
Ilias Doukas

# Perinteisen web-sovelluskehityksen ja vähäkoodialustojen vertailu

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

22.5.2022

# Tiivistelmä


| | |
|---|---|
| Tekijä: | Ilias Doukas |
| Otsikko: | Perinteisen web-sovelluskehityksen ja vähäkoodialustojen vertailu |
| Sivumäärä: | 31 sivua + 1 liite |
| Aika: | 22.5.2022 |
| Tutkinto: | Insinööri (AMK) |
| Tutkinto-ohjelma: | Tieto- ja viestintätekniikka |
| Ammatillinen pääaine: | Mobile Solutions |
| Ohjaaja: | Tutkijaopettaja Hannu Markkanen |


---


Insinöörityössä tutkittiin ja vertailtiin kahta erilaista tapaa kehittää web-sovelluksia. Vertailun kohteena olivat web-sovelluskehitys käsin ohjelmoimalla ja konfigurointiin perustuvat vähäkoodialustat. Insinöörityöhön valitut kehitystapojen edustajat ovat perinteisen web-sovelluskehittämisen osalta ReactJS ja vähäkoodialustojen osalta AppSheet. Molemmilla ratkaisuilla on mahdollista tehdä toisiaan vastaavia toteutuksia.

Insinöörityössä haluttiin selvittää, miten vähäkoodisovellus vertautuu käsin ohjelmoituun sovellukseen. Vertailu tehtiin luomalla prototyyppisovellukset molemmilla kehitystavoilla. Prototyyppisovelluksen aiheena oli työn johtaminen yrityksessä. Tämän lisäksi tutkittiin kehitystavoille ominaisia työkaluja, kehitystyöhön kuuluvia työvaiheita ja valmiiden sovellusten jatkokehitysmahdollisuuksia.

ReactJS on Facebookin kehittämä suosittu ohjelmakehys, jota hyödynnetään perinteisten web-sovellusten tekemisessä. AppSheet on Googlen omistama vähäkoodialusta, jolla on mahdollista luoda monipuolisia yrityssovelluksia ilman ohjelmointia. Työssä perehdyttiin kumpaankin kehitystapaan, niiden työvaiheisiin ja lopputuloksiin. Työ toimii apuna web-sovellusprojektin kehitystavan valinnassa.

Vähäkoodinen sovelluskehitys on lyhyessä ajassa noussut varteenotettavaksi vaihtoehdoksi haastamaan perinteistä sovelluskehitystä. Se tulee myös entistä laajemman tekijäkunnan saataville yksinkertaisempien työkalujen kautta. Käsin ohjelmoitavilla sovelluksilla on kuitenkin paikkansa nyt ja tulevaisuudessa, sillä niiden mukautus ja optimointimahdollisuudet ovat omaa luokkaansa.

Vertailussa kävi ilmi, että AppSheetillä tehty sovellus oli React-sovellukseen verrattuna viimeistellympi, vaikka sen tekemiseen kului vähemmän aikaa. React-sovelluksessa on kuitenkin laajemmat jatkokehitysmahdollisuudet.



Avainsanat: React, AppSheet, vähäkoodinen sovelluskehitys, web-sovelluskehitys

# Abstract


| | |
|---|---|
| Author: | Ilias Doukas |
| Title: | Comparing traditional web application development with low-code platforms |
| Number of Pages: | 31 pages + 1 appendix |
| Date: | 22 May 2022 |
| Degree: | Bachelor of Engineering |
| Degree Programme: | Information and Communication Technology |
| Professional Major: | Mobile Solutions |
| Instructors: | Hannu Markkanen, Senior Lecturer |



In this thesis two different ways of developing web applications are being compared and researched. Firstly, traditional development by manually coding and secondly, development by using a low-code platform. ReactJS framework will represent traditional development and AppSheet will represent development with low-code platforms. Both ways can be used to create similar applications.

This thesis aims to find out how a low-code application is comparable to a manually coded application. This comparison will be done by creating a prototype application using both ways. The purpose of the prototype application is a goal setting and leadership tool for companies. Moreover, the tools, required steps to develop an app and possible future advancements for the resulted applications are reviewed.

ReactJS is a popular framework developed by Facebook. React is used to create traditional web applications. AppSheet is a low-code platform which is owned and operated by Google. AppSheet is used to create applications for companies without any code. The thesis tries to give information on both development styles and the prototype applications created with them. This information can be used as a reference when selecting between forementioned development styles.

Low-code development has quickly emerged as a serious alternative to traditional programming. Manual coding will still stay relevant since it gives the best opportunities for customization and optimization.

The result of the comparison was that the prototype application created with AppSheet was more polished even though it took less time to create. The prototype app created with React will, however, have more options for future development.



Keywords: React, AppSheet, Low-code, Web development

## Sisällys

## Lyhenteet

HTML: *Hyper Text Markup Language*. Verkkosivujen luonnissa käytetty standardisoitu merkintäkieli.

OKR: Objectives and Key Results. Tavoitteita ja avaintuloksia seuraava johtamismalli.

JSON: JavaScript Object Notation. Avoimen standardin tiedostomuoto tiedonvälitykseen.

CSS: Cascading Style Sheets. Verkkosivun ulkoasun määrittävät tyylisivut.

# 1 Johdanto

Insinöörityössä vertaillaan kahta erilaista web-sovelluskehityksen tekniikkaa: perinteiseen ohjelmointiin nojaavaa ohjelmistokehitystä arvioidaan suhteessa uudenlaisiin vähäkoodisiin kehitysalustoihin. Eroavaisuuksia arvioidaan kehitystyön, työkalujen, valmiiden prototyyppisovellusten ja jatkokehitysmahdollisuuksien näkökulmista. Käytännössä halutaan selvittää, voidaanko vähäkoodialustoja hyödyntäen luoda tuotantotason sovelluksia ja onnistuuko se käsityönä tehtyä ohjelmointia helpommin, mutta saavuttaen silti toimivan lopputuloksen. Mikäli sovelluksiin muodostuu selkeitä eroavaisuuksia, niitä vertaillaan myös hyöty-haittasuhdetta pohtien. Suurena kiinnostuksen kohteena on myös valmiin sovelluksen ylläpidon ja jatkokehityksen helppous ja keveys. Lopuksi tutkitaan, onko kesken kehityksen tai valmiin sovelluksen kohdalla mahdollista vaihtaa kehitystavasta toiseen.

Vertailu tehdään tutkimalla kunkin kehitystavan suosittuja edustajia, joiden välillä oikeassakin kehitystyössä voidaan joutua puntaroimaan. Perinteistä web-sovelluskehitystä edustaa JavaScript-pohjainen React-kirjasto ja vähäkoodista kehitysalustaa AppSheet-alusta.

Insinöörityön aihe valikoitui tekijän nykyisten työtehtävien pohjalta ratkaisukonsulttina kotimaisessa verkkoalan yrityksessä, joka on laajentanut liiketoimiinsa ohjelmistokehityksen puolelle. Vähäkoodista kehitystyötä pidetään tämän laajennuksen keskiössä.

# 2 Perinteinen web-sovelluskehitys

## 2.1 Kehitystapa

Web-ohjelmistokehitys on sen alkuajoista lähtien tapahtunut pitkälti manuaalisesti ohjelmoimalla. Ohjelmointikielet ja -alustat ovat vuosien varrella muuttuneet, mutta nyt yleisimpänä web-ohjelmointikielenä toimiva JavaScript on sekin ollut erinäisissä muodoissaan käytössä jo vuodesta 1995 saakka [Pesquet

2020: 18]. Web-sovellusten ohjelmoiminen käsin antaa parhaissa tapauksissa tekijöille lähes rajattomat mahdollisuudet tehdä sovellukset juuri halutun kaltaisiksi ilman ylimääräisiä painolasteja. Sovelluksissa on siis vain sellaisia toiminnallisuuksia ja komponentteja, jotka kehittäjät ovat sinne itse luoneet ja lisänneet.

Ohjelmointityötä helpottamaan on kuitenkin kehitetty lukuisia ohjelmakehyksiä ja kirjastoja ajan myötä, sillä ei ole järkevää luoda samanlaisia tai samantyyppisiä toiminnallisuuksia jokaiseen sovellukseen uudestaan kerrasta toiseen. Näiden apuvälineiden käyttö on yleistynyt varsinkin viime vuosikymmenen aikana [Toledo 2018].

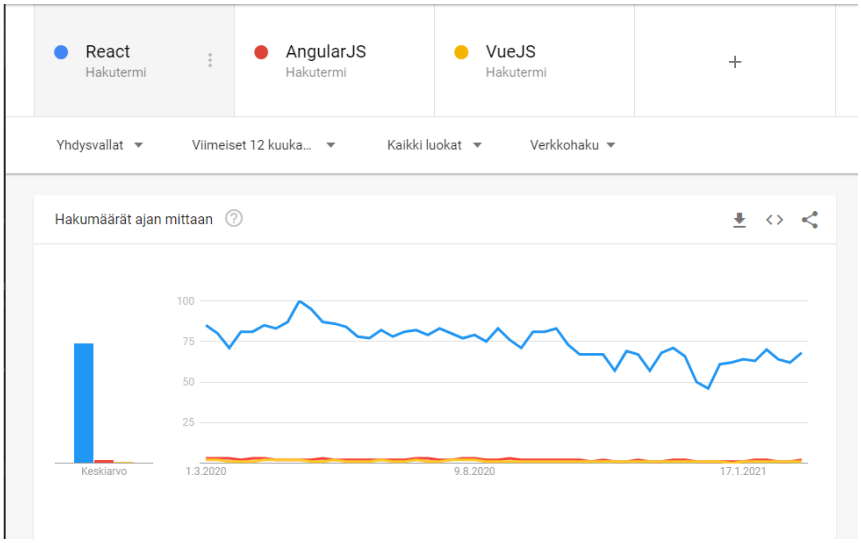
Tämä niin kutsuttu perinteinen tapa luoda web-sovelluksia ja sovelluksia on juurtunut syvälle lukemattomiin kehittäjiin ja yrityksiin. Tästä syystä ohjelmointikielien osaajia ja heille töitä tarjoavia työnantajia on kaikkialla maailmassa runsaasti [Stevens 2020].

Perinteisessä ohjelmistokehityksessä suurin kuluerä ovat henkilötyötunnit, kun taas työkalut, ohjelmointiresurssit ja kirjastot ovat usein joko maksuttomia tai suhteellisen edullisia. Web-sovelluksen ylläpito itsessään on usein edullista monilla saatavilla olevilla pilvipalveluilla, mutta niiden hinta kasvaa usein käyttäjämäärän tai varattujen kiinteiden resurssien mukaan. [Tozzi 2021.]

## 2.2 Valitut ohjelmointikielet ja kirjastot

Web-sovelluskehitykseen on tarjolla lukuisia erilaisia ohjelmointikieliä, ohjelmakehyksiä, kirjastoja ja työkaluja. Ohjelmointikielistä JavaScript on erittäin suosittu, ja esimerkiksi Stack Overflow’n käyttäjäkyselyssä se oli selvällä erolla suosituin ohjelmointikieli kyselyyn vastanneiden 58 031 ohjelmoijan keskuudessa. JavaScript on ollut kyselyn kärjessä nyt yhdeksän vuotta peräkkäin, joten sen asema laajalti ymmärrettynä ja käytettynä ohjelmointikielenä on vahva. [2021 Developer Survey 2022.] Tämän perusteella JavaScript valikoitui vertailussa käytettäväksi ohjelmointikieleksi.

Web-ohjelmakehyksistä suosituin on saman kyselyn perusteella React.js 67 593 vastaajan mukaan. JavaScript-ohjelmakehyksiä on tarjoilla useita, ja StackOverflow'n järjestämän kyselyyn mukaan suosituimpia suosiojärjestyksessä ovat mm. React, jQuery, Express, Angular ja Vue.js. Yleinen kiinnostus ja suosion kasvu näkyvät lisäksi muun muassa kehittäjien suosimalla Stack Overflow -kysymyspalstalla. Reactiin liittyvien kysymysten ja kommenttien lukumäärä on kaksinkertaistunut viimeisen kolmen vuoden aikana. [Stack Overflow Insights - Trends 2021.] Sama suosio jatkuu myös Google-hakutuloksissa, sillä React on tällä hetkellä kiinnostavin ja suosituin JavaScript-kirjasto Google Trends -hakutermiseurannan perusteella [Google Trends – Hakumäärät ajan mittaan 2021]. Kuvassa 1 on esitetty erot Reactin ja sen kilpailijoiden Angularin ja Vuen hakutermien käytössä vuoden 2020 aikana.




Kuva 1. React on suosittu Google Trends -vertailussa [Google Trends – Hakumäärät ajan mittaan 2021].

React on JavaScript-ohjelmointikielelle kehitetty kirjasto, jonka avulla kehittäjien on helpompi luoda interaktiivisia käyttöliittymiä ja verkkosovelluksia. Kirjastolla luotavien sivustojen pohjana on HTML (Hyper Text Markup Language) -kieli, jota React-komponenteissa myös paikoitellen hyödynnetään. Kirjasto perustuu valmiisiin komponentteihin, jolloin ylätason toimintoja ei tarvitse luoda alusta asti itse ja monet niistä ovat täysin käyttökelpoisia sellaisenaan. Kirjaston komponentit ovat kuitenkin vapaasti räätälöitävissä tai korvattavissa kehittäjän

omilla ratkaisuilla. Reactia hyödyntävän kehittäjän on tunnettava HTML- ja JavaScript-kielet voidakseen tehdä tuotantotason sivustoja ja ohjelmia, sillä toteutus tehdään käsin ohjelmoimalla [Hooks at a Glance 2021].

React on avoimen lähdekoodin projekti, jonka on perustanut yhdysvaltalainen sosiaalisen median yritys Facebook. Yritys ylläpitää projektia edelleen, mutta siihen osallistuu myös muita yrityksiä ja yksityisiä kehittäjiä. Avoin lähdekoodi tarkoittaa sitä, että kuka tahansa voi käydä katsomassa toteutuksen lähdekoodia ja muokata siitä omia versioitaan. Avoimen lähdekoodin projekteihin voi myös tehdä suoraan omia muutosehdotuksia, jotka tyypillisesti muut osallistuvat arvioivat ja hyväksyvät. [What is open source? 2022.]

Ensimmäinen versio nykyisestä Reactin avoimesta lähdekoodista julkaistiin JSConf-tapahtumassa vuonna 2013 [Occhino & Walke 2013]. Kirjastoa on kehitetty siitä lähtien useiden vuosien ajan, samalla kun sen suosio on jatkanut kasvamistaan.

Reactin lisäksi vaihtoehtoina olivat sen kilpailijat Angular ja Vue.js. Kaikilla kehyksillä on mahdollista luoda toisiaan vastaavia sivustoja ja sovelluksia, mutta niiden rakenteissa ja luontitavoissa on eroavaisuuksia. Angular on näistä kolmesta ohjelmakehyksestä vanhin, ja sille on ehtinyt kertyä laaja kirjo ominaisuuksia ja dokumentaatiota. Sen laajuuden heikkoutena on kuitenkin sen raskaus, sillä suuri ohjelmakehys kasvattaa projektia ja hidastaa sovellusta. Vue.js taas on ohjelmakehyksistä tuorein ja kevein, mutta se ei ole vielä kasvanut yhtä suosituksi kuin muut vaihtoehdot. [Borrelli 2022.] React on ohjelmakehyksistä suosituin, ja siinä yhdistyvät sen kilpailijoiden parhaat puolet, joten se valittiin perinteisen web-sovelluskehityksen edustajaksi.

## 2.3 Työkalut

React-pohjaisen sovelluksen työstämiseen käytetään kehitysympäristöä, kuten avoimen lähdekoodin Visual Studio Code -ohjelmaa. Tämä esimerkiksikin valittu ohjelma lisäosineen on ilmainen, joten siitä ei aiheudu kustannuksia kehittäjille.

Visual Studio Code on kevyt ja monipuolinen kehitysalusta monille ohjelmointikielille ja -kirjastoille. React on Visual Studio Coden puolesta hyvin tuettu, sillä molemmat sen käyttämät kielet, JavaScript ja HTML, ovat suoraan käytettävissä [Programming Languages 2021]. Ohjelmointikielien kirjoittamista helpottamaan on myös saatavilla lukuisia lisäosia, jotka ladataan ja asennetaan suoraan Visual Studio Coden käyttöliittymästä.

Toteutettavan sovelluksen katseluun ja testaamiseen voidaan käyttää mitä tahansa modernia Internet-selainta. Selaimiin on myös saatavilla erikseen asennettava React Developer Tools -lisäosa, jonka avulla ohjelman vianselvitys ja optimointi on helpompaa. Selainlisäosa on Reactin luojan Facebookin kehittämä, joten se on virallinen ja täysin tuettu työkalu.

Tavoitellun sovelluksen arkkitehtuurissa on käyttäjätaso (front-end) ja taustajärjestelmä (back-end). Front-endissä on käyttäjän näkemä Reactilla rakennettu käyttöliittymä ja back-endissä on Node.js-ajoympäristö. Node.js on avoimen lähdekoodin alustariippumaton JavaScript-ohjelmakehys, joka toimii paikallisena palvelimena ja mahdollistaa reaaliaikaisen datan käsittelyn. [Node.js 2021]. Käyttäjä toimii sovelluksen kanssa React-komponenttien kautta joko lukien tai syöttäen tietoa.

# 3 Vähäkoodinen sovelluskehitys

## 3.1 Kehitystapa

Vähäkoodinen sovelluskehitys nykymuodossaan on suhteellisen uusi suuntaus, jossa monenlaisia sovelluksia voidaan toteuttaa valmiita konfiguraatiotyökaluja käyttäen [Alexander 2021]. Erilaisia vähäkoodialustoja on ollut olemassa jo 1990-luvulta lähtien, mutta nykyisiä työkaluja ja alustoja kuvaava termi ”vähäkoodinen sovelluskehitys” otettiin käyttöön vasta vuonna 2014 [Midura 2019].

Tämän työtavan mielekkyys on siinä, että sovelluksia voidaan luoda nopeammin, kevyemmällä pohjakoulutuksella ja pienemmällä henkilöstöpanostuksella

[Alexander 2021]. Koodirivejä syntyy vähäkoodisissa projekteissa vähemmän, tai niitä ei tule välttämättä lainkaan. Tämän ansioista inhimillisten virheiden määrä vähenee ja vastaavasti myös teknisen ylläpidon tarve [Kotilainen 2019]. Vähäkoodisen sovelluskehityksen hyödyt korostuvat pienen mittakaavan projekteissa, sillä pienen toiminnallisuuden luomiseksi ei tarvitse käyttää suhteettoman pitkää aikaa palvelinten, tietokantojen ja sovellusrakenteiden suunnitteluun.

Valmiit rakenteet tuovat kuitenkin myös, ja se tulee ottaa huomioon vähäkoodialustoja harkittaessa. Monimutkaiset räätälöinnit tai korkean suorituskyvyn tarve puoltavat usein perinteistä sovelluskehitystä [Kotilainen 2019]. Muun muassa AppSheet-alustan kotisivuilla kuvataan vähäkoodisten sovellusten soveltuvan erityisesti arkisten usein toistuvien sähköisten tehtävien automatisointiin [Enable everyone in your organization to build and extend applications without coding 2021].

Vähäkoodialustoissa suurin kuluerä perustuu usein palvelun lisensseihin tai yleisiin ylläpitokuluihin. Esimerkiksi AppSheet laskuttaa aktiivisten sovelluksen loppukäyttäjien lukumäärän mukaan lineaarisesti tilausmallin mukaisesti [AppSheet – Pricing 2022]. Sovelluksen ylläpidon hinta on siten ennakoitavissa, mutta suuremmissa projekteissa ylläpitokustannukset voivat kertyä hyvin suuriksikin, kuten taulukon 1 hintaesimerkeistä näkyy. Esimerkiksi sadalla käyttäjällä lisenssikulut ovat jo 1000 dollaria kuukaudessa. Vastaavasti kehittäjän työaikaa tarvitaan minimaalinen määrä, joten projektin tilaajan kannattaa puntaroida vaihtoehtoja sovelluksen käyttötarpeen mukaan [Tozzi 2021].

Taulukko 1. AppSheetin hintoja eri käyttäjämäärille [AppSheet – Pricing 2022].

| **Käyttäjien lukumäärä** | **Lisenssikulut kuukaudessa** |
|---|---|
| 5 kpl | 50 $ |
| 50 kpl | 500 $ |

| | |
|---|---|
| 100 kpl | 1000 $ |
| 250 kpl | 2500 $ |

## 3.2 Alustat ja ominaisuudet

Vähäkoodisen sovelluskehityksen alustoja on markkinoilla useita, ja suosituimpien joukkoon kuuluvat mm. AppSheet, Microsoft PowerApps, Appian ja Zoho Creator [Marvin 2018]. AppSheet-alusta on alun perin samaa nimeä kantaneen startup-yrityksen tuote, joka julkaistiin vuonna 2014 [Nickelsburg 2015]. Vuonna 2020 hakukoneyhtiö Google osti yrityksen ja liitti sen suoraan omaan Google Cloud Platform -pilvipalveluyksikköönsä [Google acquires AppSheet to help businesses create and extend applications—without coding 2020].

AppSheetillä on mahdollista tehdä sovelluksia ilman koodaamista, käyttäen suoraan Internet-selaimessa toimivaa konfigurointityökalua. Ketterällä ja tehokkaalla alustalla käyttövalmiita sovelluksia voi luoda ilman ohjelmointikielten opiskelua, ja sovellus luodaan täysin työkalun tarjoamia toimintoja konfiguroimalla. AppSheetin pääominaisuuksiin kuuluvat datan kerääminen useista yleisistä lähteistä, prosessien automatisointi ja koneoppimisen hyödyntäminen. [The fastest way to build apps and automate work 2022.] Työkalun yksi oleellisista käyttötarkoituksista on korvata monien yritysten käyttämät jaetut taulukkolaskentatiedostot, sähköpostiketjut ja muut vastaavat usein toistuvat sähköiset manuaaliprosessit, joissa ihminen täyttää samoja asioita toistuvasti tai siirtää tietoa järjestelmästä toiseen.

AppSheet on maksullinen alusta, jonka hinta on edullisimmillaan 5 dollaria kuukaudessa aktiivista käyttäjää kohden, mutta laajemmilla ominaisuuksilla varustetun käyttäjäkohtaisen tilauksen hinta on 10 dollaria kuukaudessa käyttäjää kohden. Hinta sisältää työkalun lisäksi sovelluksen ylläpidon ja käyttäjävarmennuksen, jotka yleensä sovelluksen tekijän tulee järjestää ja pystyttää erikseen.

Vaihtoehtoisesti kaikille käyttäjille avoimen sovelluksen, joka ei vaadi kirjautumista tai tunnista käyttäjäprofiileja, voi julkaista kiinteällä 50 dollarin kuukausihinnalla. [AppSheet – Pricing 2021.]

Power Apps on suosittu vaihtoehto AppSheetille. Se on Microsoftin tarjoama alusta vähäkoodiseen tai täysin koodittomaan sovelluskehitykseen. Sovelluksia voi tehdä eri lähtökohdista, esimerkiksi valmista sovelluspohjaa muokkaamalla tai valmiin datamallin pohjalta. Power Apps toimii hyvin muiden Microsoftin palveluiden kanssa, kuten Power BI -raportointialustan kanssa. [Overview of creating apps in Power Apps 2022.]

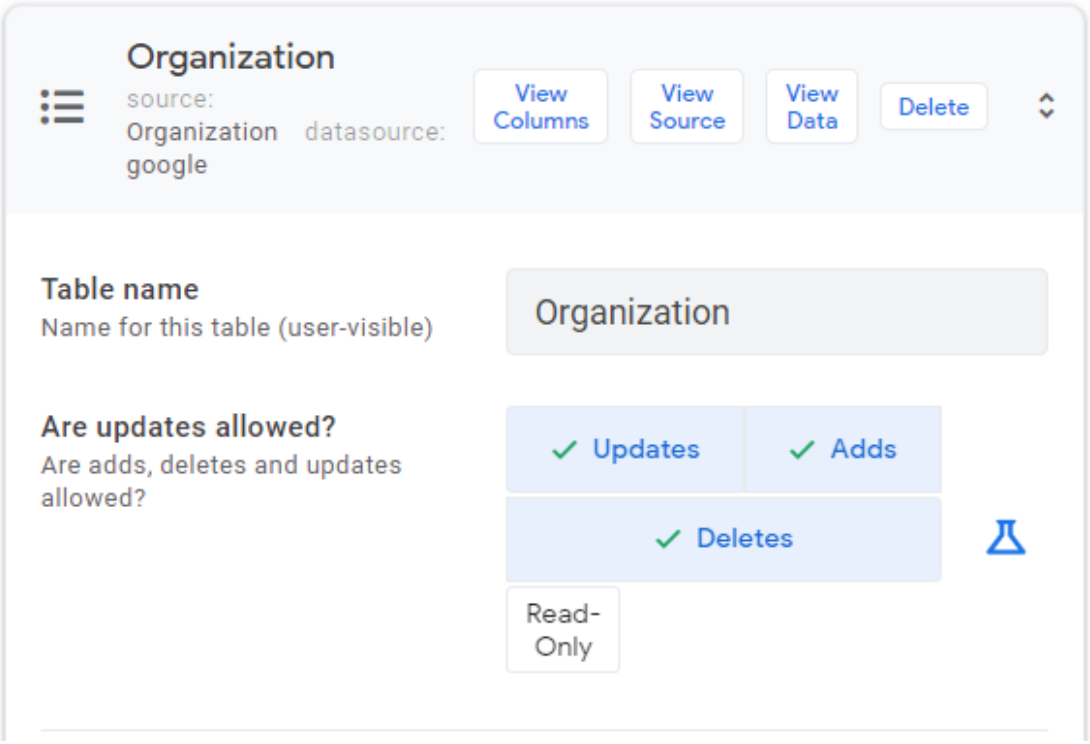
Kolmas esimerkki suosituista alustoista on Appian. Se on hyvin laaja alusta, jossa on oman sovellusrakentimensa lisäksi mahdollista tehdä automaatioita muihin sovelluksiin ohjelmistorobotiikalla. Ohjelmistorobotiikassa ohjelma opetetaan tekemään klikkauksia, hiirenvetoja tai muita toimintoja, joilla ohjataan muita sovelluksia. [Appian Low-Code Platform 2022.]

AppSheet on vähäkoodialustojen olemassaoloon suhteutettuna jo vakiintunut alusta, ja työn tilaajayritys käyttää Googlen-ekosysteemiä jo ennestään muissa liiketoiminnan alueissa. Tästä syytä AppSheet valikoitui vertailuun vähäkoodialustojen edustajaksi.

## 3.3 Työkalut

Sovelluksen ylläpitoalustana ja datalähteenä toimivat Google Cloud -pilvialusta ja Google Sheet -laskentataulukko. Erilaisia vaihtoehtoja datalähteeksi on monia, mutta selkeyden vuoksi vertailussa keskitytään vain yhteen suureen palveluntarjoajaan. Kaikki sovelluksen osat ovat teknologiayhtiö Googlen palveluvalikoimasta. AppSheet-alustaa ja Google Sheet -taulukoita käytetään Internet-selaimella osoitteissa appsheet.com ja sheets.google.com, eivätkä ne vaadi mitään asennuksia kehittäjän laitteille. Palveluiden käyttäminen vaatii kuitenkin Google-tilin.

Sovelluksen datarakennetta ja käyttöoikeuksia hallinnoidaan suoraan työkalun data-asetusten kautta. Kuvassa 2 on taulu, jossa käyttäjille on annettu täydet oikeudet tehdä muutoksia sen sisältämiin tietoihin. Tiedon muuttaminen vaatii kuitenkin myös käyttöliittymässä muokattavan elementin, kuten lomakkeen.




Kuva 2. AppSheetin data-asetukset projektin alussa. Kaikki muutokset ovat sallittuja.

## 4 Prototyyppisovellus kahdella eri kehitystavalla

Insinöörityössä verrattiin kahta kehitystapaa, perinteistä web-sovelluskehitystä ja vähäkoodialustaa luomalla prototyyppitasoinen paikallisesti toimiva web-sovellus molemmilla kehitystavoilla. Sovelluksen käyttötarkoituksena oli toimia johtamista avustavana työkalun. Sovellusten oli tarkoitus olla toiminnallisuudeltaan toisiaan vastaavia, mutta niiden ei tarvinnut olla identtisiä keskenään. Sovellusten käyttölogiikan pohjana oli OKR (Objectives and Key Results) -johtamismalli, jonka perusteella kaikki toiminnot ja käyttöliittymät suunniteltiin. OKR-mallissa johto yhdessä tiimien tai yksittäisten työntekijöiden kanssa sopii yhdessä keskustellen kolmesta viiteen tavoitetta. Näihin tavoitteisiin pyritään seuraamalla ja

toteuttamalla erilaisia avaintuloksia, joita on vastaavasti muutamia kutakin tavoitetta kohden. Avaintulosten tulee olla lähtökohtaisesti mitattavia, eli esimerkiksi kappalemääräisiä. [Panchadsaram 2022]. Nämä tavoitteet avaintuloksineen oli voitava syöttää sovellukseen, ja niiden tilaa piti pystyä tarkastelemaan ja avaintulosten edistymisiä päivittämään

Perinteisen web-sovelluskehityksen esimerkkisovellus luotiin ReactJS:n pohjalta ja vähäkoodisen sovelluskehityksen vastine AppSheetillä.

## 4.1 Perinteinen ohjelmistokehitys

### 4.1.1 Työkalut ja projektin luominen

Sovelluskehittämisen aloittamiseksi kehittäjän on asennettava tarvittavat ohjelmistot ja työkalut työasemalleen. Tässä esimerkissä tarvittavat ohjelmat olivat Visual Studio Code (kehitysympäristö), Node.js (JavaScript-ajoympäristö), Windows Terminal (komentokehote) ja Git (versiohallinta). Ohjelmat asennettiin tässä toteutuksessa Windows 11 -käyttöjärjestelmällä varustettuun tietokoneeseen. Node.js sisältää myös npm-paketinhallintatyökalun, jolla projektiin voidaan liittää valmiita JavaScript-kirjastoja helposti. Npm:n avulla kirjastoja voidaan myös päivittää olemassa olevassa projektissa uusien versioiden tullessa saataville. Tässä sovelluksessa käytetyt JavaScript-kirjastot olivat json-server (yksinkertainen paikallinen tietokanta), material-ui (käyttöliittymäkomponentit) ja React (käyttöliittymäkirjasto).

Versiohallintana ja samalla varmuuskopioratkaisuna projektissa käytettiin Git-työkalua ja siihen liitettynä palveluna GitHubia. GitHub on Microsoftin omistama suosittu palvelu, joka on yksittäisille käyttäjille ja avoimen lähdekoodin projekteille maksuton. Tiimitoteutukset yksityisine projekteineen ovat maksullisia lisäpalveluita. [Lardinois 2018.]

Sovelluksen tyylittely ja käyttöliittymän elementit toteutettiin Material UI -kirjastoa pohjana käyttäen. Kirjasto sisältää lukuisia hyödyllisiä muokattavissa olevia valmiita komponentteja, kuten listoja, elementtilaatikoita ja tekstikenttiä.

Projektia varten luotiin uusi kansio, jotta sen sisältämät tiedostot ja kytkökset pysyvät hallinnassa. Muiden toimintojen ja kirjastojen kytkemiseksi projektiin piti ensin asentaa Node.js-ajoympäristö. Node.js sisältää suoraan npm-paketinhallintatyökalun. Asennusten jälkeen ajetaan tarvittavat komennot, joilla kirjastot ja ohjelmakehykset lisätään ja asennetaan projektille. Komentojen tuloksena syntyvät React-projektin pohjatiedostot. React-projektin kansio avataan lopuksi Visual Studio Codella ja varsinainen kehitystyö voi alkaa.

Tähtäimessä oli yhden sivun sovellus, joten voitiin käyttää esimerkkinä ja pienten projektien lähtökohtana tarjottavaa ”Create React App” -ympäristöä. Komennon ”npx create-react-app <sovelluksen-nimi>” ajamalla syntyy projektikansio, jonka alle koko projektin rakenne muodostuu. Create-React-App:n mukana tulevat tarvittavat työkalut valmiiksi konfiguroituna. Mukana tulee myös muita mukavuutta lisääviä ominaisuuksia, kuten koodipohja ja reaaliaikainen kehitysympäristö, jonka avulla sovelluksen testiversio päivittyy automaattisesti aina, kun koodia muokataan. Tämä helpottaa ja nopeuttaa kehitystyötä huomattavasti, kun jokaista muutosta varten ei tarvitse rakentaa sovellusta manuaalisesti uudelleen. Käytännössä syntyy valmis sovelluksen runko, jonka päälle lähdetään tekemään lisäyksiä ja muutoksia, joiden avulla edetään kohti valmista omaa toteutusta. Tämä on erityisen hyödyllistä uusille React-käyttäjille, joille muut modernit JavaScript-kirjastot ja työkalut ovat tuttuja. Kehitysympäristön rakentamiseen ei tarvitse perehtyä käytännössä lainkaan, vaan päästään suoraan sovelluskehitykseen [Dayal 2022]. Esimerkkikomennossa 1 on esitettynä komennot uuden React-projektin luomiseksi Create-React-App:a hyödyntäen ja sen käynnistämiseksi npm:n avulla.

```
npx create-react-app reactprojekti
cd reactprojekti
npm start
```

Esimerkkikomento 1. Terminaalikomennot, joilla luodaan uusi React-projekti, siirrytään projektikansioon ja käynnistetään kehityspalvelin.

Kehitysympäristö on näiden toimien jälkeen käyttövalmis ja kehitystyö voi alkaa. Kehityspalvelimen ollessa käynnissä on sovellus koko ajan nähtävissä oletusselaimella. Aina kun Visual Studio Codessa tehdyt muutokset tallennetaan, sovellus uudelleenlatautuu selaimessa ja muutokset tulevat reaaliajassa näkyviin.

Esimerkkiprojektissa käyttöliittymäkomponentteihin käytettiin pääasiassa Material UI:n valmiita komponenttipohjia. Ohjelmakehys asennetaan npm:n avulla esimerkkikomennon 2 mukaisesti.

```
npm install @mui/material @emotion/react @emotion/styled
```

Esimerkkikomento 2. Material UI -käyttöliittymäohjelmakehyksen lisääminen osaksi projektia.

Käyttöliittymäkomponenttien asentamisen jälkeen koossa ovat kaikki tarvittavat palaset sovelluksen rakentamiseen, pois lukien tietokanta. Sovellukseen syötettävien tietojen tallentamiseen ja lukemiseen käytetään paikallista REST API:a, JSON Serveriä. JSON on avoimen standardin tiedostomuoto tiedonvälitykseen. JSON Serveriä käytetään nopeaan testaamiseen, ja se vastasikin tämän toteutuksen laajuutta erittäin hyvin. [Json Server 2022.] JSON Server asennetaan sekin npm:n avulla esimerkkikomennossa 3 esitettävällä terminaalikomennolla, joka käynnistettäessä asetetaan käyttämään db.json-tiedostoa. Tiedosto toimii varsinaisena tiedon tallennuspaikkana. JSON Server -palvelin asetettiin käynnistymään portissa 3001, sillä oletuksena käytetty portti 3000 oli jo käytössä Reactin kehityspalvelimella.

```
npm install json-server
json-server --watch db.json --port 3001
```

Esimerkkikomento 3. REST API JSON Serverin asentaminen projektiin ja sen ajaminen vapaassa portissa 3001.

### 4.1.2 Käyttöliittymä

Työssä tehdyn sovelluksen toimintojen käyttämiseen tarvittiin kaksi oleellista osaa, jotka ovat lomake uusien tavoitteiden syöttämistä varten ja näkymä olemassa olevien tavoitteiden esittämiseksi avaintuloksineen. Käyttöliittymä rakennettiin Material UI -kirjaston sisältämien komponenttien pohjalta. Kirjaston sisältämiä komponentteja yhdisteltiin ja muokattiin vastaamaan käyttötarkoitusta mahdollisimman hyvin. Komponenttien sisäänrakennettuja attribuutteja säädetään muun muassa niiden koon, asettelun tai toimintalogiikan muuttamiseksi.

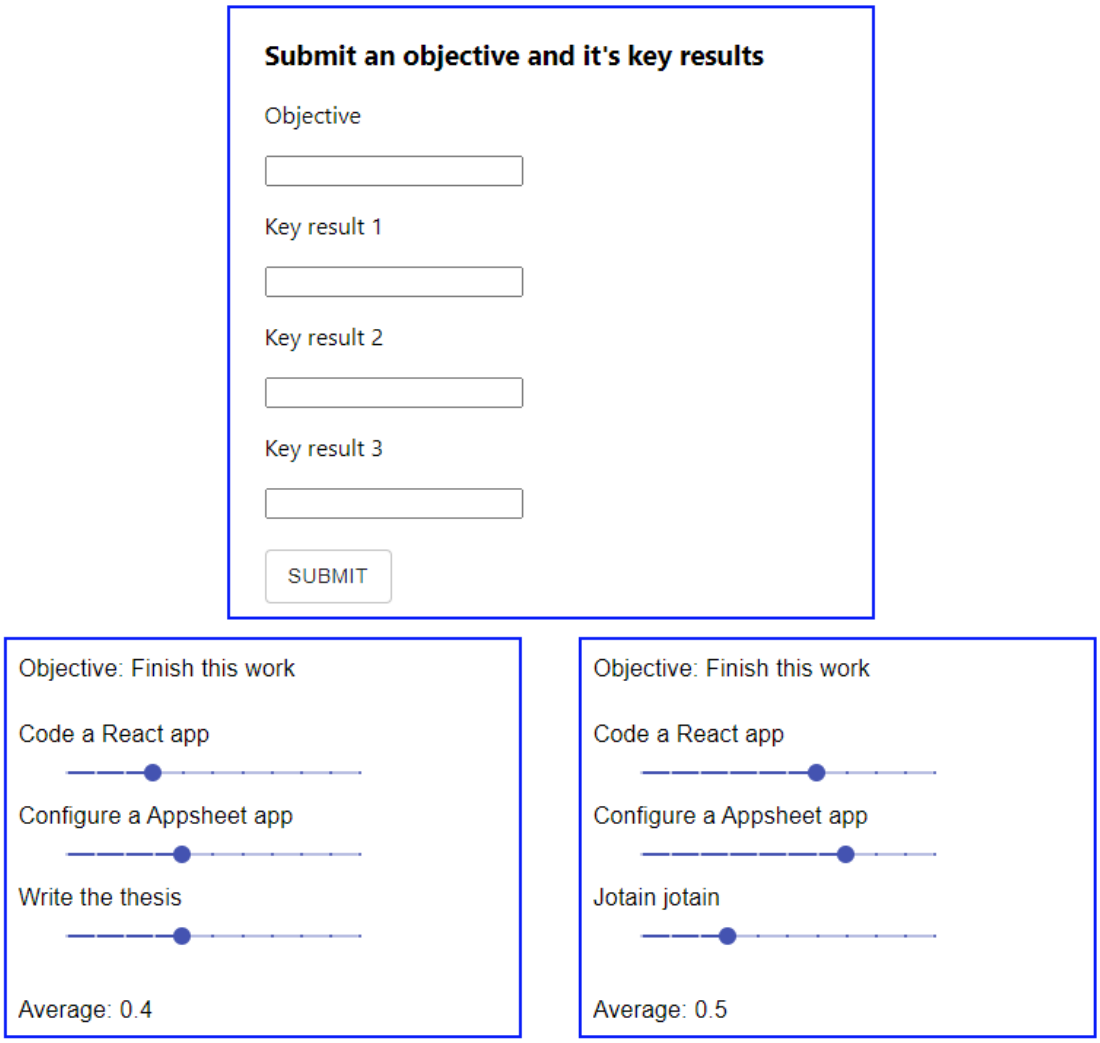
Komponenttien tyylittely tehdään CSS:llä kirjoitettuja tyylimäärittelyjä käyttäen, minkä lisäksi komponentin ominaisuuksia muokataan komponenttien omia attribuutteja säätämällä. CSS-tyyleillä määritellään muun muassa komponentin värit, fontit ja marginaalit. Tyylittelyä tehdään paikoin myös suoraan komponenttien sisällä, kun käytössä on kertakäyttöinen komponentti. Tyylien pitäminen erillään mahdollistaa kuitenkin niiden hyödyntämisen useammassakin komponentissa, ja laajemmassa toteutuksessa näin tulisikin toimia kaikissa käyttötapauksissa. Erilliset tyylit myös helpottavat mahdollisia tulevia päivityksiä tyyleihin. Esimerkkikoodissa 1 näytetään esimerkki tyylimuuttujan luomisesta. Tyylimuuttujaa käytetään komponentissa tyylien asettamiseksi.

```
const styles = (theme) => ({
  color: {
    backgroundColor: "#bedbfa",
    border: "solid 1px black",
  },
});
```

Esimerkkikoodi 1. Komponentissa käytettävä tyylimuuttuja.

Valmiiden käyttöliittymäkomponenttien kevyellä mukautuksella ohjelman tarvitsema käyttöliittymä tehdään yhdelle sivulle. Ensimmäisenä sivulle tulee lomake,

jonka alla on listattuna olemassa olevat tavoitteet erillisessä ruudukossa, kuten kuvassa 5 näkyy. Käyttöliittymä skaalautuu käytettävän tilan mukaisesti käytettävän laitteen ruudulle.




Kuva 3. Valmis React-toteutuksen käyttöliittymä.

### 4.1.3 Sovelluslogiikka ja tiedonhallinta

Sovellukseen luotiin uusia tavoitteita avaintuloksineen, ja niiden edistymistä piti pystyä päivittämään helposti. Tavoitteet avaintuloksineen kirjoitetaan JSON-tiedostoon JSON Server -rajapintaa käyttäen ennaltamääritellyn rakenteen mukaisesti. Tiedot luetaan aina automaattisesti sovelluksen käynnistyessä ja ne ase-

tetaan omiin kenttiinsä sovelluksen päänäkymään. Tämän toteuttamiseksi luotiin oma koodiluokka datan liikuttamiseen. Esimerkkikoodissa 2 JsonAPI-luokassa tietoa lisätään tietokantaan postObjAsync-funktiolla, noudetaan fetchObjective-funktiolla ja päivitetään putObjAsync-funktiolla.

```
const JsonAPI = () => {§
  const rootUrl = "http://localhost:3001";

  const fetchObjective = async (url) => {
    const notes = await fetch(rootUrl + "/" + url);
    const json = notes.json();
    return json;
  };

  const postObjAsync = async (data, url) => {
    console.log(JSON.stringify(data));
    const res = await fetch(rootUrl + "/" + url, {
      method: "POST",
      headers: {
        "Content-type": "application/json",
      },
      body: JSON.stringify(data),
    });
    return res;
  };

  const putObjAsync = async (data, url) => {
    const { id } = data || null;
    if (id) {
      const res = await fetch(rootUrl + "/" + url + "/" + id, {
        method: "PATCH",
        headers: {
          "Content-type": "application/json",
        },
        body: JSON.stringify(data),
      });
      return res;
    } else {
      return "Invalid id";
    }
  };

  return {
    postObjAsync,
    fetchObjective,
    putObjAsync
};
}

export default JsonAPI;'
```

Esimerkkikoodi 2. JsonAPI-luokka.

Apukirjastojen ja ohjelmakehysten asentaminen projektiin oli näin suoritettu, mutta ulkopuoliset kirjastot oli vielä otettava käyttöön koodissa. Sovellus jakautuu kahteen tiedostoon, joista toisessa on ohjelmalogiikka käyttöliittymineen ja toisessa JsonAPI-luokka. Kirjastoja ja niiden sisältämiä komponentteja tuodaan käytettäväksi import-komentoa käyttäen. Esimerkkikoodissa 3 on kaikki pääohjelman käyttämät kirjastot ja muut tuodut osat, kuten JsonAPI-luokka.

```
import { React, useEffect, useState } from "react";
import {
     Container,
     Typography,
     GridList,
     Box,
     Button,
     GridListTile,
     Slider,
} from "@material-ui/core";
import { withStyles } from "@material-ui/styles";
import "./App.css";
import JsonAPI from "./JsonAPI";
```

Esimerkkikoodi 3. Kirjastojen ja niiden sisältämien komponenttien, JsonAPI-tiedoston ja tyylittelyiden tuominen varsinaiseen ohjelmakoodiin.

Sovelluksessa oli näiden lisäysten jälkeen tarvittavat komponentit tavoitteiden näyttämiseksi sisältöineen. Aina sovelluksen käynnistyksen yhteydessä tarkistetaan tietokannasta mahdolliset olemassa olevat tiedot, ja mikäli tietokannassa on tavoitteita, ne tuodaan Reactin komponenttitiloihin (state) käyttäen useEffect-koukkua (hook). Koukut mahdollistavat tilojen ja muiden Reactin ominaisuuksien käyttämisen ilman, että sovellutuksia varten täytyy kirjoittaa omat luokat. [React – Tutorial 2022.] Tiloista tiedot viedään käyttöliittymän tavoiteruudukkoon dynaamisesti heti, kun tiedot on saatu onnistuneesti ladattua tietokannasta. Tietojenpäivitysfunktio asetettiin useEffect-koukun alle, kuten esimerkkikoodissa 4 näkyy. Aina kun sitä kutsutaan, tietokannasta haetaan mahdolliset olemassa olevat tavoitteet. Noudetut tavoitteet asetetaan objectives-tilaan setObjectives-funktiolla.

```
useEffect(() => {
    const getObjectives = async () => {
      const res = await fetchObjective("objectives");
      setObjectives(res);
    };
    getObjectives();
  }, []);
```

Esimerkkikoodi 4. useEffect-koukun alle asetettu tietojenpäivitysfunktio.

Samalla kun tietojen tarkistus aloitetaan, ohjelma jatkaa eteenpäin ja luo uusien tavoitteiden syöttämiseen käytettävän lomakkeen. Lomakkeelle syötetyt tiedot asetetaan dataobjektiin, joka viedään tietokantaan. Tietokannan päivittämisen jälkeen näkymä päivitetään uudelleen, ja sen seurauksena uusi lisätty tavoite avaintuloksineen tulee heti nähtäville sovellukseen. Esimerkkikoodin 5 mukaisesti dataobjekti luodaan postObjective-funktiossa. Funktiota kutsutaan, kun käyttöliittymässä painetaan lomakkeen lähetyspainiketta. Objekti lähetetään postObjAsync-funktiolla tietokantaan. Mikäli lähetys onnistuu, sivu ladataan uudelleen. Muussa tapauksessa virheilmoitus tulostuu lokiin.

```
const postObjective = async (user, objective, kr1, kr2, kr3) => {
  const data = {
    id: Math.random(1, 100000),
    user: user,
    objective: objective,
    kr1: kr1,
    kr1value: 0.0,
    kr2: kr2,
    kr2value: 0.0,
    kr3: kr3,
    kr3value: 0.0,
  };

  const res = await postObjAsync(data, "objectives");
  if (res.status === 200 || res.status === 201) {
    window.location.reload();
  } else if (res.status === 400) {
    console.log("Unable to submit idea", res.json());
  } else {
    console.log("Unexpected behaviour");
    console.log(res.body);
  }
};
```

Esimerkkikoodi 5. Dataobjektin luominen ja lähettäminen tietokantaan.

Tietojen hakemisen ja uuden tiedon lisäämisen lisäksi sovellukseen tarvittiin mahdollisuus päivittää olemassa olevia tietoja. Sovellukseen syötettyjen tavoitteiden avaintulosten edistyessä niiden tilaa tuli pystyä päivittämään. Avaintulosta päivitetään suoraan ruudukossa olevien korttien kautta. Muutokset päivittyvät välittömästi tietokantaan, ja siitä vastaavasti takaisin sovelluksen tavoitelistausnäkymään. Esimerkkikoodissa 6 updateObjectives-funktiolla lähetetään päivitetty avaintuloksen arvo sen tunnistetta (ID) vastaavaan tavoitteeseen tietokannassa.

```
const updateObjective = async (kr, val, id) => {
  var krType= kr;
  try {
    const post = {
      [krType] : val,
      id: id,
    }
    const update = await putObjAsync(post, "objectives");
    const json = await update.json();
    console.log(json);
  } catch (e) {
    console.log(e);
  }
  window.location.reload();
};
```

Esimerkkikoodi 6. Avaintuloksen arvon päivittäminen.

### 4.1.4 Kehityskokemus

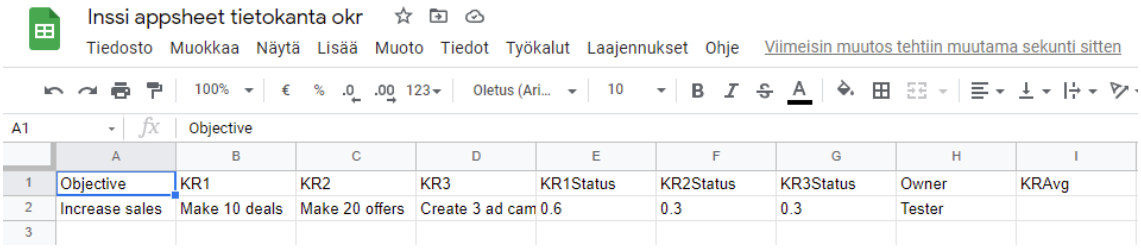
Sovelluksen kehittäminen oli kokonaisuutena selkeää, mutta työn aloittaminen vaati varsin paljon pohjatöitä. Paljon riippuu kuitenkin kehittäjän kokemuksesta käytettyjen työkalujen ja ohjelmointikielien käytössä, mutta kokeneenkin kehittäjän on perustettava kehitysympäristö, haarukoitava tarvittavat ohjelmakehykset sekä kirjastot ja asennettava kaikki toimintakuntoon. Pohjatöiden määrä on sitä suurempi, mitä laajempi projekti ja kehittäjätiimi on kyseessä. Olin itse ollut osallisena kahdessa React-projektissa ennen tämän sovellusprojektin aloitusta ja jouduin turvautumaan dokumentaatioon kohtalaisen usein, sillä en ollut vielä omaksunut ohjelmakehyksen omia ominaisuuksia täysin.

Kun kehitysympäristö on valmis, itse kehitystyö eteni vauhdilla. Varsinaista ohjelmakoodia syntyi kaiken kaikkiaan yhteensä 256 riviä. Ilman ohjelmakehyksiä ja käytettyjä kirjastoja projektin työmäärä olisi ollut varmasti paljon suurempi, mutta nytkin koko projektiin meni noin kaksi työpäivää. Itse tehtyä koodia tarvittiin tässä projektissa lähinnä tiedon kuljettamiseen tietokannan ja ohjelman käyttöliittymän välillä sekä käyttöliittymäkomponenttien muokkaamiseen.

## 4.2 Vähäkoodiversio

Prototyyppisovelluksen vähäkoodiversio tehtiin AppSheetillä. Sovelluksen käyttöliittymän ja toimintojen ytimenä toimi pohjalle valittu tietolähde, ja AppSheet tukeekin lukuisia erilaisia tietokantoja ja tietolähteitä, kuten MySQL, Google Sheets ja Zapier [Common questions when considering Google AppSheet 2022].

Tässä projektissa käytettiin Google Sheets -taulukkoa tietokantana, koska sen kanssa ei tarvitse huolehtia ylläpidosta eikä tietokantarakenteista. Tietokanta on yleisestikin ohjelmistokehityksessä sovelluksen tausta ja tukipilari, mutta AppSheetissä sovelluksen lähtöpiste rakentuu täysin tietokannan mukaisesti. Tietokantamallia joutuu pienessäkin projektissa suunnittelemaan varsin pitkälle heti aluksi, jotta sovelluksen konfiguroiminen olisi kehitysvaiheessa mahdollisimman suoraviivaista. AppSheet luo kaikki muuttujat ja näkymät automaattisesti tietokannan pohjalta uutta sovellusta aloittaessa. Tietokannan rakennetta on kuitenkin mahdollista muokata myös kesken kehityksen, mutta jälkikäteen tehtävät muutokset voivat johtaa isoihin päivitystarpeisiin sovelluksen käyttöliittymässä ja logiikassa. Prototyyppisovelluksen tietokantamalli muodostuu vain muutamista osista, sillä siihen kuuluu tavoite, kunkin kolme avaintulosta, avaintulosten tilat ja tavoitteen omistaja. Kuvassa 5 näkyy tietokantana toimiva Google Sheet -laskentataulukko esimerkkitavoite syötettynä.




| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Objective | KR1 | KR2 | KR3 | KR1Status | KR2Status | KR3Status | Owner | KRAvg |
| 2 | Increase sales | Make 10 deals | Make 20 offers | Create 3 ad cam | 0.6 | 0.3 | 0.3 | Tester | |
| 3 | | | | | | | | | |

Kuva 4. Tietokantana toimiva Google Sheet -tiedosto.

Uuden sovelluksen rakentaminen aloitetaan datalähteen valinnalla. Valinnan jälkeen AppSheet luo automaattisesti omien päätelmiensä mukaisesti oletusnäkymän ja lisätietokentät sarakkeille. Sovelluksen käyttöliittymän pohja, navigointi ja tietokantayhteys muodostuvat automaattisesti, minkä jälkeen konfiguroimisen voi aloittaa.

Sovellus on myös oletuksena suojattu ja yksityinen, eli siihen ei pääse muilla kuin projektin aloittaneen kehittäjän käyttäjätunnuksilla. AppSheet tukee kirjautumisia Apple-, Box-, Dropbox-, Google-, Microsoft-, Salesforce- ja Smartsheet-käyttäjätunnuksia käyttäen [Montoya 2022]. Sovellukseen kirjautuminen on aina pakollista jollain edellä mainituista palveluista, ellei sovelluksesta tee täysin julkista. Käyttäjien pääsy sallitaan lisäämällä valittujen palveluiden alaisten käyttäjätilien sähköpostiosoitteet tai käyttäjätunnukset. Mikäli sovelluksesta tekee täysin julkisen, siinä ei voi olla käytössä mitään käyttäjäkohtaisia toimintoja. AppSheetissä ei ole mahdollista luoda omia paikallisia käyttäjiä, eli käyttäjillä on oltava jonkin AppSheetin tukeman palvelun tunnukset käytössään. Valmiiden käyttäjätilien hyödyntäminen toimii hyvin silloin, kun käyttäjäorganisaatiolla on jokin yhteensopiva palvelu käytössä, kuten Google Workspace tai Microsoft 365. Mikäli mikään näistä palveluista ei ole käytössä, niiden hankkiminen vain AppSheetiin kirjautumista varten nostaa kustannuksia merkittävästi.

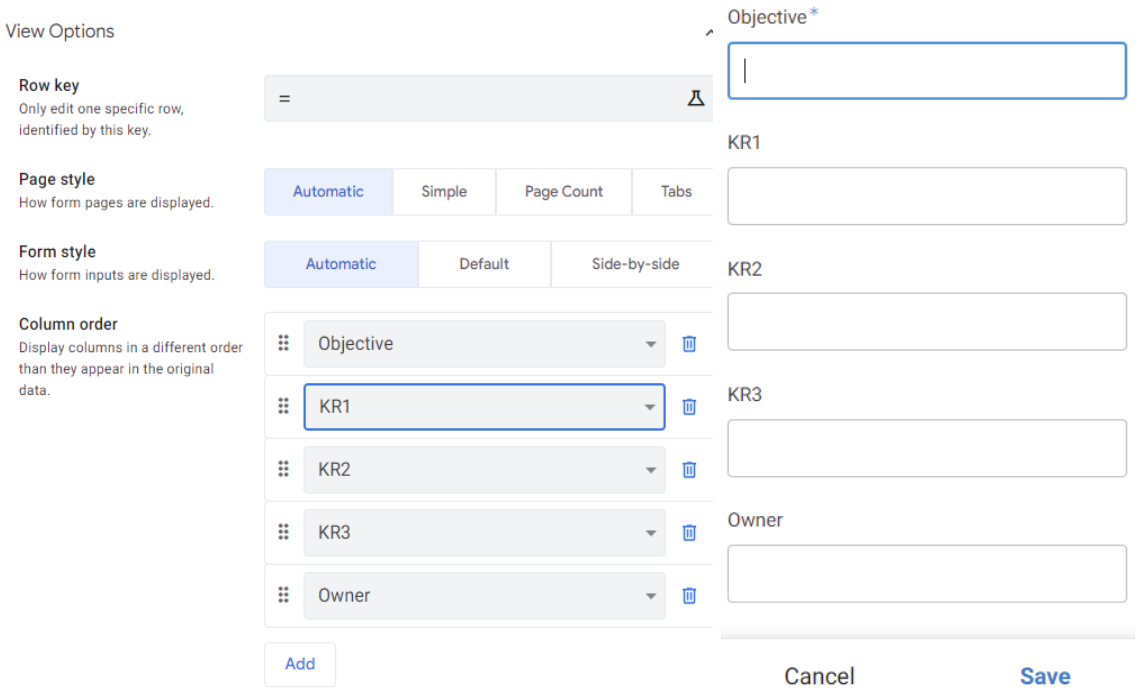
### 4.2.1 Käyttöliittymä

Käyttöliittymän oletusrakenne muodostuu automaattisesti näkymiä luodessa. Myös navigointipalkit ja eri sivujen keskinäinen linkitys alustuvat ja päivittyvät

sitä mukaa, kuin näkymiä luodaan. Käyttöliittymän toiminnot ja ulkoasu ovat kuitenkin muokattavissa. Oletuksena AppSheet tarjoaa mobiilisovelluksille hyvin tyypillistä näkymää, jossa sisältö renderöidään pystysuunnassa ja navigointipalkki on vaakatasossa alhaalla. AppSheet on suunniteltu käytettäväksi älypuhelimien lisäksi myös tableteilla ja tietokoneiden selaimilla. Sovellus on aina skaalautuva, mutta kehittäjän kannattaa silti huomioida mahdollinen pääasiallinen käyttötapa käyttöliittymää suunniteltaessa. Esimerkiksi mikäli älypuhelin on pääasiallinen tapa käyttää sovellusta, yhteen komponenttiin kannattaa laittaa rajallisemmin sisältöä. Vastaavasti tietokonekäyttöön suunnitellulla sovelluksella voi vapaammin lisäillä sisältöjä yhteen näkymään tai elementtiin. Mobiilinäkymän priorisointi sopi tämän työn sovelluksen tarpeisiin täysin, sillä avaintuloksia oli tarkoitus päivittää nopeasti älypuhelimia käyttäen.

Sovelluksessa piti pystyä syöttämään tietoa ja lukemaan jo olemassa olevaa sisältöä. AppSheet tekee automaattisesti näkymät (views) tietokannan sarakkeiden perusteella. Päänäkymäksi muodostuu listausnäkymä tavoitteista, sillä tavoitteet ovat tietokannan avainarvoja. Oletuksena muodostuvaa tavoitelistaa pitää kuitenkin muokata konfiguroimalla, jotta se palvelee sovelluksen käyttäjiä halutulla tavalla.

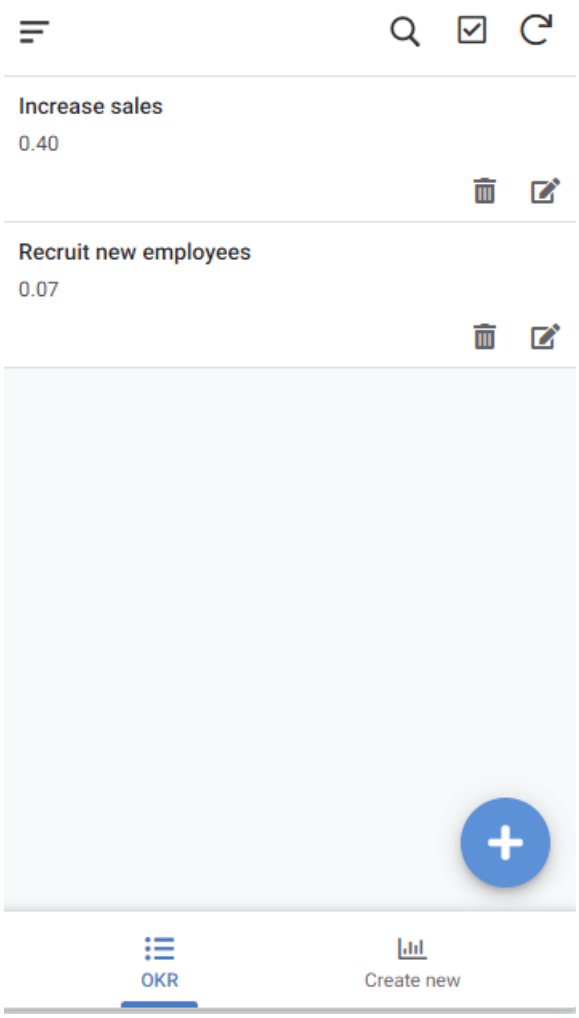
Tiedon syöttämisen lomaketta varten luotiin uusi erillinen näkymänsä. Näkymän tyypiksi valittiin alustan tarjoamista vaihtoehdoista lomake, jonka myötä alusta luo täytettävät kentät kaikille tietokannassa oleville kohteille. Sovelluksessa haluttiin käyttäjän syöttävän vain yhden tavoitteen ja korkeintaan kolme avaintulosta kerrallaan, joten muut lomakkeen kentät poistettiin käytöstä. Kuvassa 6 näkyy valmiin lomakkeen konfiguraatio ja käyttöliittymä.




Kuva 5. Valmiin lomakkeen konfigurointinäkymä vasemmalla ja käyttöliittymänäkymä oikealla.

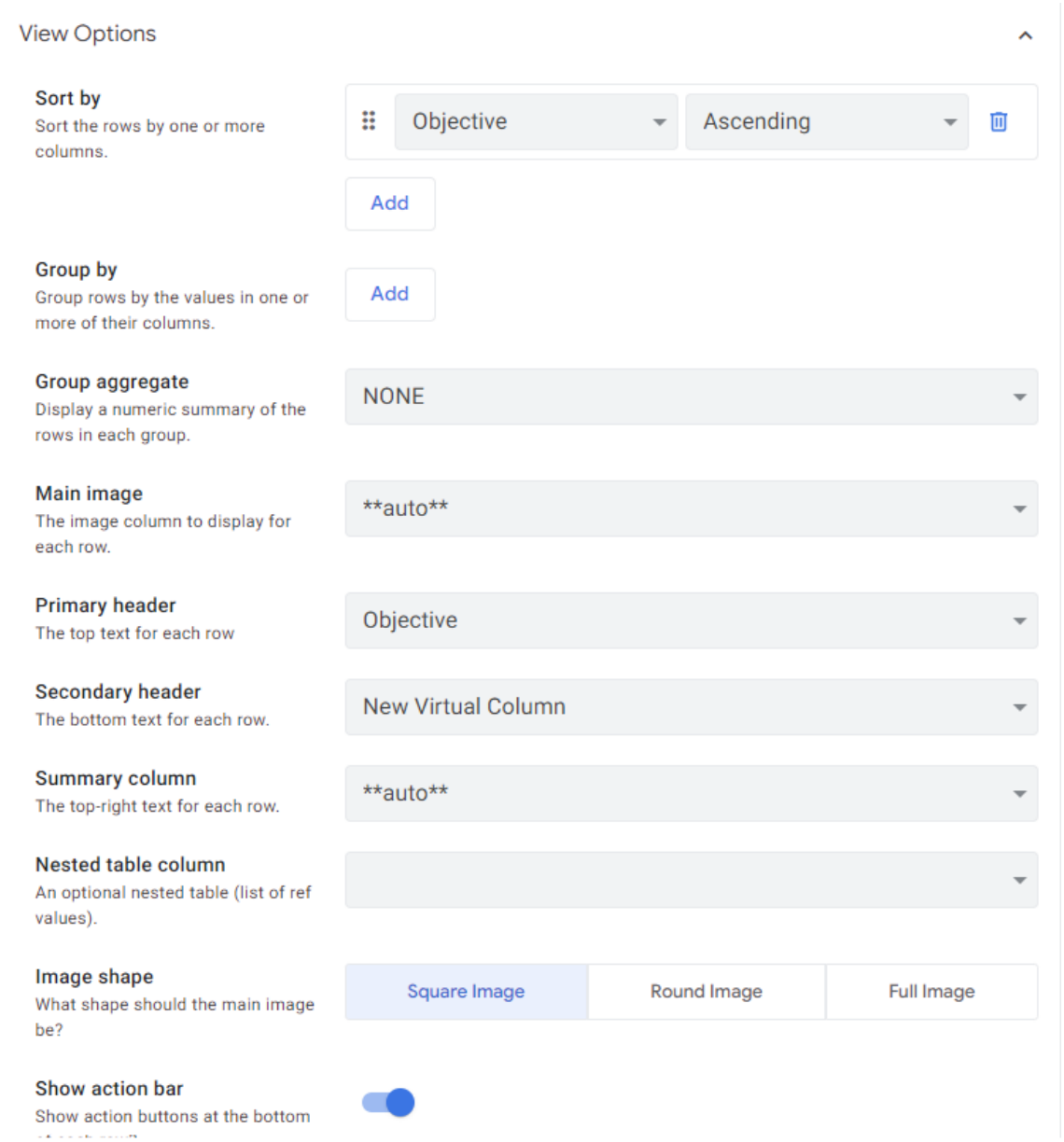
Sovelluksen päänäkymä, eli tavoitelistaus, oli muokattava paremmin tarkoitukseen sopivaksi. Listanäkymään haluttiin suoraan näkyväksi tavoitteen otsikko sekä keskiarvo avaintulosten etenemisestä.

Käyttöliittymään ei tarvinnut tässä tehdä muutoksia lähes ollenkaan, ja ainoat muutokset olivat näkymien nimien muuttaminen kuvaavammiksi sekä navigointipalkin kuvakkeiden valitseminen. Muutokset tehdään kunkin näkymän näyttöasetuksista ja kuvakkeet valitaan AppSheetin omasta galleriasta. Kuvassa 6 näkyy valmiin sovelluksen listausnäkymä.




Kuva 6. Valmiin sovelluksen listausnäkymä.

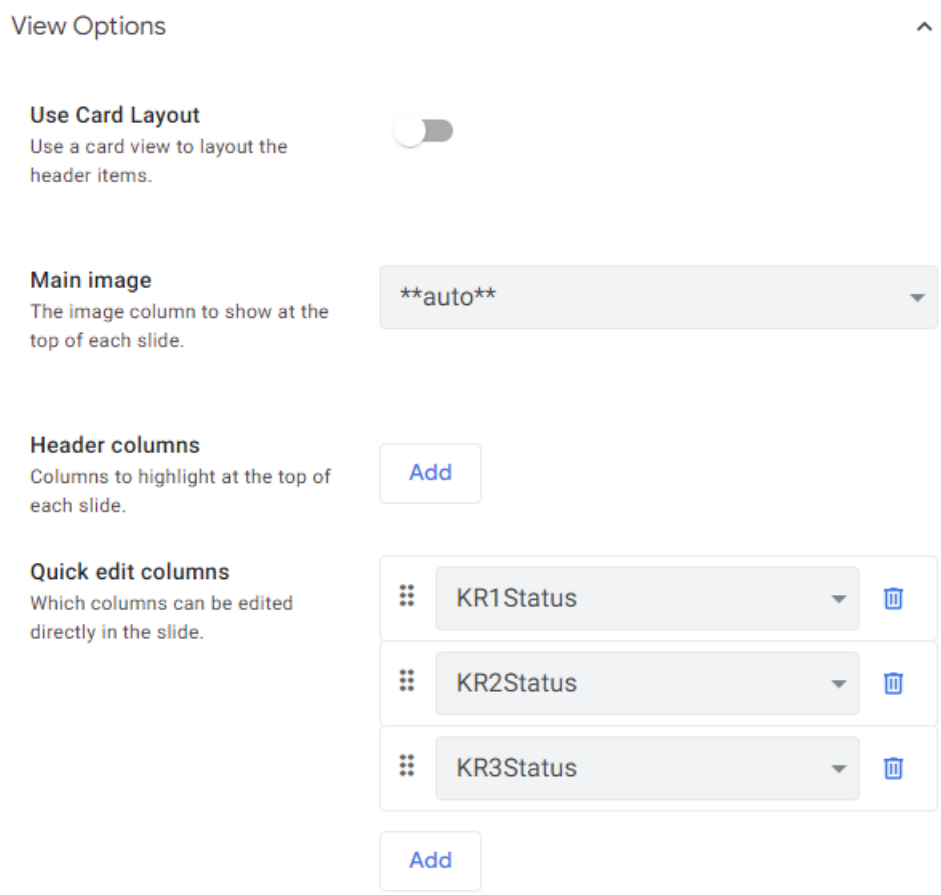
Tavoitelistanäkymää konfiguroitiin niin, että listassa näkyy suoraan tavoitteen nimi ja keskiarvo. Lista muodostuu puolestaan kokoelmasta tavoitekortteja, ja ne järjesteltiin näkymään otsikon mukaan aakkosjärjestyksessä. Listan kor-teissa on kussakin toimintanäppäimet, joiden avulla käyttäjä voi muokata tai poistaa tavoitteita. Kuvassa 7 on esitetty näkymän asetusten tila konfiguroinnin jälkeen.




Kuva 7. Tavoitelistauksen näkymän konfigurointi.

Yksittäisiä tavoitteita voi tarkastella napauttamalla listassa olevaa korttia. AppSheet luo lähtökohtaisesti kaikkiin elementteihin mahdollisuuden porautua alaspäin, kuten tässäkin. Kortin takaa löytyy kokonaisuudessaan se sisältö, joka tietokannassa vastaa valittua avainta. Tämän sovelluksen tapauksessa toiminto vastaa tarvetta suoraan ja ainoa mukautus tehdään käyttäjän muokattavissa oleviin kenttiin. Oletuksena kaikki kentät ovat muokattavissa, mutta tässä ne ra-

jataan vain avaintulosten statuskenttiin. Näin käyttäjät voivat päivittää avaintulosten edistymistä, mutta itse tavoitteita ja avaintuloksia ei voi muuttaa kesken seurantajakson. Kuvassa 8 näkyy konfiguraationäkymä rajausten jälkeen.

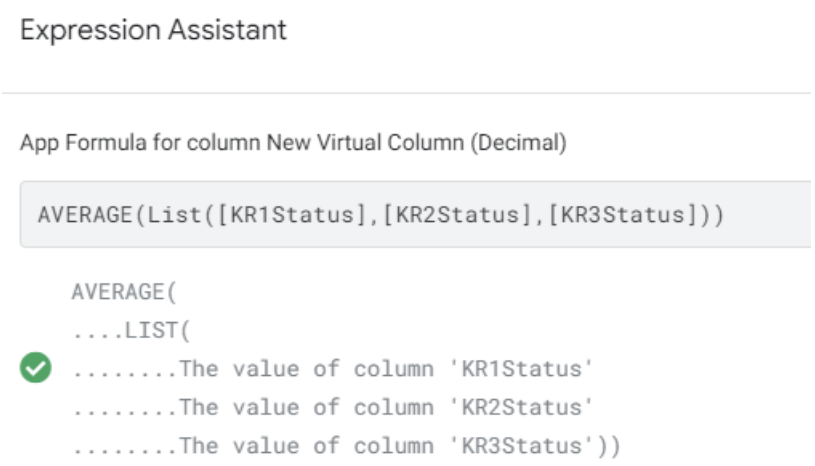



Kuva 8. Tavoitteen lisätietojen rajaus.

### 4.2.2 Sovelluslogiikka ja tiedonhallinta

Sovelluksessa tuli pystyä tallentamaan tietoa ja hakemaan sitä tietokantana olleesta Google Sheets -tiedostosta. AppSheet luo nämä toiminnot automaattisesti, kun tietoa lukevia tai tallentavia näkymiä luodaan käyttöliittymän puolelle. Sovelluslogiikkaa ei tarvitse niiden suhteen erikseen konfiguroida. Ainut tähän sovellukseen käsin tehtävä asia oli avaintulosten edistymisten keskiarvon laske-

minen. Laskentaan perustuva logiikka tehtiin AppSheetissä luomalla uusi virtuaalinen sarake datamalliin. Kuvassa 9 on virtuaaliseksi sarakkeeksi tehty keskiarvon laskemisen kaava.




Kuva 9. Virtuaalisarakkeen luontityökalu.

AppSheetissä sovelluslogiikkaa tehdään käytös- (behavior) ja automaatiokonfiguraatioilla. Näitä käytetään esimerkiksi silloin, kun halutaan käyttäjän siirtyvän poikkeavaan näkymään tai kun tietoa siirretään sovelluksen ulkopuolelle. Tässä sovelluksessa tarpeita tällaisille lisätoiminnoille ei ollut, joten käsin määritelty logiikka jäi lähes olemattomaksi.

# 5 Tulokset

## 5.1 Kehitystyö ja työkalut

Perinteisen web-sovelluskehityksen aloittamiseksi on tekijöillä oltava kattavat taidot valituista tekniikoista, ohjelmointikielistä ja työkaluista. Usein kehittäjät joutuvat opiskelemaan uusia projektikohtaisia asioita, kuten käytettävät kirjastot, ja käymään niitä läpi mahdollisten samaan projektiin kuuluvien muiden kehittäjien kanssa. Ennen ohjelmointityön aloittamista sovelluksen runko datarakenteineen on suunniteltava mahdollisimman tarkasti, jotta työn eteneminen olisi sujuvaa. Projektin koon mukaan tässä voi mennä paljonkin aikaa, mutta alustavien

toimien on tarkoitus vähentää vikoja ja niihin liittyviä ylimääräisiä korjaustöitä. Lisäksi tässä vaiheessa viimeistään saadaan tarkempi arvio tarvittavista resursseista [Budgen 2003: 55].

Insinöörityössä tehdyn prototyyppisovelluksen kohdalla pohjatöihin kului aikaa kolmisen tuntia. Se on lähes yhtä kauan, kuin koko AppSheet-projektissa kesti ensikirjautumisesta valmiiseen toteutukseen. Pohjatöiden ja sovelluksen perusrungon valmistumisen jälkeen loppuprojekti sujui tasaisesti, mutta tähän pieneenkin sovellukseen mahtui jonkin verran aikaa vievää ongelmanselvitystä mm. tietokannan tietojen päivittämiseen liittyen.

Vähäkoodisen ohjelmistokehityksen osalta taas työt voi aloittaa useimmiten heti käyttäjätilin luomisen jälkeen ja alusta itsessään huolehtii sovelluksen pohjan luomisesta. Näin on etenkin vertailuun valitun AppSheet-alustan kohdalla, mutta samanlaisia ominaisuuksia on myös monilla muilla palveluilla [Tozzi 2021]. Alustoissa ylläpito, virheidenseuranta ja datarakenne muodostuvat automaattisesti. Automaattisesti luodut navigointivalikot, listat ja yksityiskohtanäkymät kantavat monessa projektissa jo pitkälle, ja käsin tehtävän määrittelyn tarve voi jäädä hyvinkin vähäiseksi. Prototyyppisovelluksessa suurin osa lopputuloksesta on alustan datan pohjalta automaattisesti luomaa.

Kun työkalujen ja sovelluksen suunnitelmat ovat kunnossa, pääsee myös perinteinen ohjelmistokehitys vauhtiin. Hyödyntämällä olemassa olevia kirjastoja, kuten vertailuun nostettua Reactia, perustoiminnot saa luotua nopeasti valmiita malleja hyödyntäen. Kehittäjä pystyy kuitenkin vapaasti muokkaamaan sovelluksen komponentteja tai korvaamaan ne omillaan, jolloin valmisratkaisut eivät ole esteenä oman sovelluksen tavoitteiden saavuttamiseksi. Ulkoasu ja grafiikka on myös hoidettava sovellukseen itse, vaikkakin käyttöliittymän luomiseenkin löytyy kirjastoja kehittäjän avuksi. Kehittäjät voivat osaamisensa puitteissa tehdä ne itse, tai työn voi ulkoistaa yhteistyökumppanille tai erikseen nimitetylle suunnittelijalle. React-sovelluksen kehittämiseen kului yhteensä noin 15 tuntia, eli melkein kaksi työpäivää. Työssä syntynyt koodi on tarkasteltavissa liitteen 1 GitHub-linkin kautta.

Vähäkoodisella alustalla vauhti säilyy koko kehityksen ajan, mutta valitun alustan mukaan mukauttamisen taso ja viimeistely voivat jäädä rajallisiksi [Tozzi 2021]. AppSheet-prototyyppisovellus syntyi yhden iltapäivän aikana, vajaassa neljässä tunnissa, joten nopeuden osalta vähäkoodinen kehitys lunasti lupauksensa hienosti. Kehittäjän palkka on usein merkittävä osa kehityskustannuksia, joten kehitysajan kutistuminen neljännekseen on merkittävä etu. Nopea kehitysaika itsessäänkin pienentää kynnystä käynnistää projekteja, sillä tuloksia pääsee kokeilemaan käytännössä hyvin nopeasti.

Työkalun omat ominaisuudet rajoittavat sillä tehtäviä toimintoja, ja työkalujen kehittyminen on sen omista kehittäjistä riippuvainen asia. Osa alustoista sallii kuitenkin oman koodin lisäämisen ja siten vapaamman mukauttamisen, mutta esimerkiksi valittu AppSheet ei sisällä tällaista ominaisuutta.

## 5.2 Valmiiden sovellusten vertailu

Insinöörityössä tehdyt perinteisen web-sovelluskehityksen ja vähäkoodialustan prototyyppisovellukset tekevät samat asiat, joten käyttötarkoitusta vastaavan sovelluslogiikan osalta sovellukset ovat toisiaan vastaavia. Muissa ominaisuuksissa ja käyttöliittymän sekä yleisen viimeistelyn osalta AppSheet-versio on kuitenkin huomattavasti edistyneempi. React-sovellus on prototyyppitasolla, eli käytännössä se toimii vain kehittäjän tietokoneella paikallisesti, käyttäen paikallista tietokantaa. Siinä ei myöskään ole kirjautumista, salauksia tai muita tietoturvaominaisuuksia.

AppSheet-sovellus taas on projektin aloittamisesta saakka ylläpidossa Googlen pilvialustalla. Siinä on myös tuki käyttäjäprofiileille ja kirjautumiselle eri alustojen kautta, ja se on suojattu alustan toimesta. Kaikki nämä ominaisuudet aktivoituivat automaattisesti sovelluksen luontivaiheessa. AppSheet-sovelluksen voisi ottaa nykyisessä tilassaan työpaikalla käyttöön vaikka heti, kunhan vain siihen syöttäisi hyväksyttävät käyttäjät tai toimialueet ja ostaisi niitä vastaavan määrän käyttölisenssejä.

## 5.3 Jatkokehitysmahdollisuudet

Käsin koodattu web-sovellus vaatii myös manuaalista ylläpitotyötä, sillä siihen sidoksissa olevat ja sen ympärillä olevat tekniikat kehittyvät vääjäämättä. Lisäksi käytössä voi ilmetä vikatilanteita tai puutteita, jotka kehitysvaiheessa ovat jääneet huomaamatta. Päivityksiä voidaan tehdä myös silloin, kun sovellukselle haetaan kasvua uusien ominaisuuksien ja sitä myötä uusien käyttäjien toivossa [Blair 2018]. Nämä asiat on otettava huomioon jälkisuunnittelussa ja vastuut sovittava sidosryhmien kesken.

Useimmiten vähäkoodialustat hoitavat turvapäivitykset ja vianseurannan automaattisesti. Esimerkiksi AppSheet on kokonaisuudessaan Google Cloud -alustalla, jossa Google huolehtii palvelun toiminnasta ja tietoturvasta [Trusted infrastructure 2022]. Sovellusta tarvitsee lähtökohtaisesti päivittää vain, mikäli on tarve muokata ominaisuuksia tai jokin sovellukseen liitetty ulkopuolinen kolmannen osapuolen palvelu, kuten datalähde, muuttuu sovelluksen toimintaan vaikuttavalla tavalla. AppSheet-prototyyppisovelluksessa on jo valmiina navigointi ja näkymien eriyttäminen, joten sovelluksen jatkokehitystä varten on jo runko olemassa. Riittää, että lisää datalähteelle mahdolliset uudet tietosarakkeet ja luo tarvitsemansa sivut käyttöliittymään tietojen pohjalta.

ReactJS-projektissa on enemmän työtä viedä sovellus seuraavalle tasolle ja kohti julkaisuvalmista sovellusta. Varsinaista rajaa sen jatkokehitykselle ei kuitenkaan ole, sillä itse ohjelmoimalla voi teoriassa tuoda mitä tahansa uusia ominaisuuksia sovellukseen.

AppSheetin ja ReactJS:n välillä ei voi vaihtaa ristiin, eli niiden koodi, tietokanta tai muutkaan osat eivät ole lainkaan keskenään yhteensopivia. AppSheetistä ei voi vaihtaa edes toiseen vähäkoodialustaan, joten alustan toimivuudesta omaan projektiin kannattaa olla varma. Sovellus on siis täysin AppSheetin toimivuuden ja olemassaolon varassa, eikä kehittäjä voi vaikuttaa siihen mitenkään.

ReactJS-projektin voi teoriassa vaihtaa toiseen JavaScript-kirjastoon perustuvaksi, mutta se ei ole automaattinen tai edes helppo prosessi. Käyttöliittymäkomponentit vähintään on rakennettava uudelleen, mutta tyylittelyjä ja sovelluslogiikkaa voi paikoin uudelleenkäyttää. Tietokannat ja taustajärjestelmät ovat kuitenkin lähtökohtaisesti yhteensopivia kaikkien käyttöliittymäkirjastojen kanssa, ja esimerkiksi prototyyppisovelluksen JSON Server sopisi suoraan vaikkapa Vue.js-projektiin. Uuden alustan kanssa saa siis ikään kuin lentävän lähdön, mutta ei suoraa muunnosta.

# 6 Yhteenveto

Kumpaankin kehitystapaan tutustuttua, perinteiseen web-sovelluskehitykseen ja vähäkoodialustoihin, työkalujen, työvaiheiden ja kustannusten näkökulmasta voi todeta, että kumpikaan kehitystapa ei ole yksiselitteisesti toista parempi. Yllättävää kuitenkin on se, että vähäkoodialustat ovat varsin nopeasti nousseet varteenotettavaksi vaihtoehdoksi. Moni pienemmän mittakaavan toteutus kannattanee jatkossa tehdä konfiguraatiotyökaluilla totutun käsin koodaamisen sijasta, mutta joka tapauksessa vaihtoehto kannattaa pitää mielessä mitä tahansa sovellusprojektia käynnistettäessä.

Vähäkoodialustoja on tullut saataville kasvavalla tahdilla, ja olemassa olevia ratkaisuja on kehitetty jatkuvasti. Monia palveluita on myös sulautettu isompiin toimijoihin yrityskauppojen seurauksena. Samaan aikaan myös perinteistä ohjelmistokehitystä kehitetään uusilla ohjelmointikielillä ja kirjastoilla. Ohjelmistoala on ollut jatkuvassa kasvussa ja vielä lähitulevaisuudessakin osaajia tarvitaan, jatkossa tosin työpaikkailmoituksissa voi olla mainintoja myös vähäkoodialustaosaamisesta.

Perinteinen sovelluskehitys ei kuitenkaan näillä näkymin ole kokonaan katoamassa, ja koodareille riittää töitä jatkossakin. Heti, kun sovellukselle haetaan erittäin yksilöllisiä toimintoja, erityistä keveyttä tai suurta ominaisuuksien kirjoa, on kutakuinkin pakko tehdä sovellus käsin koodaamalla. Lopputulos voi myös olla parempi pienemmissäkin toteutuksissa, mutta silloin tulee arvioida hinta-

hyötysuhdetta. Toisin sanoen: Ovatko nopeus- ja viimeistelyerot lisätyöpäivien arvoisia? Entä jos ero kehitysajoissa onkin kuukausia?

Näyttää siltä, että varsinkin nyt yleistyvän etätyöskentelyn aikana erilaisilla yrityksillä tulee nopeasti tarve digitalisoida manuaalisia prosessejaan. Tähän ongelmaan vähäkoodialustat tuovat nopean ratkaisun, ja todennäköisesti alan toimijat panostavat työkalujen helppokäyttöisyyteen ja lisäominaisuuksiin jatkossa entisestään.

**Lähteet**

2021 Developer Survey. 2022. Verkkoaineisto. Stack Overflow. <https://insights.stackoverflow.com/survey/2021>. Luettu 20.4.2022.

Alexander, Forsyth. 2021. What Is Low-Code? [2021 Update]. Verkkoaineisto. Outsystems. <https://www.outsystems.com/blog/posts/what-is-low-code/>. Luettu 2.3.2021.

Appian Low-Code Platform. 2022. Verkkoaineisto. Appian. <https://appian.com/platform/overview.html>. Luettu 20.4.2022.

AppSheet – Pricing. 2021. Verkkoaineisto. AppSheet. <https://solutions.appsheet.com/pricing>. Luettu 1.3.2021.

Blair, Ian. 2018. How Often Should You Update Your App. Verkkoaineisto. Buildfire. <https://buildfire.com/how-often-should-you-update-your-app/>. Luettu 2.3.2021.

Borrelli, Piero. 2021. Angular vs. React vs. Vue.js: Comparing performance. Verkkoaineisto. LogRocket. <https://blog.logrocket.com/angular-vs-react-vs-vue-js-comparing-performance/>. Luettu 11.4.2022.

Budgen, David. 2003. Software design. E-kirja. Addison-Wesley.

Common questions when considering Google AppSheet. 2022. Verkkoaineisto. AppSheet. <https://about.appsheet.com/faq/>. Luettu 5.5.2022.

Dayal, Nitish. Create-React-App: A Closer Look. Verkkoaineisto. GitHub. <https://github.com/nitishdayal/cra_closer_look>. Luettu 21.3.2022.

Enable everyone in your organization to build and extend applications without coding. 2021. Verkkoaineisto. Google. <https://cloud.google.com/appsheet>. Luettu 1.3.2021.

Google acquires AppSheet to help businesses create and extend applications—without coding. 2020. Verkkoaineisto. Google. <https://cloud.google.com/blog/topics/inside-google-cloud/helping-businesses-create-and-extend-applications-without-coding>. Luettu 5.5.2022.

Google Trends – Hakumäärät ajan mittaan. 2021. Verkkoaineisto. Google. < https://trends.google.com/trends/explore?geo=US&q=React,AngularJS,VueJS>. Luettu 28.2.2021.

Hooks at a Glance. 2022. Verkkoaineisto. ReactJS. < https://reactjs.org/docs/hooks-overview.html>. Luettu 22.3.2022.

JSON Server. 2022. Verkkoaineisto. Npm. <https://www.npmjs.com/package/json-server>. Luettu 21.3.2022.

Kotilainen, Samuli. 2019. Low-coden nopeus yllätti – ”ei olisi ollut ikinä mahdollista ilman”. Verkkoaineisto. Tivi. <https://www.tivi.fi/uutiset/low-coden-nopeus-yllatti-ei-olisi-ollut-ikina-mahdollista-ilman/f5f77edc-41ab-4354-8aaa-f951369ba899>. Luettu 5.5.2022.

Lardinois, Lunden. 2018. Microsoft has acquired GitHub for $7.5B in stock. Verkkoaineisto. TechCrunch. <https://techcrunch.com/2018/06/04/microsoft-has-acquired-github-for-7-5b-in-microsoft-stock>. Luettu 10.10.2021.

Marvin, Rob. 2018. The Best Low-Code Development Platforms. Verkkoaineisto. PCMag UK. <https://uk.pcmag.com/migrated-46739-onlinecloud-backup-services/89789/the-best-low-code-development-platforms>. Luettu 2.3.2021.

Montoya, Santiago Uribe. 2022. Require Sign-In: The Essentials. Verkkoaineisto. AppSheet. <https://help.appsheet.com/en/articles/954491-require-sign-in-the-essentials>. Luettu 9.3.2022.

Overview of creating apps in Power Apps. 2022. Verkkoaineisto. Microsoft. <https://docs.microsoft.com/en-gb/power-apps/maker/>. Luettu 11.4.2022.

Nickelburg, Monica. 2015. Startup Spotlight: AppSheet lets non-developers build custom mobile apps. Verkkoaineisto. GeekWire. <https://www.geekwire.com/2015/appsheet/>. Luettu 2.3.2021.

Node.js. 2021. Verkkoaineisto. Node.js. <https://github.com/nodejs/node> Luettu 22.4.2021.

Occhino, Tom & Walke, Jordan. 2013. JS Apps at Facebook. Verkkoaineisto. YouTube. < https://www.youtube.com/watch?v=GW0rj4sNH2w>. Luettu 18.2.2021.

Panchadsaram, Ryan. 2022. What is an OKR? Definition and Examples. Verkkoaineisto. What Matters. <https://www.whatmatters.com/faqs/okr-meaning-definition-example>. Luettu 20.2.2022.

Pesquet, Baptiste. 2020. The JavaScript Way. E-kirja. AFNIL.

Programming Languages. 2021. Verkkoaineisto. Microsoft. <https://code.visualstudio.com/docs/languages/overview>. Luettu 2.3.2021.

React – Tutorial. 2021. Verkkoaineisto. ReactJS. <https://reactjs.org/tutorial/tutorial.html>. Luettu 2.3.2021.

Stack Overflow Insights - Trends. 2021. Verkkoaineisto. Stack Overflow. https://insights.stackoverflow.com/trends?tags=reactjs%2Cvue.js%2Cangular%2Cangularjs. Luettu 28.2.2021.

Stevens, Emily. 2020. Should you learn JavaScript? Advice For Newbie Web Developers. Verkkoaineisto. CareerFoundry. <https://careerfoundry.com/en/blog/web-development/should-you-learn-javascript>. Luettu 2.3.2021.

The fastest way to build apps and automate work. 2022. Verkkoaineisto. AppSheet. <https://about.appsheet.com/home/>. Luettu 5.5.2022.

Tozzi, Chris. 2020. A practical take on low-code vs. traditional development. Verkkoaineisto. TechTarget. <https://searchsoftwarequality.techtarget.com/tip/A-practical-take-on-low-code-vs-traditional-development>. Luettu 2.3.2021.

Toledo, Judi. 2018. Why Millions of Developers use JavaScript for Web Application Development. Verkkoaineisto. Torque. <https://torquemag.io/2018/06/why-millions-of-developers-use-javascript-for-web-application-development/>. Luettu 2.3.2021.

Trusted infrastructure. 2022. Verkkoaineisto. Google. <https://cloud.google.com/security/infrastructure>. Luettu 5.5.2022.

What is open source? 2022. Verkkoaineisto. Red Hat, Inc. <https://opensource.com/resources/what-open-source>. Luettu 20.2.2022.

## ReactJS -projektin lähdekoodi

Linkki GitHub-varastoon: https://github.com/dawcules/inssi