Azure Cloud -resurssien valinta sovelluksen käyttöönotossa

Iuliia Kokorieva

Haaga-Helia ammattikorkeakoulu
Tradenomi, Tietojenkäsittelyn koulutusohjelma
Toiminnallinen opinnäytetyö
2024

Tiivistelmä

Tekijä(t)
Iuliia Kokorieva

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
Azure Cloud -resurssien valinta sovelluksen käyttöönotossa

Sivu- ja liitesivumäärä
51 + 3

Tämä opinnäytetyö käsittelee Azure Cloud -resurssien valintaa sovelluskehityksessä. Työ on tehty toiminnallisena opinnäytetyönä, ja se käsittelee Azure Cloud -resurssien valintaa sovelluskehityksessä. Opinnäytetyön perustana käytetään kolmiokomponenttista sovellusta, johon sisältyy Node.js:llä kirjoitettu Azure Functions -backend, Azure SQL -tietokanta ja React-pohjainen frontend, joka on otettu käyttöön Azure App Services -palveluna.

Teoreettinen osa kattaa pilviteknologioiden peruskäsitteet, erityisesti ne, jotka liittyvät projektin toteutukseen. Toisena aiheena ovat DevOps ja CI/CD -käytännöt, joita tekijä hyödynsi Agile-lähestymistavalla työskennellessään projektin parissa.

Empiirinen osa seuraa jokaisen sovelluskomponentin toteuttamista. Tämä sisältää tutkimus-, arviointi- ja perusteluprosessin asianmukaisten resurssien valinnan osalta sekä valittujen ratkaisujen vaiheittaisen kuvauksen toteutuksesta. Komponenttien välinen integraatio, salaisuudenhallinta sekä CI/CD ovat olennainen osa projektia.

Yhteenveto- ja päätösosa käsittelee tehtyjä päätöksiä, ehdottaa parannuksia ja tutkii tulevaisuuden kehitysvaihtoehtoja. Työn arviointi suoritetaan tarkastelemalla niitä etuja, joita se tuottaa tekijän työllistymiselle pilviteknologioiden alalla.

Asiasanat
Pilvipalvelut, Azure Cloud, Fullstack, CI/CD

Sisällys

Sanasto 1

1 Johdanto 2

2 Opinnäytetyön yleiskatsaus 4

2.1 Valitun aiheen perustelu 4

2.2 Opinnäytetyön rakenne 4

2.3 Tutkimustavoitteet ja odotetut tulokset 5

3 Lyhyt katsaus pilviteknologiaan ja niiden edut 7

3.1 Pilvipalveluiden peruskäsitteet 7

3.2 Pilviteknologian markkinoiden suurimmat toimijat ja Azuren valinnan perustelu tähän työhön 8

4 Agile-lähestymistapa ja CI/CD-käytännöt nykyisen projektin kontekstissa 10

5 Johdanto empiirisen projektin toteuttamiseen 12

5.1 Empiirisen tutkimusidean alkuperä 12

5.2 Projektityön suunnittelu 13

5.2.1 Agile-lähestymistapa ja Trello-taulun käyttö projektin edistymisen seurantaan ja hallintaan 13

5.2.2 Versiohallinnan käyttö 14

6 Fullstack-sovelluksen pilvipalveluun käyttöönoton toteutus 17

6.1 Projektin pilviarkkitehtuurin suunnittelu 17

6.2 Ratkaisun tutkimus ja valinta relaatiotietokannan isännöimiseksi Azuren ympäristössä 18

6.2.1 Azure SQL -tietokannan valinnan perustelut 19

6.2.2 Tietokannan kehittäminen ja käyttöönotto Azure-ympäristössä 20

6.3 Ratkaisun tutkimus ja käyttöönotto backendin isännöimiseksi Azuren ympäristössä 22

6.3.1 Perustelut Azure Functions -palvelun valinnalle Azure-resurssina backendille 23

6.3.2 Azure Functions- ja Azure App Service -palvelun käytön vertailu nykyisessä projektissa 24

6.3.3 Empiirinen raportti Azure Functions -sovelluksen kehittämisestä 25

6.4 Ratkaisu frontendin käyttöönottoon Azure-ympäristössä 30

6.4.1 Azure App Services -palvelun valinnan perustelut 30

6.4.2 React-sovelluksen kehittäminen ja julkaisu Azure App Services -palveluun 31

6.4.3 GitHub-putkiston asettaminen frontendin jatkuvaa julkaisua varten 33

6.4.4 Salaisuuksien hallinta toteutetussa ratkaisussa 36

6.5 Tulosten yleiskatsaus 37

6.5.1 Keinot sovelluksen parantamiseen ja jatkokehittämiseen 38

6.5.2 Budjettikatsaus 40

7 Pohdinta 42

7.1 Johtopäätökset ja korjausehdotukset toteutetusta projektista 42

7.2 Työn tulosten sovellettavuus tekijän työtehtäviin ja tavoitteisiin 44

8 Lähteet 45

9 Liitteet 48

Liite 1. Projektiin liittyvät linkit 48

Liite 2. Putkistotiedoston sisältö 49

Sanasto

API – Sovellusliittymä: ohjelmointirajapinta, joka mahdollistaa eri ohjelmistojen tai palveluiden välisen viestinnän.

CI/CD – Jatkuvaa integraatiota ja jatkuvaa toimitusta: ohjelmistokehityksen käytäntö, joka automatisoi testauksen ja toimituksen prosesseja.

CLI – Komentorivikäyttöliittymä: käyttöliittymä, joka sallii käyttäjän kommunikoida tietokoneen kanssa tekstikomentojen avulla.

IaaS – Infrastruktuuri palveluna: pilvipalvelumalli, jossa tarjotaan virtuaalista laitteistoa ja muita infrastruktuurin komponentteja.

IaC – Infrastruktuurin koodaus: käytäntö, jossa infrastruktuurin hallinta automatisoidaan koodilla.

PaaS – Alusta palveluna: pilvipalvelumalli, jossa tarjotaan kehitys- ja käyttöympäristöä sovellusten luomiseen.

PAYG – Pay-as-you-go / maksa käytön mukaan: hinnoittelumalli, jossa maksat palvelusta vain sen mukaan, kuinka paljon sitä käytät.

SPA – Yhden sivun sovellus: verkkosovellus, joka toimii yhdellä sivulla ilman sivujen uudelleenlatausta.

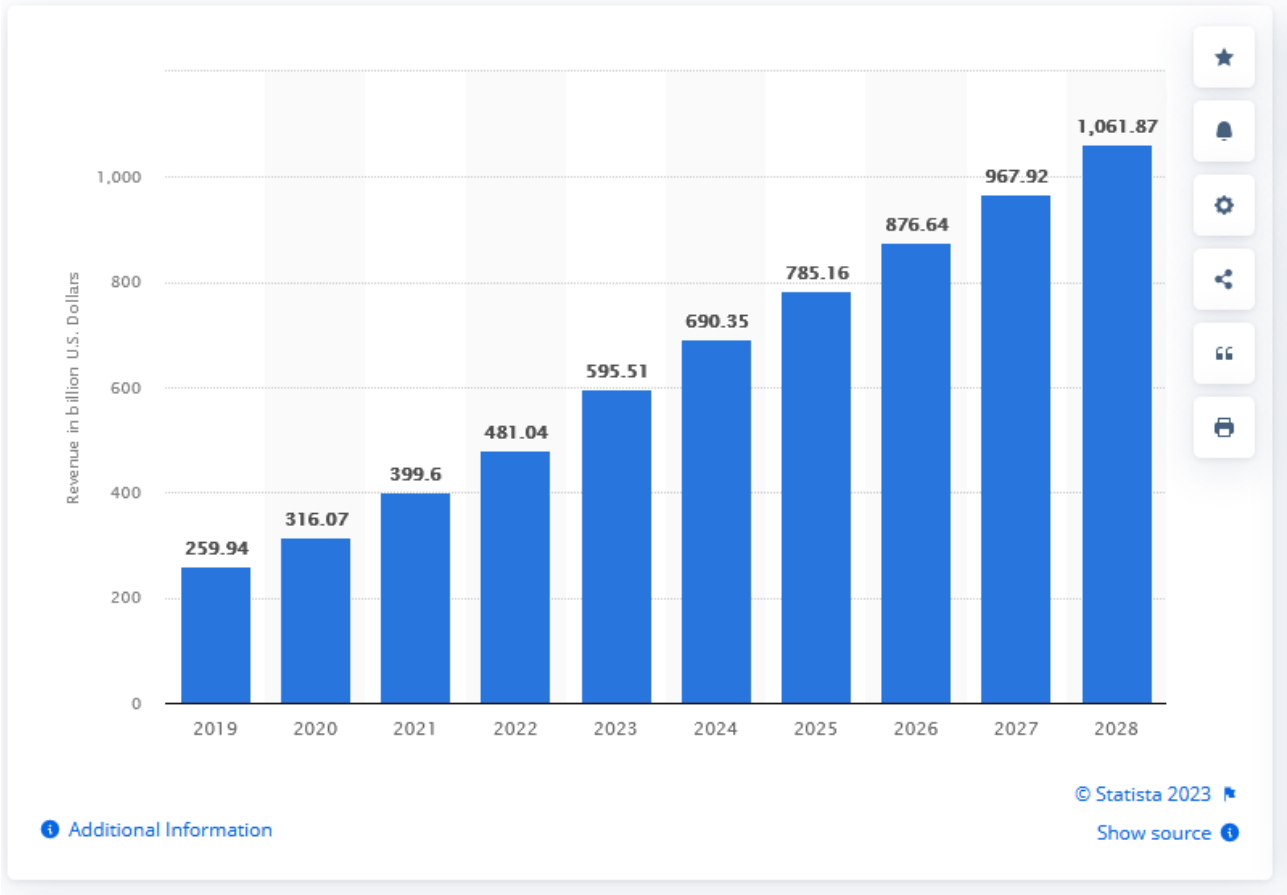
1 Johdanto

Viime vuosikymmenen aikana pilvipalvelut ovat lisänneet suosiota merkittävästi muuttaen teknologista maisemaa ennennäkemättömällä tehokkuudella ja saavutettavuudella. Pilviteknologian nykyinen aikakausi alkoi noin vuonna 2006, kun Amazon Web Services (AWS) julkaisi Elastic Compute Cloud (EC2) -palvelunsa, mikä oli merkittävä hetki pilvipohjaisiin palveluihin siirtymisessä. Tämän jälkeen vuonna 2010 julkaistiin MS Azure ja vuonna 2011 Google Cloud, mikä vahvisti entisestään pilvipalveluiden suosiota. Vuonna 2014 AWS toi markkinoille uuden palvelunsa, Lambdan, joka esitteli uudenlaisen palvelutyypin – palvelimettoman tietojenkäsittelyn (serverless computing) nostaen julkisen pilvimarkkinan kehityksen uudelle tasolle.

Julkiset pilvimarkkinat ovat kokeneet jatkuvaa kasvua alkuajoistaan lähtien. Tätä laajentumisen dynamiikkaa voidaan kuvastaa seuraavaksi kuvatuilla tilastotiedoilla.

Esimerkiksi, nykyään niin yksityishenkilöt kuin yrityksetkin ovat yhä riippuvaisempia pilvipalveluista. Viimeaikainen Google Cloudin raportti korostaa pilviosaamisen nostamista ensisijaiseksi painopisteeksi organisaatioiden henkilöstöaloitteissa. Huomattavasti yli puolet yrityksistä joko palkkaa uutta henkilöstöä tai kouluttaa olemassa olevaa henkilökuntaa optimoidakseen pilvikustannuksiaan. Yleinen strategia on "pilvi ensin" -lähestymistavan omaksuminen uusien sovellusten käyttöönotossa, ja 47 % eri toimialojen organisaatioista valitsee julkiset pilvipalvelut tällaisiin tarkoituksiin. Pilvipalvelujen suosio jatkaa kasvuaan. Vuonna 2022 76 % ihmisistä ilmoitti käyttävänsä julkista pilveä, mukaan lukien useita pilvipalveluita – merkittävä nousu verrattuna vuoden 2021 56 %:iin (Franklin 2023).

Kuva 1 alla tarjoaa graafisen esityksen rekisteröidystä ja ennustetusta markkinakasvusta. Tutkimustulosten mukaan odotettavissa oleva kasvu vuosien 2023 ja 2028 välillä on +78 %.



Kuva 1. Julkisen pilvimarkkinan tulot maailmanlaajuisesti vuosina 2019–2028 (Statista 2023).

Vastaavia tilastoja tarjoaa Gartner, arvostettu amerikkalainen tutkimus- ja konsultointiyritys tietotekniikan alalla. Heidän ennusteensa osoittavat myös, että vuoteen 2026 mennessä 75 % yrityksistä on omaksunut uuden digitaalisen mallin pilviteknologioiden pohjalta (Gartner 2023).

Suomea koskevien tilastojen kohdalla on huomionarvoista, että vuosina 2020 ja 2021 Suomi johti Euroopan unionia yritysten käytössä olevien pilvipalveluiden käyttöönotossa saavuttaen vuosina 2020 ja 2021 75 % ja 75 % käyttöasteet, kun EU:n vastaavat keskiarvot olivat 36 % ja 41 % (Eurostat 2021). Viimeisimpänä Microsoft on ilmoittanut uuden tietokeskuksen ja Azure-alueen aloittamisesta Espoossa ja Kirkkonummella (YLE 17.03.2022). Tämä kehitys korostaa pilvipalveluiden merkitystä Suomessa ja kasvattaa samalla kysyntää alan ammattilaisille.

2 Opinnäytetyön yleiskatsaus

Tämä opinnäytetyö perustuu ohjelmistoprojektin toteutukseen. Ohjelmistoprojekti, joka toimii tämän raportin perustana, juontaa juurensa tekijän Azuren teknologioiden oppimisprosessista. Toiminnallisen opinnäytetyön vaatimusten mukaisesti tämä raportti kuvaa ne teoreettiset perusteet, joihin projekti perustuu. Aiheina ovat pilviteknologiat pilvipohjaista kehitystä varten ja DevOps-työskentelytapa. Opinnäytetyön pääosana on yksityiskohtainen selostus sovelluksen toteutuksesta ja käyttöönotosta. Erityinen huomio on siinä, miksi juuri tietyt Azuren resurssit valittiin tähän projektiin.

2.1 Valitun aiheen perustelu

Tutkimusaiheen valinta perustuu tekijän ammatilliseen oppimiseen hänen työskennellessään Tietoevryllä, suurella IT-konsultointiyrityksellä. Tietoevry tarjoaa monipuolisia palveluita, kuten räätälöityä ohjelmistokehitystä, digitaalisointia ja muita erilaisia ratkaisuja asiakkailleen. Erityisesti pilviteknologiat, jotka ovat kokeneet voimakasta kasvua viimeisten 10–15 vuoden aikana, muodostavat tärkeän osan yrityksen osaamista.

Tekijän ensimmäisen työvuoden aikana Tietoevryllä hän oli mukana projektissa, johon sisältyi Microsoft Azuren pilvipohjaisen palvelun kehittämisen teollisuusasiakkaan käyttämän vanhan järjestelmän korvaamiseksi. Ennen tämän projektin aloittamista tekijällä oli vain vähän kokemusta pilviteknologioiden alalta. Siksi jotta hän voisi osallistua projektityöhön, hän käytti aikaa tarvittavan tiedon ja taitojen hankkimiseen. Sovellus ja sen käyttöönottoratkaisu, jotka toimivat tämän opinnäytetyön perustana, syntyivät tekijän oppimisen tuloksena ennen tätä projektia ja sen aikana, vuoden 2023 ensimmäisellä puoliskolla. Vaikka työnantaja ei ollut virallisesti pyytänyt tämän projektin kehittämistä, se syntyi orgaanisesti oppimisen haasteista ja välttämättömyyksistä, tehden siitä suoran vastauksen työpaikan vaatimuksiin. Siten tämä opinnäytetyö heijastaa teoreettista tutkimusta ja empiiristä oppimista, jotka nousivat esiin tämän ajanjakson aikana.

2.2 Opinnäytetyön rakenne

Opinnäytetyö on jaettu selkeisiin osiin, joista jokaisella on oma erityinen tarkoituksensa, ja ne kuvataan lyhyesti tässä kappaleessa.

Ensimmäinen osio koostuu luvuista 1: Johdanto ja 2: Opinnäytetyön yleiskatsaus. Nämä luvut osoittavat opinnäytetyön ajankohtaisuuden pilviteknologioiden alalla. Ne kertovat tekijän ammatillista kokemuksesta tältä alalta ja tarjoavat yleiskatsauksen opinnäytetyön sisällöstä. Lisäksi

nämä luvut esittävät tietoa seuraavien lukujen sisällöstä, tutkimuksen tavoitteista ja odotetuista tuloksista.

Seuraava osio on teoreettinen ja sisältää luvun 3: Lyhyt katsaus pilviteknologiaan ja niiden edut, joka tarjoaa kattavan yleiskatsauksen pilvipalvelukonsepteista ja suurimmista markkinoilla toimivista tarjoajista, ja luvun 4: Agile-lähestymistapa ja CI/CD-käytännöt nykyisen projektin kontekstissa, jossa käsitellään Agilen, DevOpsin ja CI/CD:n perusperiaatteet ja niiden väliset suhteet.

Laajin osuus muodostuu empiirisestä näkökulmasta ja sisältää luvut 5: Johdanto empiirisen projektin toteuttamiseen ja 6: Fullstack-sovelluksen pilvipalveluun käyttöönoton toteutus. Edellinen luku tarjoaa näkemyksiä tekijän kehittämästä fullstack-projektista, mukaan lukien sen arkkitehtuurirakenne, jota käytettiin tulevien kehitysvaiheiden perustana tässä opinnäytetyössä. Jälkimmäinen luku puolestaan pureutuu empiirisen tutkimuksen toteutukseen kattaen valmistelun ja pilvi-infrastruktuurin perustamisen web-sovelluksen kaikkiin osiin, mukaan lukien tietokantaan sekä backend- ja frontend-kehitykseen. Erityistä huomiota kiinnitetään kunkin osan käyttöönottovaihtoehtojen analysointiin ja valitun lähestymistavan perustelemiseen.

Lopuksi luku 7: Pohdinta on omistettu työn tulosten arvioimiselle ja niiden yhteensovittamiselle tutkimustavoitteiden kanssa. Lisäksi tässä luvussa pohditaan tekijän oppimiskokemusta ja kuinka se edistää hänen ammatillista kehittymistään.

Ammattikorkeakoulun kirjoituskonventioiden mukaisesti opinnäytetyö päättyy lähde- ja liiteluetteloihin viittauksia varten.

2.3 Tutkimustavoitteet ja odotetut tulokset

Tämän opinnäytetyön ensisijainen tavoite on luoda käytännöllinen ja kestävä ratkaisu kolmikerroksisen web-sovelluksen käyttöönottoon Azure-ympäristössä. Suunniteltava ratkaisu perustuu tutkimukseen ja empiiriseen oppimiseen. Lisäksi kiinnitetään merkittävää huomiota tehokkaan putkiston suunnitteluun ja toteutukseen jatkuvan integroinnin/jatkuvan toimituksen (CI/CD) lähestymistavan mukaan. Näin ollen opinnäytetyön tutkimuskysymykset (TK) kuuluvat seuraavasti:

- TK1: Mikä muodostaa sopivan Azure-arkkitehtuurin yksinkertaisen fullstack-web-sovelluksen käyttöönotossa?
- TK2: Millainen on tehokas ratkaisu CI/CD-lähestymistavan integroinnissa?

Työn tuloksena tulee esiin seuraavat asiat:

- Valitun Azure-arkkitehtuurin ja resurssien selitys.
- Toimiva fullstack-sovellus, joka toimii Azure-ympäristössä.
- Oikein toimiva GitHub Actions -putkisto, joka suorittaa testit ja päivittää käyttöön otetun sovelluksen jokaisen koodimuutoksen yhteydessä, mikäli testit menevät läpi.
- Budjettikatsaus tällaiselle yksinkertaiselle projektille.

Suoritetun työn seurauksena odotetaan seuraavien tuotosten syntyvän:

- Azure-pilviarkkitehtuurin suunnittelu web-sovelluksen isännöintiä varten.
- Infrastruktuurin toteutus Azure-ympäristössä.
- CI/CD-putkiston toteutus.

Sovelluksen koodi itsessään (React-frontend, SQL-kyselyt, joita käytetään tilastollisten laskelmien tuottamiseen tietokannasta) jätetään opinnäytetyön ulkopuolelle. Node.js:llä kirjoitettua koodia, jota käytetään Azure Functions -backendissa, käsitellään rajoitetusti osana Azure Functions -oppimista ja selittämistä.

3 Lyhyt katsaus pilviteknologiaan ja niiden edut

Pilvipalvelut ovat mullistaneet tavan, jolla yritykset ja yksilöt käyttävät ja hallitsevat laskentaresursseja. Tiivistetysti pilvipalvelut tarkoittavat laskentapalveluiden, kuten palvelimien, tallennustilojen, tietokantojen, verkottumisen, analytiikan, ohjelmistojen ja tekoälyn, toimittamista internetin kautta, jolloin voidaan tarjota nopeampia innovaatioita, joustavampia resursseja ja skaalautuvuuden etuja. Sen sijaan, että omistettaisiin ja ylläpidettäisiin fyysisiä palvelimia tai infrastruktuuria, käyttäjät voivat käyttää näitä palveluita pay-as-you-go-hinnoitteluperiaatteella, mikä tekee pilvipalveluista kustannustehokkaan ja tehokkaan ratkaisun.

Tämä luku on kirjoitettu viitaten teoksen "Software Architecture in Practice" lukuun 17 "The Cloud and Distributed Computing" (Bass, Clements, Kazman 2021), Tech-Targetin laajaan oppaaseen pilvipalveluista (Bigelow S.J., Neenan S., Casey K.) sekä tekijän tietoon, jonka hän on saanut osallistuessaan useisiin Microsoftin koulutuksiin työssään Tietoevryllä ohjelmistokehittäjänä.

3.1 Pilvipalveluiden peruskäsitteet

Pilvipalvelut lukuisine alikäsitteineen on laaja ja jatkuvasti kehittyvä aihealue. Seuraavissa osioissa selostetaan joitakin tärkeitä käsitteitä, jotka ovat merkityksellisiä tämän opinnäytetyön perustana käytetylle empiiriselle projektille.

Pilvipalveluissa on kolme ensisijaista palvelumallia: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) ja Software as a Service (SaaS). Nämä eroavat toisistaan pilvipalveluntarjoajan ja asiakkaan vastuualan perusteella. IaaS tarjoaa virtualisoituja laskentaresursseja internetin yli, PaaS tarjoaa kehittäjille alustan sovellusten rakentamiseen, käyttöönottoon ja hallintaan ilman, että palveluasiakkaan tarvitsee huolehtia taustainfrastruktuurista, ja SaaS toimittaa ohjelmistoja internetin yli tilauspohjaisesti.

Virtualisointi on perusteknologia pilvipalveluissa. Se mahdollistaa virtuaalisten versioiden luomisen laskentaresursseista, kuten palvelimista, tallennuksesta ja verkostoista. Tämä puolestaan mahdollistaa resurssien tehokkaamman käytön, helpomman hallinnan ja lisääntyneen joustavuuden resurssien jakamisessa kysynnän perusteella.

Pilvipalvelut tarjoavat erilaisia käyttöönottomalleja erilaisten tarpeiden tyydyttämiseksi. Julkiset pilvet ovat kolmannen osapuolen omistamia ja operoimia, jolloin resurssit voidaan saattaa yleisön saataville. Yksityiset pilvet ovat puolestaan vain yhden organisaation käytössä, ja ne tarjoavat enemmän hallintaa ja mukauttamismahdollisuuksia. Hybridipilvet yhdistävät sekä julkisten että yksityisten pilvien elementtejä tarjoten korkeamman joustavuuden ja hallinnan tason. Turvallisuus

ja yksityisyys ovat keskeisiä huolenaiheita pilvipalveluiden tarjoamisessa. Pilvipalveluntarjoajat toteuttavat vahvoja turvatoimia, kuten tietojen salausta, pääsyoikeuksien hallintaa ja säännöllisiä tarkastuksia, suojellakseen käyttäjätietoja. On tärkeää, että käyttäjät ymmärtävät palveluntarjoajansa turvallisuuskäytännöt ja toteuttavat lisätoimenpiteitä, kuten arkaluontoisten tietojen salaamista ja vahvojen pääsyoikeuksien käyttöä, jotta voidaan taata yksityisyys ja turvallisuus heidän palveluilleen.

Tärkeä näkökulma pilvipalveluihin liittyen on palvelutasosopimukset (Service Level Agreements – SLAs). SLA määrittelee pilvipalveluntarjoajan tarjoaman palvelun ehdot ja edellytykset. Ne selventävät suorituskyvyn mittareita, käytettävyyttä ja kummankin osapuolen vastuita. SLA:n ymmärtäminen ja neuvottelu on ratkaisevaa sen varmistamisessa, että pilvipalvelut vastaavat käyttäjän liiketoiminnan vaatimuksia ja odotuksia.

Alla on esitetty keskeiset käsitteet pilvipalvelusta, joita käytetään tässä opinnäytetyössä esitetyssä ohjelmistoprojektissa:

1. **Pay-as-you-go (PAYG) tai kulutuspohjainen hinnoittelumalli**: Tämä malli varmistaa, että käyttäjiä laskutetaan todellisten käytettyjen laskentaresurssien perusteella. Se helpottaa kustannusten optimointia ja lisää joustavuutta resurssien käytössä sovittaen kustannukset yhteen todellisen käytön kanssa.
2. **Palvelimeton laskenta (serverless computing)**: Tämä lähestymistapa mahdollistaa kehittäjille keskittymisen koodin kirjoittamiseen ilman taustainfrastruktuurin hallintaa. Se mahdollistaa automaattisen skaalautumisen ja vähentää operatiivista monimutkaisuutta, sillä pilvipalveluntarjoaja hoitaa palvelinhallinnan mahdollistaen tehokkaan ja skaalautuvan sovelluskehityksen.
3. **Käytettävyysalueet (availability zones)**: Palveluntarjoajat operoivat pilvipalveluita loogisesti eristetyissä paikoissa julkisten pilvien alueilla. Nämä paikat, joita kutsutaan käytettävyysalueiksi, koostuvat tyypillisesti kahdesta tai useammasta toisiinsa yhteydessä olevasta, saatavilla olevasta fyysisestä datakeskuksesta. Pilviresursseja voidaan replikoida useille käytettävyysalueille redundanssia varten ja suojautumiseksi katkoksia vastaan.

Kaikkia edellä mainittuja ominaisuuksia hyödynnetään tämän opinnäytetyön kuvatussa ohjelmistoprojektissa.

3.2 Pilviteknologian markkinoiden suurimmat toimijat ja Azuren valinnan perustelu tähän työhön

Pilvipalveluiden kenttää hallitsevat kolme suurta tarjoajaa: Amazon Web Services (AWS), Microsoft Azure ja Google Cloud Platform (GCP). Ne yhdessä kattavat 66 % vuoden 2022

kokonaispilvimarkkinoista (Griffinths 2023). Jäljelle jäävät julkisten pilviteknologioiden markkinat jakautuu IBM:n, Alibaban, Oraclen ja useiden pienempien toimijoiden kesken.

Taulukko 1. Lyhyt vertailu kolmesta suurimmasta pilvipalveluntarjoajasta (Posey 2022 ja Griffinths 2023 mukaan).

	Markkinaosuus Q1 2023, %	Alueet	Käytettävyysalueet
AWS	32	27	87
MS Azure	23	60 (mukaan lukien 27 aluetta, jotka tukevat useita käytettävyysalueita)	116
GCP	10	34	103

AWS:llä on lähes neljän vuoden ajallinen etumatka, ja tällä hetkellä se tarjoaa merkittävästi laajemman valikoiman palveluita verrattuna alan kilpailijoihin. Microsoftilla on kuitenkin pitkä historia ohjelmistojen tarjoamisessa yritysasiakkaille, ja se on usein suurten yritysten suosikkivalinta (Pletcher 2023). Myös tekijän työnantajaorganisaatio Tietoevry ylläpitää jatkuvaa strategista kumppanuutta Microsoftin kanssa ja pyrkii kehittämään henkilöstöä Microsoft Azuren alalla (Tietoevry 2020). Näin ollen oppimisprojektissa ei suoritettu nimenomaista valintaprosessia pilvipalveluntarjoajan osalta, vaan sen sijaan Azure oli oletusvalinta.

4 Agile-lähestymistapa ja CI/CD-käytännöt nykyisen projektin kontekstissa

Tietojenkäsittelyn kehitysprojekteissa on kaksi yleistä lähestymistapaa projektinhallintaan: ketterä (Agile) ja vesiputous (Waterfall). Hoory ja Bottorff (2022) kuvaavat, että vesiputous on perinteisempi metodologia, joka on lineaarinen luonteeltaan ja jossa eteneminen on jaettu peräkkäisiin vaiheisiin ja jossa on siten vain vähän joustavuutta. Suunnitellut tulokset määritetään alussa, ja eteneminen seuraavaan vaiheeseen edellyttää edellisen vaiheen toimitusten suorittamista. Ketterä lähestymistapa puolestaan on joustava ja iteratiivinen lähestymistapa, jossa määräajat ovat lyhyempiä ja palautteeseen reagoidaan nopeasti. Sen avulla muutoksia voidaan käsitellä myös myöhäisessä vaiheessa, ja ketterä lähestymistapa kannustaa tiimin aloitteellisuuteen. Keskeistä on, että ketterä lähestymistapa edistää jatkuvaa parantamista koko kehitysprosessin ajan.

Tekijän henkilökohtainen kokemus on ollut yksinomaan ketterän lähestymistavan viitekehyksessä niin osana opintoja Haaga-Helia AMK:ssa kuin osana päivittäistä työtä. Näin ollen tämä lähestymistapa on tullut luonnolliseksi ja intuitiiviseksi valinnaksi, ja tässä osiossa annetaan syvällisempi selostus ketterästä lähestymistavasta ja siihen liittyvistä käytännöistä.

Organisaation näkökulmasta työ ketterän lähestymistavan mukaan järjestetään iteratiivisina jaksoina, ns. sprintteinä, joissa monitoiminnalliset tiimit yhdessä käsittelevät backlogista tulevia tehtäviä. Backlog on dynaaminen lista priorisoituja työkohteita. Päivittäisissä palavereissa (Daily) tiimi käsittelee haasteita ja linjaa tavoitteita, ohjaa sprinttien suunnittelua ja pohtii projektin tarpeita. Säännölliset retrospektiivit mahdollistavat tiimille prosessien reflektion ja tehokkuuden jatkuvan parantamisen (The 2020 Scrum Guide). Tämä iteratiivinen sykli, jolla on lyhyitä kehitysvaiheita, varmistaa herkkyyden muuttuville vaatimuksille, nopealle päätöksenteolle ja maksimoi yhteisen asiantuntemuksen virtaviivaistaen asiakaskeskeistä ohjelmistokehitystä.

Ketterä metodologia ei ole erillinen ilmiö, vaan se on osa laajempaa ekosysteemiä, johon kuuluvat DevOps ja jatkuva integrointi/jatkuva tuotanto (Continuous Integration/Continuous Deployment – CI/CD). Bassin ja kumppaneiden (2021) mukaan DevOps on liikehdintä ja joukko käytäntöjä, jonka tavoitteena on lyhentää kehittäjän toteuttamien koodimuutosten ja loppukäyttäjän muutosten käyttöönoton välillä kuluvaa aikaa, kun samalla varmistetaan ohjelmistokehityksen korkea laatu.

Syvempää ja havainnollisempaa selitystä DevOps-työtavan hyödyistä verrattuna perinteiseen ohjelmiston kehittämiseen ja ylläpitoon voi löytää Gene Kimin kirjoista. Hän on arvostettu IT-kirjailija, joka on tunnettu vaikuttavista teoksistaan, kuten "The Phoenix Project" ja "The DevOps Handbook", jotka hän on kirjoittanut yhdessä muiden kirjoittajien (Behr K., Spafford G., Humble, J., Debois, P., Willis, J.) kanssa. Nämä kirjat ovat vaikuttaneet syvällisesti alan käsitykseen DevOps-periaatteista ja käytännöistä (Kim G., Behr K., Spafford G. 2018 ja Kim, G., Humble, J., Debois, P.,

Willis, J. 2016). Näihin etuihin kuuluvat nopeampi ja luotettavampi ohjelmiston toimitus, parantunut yhteistyö ja kommunikaatio, tuotteen laadun ja vakauden parantuminen sekä kustannustehokkuuden kasvu.

Jatkuva integrointi/jatkuva tuotanto (CI/CD) on toinen modernin ohjelmistokehityksen kulmakivi. CI sisältää koodimuutosten usein tapahtuvan integroinnin jaettuun varastoon antaen tiimille mahdollisuuden havaita ja käsitellä ongelmia varhain kehitysprosessissa. CD laajentaa tätä automatisoimalla koodin käyttöönoton tuotantoympäristöön varmistaen nopean ja luotettavan julkaisusyklin. Ketterän lähestymistavan iteratiivinen luonne resonoi CI/CD:n jatkuvan näkökulman kanssa vahvistaen sitoutumista toimivan ohjelmiston toimittamiseen pienissä, inkrementaalisissa askelissa.

Ketterän lähestymistavan, DevOpsin ja CI/CD:n vuorovaikutus on ratkaiseva saumattoman ja tehokkaan kehitysputken saavuttamiseksi. Ketterä lähestymistapa tarjoaa iteratiivisen kehyksen, DevOps edistää yhteistyötä ja automaatiota ja CI/CD varmistaa jatkuvan ja luotettavan toimitusprosessin. Yhdessä ne muodostavat ekosysteemin, joka mahdollistaa kehitystiimien reagoinnin muuttuviin vaatimuksiin, ohjelmiston toimittamisen aiempaa nopeammassa aikataulussa ja korkean laadun ylläpitämisen koko kehityssyklin ajan.

5 Johdanto empiirisen projektin toteuttamiseen

Edellisissä luvuissa käsiteltiin teoreettisia käsitteitä, jotka liittyvät projektin kohteena olevan sovelluksenkehittämiseen. Seuraavaksi opinnäytetyössä esitetään yksityiskohtainen kuvaus päätöksentekoprosessista sekä sovelluksen komponenttien kehittämisestä ja käyttöönotosta, jotka muodostavat tutkielman empiirisen perustan. Ennen tätä, luvuissa 5.1 ja 5.2 tarkastellaan projektin aiheen valintaa ja perustellaan projektinhallintakäytäntöjä, jotka otettiin käyttöön kehitystyön aikana. Seuraavien osien tavoitteena on antaa parempi käsitys empiirisen projektin taustasta ja asetelmasta.

5.1 Empiirisen tutkimusidean alkuperä

Kuten luvussa 2.1 mainittiin, työssään tekijä oli päässyt mukaan projektiin, joka keskittyi pilviteknologioihin, erityisesti Microsoft Azureen. Tekijälle oli varattu ajanjakso uuden toimialan syventämiseen ja tarvittavien taitojen hankkimiseen. Tekijä suoritti useita teoreettisia Azure-kursseja: Microsoftin oppimispolkuja Microsoft Learn -portaalissa (https://learn.microsoft.com/en-us/training/browse/), Azure Fundamentals -kurssin Pluralsight -portaalissa (https://www.pluralsight.com/) sekä AZ-900 Microsoftin ohjaajan vetämää koulutusta. Oppimisen seuraava vaihe sisälsi Proof-of-Concept (PoC) -sovelluksen luomisen. Tämän PoC:n tarkoituksena oli oppia Azure-ympäristöä ja tutkia Azure-resurssien hyödyntämistä eri verkkosovelluksen komponenttien käyttöönottoon. Idea kokeelliselle sovellukselle saatiin testitehtävästä junioriohjelmistokehittäjän paikan yhteydessä eräässä IT-konsulttiyrityksessä. Tämä tehtävä löytyi GitHubista, linkki ei ole enää saatavilla hakuajan päättymisen vuoksi.

Tehtävän mukaan sovelluksen piti hyödyntää suurta datan joukkoa, joka käsitteli Helsingin kaupunkipyörillä tehtyjä pyöräilyjä ja tietoja kaupungin pyöräasemista. Käyttöliittymän piti pystyä näyttämään luettelo kaikista pyöräilyistä, kaikkien asemien luettelon ja yksityiskohtaiset tiedot yksittäisistä asemista, mukaan lukien tilastolliset laskelmat koskien pyöräilyjä, jotka alkavat tai päättyvät tietyllä asemalla, sekä aseman sijainti kartalla. Teknologioiden tarkka valinta ja tämän tehtävän suorittamistapa oli testitehtävän tekijän päätettävissä. Päätöksenä oli rakentaa pilvipohjainen sovellus, joka noudatti määriteltyjä vaatimuksia.

Seuraavassa alaluvussa esitellään kattava tarkastelu sovelluksesta sekä sen oleellisista osista, mukaan lukien arkkitehtoninen suunnittelu, tiettyjen ratkaisujen tutkimus ja valinta, toteutusmenetelmät ja testausprosessit. Kuitenkin ennen sitä esitellään lyhyt yleiskatsaus projektin aikana käytetyistä työmenetelmistä.

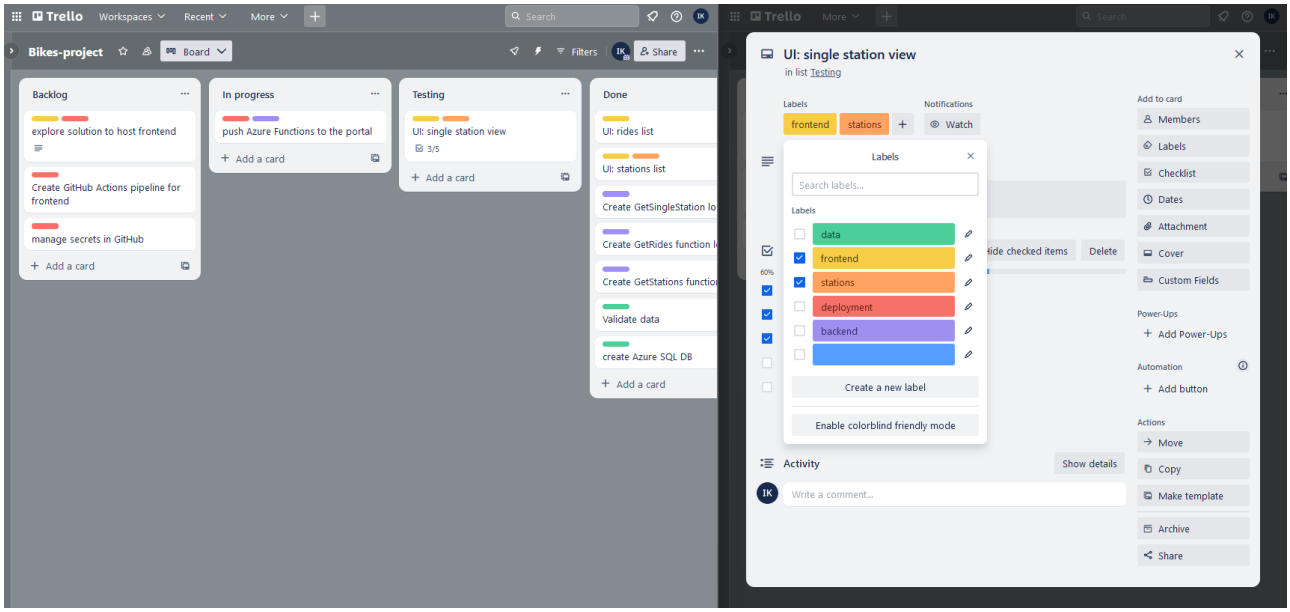
5.2 Projektityön suunnittelu

Suunnitelmana oli suorittaa sovellustyö noudattaen luvussa 4 esitettyjä periaatteita sekä ketterää kehitystapaa. Tässä yhteydessä joustavuus tarkoittaa kykyä tutkia mahdollisia ratkaisuja kehitysprosessin aikana, mikä poistaa tarpeen jäykälle, etukäteen tehtävälle projektisuunnitelmalle. Huolimatta itsenäisestä työskentelystä, tekijä ylläpiti ominaisuuksien ja ongelmien backlogia. Myös versionhallinnan selkeä ja tarkka hyödyntäminen oli yksi tärkeimmistä käytännöistä. Projektikehyksen yksityiskohtaisempi selitys annetaan seuraavissa osioissa 5.2.1 ja 5.2.2.

5.2.1 Agile-lähestymistapa ja Trello-taulun käyttö projektin edistymisen seurantaan ja hallintaan

Suunnitellessaan työtä projektissa päätettiin ottaa käyttöön ketterä lähestymistapa, jota käytettiin laajalti tekijän työpaikalla, vaikka tässä tapauksessa tekijä työskenteli yksin ilman tiimiä. Ketterän lähestymistavan ydinarvot, kuten joustavuus, sopeutumiskyky ja iteratiivinen eteneminen (Agile-manifesto, 2001), osoittautuivat arvokkaiksi projektin tavoitteiden saavuttamisessa. Tehtävien backlogin ylläpitäminen auttoi suunnittelemaan työn loogisesti ja priorisoimaan tärkeimmät ja suurimman vaikutuksen omaavat tehtävät samalla pysyen sopeutumiskykyisenä tutkimuksen ja kehityksen aikana saatujen uusien näkökulmien suhteen.

Käytännön toteutuskeinona käytettiin Trelloa, verkkopohjaista projektinhallintatyökalua. Vaikka sitä käytettiin suhteellisen laajasti, se paransi merkittävästi tekijän käsitystä projektin edistymisestä ja auttoi tunnistamaan mahdolliset esteet. Trello oli myös ratkaisevan tärkeä hyvin järjestetyn tehtäväluettelon ylläpitämisessä, tarkoittaen sitä, ettei tärkeitä tehtäviä jäänyt huomaamatta tai unohdetuiksi. Selkeän tunnistejärjestelmän käyttö osoittautui erityisen hyödylliseksi tässä yhteydessä. Se mahdollisti samanaikaisen työn sovelluksen eri osien parissa ja antoi tekijälle mahdollisuuden seurata ja hallita muutoksia ja uusia lisäyksiä kuhunkin osaan tehokkaasti. Kuva 2 esittää yleiskuvan Trello-taulusta ja yhden tehtävän näkymän joidenkin tunnisteiden valinnalla.

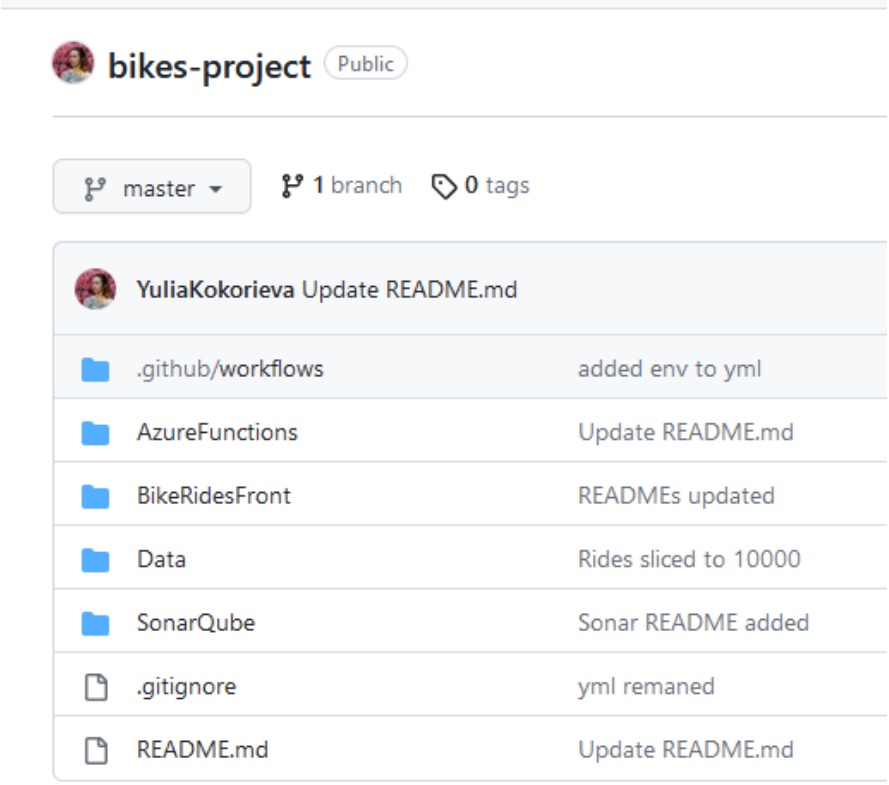


Kuva 2. Trello-taulukon kuvakaappaus sovelluskehitysprojektin seurantaa varten (tekijä: Iuliia Kokorieva)

5.2.2 Versiohallinnan käyttö

Toinen lähestymistapa, joka yleensä otetaan käyttöön tiimityössä, mutta joka otettiin tehokkaasti käyttöön myös tekijän yksin tekemässä projektissa, on versionhallinta. Sovelluskehityksen tarpeisiin luotiin GitHubin etätietovarasto (repository). Ottaen huomioon, että projekti koostuu kolmesta erillisestä osasta, tekijä päätti, että näiden osien koodipohjien ylläpitäminen yhdessä tietovarastossa, vaikkakin erillisissä hakemistoissa, olisi kaikkein järkevin lähestymistapa. Näin ollen perustettiin kolme erillistä kansioita näiden osien koodipohjille:

- `Data` – datan validointiin tarkoitetulle Python-skriptille.
- `AzureFunctions` – taustajärjestelmälle (backend).
- `BikeRidesFront` – käyttöliittymälle (frontend).



Kuva 3. Kuvakaappaus projektin etäisestä GitHub-varaston juurikansion sisällöstä (tekijä: Iuliia Kokorieva)

Kuvasta 3 voidaan nähdä, että projektin edetessä lisäresursseja luotiin kolmen osan koodipohjien lisäksi:

- `.github/workflows` -kansio GitHub Actions -putkiston `.yml`-tiedoston säilyttämiseen.
- `SonarQube` -kansio, joka sisältää `docker-compose.yml`-tiedoston SonarQube-kontin suorittamiseen paikallisesti.
- Perinteisesti pää-`README.md`, joka sisältää projektin kuvauksen, joka on nähtävissä tietovaraston pääsivulla, sekä `.gitignore`-tiedosto, jossa on luettelo tiedostoista ja kansioista, joita ei tule sisällyttää versionhallintaan ja jotka näin ollen eivät siirry etävarastoon.

Työskennellessään etävaraston parissa tekijä noudatti sitoutumissuosituksia, mukaan lukien usein tehtyjä sitoumuksia, yhden asiaan liittyvän muutoksen per sitoumus -periaatetta ja selkeitä ja informatiivisia sitoutumisviestejä (Git Commit Best Practices). Kuitenkin yksi huomattava asia, joka jätettiin huomiotta ja siten edustaa menetettyä mahdollisuutta, on erillisten haarojen ja vetopyyntöjen hyödyntäminen.

Vaikka se ei ole välttämätöntä yksin työskennellessä, se voisi olla hyvä mahdollisuus harjoitella tiimiprojektityössä hyödyllisiä taitoja. Lisäksi se olisi voinut helpottaa GitHubin projektien käyttöä Trellon sijaan tarjoten etuna mahdollisuuden linkittää tiettyjä vetopyyntöjä yksittäisiin ongelmiin tai tehtäviin. Tämä mahdollistaisi selkeät ja jäljitettävät yhteydet ongelmien ja niiden vastaavien ratkaisujen välillä.

Etäinen GitHub-varaston linkki on seuraava: https://github.com/YuliaKokorieva/bikes-project, ja projektin työnimi on `”Bikes-project”`.

6 Fullstack-sovelluksen pilvipalveluun käyttöönoton toteutus

Kuten on kuvattu luvussa 5, tekijän lähestymistapa sovelluksen kehittämiseen oli varsin joustava. Tämä mahdollisti jatkuvan ratkaisujen tutkimuksen olennaisena osana kehitysprosessia. Tämä lähestymistapa oli erityisen hyödyllinen projektin oppimislähtöisen luonteen vuoksi. Vaikka tekijällä oli joitakin tietoja Azure Cloud -ympäristöstä ja resursseista, hän näki sovelluksen kehittämisen ensisijaisesti mahdollisuutena laajentaa omaa osaamistaan. Näin ollen kehitystyön aikana ensimmäisessä vaiheessa kiinnitettiin paljon huomiota ja aikaa kunkin sovelluksen osan kannalta sopivimman ratkaisun valintaan. Työn toinen näkökohta oli oppia käyttämään valittua resurssia ja luoda oikea ja turvallinen integraatio komponenttien välille. Molemmat edellä mainitut näkökohdat käsitellään tämän luvun vastaavissa osioissa.

6.1 Projektin pilviarkkitehtuurin suunnittelu

Projektin toteutuksessa käytettyjen lähestymistapojen kuvauksen jälkeen tämän raportin seuraava osuus siirtyy projektin toteutusvaiheen määrittelyyn.

Ennen toteutusvaiheen aloittamista on tarpeen laatia projektille arkkitehtuurisuunnitelma. Kuten yllä mainittiin, testitehtävä ei sisällä erityisiä vaatimuksia työkaluista tai valittavista lähestymistavoista. Ottaen huomioon projektin suuren datajoukon, laajan tilastollisen laskennan tähän dataan ja pitkän luettelon käyttöliittymään toteutettavista toiminnoista, tekijän luonnollinen valinta oli mikropalveluarkkitehtuuri.

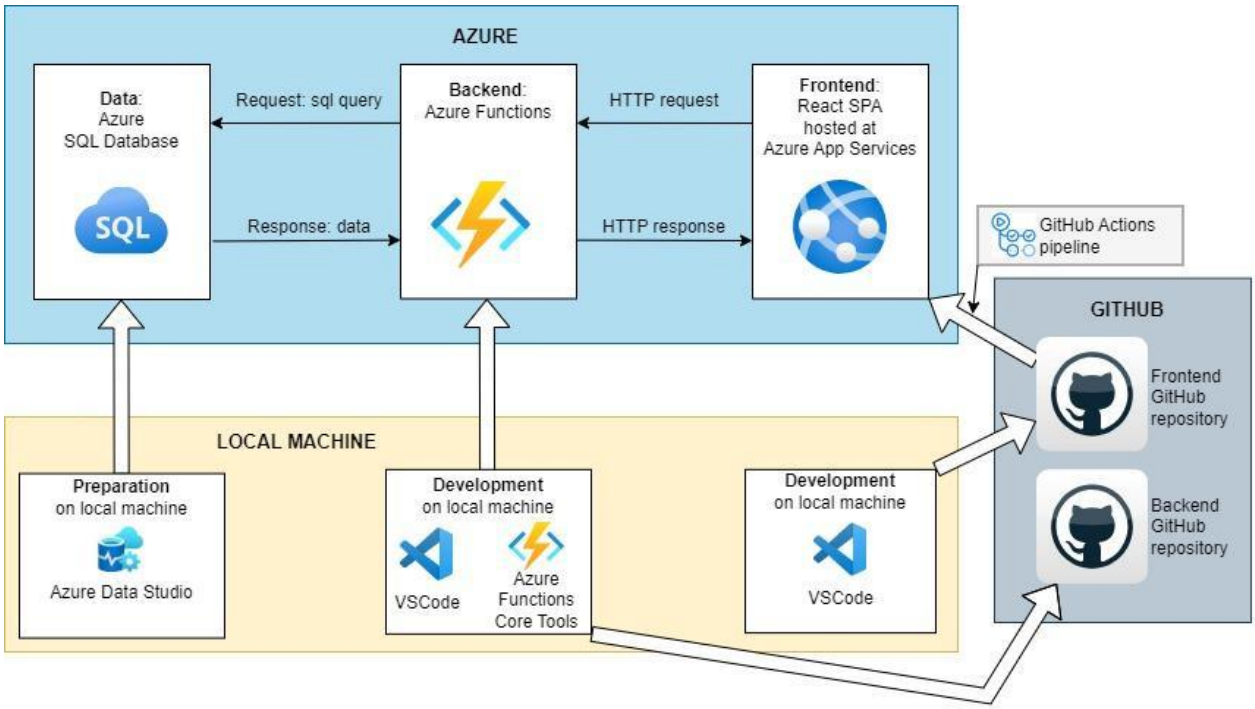
Bassin ja kumppaneiden mukaan (2021, luku 5.2), ”mikropalveluarkkitehtuurin kuvio rakentaa järjestelmän joukoksi itsenäisesti käyttöönotettavia palveluita, jotka kommunikoivat vain viestien välityksellä palveluiden rajapintojen kautta”. Tarkemmin sanottuna tietokanta palvelisi vain nimettyä backendia, eikä näillä osilla olisi suoraa pääsyä toisiinsa. Niiden vuorovaikutus olisi yksinomaan viestipohjaista viestintää.

Niinpä tekijä suunnitteli luoda erilliset käyttöönotot kolmelle osalle: tietokanta, backend ja frontend. Kaikkien komponenttien yksityiskohtainen kuvaus ja valitun käyttöönottostrategian perustelu esitetään seuraavissa luvuissa. Korkealla tasolla arkkitehtuuri kuvataan kuvassa 4. Komponentit ovat seuraavat:

- Data: Relaatiotietokanta SQL, kehitetty paikallisesti Azure Data Studiota käyttäen, ja se on otettu käyttöön samalla työkalulla Azure SQL -tietokantapalveluun.

- Backend: Azure Functions -palvelimeton ratkaisu, kehitetty paikallisesti VSCodea käyttäen tarvittavilla Azure-laajennuksilla ja Azure Functions Core Tools -työkalulla. Se on jatkuvasti otettu käyttöön Function App -palveluun CI/CD-käytäntöjen mukaisesti, ja versionhallinta hoidetaan GitHubin kautta.

- Frontend: React-yksisivuinen sovellus (single-page application, SPA), kehitetty paikallisesti käyttäen VSCodea ja otettu käyttöön Azure App Service -palveluna käyttäen GitHub Actions -putkistoa.



Kuva 4. Korkean tason opinnäytetyön projektin arkkitehtuurikaavio (tekijä: Iuliia Kokorieva).

Seuraavissa luvuissa esitetään kehityksen ja käyttöönottoprosessien yksityiskohtaiset menettelytavat sekä kattava yleiskatsaus Azure-keskus- ja tukiresursseista, jotka luotiin projektin osiin liittyen.

6.2 Ratkaisun tutkimus ja valinta relaatiotietokannan isännöimiseksi Azuren ympäristössä

Erityisvaatimusten mukaisesti sovelluksen tulee käyttää tietojoukkoa, joka koostuu kahdesta toisiinsa liittyvästä taulukosta: yksi sisältää kaupunkipolkupyöräasemien luettelon ja toinen dokumentoi pyöräilyt. Jälkimmäinen sisältää tietoja pyöräasemista, joissa kunkin pyöräilyn alku ja

loppu tapahtuivat. Tämä taulukkojen välinen suhde korostaa, että tietokannan tulee olla relaatiotietokanta, jossa on asianmukaiset vierasavaimet kuvaamaan yhteyksiä taulukoiden välillä.

6.2.1 Azure SQL -tietokannan valinnan perustelut

Azure Cloud tarjoaa vaihtoehtoja sekä relaatiotietokantojen että ei-relaatiotietokantojen kanssa työskentelyyn erilaisilla tasoilla taustalla olevan infrastruktuurin hallitsemiseen (IaaS vs. PaaS-tarjonta). Ohjelmistokehittäjä Marc Heath, Microsoftin MVP (Most Valuable Professional – arvostetuin asiantuntija) ja Pluralsight-kurssien kirjoittaja Azuresta, mainitsee ajatuksen, että työskennellessä palvelimettoman sovelluksen parissa kehittäjä mieluummin välttäisi oman palvelimen hallintaa. Tästä syystä IaaS-lähestymistapa ja tietokannan manuaalinen asentaminen virtuaalikoneeseen eivät olisi looginen valinta (Heath 18. kesäkuuta 2020).

Hän esittää myös seuraavat PaaS-vaihtoehdot, kun kyse on tietokannan valinnasta palvelimettomalle sovellukselle: relaatiotietokanta, kuten Azure SQL Database, dokumenttitietokanta (Azure Cosmos DB) tai varastotili edullisempana vaihtoehtona vähimmäiskustannuksille. Tietyn vaihtoehdon valinta riippuisi tietotyypistä ja suhteiden entiteettien välisestä olemassaolosta tai halutusta datamallin joustavuudesta. Muita harkittavia tekijöitä ovat halukkuus hyödyntää olemassa olevia Azure Functions -sitomisia, maailmanlaajuinen jakelu ja skaalautuvuus sekä kustannukset (Bhatt 14.04.2023, Microsoft Corporation 21.07.2023, Heath 18.06.2020).

Taulukko 2 esittää projektin vaatimukset koskien tietokantaa ja datan ominaisuuksia, jotka osoittautuivat merkityksellisiksi käyttöönotto-optiosta päätettäessä, sekä sopivimman ratkaisun vastaukseksi näihin ongelmiin.

Taulukko 2. Ongelmat ja ehdotetut ratkaisut päätöksenteossa tietokannan isännöinnistä Azure-palvelussa.

	Ongelma	**Ehdotettu ratkaisu**
1.	Data on rakenteellista ja vaatii relaatiokyselyjä yhdistääkseen tiedot kahdesta eri taulusta.	Relaatiotietokanta.
2.	Ei tarvita käyttöjärjestelmän tai tietokannan moottorin hallintaa.	PaaS-tarjonta IaaS:n sijasta.
3.	Koska tämä ei ole tuotantosovellus, ei ole vaatimuksia pienen viiveen tai skaalautuvuuden suhteen.	Mikä tahansa.

4.	Koska tämä on kokeellinen sovellus, sen tulisi olla kustannustehokas.	Azure SQL on kustannustehokkaampi kuin Cosmos DB. (Bhatt 14.06.2023)

Kun kaikki keskeiset ongelmat on käyty läpi ja datan ominaisuudet on otettu huomioon, tekijä valitsi Azure SQL Database -palvelun datan käyttöönottoon.

6.2.2 Tietokannan kehittäminen ja käyttöönotto Azure-ympäristössä

Valmistautuminen tietokannan luomiseen sisälsi datan lataamisen ja analysoinnin sekä tiettyjen validointien suorittamisen Python-skriptien avulla. Nämä asiat eivät kuitenkaan sisälly tämän raportin laajuuteen, koska se ei liity suoraan opinnäytetyön aiheeseen, eli resurssien valintaan ja toteutukseen Azure-pilvipalvelussa.

Kehitys ja tietokannan käyttöönotto toteutettiin käyttäen Azure Data Studio -työkalua. Tietokannan luominen ja käyttöönotto Azure Data Studio -työkalua käyttäen sisälsi seuraavat vaiheet:

1. Tietokantapalvelimen luominen Azure-portaalissa.
2. Palvelinyhteyden luominen Azure Data Studio -sovelluksessa.
3. Datan tuonti tietokantaan (ja tarvittaessa muokkaukset).

Dokumentaation mukaan (Microsoft Corporation 18.10.2022a) Azure-resurssien luominen on mahdollista eri tavoin:

- Azure-portaalin kautta.
- Skriptin avulla käyttäen Azure CLI- tai Azure PowerShell -työkaluja.
- Infrastruktuuri koodina (Infrastracture as Code, IaC) -työkalujen avulla.

Tämän projektin tarkoituksiin valittiin manuaalinen luonti portaalissa, sillä se on vähiten monimutkainen reitti.

SQL-tietokannan luomisen yhteydessä on tarpeen valita joko olemassa oleva SQL-palvelin tai luoda uusi. SQL-palvelin on looginen palvelin, joka isännöi yhtä tai useampaa SQL-tietokantaa. Tämä palvelin toimii hallinta- ja todennuskerroksena niille tietokannoille, joita se isännöi. Ennen resurssien luomista on hyvä arvioida mahdollisia kokoonpanovaihtoehtoja:

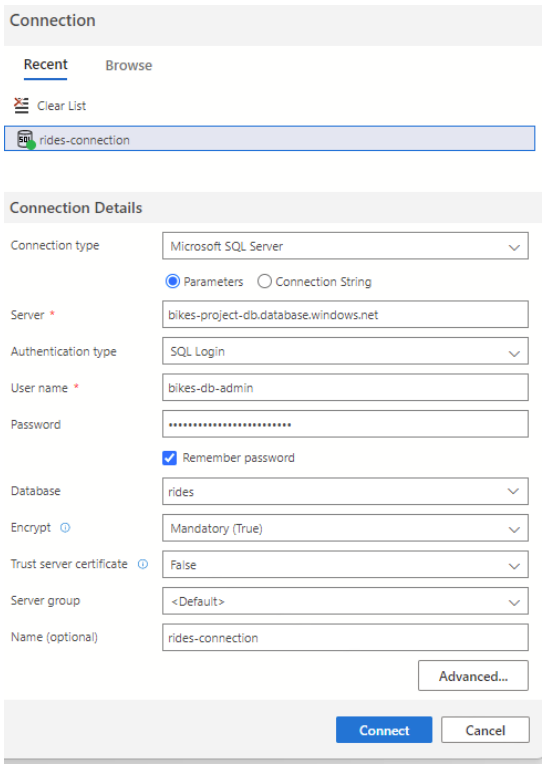
1. Todennusmenetelmä uudelle SQL-palvelimelle.

2. Elastinen allas (elastic pool) -resurssienhallintomahdollisuus, joka mahdollistaa useiden tietokantojen jakamisen ja resurssien, mukaan lukien CPU:n ja muistin, yhdistämisen yhdellä tietokantapalvelimella.

3. Varmuuskopioiden tallennuksen redundanssi -vaihtoehto tietojen suojaamiseksi varmuuskopioiden monistamalla niitä maantieteellisesti monipuolisissa Azure-datakeskuksissa.

4. Suorituskykykerros ja taso.

Tässä projektissa luotiin palvelin nimeltä "`bikes-project-db`", SQL-todennusmenetelmä (SQL-kirjautumistunnus ja -salasana), yksi tietokanta nimeltä "`rides`", vyöhykeriippumaton varmuuskopioiden tallennus ja ilman elastista allasta. Azure Data Studio -sovelluksessa muodostettiin "`rides-connection`"-niminen yhteys etä-SQL-palvelimeen, ja tarvittavat taulut luotiin (katso kuva 5).

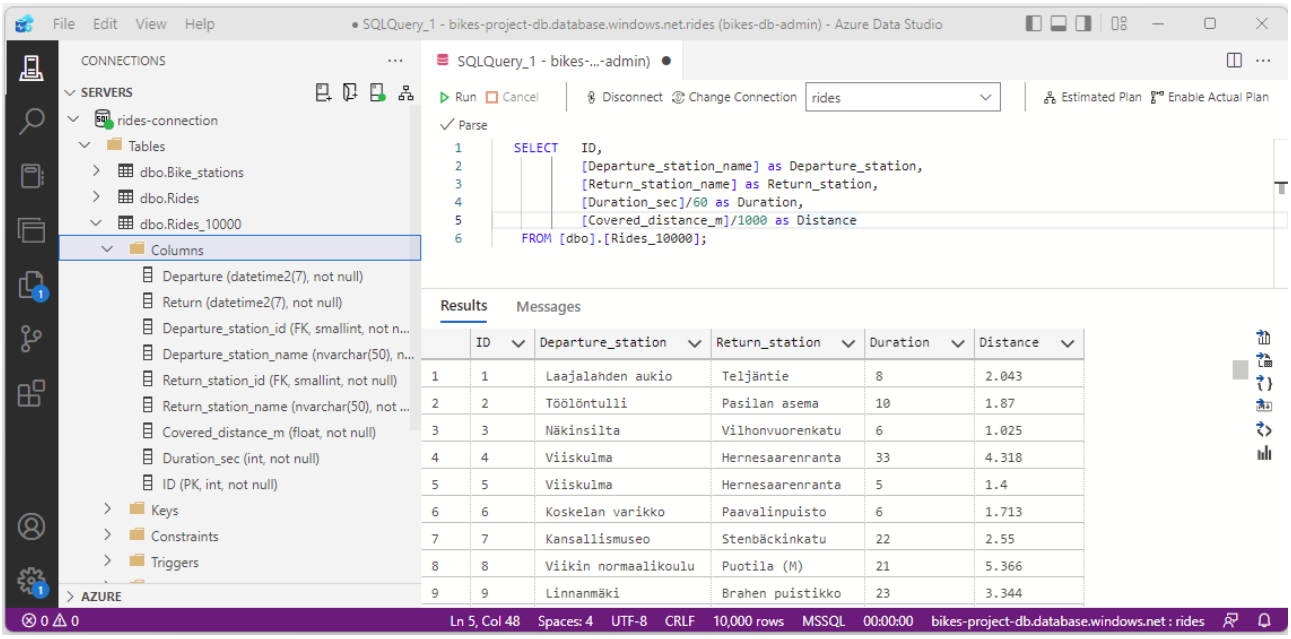


Kuva 5. Azure Data Studio -sovelluksen kuvakaappaus Azure SQL Serverin yhteydestä (tekijä: Iuliia Kokorieva).

Jotta voitaisiin suorittaa datan tuonti `.csv`-tiedostoista, asennettiin tarvittava SQL Server Import -laajennus. Tämän laajennuksen Tuontiavustajalla (Import Wizard) on mahdollista tuoda data

automaattisesti tietokantaan eri tiedostotyypeistä, luoda esikatselu ja muokata sarakkeita ennen varsinaista tuontia.

Azure Data Studio -työkalua voidaan lisäksi käyttää datan manuaaliseen tutkimiseen ja tietokantakyselyjen tekemiseen. Tekijä käytti sitä testatakseen kyselyjä, jotka myöhemmin otettiin käyttöön Azure Functions -sovelluksessa tarvittavan tiedon noutamiseen. Esimerkki on annettu alla olevassa kuvassa 6.



Kuva 6. Azure Cloud Studio -työtilan kuvakaappaus, jossa tietokantaselain, esimerkkilauseke SQL-kyselylle ja sen tulokset (tekijä: Iuliia Kokorieva).

6.3 Ratkaisun tutkimus ja käyttöönotto backendin isännöimiseksi Azuren ympäristössä

Backend-työskentelyn aikana koodia ei kirjoitettu, ennen kuin käyttöönottoratkaisu oli valittu. Tämä johtuu siitä, että tekijän aikomuksena oli tutkia perusteellisesti ja mahdollisesti hyödyntää Azuren Function App -ratkaisua. Se on palvelimeton ratkaisu ja edellyttää tiettyjen mallien ja tietyn syntaksin käyttöä koodin kirjoittamisen yhteydessä. Ainoa edellytys, jonka tekijä asetti, oli hänen mieltymyksensä Node.js-ympäristön käyttämiseen, ja Azure Functions tarjoaa tällaisen mahdollisuuden. Tämä ei kuitenkaan ollut lopullinen valinta valmisteluvaiheessa, ja lisätutkimuksia saatavilla olevista ratkaisuvaihtoehdoista oli suoritettava.

6.3.1 Perustelut Azure Functions -palvelun valinnalle Azure-resurssina backendille

Azure-dokumentaatio (Microsoft Corporation 18.10.2022b) tarjoaa seuraavan luettelon vaihtoehdoiksi sovelluksen isännöimiseen Azure-ympäristössä:

- Azure App Service (PaaS): täysin hallittu alusta verkkosovellusten, rajapintojen ja mobiilisovellusten rakentamiseen, käyttöönottamiseen ja skaalaamiseen.
- Static Web Apps (PaaS, Serverless): staattisten verkkosovellusten palvelu, joka on ideaali staatisen frontendin käyttöönottamiseen.
- Azure Functions (PaaS, Serverless): tapahtumavetoinen, palvelimeton laskupalvelu koodin suorittamiseen tapahtumien reagointiin.
- Azure Spring Apps (PaaS): hallittu palvelu Spring-pohjaisten Java-sovellusten käyttöönottamiseen ja skaalaamiseen Azure-ympäristössä.
- Azure Kubernetes Service (PaaS): hallittu konttien orkestrointipalvelu sovellusten käyttöönottoon, hallintaan ja skaalaamiseen Kubernetes-alustan avulla.
- Azure Container Instances (PaaS, Serverless): palvelu konttien suorittamiseen ilman taustainfrastruktuurin hallintaa, sopii lyhytaikaisiin tehtäviin.
- Azure Virtual Machines (IaaS): palvelu, joka tarjoaa pyynnöstä virtuaalikoneita täyden hallinnan saamiseksi infrastruktuurista, sopii räätälöityjen työkuormien suorittamiseen.

Sovelluksen backendin isännöintiä varten tekijä haluaisi palvelun, jonka infrastruktuuri olisi mahdollisimman kevyt ja helppo hallita. Näin ollen Virtual Machines -palvelu (IaaS), joka vaatii laajaa infrastruktuurin konfigurointia ja hallintaa, ei ole paras valinta. Static Web Apps ei sovi tähän työhön, koska se on frontendin isännöinnin tarkoitettu palvelu. Koska tekijän suosima kieli koodin kirjoittamiseen on JavaScript (Node.js / Express.js -kehyksen avulla backendia varten), Java-sovelluksille tarkoitettua Azure Spring Apps -palvelua ei valita. Sovelluskonttien luominen olisi hyvä käytäntö, mutta tekijän rajallinen kokemus tästä teknologiasta tekee sen toteuttamisesta haastavaa. Tällaisten tehtävien käsittelyyn tarvittaisiin syvempää asiantuntemusta. Tämän vuoksi Azure Container Instances, Azure Kubernetes Service sekä App Services konttien isännöinnin yhteydessä jäävät nyt sivuun.

Jäljellä olevat vaihtoehdot ovat Azure App Service ja Azure Functions. Tässä vaiheessa on syytä syventyä molempien palvelujen etuihin ja haittoihin niin yleisellä tasolla kuin kehityksen nykyisen sovelluksen yhteydessä.

6.3.2 Azure Functions- ja Azure App Service -palvelun käytön vertailu nykyisessä projektissa

Kuten edellisessä osiossa mainittiin, nykyisen projektin vuoksi valittiin kaksi vaihtoehtoa backendin käyttöönottoon tarkastelua varten: Azure Functions ja Azure App Service. Seuraava keskustelu ja vertailu perustuvat tekijän osallistumiseen joulukuussa 2022 pidettyyn koulutukseen (Microsoft Corporation, joulukuu 2022).

Azure Functions -työkalu toimii siten, että pienet, tapahtumavetoiset koodinpätkät suoritetaan tiettyjen laukaisijoiden tai tapahtumien reagoidessa. Ne ilmentävät palvelimettoman laskennan käsitettä, joka mahdollistaa kehittäjän keskittymisen pelkästään koodin kirjoittamiseen ilman huolta taustalla olevasta infrastruktuurista. Tämä yksinkertaisuus voi nopeuttaa kehitystä, vähentää monimutkaisuutta ja lopulta johtaa nopeampaan kehitysaikaan. Tällainen lähestymistapa tarjoaa eduksi maksaa vain niistä laskentaresursseista, jotka kulutetaan tehtävien suorituksen aikana, minimoiden näin ylimääräiset kustannukset. Azure Functions -työkalu sopii erityisesti pieniin, määritettyihin tehtäviin ja sovelluksiin, jotka tarvitsevat dynaamista skaalausta vaihtelevien työmäärien reagointina.

Toisaalta Azure App Service tarjoaa PaaS-ympäristön verkkosovellusten, REST-rajapintojen ja mobiilipalvelinten käyttöönottoon. Se tarjoaa perinteisemmän isännöintimallin, jossa on enemmän hallintaa infrastruktuurin yli, mikä sopii suuremmille ja monimutkaisemmille projekteille. Vaikka se tarjoaa kattavan ja joustavan isännöintiympäristön, se tuo myös mukanaan korkeammat kustannukset ja mahdollisesti enemmän hallinnollisia tehtäviä.

Skaalautuvuuden osalta sekä Azure Functions että Azure App Service tarjoavat mahdollisuuksia, mutta edellinen erottuu tarjoamalla automaattisen ja tarkan skaalauksen kysynnän perusteella. Azure Functions voi reagoida välittömästi liikenteen huippuihin ja skaalautua alas, kun kuormitus vähenee. Azure App Service kykenee skaalautumaan, mutta se vaatii enemmän manuaalista määritystä ja hallintaa.

Kustannusarvioinnin osalta Azure Functions osoittautuu usein kustannustehokkaammaksi yksinkertaisille, lyhytaikaisille tehtäville. "Pay-as-you-go" ("maksa käytön mukaan") -hinnoittelumalli varmistaa, että omistajaa veloitetaan vain tehtävien suorituksen aikana käytetyistä laskentaresursseista. Toisaalta Azure App Service sisältää ennakoitavamman, mutta mahdollisesti myös korkeamman kustannusrakenteen.

Kaikki edellä esitetyt tekijät huomioon ottaen ja "helpompi on parempi" -periaatteen mukaisesti Azure Functions nousee suosituksi valinnaksi nykyisen projektin backendin käyttöönottoon, taaten kustannustehokkaan, ketterän ja vaivattoman kehitys- ja käyttöönottoprosessin.

6.3.3 Empiirinen raportti Azure Functions -sovelluksen kehittämisestä

Dokumentaation mukaan (Microsoft Corporation 14.09.2023) suositeltu tapa kehittää Azure Functions -sovelluksen on tehdä se paikallisesti ja sen jälkeen julkaista sen Azure Cloudissa Azure Functions -palveluun. Tämän lisäksi on myös mahdollista luoda ja muokata funktioita Azure-portaalissa (ainoastaan Node.js -versioiden 3 ja 4 funktioita varten).

Seuraavia työkaluja käytettiin funktioiden paikalliseen kehitykseen:

1) IDE: VSCode tarvittavilla laajennuksilla (Azure Account, Azure Functions).

2) Azure Functions Core Tools -työkalu.

3) Azure CLI asennettuna tietokoneeseen.

4) Node asennettuna tietokoneeseen (Node.js-funktioita varten).

Tarvitaan myös Azure-tilaus. Tekijällä on Azure for students -tilaus, joka tarjoaa opiskelijoille ilmaisen pääsyn monipuolisiin pilvipalveluihin ja 100 dollarin hyvityksen, jota voidaan käyttää tiettyihin palveluihin.

Seuraavat vaiheet tulisi suorittaa Azure Functions -projektin luomiseksi paikallisesti ja myöhemmin sen julkaisemiseksi pilveen.

1. Paikallisen projektin luominen.

VSCode:ssa Azure-välilehdessä Workspace-välilehden kautta Azure Functions -pudotusvalikosta tulisi luoda uusi paikallinen projekti. Luomisprosessin aikana kehotetaan kehittäjää antamaan määritystietoja, kuten kansion sijainti, ohjelmointikieli ja ohjelmointimalli, mallipohja funktiolle (suunnitellun laukaisimen perusteella) ja funktion nimi.

Kun luontiprosessi on valmis, voidaan havaita kansio, jossa on tarvittavat tiedostot ja alikansiot. Itse funktiokoodi koostuu tiedostoista `index.js` (koodi) ja `function.json` (funktion määritystiedosto), joiden lisäksi on myös tiedostoja, jotka liittyvät koko projektiin: `host.json` (kaikkia funktioita Function App -sovelluksessa koskeva määritys), `local.settings.json`

(asetukset ja yhteydet funktioiden paikalliseen suorittamiseen; ei julkaista Azureen), `package.json` (pakettiriippuvuudet, skriptit jne.).

2. Funktioiden määritteleminen paikallisesti.

Seuraava vaihe on luoda funktioiden sisältö. Tässä projektissa on luotu 3 funktiota:

1. `GetRides`.
2. `GetStations`.
3. `GetSingleStation`.

Kaikilla kolmella funktiolla on sama rakenne ja logiikka. Ne ovat HTTP-laukaisijoita, mikä tarkoittaa, että ne alkavat suorittaa, kun ne saavat HTTP-pyynnön vastaavilla loppupisteillä. Käynnistyttyään jokainen funktio lähettää SQL-kyselyn tietokantaan datan hakemiseksi, vastaanottaa vastauksen tietokannasta ja lähettää sen eteenpäin HTTP-vastauksena. Sovelluksen logiikan vuoksi funktiot eivät luo, päivitä tai poista tietokannan tietueita. Näin ollen ainoa tuettu metodi on GET (määritelty `function.json`-tiedostossa). Koko `function.json`-tiedoston sisältö (sama kaikille kolmelle funktiolle) näkyy alla olevassa kuvassa 7. Lisäksi se määrittelee sisääntulevan sitomisen, laukaisimen tyypin (HTTP), todennustason (anonyymi) ja lähtöliitännän (HTTP-vastaus).

Kuva 7. VSCode:ssa luodun `GetRides`-funktion `function.json`-tiedoston kuvakaappaus (tekijä: Iuliia Kokorieva).

Kuva 8 alla esittelee `GetRides`-funktion `index.js`-tiedoston. Muiden kahden funktion vastaavat tiedostot eroavat SQL-kyselystä. Mainittu tiedosto sisältää seuraavat loogiset osiot:

- Tarvittavien moduulien (`mssql`) tuonti: rivi 1.
- Yhteyden määritys tietokantaan käyttäen tunnistetietoja: rivit 4–18.
- SQL-kyselyn lähettäminen asynkronisesti: rivit 20–23.
- Vastauksen vastaanottaminen ja sen lähettäminen eteenpäin HTTP-vastauksena: rivit 25–27.
- Virheiden käsittely: rivit 28–32.
- Yhteyden sulkeminen tietokantaan: rivi 34.

Kuva 8. `GetRides`-funktion `index.js`-tiedoston kuvakaappaus VSCode:ssa (tekijä: Iuliia Kokorieva).

3. Funktioiden testaaminen paikallisesti.

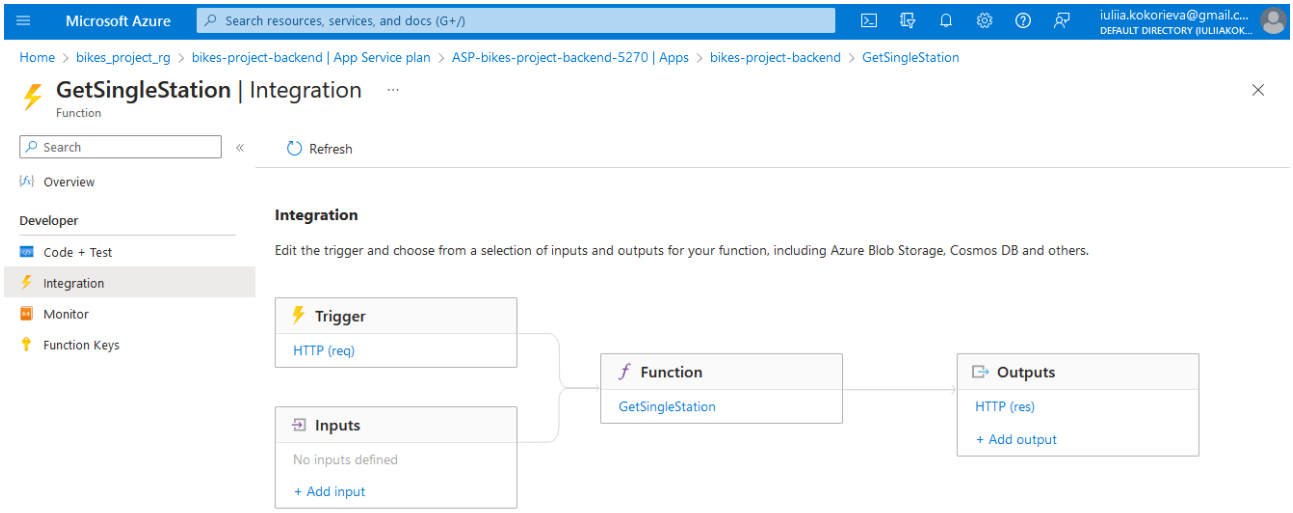
Ennen funktion ensimmäistä suoritusta, on luotava varastotili Azure-portaalissa. Varastotili tarvitaan, kun luodaan Azure Functions -sovellus, jotta yhteyden merkkijonot ja muut projektin tiedot voitaisiin tallentaa.

Funktio voidaan käynnistää Azure-välilehdessä VSCode:ssa Workspace-välilehdestä tai suorittamalla komento `func start` Azure Functions -projektin juuressa olevassa pääteikkunassa. Oletuksena funktion loppupiste paikallisella palvelimella olisi localhost:7071/api/<functionName>. Funktion tarjoama vastaus voidaan tarkistaa selainikkunasta.

4. Funktioiden julkaiseminen Azureen.

Ennen funktioiden julkaisua on tarpeen kirjautua Azure-tilille VSCode:sta. Tämä voidaan tehdä pääteikkunassa suorittamalla komento `az login` (jos Azure CLI on asennettu) tai Azure-välilehdellä. Seuraava vaihe on luoda Azure Functions -sovellus Azureen. Tämä voidaan tehdä portaalin kautta tai VSCode:n Azure-välilehden Resurssit-välilehdestä. Kun tarvittavat resurssit on luotu, paikallinen projekti voidaan julkaista Azureen Azure-välilehden Workspace-välilehden kautta.

Funktioiden julkaisemisen jälkeen ne saavat kukin oman URL-osoitteensa yhteisen Function Appin sisällä. Niitä voi tarkistaa ja muokata portaalista, vaikka koodin muokkausta tällä tavoin ei suositella (Microsoft Corporation 14.09.2023). Lisäksi voidaan suorittaa testiajo, muokata kunkin yksittäisen funktion integraatiota ja seurata toimintaa. Alla oleva kuva 9 havainnollistaa yksittäisen funktion Integraatio-välilehteä, jota voidaan käyttää muuttamaan funktion sisääntulevia ja lähteviä liitännöitä.

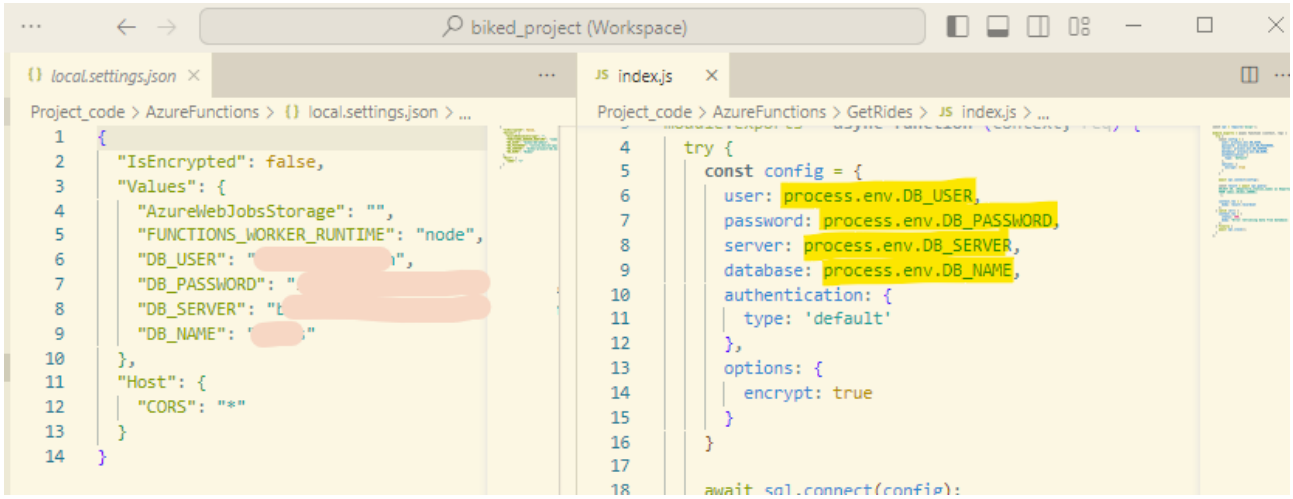


Kuva 9. Yksittäisen funktion näkymän kuvakaappaus, jossa on avoinna Integraatio-välilehti. Kuvakaappaus Azure portaalista (tekijä: Iuliia Kokorieva).

Alkuperäisen julkaisun jälkeen koodia voidaan edelleen muokata paikallisesti ja viedä pilveen, mikä johtaa funktion päivitykseen samalla tavalla kuin alkuperäinen julkaisu, CI/CD-menetelmällä.

5. Ympäristömuuttujien säilyttäminen.

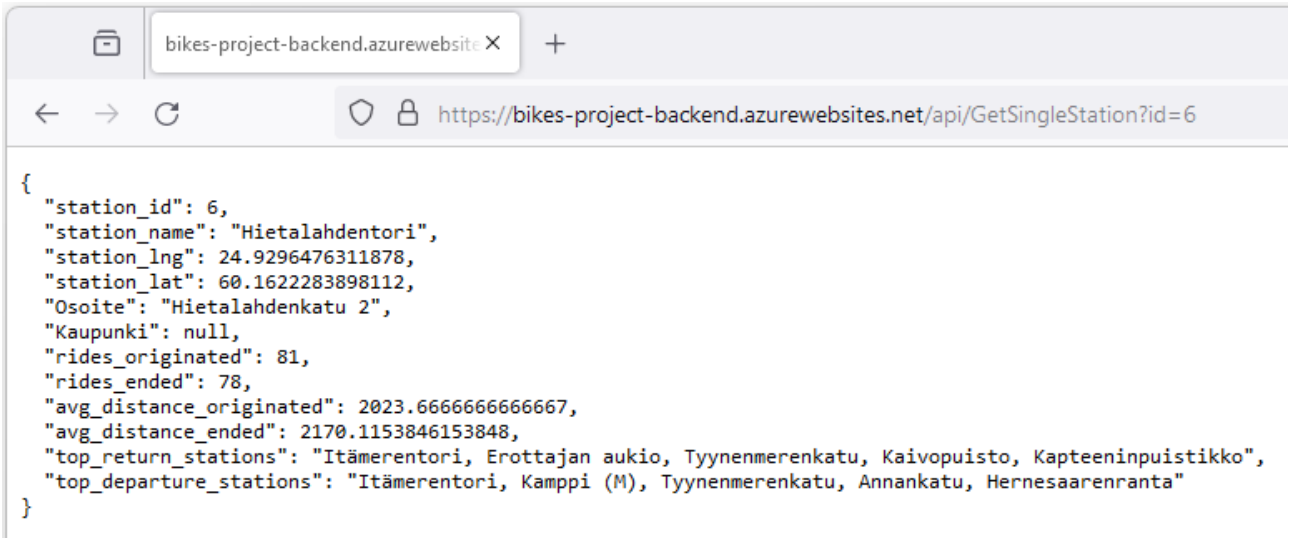
Nykyinen sovellus käyttää tietokantaan yhteyden muodostamista, mikä vaatii yhteystietojen tallentamista. Paikallista kehitystä varten ne tallennettiin projektin juurikansiossa olevaan `local.settings.json`-tiedostoon ja viitattiin koodissa seuraavassa muodossa, kuten kuvassa 10 esitetään. Vasemmalla puolella on ympäristömuuttujien määrittely `local.settings.json`-tiedostossa ja oikealla puolella niiden viittaus funktiokoodissa.



Kuva 10. Kuvakaappaus VSCode-työtilasta, jossa esitetään ympäristömuuttujien määrittely ja käyttö funktiosovellusprojektissa (tekijä: Iuliia Kokorieva).

`local.settings.json`-tiedostoa ei kuitenkaan julkaista pilveen, joten tietokantatunnukset määriteltiin alla manuaalisesti Azure Function Appin Configuration -välilehdellä, Application Settings -osiossa. Viittaus koodissa pysyy ennallaan.

Kun julkaisu on valmis ja ympäristömuuttujat on tallennettu, funktioiden oikea toiminta voidaan todentaa, ja ne toimittavat JSON-muotoisen datan vastauksena HTTP-pyyntöön. Kuva 11 havainnollistaa `GetSingleStation`-funktion antamaa vastausta. Huomattavaa on, että se hyväksyy tarvittavan aseman ID:n URL-parametrina.



Kuva 11. Vastaus, jonka `GetSingleStation`-funktio tarjoaa pilvessä suoritettaessa. Kuvakaappaus selaimesta (tekijä: Iuliia Kokorieva).

6.4 Ratkaisu frontendin käyttöönottoon Azure-ympäristössä

Kuten yllä mainittiin, "Bikes project":n frontend-koodi ei sisälly tähän opinnäytetyöhön. Kuitenkin paremman ymmärtämisen vuoksi on syytä mainita, että frontend toteutettiin yksisivuisena sovelluksena, tunnettu myös nimellä SPA (single page application), käyttäen React-kehystä tarvittavilla ulkoisilla kirjastoilla. SPA-arkkitehtuurin lähestymistapa tarkoittaa sitä, että sen sijaan, että ladataan useita uusia sivuja, verkkosivu (web-sovellus) lataa vain yhden HTML-tiedoston ja päivittää näytettävän tiedon tarpeen mukaan tai käyttäjän toimien mukaisesti (Schwarzmüller 2022, aliluku " Introducing Single Page Applications"). Toinen huomioitava tekijä on, että kyseessä on dynaaminen web-sovellus, mikä tarkoittaa, että se on aktiivisessa vuorovaikutuksessa palvelimen kanssa, käsittelee monimutkaisia kyselyitä ja laskelmia tarjotakseen reaaliaikaista, datavetoista sisältöä ja responsiivisen käyttäjäkokemuksen.

Frontendin käyttöönottoon Azure-ympäristöön liittyvät päätökset ja suoritetut vaiheet selitetään tämän luvun myöhemmissä osioissa.

6.4.1 Azure App Services -palvelun valinnan perustelut

Kohdassa 6.3.1 esitellään saatavilla olevat vaihtoehdot web-sovellusten isännöimiseen Azure-alustalla ja selitetään myös tekijän tekemä backend-palvelun valinta "Bike Rides" -projektille. Sama vaihtoehtojen luettelo voidaan soveltaa myös suunniteltaessa frontendin isännöintiratkaisun valintaa. On huomattava, että IaaS-vaihtoehdot, kontteihin liittyvät vaihtoehdot ja Java-spesifit vaihtoehdot hylättiin arviointiprosessin aikana, sillä niitä ei katsottu relevanteiksi React-sovelluksen

yksinkertaiseen isännöintiin. Azure Functions -palvelua ei myöskään pidetty sopivana, sillä se on tarkoitettu pienten, tapahtumaohjautuvien koodinpätkien suorittamiseen. Lisäksi Static Web Apps -palvelu ei tullut kysymykseen, koska tämä palvelu erikoistuu staattisiin web-sovelluksiin, kun taas "Bike Rides" -projektin frontend on dynaaminen. Laajan arvioinnin ja edellä mainittujen vaihtoehtojen hylkäämisen jälkeen tekijä valitsi lopulta Azure App Services -ratkaisun.

Azure App Services tarjoaa React SPA:n isännöintiin seuraavat edut:

- Helppo julkaisu ja integroitu DevOps: Kehittäjä voi julkaista joko koodin tai kontin ja valita vastaavan käyttöjärjestelmän ja suoritusaikaympäristö (runtime stack). On myös mahdollista ja varsin suoraviivaista ottaa käyttöön jatkuva julkaisu GitHub Actions -työkalun avulla, kuten tässä projektissa on tehty
- Automaattinen skaalaus: Hinnoittelutasosta riippuen on mahdollista määrittää skaalaustapa (manuaalinen, automaattinen tai sääntöihin perustuva). Tämä on ns. vaakasuuntaista skaalausta (scale out/in), mikä tarkoittaa, että valitun menetelmän perusteella lisätään tai poistetaan tarvittava määrä sovelluksen instansseja vastaamaan kysyntää. Pystysuuntainen skaalaus (scale up/down: hinnoittelutason muuttaminen lisäämään laskentatehoa, tallennustilaa jne.) on myös mahdollista, mutta se on tehtävä manuaalisesti.
- Seuranta ja diagnostiikka: Alusta tarjoaa sisäänrakennetun sovelluksen suorituskyvyn seurannan ja diagnostiikan, mikä helpottaa ongelmien tunnistamista ja ratkaisemista.

Kustannusten hallinta voi olla sekä etu että haitta projektin laajuudesta ja vaatimuksista riippuen. Tämän projektin tarkoituksiin ilmainen hinnoittelutaso riittää, vaikka se sisältää tiettyjä rajoituksia, kuten skaalaustavan mukauttamistoiminnon puuttumisen palvelupaketista. Suuremmissa projekteissa kustannukset voivat kuitenkin nousta merkittävästi, ja epätasainen kulutus voi johtaa odottamattomiin kustannuksiin. Tämän välttämiseksi kustannusten hallintaan on kiinnitettävä tarkkaa huomiota, ja budjetteja ja kustannusilmoituksia voidaan ottaa käyttöön.

6.4.2 React-sovelluksen kehittäminen ja julkaisu Azure App Services -palveluun

React-sovellus kehitettiin paikallisella tietokoneella, jossa oli asennettuna Node.js ja npx- pakettien ajo-ohjelma. Kehitystyö tehtiin käyttäen VSCode-kehitysympäristöä. Versionhallintaa varten hyödynnettiin yhteistä GitHub-projektin etävarastoa, projektin osaa varten erityisesti perustettua kansiota: https://github.com/YuliaKokorieva/bikes-project/tree/master/BikeRidesFront.

Jotta voitaisiin työskennellä sekä paikallisen että pilvipohjaisen Azure Functions -backendin parissa, kaksi vastaavaa käynnistyskomentoa tallennettiin `package.json`-tiedostoon, ja

asiaankuuluvat arvot `REACT_APP_FUNC_SOURCE`- ympäristömuuttujalle (`local` tai `cloud`) määriteltiin (ks. kuva 12 alla).

Kuva 12. Kuvakaappaus `BikeRidesFront`-projektin `package.json`-tiedoston osasta, joka näyttää sovelluksen käynnistyskomennot (tekijä: Iuliia Kokorieva)

Ensimmäinen vaihe sovelluksen julkaisemisessa Azure App Services -palveluun liittyy Web App -resurssin luomiseen Azure-ekosysteemissä. Azure-resurssien luomiseen on useita erilaisia menetelmiä. Nämä menetelmät sisältävät manuaalisen luontiprosessin Azure-portaalissa, komentoriviliittymien käytön (kuten Azure CLI tai Azure PowerShell) tai infrastruktuurin koodina (IaC) -menetelmien hyödyntämisen, kuten Terraform, ARM-mallipohjat tai Azure Bicep. Luontiprosessin aikana on tehtävä seuraavat perustavat valinnat:

1. Resurssiryhmä ja sijainti.
2. Julkaisu koodina / Docker-konttina / Static Web App:na.
3. Suoritusaikaympäristö / runtime stack (esim. .NET, Java, Node, PHP tai Python, tarvittaessa tietyllä versiolla).
4. Käyttöjärjestelmä (Linux tai Windows).

Valittujen parametrien perusteella tulisi valita App Service Plan eli ASP. Se määrittää taustalla olevan infrastruktuurin ja isännöinnin kokoonpanon Azure App Services -palveluun julkaistuille verkkosovelluksille. Verkkosovellusten suorituskykyä, skaalautuvuutta ja niihin liittyviä kustannuksia voidaan hallita valitsemalla sopiva ASP.

ASP:t liittyvät Azure App Services -instansseihin, ja useat verkkosovellukset voivat jakaa saman ASP:n resurssien käytön optimoimiseksi (edellyttäen, että sovelluksilla on sama suoritusaikaympäristö (runtime stack), sijainti ja käyttöjärjestelmä). Hinnoittelusuunnitelman mukaan voidaan valita vaihtoehtoinen varajärjestelmä. Tämä tarkoittaa sitä, että Azure App

Services -palvelussa isännöidyt verkkosovellukset jaetaan useisiin saatavuusalueisiin samassa sijainnissa, mikä parantaa viansietokykyä ja korkeaa käytettävyyttä. Jos alueella tapahtuu vika, sovellukset jatkavat toimintaansa toisesta alueesta, mikä minimoi käyttökatkokset ja varmistaa luotettavuuden.

Bike Rides -sovelluksen frontendin isännöinnin tarkoituksessa luotiin Web App seuraavilla parametreilla:

1. Resurssiryhmä: yhteinen koko projektia varten, nimeltään "`bikes_project_rg`".
2. Sijainti: North Europe.
3. Julkaisumalli: Koodi.
4. Ajonaikainen kehys: Node – 16-lts.
5. Käyttöjärjestelmä: Linux.
6. Hinnoittelutaso: Ilmainen (F1), yhdellä instanssilla käytettävissä ilman varajärjestelyä.

Alkuperäisen luomisen jälkeen Web App -sovellus ei sisällä vielä mitään koodijulkaisua. CI/CD-menetelmän käyttöä suositellaan aina kun, mahdollista. Siksi tässä projektissa sovelluksen jatkuva julkaisu GitHubin etävarastosta GitHub Actions -työkalun kautta on otettu käyttöön. Tämän prosessin lisätietoja annetaan seuraavassa aliluvussa.

6.4.3 GitHub-putkiston asettaminen frontendin jatkuvaa julkaisua varten

Luvussa 4 esiteltyjen periaatteiden ja CI/CD-käytäntöjen mukaisesti tekijä tuotti julkaisuputkiston, joka päivittyy sovellusta frontend-etävarastoon tehtyjen sitoumusten perusteella.

Ensimmäinen vaihe on yhdistää GitHub-tili Azure App Services -palvelun Deployment Center -välilehdeltä uudelleen luodusta Web App -sovelluksesta Deployment-osiossa, kehittäjän tulisi valita GitHub Source-pudotusvalikosta. Onnistuneen todennuksen jälkeen on valittava organisaatio, etävarasto ja haara liittyen asianmukaiseen GitHub-projektiin. On mahdollista generoida työnkulkutiedosto (`.yml`) ja sitouttaa se etävarastoon, mutta tekijä suosi manuaalisen työnkulun luomista etävarastoon ja sen käyttöä julkaisuprosessissa. Tämän tekemiseksi ensin on generoitava julkaisuprofiili eli sovellustason tunnus. Tämä profiili tulisi ladata Web App -sovelluksen Overview-välilehdeltä. Tämä tiedosto tulisi seuraavaksi tallentaa GitHubiin salaisuuksiksi (Repository → Settings → Secrets and variables → Actions) ja osoittaa oikeaan

muotoon työnkulkutiedostossa. Tämä mahdollistaa yhteyden muodostamisen etävaraston ja Web Appin välille ja sallii automaattiset julkaisut.

Seuraava vaihe on `.yml`-tiedoston luominen ja sijoittaminen etävaraston `.github/workflows`-kansioon. Projektin etävarastossa luotiin tiedosto nimeltään `master_bikes-app_front.yml`. Sen täydellinen sisältö on esitetty tämän opinnäytetyön liitteenä (Liite 2). Kätevyyden vuoksi, kun määritellään putkistoa ja testataan suorituksia, työnkulun tiedostossa on kaksi vaihtoehtoista laukaisinta. Niitä ei voi käyttää samanaikaisesti, toinen niistä tulee aina kommentoida pois (ks. kuva 13 alla):

1. Manuaalinen käynnistin testitarkoituksiin (rivi 8, kuvakaappauksessa kommentoitu pois).
2. Automaattinen käynnistin tuotantoa varten (rivit 10-14, se laukaisee työnkulun, kun uusi sitoumus työnnetään `master`-haaraan. On tärkeää mainita, että koska frontend-koodi ei sijaitse etävaraston juurikansiossa, polku on annettu asianmukaisessa muodossa.

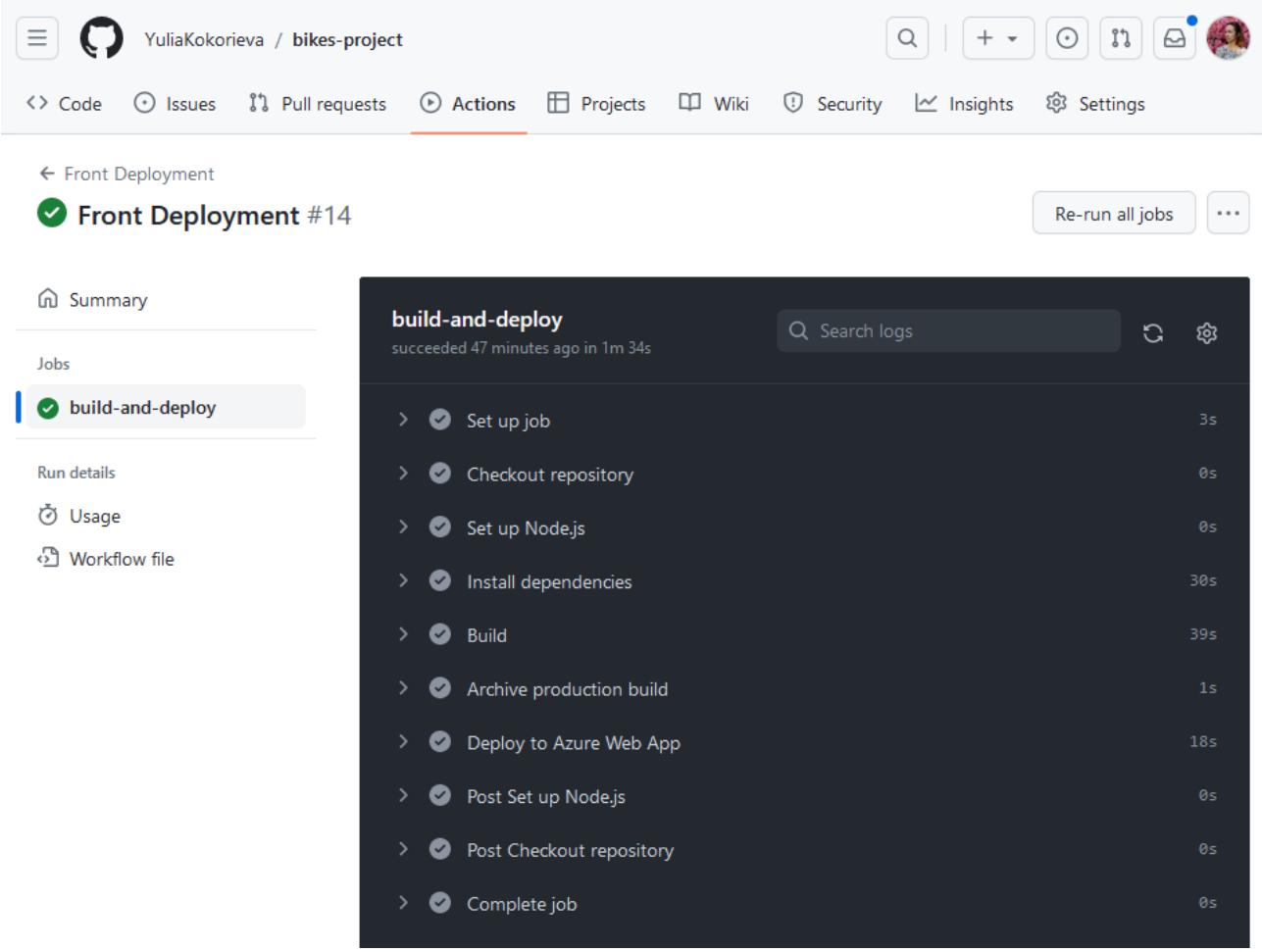
```
7
8     # on: workflow_dispatch
9     on:
10      push:
11        branches:
12          - master
13        paths:
14          - 'BikeRidesFront/**'
15
```

Kuva 13. Tapahtumat, jotka käynnistävät työnkulun. Kuvakaappaus `master_bikes-app_front.yml` -tiedoston osasta (tekijä: Iuliia Kokorieva).

Yksinkertaisessa työnkulussa on määritelty yksi työ (job), nimeltään "`build-and-deploy`". Työnkulussa luodaan ja arkistoidaan tuotantoversio ja sen jälkeen käyttämällä Azure WebApp -tehtävää (`azure/webapps-deploy@v2`) luotu artefakti julkaistaan ZIP-pakettina. Huomattavasti tämä jälkimmäinen vaihe käyttää aiemmin tallennettua julkaisuprofiilin salaisuutta:

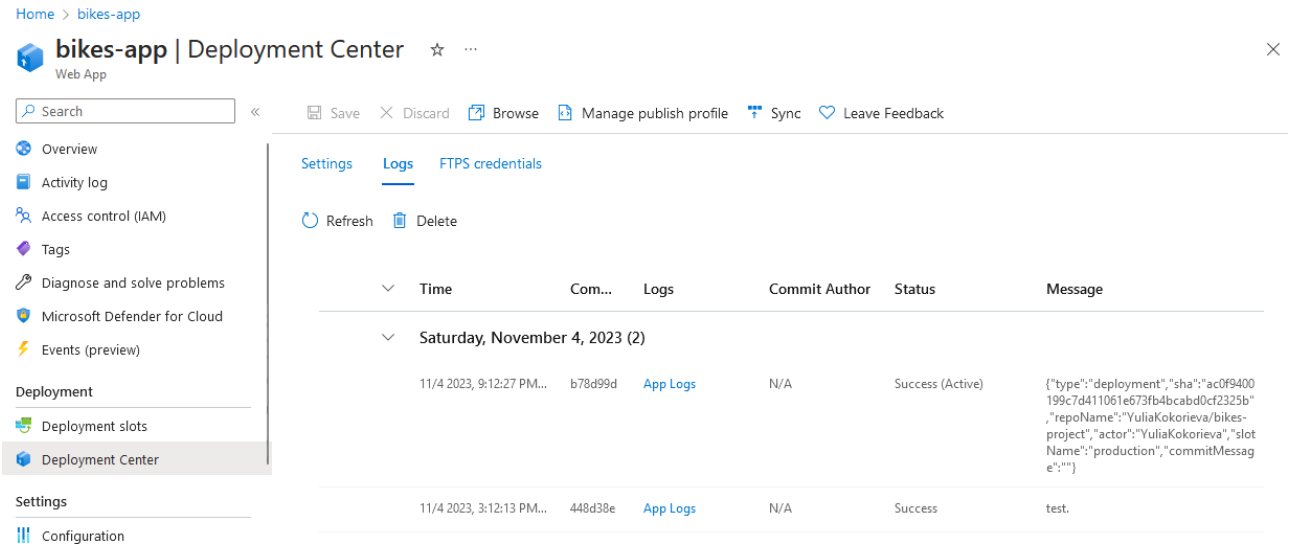
```
publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}.
```

Kuva 14 esittää onnistuneesti suoritetun ajon yleiskatsauksen.



Kuva 14. Onnistuneesti suoritetun työnkulun yleiskatsaus. Kuvakaappaus GitHub Actions -sivusta (tekijä: Iuliia Kokorieva).

Samanaikaisesti julkaisulokeja voi tarkastella Web App -sovelluksen Deployment Center -osion "Logs" -välilehdeltä (ks. kuva 15 alla).



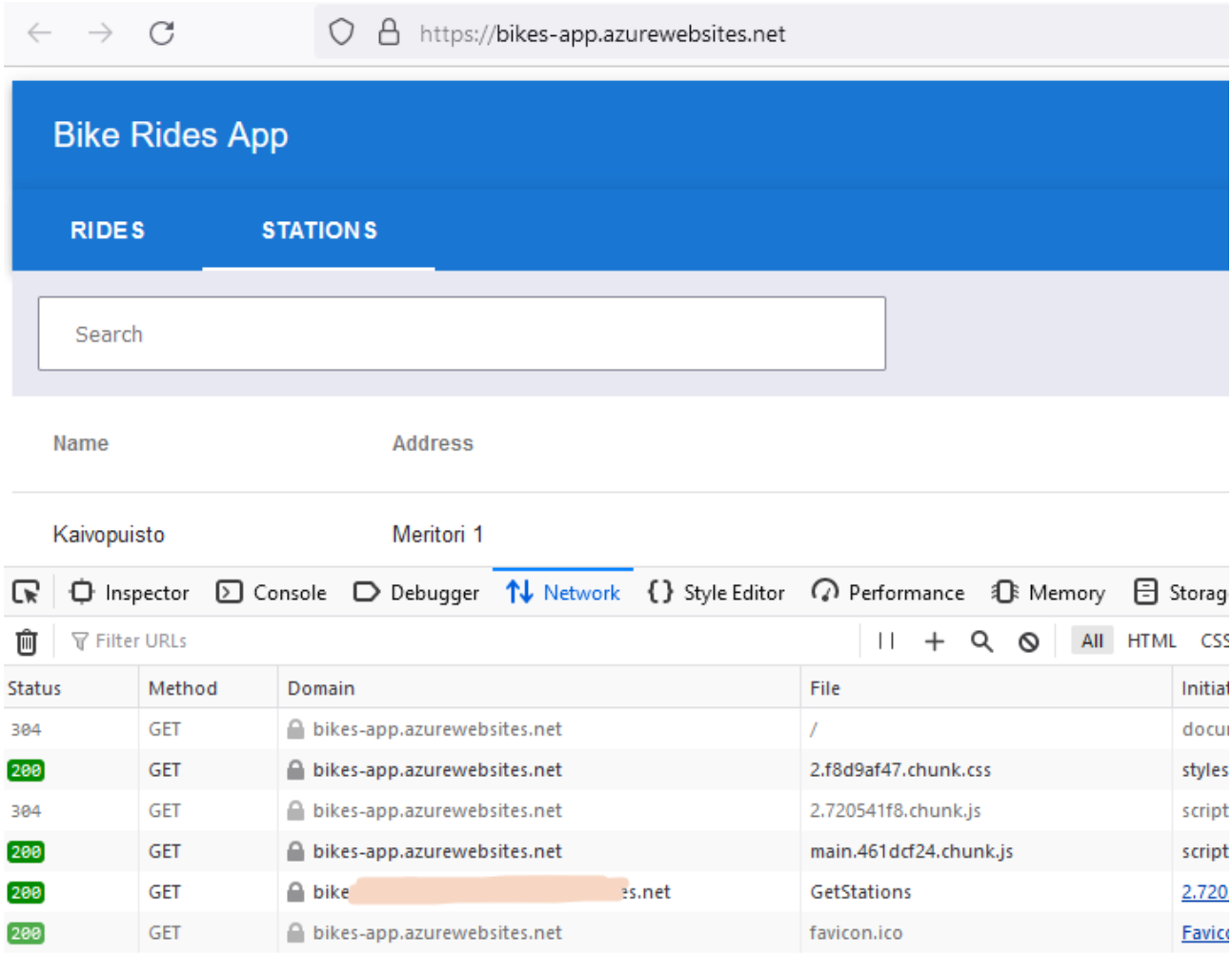
Kuva 15. ”bikes-app” -sovelluksen julkaisulokit. Kuvakaappaus Azure portaalista (tekijä: Iuliia Kokorieva).

6.4.4 Salaisuuksien hallinta toteutetussa ratkaisussa

Yksi jatkuvaan julkaisuun liittyvä näkökohta, jota ei käsitelty edellisessä luvussa, on salaisuuksien hallinta. Kun sovelluskoodi on kehitetty ja testattu paikallisesti, se viittaa `.env`-tiedostoon tallennettuihin salaisuuksiin. Tätä tiedostoa ei kuitenkaan lisätä versionhallintaan, koska ei ole turvallista säilyttää tunnistetietoja ja muita salaisia muuttujia avoimesti koodissa. Siksi tarvittavat tunnukset (`REACT_APP_BASE_URL` - päätepiste HTTP-pyyntöjen lähettämiseen backendille ja `REACT_APP_GOOGLE_API_KEY` Google Maps:in käyttämiseen yksittäisen aseman paikantamiseen vastaavassa yksittäisen aseman näkymässä) on tallennettu GitHub-etävarastoon Settings-sivulle Actions-salaisuuksina, samalla tavalla kuin Azure WebApp -julkaisuprofiili. Myöhemmin niihin viitataan työnkulutiedostossa seuraavalla tavalla ja niitä käytetään sovelluksen rakentamisessa julkaisua varten:

```
env:
  REACT_APP_BASE_URL: ${{ secrets.REACT_APP_BASE_URL }}
  REACT_APP_GOOGLE_API_KEY : ${{ secrets.REACT_APP_GOOGLE_API_KEY }}
```

Selaimen ikkunan Web Developer -työkalujen Network-paneelin tarkistaminen Bikes App -sovelluksen ollessa käynnissä todistaa, että linkit ovat viitatut oikein (ks. kuva 16 alla).



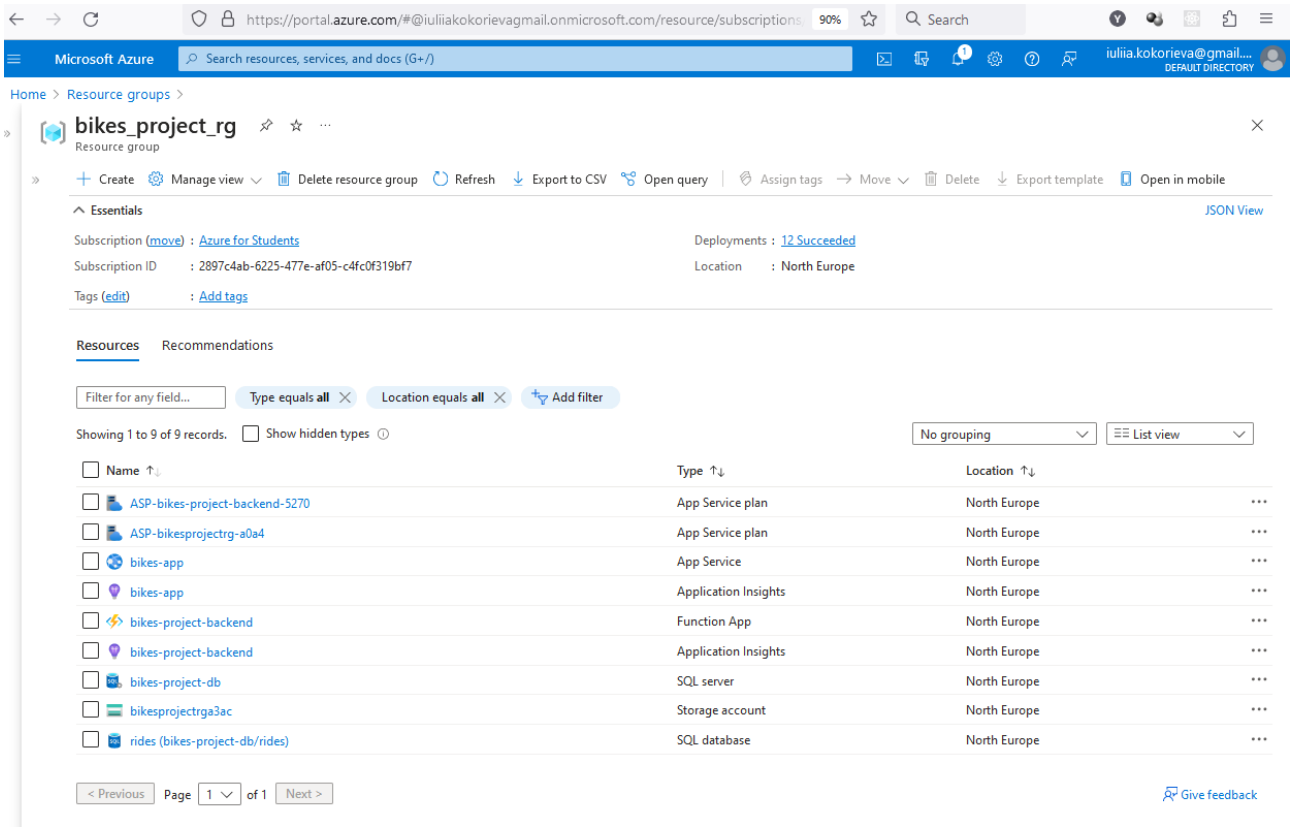
Kuva 16. Kuvakaappaus Chrome selaimen Web Developer -työkalujen Network-paneelista (tekijä: Iuliia Kokorieva).

Tämä alaluku päättää keskustelun projektin komponenttien kehittämisestä ja käyttöönotosta. Seuraava osio esittelee toteutetun ratkaisun yleiskatsauksen.

6.5 Tulosten yleiskatsaus

Alaluvussa 6.1 esitettiin sovelluksen yleinen suunniteltu arkkitehtuuri. Tämän jälkeen alaluvut 6.2–6.4 tarjosivat yksityiskohtaisen kuvauksen toteutusratkaisuista sekä sovelluskomponenttien kehitys- ja käyttöönottoprosessista. Tässä luvussa annetaan toteutetun ratkaisun yleiskuva.

Kuva 17 alla näyttää koko `bikes_project_rg` -resurssiryhmän sisällön tekijän Azure-tilauksessa.



Kuva 17. Kuvakaappaus `bikes_project_rg` resurssiryhmän sisällöstä Azure-tilauksessa (tekijä: Iuliia Kokorieva).

SQL Server ja SQL Database viittaavat tietokantapalveluihin, jotka on kuvattu luvussa 6.2; `bikes-project-backend` Function App -resurssi ja siihen liittyvä App Service Plan (`ASP-bikes-project-backend-5270`) -resurssi vastaa Azure Functions -sovelluksesta, joka toimii sovelluksen backendina (luku 6.3); ja lopuksi `bikes-app` App Service -resurssi vastaavan App Service Plan (`ASP-bikesprojectrg-a0a4`) -resurssin kanssa isännöi sovelluksen React-käyttöliittymää (luku 6.4). Lisäresurssit (Application Insights kullekin App Service / Function App -palveluille) on luotu automaattisesti seurantatarkoituksia varten. Seuraavissa luvuissa annetaan arviointi toteutetusta sovelluksesta yhdessä siihen liittyvän kustannuskatsauksen kanssa.

6.5.1 Keinot sovelluksen parantamiseen ja jatkokehittämiseen

Tässä osiossa käsitellään mahdollisia tapoja projektin jatkokehitykseen ja parantamiseen.

Ensinnäkin on todettava, että prototyyppisovelluksen kehitysvaiheessa tekijä oli Azure-opintojensa alkuvaiheissa, ja näin ollen useita tärkeitä mahdollisuuksia, jotka julkisen pilven tarjoaja tarjoaa, jäi

käyttämättä. Samalla ne ovat selkeitä tavoitteita tuleville oppimisvaiheille ja sovelluksen parantamiselle.

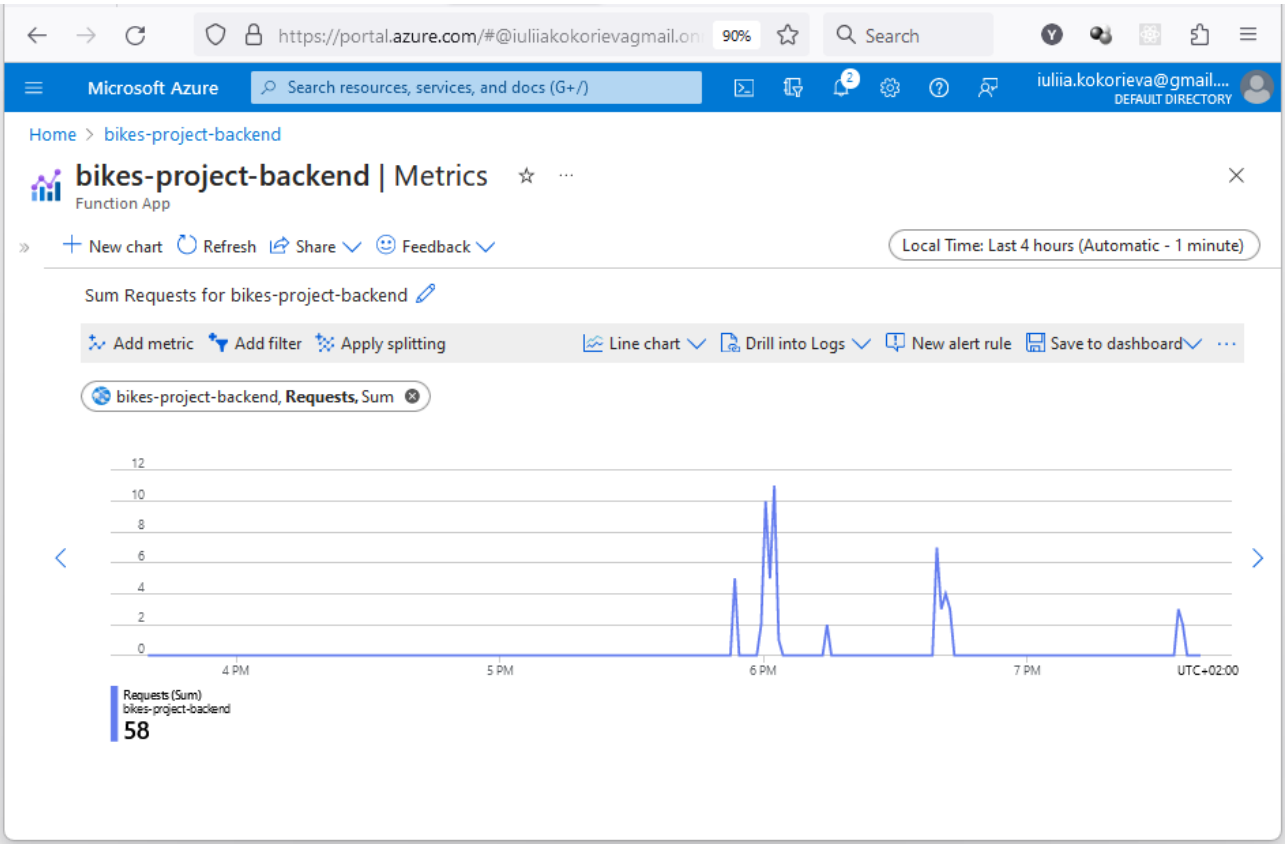
Ensimmäinen ja ehkä tärkein näkökohta on **tietoturvallisuus**. Nykyisessä versiossa Azure Functions -backend hyväksyy nimettömiä HTTP-pyyntöjä, jotka on lähetetty funktion loppupisteisiin. Mahdollisen jatkokehityksen kohdalla pääsy funktioihin tulisi rajoittaa. Tässä on muutamia esimerkkejä toteutustavoista:

1. Jokaisen funktion valtuustason muuttaminen Functions App -palvelussa nimettömästä *function*- tai *admin*-tasolle.
2. Edistyneemmän ratkaisun toteuttaminen, kuten hallittu identiteetti (Managed Identity), joka mahdollistaa tarkemman hallinnan pääsystä Function App -palveluun hyödyntäen tehokasta RBAC-järjestelmää (Role Based Access Control). Tämä on turvaparadigma, joka rajoittaa järjestelmän pääsyoikeuksia roolien perusteella, jotka on määritelty yksittäisille käyttäjille ja muille sovelluksille, mahdollistaen hienojakoisen hallinnan ja estäen luvattomat toimet.

Lisäturvaa ympäristömuuttujien (tietokantayhteyden salasanatiedot yms.) tallentamiseen voidaan tarjota käyttämällä KeyVault -palvelua, erityisesti tätä tarkoitusta varten suunniteltua resurssia, sen sijaan että käytettäisiin Function App -palvelun sovellusasetuksia.

Seuraava vaihtoehto ei kuulu suoraan suojausluokkaan, mutta liittyy kuitenkin siihen melkoisesti: **API Management**. Azure API Management on täysin hallittu palvelu, joka mahdollistaa organisaatioiden API:en luomisen, julkaisun ja suojaamisen. Se toimii porttina, jonka avulla voidaan hallita pääsyä, seurata käyttöä ja tarjota analytiikkaa API:ille. Kun sitä käytetään yhdessä Azure Function App -palvelun kanssa, API Management tarjoaa keskitetyn hallinnan, todentamisen ja rajoitukset palvelimettomille funktioille. Se myös virtaviivaistaa funktioiden esittelyä ja käyttöä API:ina. Tämä integraatio parantaa skaalautuvuutta, turvallisuutta ja analytiikkaa yksinkertaistaen samalla koko API-elinkaaren hallintaa.

Ja viimeiseinä (muttei vähäisimpinä) ovat Azuren tarjoamat **seurantamahdollisuudet**. Azure-alusta tarjoaa merkittävän määrän oletusarvoisia mittareita Azure Monitor -työkalulla, mutta vielä laajempaa analytiikkaa tarjoaa Application Insights. Se on erityisominaisuus Azure Monitor -työkalussa, joka on suunniteltu sovellusten suorituskyvyn ja käytön seurantaan ja ymmärtämiseen. Esimerkiksi tietoja komponenttien integraatiosta voidaan esitellä muun muassa graafisesti tai tietoja sovelluksen käytetyistä instansseista. Application Insights vaatii tiettyjä mukautuksia sovelluksen koodiin, mikä on tutkittava tarkemmin, jotta se voidaan ottaa käyttöön. Kuvassa 18 alla on esimerkki Funktion App -palvelun mittareiden käytöstä. Viivakaaviossa on esitetty HTTP-pyyntöjen määrä, jotka sovellus on vastaanottanut viimeisten 4 tunnin aikana.



Kuva 18. Azure Monitor -palvelun ”Requests”-mittarin viivakaavio. Kuvakaappaus Azure portaalista (tekijä: Iuliia Kokorieva).

Puhuttaessa **frontend**-komponentista, on todettava, että se olisi voitu julkaista konttina. Tässä on yleiskuva tarvittavista vaiheista:

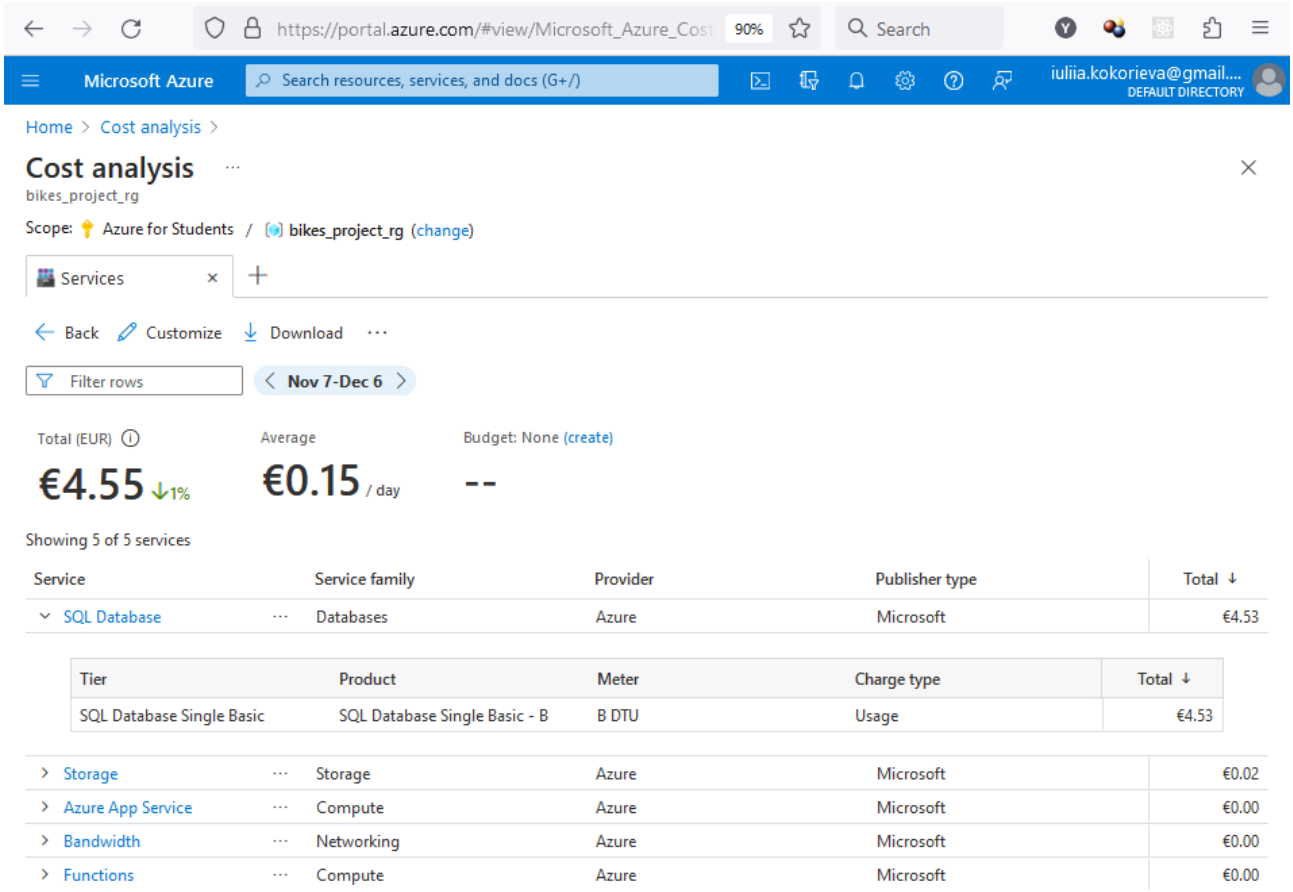
1. Sovelluksen kontittaminen luomalla asianmukainen Dockerfile.
2. Imagen työntäminen Azure Container Registry -palveluun.
3. Kontin julkistaminen Azure App Service- tai Azure Container Instance -palveluun.

Tämä lähestymistapa vaatii laajempaa kokemusta tekniikasta, kuin mitä tekijällä tällä hetkellä on. Kuitenkin modernin web-kehityksen kasvavan suosion vuoksi tämä aihealue on epäilemättä tutustumisen arvoinen.

6.5.2 Budjettikatsaus

Projektiin liittyvien kustannusten arviointi oli yksi tämän opinnäytetyön tehtävistä. Kustannusten seuranta on oletuksena käytettävissä Azure-tilauksessa. Tällä tavoin yksi tavoitteen osa, nimittäin kustannusten hallinnan harjoittelu, on saavutettu. Cost Management -palvelu tarjoaa laajat mahdollisuudet järjestellä, suodattaa ja visualisoida projektiin liittyviä kustannuksia. Kuvassa 19

alla on esimerkki kustannusten esittämisestä: kustannukset palvelun mukaan viimeisten 30 päivän aikana. Tämä on vain esimerkki, ja muita näkymiä on runsaasti saatavilla.



Kuva 19. `bikes-project-rg` -resurssiryhmän resurssien kustannukset kuukauden aikana. Kuvakaappaus Azure portaalista (tekijä: Iuliia Kokorieva).

Kaikesta huolimatta suunnitelma kustannustason arvioinnista, joka liittyy tämän tyyppiseen sovellukseen, on epäonnistunut. Opinnäytetyön tulosta suunnitellessa huomiotta jäävät tekijät ovat seuraavat: Ensinnäkin pilvisovelluksesta aiheutuvat kustannukset riippuvat suuresti itse sovelluksen todellisesta (tai suunnitellusta) kuormituksesta. Prototyyppisovelluksen tapauksessa liikenne palveluihin oli lähes olematonta. Toiseksi tässä oppimissovelluksessa valittiin resurssien edullisimmat hinnoitteluluokat. Tämä minimoi kustannukset verrattuna sovelluksen mahdolliseen todelliseen tuotantokäyttöön, koska tuotannossa ei käytetä ilmaisia tai edullisimpia luokkia. Näin ollen toteutettua sovellusta ei voida käyttää edes likimääräiseen kustannusarvioon.

7 Pohdinta

Työn tuloksena on kehitetty ja otettu käyttöön kolmiosainen sovellus Azure-pilvessä. Sovellus noudattaa mikropalveluarkkitehtuurin lähestymistapaa, mikä tarkoittaa, että jokainen komponentti kehitetään ja toimii itsenäisesti samalla palvellen toisiaan viestinnän avulla. Tietokerros on toteutettu relaatiotietokantana, joka on otettu käyttöön Azure SQL -tietokantapalveluna. Palvelinpuoli on toteutettu palvelimettomana Function App -palveluna, ja käyttöliittymä on React SPA, joka on otettu käyttöön Azure App Services -palveluna.

Tämä oli oppimisprojekti, ja tässä yhteydessä tavoite tutkia saatavilla olevia vaihtoehtoja pilviympäristössä ja oppia käyttämään niitä oikein on saavutettu. Tämä osoittautui erittäin hyödylliseksi tekijän tulevissa työtehtävissä. Tästä annetaan yksityiskohtaisempaa tarkastelua myöhemmin tässä luvussa, osiossa 7.2. Seuraavissa luvuissa on aika keskittyä arvioimaan sovelluksen toteutusta ja käyttöönottoa näkökulmasta, joka on määritelty osiossa 2.3.

7.1 Johtopäätökset ja korjausehdotukset toteutetusta projektista

Osion 2.3 määrittelemien tavoitteiden mukaisesti projekti keskittyi kolmeen ensisijaiseen tutkimus- ja toteutusalueeseen:

1. Pilvisovelluksen arkkitehtoninen suunnittelu ja sen jälkeinen toteutus.
2. Toimivan CI/CD-putken kehittäminen.
3. Budjettiharkintojen arviointi.

Samanaikaisesti annetaan vastauksia samassa osiossa esitettyihin tutkimuskysymyksiin. Jokainen tavoite tarkastellaan erikseen.

1. Projektin arkkitehtuuri ja toteutus

TK1: Mikä muodostaa sopivan Azure-arkkitehtuurin yksinkertaisen fullstack-web-sovelluksen käyttöönotossa?

On tärkeää huomauttaa, että optimaalinen arkkitehtuuri riippuu täysin projektin teknisistä määrityksistä. ”Bikes project”:in tapauksessa tehokkaimmaksi arkkitehtuuriksi osoittautui mikropalveluarkkitehtuuri (selitys annettu osiossa 6.1). Kyseisen projektin tapauksessa tämä lähestymistapa mahdollistaa laajan valikoiman erilaisiin tehtäviin sopivien Azure-resurssien tutkimisen sekä tarjoaa joustavan perustan jatkokehitykselle. Kunkin komponentin erityisten resurssien valinta on selitetty yksityiskohtaisesti luvussa 6.

Toteutettu projekti heijastaa tekijän varhaista vaihetta Azure-teknologioiden opiskelussa. Tässä yhteydessä tekijä ilmaisee täyden tyytyväisyyden valittuihin ratkaisuihin projektin toteutuksessa. Tästä huolimatta, kuten on mainittu osiossa 6.5.1, voidaan esittää selkeitä suosituksia jatkokehitykseen ja parannuksiin. Ainoa päätös, joka on avoin uudelleenarvioinnille, on valinta käyttää ilmaisia tai edullisia hinnoittelutasoja valituille resursseille. Tuotantokeskeisten tasojen valitseminen voisi tarjota mahdollisuuden käyttää kehittyneempiä ominaisuuksia, kuten Deployment Slots ja automaattista skaalausta. (Azure Deployment Slots ovat verkkosovelluksen eristettyjä instansseja, jotka mahdollistavat päivitysten testauksen, vaiheittaisen käyttöönoton ja päivitysten asteittaisen käyttöönoton ilman vaikutusta tuotantoympäristöön.) Samanaikaisesti tämä valinta ei johtaisi odottamattoman suuriin kustannuksiin, jos resurssit poistetaan käytöstä projektin onnistuneen toteutuksen jälkeen.

Infrastruktuuri koodina (IaC) -lähestymistavan käyttäminen, kuten tekijälle aiemmin tuttu Terraform-työkalu, edistäisi kustannusten alentamista merkittävästi. Tämä saavutettaisiin infrastruktuurin automatisointi-, poisto- ja uudelleenkäyttöprosesseilla todellisten tarpeiden mukaisesti. Tämän seurauksena ei olisi tarvetta ylläpitää infrastruktuuria Azure-ympäristössä kehitysjaksojen välillä. Se poistaisi tarpeen maksaa käyttämättömistä resursseista.

2. CI/CD-käytännöt

TK2: Millainen on tehokas ratkaisu CI/CD-lähestymistavan integroinnissa?

Azure Functions -backend julkaistaan jatkuvasti Azure Function Core Tools -työkalujen avulla, kun taas käyttöliittymäkomponentin CI/CD:ssä käytetään GitHub Actions -putkistoa. Sitä voidaan kehittää edelleen esimerkiksi lisäämällä tarvittavia testejä, mutta kokonaisuutena se on oikein määritelty, toimii hyvin ja salaisuudet säilytetään turvallisesti. Molemmat työkalut vaikuttavat loogisilta valinnoilta, ottaen huomioon kehitystyökalujen, kuten Azure Functions Core Toolsin ja GitHub-repositorion, käytön.

CI/CD-käytäntöjen toteutuksen tavoite katsotaan tekijän toimesta onnistuneesti suoritetuksi.

Mikäli, kuten edellisessä kappaleessa mainittiin, infrastruktuurin automaatiota Terraformin avulla käytettäisiin, tarvittaisiin lisäksi putkisto infrastruktuurin päivittämiseksi automaattisesti sitoutettujen koodimuutosten perusteella.

3. Kustannusten yleiskatsaus

Tätä tavoitetta ei ole saavutettu onnistuneesti, kuten selostettu osiossa 6.5.2. Toisin sanoen toteutettua sovellusta ei voida käyttää budjettivertailukohtana. Tätä varten olisi pitänyt valita tuotantotason hinnoittelutasot.

Yleisesti ottaen tekijä arvioi työtään erittäin hyödylliseksi oppimisponnisteluksi, joka johti täysin toimivan yksinkertaisen sovelluksen käyttöönottoon. Kuitenkin osioissa 6.5.1 ja tässä kohdassa mainitut näkökohdat edellyttävät lisätutkimuksia tuotteen laadun parantamiseksi.

7.2 Työn tulosten sovellettavuus tekijän työtehtäviin ja tavoitteisiin

Kuten luvussa 2 mainittiin, Microsoft ja tekijän työnantajayritys Tietoevry ovat strategisia kumppaneita. Tietoevry panostaa aktiivisesti henkilöstönsä kouluttamiseen Azure-teknologioiden alalla, ja näitä taitoja pidetään erittäin arvokkaina. Tässä yhteydessä kaikkiin työntekijöiden suorittamiin oppimisponnisteluihin, niin itsenäisiin kuin yrityksen järjestämiin, kannustetaan yrityksen taholta.

Tekijän oppimisprojekti, joka muodostaa tämän opinnäytetyön perustan, toimi alustana soveltaa tietoja, jotka hän hankki Microsoftin ohjaajan vetämissä koulutuksissa loppuvuodesta 2022. Lisäksi syvemmän sitoutumisensa Azureen vuoksi hänet valittiin osallistumaan yrityksen sisäisen henkilöstön kehittämiskeskuksen Tech Services Academyn järjestämään Azure Developer -koulutusohjelmaan. Osallistumisen edellytyksenä oli AZ-900: Azure Fundamentals -sertifikaatin suorittaminen, joka on perustason sertifikaatti Se kattaa yleiset pilviteknologian käsitteet ja erityisesti Azure-pilven.

Koulutuksen jälkeen marraskuussa 2023 hän suoritti AZ-204: Azure Developer Associate -sertifikaatin. Se on ammattilaistason sertifikaatti, joka keskittyy taitoihin, joita tarvitaan sovellusten kehittämiseen Azure-pilvessä. Käsiteltäviä aiheita ovat laskentaratkaisut, tallennus, turvallisuus, seuranta ja vianmääritys sekä integrointi ja viestinvälitys. Nämä sertifikaatit ovat käytännön varallisuutta, jotka merkittävästi lisäävät tekijän uskottavuutta pilviteknologioiden ammattilaisena. Tekijällä on selkeät suunnitelmat jatkaa uransa kehittämistä kohti Cloud Developer -tehtäviä, ja tämä opinnäytetyö, osana laajempaa Azure-aihealueen oppimisponnisteluaan, teki epäilemättä merkittävän panoksen tähän tavoitteeseen.

8 Lähteet

The 2020 Scrum Guide. Luettavissa: https://scrumguides.org/scrum-guide.html. Luettu: 11.11.2023.

Agile Manifesto. 2001. Principles behind the Agile Manifesto. Luettavissa: https://agilemanifesto.org/principles.html. Luettu: 13.10.2023

Bass, L., Clements, P., Kazman, R. 2021. Software Architecture in Practice, 4. painos. Addison-Wesley Professional. E-kirja. Luettu: 10.10.2023.

Bhatt, S. 14.04.2023. Azure Cosmos DB vs. Azure SQL Database: Which One is Right for Your Data Needs? Luettavissa: https://the-tech-guy.in/2023/04/14/azure-cosmos-db-vs-azure-sql-database/. Luettu: 10.09.2023.

Bigelow S.J., Neenan S., Casey K. What is public cloud? Everything you need to know. Luettavissa: https://www.techtarget.com/searchcloudcomputing/definition/public-cloud. Luettu: 25.11.2023.

Eurostat. 2021. Cloud computing - statistics on the use by enterprises. Luettavissa: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Cloud_computing_-_statistics_on_the_use_by_enterprises. Luettu: 30.11.2023.

Franklin, B. 27.01.2023. The digital forecast: 40-plus cloud computing stats and trends to know in 2023. Luettavissa: https://cloud.google.com/blog/transform/top-cloud-computing-trends-facts-statistics-2023?utm_source=twitter&utm_medium=unpaidsoc&utm_campaign=fy23q2-googlecloud-blog-transform-in_feed-no-brand-global&utm_content=-&utm_term=-&linkId=9080396. Luettu: 28.10.2023.

Gartner. 19.04.2023. Gartner Forecasts Worldwide Public Cloud End-User Spending to Reach Nearly $600 Billion in 2023. Luettavissa: https://www.gartner.com/en/newsroom/press-releases/2023-04-19-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-reach-nearly-600-billion-in-2023#:~:text=Worldwide%20end-user%20spending%20on%20public%20cloud%20services%20is,generative%20artificial%20intelligence%20%28AI%29%2C%20Web3%20and%20the%20metaverse. Luettu: 28.10.2023.

Git Commit Best Practices. Luettavissa: https://gist.github.com/luismts/495d982e8c5b1a0ced4a57cf3d93cf60. Luettu: 14.10.2023

Griffinths C., 02.10.2023. The Latest Cloud Computing Statistics (updated October 2023). Luettavissa: https://aag-it.com/the-latest-cloud-computing-statistics/. Luettu: 25.11.2023.

Heath, M. 18.06.2020. Which Database should I use in my Azure Serverless App? Luettavissa: https://markheath.net/post/azure-serverless-database. Luettu: 10.09.2023.

Hoory L.,Bottorff C. 10.08.2022. Agile Vs. Waterfall: Which Project Management Methodology Is Best For You? Luettavissa: https://www.forbes.com/advisor/business/agile-vs-waterfall-methodology/. Luettu: 28.11.2023.

Kim G., Behr K., Spafford G. 2018. The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win. Fifth Anniversary Edition. IT Revolution. Portland.

Kim, G., Humble, J., Debois, P., Willis, J. 2016. The DevOps Handbook excerpt “The Three Ways”. IT Revolution. Portland.

Microsoft Corporation. 2022. AZ-204 Developing Solutions for Microsoft Azure. Online ohjaajan vetämä koulutus. Helsinki.

Microsoft Corporation. 18.10.2022a. How do I create and manage resources in Azure? Luettavissa: https://learn.microsoft.com/en-us/azure/developer/intro/azure-developer-create-resources. Luettu: 24.10.2023.

Microsoft Corporation. 18.10.2022b. Hosting applications on Azure. URL: https://learn.microsoft.com/en-us/azure/developer/intro/hosting-apps-on-azure. Luettu: 24.10.2023.

Microsoft Corporation. 21.07.2023. Review your data options. Luettavissa: https://learn.microsoft.com/en-us/azure/architecture/guide/technology-choices/data-options. Luettu: 10.09.2023.

Microsoft Corporation. 14.09.2023. Azure Functions developer guide. Luettavissa: https://learn.microsoft.com/en-us/azure/azure-functions/functions-reference?tabs=blob&pivots=programming-language-javascript. Luettu: 20.09.2023.

Pletcher, S. 08.06.2023. AWS vs Azure vs GCP: The big 3 cloud providers compared. Luettavissa: https://www.pluralsight.com/resources/blog/cloud/aws-vs-azure-vs-gcp-the-big-3-cloud-providers-compared. Luettu: 25.11.2023.

Posey B. 10.11.2022. Top public cloud providers of 2023: A brief comparison. Luettavissa: https://www.techtarget.com/searchcloudcomputing/tip/Top-public-cloud-providers-A-brief-comparison. Luettu: 25.11.2023.

Schwarzmüller, M. 2022. React Key Concepts. Packt Publishing. E-kirja. Luettu: 10.10.2023.

Statista. 2023. Author hidden. Revenue of the public cloud market worldwide from 2019 to 2028. Luettavissa: https://www.statista.com/forecasts/963841/cloud-services-revenue-in-the-world. Luettu: 1.12.2023.

Theekshana, D. 14.07.2022. Deploy a React web application with Azure App Service. Luettavissa: https://dasuntheekshana.medium.com/deploy-a-react-web-application-with-azure-app-service-d73e64cda0dd. Luettu: 25.10.2023.

Tietoevry, 29.01.2020. TietoEVRY and Microsoft announce strategic partnership in public cloud services. Luettavissa: https://www.tietoevry.com/en/newsroom/all-news-and-releases/press-releases/2020/01/tietoevry-and-microsoft-announce-strategic-partnership-in-public-cloud-services/. Luettu: 25.11.2023.

YLE. 17.03.2022. Microsoft rakentaa Suomeen uuden datakes-kusa-lueen, tontit Espoossa ja Kirkkonummella – Fortumin asiakkaille päästö-töntä kauko-lämpöä. Luettavissa: https://yle.fi/a/3-12363078. Luettu: 1.12.2023.

9 Liitteet

Liite 1. Projektiin liittyvät linkit

https://github.com/YuliaKokorieva/bikes-project : projektin GitHub etävarasto.

https://bikes-app.azurewebsites.net/ : sovelluksen käyttöliittymän URL.

Liite 2. Putkistotiedoston sisältö

master_bikes-app_front.yml

```yaml
name: Front Deployment


env:
  REACT_APP_BASE_URL: ${{ secrets.REACT_APP_BASE_URL }}
  REACT_APP_GOOGLE_API_KEY : ${{secrets.REACT_APP_GOOGLE_API_KEY}}

on:
  push:
    branches:
      - master
    paths:
      - 'BikeRidesFront/**'

# on: workflow_dispatch

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout repository
      uses: actions/checkout@v2

    - name: Set up Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '16'

    - name: Install dependencies
      run: |
        cd BikeRidesFront
        npm ci

    - name: Build
      run: |
        cd BikeRidesFront
        npm run build

    - name: Archive production build
      uses: actions/upload-artifact@v2
      with:
        name: build
        path: BikeRidesFront/build
```

```
    - name: Deploy to Azure Web App
      uses: azure/webapps-deploy@v2
      with:
        app-name: 'bikes-app'
        publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
        package: BikeRidesFront/build/
```