LAB University of Applied Sciences

Nopean toimituksen sovelluskehitys BaaS ympäristössä

LAB-ammattikorkeakoulu
Tradenomi (AMK)
2022
Samu Kaijansinkko, Juho Turkki

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Samu Kaijansinkko Juho Turkki	Opinnäytetyö, AMK	2022
	Sivumäärä 32	

Työn nimi

Nopean toimituksen sovelluskehitys BaaS ympäristössä

Tutkinto ja koulutusala

Tradenomi (AMK)

Toimeksiantajaorganisaatio

Suomen Vitamiinikeskus Oy

Tiivistelmä

Opinnäytetyön tavoitteena oli toteuttaa myyntikisasovellus palveluna ja selvittää, kuinka hyvin Google Firebase toimii alustana ohjelmiston backend-kehitykseen ja sovelluksen julkaisuun.

Opinnäytetyön toimeksiantaja oli Suomen Vitamiinikeskus Oy. Tarkoituksena oli digitalisoida yrityksen sisäinen myyntikisajärjestelmä käyttäen moderneja web-teknologioita.

Tehdyn työn perusteella voidaan todeta, että Google Firebase soveltuu nopean toimituksen sovelluskehitykseen. Työn tuloksena toimeksiantaja sai käyttöönsä toimivan, vaatimuksien mukaisen myyntikisajärjestelmän.

Asiasanat

Serverless, sovelluskehitys, Google Firebase

Abstract

Author(s)	Type of Publication	Published
Samu Kaijansinkko Juho Turkki	Thesis, UAS	2022
	Number of Pages 32	

Title of Publication

Fast delivery application development in BaaS environment

Degree, Field of Study

Business Administration (UAS)

Organisation of the client

Suomen Vitamiinikeskus Oy

Abstract

The aim of the thesis was to implement a sales competition application as a service and to find out how well Google Firebase works as a platform for the application's backend development and hosting.

The client of the thesis was Suomen Vitamiinikeskus Oy. The purpose was to digitize the company's internal sales competition system using modern web technologies.

Based on the thesis, it can be concluded that Google Firebase is suitable for fast delivery application development. End product of this work was a sales competition system which was published and had all the necessary functions.

Keywords

Serverless, web development, Google Firebase

Sisällys

1 Johdanto ... 1
2 Vaatimukset ... 3
2.1 Taustatilanne ... 3
2.2 Tekniset vaatimukset ... 3
2.3 CI/CD – jatkuva integrointi ja julkaisu ... 5
3 Teknologiakokonaisuus ... 7
3.1 Ohjelmistokokonaisuus ... 7
3.2 Backendin kuvaus ... 7
3.3 Ohjelmiston perustoiminnot ... 8
3.4 Firebase ... 8
3.5 NoSQL ... 10
4 Toteutus ... 12
4.1 Projektin lisääminen Firebase-alustalle ... 12
4.2 Kehitysympäristö ... 12
4.3 Autentikointi ... 13
4.3.1 Käyttöönotto ... 13
4.3.2 Rekisteröinti ... 14
4.3.3 Sisäänkirjautuminen ja tilan hallinta ... 15
4.4 Firestore ja rajapinta ... 15
4.5 Storage ... 18
4.6 Funktiot ... 19
5 Analysointi ... 22
5.1 Analysoinnin tavoite ... 22
5.2 Vaihtoehtoiset teknologiavalinnat ... 22
5.3 Virheiden vähentäminen ... 23
5.4 Resurssien vapautus ... 24
5.5 Aikataulussa pysyminen ... 24
5.6 Sovelluksen toimiminen ... 25
5.7 Käyttäjäkokemus ... 26
5.8 Asiakkaan palaute ... 27
5.9 Omat havainnot ... 28
6 Yhteenveto ja pohdinta ... 31
Lähteet ... 32

Lyhenteet

API	Ohjelmistorajapinta (Application Programming Interface)
CI/CD	Jatkuva integrointi / julkaisu (Continuous Integration / Continuous Deployment)
CLI	Komentorivi (Command Line Interface)
CRUD	Lisää, Lue, Päivitä, Poista (Create, Read, Update, Delete)
HTTP	Verkkoliikenneprotokolla (Hypertext Transfer Protocol)
HTTPS	Suojattu verkkoliikenneprotokolla (Hypertext Transfer Protocol Secure)
ID	Tunniste (Identifier)
JSON	Tiedostotyyppi (JavaScript Object Notation)
PWA	Progressiivinen verkkosovellus (Progressive Web Application)
REST	Ohjelmointirajapintojen arkkitehtuurityyli (Representational State Transfer)
SaaS	Sovellus palveluna (Software as a Service)
SDK	Ohjelmistokehityksen työkalukokoelma (Software Development Kit)
SMTP	Sähköpostiprotokolla (Simple Mail Transfer Protocol)
SQL	Kyselykieli (Structured Query Language)
URL	Tietyn tiedon paikan kertova merkkijono (Uniform Resource Locator)
XML	Merkintäkielistandardi (eXtensible Markup Language)

1 Johdanto

Tämän opinnäytetyön tavoitteena on toteuttaa myyntikisasovellus palveluna ja selvittää, kuinka hyvin Google Firebase toimii alustana ohjelmiston backend-kehitykseen ja sovelluksen julkaisuun. Toimeksiantajana toimii Suomen Vitamiinikeskus Oy. Yritys on perustettu vuonna 2009. Yrityksen kotipaikkana toimii Kuopio ja toiminta-alueena on koko Suomi. Yritystoiminta perustuu luontaistuotteiden vähittäiskauppaan.

Toimeksiantoon johti tarve, jossa manuaalisesti tehtävä työ haluttiin muuttaa digitaaliseen muotoon. Tämän avulla halutaan vähentää inhimillisiä virheitä ja vapauttaa työntekijöiden resursseja. Toimeksianto sisältää koko sovelluskehityksen elinkaaren, mutta tässä opinnäytetyössä keskitytään sovelluksen backend-puolen tutkimus- ja kehitystyöhön. Tavoitteena on, että Suomen Vitamiinikeskus Oy voi jatkossa käyttää sovellusta erilaisten myyntikisojen järjestämiseen.

Luvussa kaksi käydään läpi toimeksiantoon johtaneita tarpeita ja sovellukselle asetettavia vaatimuksia. Luvussa on mukana kuvaus ja esimerkkejä aiemmista keinoista pitää myyntikisoja yrityksen sisällä. Näiden esimerkkien avulla tämän raportin kirjoittajat toivovat, että lukija ymmärtää vaatimukset paremmin. Vaatimusten ymmärtäminen on tärkeää teknologiavalintaa tehdessä, jotta ei tule tilannetta vastaan, että ei voi jotain asiaa suorittaa heikon teknologiavalinnan vuoksi. Vaatimukset vaikuttavat sovelluksen kehityksessä käytettäviin teknologiavalintoihin.

Luvussa kolme siirrytään teknologiavalinnan kuvaukseen. Teknologiavalinnassa päädytään Google Firebase -sovelluskehitysalustaan. Tämän alustan uskotaan säästävän aikaa ja kykenevän ketterämpään ja nopeampaan ohjelmistokehitykseen verrattuna itse kehitettyyn backendiin. Luvussa käydään läpi Firebaseen ja projektiin liittyen keskeisintä teoriatietoa.

Neljännessä luvussa käydään tarkemmin läpi kehitystyön vaiheita ja sitä, kuinka Google Firebasea käytettiin osana kehitysympäristöä. Lukija johdatellaan luvussa aiheesta seuraavaan mahdollisimman pitkälle samassa järjestyksessä kuin kehitystyö on toteutunut. Tämän luvun jälkeen lukijalla on karkea käsitys siitä, miten Google Firebasen käyttöönotto tapahtuu. Käyttöönoton lisäksi luvussa käsitellään backendille eli taustapalvelimelle ominaisia asioita kuten käyttäjänhallintaa, tietokantaa ja yhteyden muodostamista frontend puoleen. Näiden lisäksi luvun lopussa on esitys Firebase Functions ja Extensions -toimintojen käytöstä sovelluksen yhteydessä.

Luvussa viisi analysoidaan toteutunutta työtä ja käydään läpi asiakkaan kokemuksia sekä palautetta. Analysointi kohdistuu siihen, miten yhteensopiva valittu teknologiaratkaisu oli toimeksiannolle ja mitä sen avulla saavutettiin. Myös havaitut heikkoudet tuodaan ilmi

viidennessä luvussa. Mukana on toimeksiantajan haastattelussa nousseita aiheita ja esimerkkejä, jotka toimivat vertailukohtana opinnäytetyön tekijöiden omaan pohdintaan.

Luku kuusi sisältää opinnäytetyön kirjoittajien yhteenvedon ja vastauksen tutkimuskysymykseen; soveltuuko Google Firebase pienien ja nopeasti kehitettävien ohjelmistoprojektien työkaluksi.

2 Vaatimukset

2.1 Taustatilanne

Suomen Vitamiinikeskus Oy järjestää työntekijöiden kannustamiseen erilaisia myyntikisoja, joissa tehtyjen myyntien perusteella on mahdollista ostaa palkintoja myynnistä saaduilla pisteillä. Tähän mennessä pisteiden ja palkintojen hallinta on toteutettu paperilla ja kommunikointi suullisesti, sähköpostilla tai WhatsApp-viestitse, joten järjestelmä ei ole aina reaaliaikainen ja virheiden mahdollisuus on suhteellisen suuri.

Toimeksiannon tarkoituksena on nykyaikaistaa järjestelmää, jonka vuoksi tämä opinnäytetyö on tehty. Tavoitteena on digitalisoida myyntikisan prosessi kokonaan, jolloin palvelu on reaaliaikainen ja palkintojen ostojen käsittely ei ole enää riippuvainen ihmisestä. Tämä vähentää prosessin virheiden mahdollisuutta.

2.2 Tekniset vaatimukset

Teknisiltä vaatimuksiltaan sovellus on yksinkertainen. Tietokantaan tallennettaviin tietoihin kuuluu käyttäjät, palkinnot ja tuotteet sekä sähköpostiviestit. Esimerkiksi myydyt tuotteet ja ostetut palkinnot voidaan tallentaa taulukkomuodossa käyttäjän tietoihin. Vaatimusten mukainen tietokantaliikenne saadaan toteutettua toiminnoilla lisää, lue, päivitä ja poista.

Käyttäjähallinnan kannalta on tärkeää, että sovelluksen käyttäjä pystyy rekisteröitymään sovelluksen käyttäjäksi sekä palauttamaan salasanan itsenäisesti. Salasanan kuuluisi olla ainoastaan käyttäjän itsensä tiedossa. Muutenkin tietoturva pitää taata niin, että henkilötietojen käsittely tapahtuu sovellettavan lain ja toimitussopimuksen mukaisesti.

Teknistä ratkaisua valitessa on otettava huomioon sovelluksen backendin nopea rakentaminen niin, että REST-rajapinta toteutuu, mikäli valittu teknologia sallii sen käyttämisen. REST-rajapintaa käytetään yleisesti, sillä se helpottaa frontendin kehitystä, kun voidaan olettaa, mistä ja millä tavalla tieto haetaan backendistä. REST-rajapinnalla tarkoitetaan ohjelmallista rajapintaa, jolla tietoja saadaan tuotua ja vietyä sisään yksinkertaisessa JSON-muodossa. REST ei ole varsinainen standardi, vaan yleinen tapa tietojärjestelmien tiedonvaihtoon. (Fielding & Taylor 2000.) Rajapinnat ja REST-koulutusmateriaalin mukaan REST-rajapinnoilla on tyypillisesti kolme ominaisuutta. Ensimmäinen näistä ominaisuuksista on juuriosoite resurssien käsittelyyn. Toisena resurssien erityismuodon määrittelevä mediatyyppi, mikä voi olla esimerkiksi HTML tai JSON. Tämä kertoo asiakkaalle, miten resurssiin liittyvä data tulee käsitellä. Kolmantena REST-rajapinnoille tyypillisenä asiana artikkelissa mainitaan HTTP-protokolla, jonka avulla

resursseja voidaan käsitellä protokollalle tyypillisillä metodeilla GET, POST, PUT ja DELETE. (Lokesh Gupta 2021.)

Yksi sovellukselle asetetuista vaatimuksista, joka koskee backendin suunnittelua on sähköpostin lähettäminen ennalta määritetylle henkilölle. Kun myyjä on kerännyt riittävästi pisteitä ja on oikeutettu ostamaan ansaitsemillaan pisteillä palkinnon, pitää palkinnon ostaminen hyväksyä. Hyväksymisen suorittaa myyjälle osoitettu toimistopäällikkö. Hyväksyminen tulee vaatimusten mukaisesti tapahtumaan sovelluksen sisällä. Jokaisesta uudesta hyväksymispyynnöstä pitää lähettää ilmoitus sähköpostitse.

Ennen kehityksen aloittamista vaatimuksista ja toimintojen kuvauksista laadittiin vaatimusluettelo. Toimituksen edellytyksenä on, että sovelluksessa voidaan suorittaa listatut toimenpiteet. Toimenpiteet on listattu kuvassa 1.

Toiminto / kuvaus:
Myynneistä kerätään pisteitä, joilla voidaan ostaa palkintoja.
Käyttäjä pystyy lisäämään myyntinsä, jonka kautta kertyneet pisteet lasketaan.
Järjestelmä tarkistaa, että käyttäjällä on oikean verran pisteitä käyttääkseen pisteet palkinnon ostoon.
Palkintojen lisäyksessä kerättäviä tietoja: nimi, kuvaus, tiedot, määrä, hinta.
Kaikki saatavilla olevat palkinnot listataan palkintonäkymässä.
Palkintoja voi lisätä, muokata ja poistaa pääkäyttäjän toimesta.
Käyttäjänhallinta, sisään- ja uloskirjautuminen
Järjestelmään mahdollista lisätä käyttäjiä.
Käyttäjäroolit: pääkäyttäjä, toimistopäällikkö, myyjä. Pääkäyttäjä hallinnoi rooleja.
Käyttäjän tietoja mm: nimi, sähköposti, kaupunki, mahdolliset palkinnot, pistemäärä/saldo.
Toimistopäälliköllä oikeus poistaa käyttäjiä.
Pääkäyttäjä voi lisätä myytävän tuotteen.
Pääkäyttäjä voi muokata myytävää tuotetta.
Pääkäyttäjä voi poistaa myytävän tuotteen.
Myytävän tuotteen tietoja: nimi, kuinka pitkä jakso, paljonko saa pisteitä, uusmyynti vai nykyasiakas (vaikuttaa pisteisiin), tuotteen myyntihinta.
Sovellus kehitetään toimimaan ensisijaisesti google chrome selaimella.
Sovellus kehitetään lähtökohtaisesti tietokoneen ruudulle, mutta mobiiliresponsiivisuus huomioidaan parhaan mukaan. Sovelluksen on kuitenkin toimivattava myös mobiilissa.

Kuva 1. Vaatimusluettelo

Ilmarisen ja Koskelan (2015, 121) mukaan itsepalvelu on yrityksille myös tärkeä tehokkuuden lisääjä, kun tehtäviä on siirtynyt ja siirtyy edelleen asiakkaiden hoidettavaksi digitaalisesti. Tässä tapauksessa yrityksen työntekijät luetaan esimerkin asiakkaiksi. Myyntikisan prosessin manuaalinen työ siirretään myyjien tehtäväksi digitaalisessa muodossa. Sovelluksen odotetaan tehostavan työskentelyä ja sitä kautta vaikuttavan positiivisesti tuloksiin.

2.3 CI/CD – jatkuva integrointi ja julkaisu

Vaatimusten mukainen sovellus ei tule tarvitsemaan suurta palvelintehoa tai tietokantaa ja tietoliikenteen määrä on hyvin ennakoitavissa. Toimeksiantaja toivoo, että sovellusta olisi jatkossa mahdollista päivittää ja tarpeen vaatiessa sovellukseen tehtävät muutokset saataisiin nopeasti sovelluksen käytössä olevaan versioon.

Sovelluksen muutoksia voidaan tarkastella GitHubiin tallennettavan, ja sen kautta ylläpidettävän, versionhallinnan avulla, joka jää toimeksiannonkin jälkeen toimittajan haltuun. Tuotannossa olevasta sovelluksesta voidaan tehdä sivuhaara, jota käytetään uuden päivityksen kehittämiseen. Kun kehitystyö on valmis, voidaan sivuhaara liittää päähaaraan ja siirtää muutokset tuotantoon.

Uuden tuotantoversion julkaiseminen tapahtuu helposti ja nopeasti Firebasen tarjoaman Firebase CLI:n avulla käyttäen Firebasen Hosting-palvelua, joka kuuluu Firebasen perusominaisuuksiin. Firebase CLI:n käyttöönoton aloittaminen on jaettu kolmeen vaiheeseen, jonka ensimmäisessä vaiheessa ladataan palvelun uusin versio. Uuden version lataaminen onnistuu kehitysympäristön konsolissa komennolla npm install -g firebase tools. Tämä komento mahdollistaa firebase-komennon käyttämisen konsolissa. Komennolla voidaan suorittaa monia eri toimintoja suoraan kehitysympäristössä. Kuvassa 2 on esimerkki firebase-komennon käyttämisestä. Kuvassa on käytetty komentoa firebase apps:list, joka listaa rekisteröidyt Firebase-sovellukset aktiivisessa projektissa. (Firebase a.)

```
samu-MBP-2:my-app samu$ firebase apps:list
✔ Preparing the list of your Firebase apps

┌──────────────────┬──────────────────────────────────────┬──────────┐
│ App Display Name │ App ID                               │ Platform │
├──────────────────┼──────────────────────────────────────┼──────────┤
│ svk-kisa         │ 1:10[redacted]799:web:99[redacted]48a │ WEB      │
└──────────────────┴──────────────────────────────────────┴──────────┘

1 app(s) total.
```

Kuva 2. Esimerkki komennon firebase käyttämisestä

Toiseen vaiheeseen kuuluu projektin alustaminen. Paikalliset projektitiedostot saadaan yhdistettyä Firebase-projektiin suorittamalla komento firebase init hosting paikallisen projektihakemiston juuressa.

Kolmannen vaiheen lopussa sovellus on julkaistu. Sovelluksen siirtäminen palvelimelle julkisesti saataville tapahtuu käyttämällä komentoa firebase deploy --only hosting. Jatkossa kolmas vaihe riittää uuden tuotantoversion siirtämiseksi palvelimelle. Julkaisemisessa on

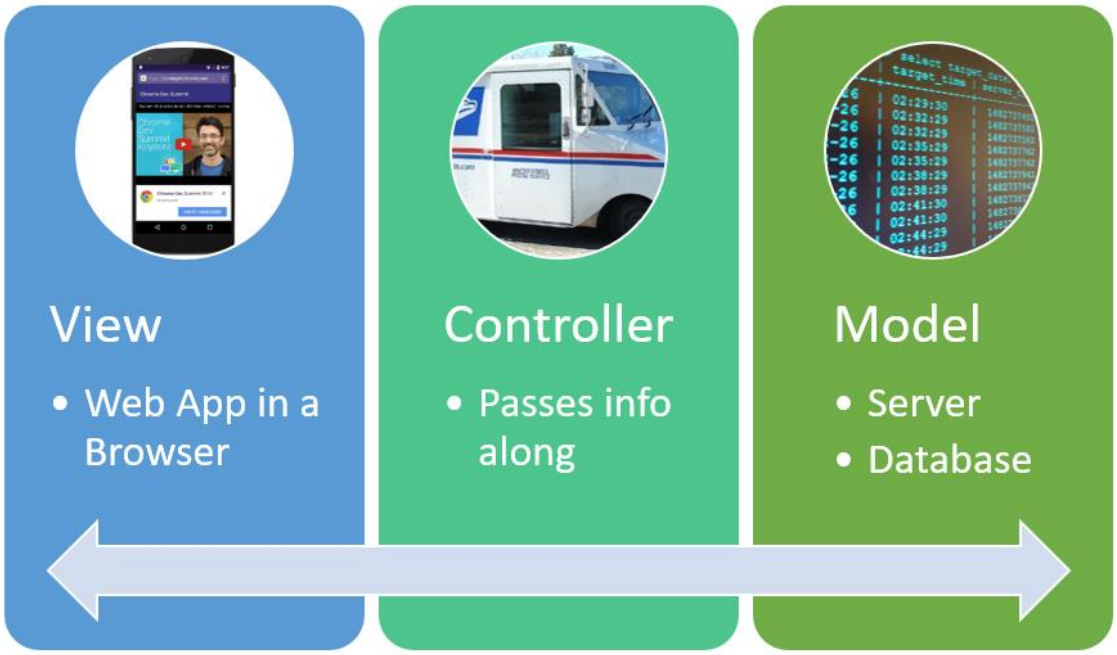
otettava huomioon käyttöympäristöt, sillä kehityksessä käytetään tiedostoa environment.ts ja julkaisussa environment.prod.ts.

On myös tärkeää ottaa huomioon tietojen säilyminen virhetilanteissa sekä alustan oleminen jatkuvasti saatavilla vähintään normaaliin työaikaan. Firebasen (2020b) palvelukuvauksen mukaan verkkoisännöinti ja reaaliaikainen tietokanta pyritään pitämään 99,95 prosenttisesti saavutettavissa. Varmuuskopio on mahdollista toteuttaa Firebasen Firestore -tietokannassa viemällä tietokanta Googlen Cloud Storagessa sijaitsevaan Bucketiin. Tässä kohtaa ei tarvitse tehdä varmuuskopioinnista automaattista, vaan riittää, että tiedot varmuuskopioidaan noin viikoittain tekemällä varmuuskopiointi manuaalisesti.

3 Teknologiakokonaisuus

3.1 Ohjelmistokokonaisuus

Useimmiten verkkosovellukset rakennetaan kolmijakoiseen arkkitehtuuriin, joka sisältää kolme tärkeää tasoa: tieto, logiikka ja esitys. Yleinen tapa implementoida tämä malli verkkosovellukseen on käyttää MVC-mallia. Mallin kolme tasoa ovat Model, View ja Controller, joista Model käsittelee tiedon manipulointia ja View näkyvää osaa, ja Controller vastaa sovelluksen ja käyttäjän tapahtumiin komentamalla kahta muuta tasoa (Havis ym. 2016, 4). Kuvassa 3 on visualisointi MVC-mallin toiminnasta.



Kuva 3. MVC-arkkitehtuurin kuvaus (Educative 2022)

3.2 Backendin kuvaus

Backend muodostuu palvelimella ajettavasta koodista, joka vastaanottaa pyyntöjä asiakkailta, ja sisältää logiikan asianmukaisten tietojen lähettämiseksi takaisin asiakkaalle. Backendiin kuuluu myös tietokantayhteyttä ja tietokantaa yleisesti manipuloivan koodin. Backend sisältää siis kaiken koodin, jota tarvitaan saapuvan pyynnön käsittelemiseen ja vastauksen luomiseen ja lähettämiseen asiakkaalle. (Codeacademy 2022a).

Iso osa koodia ja tiedon muokkausta on mahdollista suorittaa sekä frontendin että backendin puolella. Yksinkertaisimmillaan backend voi toimia vain tiedonvälittäjänä sovelluksesta tietokantaan. Sovellus on myös mahdollista rakentaa niin, että frontendin

puolella muokataan mahdollisimman vähän tietoa ja myös osa toiminnoista sijaitsee backendissä, josta niitä voidaan kutsua. Backend toimii sovelluksessa isona tietoturvan tekijänä ja riskinä.

3.3 Ohjelmiston perustoiminnot

Sovellusliittymiä rakennettaessa halutaan Model-tasossa olevan neljä perustoimintoa. Mallin on voitava luoda, lukea, päivittää ja poistaa resursseja. Tietojenkäsittelytieteilijät viittaavat näihin toimintoihin usein lyhenteellä CRUD. Mallin tulee pystyä suorittamaan nämä neljä toimintoa, jotta se olisi täydellinen. Jos toimintoa ei voida kuvata jollakin näistä neljästä operaatiosta, sen pitäisi mahdollisesti olla oma mallinsa. (Codeacademy 2022b).

CRUD-paradigma on yleinen verkkosovellusten rakentamisessa. Se tarjoaa mieleenpainuvan kehyksen muistuttamaan kehittäjiä täydellisten, käyttökelpoisten mallien rakentamisesta. Luo on funktio, joka koostuu toiminnallisuuksista, kun uutta tietuetta lisätään tietokantaan. Kun tämän toiminnon suorittaa, kuuluisi tietokannasta löytyä uusi tietue. Lue koostuu toiminnosta, joka kutsuttaisiin, kun halutaan katsoa kaikki luettelossa tällä hetkellä olevat tietueet. Tämä funktiokutsu ei muuta luettelossa olevia tietueita, vaan yksinkertaisesti noutaa resurssin ja näyttää tulokset. On myös mahdollista hakea yksittäinen resurssi, esimerkiksi ID:n perusteella, joka palauttaa vain yhden tuloksen. Päivitä on toiminto, jota kutsutaan, kun tietoja on muutettava. Funktiota kutsuva ohjelma antaa uudet arvot. Funktiokutsun jälkeen resurssien vastaava merkintä sisältää uudet tai muokatut kentät. Poista on toiminto, jolla resurssi poistetaan luettelosta. Funktiota kutsuva ohjelma antaa yhden tai useamman arvon resurssin tunnistamiseksi, minkä jälkeen tämä resurssi poistetaan luettelosta. Kun tämä toiminto on kutsuttu, luettelon tulee sisältää kaikki resurssit, jotka sillä oli aiemmin, paitsi juuri poistettu. (Codeacademy 2022b).

3.4 Firebase

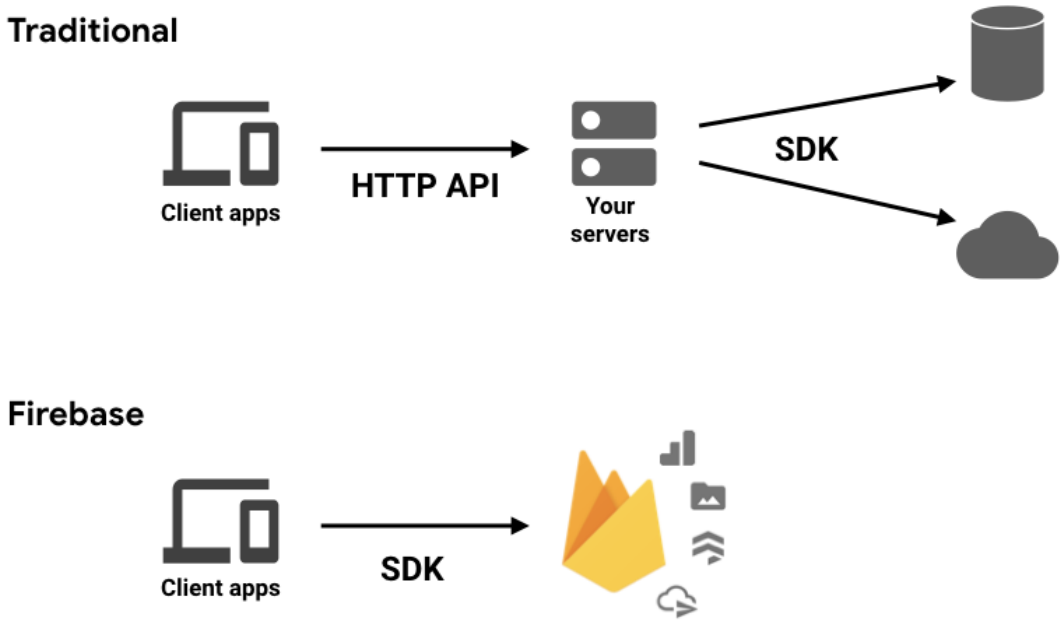
Domesin (2017, 10) mukaan progressiivisen verkkosovelluksen eli PWA:n rakentaminen on suurimmaksi osaksi käyttöliittymäprosessi. PWA:t myös välittävät vähän siitä, miten paljon dataa kuluu. Tässä projektissa on tarkoituksena pitää sovelluksen backend minimissä. Vaatimuksia läpikäydessä vahvistui ajatus toteuttaa sovelluksen julkaisu, tietokantaratkaisu ja backendin rakentaminen Googlen Firebase-sovelluskehitysalustalle, joten toimeksiantoon valittiin Google Firebase, joka käytännössä mahdollistaa rajapintojen automaattisen muodostumisen ja toimii autentikoinnin käsittelijänä tietokannan ja sovelluksen välillä hyvin pienellä vaivalla konfiguroituna.

Firebase on Googlen projekti, joka on suunniteltu auttamaan kehittäjiä luomaan sovelluksia huolehtimatta taustainfrastruktuurista. Se toimii freemium-mallilla, joka perustuu pyyntöjen määrään, joihin taustajärjestelmän on vastattava, sekä projektin tarvitseman tallennustilan määrään. Freemium mallissa kuluttajat vaihtavat dataansa tai yksityisyyttään tuotteen ”maksuttomien” ominaisuuksien käyttömahdollisuuteen. Lisäksi osa kuluttajista maksaa rahallisen korvauksen joidenkin tuotteen ominaisuuksien käyttämisestä. (Sitra).

Toimeksiannon vaatimuksiin perustuen tällainen ratkaisu sopii täydellisesti, sillä tarkoitus on kehittää sovellus nopeasti. Nopeasta kehityksestä huolimatta Firebase takaa ohjelmiston skaalautuvuuden tarpeen vaatiessa. Yksi toimeksiannon vaatimista tärkeimmistä palveluista on reaaliaikainen tietokanta. Tämä tarkoittaa, että yhden käyttäjän tekemät muutokset välitetään automaattisesti kaikille käyttäjille. Sovellukseen ei tarvitse rakentaa uuden tiedon tarkistamista.

Firebasen avulla saadaan käyttöön muun muassa reaaliaikainen tietokanta, hosting-palvelu ja sisäänrakennettu autentikointi. Tämän lisäksi se tarjoaa käyttöön Googlen Cloud Funktiot, joiden avulla voimme suorittaa automaattisia komentoja vasteena tiettyihin tapahtumiin, kuten palkinnon ostamiseen. (Tanna & Singh 2018, 11–22).

Firebasen pitäisi säästä paljon aikaa verrattuna oman taustapalvelin/hosting-ratkaisun rakentamiseen. Tämä on ehdottomasti projektin kannalta tärkeä asia. Kuvasta 4 nähdään Firebasen ja oman taustapalvelin/hosting-ratkaisun ero. Kuvan 4 kohdan ”Traditional” keskimmäinen osio ”Your servers” sisältää perinteisesti backendin lähdekoodin.



Kuva 4. Perinteisen backendin vastaan Firebasen työn kulku (Medium 2018)

Mediumin (2018) mukaan Firebase-tuotteilla ei todellakaan ole mitään rajoituksia sovellusten tyypeille. On vain rajoituksia alustoille, joilla sitä voidaan käyttää. iOS ja Android ovat Firebase SDK:iden ensisijaiset kohteet, ja verkko-, Flutter-, Unity- ja C++-tuki on lisääntynyt. Useille kielille on myös saatavana Admin SDK, jota voidaan käyttää kaikkien tarvitsemien taustakomponenttien kanssa.

Näiden SDK:iden lisäksi on FirebaseUI-niminen kirjasto (Android, iOS, web), joka tarjoaa joukon hyödyllisiä apuohjelmia, jotka tekevät Firebasen kanssa kehittämisestä entistä helpompaa. On myös projekteja, kuten AngularFire, jotka käärivät verkko-SDK:t käytettäväksi Angularin kanssa. (Medium 2018).

3.5 NoSQL

NoSQL on dokumenttipohjainen tietokanta tai dokumenttivarasto, joka on suunniteltu dokumenttisuuntautuneen tiedon tallentamiseen, hakemiseen ja hallintaan. Asiakirjapohjaiset tietokannat ovat yksi NoSQL-tietokantojen pääluokista, ja termin "dokumenttisuuntautunut tietokanta" suosio on kasvanut itse NoSQL-termin käytön myötä. Asiakirjapohjaiset tietokannat ovat luonnostaan avainarvovaraston alaluokka, toinen NoSQL-tietokantakonsepti. Ero on tietojen käsittelytavassa; avainarvovarastossa tietojen katsotaan olevan tietokannan kannalta luonnostaan läpinäkymättömiä, kun taas dokumenttisuuntautunut järjestelmä luottaa dokumentin sisäiseen rakenteeseen poimiakseen metatietoja.

Asiakirjatietokannat eroavat vahvasti perinteisestä relaatiotietokannasta. Relaatiotietokannat tallentavat tiedot yleensä ohjelmoijan määrittämiin erillisiin taulukoihin, ja yksi objekti voidaan hajauttaa useiden taulukoiden kesken. Asiakirjatietokannat tallentavat kaikki tiedot tietystä objektista yhteen esiintymään tietokannassa, ja jokainen tallennettu objekti voi olla erilainen kuin toinen. Tämä eliminoi objekti-relaatiokartoituksen tarpeen, kun tietoja ladataan tietokantaan.

NoSQL on yleistyvä tietokanta ratkaisu, joka ei pohjaudu perinteisen relaatiotietokannan mukaan taulukoihin ja hierarkkisiin objekteihin, vaan tallentaa tiedon hierarkkisessa kokonaisuudessaan tavalliseen muotoon kuten JSON tai XML (Havis & Mejia & Onodi 2016, 75-76.) Yksinkertaistettuna NoSQL voi palauttaa tietokannalta JSON-muodossa olevan objektin ilman muokkaamatta palautettua tietoa tietokannasta, kun taas relaatiotietokanta voi hakea samaa tietoa kahdesta tai kolmesta eri taulusta. Kuvassa 5 havainnollistetaan esimerkin avulla eroa saman tiedon tallentamista relaatiotietokantaan ja NoSQL-tietokantaan.

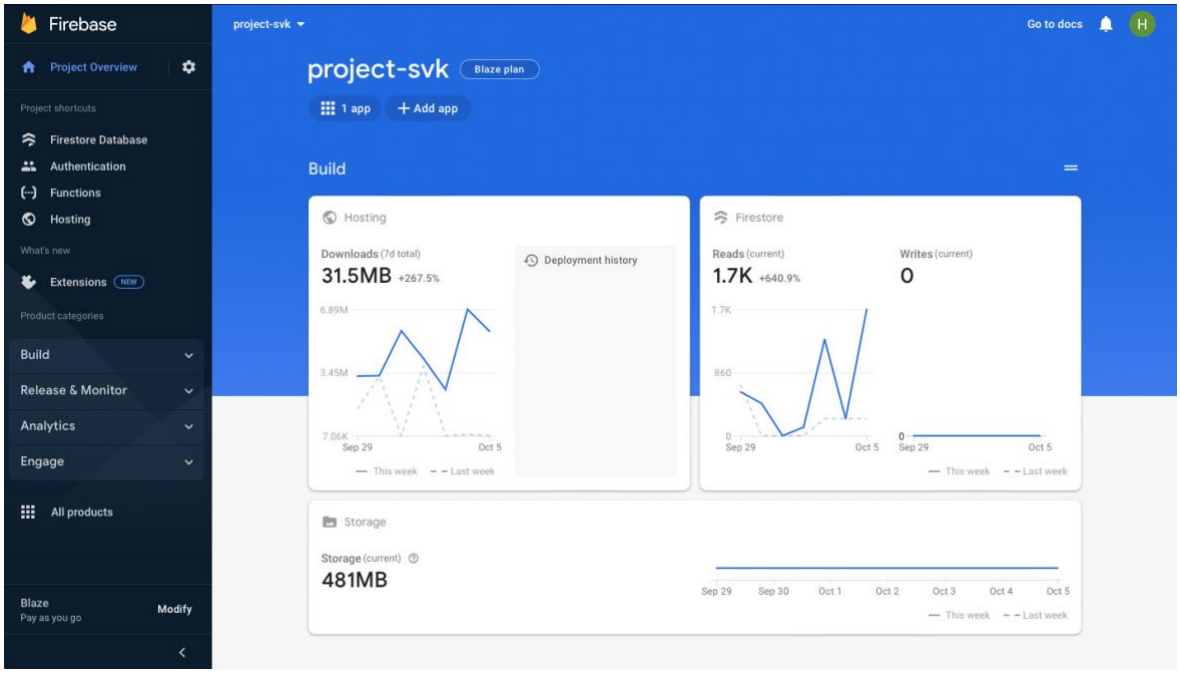
Kuva 5. Saman tiedon tallentaminen relaatiotiekantaan ja NoSQL-tietokantaan

JSON-dokumentti koostuu avain-arvo-pareista. Avainta vastaava arvo merkitään kaksoispisteellä erotettuna avaimen yhteyteen (Github 2022).

4 Toteutus

4.1 Projektin lisääminen Firebase-alustalle

Firebasen käyttöönotto aloitetaan kirjautumalla Google Firebase hallintapaneeliin sisälle. Hallintapaneelissa valitaan Add project ja syötetään projektin nimi. Alusta tarkistaa automaattisesti onko nimi varattu eikä luo uutta projektia, mikäli nimi on jo varattu. Kun projektille on annettu vapaana oleva nimi, siirrytään valintaruutuun, jossa projekti yhdistetään kirjautuneeseen käyttäjään. Projektin valitsemalla aukeaa kuvan 6 mukainen hallintapaneeli. Hallintapaneeli koskee valittua projektia ja yhdellä käyttäjällä voi olla monia eri projekteja tai olla osallisena muiden projekteihin. Firebase valittiin käyttämään Google Cloud Platformin eur3 (europe-west) resurssisijaintia.



Kuva 6. Valitun projektin hallintapaneeli

4.2 Kehitysympäristö

Tässä projektissa Google Firebasen käyttö ei vaatinut muutoksia kehitysympäristöön. Sovelluksen lähdekoodin rakentamiseen käytettiin JetBrainsin WebStorm nimistä alustaa, joka on integroitu kehitysympäristö JavaScriptille ja siihen liittyville teknologioille. Sen avulla pystytään suorittamaan Angularia, kuten oikealla palvelimella, mutta paikallisesti, ja npm-komentoja, joilla pystytään muun muassa integroimaan Firebasen vaatimia Node-paketteja projektiin.

Kehitystyötä tukevana työkaluna käytettiin Git-versionhallintaohjelmaa. Gitin avulla lähdekoodi voidaan jakaa erillisiin haaroihin, joita muokataan paikallisesti omalla tietokoneella. Tämä mahdollistaa sen, että useampi kehittäjä voi muokata lähdekoodia samanaikaisesti ilman, että se vaikuttaa muiden muutoksiin. Kun tarvittavat muutokset on tehty paikallisesti, ne voidaan siirtää päähaaraan. Päähaara on kaikkien kehittäjien käytössä. Muutosten siirtämisen jälkeen ajantasaisin versio on kaikkien käytössä.

4.3 Autentikointi

Firebase Authenticationin tavoitteena on tehdä turvallisten todennusjärjestelmien rakentamisesta helppoa ja samalla parantaa loppukäyttäjien kirjautumis- ja käyttöönottokokemusta. Se tarjoaa kokonaisvaltaisen identiteettiratkaisun, joka tukee sähköposti- ja salasanatilejä, puhelimen todennusta ja Google-, Twitter-, Facebook- ja GitHub-kirjautumista sekä monia muita kirjautumispalveluita. (Firebase c).

Firebase Authentication huolehtii sovelluksessa siitä, että käyttäjät kirjautuvat sisään ja ne pystytään tunnistamaan. Tämä tuote on välttämätön, jotta muut ominaisuudet määritetään oikein, varsinkin kun sovelluksessa on rajoitettava pääsyä käyttäjäkohtaisiin tietoihin. Firebase Authenticationin erikoisuus on se, että sen avulla on helppo suorittaa kirjautumisia, joiden rakentaminen itse vaatisi paljon enemmän aikaa ja työtä.

4.3.1 Käyttöönotto

Otetaan käyttöön Firebase konsolissa autentikointivalinta Email/Password. Rekisteröityneet käyttäjät tulevat näkyviin konsolin Authentication välilehdelle. Konsolin kautta käyttäjän tietoja ei pääse muokkaamaan, mutta sinne voi lisätä ja poistaa käyttäjän sekä lähettää sähköpostiin salasanan nollauslinkin. Konsolin kautta näkee käyttäjän sähköpostin, luomis- ja kirjautumispäivän ja käyttäjän tunnisteen, ID:n. Autentikointiin tallentuu kuitenkin muutakin tietoa, kuten displayName, johon pääsee käsiksi koodin kautta.

Sisäänkirjautumista ja rekisteröintiä varten otetaan käyttöön FirebaseUI-Angular niminen npm-paketti. Tämän avulla sisäänkirjautumista ja rekisteröintiä ei tarvitse itse ohjelmoida. Asennetaan Angulariin komennolla npm i firebaseui-angular. Määritetään paketin ominaisuudet luomalla muuttuja app.module.ts-tiedostossa kuvan 7 mukaisesti käyttämään oikeaa kirjautumistapaa. (Jenni).

```
const firebaseUiAuthConfig: firebaseui.auth.Config = {
  signInFlow: 'popup',
  signInOptions: [
    firebase.auth.EmailAuthProvider.PROVIDER_ID
  ], credentialHelper: firebaseui.auth.CredentialHelper.GOOGLE_YOLO
};
```

Kuva 7. FirebaseUI-Angular npm-paketin määritys app.module.ts-tiedostossa

4.3.2 Rekisteröinti

Rekisteröinti tapahtuu FirebaseUIn kautta ja sen elementti esitellään login-komponentin login.html tiedostossa kuvan 8 mukaisesti. Rekisteröinnin tapahtumat käsitellään komponentin ts-tiedostossa.

```
<firebase-ui (signInSuccessWithAuthResult)="login($event)"
             (signInFailure)="error($event)"
             (uiShown)="uiShownCallback()"></firebase-ui>
```

Kuva 8. FirebaseUIn näyttäminen komponentin login login.html sivulla

Onnistuneen rekisteröinnin jälkeen siirrytään tarkistamaan löytyykö käyttäjän tietoja tietokannasta. Tarkistetaan kuvassa 9 olevalla funktiolla, löytyykö käyttäjän sähköpostiosoitteella tietokenttää, jossa sähköpostiosoite täsmäisi. Mikäli käyttäjä löytyy, ohjataan käyttäjä etusivulle, muuten ohjataan käyttäjä täyttämään käyttäjän tiedot.

```
this.userService.getUserByEmail(user.email).then(r => {
      if (r.email) {
        this.userData = r;
        this.SetUserData(this.userData)
        this.router.navigate( commands: ['']);
      }
      else
        this.router.navigate( commands: ['new-user']);
    })
```

Kuva 9. Käyttäjän tietojen noutaminen

4.3.3 Sisäänkirjautuminen ja tilan hallinta

Sisäänkirjautuminen tapahtuu myös FirebaseUIn kautta login.html komponentissa käyttäen samaa elementtiä kuin rekisteröityminen. Ohjataan käyttäjä onnistuneen kirjautumisen jälkeen kirjautuneena etusivulle tai mikäli käyttäjän tietoja ei löydy tietokannasta, ohjataan käyttäjä new-user sivulle kuvan 9 mukaisesti täyttämään käyttäjätiedot.

Käytetään luokkaa AngularFireAuth muuttujalla afAuth avuksi kirjautuneen käyttäjän tilan hallintaan. Käyttäjän tiedot haetaan app.component.ts-tiedoston konstruktorissa kuvan 10 mukaisesti, jossa haetaan afAuth-muuttujalla kirjautunut käyttäjä Firebase autentikoinnista ja etsitään käyttäjän tiedot tietokannasta.

```
this.afAuth.authState.subscribe( next: (r : User | null ) => {
  if (r != null) {
    this.isLoggedIn = true;
      if (r.email) {
        this.userService.getUserByEmail(r.email).then(user => {
          this.user = user;
          this.userRewards = user.rewards;
          this.authService.SetUserData(this.user);
        })
```

Kuva 10. Sisäänkirjautumisen tilan ja käyttäjän tietojen hakeminen

Käyttäjän tietoja tarvitaan myös komponenttien sisällä. Otetaan käyttöön tietojen välitys @Input-toiminnolla lisäämällä komponentin viittaukseen [user]=”this.user”, jossa this.user on aikaisemmin määritelty käyttäjä. Komponentissa otetaan tieto vastaan @Input user: User() muuttujalla. Näin saadaan vähennettyä kyselyitä backendiin sekä vähennettyä toiston tarvetta koodissa.

4.4 Firestore ja rajapinta

Koska käytetään Firestore-tietokantaa, on lisättävä AngularFirestoren käyttöönotto kuvan 11 mukaisesti product.service.ts tiedoston konstruktoriin, joka määrittää yhteyden ja muodostaa sen sekä noutaa määritetyillä tiedoilla kyseisen tiedon reaaliaikaisesti. Yhteydenotto otetaan samalla tavalla käyttöön kaikissa service-tason tiedostoissa, joilla otetaan yhteys tietokantaan. Raportin selkeyttämiseksi on kuvattu vain yksi esimerkki jokaisesta rajapintatapahtumasta.

```
constructor() {
  initializeApp(environment.firebaseConfig);
  this.db = getFirestore();
  this.productCol = collection(this.db, path: 'products');

  // Get Realtime Data
  onSnapshot(this.productCol, onNext: (snapshot) => {
    this.updatedSnapshot.next(snapshot);
  }, onError: (err) => {
    console.log(err);
  })
}
```

Kuva 11. Firestore-tietokannan lisääminen konstruktoriin

Kun yhteys tietokantaan on määritetty, pystytään tekemään Firestore-rajapinnan kautta REST arkkitehtuurin mukaisia pyyntöjä palvelimelle. Käytetään esimerkkinä GET ja POST pyyntöjä. Kuvassa 12 on GET pyynnön suorittava koodi tiedostosta product.service.ts.

```
async getProducts() {
  const snapshot = await getDocs(this.productCol);
  return snapshot;
}
```

Kuva 12. REST arkkitehtuurin mukainen GET pyyntö palvelimelle

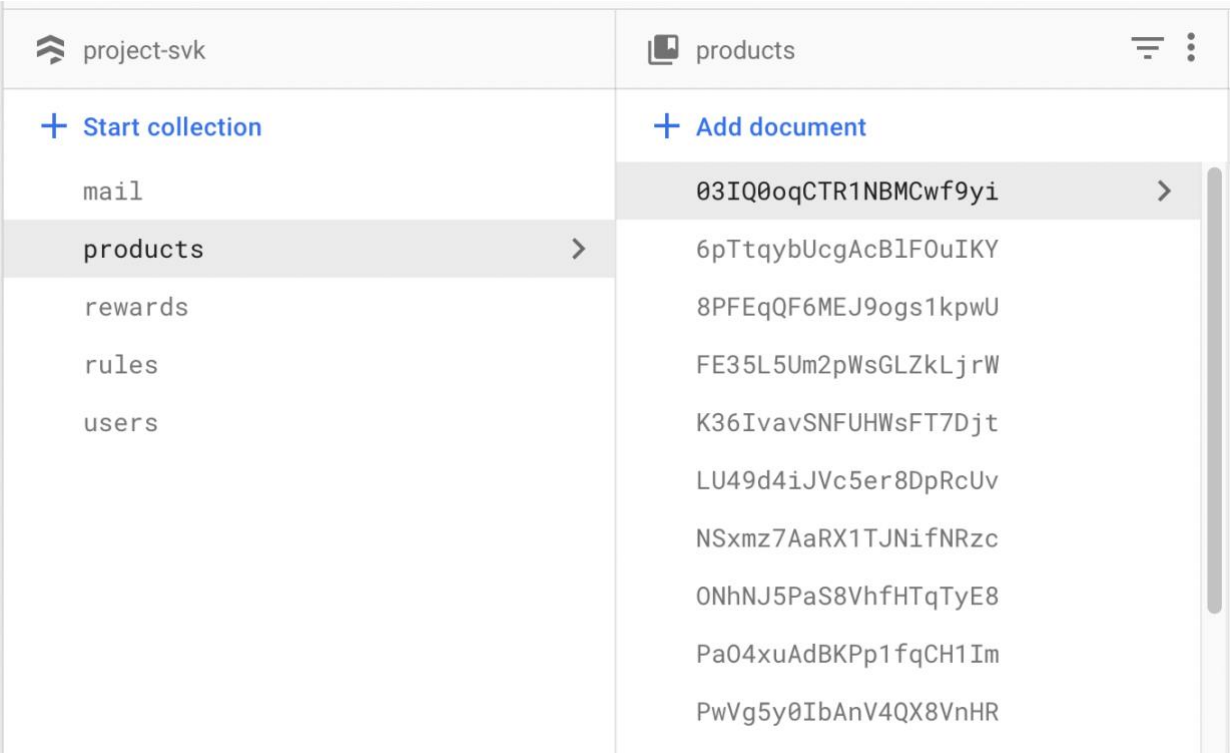
Haetaan tietokannasta useampaa tietuetta tekemällä pyyntö getProducts(). Tässä tapauksessa pyyntö hakee asynkronisesti listaa kaikista tuotteista. Pyynnöt voivat olla myös monimutkaisempia, mikäli pyynnölle on määritetty ehtoja. Ehdot tulee huomioida pyyntöä rakentaessa.

Kun tuotteita halutaan lisätä tietokantaan, käytetään sovelluksessa REST arkkitehtuurin mukaista POST pyyntöä. Kuvassa 13 on POST pyynnön suorittava koodi tiedostosta product.service.ts, joka lisää uuden tuotteen annetuilla tiedoilla.

```
async addProduct(name: string,
                 existingCustomerPoints: number,
                 newCustomerPoints: number,
                 orderLength: number,
                 price: number) {
  await addDoc(this.productCol, data: {
    name,
    existingCustomerPoints,
    newCustomerPoints,
    price,
    orderLength
  })
  return;
}
```

Kuva 13. REST arkkitehtuurin mukainen POST pyyntö palvelimelle

Tuotteen lisäyksen yhteydessä käyttäjä antaa tuotteelle arvon kenttiin: name, existingCustomerPoints, newCustomerPoints, orderLength ja price. Nämä arvot tallentuvat uudelle tuotteelle ja ovat jatkossa näkyvissä tietoa haettaessa. Huomioitavaa on, ettei käyttäjä määrittele identifioivaa tietoa kuten ’id’ kenttää tallennettavalle tuotteelle. Jotta tallennettavat tiedot ovat uniikkeja tietueita tietokannassa, tekee Firestore tämän työn kehittäjän puolesta. Jokaiselle tallennetulle tietueelle muodostuu automaattisesti yksilöitävä merkkijono. Kuvassa 14 on malli Cloud Firestoren konsolista, jossa products välilehti on avattuna ja alla on lista yksilöivistä merkkijonoista. Jokainen listan merkkijono on tässä tapauksessa yksi tuote.



Kuva 14. Products taulun yksilöivät merkkijonot

4.5 Storage

Saatavilla oleville palkinnoilla on tallennettava kuva. Käytetään Firebase Storagea kuvien tallentamiseen. Palkinnon tietoihin tietokannassa lisätään tietokentät image ja imageUrl, joihin tallennetaan tieto kuvan nimestä ja kuvan sijainti. Käytetään kuvan 15 mukaista metodia kuvan tallentamiseen palvelimelle. Metodiin päädytään käyttäjän valitessa ladattavan kuvan. Tämän jälkeen tarkistetaan, että kuva on saatavilla ja tallennetaan kuvan tiedoston nimi palkinnon kuva-tietokenttään, joka toimii kuvan nimenä. Seuraavaksi luodaan referenssi Firebase Storageen ja ladataan kuva. Lopuksi ilmoitetaan käyttäjälle, että kuvan lataus onnistui.

```
upload(event: Event) {
  // @ts-ignore
  if(event.target.files){
    // @ts-ignore
    this.reward.image = event.target.files[0].name
    // @ts-ignore
    const storageRef = ref(this.storage, event.target.files[0].name);
    // @ts-ignore
    uploadBytes(storageRef, event.target.files[0]).then((snapshot : UploadResult ) => {
      this.openSnackBar( message: "Kuvan lataus onnistui!",  action: "OK");
    });
  }
}
```

Kuva 15. Kuvan lähetyksen metodi

Kuvan tallennuksen jälkeen on vielä päivitettävä palkinto. Kuvan latauksessa tallennettiin kuvan nimi palkintoon. Tämän lisäksi on otettava talteen kuvan verkkosijainti (Kuva 16). Tarkistetaan ensin, että palkinnolta löytyy tieto kuvan nimestä, jotta kuva voidaan hakea. Seuraavaksi tehdään asynkroninen pyyntö Firebase Storageen storage-referenssillä ja kuvan tiedostonimellä. Tällä pyynnöllä saadaan vastauksena kuvan verkkosijainnin URL, joka tallennetaan palkintoon. Prosessi saatetaan loppuun tallentamalla muutokset sovelluksessa, jonka jälkeen sovellus lähettää päivityspyynnön tietokantaan.

```
if (this.reward.image)
  await getDownloadURL(ref(this.storage, this.reward.image))
    .then((url : string ) => {
      rewardUpdate.imageUrl = url;
    })
```

Kuva 16. Kuvan verkkosijainnin noutaminen

4.6 Funktiot

Käytetään sähköpostien lähettämiseen Firebasen julkaisemaa lisäosaa Trigger Email. Otetaan se käyttöön Firebase konsolin Extensions-välilehdellä ja määritetään sen asetukset käyttämään ulkopuolista SMTP-palvelua, sillä se ei kuulu Firebasen tilaukseen. Lisäosa vaatii Firebase Blaze -tilauksen, sillä sen käytöstä syntyy arviolta $0,01 kuluja kuukaudessa, vaikka sitä ei käyttäisi (Firebase d).

Koodin puolella lisäosa otetaan käyttöön komennolla npm install -g firebase-tools. Sähköpostiviesti muokataan oikeaan muotoon sovelluksessa Node-paketti Nodemailerin avulla, jonka jälkeen se lähetetään tietokannassa olevaan mail-kollektioon.

Kun lähetetään uutta sähköpostia, muodostuu sähköpostiviesti uutena tietokenttänä Firestore tietokannassa lisäosan Trigger Email luomaan mail-kollektioon. Lisäosan funktio ext-firestore-send-email-processQueue huomaa uuden kentän tietokannassa automaattisesti ja alkaa käsittelemään tietoa. Kuvassa 17 näkyy lähtevän viestin tiedot, jotka tulevat sovelluksesta.

```
▼ message
    html: "<h1>Käyttäjän Haikea palkinto odottaa tarkistusta.</h1><p>Voit
           käydä hyväksymässä palkinnon kirjautumalla järjestelmään,
           valitsemalla hallinta ja käyttäjän kohdalla 'Tarkista'</p>
           <h4>Palkinnon tiedot:</h4><ul><li>Nimi: asd</li><li> Kuvaus: 4</li>
           </ul><br><br><a href='https://vitskuliiga.fi/admin'"
    subject: "Käyttäjän palkinto odottaa tarkistusta"
▼ to
    0 "juho.turkki@haikea.fi"
    1 "samu.kaijansinkko@haikea.fi"
```

Kuva 17. Lähtevän viestin tiedot, jotka välitetään sovelluksesta tietokantaan

Lisäosa Trigger Email lisää lähtevän viestin tietokenttään delivery-taulukon, joka sisältää tiedot viestin lähetyksen tilasta (Kuva 18). Viestin ollessa lähtevänä taulukon tiedon pending-tila on tosi ja mikäli viestin lähetys epäonnistuisi tai siinä tulisi jokin muu virhe, tulisi error tai rejected-kohtaan virheilmoitus. Onnistunut viestin lähetys näkyy state-kohdassa tilana SUCCESS.

```
▼ delivery
    attempts: 1
    endTime: October 7, 2022 at 4:10:29 PM UTC+3
    error: null
  ▼ info
    ▼ accepted
        0 "juho.turkki@haikea.fi"
        1 "samu.kaijansinkko@haikea.fi"
      messageId: "<60c30f79-1058-b7a0-ecd4-
                  7aef4b8e64f9@vitskuliiga.fi>"
    ▼ pending
    ▼ rejected
      response: "250 Message queued as <60c30f79-1058-b7a0-ecd4-
                 7aef4b8e64f9@vitskuliiga.fi>"
    leaseExpireTime: null
    startTime: October 7, 2022 at 4:10:26 PM UTC+3
    state: "SUCCESS"
```


Kuva 18. Sähköpostiviestin toimituksen tiedot

5 Analysointi

5.1 Analysoinnin tavoite

Analysoinnin tavoitteena on arvioida Google Firebase kehitystyökalun sopivuutta toimeksiannon mukaiseen sovelluskehitykseen. Analysoinnissa on käytetty hyväksi toimeksiantajan edustajan haastattelua 19.10.2022. Analysointi kattaa useamman osa-alueen sekä pohdintaa kehityksen onnistumisesta. Analysoinnin lähtökohta on jokaisen aiheen kohdalla pohtia sovelluksen backend toteutusta Google Firebasen avulla.

5.2 Vaihtoehtoiset teknologiavalinnat

Google Firebasen käyttäminen ei ollut suora valinta, vaan muitakin mahdollisuuksia harkittiin. Eri vaihtoehtojen välillä mietittiin niiden hyvät ja huonot puolet tämän projektin osalta, jonka jälkeen päädyttiin nykyiseen valintaan.

Yleinen nykyaikaisesti rakennettu verkkosovellus voi käyttää MEAN-ohjelmistokokonaisuutta. Kokonaisuuden kirjaimet tulevat siihen kuuluvista teknologioista, MongoDB (tietokanta), Express.js (ohjelmistokehys), Angular (frontend) ja Node.js (backend). Tässä toimeksiannossa käytetään Angularia frontendin rakentamiseen, joten mikäli backend päädyttäisiin rakentamaan itse, olisi kyseinen ohjelmistokokonaisuus sopiva vaihtoehto.

MEAN-ohjelmistokokonaisuudessa on hyvinä puolina muun muassa saman ohjelmistokielen, JavaScriptin käyttäminen ympäri ohjelmistokokonaisuutta ja nopeus. Kokonaisuuden nopeuteen vaikuttaa erityisesti Node.js:n ominaisuus, joka mahdollistaa useiden operaatioden samanaikaisen suorituksen. Tämä onnistuu keskeyttämättömän sisään/ulostulo järjestelmän ansiosta, jolloin sovellus ei pysähdy odottamaan levy- tai verkkopyyntöjen valmistumista, vaan jatkaa muuta suorittamista. Suoritettava operaatio ei estä toisen operaation suorittamista, sillä operaation valmistuessa sovellus käsittelee siitä seuraavan pyynnön. (OpenJS Foundation).

Toinen toimeksiantoon sopiva ohjelmistokokonaisuus olisi käyttää MySQL (tietokanta), PHP (backend) ja Angular (frontend). Tämän ratkaisun huonona puolena on, että palvelinympäristö ja PHP ohjelmistokielenä eivät ole opinnäytetyön tekijöille niin tuttuja kuin JavaScript. Lisäksi kyseessä on vanhempaa teknologiaa, minkä vaikutus voi tulla esille esimerkiksi skaalautuvuuden ja reaaliaikaisuuden kohdalla.

5.3 Virheiden vähentäminen

Sovelluksen valmistumisen myötä sovellusta on päästy testaamaan sen omassa käyttöympäristössä. Käytännön testaus on osoittanut sen, että inhimilliset virheet vähenevät. Tyypilliset tapahtuneet virheet ovat olleet vanhasta järjestelmästä ja inhimillisistä tekijöistä johtuvia. Esimerkkinä virheestä Suomen Vitamiinikeskuksen toimitusjohtaja Tomi Runsala (2022) pitää unohtamista. Myyjä on voinut ilmoittaa tehneensä pisteisiin oikeuttavan myynnin, mutta asiasta eteenpäin ilmoittamaan velvoitettu henkilö unohtaa välittää ajantasaisen tiedon muille. Runsalan mukaan samankaltaisia virheitä on sattunut useita. Hän kuvailee niitä harmillisiksi, sillä on voinut käydä niin, että palkintoon oikeutettu henkilö ei pystynytkään tekemään ostoa, koska tiedot eivät olleet päivittyneet muille ajoissa. Näin ollen joku muu on voinut ostaa palkinnon aiemmin, vaikka olisi tehnyt ostoon oikeuttavat myynnit myöhemmin. Toisena esimerkkinä Runsala kertoo tulostimen vioittumisen toimistolla, joka on voinut estää myyjiä näkemästä kyseisen päivän tilannetta.

Hyvä käyttökokemus virheiden vähentämisen osalta on saavutettu sen johdosta, että tieto liikkuu reaaliaikaisesti ja tietokone tekee laskutoimitukset ihmisen puolesta. Runsala on ollut tyytyväinen testauksen tuloksiin ja kuvailee sovelluksen täyttävän sille asetetut vaatimukset.
Sovelluksen kehityksessä on oltu tyytyväisiä Google Firebase -teknologiaratkaisuun virheiden vähentämisen osalta. Firebasen käyttöönotto on ollut ketterää, eikä sen käyttäminen ole estänyt vaatimusluettelon kaltaista toimintaa datan käsittelyssä.

Sovelluksen käyttämisestä tehtiin huomioita kehitystyön aikana. Huomioiden perusteella inhimillisiä virheitä ei voida estää kokonaan. Vaikka monien aiemmin tapahtuneiden virheiden uusiminen on miltei mahdotonta uudella järjestelmällä, on silti mahdollista, että käyttäjä tekee virheitä. Käyttäjän tekemä virhe voi olla esimerkiksi väärän tuotteen valinta myytyihin tuotteisiin. Google Firebasen avulla kehittäjät päätyivät ratkaisuun, että käyttäjän tekemistä palkintojen tilauksista lähetetään sähköpostitse varmistusviesti käyttäjän toimistopäällikölle sen lisäksi, että palkinnon saldo vähentyy yhdellä. Tällöin palkinnon voidaan sanoa käyttäjän varmistaneen palkinnon itselleen, kun taas vanhalla tavalla tehdessä ei pystytty luottamaan palkintojen reaaliaikaiseen saatavilla olevaan määrään.

Sovelluksen toimituksen ja käyttöönottotestauksen yhteydessä asiakkaalle on pidetty laaja perehdytys sovelluksen käyttöön liittyen. Näin on pyritty vähentämään virheiden tapahtumisen riskiä.

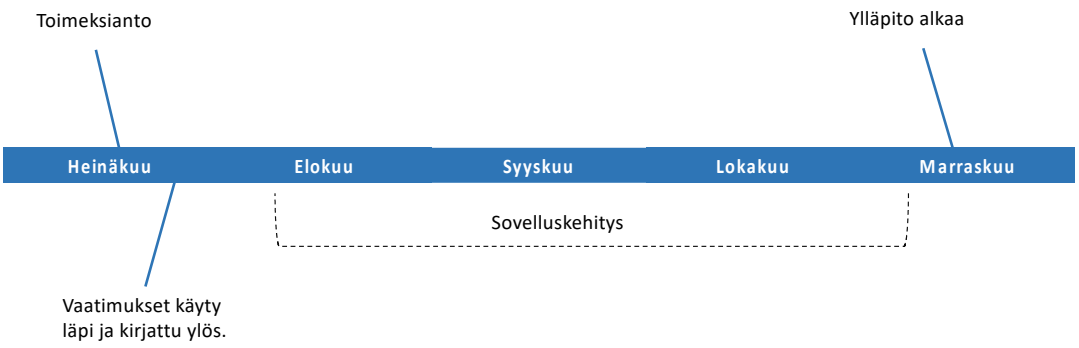
5.4 Resurssien vapautus

Runsalan (2022) arvion mukaan aiempina vuosina myyntikisat ovat työllistäneet toimistopäälliköitä useita tunteja viikossa. Tämän lisäksi hän mainitsee, että myös hänen omaa aikaansa on kulunut paljon, sillä päivitettyjä listoja ja palkintoesitteitä on pitänyt tehdä manuaalisesti ja lähettää ympäri Suomea sähköpostitse.

Päätös käyttää Google Firebasen tarjoamaa backend ratkaisua vaikuttaisi olleen hyvä valinta, sillä se auttoi säästämään resursseja myös kehitystyön aikana. Kehitystyön aikaisia resursseja onnistuttiin säästämään sillä, ettei omaa backendiä eli taustapalvelinjärjestelmää rakennettu.

5.5 Aikataulussa pysyminen

Tämän raportin kirjoittaminen sai alkunsa, kun sovelluskehityksen toimeksianto otettiin vastaan. Vaatimusten läpikäymisen myötä, luotiin aikataulu, jossa nojauduttiin Google Firebase -palvelun tarjoamiin ratkaisuihin. Sovelluskehitys edistyi suunniteltua aikataulua nopeammin, koska taustapalvelinjärjestelmää ei tarvinnut rakentaa itse. Kuvassa 19 on sovelluskehityksen suunniteltu aikataulu.



Kuva 19. Sovelluksen suunniteltu aikataulu

Firebasen oletuksena tietoturvalliset ratkaisut sopivat projektiin oletusten mukaisesti ja auttoivat nopeuttamaan sovelluksen kehittämistä. Tämä näkyi positiivisena yllätyksenä asiakkaalle, joka sai sovelluksen testikäyttöön etuajassa.

5.6 Sovelluksen toimiminen

Toimeksiannon sopimuksessa asiakkaan kanssa sovittiin, että sovelluksen toimiminen hyväksytysti mitataan yhdessä laaditun vaatimusluettelon mukaan. Vaatimukset on esitetty aiemmin tässä raportissa.

Erityinen huomio tehtiin käyttäjähallintaa rakentaessa. Firebasen käyttäjähallinta sisältää rekisteröinnin, autentikoinnin ja salasanan palauttamisen sekä kaiken muun, mitä nykyaikaiselta sovellukselta voi odottaa. Firebasen kattava käyttäjähallinta on oletuksena tietoturvallinen sekä nopea ottaa käyttöön.

Suunnitellusti toimivan käyttäjähallinnan rakentaminen toimi pohjana sovelluksen muiden vaatimusten toteuttamiselle. Kun pohja oli valmis, pystyttiin sovellukseen rakentamaan loput ehdollisista toiminnoista. Näistä yhtenä esimerkkinä toimii aiemmin esitetty käyttäjän tietojen löytyminen sähköpostin perusteella. Kun autentikoitu käyttäjä saadaan varmistettua, voidaan sovelluksessa näyttää ehdollisesti tietoja perustuen käyttäjän rooliin tai haettua käyttäjän omat tiedot hänen profiiliinsa.

Sovellus suunniteltiin käytettäväksi niin, että käytössä on useampi rooli. Tämä takasi asiakkaalle mahdollisuuden hierarkkiseen toimintaan, kuten rajoittaa käyttäjänhallintaa tai antaa siihen mahdollisuuden. Toteutus onnistui ja sen oletetaan vähentävän asiakkaan tarvetta konsultoida sovelluksen ylläpitäjää.

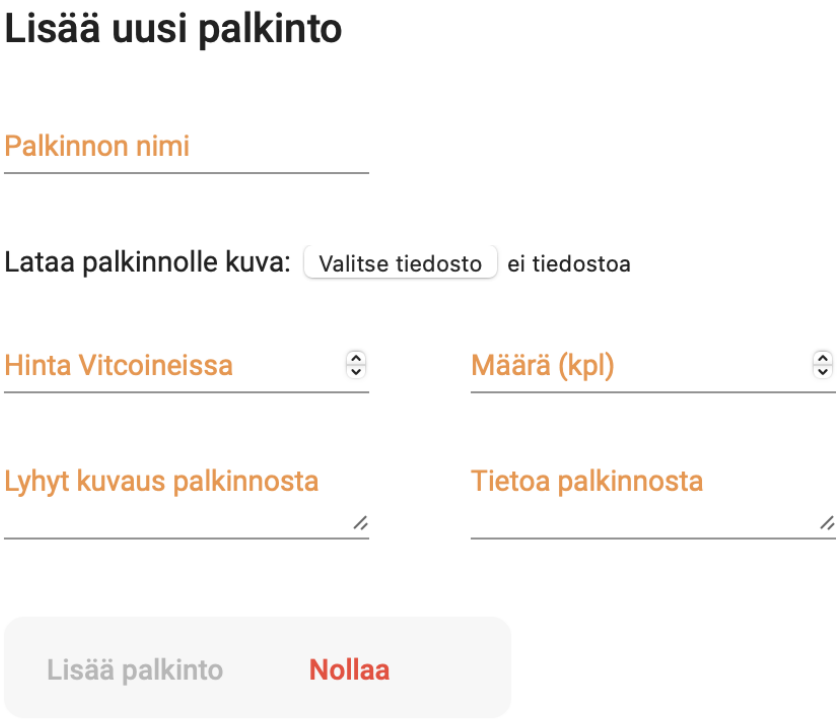
Firebasen avulla sovelluksen yhteys tietokantaan saatiin nopeasti rakennettua toimivaksi. Vaatimusluettelon mukaiset toiminnallisuudet rakennettiin suunnitelmien mukaisesti. Lähdekoodin service-tasolle rakennettiin tietokannan kanssa keskustelevat funktiot, jotka suorittavat autentikoinninnasta riippuen pyydetyn komennon. Kuvassa 20 on product.service.ts tiedoston funktiot nimettynä niin, että perinteiset CREATE, READ, UPDATE ja DELETE toiminnot ovat suoritettavissa.

```
async getProducts() {...}

async getProductById(id: string) {...}

async addProduct(name: string,
                 existingCustomerPoints: number,
                 newCustomerPoints: number,
                 orderLength: number,
                 price: number) {...}

async deleteProduct(docId: string) {...}

async updateProduct(docID: string, product: Products) {...}
}
```

Kuva 20. CRUD-toiminnallisuudet tuotteet-tietokannalle

5.7 Käyttäjäkokemus

Kun sovelluskehitys oli saatu vaatimusten mukaisesti valmiiksi, suoritettiin sovelluksen oikeassa käyttöympäristössä testaus. Asiakas järjesti viikon mittaisen, sovelluksen tarkoitukseen sopivan myyntikisan, ja käytti sovellusta, kuten sitä tuotantoympäristössä kuuluisikin käyttää. Asiakkaalle pidettiin kattava käyttökoulutus ennen testauksen aloittamista. Runsalan (2022) mukaan pääkäyttäjän toiminnot sujuivat suunnitellusti eikä sovelluksen tietoliikenteestä löytynyt virheitä. Kuvassa 21 on esitetty kentät, joihin pääkäyttäjä syöttää uuden palkinnon tiedot ennen sen tallentamista tietokantaan Firebasen rajapinnan avulla.



Kuva 21. Uuden palkinnon lisääminen

Myös sovelluksen loppukäyttäjät eli myyjät kokivat sovelluksen toimivaksi, eivätkä havainneet testiviikon aikana virheitä tietoliikenteessä. Sovelluksessa ilmoitetut myynnit päivittyivät reaaliaikaisesti myyjän henkilökohtaiseen profiiliin. Myyjällä oli ajantasaisen tieto omasta piste saldostaan. Runsalan mukaan myyjien palaute oli poikkeuksetta positiivista.

5.8 Asiakkaan palaute

Testiviikon jälkeen tavattiin toimeksiannon yhteyshenkilö, Suomen Vitamiinikeskus Oy:n toimitusjohtaja Tomi Runsalan. Tapaamisen tarkoituksena oli kerätä asiakaspalautetta sekä mahdollisia takuun alaisia korjauskohtia. Runsala itse kommentoi sovellusta seuraavasti:

> *Sovellus vaikuttaa testausjakson perusteella toimivan suunnitellusti enkä havainnut siinä virheitä. Muutaman seikan huomasimme käyttöliittymässä, mitä voisi jatkossa muokata, mutta nehän eivät vaatimuksiin kuuluneet, joten niihin voimme palata myöhemmin. Otamme sovelluksen mieluusti laajempaan käyttöön myyjien palautteen perusteella. On hienoa, ettei meidän jatkossa tarvitse kynän ja paperin kera pitää kirjaa tilanteesta vaan sovellus tekee työn puolestamme.*

Asiakkaalta saatu palaute sekä viikon testausjakso vakuuttivat opinnäytetyön tekijät Firebasen toimivuudesta.

5.9 Omat havainnot

Firebasen käyttö osana projektin teknologiakokonaisuutta osoittautui hyväksi ratkaisuksi. Sen avulla saavutettiin nopea kehitystyö varmistaen samalla laadukas ja tietoturvallinen toteutus. Backend toteutuksen korvaamisen lisäksi Firebase tarjosi projektiin paljon muutakin, kuten tietokannan ja hosting palvelun. Näin ollen ylläpidettävät ympäristöt pystyttiin pitämään minimissä. Tämä helpottaa sovelluksen ylläpitoa jatkossa.

Google suurena alan toimijana on pitänyt huolen, että sen tarjoamallaan Firebase palvelulla on kattava ja ajantasainen dokumentaatio saatavilla. Tämän lisäksi Firebase vaikuttaisi olevan suosittu alan työntekijöiden keskuudessa, sillä kehitystyötä tukevaa tietoa ja dokumentaatiota löytyi mittavasti erilaisilta foorumeilta ja keskustelupalstoilta.

Firebasen suuresta suosiosta ja kattavasta dokumentaatiosta huolimatta, kaikkea ei osattu suunnitella etukäteen. Sovellusta kehitettäessä huomattiin käyttäjänhallintaa rakentaessa rajoittavia tekijöitä. Uutta käyttäjää rekisteröitäessä ei ole mahdollista lisätä tietokenttiä käyttäjälle, vaan käytössä on Firebasen määrittelemä käyttäjä ja sille asetettavat arvot. Yksi käyttäjän arvoista on ”metadata” niminen tietokenttä, joka nähtiin vaihtoehtoisena ratkaisuna, mutta arvion mukaan se olisi voinut jatkossa hankaloittaa sovelluksen logiikkaa ja tilalle täytyi löytää toinen vaihtoehto. Kuvassa 22 on esimerkki koodista, jossa haetaan rekisteröity käyttäjä ja osa tiedoista.

Web version 9 (modular) | Web version 8 (namespaced)

Learn more about the tree-shakeable Web v9 modular SDK and upgrade from version 8.

```
import { getAuth } from "firebase/auth";

const auth = getAuth();
const user = auth.currentUser;

if (user !== null) {
  user.providerData.forEach((profile) => {
    console.log("Sign-in provider: " + profile.providerId);
    console.log("  Provider-specific UID: " + profile.uid);
    console.log("  Name: " + profile.displayName);
    console.log("  Email: " + profile.email);
    console.log("  Photo URL: " + profile.photoURL);
  });
}
```

auth_get_user_profile_provider.js

Kuva 22. Käyttäjänhallinnan ohje verkkosovelluksissa (Firebase e)

Projektissa käytetylle NoSQL-tietokannalle tyypillisesti tarkoituksena oli tallentaa käyttäjän tietoihin yksittäisten tietueiden lisäksi taulukoita. Taulukot tulisivat sisältämään palkinto- ja myynti-tietueita kuvan 23 mukaisesti.

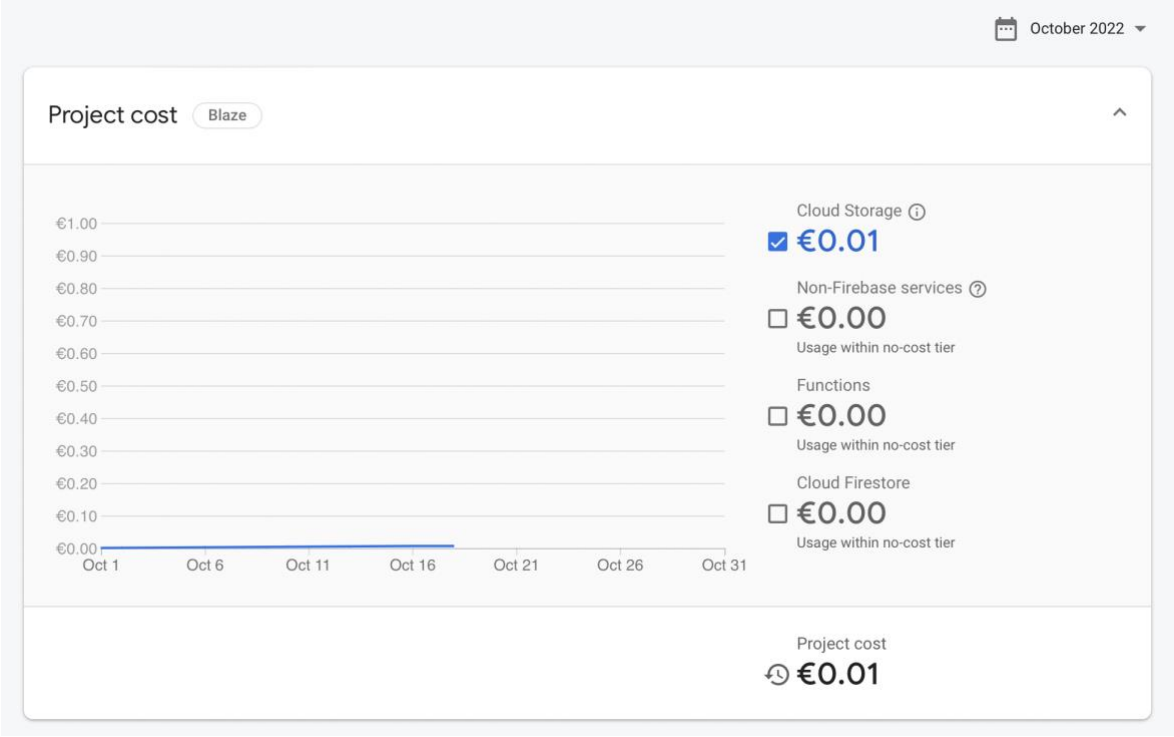
```
    balance: 0
    email: "info@haikea.fi"
    id: "JFkRbDSWSfTPP95qgZEZ"
    location: 1
    name: "Haikea"
▼   rewards
    role: 0
▼   sales
    totalPoints: 0
```

Kuva 23. Käyttäjän tallennetut tiedot

Jotta tiedot saatiin tallennettua, täytyi sovellus rakentaa niin, että rekisteröityessä sovellukseen käyttäjän tiedot tallentuvat kahteen paikkaan, autentikointiin ja tietokantaan. Sovelluksen lähdekoodi rakennettiin toimimaan molempia käyttäen. Sovellusta käyttäessä tarkistus tehdään kahteen paikkaan. Firebase autentikoinnin avulla saamme tarkistettua, onko käyttäjä kirjautunut sisälle. Mikäli vastauksena tulee arvo tosi, pyydetään autentikointia palauttamaan sisään kirjautuneen käyttäjän, ja palautetun käyttäjän avulla tehdään tietokantaan kysely. Kyselyssä hyödynnetään palautetun käyttäjän sähköpostiosoitetta, sillä se on molempia tietueita yhdistävä uniikki tieto. Tämän olisi voinut suunnitella paremmin tutustumalla dokumentaatioon etukäteen tarkemmin ja esimerkiksi hyödyntämällä autentikointiin tallennettavaa käyttäjän ID-merkkijonoa tai tietokannan indeksointia ja näin säästää vielä lisää aikaa.

Hosting-palvelun käyttöönotto oli yksinkertaista ja yllätti, miten mutkattomaksi se on tehty. Asetusten määrittämisen jälkeen uuden tuotantoversion julkaisu voidaan suorittaa WebStormissa kehitysympäristön terminaalissa. Komennon alkaessa alkaa esimääritelty tapahtumaketju, jonka aikana rakentuu uusi pakattu tuotantoversio ja se siirretään Google Firebase palvelimelle. Firebasen projektinäkymästä on mahdollista tarkastella erilaista dataa sekä versiohistorioiaa. Näkymän kautta on mahdollista palauttaa jokin aikaisemmista versioista tai katsoa lisätietoa, mikäli palvelimelle julkaisussa on käynyt virhe. Palvelu tarjoaa kattavasti monipuoliset työkalut pienten ohjelmistosovellusten kehitykseen ja ylläpitoon.

Projektille asetettiin kuukausittainen maksimibudjetti, jotta ei tulisi yllättäviä kuluja. Yllättäviä kuluja olisi voinut tulla esimerkiksi koodista, joka jäisi pyörittämään kyselyitä tai tekisi liikaa kyselyitä, tai tiedon tallennuksesta moninkertaisena. Maksimibudjetista ei kuitenkaan tarvinnut huolehtia, sillä projektin kuukausikustannukset ovat 0.01 euroa kuukaudessa (Kuva 24). Ainoat kustannukset syntyivät siis sähköpostin lähettämiseen käytetystä lisäosasta, jonka käyttöohjeissa oli maininta kyseisestä kustannuksesta.



Kuva 24. Projektin kuukausittainen kustannus

Sovelluksen vaatimuksia laatiessa tekijät huomioivat REST rajapinnan toteuttamisen, mutta valitulla teknologialla sille ei ollut tarvetta. Kuten tästä raportista on käynyt ilmi, perinteinen backend rakennetaan itse. Firebasea käyttäessä backend toteutus on olemassa ja riittää, kun service-tasolla kutsutaan oikeaa tietuetta ja käytetään Firebase SDK valmiita esirakennettuja funktioita.

6 Yhteenveto ja pohdinta

Toimeksianto suoritettiin suunnitellun aikataulun rajoissa niin, että vaatimusluettelon edellyttämät toiminnot on toteutettu. Google Firebasen käyttöönotto oli selkeää alustalle osoitetun dokumentaation ansiosta. Firebasen käyttäminen ei johtanut kompromisseihin kehitystyössä, vaan sen avulla pystyttiin rakentamaan toiminnallisuudet, kuten itse rakennetulla backendillä, helpommin. Suurin ero syntyi kehitystyöhön käytetystä ajasta. Firebasen avulla säästettiin huomattava määrä aikaa ja pystyttiin toimittamaan sovellus asiakkaalle nopeammin.

Tämän projektin kokemuksen perusteella Google Firebase soveltuu erinomaisesti pienien ja nopeasti kehitettävien ohjelmistoprojektien työkaluksi. Laajempien ja pitkäkestoisempien projektien kohdalla teknologiavalintaan voi vaikuttaa erilaiset vaatimukset, joten niiden kohdalla edut itse rakennetusta backendistä voivat mahdollisesti syrjäyttää Firebasen edut.

Opinnäytetyön tuloksena asiakas sai julkaistun verkkosovelluksen. Julkaistu sovellus on kuitenkin vielä alkuvaiheilla ja sitä voisi jatkossa kehittää muun muassa optimoimalla Firebasen käyttöä ja integroimalla lisää ominaisuuksia.

Lähteet

Codeacademy. 2022a. Back-End Web Architecture. Viitattu 22.10.2022. Saatavissa. https://www.codecademy.com/article/back-end-architecture

Codeacademy. 2022b. What is CRUD? Viitattu 22.10.2022. Saatavissa https://www.codecademy.com/article/what-is-crud

Domes, S. 2017. Progressive web apps with React : create lightning fast web apps with native power using React and firebase. 1st edition. Birmingham, England.

Educative. 2022. What is MVC architecture? Viitattu 22.10.2022. Saatavissa. https://www.educative.io/blog/mvc-tutorial

Fielding, R. & Taylor, R. 2000. Principled Design of the Modern Web Architecture. Viitattu 31.8.2022. Saatavissa https://www.ics.uci.edu/~fielding/pubs/webarch_icse2000.pdf

Firebase. a. Firebase CLI reference. Viitattu 13.10.2022. Saatavissa https://firebase.google.com/docs/cli

Firebase. 2020b. Service Level Agreement for Hosting and Realtime Database. Viitattu 15.10.2022. Saatavissa https://firebase.google.com/terms/service-level-agreement

Firebase. c. Simple, multiplatform sign-in. Viitattu 6.10.2022. Saatavissa https://firebase.google.com/products/auth?gclid=Cj0KCQjw-fmZBhDtARIsAH6H8qj8pCRI1GYGrwegtFkjsTr0TideuhnzDndQMiQyvfoT-52vERRX9wUaAqfvEALw_wcB&gclsrc=aw.ds

Firebase. d. Trigger Email. Viitattu 16.10.2022. Saatavissa https://firebase.google.com/products/extensions/firebase-firestore-send-email

Firebase. e. Manage users. Viitattu 22.10.2022. Saatavissa https://firebase.google.com/docs/auth/web/manage-users?hl=en&authuser=3

Github. 2022. fullstack-hy2022. Viitattu 22.10.2022. Saatavissa. https://github.com/fullstack-hy2020/misc/blob/master/dokumenttitietokannat.MD

Helsingin yliopisto. Rajapinnat ja REST. Viitattu 30.8.2022. Saatavissa https://web-palvelinohjelmointhttps://web-palvelinohjelmointi-21.mooc.fi/osa-6/4-rajapinnat-ja-rest

Haviv, A., Mejia, A. & Onodi, R. 2016. Web application development with MEAN : unlock the power of the MEAN stack by creating attractive and real-world projects : a course in three modules. Birmingham, England ; Mumbai, India : Packt Publishing.

Ilmarinen, V. & Koskela, K. 2015. Digitalisaatio : yritysjohdon käsikirja. Helsinki: Talentum.

Jenni, R. . FirebaseUi-Angular. Viitattu 15.10.2022. Saatavissa https://www.npmjs.com/package/firebaseui-angular

Logesh Gupta. 2021. HTTP Methods. Viitattu 31.8.2022. Saatavissa https://restfulapi.net/http-methods/

Medium. 2018. What is Firebase? The complete story, abridged. Viitattu 12.10.2022. Saatavissa https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0

OpenJS Foundation. About Node.js. Viitattu 20.10.2022. Saatavissa https://nodejs.org/en/about/

Runsala, T. 2022. Toimitusjohtaja. Suomen Vitamiinikeskus Oy. Haastattelu 19.10.2022.

Sitra. Tulevaisuussanasto. Viitattu 12.10.2022. Saatavissa https://www.sitra.fi/tulevaisuussanasto/freemium-malli/

Tanna, M. & Singh, H. 2018. Serverless Web Applications with React and Firebase: Develop real-time applications for web and mobile platforms. Birmingham: Packt Publishing, Limited.