järjestelmään hän ei voi ottaa autentikointia pois. (Kaavio 2)



KAAVIO 2. Autentikointi prosessikaavio

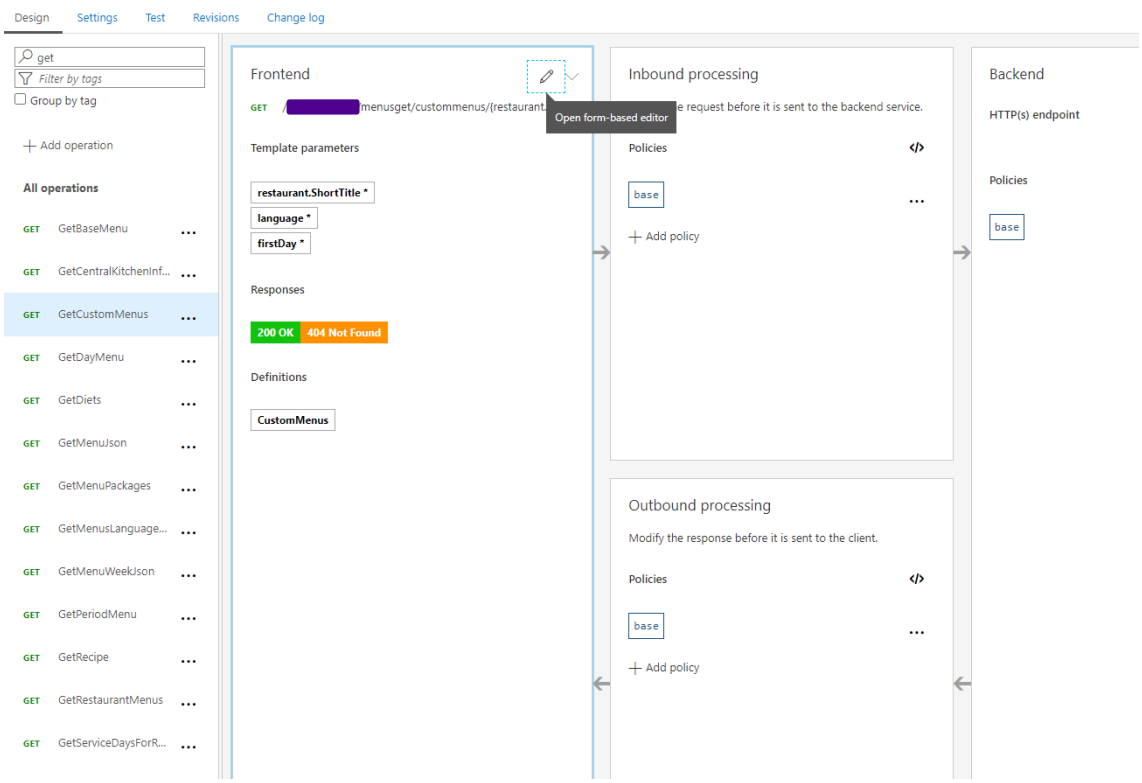
Autentikaatio toimii ASP.NET JSON Web Tokens autentikaatiolla. Autentikaatio toimii hakemalla HTTP kutsusta autentiikaatioavaimen kutsun bodysta. Mikäli toiminto asetetaan Azuren App Service:ssa toimivaan applikaatioon, kysytään autentikoituvalta hänen Azuren AD.

Avaimesta saadaan esim. käyttäjän ID, omistaja, itse avain ja rooli. Mikäli käyttäjällä ei ole sallittua roolia tiedon muokkaaminen API managementin kautta ei onnistu.

4.3 Azure App Service

4.3.1 Azure API management

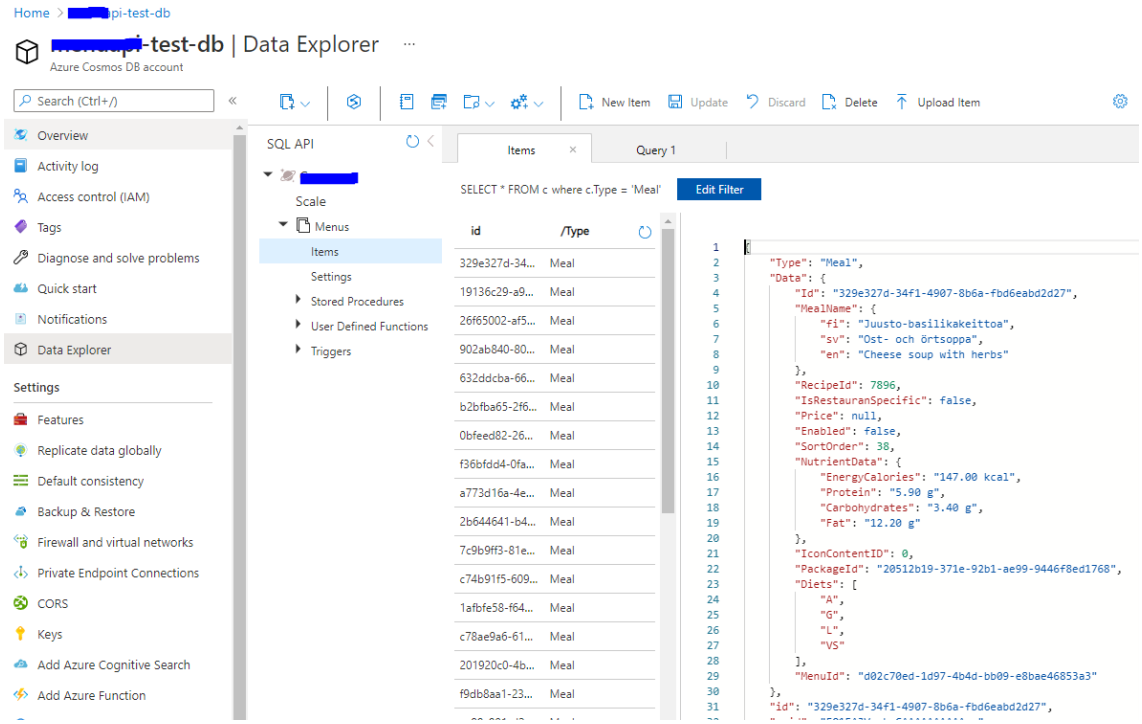
Web API toimii Azuren App Service kontissa, joka on liitettynä resurssi ryhmään. Kontille on mahdollisuus tehdä API management instanssi, joka hallitsee ASP.NET API:n ja Cosmos DB välisen kommunikoinnin, autentikoinnin ja salaisuuksien hallinnan. Salaisuudet ovat esim. apeissa useasti toistuvat muuttujat.



KUVA 2. Azure Api Management UI

Jokaiselle API:lle oli tehtävä vastaava toteutus pilveen. (Kuva 2) Azuren API:t käyttävät definitioneja, jotka vastaavat koodi puolen luokkia ja täten seriliasoinnille ei ole tarvetta.

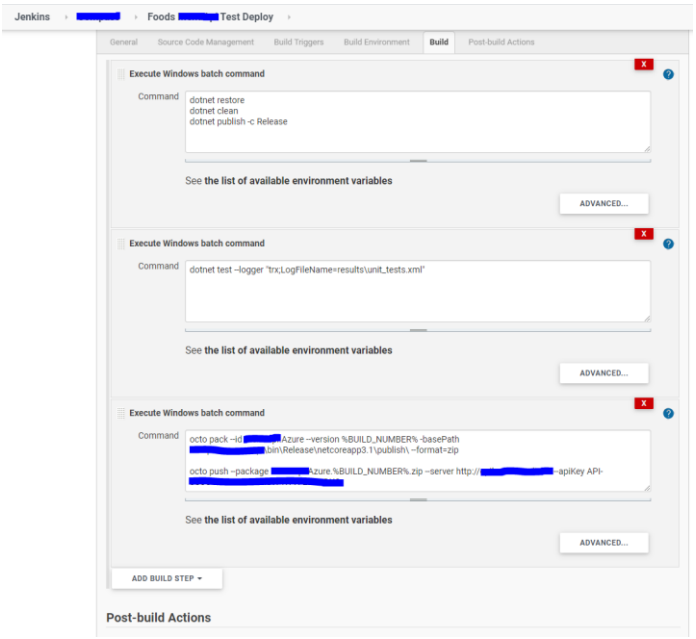
Azure APIM tavoin tietokannan lisääminen resurssi ryhmään käy kokonaan Azuren UI:n kautta. Cosmos DB toimii Azuren tavoin konteilla, joiden sisällä on tallennettu data. Kontin sisällä pystyy myös tekemään kevyitä kantakyselyitä. (Kuva 3)



KUVA 3. Azure Cosmos DB UI

4.4 Versionhallinta

Versionhallinnassa ensimmäinen vaihe on rakentaa, pakata ja lähettää tehty sovellus käyttöön otettavaksi. Tämä tehtiin käyttämällä automatisoidulla Jenkins serverillä. Kuten kuvassa 4 nähdään, sovellus ensin rakennetaan käyttäen profiilia ”Release”, ajetaan testit ja lähetetään Octopus Deployhin.



KUVA 4. Jenkins build

Octopus Deployn saatua lähetetty voidaan rakentaa uusi deploy. Azureen lähettäminen tapahtuu konfiguroidulla autentikoidulla käyttäjällä. Octopus Deployhin voi myös integroida hälytyksiä ja notifikaatioita, jotka helpottavat entisestään version seuraamista.

5 LOPPUPOHDINTA

Työn tavoitteena oli siirtää ruokalistojen, menujen ja ravintoloiden tarjonta CMS sisällönhallinta palvelusta itsenäiseksi järjestelmäksi. Entinen palvelunmuoto todettiin olevan liian vaivalloinen ylläpitämisen ja järjestelmän jaettavuuden kannalta. Toteutetun järjestelmän toivottiin olevan täysin erillään entisestä toteutuksesta, hyödyntävän skaalautuvaa pilvipalvelua ja toteutuksen ylläpitäminen mahdollisimman edullista.

Pilvipalveluun siirtyminen antoi järjestelmälle suuremman tavoitettavuusajan, vähensi ylläpitäjien työtaakkaa ja antoi hyvän pohjan myöhemmille lisäosille. Projektin sisäisessä testaamisessa havaittiin tavoitettavuudessa olevan viivettä useamman samanaikaisen kutsun tapahtuessa. Tämä luultavimmin korjataan tilaamalla Azuresta enemmän prosessointi kykyä.

Projektin lisääminen aiemmin olemassa olevaan pilvipalveluun oli ajallisesti ja rahallisesti säästävä päätös. Versionhallinta toimii moitteettomasti, mutta siitä olisi voinut karsia muutaman vaiheen ja käyttää modernisempaa vaihtoehtoa kuten esim. käyttämällä GitHub Actionsia.

Projektissa saavutettiin kaikki toivotut toiminnot. Järjestelmän toiminnallisuus siirrettiin täysin pilvipalveluun ja ulkopuolisilla käyttäjillä on mahdollisuus käyttää heille tarkoitettuja toimintoja. Järjestelmä on tarpeeksi turvattu hyökkäyksiltä, mikäli semmoisia tapahtuu. Ainoat ongelmat projektin tuotantoon viennin jälkeen voivat syntyä koodipohjan vanhetessa tai pilvipalvelun katkoksissa. Projektia voidaan jatkokehittää helpottamalla CI/CD putken koodipohjan lähettämistä ja optimoimalla API kutsuja.

LÄHTEET

Microsoft Azure dokumentaatio, Luettu 10.05.2021, https://azure.microsoft.com/en-us/

Comsos DB dokumentaatio, Luettu 11.05.2021, https://docs.microsoft.com/en-us/azure/cosmos-db/introduction

Microsoft ASP.NET WEB API, Luettu 16.05.2021, https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-5.0

Google Cloud hinnasto ja dokumentaatio, Luettu 18.05.2021, https://cloud.google.com/docs

Amazon Web Services hinnasto ja dokomentaatio, Luettu 18.05.2021, https://docs.aws.amazon.com/

KUVA 1, Luettu 16.05.2021, https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-5.0&tabs=visual-studio

KAAVIO 3, Luettu 16.05.2021, https://docs.microsoft.com/en-us/azure/app-service/overview-authentication-authorization