Tuomas Rajala

Web-sovelluksen päivittäminen Vue.js-ohjelmistokehyksellä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

24.5.2023

Tiivistelmä

Tekijä:	Tuomas Rajala
Otsikko:	Web-sovelluksen päivittäminen Vue.js-ohjelmistokehyksellä
Sivumäärä:	34 sivua
Aika:	24.5.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Ilpo Kuivanen Head of Dynamo Pasi Nummisalo

---

Tämän insinöörityön tavoitteena oli kehittää päivitetty versio nykyisin käytössä olevasta web-pohjaisesta käyttöliittymäsovelluksesta, joka kykenisi suoriutumaan sille vaadituista ydintoiminnoista. Työ on Documill-nimisen suomalaisen yrityksen sisäinen hanke Dynamo-tuotteelle. Dynamo on yksinkertaisuudessaan dynaamisten dokumenttien luomiseen tarkoitettu pilvipalvelusovellus, joka mahdollistaa muun muassa sähköisen allekirjoituksen PDF-tiedostoihin. Nykyisin käytössä olevan toteutuksen ohjelmistokehyksen tuki ja jatkokehitys on päättynyt. Tämä loi osaltaan motivaation kehittää täysin uusi sovellus, joka kuitenkin kykenee suoriutumaan samoista toiminnoista kuin nykyinen toteutus.

Työn aikana tutustuttiin ja otettiin käyttöön useita eri teknologioita, jotka mahdollistavat sovelluksen toteutuksen. Insinöörityössä saatiin luotua sovellus, joka täyttää sen primääritavoitteen. Tähän kuuluvat sovelluksen aloitus, prosessin tilan kyseleminen rajapinnasta, saapuvan datan oikea käsittely ja renderöinti sekä prosessin jatkaminen. Tämän lisäksi sovelluksen muitakin ominaisuuksia lisättiin toteutukseen. Käyttöliittymäsovelluksen lisäksi työhön sisältyvät sovelluksen ulkoisten rajapintojen määritykset sekä valmiin paketin vieminen hosting-palvelimelle.

Sovellus vaatii vielä joitakin kriittisiä ominaisuuksia, jotta se saataisiin tuotantokuntoon. Sovelluksen on oltava taaksepäin yhteensopiva nykyisen kanssa. Tämän varmistamiseksi tarvitaan kattavat testitapaukset. Myös mahdolliset optimoinnit on tehtävä suorituskyvyn varmistamiseksi.

Avainsanat: Vue.js, AWS, Dynamo, web-kehitys

Abstract

Author: Tuomas Rajala
Title: Development of Modern Web Application with Vue.js
Number of Pages: 34 pages
Date: 24 May 2023

Degree: Bachelor of Engineering
Degree Programme: Information and Communication Technology
Professional Major: Software Engineering
Supervisors: Ilpo Kuivanen, Senior Lecturer
Pasi Nummisalo, Head of Dynamo

The objective of the study was to develop an updated version of the currently used web-based application, which would be capable of performing its required core functions. The project is an internal project of Documill for their Dynamo product. In its simplicity, Dynamo is a cloud-based application designed to create dynamic documents and enabling e-signatures for PDF-files. Support and further development of the currently used software framework have ended, which created a motivation to develop a completely new application that can still perform the same functions as the current implementation.

During the project, several different technologies were learned and implemented to enable its realization. The project resulted in the creation of an application that fulfills its primary objectives including application startup, querying the process status from the interface, correct handling and rendering of incoming data, and continuation of the process. In addition, other features of the application were included. The study also included defining the external interfaces of the application and deploying the final package to a hosting service.

In order to make the updated version production-ready, it still requires some critical features. The application must be backward compatible with the current one. To ensure this, comprehensive test cases are needed. Any potential optimizations also need to be made to ensure performance.

Keywords: Vue.js, AWS, Dynamo, web development

Sisällys

Lyhenteet

1 Johdanto 1

2 Taustaa 2

2.1 Dynamo 2
2.1.1 Dynamo API-käytössä 4
2.1.2 Dynamo GUI-käytössä 4
2.2 Miksi uusi ohjelmistokehys? 5
2.2.1 Hyöty yritykselle 5
2.2.2 Hyöty loppukäyttäjälle 6

3 Kehitysteknologioiden esittelyä 7

3.1 Vue.js 7
3.1.1 Vertailua muihin kehyksiin 7
3.1.2 Syitä valinnalle 8
3.1.3 Versiot 9
3.2 Vite 11
3.3 Vitest 11
3.4 Amazon Web Services 12
3.4.1 AWS Cloud Development Kit 13
3.4.2 AWS Lambda 14
3.4.3 Amazon API Gateway 15

4 Kehitysprosessi 15

4.1 Lähtökohdat ja vaatimusmäärittely 15
4.2 Sovelluksen rakenteellinen muutos 17
4.3 Sovelluksen logiikka 19
4.3.1 Sovelluksen ja palvelimen kommunikaatio 19
4.3.2 Sovelluksen ajon aloittaminen 19
4.3.3 Sovelluksen sisäinen toimintalogiikka 21
4.3.4 Ajon jatkaminen 23
4.3.5 Ajonaikaisia toiminnallisuuksia 24
4.4 Testaus 25
4.5 Tuotos 26

4.6 Haasteet 27
4.7 Jatkokehitys 28

5 Yhteenveto 29

Lähteet 31

Lyhenteet

API: *Application Programming Interface.* Ohjelmointirajapinta kahden eri järjestelmän komponenttien välillä, mikä mahdollistaa niiden välisen keskustelun.

AWS: *Amazon Web Services.* Amazonin omistama pilvipalvelualusta.

CDK: *Cloud Development Kit*. Pilvisovellusten kehitykseen keskittynyt ohjelmistokehys.

CLI: *Command Line Interface*. Komentopohjainen ihmisen ja tietokoneen välinen käyttöliittymä.

CRM: *Customer Relationship Management*. Asiakassuhteiden hallintaa. Viitataan usein järjestelmään, jolla ratkaistaan asiakkuudenhallinta.

DAE: *Dynamo Template Builder*. Dynamo-rakennemallien luomiseen tarkoitettu web-sovellus.

DAP: *Dynamo Application template*. Dynamo-rakennemalli, joka kokoaa sekä dynaamisesti rakennettavan dokumentin että logiikan sen ympärillä.

DOM: *Document Object Model.* Tapa kuvata esimerkiksi HTML:n rakenne puuna, jolloin sen käsittely on mahdollista.

GUI: *Graphical User Interface.* Graafinen käyttöliittymä sovelluksessa, mikä mahdollistaa käyttäjän interaktion visuaalisten komponenttien avulla.

HMR: *Hot Module Replacement*. Sovelluksen eri moduulien päivittämisen mahdollistava ominaisuus ajon aikana siten, ettei koko sivua tarvitse päivittää.

HTML: *HyperText Markup Language.* Standardi web-selaimille tarkoitetusta dokumentin rakenteesta.

HTTP: *Hypertext Transfer Protocol.* Protokolla, jota käytetään selainten ja web-palvelimien tiedonsiirtoon.

ISV: *Independent Software Vendor.* Itsenäinen ja riippumaton ohjelmistotuottaja.

JSON: *Javascript Object Notation.* Formaatti välittää tietoa ymmärrettävässä muodossa avain-arvo-pareina.

REST: *Representational State Transfer.* Arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen.

SPA: *Single-page application.* Yksisivuinen web-sivujen tai sovellusten arkkitehtuurimalli, jossa sisältö on yhtenä dokumenttina.

SRS: *Software requirements specification.* Dokumentti, jossa kuvataan ja määritellään ohjelmistoprojektin vaatimukset ja tavoitteet.

1 Johdanto

Ohjelmistokehyksiä modernin web-sovelluksen kehittämiseen on nykypäivänä useita. Vaihtoehtojen runsaus ja niiden erilainen toimintaperiaate sekä mahdollistavat että rajoittavat joidenkin ominaisuuksien implementoimisen sovellukseen. Jotta tarkoituksenmukainen ohjelmistokehys otetaan käyttöön, on hyvä tietää sovelluksen asettamat vaatimukset ennalta.

Insinöörityön toimeksiantaja on suomalainen yritys nimeltään Documill Oy. Ohjelmistoyritys on edelläkävijä sähköisten sopimusasiakirjojen ja dokumenttien dynaamisessa generoimisessa. Dokumenttien luomisen perustana on automaatio, joka tehostaa niiden käsittelyä ja lisää niiden sisällön täsmällisyyttä vähentäen inhimillisiä virheitä, joita voisi tapahtua, mikäli asiakirja luotaisiin manuaalisesti pala palalta. Nopeuttamalla sähköisten sopimusasiakirjojen luontia ja käsittelyä, tehostaa se suoraan yritysten tai organisaatioiden välistä kaupankäyntiä ja selkeyttää sopimusprosessia. Tällöin myös osapuolien välinen luottamus toisiinsa paranee.

Työn aiheena on toimeksiantajan Dynamo-nimisen tuotteen käyttöliittymäpuolen ajamiseen vaadittavan sovelluksen ohjelmistokehyksen uusiminen. Vaikka merkittäviä arkkitehtuurisia muutoksia osana uudistusprosessia tapahtui myös back-endiin, käsittelee insinöörityö pitkälti front-puolta aiheen rajaamiseksi. Nykyisin käytössä olevan ohjelmistokehyksen jatkokehitys ja tuki on päättynyt. Tämä luo suoraan turvallisuusriskin väärinkäytökselle. Tavoitteena on luoda taaksepäinyhteensopiva, modernilla ohjelmistokehyksellä toimiva sovellus, joka kykenee suoriutumaan ydintoiminnoistaan.

Aluksi insinöörityössä taustoitetaan Documillia sekä sen Dynamo-tuotekokonaisuutta. Tämän jälkeen käsitellään motivaatiot ja lähtökohdat työn perustalle. Perustavanlaatuisena kysymyksenä tällöin on se, mitä hyötyä kehyksen uudistaminen tuo, mikäli nykyinen toimii jo ennestään. Taustoituksen jälkeen esitellään työssä käytettyjä teknologioita. Uudeksi ohjelmistokehykseksi valikoitui Vue.js, jota esitellään ja jonka valintaa perustellaan. Tässä luvussa

myös perustellaan sen käyttöä ja vertaillaan sitä kahteen muuhun suosittuun ohjelmistokehykseen, Angulariin ja Reactiin. Itse kehitysprosessiin päästään seuraavassa luvussa. Siinä käsitellän sovelluksen dokumentaatiota, sisäistä logiikkaa aina aloituksesta loppuun esitellen toiminnallisuuksia. Ennen yhteenvetoa käydään läpi saavutettuja tuloksia samalla pohtien mahdollisia jatkokehityskohtia.

Vaikka työn pääpaino on front-endissä, käydään siinä läpi myös merkityksellisiä osatekijöitä rajapinnassa ja back-endissä tapahtuvista asioista. Osa työn yksityiskohdista näissä osatekijöissä jätetään tarkoituksella raportoimatta yrityssalaisuuksien vuoksi.

2 Taustaa

2.1 Dynamo

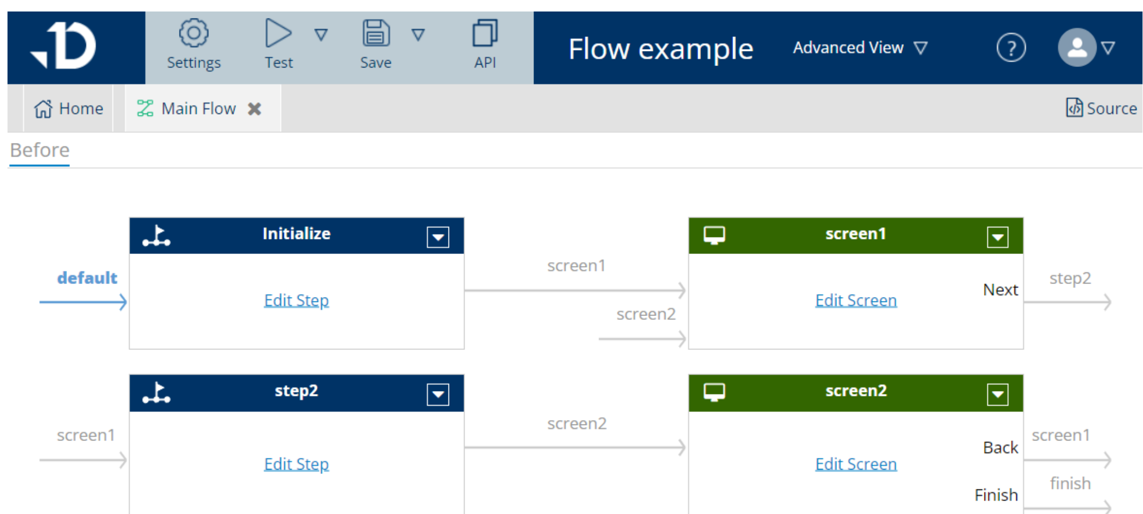
Pohjoismaiden suurimman Salesforce-painotteisen itsenäisen ohjelmistotuottajan (Independent Software Vendor, ISV) Documillin tunnetuin tuote on Dynamo [1]. Se on pohjimmiltaan automatisoitu dynaaminen dokumenttigeneraattori, joka mahdollistaa muun muassa sähköisten asiakirjojen luonnin.

Dynaamisuudella viitataan tapaan tuoda luotavaan dokumenttiin sisältöä, joka muuttuu käyttötapauksesta riippuen. Usein etenkin suuremmissa organisaatioissa on olemassa jo valmiita pohjia asiakirjoille tai lomakkeille, joissa osa tiedoista pysyy muuttumattomana. Usein myös kyseisissä asiakirjoissa on jätetty tyhjäksi tietyt kohdat muuttuvalle sisällölle käyttäjän täytettäväksi. Dynamo automatisoi näiden tyhjien kenttien täyttämisen hyödyntämällä sekä palveluntarjoajan että käyttäjän käyttämää asiakkuudenhallintajärjestelmää, Salesforcea.

Asiakassuhdejärjestelmät eli CRM-järjestelmät (Customer Relationship Management) mahdollistavat sekä nykyisten asiakkaiden tai kontaktien että

potentiaalisten uusien asiakkuuksien välisen yhteydenpidon ja asiakastuen [2]. Dynamo on integroitu Salesforceen ja on saatavilla Salesforcen AppExchange -markkinapaikasta, mikä tarkoittaa sitä, ettei käyttäjän tarvitse poistua CRM-järjestelmästä käyttääkseen tuotetta [3]. Tutun CRM-järjestelmän ja siihen integroituneen Dynamo-tuotteen yhteiskäyttö tehostaa yritysten ajankäyttöä sekä parantaa tulosten laatua [2].

Logiikka, jonka perusteella jonkin asiakirjan kentät täytetään, tapahtuu Dynamo Template Builder (DAE) taustapalvelussa (kuva 1). Dynamo mahdollistaa myös monimutkaisemman sisäisen logiikan rakentamisen pelkän dokumentin tuottamisen sijaan. Esimerkiksi voidaan tarvittaessa haluta eriyttää polut, joiden kautta dokumenttia tuotetaan, tai voidaan tarvittaessa jopa estää sen tuottaminen. Tätä jatkumoa kutsutaan flow-prosessiksi. DAE:n avulla siis rakennetaan sekä malli dokumentille että logiikka sen generoimiseen. Tämä kokonaisuus voidaan paketoida yhteen rakennemalliin, DAP:iin (Dynamo Application template).



Kuva 1. DAE:n avulla määritetty rakennemallilogiikka.

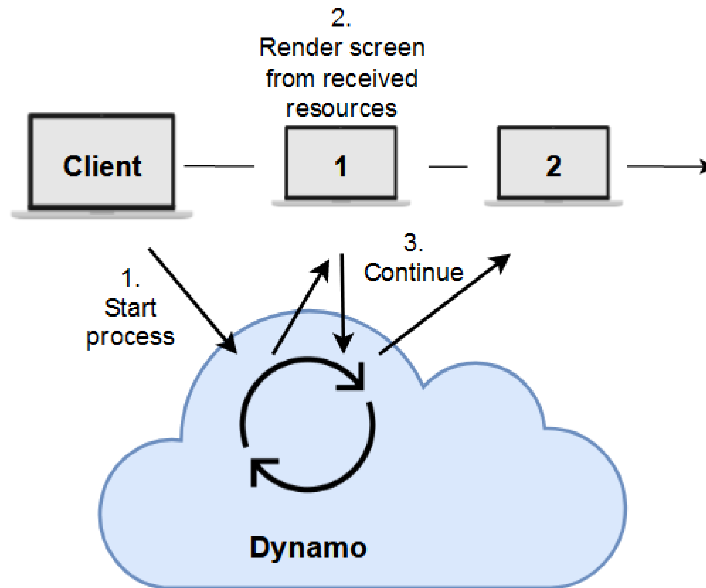
Dynamoa voidaan käyttää sekä API-käytössä että suoraan graafisessa käyttöliittymässä selaimen kautta. Graafisen käyttöliittymän mahdollistava sovellus onkin insinöörityön aiheena.

2.1.1 Dynamo API-käytössä

API:lla (Application Programming Interface) tarkoitetaan eri mekanismeja, jotka mahdollistavat kahden tietokoneohjelmistokomponentin kommunikoinnin keskenään tiettyjen ennalta laadittujen määrittelyjen ja protokollien avustuksella [4]. Esimerkkitapaus Dynamon hyödyntämisestä API-käytössä voisi olla tilanne, jossa halutaan aina automaattisesti luoda viikon päätteeksi yrityksen sisäinen raportti. Raportti voisi sisältää ennalta määrätyt asiakohdat sekä niiden tulokset dynaamisena.

2.1.2 Dynamo GUI-käytössä

Dynamoa voidaan ajaa myös graafisessa käyttöliittymässä. Graafinen käyttö, tai GUI-käyttö (Graphical User Interface), mahdollistaa käyttäjän interaktion ohjelmistojen tai laitteiden kanssa erinäköisten visuaalisten komponenttien avustuksella [5]. DAP:in logiikkaan voidaan sisällyttää graafisia elementtejä (myöhemmin Screen-tila), jotka mahdollistavat tarkoituksenmukaisen GUI-käytön. Kuvassa 2 esitellään Dynamon GUI-käytön logiikkaa. Prosessin aloituksen jälkeen asiakasohjelmalle saapuu esiteltävä ruutu, jonka kanssa käyttäjä voi toimia. Esiteltäviä ruutuja voi olla useampia, ja niissä navigoiminen onnistuu nappeja painamalla. Tyypillisessä esimerkissä käyttäjä voi ensin lukea eteen saapuvan sopimusasiakirjan ja tämän jälkeen joko hyväksyä ja allekirjoittaa sopimuksen tai hylätä sen. Uuden ohjelmistokehyksen tavoitteena onkin onnistuneesti käsitellä ja esitellä kyseenomaiset Screen-tilat.



Kuva 2. Dynamon toiminta GUI-käytössä.

2.2 Miksi uusi ohjelmistokehys?

Syitä ohjelmistokehyksen vaihtamiselle on useita. Merkityksellisin syy vaihdolle on nykyisin käytössä olevan kehyksen AngularJS:n jatkokehityksen ja aktiivisen ylläpidon loppuminen [6 ; 7]. Esimerkiksi vuonna 2022 haavoittuvuuksia havaittiin jo kaksi, ja niitä oletetaan löydettävän lisää etenkin tuen loppuessa [8]. Hyödyt ohjelmistokehyksen päivittämisestä kohdistuvat loppukäyttäjälle ja yritykselle itselleen.

2.2.1 Hyöty yritykselle

Hyödyt kehyksen päivittämisestä yritykselle itselleen ovat moninaiset. Eräänä keskeisenä hyötynä on koodin modularisointi. Sen tarkoituksena on pilkkoa koodi omiin funktioihinsa eli niin sanottuihin rakennuspalikoihin, joilla jokaisella on oma tarkoituksensa suuremmassa kokonaisuudessa. Modularisoinnin voidaan ajatella tapahtuvan useammalla tasolla; jakamalla funktiot tiedostoihin, tiedostot kansioihin ja kansiot projektiin. Tämä mahdollistaa kansioiden ja

tiedostojen nimikonventioiden hyödyntämisen ja yksinkertaistaa lähdekoodin selaamista. Modularisointi helpottaa testaamista, sillä yksikkötestien kirjoittaminen on modulaarisessa ympäristössä yksinkertaisempaa. Jos virhetilanteita tapahtuu, niiden jäljittäminen ja korjaus on nopeampaa. [9 ; 10.]

Dynamo-tuotekokonaisuuteen tuodaan jatkuvasti lisää uusia korkealaatuisia ominaisuuksia. Jatkokehitystä siis tapahtuu jatkuvasti, minkä vuoksi koodin on oltava luettavuudeltaan ymmärrettävää. Modularisointi edesauttaa luettavuutta, joka on aivan keskeinen osa silloin, kun useammat ohjelmistokehittäjät jatkokehittävät tai refaktoroivat samaa koodipohjaa. [11.]

Modularisoinnin ohella, kehyksen päivityksen yhteydessä, päästään eroon myös ylimääräisistä sekä osittain vanhentuneista kirjastoista ja kooditiedostoista, joita ei tulla tarvitsemaan enää. Lähdekoodin ylläpidon helpottaminen ketterässä ohjelmistokehityksessä tulee tehostamaan refaktorointimahdollisuuksia sekä vähentämään tarvetta kirjastojen päivittämiselle. [12.]

2.2.2 Hyöty loppukäyttäjälle

Asiakkaan eli loppukäyttäjän näkökulmasta yhtenä tavoitteena on parantaa suorituskykyä ja täten käyttökokemusta. Toisin sanoen käyttöliittymä tulee olemaan sulavampi kuin nykyisin käytössä oleva AngularJS [13]. Tämän tulevat mahdollistamaan sekä uusi kehys että arkkitehtuuriset muutokset palvelinpuolella. Kovassa kuormituksessa AngularJS:n suorituskyvyn tiedetään heikkenevän merkittävästi [14]. Uudessa toteutuksessa suorituskyvyn heikkenemistä pyritään välttämään.

Selainten kehittyessä on mahdollista, että osa AngularJS:n komponenteista lakkaa toimimasta, mikä vaikuttaa suoraan käyttökokemukseen [15]. Suurin osa internet-liikenteestä tapahtuu mobiililaitteilla, joten web-sovellusten oletetaan nykypäivänä täyttävän responsiivisuushaasteet [16]. Dynamon GUI-käytön oletetaan suoriutuvan yhtä lailla myös mobiilinäkymässä.

3 Kehitysteknologioiden esittelyä

3.1 Vue.js

Kuten jo aiemmin on mainittu, uudeksi ohjelmistokehykseksi valikoitui Vue.js, tarkemmin versio 3. Avoimen lähdekoodin ohjelmistokehystä voidaan käyttää useisiin käyttötarkoituksiin [17 ; 18]. Yksi käyttötarkoituksista on tämänkin insinöörityön kannalta oleellinen Single Page Application (SPA) eli yksisivuinen sovellus. SPA:n ideana on ladata vain yksi web-dokumentti palvelimelta, minkä jälkeen vain dokumentin sisältöä muutetaan dynaamisesti joko vaihtamalla näkymä toiseen reitityksellä tai hakemalla vain uusi sisältö palvelimelta [19]. Yhä suositumman SPA-arkkitehtuurin ansiosta web-sivujen käyttö nopeutuu, yksinkertaistuu sekä tehostuu [20].

3.1.1 Vertailua muihin kehyksiin

Angular eroaa rakenteellisesti eniten Vuesta jakamalla tyyli-, skripti- ja hypertekstin erillisiin tiedostoihin. Reactissa ja Vuessa nämä voivat olla samassa tiedostossa. Angular käyttää natiivisti TypeScriptiä ohjelmointikielenä, kun taas React ja Vue JavaScriptiä. Typescript auttaa kehittäjiä muun muassa löytämään virheitä koodista jo ennen sen ajamista. Angularia usein käytetäänkin suuriin projekteihin, joissa kehittäjiä on useita ja joissa pyritään tarkkoihin säännönmukaisuuksiin. Vaikka Angular on monimutkaisempi ohjelmistokehys verrattuna Vuen yksinkertaisuuteen, Angularin suuri yhteisö ja dokumentaatio auttavat kehittäjiä ratkaisemaan monimutkaisiakin ongelmia. Se on myös hyvin laajennettavissa ja tarjoaa laajan valikoiman moduuleja ja kirjastoja. Oppimiskäyrä Angularissa on kuitenkin huomattavasti korkeampi vertailtaessa Vueen tai Reactiin. [21.]

React ja Vue käyttävät niin kutsuttua virtual DOM:ia (Document Object Model), mutta Angular ei [22]. DOM tarkoittaa tapaa kuvata web-dokumenttien, esimerkiksi HTML:n rakenne puuna, mikä mahdollistaa sen rakenteen ja tyylien ohjelmallisen muokkaamisen [23]. Virtual DOM on ohjelmointikonsepti, jossa

virtuaalinen representaatio käyttöliittymästä pidetään muistissa ja synkronoidaan oikean DOMin kanssa [24]. Tämä mahdollistaa sen, ettei koko DOM:ia tarvitse rakentaa alusta loppuun, mikäli muutoksia tulee vain yhteen osaan DOMista, vaan vain kyseinen osa renderöidään [25]. Tämän johdosta muistinhallinta sekä nopeus ovat merkittävästi parempia.

Web-kehityksessä käytettävien ohjelmistokehysten suosio on jatkuvassa muutoksessa, ja vaikka React on ohjelmistokehyksistä tällä hetkellä suosituin, nostaa Vue päätään vuosi vuodelta yhä enemmän [26]. Vuen kasvava suosio johtuu sen monista eduista verrattuna muihin ohjelmistokehyksiin. Sen toiminnot, kuten sen käynnistymisaika, muistinhallinta sekä loogiset operaatiot kuten rivien hallinnointi, tulevat olemaan nopeammat ja tehokkaammat vertailtaessa muihin suosittuihin moderneihin ohjelmistokehyksiin, kuten Reactiin tai Angulariin [13]. Nämä edut tekevät Vuesta houkuttelevan vaihtoehdon kehittäjille, jotka haluavat tehokkaan ja helppokäyttöisen ohjelmistokehyksen, joka auttaa heitä kehittämään moderneja verkkosovelluksia nopeasti ja tehokkaasti.

3.1.2 Syitä valinnalle

Merkittävin syy Vuen valinnalle liittyy eräisiin Dynamon sisäisiin Bound-komentoihin. Niiden avulla käyttäjän on mahdollista kontrolloida saapuvaa dataa. Kyseiset komennot vaativat sen, että ohjelmistokehys olisi kykenevä dynaamisesti käsittelemään saapuvan HTML-sisällön DOM-muotoon niin, että sen sisältävät komponentit ja direktiivit käsiteltäisiin myös (ns. template compiler). Esimerkiksi sisällön mukana voi saapua eräänlainen alkiokokoelma, jota täytyy pystyä dynaamisesti muokkaamaan lisäämällä, poistamalla tai uudelleenjärjestelemällä kyseinen kokoelma. Koska sisältö saapuu dynaamisesti renderöitäväksi interaktiiviseen muotoon, ei sen esikääntäminen ole toivottua.

Tarkoituksena oli hakea myös kevyt, modularisoitava ja moderni ohjelmistokehys, joka kykenee suoriutumaan taaksepäin

yhteensopivuushaasteista. Suuren kehittäjäyhteisön, helpon opittavuuden sekä kattavan dokumentaation ansiosta Vue loi puitteet mahdolliselle valinnalle. Näiden lisäksi useamman ohjelmointikonvention sekä progressiivisen lähestymistavan vuoksi syyt Vuen valinnalle olivat ilmeiset.

3.1.3 Versiot

Yksi Vuen vahvuuksista on myös tapa, kuinka komponetteja voidaan määritellä. Vue-komponentit voidaan määrittää käyttämään kahta erilaista API-tyyliä kehitysvaiheessa [27]. Kumpaakin tyyliä voidaan käyttää samanaikaisesti, vaikkei tätä suositella [28]. API-tyylit ovat

- Options API
- Composition API.

Options API -tyylissä komponenttien ominaisuuksien logiikka määritellään luokkapohjaisiksi. Esimerkkikoodissa 1 näitä ominaisuuksia ovat data, methods sekä mounted. Avainsanalla ’this’ on mahdollista kohdistaa nimenomainen kutsu ominaisuuksiin kyseisen komponentin instanssissa. Options API on tyylillisesti selkeämpi käyttäjälle, jolle oliopohjainen ohjelmointi on tuttua. Rakenteellisesti sen voidaan sanoa johdattelevan organisoituun koodiin ryhmitellessä eri ominaisuudet. Kuitenkin komponentin koon ja logiikan kasvaessa sen kompleksisuus moninkertaistuu ja hallittavuus huononee. Tämän lisäksi loogisten operaatioiden eriyttäminen ja uudelleenkäyttö on perin hankalaa. [27 ; 28.]

```
<script>
export default {
  data() {
    return {
      count: 0
    }
  },
  methods: {
    increment() {
      this.count++
    }
  },
  mounted() {
    this.increment()
  }
}
</script>
```

Esimerkkikoodi 1. Options API -esimerkki.

Composition API on tarkemmin määriteltynä joukko erilaisia API-määrittelyjä, jotka kattavat komponenttien reaktiivisuuden, elinkaaren sekä riippuvuuden. Yksisivuisen komponentin logiikka määritellään tuotujen API-funktioiden avulla tyypillisesti script setup -määreen kanssa (esimerkkikoodi 2). Composition API on rakenteellisesti vapaamuotoisempi, mutta vaatii ymmärrystä reaktiivisuudesta. Joustavuus komponenttien määrittelyssä mahdollistaa koodin organisoinnin sekä logiikan uudelleenkäytön Options API:a tehokkaammin. [27 ; 28.]

```
<script setup>
import { ref, onMounted } from 'vue'

const count = ref(0)

function increment() {
  count.value++
}

onMounted(() => {
  increment()
})

</script>
```

Esimerkkikoodi 2. Composition API -esimerkki.

Työssä käytetään tyylinä Composition API:a, koska se suoriutuu vahvasti tyypitetystä ohjelmointikielestä, TypeScriptistä, tehokkaammin. Options API:a

suunnitellessa ei tyypitystä ajateltu, minkä johdosta sen implementoiminen myöhemmässä vaiheessa oli hankalaa. Tämän vuoksi on mahdollista, että Options API voi hajota tyypittäessä kieltä. [28.]

3.2 Vite

Vite on frontend -kehitykseen suunniteltu työkalu sovelluksen kokoamiseen sekä toimintakuntoon saattamiseen. Se tarjoaa nopeamman ja ketterämmän kehityskokemuksen modernille web-projektille. Viteä hyödynnetään sovelluksen kehityksessä, koska sillä on saumaton integroitumiskyky Vuen kanssa. Se myös mahdollistaa nykypäivän kehitysperiaatteiden sekä -vaatimusten täyttämisen ominaisuuksillaan. [29.]

Eräs näistä ominaisuuksista on monipuolinen kehityspalvelinkokonaisuus, josta tyyppiesimerkki on ns. Hot Module Replacement (HMR). HMR mahdollistaa koodimuutosten näkymisen kehittäjälle ilman, että näkymää täytyy erikseen päivittää. Tämän johdosta sovelluksen tila sekä muuttujat pysyvät ennallaan. Kehittäjän ei esimerkiksi tarvitse erikseen navigoida sovelluksessa kehityspisteeseen päivittäessään lähdekoodia. [30.]

Tämän käyttöliittymäsovelluksen ominaisuuksien sekä vaatimusten vuoksi Vite on oiva työkalu kehitykseen. Koska esitettävä data saapuu palvelimelta dynaamisesti, on huomattavasti ketterämpää kehittää sovellusta, kun sen tila pysyy ennallaan eivätkä sisäisten muuttujien arvot nollaudu. Tämän lisäksi kehityspalvelimen käynnistämiseen eikä sovelluksen rakennusvaiheeseen kulu tarpeetonta aikaa.

3.3 Vitest

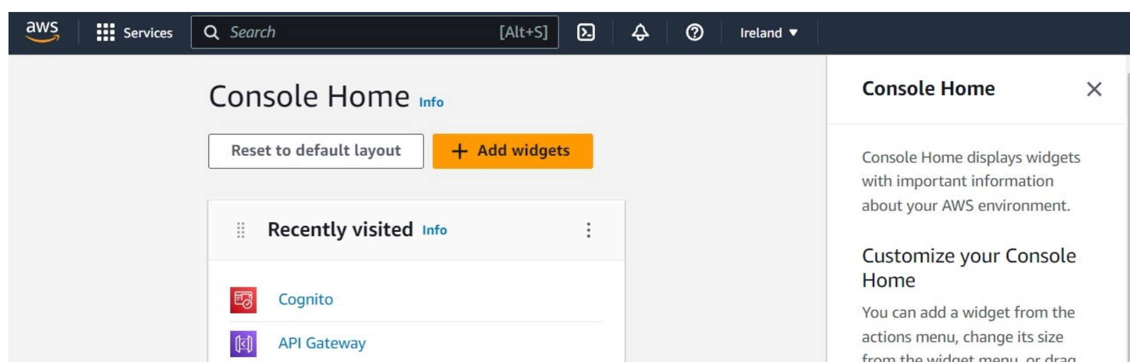
Testaustyökaluna käytetään Vitest-nimistä työkalua. Kuten Vite, Vitest toimii Vuen kanssa yhteen vaivattomasti. Pääsyynä näiden työkalujen saumattomaan integraatioon Vuen kanssa on se, että samat kehittäjät ovat näiden takana [31].

Vitest on rakennettu Vite-työkalun päälle mahdollistaen esimerkiksi useamman samanaikaisen yksikkötestin ajamisen [32].

3.4 Amazon Web Services

Amazon Web Services (AWS) on pilvipalvelualusta, jonka omistaa monikansallinen yhtiö Amazon. Pilvialustana AWS oli suurin ja täten suosituin toimija vuonna 2022 [33]. AWS tarjoaa käyttäjälleen useita korkealaatuisia laskemiseen, tallennukseen, sovelluksen pyörittämiseen sekä moneen muuhun samankaltaiseen tarkoitettuja pilvipalveluita. AWS:n palveluiden hyödyntäminen insinöörityössä oli itsestäänselvää, sillä osa Dynamosta nojaa jo ennalta AWS:n palveluihin.

Eri AWS:n tarjoamien palveluiden selaaminen on helppoa AWS-konsolin kautta (kuva 3). Se on AWS:n tarjoamiin palveluihin suunniteltu web-käyttöliittymä. Konsolin kautta onnistutaan suoraan määrittelemään ja hallinnoimaan pilvipalveluratkaisuja. Nämä konfiguraatiot voidaan myös määritellä itse projektitasolla koodin välityksellä. AWS:n palveluista ja tekniikoista oleellisimmat insinöörityön kannalta ovat AWS CDK, Lambda-funktiot ja API Gateway.

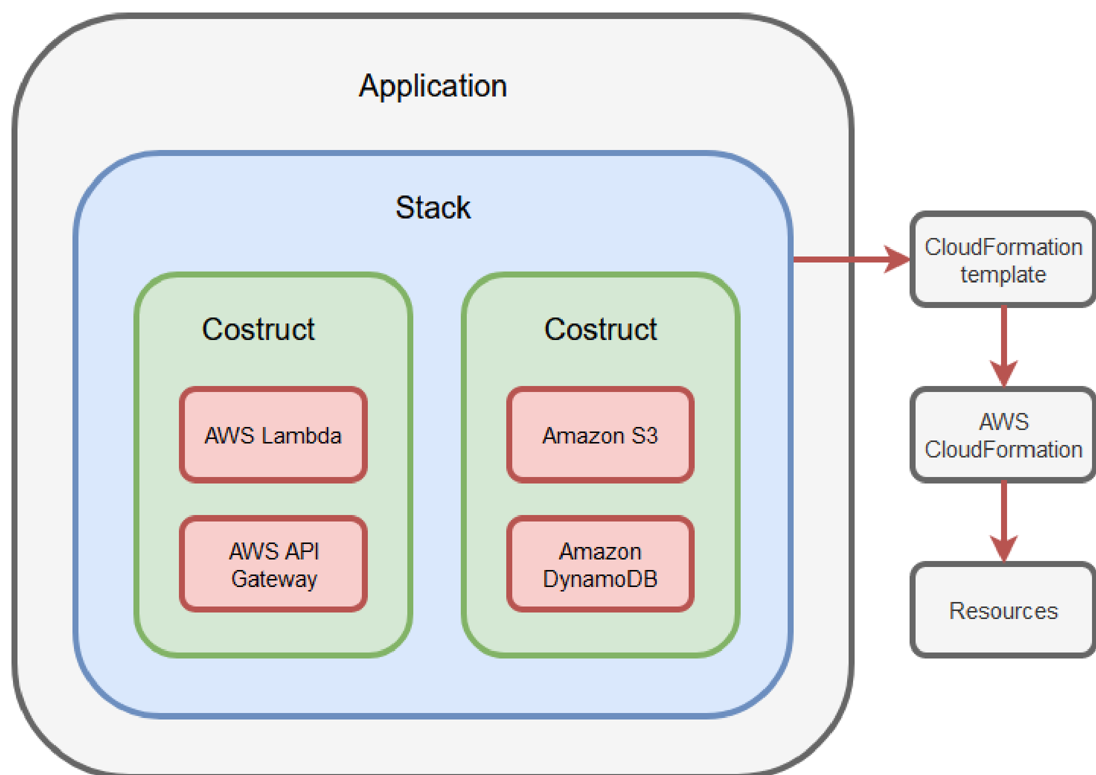


Kuva 3. AWS-konsolin etusivu.

3.4.1 AWS Cloud Development Kit

AWS Cloud Development Kit (AWS CDK) on avoimen lähdekoodin pilviarkkitehtuuriin keskittynyt ohjelmistokehys [34]. Sen avulla kehittäjän on mahdollista keskittää resurssointi sekä rakentaa luotettava ja skaalautuva pilvisovellus tutuilla ohjelmointikielillä tehokkaasti [35]. Tehokkuudella viittaan kirjoitetun koodin määrään; se mitä aikaisemmin kirjoitettiin useampi sata riviä määrittelyjä pilvisovellusinfrastruktuurille AWS CloudFormation mallipohjassa, tapahtuu CDK:n avulla muutamassa kymmenessä rivissä [35]. AWS CloudFormation -palvelu auttaa mallintamaan ja määrittämään AWS-resurssit [36].

CDK käsittää rakenteita (Construct), jotka esittävät sovelluksen käyttämiä AWS-resursseja. Esimerkkinä tämänkaltaisesta rakenteesta ovat Lambda-funktiot, joihin sovelluksen ulkoiset pyynnöt kohdistuvat. Rakenteet muodostavat pinon (Stack), joka taas muodostaa itse sovelluskokonaisuuden (kuva 4).



Kuva 4. AWS CDK:n toimintaperiaate korkealla tasolla.

Valmis pilvisovellusmäärittely onnistutaan ottamaan käyttöön yhdellä komennolla (esimerkkikoodi 5) CDK:n tarjoamalla komentopohjaisella käyttöliittymällä, CLI-työkalulla (Command Line Interface). Komento tuottaa sovelluksen infrastruktuurin määrittävän CloudFormation-mallipohjan. Mallipohja viedään tämän jälkeen AWS CloudFormation-palveluun, jonka kautta sovelluksen valmiit resurssit saadaan käyttöön.

3.4.2 AWS Lambda

AWS Lambda tarjoaa palvelittoman tavan ajaa koodia tapahtumaperustaisesti. Lambda-funktiot mahdollistavat siis funktion ajamisen ilman, että sitä tarvitsee erikseen ylläpitää palvelimella. Merkittävinä hyötyinä on mm. skaalautuvuus, ylläpidettävyys ja kustannussäästöt. [37.]

AWS skaalaa automaattisesti Lambda-funktion toteutusympäristöjen määrää, mikäli pyyntöjä saapuu useita samanaikaisesti [38]. Lambda-funktioiden laskutusperuste on sen ajoaika. Kustannus per aikayksikkö määräytyy funktiolle kohdennetusta muistin määrästä [39]. Yksinkertaisuudessaan Lambda-funktion voidaan määritellä ottavan tapahtuman yhteydessä parametrin vastaan ja palauttavan sitten funktion kutsujalle vastauksen esimerkkikoodi 3:n mukaisesti.

```
exports.handler = async (event, context) => {
  const message = "Hello, ${event.name}!";
  const response = {
     statusCode: 200,
     headers: {
       "Content-Type": "application/json"
     },
     body: JSON.stringify({ message })
  }
  return response;
};
```

Esimerkkikoodi 3. Lambda-funktio, joka ottaa vastaan parametrin ja palauttaa vastauksen funktion kutsujalle.

Työssä Lambda-funktioita käytetään silloin, kun sovellus ottaa yhteyden ulkoisiin API-rajapintoihin pyytäessään eri näköistä dataa. Tällöin Lambda-

funktio aktivoituu ja suorittaa sille määrätyn toiminnon. Tästä on lisää luvuissa 4.3.1 sekä 4.3.2.

3.4.3 Amazon API Gateway

API Gateway on AWS:n tarjoama palvelu, joka mahdollistaa erinäisten API-palveluiden kehittämisen, ylläpidon sekä suojaamisen. Samanaikaisten kutsujen käsittely ja niiden ohjaaminen oikeaan osoitteeseen onnistuu palvelun avulla tehokkaasti. Esimerkkikoodissa 4 määritellään CDK:n avulla API Gatewayn kanssa eräs työssä käytetty API-rajapinta. Siinä annetaan rajapinnalle reitti, hyväksyttävät pyyntötyypit, integraatio eli mihin pyyntö ohjataan sekä valtuutus. [40.]

```
httpApi.addRoutes({
  path: "/composer/start",
  methods: [apigwv2.HttpMethod.ANY],
  integration: startComposerIntegration,
  authorizer: noneAuthorizer
});
```

Esimerkkikoodi 4. API-reitin ja asetusten määrittelevä koodinpätkä CDK:n avulla.

4 Kehitysprosessi

4.1 Lähtökohdat ja vaatimusmäärittely

Projektia on dokumentoitu jo hyvissä ajoin ennen muutosprosessin alkua mm. suunnittelulla ja vaatimusmäärittelyllä (SRS). Vaatimusmäärittelyn (Software Requirements Specification) tarkoituksena on määritellä rakennettavan ohjelmiston tavoitteet ja käyttötarkoitus, määrittää ohjelmiston vaatimukset sekä kuvata mahdolliset rajoitteet [41]. Yksi konkreettinen vaatimus uudella toteutuksella on taaksepäinyhteensopivuus. Sen on kyettävä suoriutumaan samoista tehtävistä kuin nykyisin käytössä oleva toteutus.

Sisäinen dokumentaatio voidaan karkeasti jakaa kolmeen osa-alueeseen, jotka ovat

- suunnitteluvaihe
- kehitysvaihe
- käyttöönottovaihe.

Suunnitteluvaihe (design) lähtee liikkeelle uudistuksen tarkoituksesta sekä motiiveista. Se käsittää käytössä olevat sekä käyttöön tulevat komponentit sekä niiden suhteet kaavioiden avustuksella korkealla tasolla. Uudistuksen tavoitteisiin kuuluuvat mm. kevytrakenteisuus, modulaarisuus sekä taaksepäin yhteensopivuus tietyiltä osin. Sovelluksen halutaan olevan mahdollisimman kevytrakenteinen, jotta se palvelisi loppukäyttäjää parhaiten. Modulaarisuudella on merkittäviä hyötyjä kehittäjien näkökulmasta. Nämä hyödyt heijastuvat vääjäämättä myös sovelluksen käyttäjälle itselleen. Taaksepäin yhteensopivuus asettaa raamit sille, mihin sovelluksen on kyettävä.

Kehitysvaihe (development) koostaa ohjeistuksen siitä, kuinka lähdekoodiin pääsee käsiksi. Siinä kerrotaan myös, mitä täytyy tehdä ennen itse kehitystä ja miten kehitys etenee. Ennen kehitystä on asennettava tarvittavat kirjastot. Koska sovelluksessa pyritään modulaarisuuteen, on tiedostettava kehittämisen yhteydessä pakkauskomponenttien merkitys. Sisäiset rajapinnat, palvelut sekä muuttujien määritykset ovat kaikki omissa pakkauksissaan. Kehitysvaiheen dokumentaatio ohjeistaa myös sen, kuinka kehityspalvelin käynnistetään.

Käyttöönottovaihe (deployment) ohjeistaa, kuinka projekti saatetaan palvelimelle käyttövalmiiksi. Vaihe jakautuu kahteen osaan. Ensin projekti rakennetaan (build), jotta se olisi käyttökunnossa. Rakennusvaiheessa syntyy kolme staattista tiedostoa, jotka muodostavat yhdessä sovelluksen. Tämän jälkeen annetaan ohjeistus siitä, kuinka valmis paketti viedään hosting-palvelimelle. Oleellista toisessa vaiheessa on määritellä, halutaanko projekti viedä kehitys-, testi- vai tuotantopalvelimelle. Kun tämä on päätetty, käyttöönoton voi tehdä käytännössä joko manuaalisesti raahaamalla paketti

haluttuun kansioon, tai automaattisesti ajamalla komento komentoikkunassa (esimerkkikoodi 5).

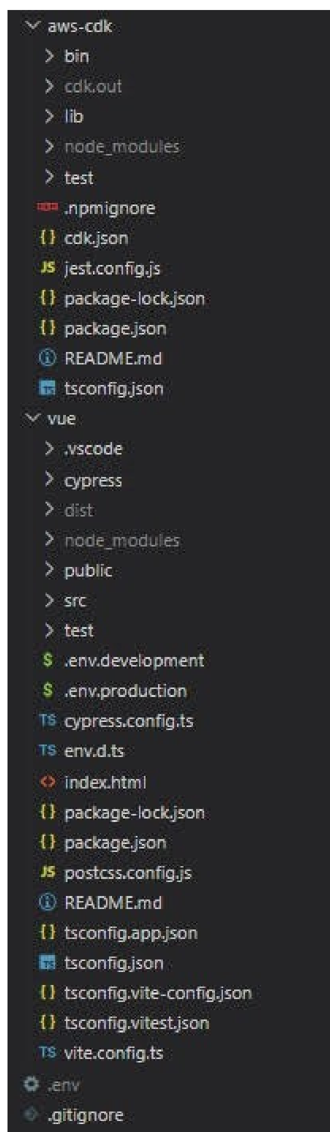
```
cdk deploy ComposerUIStack-Dev
```

Esimerkkikoodi 5. Komento paketin viemisestä hosting-palvelimelle.

4.2 Sovelluksen rakenteellinen muutos

Osana suurempaa muutosprosessia, johon liittyy vahvasti myös palvelinpuolen muutokset, tavoitteena on eriyttää Dynamo-kokonaisuuden eri komponentit toisistaan. Nykyinen toteutus on integroitu palvelimen kanssa samaan kehityskokonaisuuteen. Se on samaa rakennetta, joka luo omat haasteensa kehitysmielessä. Esimerkiksi pienen muutoksen yhteydessä on palvelinosuus ensin pysäytettävä, koodimuutos tehtävä ja projekti rakennettava toimintakuntoon, minkä jälkeen palvelin on käynnistettävä uudelleen. Tämä prosessi vie tarpeettomasti arvokasta aikaa. Uuden toteutuksen korkealla tasolla oleva rakenteellinen muutos on merkittävissä määrin sen eriyttäminen kokonaan sitä käyttävästä järjestelmästä. Lähtökohtaisesti ymmärrys kokonaisuudenhallinnasta kasvaa. Myös itse kehitysprosessi tulee tehostumaan merkittävästi etenkin, mikäli kehittäjälle taustajärjestelmä ei ole niin tuttu.

Toteutus lähti liikkeelle täysin uuden Vue-sovelluksen sekä sen rinnalla toimivan CDK-applikaation pystyttämisestä. Käyttöliittymäkomponentti ja CDK-määritykset Lambda-funktioineen muodostavat yhdessä projektikokonaisuuden (kuva 5). Uusi toteutus tuo modularisaation hyödyt aina kuormituksen tasaamisesta virhetilanteiden selvittämisen helpottamiseen. Koska sovelluksen staattinen sijainti on eriytetty aikaisemmasta toteutuksesta, on sen kehittäminen ja ylläpito merkittävästi helpompaa.



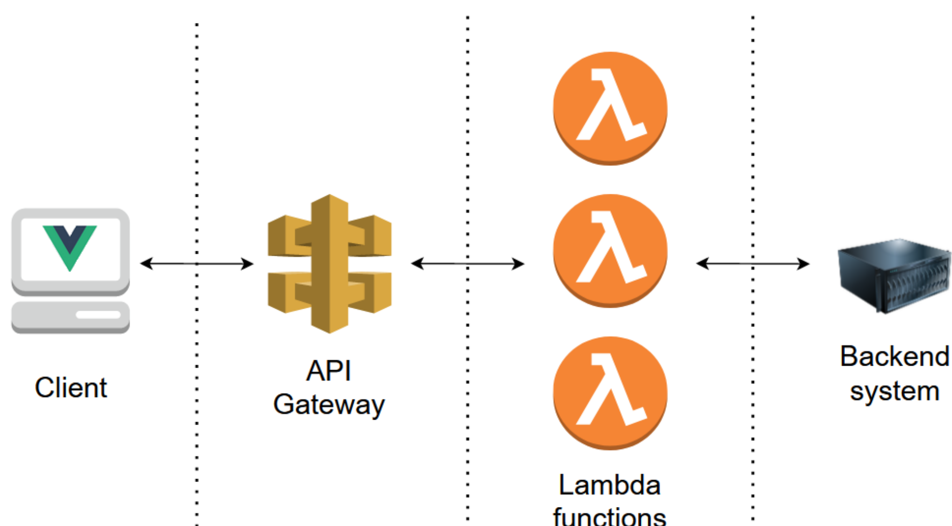
Kuva 5. Projektirakennekokonaisuus, jossa on esiteltynä Vue ja CDK.

Nykytoteutus vaatii hitaan ja raskaan operaation, mikäli uusia ominaisuuksia halutaan loppukäyttäjälle. Uuden päivitetyn version saattaminen testi- ja tuotantoversioksi nopeutuu ja helpottuu olennaisesti CDK-applikaation ansiosta. Koska uusi moderni versio sovelluksesta sijaitsee staattisena kokonaisuutena web-palvelimessa, riittää, että vain se päivitetään tarpeen vaatiessa. Tähän operaatioon riittävät vain sovelluksen rakennus sekä sen vieminen resurssiksi esimerkkikoodi 5:n mukaisesti.

4.3 Sovelluksen logiikka

4.3.1 Sovelluksen ja palvelimen kommunikaatio

Kaikki kommunikaatio sovelluksesta ulospäin tapahtuu julkisten RESTful API-rajapintojen välityksellä. REST-rajapinnalla (Representational State Transfer) viitataan ryhmään ohjelmistoarkkitehtuurisia tyylien rajoituksia, jotka saavat aikaan tehokkaita, luotettavia ja skaalautuvia hajautettuja järjestelmiä [42]. Sovelluksen ja Lambda-funktioiden välillä toimii API Gateway -palvelu, joka ohjaa sovelluksen pyynnöt oikeaan Lambda-funktioon (kuva 6).



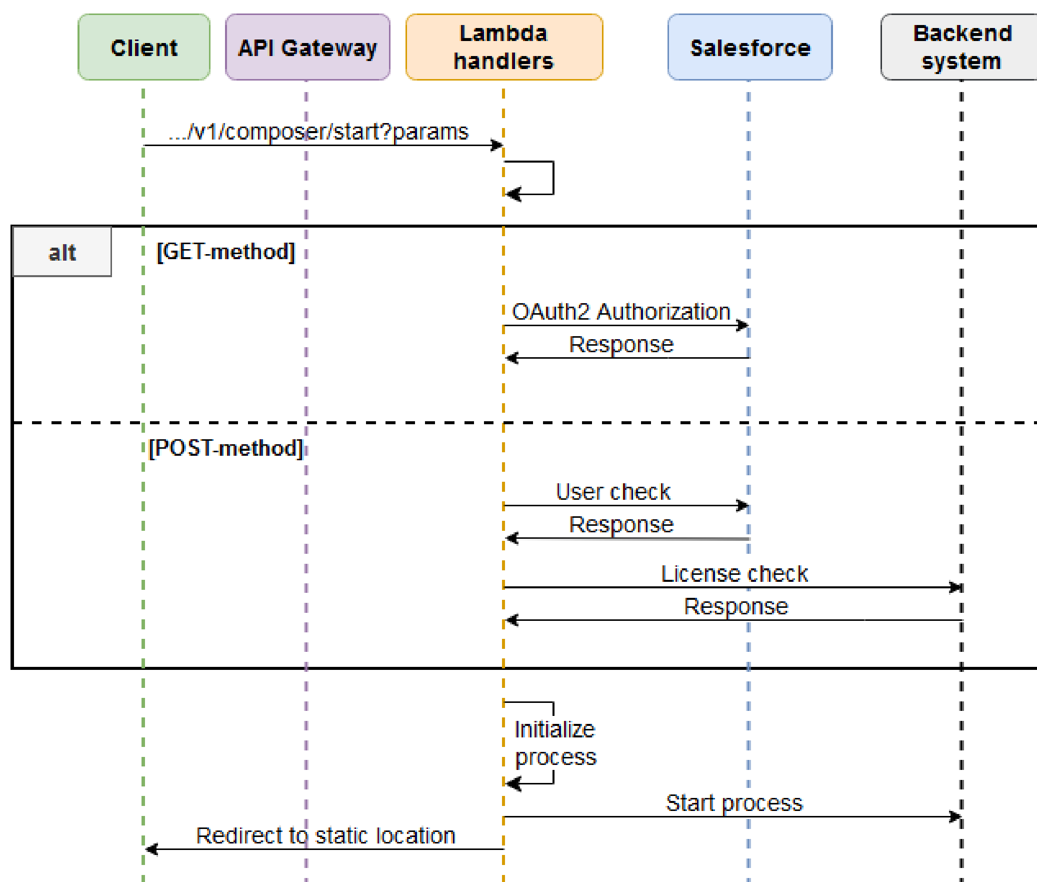
Kuva 6. Sovelluksen kommunikaatio ulospäin.

4.3.2 Sovelluksen ajon aloittaminen

Sovellus voidaan käynnistää kahta eri HTTP-pyyntöä käyttämällä. HTTP (Hypertext Transfer Protocol) mahdollistaa tiedostojen, kuten HTML-tiedostojen tai kuvien, siirron selainten ja palvelimien välillä [43]. Molemmissa pyynnöissä tehdään API-kutsu julkiseen rajapintaan. Pyyntö aktivoi Lambda-funktion, joka tekee tarvittavat toimenpiteet ajon käynnistämiseksi. Käytetyt HTTP-metodit ovat GET ja POST. Molemmat tavat ovat esiteltynä sekvenssikaavion muodossa kuvassa 7.

GET-pyynnössä suoritetaan OAuth2-valtuutusprotokollan mukainen uudelleenohjaus, jolloin varmennetaan käyttäjän oikeellisuus. OAuth2 on standardi erinäisille verkkovaltuutuksille tarjoamalla suostumuksellisen pääsyn tai rajoittamalla sovellusten toimintaa käyttäjän puolesta [44]. Pyynnön yhteydessä annetaan erinäisiä parametreja, joista yksi oleellinen on Dynamo-rakennemallin yksilöivä tunnus. Sen tarkoitus on kertoa taustajärjestelmälle, minkä rakennemallin mukainen logiikka tullaan ajamaan ja minkä siitä saatavan prosessin tilaa tullaan myöhemmin kysymään yhtäjaksoisesti.

POST-pyyntö käynnistyy saman API-rajapinnan kautta kuin GET-pyyntö. Tällöin haetaan lisätietoja käyttäjästä Salesforcen API-rajapinnan kautta käyttämällä parametrina saatua istunnon yksilöivää tunnusta. Lisätiedoilla tarkistetaan tämän jälkeen tietokannasta valtuutus ajon suorittamiseen.



Kuva 7. Yksinkertaistettu sekvenssikaavio GET- ja POST-pyyntöjen toiminnasta.

Onnistuneessa kutsussa mahdollisten lisenssien tarkistusten ja autentikaation jälkeen kutsujalle palautetaan uudelleenohjauspyyntö parametreineen toiseen URL-osoitteeseen, jossa sovellus sijaitsee staattisesti (esimerkkikoodi 6). Eräs näistä parametreista on ajon yksilöivä tunniste eli ID, jonka tilaa eli statusta sovellus aloittaa kyselemään.

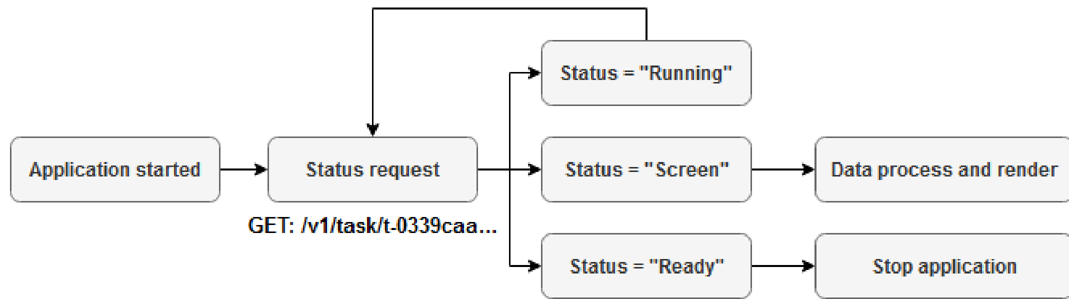
```
const response = {
     statusCode: 303,
     headers: {
          Location: "https://localhost:3000/taskID=t-
                     0339caa0f50de5efadd40c04669dd337"
     }
}
```

Esimerkkikoodi 6. Lambda-funktion palauttama uudelleenohjauspyyntö sovelluksen sijaintiin.

4.3.3 Sovelluksen sisäinen toimintalogiikka

Ensimmäisenä tavoitteena oli rakentaa sovelluksen primäärilogiikka. Tämä sisältää flow-prosessin tilan yhtäjaksoisen kyselyn heti, kun sovellus on käynnistetty onnistuneesti sekä ajon jatkamisen. Kyselyt tapahtuvat julkisten rajapintojen välityksellä GET-pyynnöillä. Tiloja voi olla useammanlaisia, ja ne kertovat ulospäin, mikä vaihe prosessissa on käynnissä. Oleellisimmat käyttöliittymään vaikuttavat tilat ovat Running-, Screen- ja Ready-tilat. Sovellus jatkaa tilan kyselyä, kunnes se on joko Screen tai Ready (kuva 8). Tieto prosessin tilasta saapuu käyttöliittymälle JSON-formaatissa (JavaScript Object Notation), joka on helppolukuinen ja yleinen tiedostomuoto tiedonvälitykseen [45].

Running-tila kertoo käyttöliittymälle, että flow-prosessi on käynnissä palvelimen puolella. Tilan kyselyä jatketaan, kunnes tila on joko Screen tai Ready. Mikään sovelluksen primäärilogiikassa ei siis muutu, mutta tiedetään kuitenkin se, että rakennemallin ajo on lähtenyt onnistuneesti käyntiin.



Kuva 8. Sovelluksen käynnistymisen jälkeinen prosessin tilan kyselyperiaate.

Mikäli tila on Screen, pyynnön yhteydessä saapuvat JSON-muotoisena sitä esittävä HTML ja CSS tekstimuotoisena sekä mahdollisesti muuta tarpeellista dataa (esimerkkikoodi 7). Tällöin saapuva data käsitellään muotoon, jossa se on mahdollista esittää käyttäjälle ja jossa käyttäjä voi tarpeen vaatiessa olla interaktiossa kyseisen sisällön kanssa. Tyypillinen esimerkki saapuvan datan käsittelystä on tilanne, jossa logiikan jatkamiseen tarkoitettuun button-elementtiin sidotaan click-attribuutti. Käyttäjän napsauttaessa kyseenomaista nappia, suoritetaan tarpeelliset datan käsittelyyn tarkoitetut toiminnallisuudet ja rakennemallin ajoa jatketaan.

```
{
	"statusMessage": "Screen",
	"screenData": {
		"html": "<div> … </div>",
		"screenCSS": ".document { … }",
		"controlledBlocks”: " … ",
		"screenData": " … "
	},
	"log": { … }
}
```

Esimerkkikoodi 7. Käyttöliittymälle palautuva data JSON-formaatissa, kun tilana on Screen.

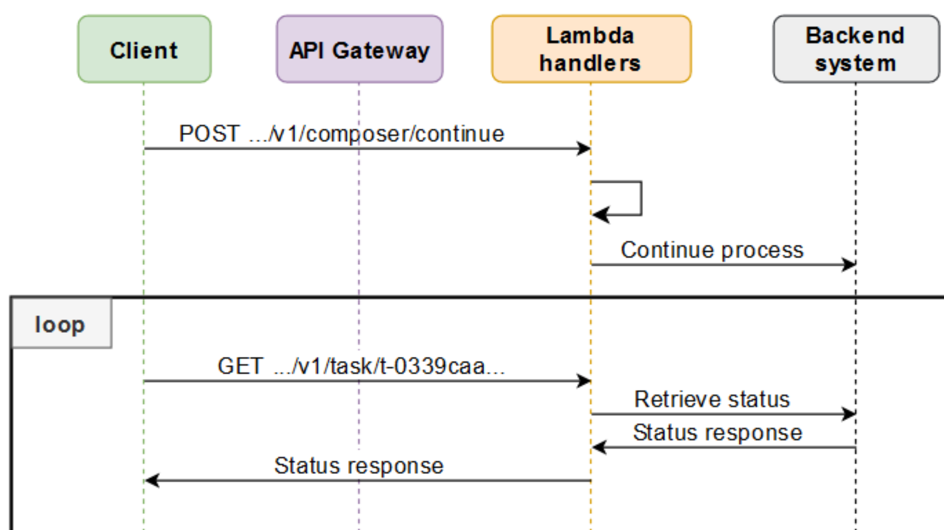
Jos tila on Ready eli valmis, tietää sovellus, että kyseinen prosessi on tietokantapuolella päättynyt, ja näin ollen myös itse päättää ajonsa. Mallin logiikkaan voidaan määritellä uudelleenohjausweb-osoite, johon sovellus ohjaa käyttäjän ajon päätyttyä.

4.3.4 Ajon jatkaminen

Käyttäjän on mahdollista jatkaa rakennemallin logiikan ajamista nappia painamalla. Sovelluksen on tehtävä tiettyjä sisäisiä operaatioita ennen yhteyttä palvelimen API-rajapintaan. Merkittävimpänä näistä on koota yhteen muuttujat, jotka halutaan viedä palvelimelle ja joita mahdollisesti tullaan käyttämään jatkossa. Voidaan ajatella tilanne, jossa ajon ensimmäisessä Screen-tilassa käyttäjä muuttaa dokumentin tiettyä kenttää halutunlaiseksi. Tämä muutos on vietävä myös palvelimelle käsiteltäväksi. Saman ajon seuraavassa Screen-tilassa dokumentin on oltava ajan tasalla muutetun kentän kanssa.

Mallin logiikka voi haarautua useaan eri polkuun. Polku voidaan asettaa ennalta esimerkiksi virheen tapahtuessa, tai siinä voidaan navigoida nappien välityksellä. Koska ajon jatkamista käsitteleviä nappeja voi olla useita, rakennettavaan JSON-pakettiin viedään tieto siitä, mitä nappia painettiin.

Kun JSON-muotoinen datapaketti on rakennettu, tehdään POST-pyyntö ajon jatkamiseen tarkoitettuun API-reittiin (kuva 9). Pyynnön mukana kuljetetaan rakennettu datapaketti ja asetetaan vaadittavat HTTP-otsakkeet. Lopulta viesti ajon jatkamisesta saapuu palvelimelle, ja käyttöliittymä aloittaa uudelleen ajon tilan kyselemisen.



Kuva 9. Sekvenssikaavio ajon jatkamisperiaatteesta.

4.3.5 Ajonaikaisia toiminnallisuuksia

Primäärilogiikan kehityksen jälkeen pyrin saamaan erinäköisiä toiminnallisuuksia liitettyä sovellukseen. Samat toiminnallisuudet esiintyvät vanhemmassa toteutuksessa, joten niiden lisääminen on edellytys taaksepäin yhteensopivuudelle.

Eräänä tärkeänä toiminnallisuutena on virheilmoitusten näyttäminen, mikäli niitä tapahtuu ajon aikana (kuva 10). Käytännössä tilan kyselyn mukana saapuu mahdollisten virheiden määrä sekä tarkemmat kuvaukset niistä. Nämä tiedot on tarkoitus näyttää käyttöliittymässä loppukäyttäjälle selkeässä muodossa. Yleensä nämä virheet liittyvät mallin rakennuksen logiikkavirheistä, eli ovat toisin sanoen käyttäjän itsensä luomia. Niissä pyritään kertomaan käyttäjälle tarkasti, missä virhe on tapahtunut. Virheet voivat kuitenkin myös liittyä loogisiin virheisiin flow-prosessissa, eli ovat kehittäjien itsensä tuottamia. Näissä tapauksissa asiakkaan toimintamahdollisuudet sen välttämiseksi ovat pienet. Notifikaatiot auttavat siis sekä asiakasta että itse kehittäjää testaamaan ja kehittämään loogisia operaatioita.

Kuva 10. Screen-tilassa esiintyvä aikakatkaisu- ja virheilmoitus.

Toinen merkittävä toiminnallisuus liittyy Screen-tilan näytön aikakatkaisuun (kuva 10). Sen avulla käyttäjien oleskelua Screen-tilassa voidaan hallita ja estää heitä viipymästä liian kauan yhdessä ruudussa. Syyt rajoitukseen ovat teknisiä ja turvallisuuspainotteisia. Kun käyttäjä viipyy liian kauan näytöllä, ilmoitus aikakatkaisusta tulee näkyviin. Tämän ilmoituksen tarkoituksena on varoittaa käyttäjää ja tarjota hänelle mahdollisuus jatkaa sovelluksen käyttöä. Jos käyttäjä ei reagoi ilmoitukseen tietyn ajan kuluessa, käyttöliittymän toiminta keskeytetään, eikä käyttäjä ei voi enää jatkaa sovelluksen käyttöä, ellei hän käynnistä prosessia uudelleen.

4.4 Testaus

Projektin testausfilosofia lähtee liikkeelle yksikkötestien suunnittelusta ennen toiminnallisuuden implementaatiota. Sovelluksen modulaarisuuden ansiosta yksikkötestit ovat helpompia luoda. Kuitenkin projektin nopeiden kehitysvaiheiden vuoksi sekä primäärilogiikan että tärkeiden ominaisuuksien osalta yksikkötestien kirjoittaminen on jäänyt vähemmälle huomiolle.

Joiltain osin testien kirjoittaminen ja niiden ajaminen projektissa on haastavaa, sillä sovelluksen palvelimelta saapuva sisältö, joka renderöidään ruudulle, on itsessään haastava testattava. Toki yksikkötesteihin olisi mahdollista antaa parametrina vakioituja muuttujia, esimerkiksi HTML:ää, mutta sen käyttäytyminen ruudulla sekä siihen sidottujen toiminnallisuuksien varmentaminen on koneellisesti haastavaa. Perustavanlaatuisena kysymyksenä voikin miettiä, kuinka paljon aikaa kannattaa käyttää haastavien yksikkötestien kirjoittamiseen.

Yksittäisten irrallisten toiminnallisuuksien testaaminen on toki mahdollista. Esimerkiksi Screen-tilan aikakatkaisuun liittyvä ominaisuus tai palvelimelta saapuvan datan oikea käsittely voidaan helpostikin varmentaa. Kun sovellukseen lisätään vaadittavia yksittäisiä ominaisuuksia taaksepäin yhteensopivuuden takia, on yksikkötestejäkin toivon mukaan mahdollista implementoida yhä enemmän. Kuvassa 11 näkyy Vitestin avulla suoritetut

testitapaukset. Kuten kuvasta näkyy, yksikkötestit voidaan jakaa eri tiedostoihin. Testitapauksien ajon jälkeen ilmoitetaan tarkalleen, mikä testeistä ei onnistunut ja missä vaiheessa.

```
$ npm test

> protocomposer@0.0.0 test
> vitest


 DEV  v0.9.4 C:/Users/tuomasr/Desktop/dynamo-composer-ui/vue

stderr | test/components/ScreenComponent.test.ts > ScreenComponent base functionality > Screen contains given static HTML
[Vue warn]: Unhandled error during execution of setup function
  at <ScreenComponent content="<div><h1>HTML for ScreenComponent</h1></div>" data= {} ref="VTU_COMPONENT" >
  at <VTUROOT>

 √ test/services/screen-variable-service.test.ts (6)
 √ test/components/NotificationDisplay.test.ts (1)
 ❯ test/components/ScreenComponent.test.ts (1)
   ❯ ScreenComponent base functionality (1)
     × Screen contains given static HTML

─────────────────────────────────────────────── Failed Tests 1 ───────────────────────────────────────────────

 FAIL  test/components/ScreenComponent.test.ts > ScreenComponent base functionality > Screen contains given static HTML
TypeError: Cannot read property 'length' of undefined
 ❯ Function.parse node_modules/happy-dom/lib/xml-parser/XMLParser.js:93:29
 ❯ DOMParser.parseFromString node_modules/happy-dom/lib/dom-parser/DOMParser.js:42:42
 ❯ Module.initializeScreenDirectives src/services/screen-initialization-service.ts:200:23
    198|    let parser = new DOMParser();
    199|    let screenDoc = parser.parseFromString(screenHTML, 'text/html');
    200|    let cBlocks = parser.parseFromString(controlledBlocks, 'text/html');
       |                         ^
    201|
    202|    initializeDirectives(screenDoc, screenData, screenDataPropPrefix, cBlocks);
 ❯ setup src/components/ScreenComponent.vue:44:37
 ❯ callWithErrorHandling node_modules/@vue/runtime-core/dist/runtime-core.cjs.js:157:22
 ❯ setupStatefulComponent node_modules/@vue/runtime-core/dist/runtime-core.cjs.js:7079:29
 ❯ setupComponent node_modules/@vue/runtime-core/dist/runtime-core.cjs.js:7034:11
 ❯ mountComponent node_modules/@vue/runtime-core/dist/runtime-core.cjs.js:5396:13
 ❯ processComponent node_modules/@vue/runtime-core/dist/runtime-core.cjs.js:5371:17
 ❯ patch node_modules/@vue/runtime-core/dist/runtime-core.cjs.js:4973:21

──────────────────────────────────────────────────────────────────────────────────────────────────[1/1]─

Test Files  1 failed | 2 passed (3)
     Tests  1 failed | 7 passed (8)
      Time  3.17s (in thread 61ms, 5192.89%)


 FAIL  Tests failed. Watching for file changes...
       press h to show help, press q to quit
```

Kuva 11. Esimerkki Vitestin avulla ajetuista yksikkötesteistä.

4.5 Tuotos

Sovelluksen primäärilogiikka saatiin toimimaan tavoitteen mukaisesti. Siihen kuuluu sovelluksen ajon aloittaminen, tilan kyseleminen, näytön esittäminen sekä ajon jatkaminen. Osa muistakin merkittävistä ominaisuuksista on käyttövalmiina. Esimerkiksi virheilmoitusten näkyminen on valmiista

ominaisuuksista sellainen, joka on merkittävässä roolissa varsinkin käyttökokemusta ajatellen.

Toistaiseksi sovellus ei ole vielä käyttökelpoinen tuotantoon menemiseksi. Siitä puuttuu vielä useita merkittäviä ominaisuuksia, jotta taaksepäin yhteensopivuus vaatimukset täyttyisivät. Tämä voi vaatia lisäksi vielä joitain muutoksia myös palvelimeen, joka koko prosessia pyörittää.

4.6 Haasteet

Haasteita työn aikana tuli useita vastaan. Päällimmäisenä ja ehkä merkittävimpänä näistä on uusiin teknologioihin tutustuminen, joka vei osaltaan oman aikansa. Dokumentaatioiden läpikäynti ja esimerkkitoteutusten tutkiminen olivat iso osa prosessia. Nämä asiat ovat kuitenkin keskiössä projektin onnistumisen kannalta. Parhaiden käytäntöjen omaksuminen myös edesauttaa tulevissa projekteissa. Esimerkiksi sovelluksen vieminen oikeilla parametreilla oikeaan sijaintiin resurssipalvelussa CDK:n avulla osoittautui yllättävän hankalaksi.

Taaksepäin yhteensopivuushaaste sekä nykyinen käytössä oleva sovellus määrittelee hyvin pitkälti uuden toteutuksen suunnan. Toisena haasteena oli ymmärtää ja läpikäydä nykyistä käytössä olevaa sovellusta. Se poikkeaa rakenteellisesti huomattavissa määrin uudemmasta. Myös syntaksin erilaisuus sekä esimerkiksi globaalien muuttujien käyttäminen luovat omat haasteensa. Nämä poikkeamat selittyvät pitkälti ohjelmistokehysten erilaisuudesta.

Aikataulutus on ollut haasteena projektin aikana ja tulee olemaan jatkossakin sen valmiiksi saattamisessa. Haasteet selittyvät sekä firman sisäisillä että ulkoisilla tekijöillä. Ulkoisista tekijöistä mainittakoon Angularin tuen ja jatkokehityksen päättyminen. Tavoitteena olisikin saada uusi sovellus käyttökuntoon mahdollisimman nopeasti.

4.7 Jatkokehitys

Koska ensisijainen toimintalogiikka toimii tällä hetkellä hyvin, on siihen eri ominaisuuksien tuominen ja niiden testaaminen suhteellisen suoraviivaista. Sovellusta tullaan kehittämään valmiiksi versioksi asti asiakkaan käytettäväksi. Valmis versio täyttää samat ominaisuudet kuin nykyinenkin käytössä oleva versio. Jotta tähän tilanteeseen päästään, on sovellukseen implementoitava etenkin kriittisimmät palaset. Yksi näistä palasista on itse dokumentin näyttäminen PDF-muotoisena oikein tyyliteltynä.

Testaukseen tullaan kiinnittämään jatkossa huomiota enemmän. Yksikkötestejen kirjoittaminen sekä niiden säännöllinen ajaminen mahdollisuuksien ja ajankäytön vuoksi voivat kuitenkin luoda omat haasteensa. Voi olla, että nykyisten testauskäytäntöjen uudelleentarkastaminen projektissa saattaisi johtaa merkittäviin parannuksiin testausprosessissa. Uusien tapojen löytäminen voisi saavuttaa parempia tuloksia kustannustehokkaammin. Lopulta tarkoituksena on kuitenkin vastata asiakkaiden ja loppukäyttäjien tarpeita sekä vaatimuksia.

Jotta sovelluksesta tuli mahdollisimman tehokas, on myös tarkasteltava jo luotuja ratkaisuja ja määreitä. Nopeuden maksimoimiseksi pyritäänkin arvioimaan hidastavia tekijöitä sovelluksessa. Yksi tällainen parametri on viive, joka asetetaan flow-prosessin tilan kyselyn väliseksi ajaksi. Pienentämällä tätä väliä voitaisiin sovelluksen käyttäjäystävällisyyttä oletetusti parantaa.

Kriittisimpien ominaisuuksien, kattavien testausten sekä optimoinnin jälkeen sovelluksen vienti tuotantoon on mahdollista. Nykyistä toteutusta on kuitenkin syytä pitää uudemman rinnalla. On erityisen tärkeää arvioida ja seurata jatkuvasti sovelluksen toimivuutta tuotanto-olosuhteissa ja reagoida nopeasti havaittuihin ongelmiin tai puutteisiin.

Jälkikäteen arvioituna monia asioita olisi voinut tehdä toisin. Myös joiltain osin avoimempi yhteistyö tiimin kanssa on asia, joka estää mahdollisten virheiden syntyä. On esimerkiksi ollut tilanteita, joissa sovelluksen primäärilogiikka on

hajonnut vain sen vuoksi, etteivät kaikki osalliset ole ymmärtäneet sen toimintatapaa riittävällä tarkkuudella. Tämän vuoksi kommunikaatio työyhteisön välillä on erityisen tärkeää. Lisäksi on tärkeää huomioida, että vaikka suunnitelma sovelluksen toimintaperiaatteesta olisi tehty huolellisesti, voi siihen silti tulla merkittäviäkin muutoksia matkan varrella. Tulevaisuutta ajatellen onkin hyvä pohtia, mitä opittavaa tähänastisessa kehityskulussa on ja miten sitä voitaisiin tehostaa.

5 Yhteenveto

Tämän insinöörityön tavoitteena oli rakentaa uusi web-pohjainen käyttöliittymäsovellus Dynamo-tuotteelle pisteeseen, jossa se kykenee suoriutumaan ydintoiminnoistaan. Nykyisen käytössä olevan sovelluksen tuki ja jatkokehitys on tiensä päässä. Työssä rakennettiin toimiva sovellus Vue-ohjelmistokehyksellä, joka suoriutuu vaadittavasta ensisijaisesta toimintalogiikastaan. Myös osa sovelluksen muista ominaisuuksista saatiin käyttökuntoon.

Työ lähti liikkeelle vaatimusmäärittelyistä sekä tavoitteiden asettamisesta. Osana dokumentaatiota pyrittiin myös korkealla tasolla selvittää käyttöliittymäkomponentin suhdetta ulkoisiin komponentteihin. Tämän jälkeen valittiin sovelluksen mahdollistava ohjelmistokehys, Vue.js. Kun tarkoituksenmukaiset teknologiat oli valittu, sovelluksen ydintarkoitus oli rakennettava. Se kattaa sovelluksen ajon aloittamisen, ajon aikaisen tilan kyselemisen ja siihen reagoimisen sekä ajon jatkamisen. Kun tämä primäärilogiikka saatiin kuntoon, pystyttiin sovellukseen lisäämään muita sen vaatimia ominaisuuksia. Tämän rinnalla toimiva CDK-applikaatio kehitettiin pisteeseen, joka käsittää oleelliset Lambda-funktiot ja jolla onnistutaan sovellus viemään staattisena hosting-palveluun.

Työlle asettamat tavoitteet saavutettiin. Vaikka sovelluksen primäärilogiikka toimii oletetusti, vaatii se vielä jonkin verran työtä muiden ominaisuuksien osalta. Koska tavoitteena on lopulta saada sovellus loppukäyttäjälle, vaatii se

vielä työtä sekä ominaisuuksien osalta että testauksen suhteen. Työ toimii kuitenkin hyvänä pohjana jatkokehitykselle ja antaa suuntaviivoja siihen, mitä vaaditaan toimivaan sovellukseen. Lopullisena tavoitteena on saada käyttöön laadukas ja virheetön käyttöliittymäsovellus, joka on myös taaksepäin yhteensopiva.

Päämääränä raportin kirjoittamisessa on ollut antaa lukijalle käsitys, mikä on tämän insinöörityön aihe, mitä teknologioita siinä on käytetty ja miten, miten sovellus käytännössä toimii sekä miten projektia tullaan jatkamaan tulevaisuudessa. Olen tarkoituksella jättänyt osan kriittisestä informaatiosta kirjoittamatta yrityssalaisuuksien vuoksi. Kuvien avulla lukijan on helpompi sisäistää, mistä tekstissä on kyse.

Insinöörityön kautta olen oppinut valtavasti pilvipalveluiden hyödyistä, mahdollisuuksista, mutta myös rajoituksista AWS:n osalta. Mielenkiintoni etenkin turvallisuuden osalta on kasvanut. Omakohtainen osaaminen on kasvanut sekä yksittäisten teknologioiden osalta mutta myös laajemmin systeemitasolla. Kokonaisuuden ymmärtäminen eli kuinka eri komponentit vaikuttavat toisiinsa on aivan keskeistä ohjelmistoprojekteja tehdessä. Jatkuva oppiminen ja uuden tiedon aktiivinen hakeminen tulevat mahdollistamaan tämänkin sovelluksen onnistuneen jatkokehityksen.

Lähteet

1 About us. Verkkoaineisto. Documill Oy. <https://www.documill.com/about-documill>. Luettu 24.2.2023.

2 CRM 101: What is CRM? Verkkoaineisto. Salesforce Inc. <https://www.salesforce.com/crm/what-is-crm/>. Luettu 24.2.2023.

3 Documill Dynamo – Document Automation: Generate, Review, Track, and E-sign. Verkkoaineisto. Documill Oy. <https://appexchange.salesforce.com/listingDetail?listingId=a0N30000009whyxEAA>. Luettu 24.3.2023.

4 What Is An API (Application Programming Interface)? Verkkoaineisto. Amazon Web Services Inc. <https://aws.amazon.com/what-is/api/>. Luettu 24.2.2023.

5 Rouse, Margaret. 2021. What Does Graphical User Interface Mean? Verkkoaineisto. Techopedia. <https://www.techopedia.com/definition/5435/graphical-user-interface-gui>. Päivitetty 28.5.2021. Luettu 24.2.2023.

6 Thompson, Mark. 2022. Discontinued Long Term Support for AngularJS. Verkkoaineisto. Angular Blog. <https://blog.angular.io/discontinued-long-term-support-for-angularjs-cc066b82e65a>. 12.1.2022. Luettu 17.2.2023.

7 Version Support Status: What does end of support mean? Verkkoaineisto. AngularJS. <https://docs.angularjs.org/misc/version-support-status>. Luettu 17.2.2023.

8 Recent Angular JS Security Vulnerabilities. Verkkoaineisto. StackWatch. <https://stack.watch/product/angularjs/>. Luettu 11.3.2023.

9 Macdonald, Millie. 2020. Modular programming: Beyond the spaghetti mess. Vekkoaineisto. <https://www.tiny.cloud/blog/modular-programming-principle/>. 28.7.2020. Luettu 24.2.2023.

10 Meruliya, Pinal. 2022. Benefits of modular programming and how to avoid spaghetti mess with DhiWise. Verkkoaineisto. Medium. <https://medium.com/dhiwise/benefits-of-modular-programming-and-how-to-avoid-spaghetti-mess-with-dhiwise-4f37212fa074>. 13.4.2022. Luettu 24.2.2023.

11 Villa Fernandez, Santiago. 2019. Code readability matters. Verkkoaineisto. The Guardian. <https://www.theguardian.com/info/2019/jan/29/code-readability-matters>. 29.1.2019. Luettu 24.2.2023.

12 Mehta, Darshan. 2022. The Importance of Deleting Unused Code. Verkkoaineisto. Goldman Sachs. <https://developer.gs.com/blog/posts/importance-of-deleting-unused-code>. 13.7.2022. Luettu 25.2.2023.

13 JS framework benchmark. Verkkoaineisto. <https://krausest.github.io/js-framework-benchmark/current.html>. Luettu 25.2.2023.

14 AngularJS Performance. 2015. Verkkoaineisto. Packt Publishing Limited. <https://hub.packtpub.com/angularjs-performance/>. 4.3.2015. Luettu 24.2.2023.

15 AngularJS End of Life and What Does It Mean? 2021. Verkkoaineisto. Existek. <https://existek.com/blog/angularjs-end-of-life-and-what-does-it-mean/>. 19.3.2021. Luettu 24.2.2023.

16 Howarth, Josh. 2023. Internet Traffic from Mobile Devices (Mar 2023). Verkkoaineisto. <https://explodingtopics.com/blog/mobile-internet-traffic>. 6.3.2023. Luettu 24.3.2023.

17 You, Evan. Verkkoaineisto. Github. <https://github.com/vuejs/core>. Luettu 10.3.2023.

18 Introduction: The Progressive Framework. Verkkoaineisto. Vue.js. <https://vuejs.org/guide/introduction.html#the-progressive-framework>. Luettu 11.3.2023.

19 SPA (Single-page application). 2023. Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Glossary/SPA>. Päivitetty 22.2.2023. Luettu 17.3.2023.

20 Maheshwari, Deepak. 2021. Why Single Page Application (SPA) architecture is so popular? Verkkoaineisto. Medium. <https://medium.com/nerd-for-tech/why-single-page-application-spa-architecture-is-so-popular-141b85400204>. 24.4.2021. Luettu 17.3.2023.

21 Daityari, Shaumik. 2023. Angular vs React vs Vue: Which Framework to Choose. Verkkoaineisto. CodeinWP. <https://www.codeinwp.com/blog/angular-vs-vue-vs-react/>. 24.3.2023. Luettu 31.3.2023.

22 Stoyko, Tetiana. 2022. Angular vs React vs Vue: The Main Differences and Use Cases. Verkkoaineisto. Incora Software Development Company. <https://incora.software/insights/react-vs-angular-vs-vue>. 4.2.2022. Luettu 10.3.2023.

23 Introduction to the DOM. 2023. Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction>. Päivitetty 22.2.2023. Luettu 10.3.2023.

24 Virtual DOM and Internals. Verkkoaineisto. ReactJS. <https://reactjs.org/docs/faq-internals.html>. Luettu 10.3.2023.

25 What is Virtual DOM? Verkkoaineisto. Stack Overflow. <https://stackoverflow.com/questions/21965738/what-is-virtual-dom>. Luettu 10.3.2023.

26 Angular vs React vs Vue. Verkkoaineisto. NPM Trends. <https://npmtrends.com/angular-vs-react-vs-vue>. Luettu 10.3.2023.

27 Introduction: The Progressive Framework. Verkkoaineisto. Vue.js. <https://vuejs.org/guide/introduction.html#api-styles>. Luettu 24.2.2023.

28 Composition API FAQ. Verkkoaineisto. Vue.js. <https://vuejs.org/guide/extras/composition-api-faq.html>. Luettu 24.2.2023.

29 Getting started: Overview. Verkkoaineisto. ViteJS. <https://vitejs.dev/guide/>. Luettu 1.4.2023.

30 Features: Hot Module Replacement. Verkkoaineisto. ViteJS. <https://vitejs.dev/guide/features.html#hot-module-replacement>. Luettu 1.4.2023.

31 Testing: Recommendation. Verkkoaineisto. VueJS. <https://vuejs.org/guide/scaling-up/testing.html#recommendation>. Luettu 1.4.2023.

32 Why Vitest: The need for a Vite native test runner. Verkkoaineisto. Vitest. <https://vitest.dev/guide/why.html#the-need-for-a-vite-native-test-runner>. Luettu 1.4.2023.

33 Richter, Felix. 2022. Amazon, Microsoft & Google Dominate Cloud Market. Verkkoaineisto. Statista. <https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/>. 23.12.2022. Luettu 17.3.2023.

34 AWS Cloud Development Kit (AWS CDK). Verkkoaineisto. Github. <https://github.com/aws/aws-cdk>. Luettu 17.3.2023.

35 What is the AWS CDK? Verkkoaineisto. Amazon Web Services. <https://docs.aws.amazon.com/cdk/v2/guide/home.html>. Luettu 17.3.2023.

36 What is AWS CloudFormation? Verkkoaineisto. Amazon Web Services. <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html>. Luettu 17.3.2023.

37 AWS Lambda: Run code without thinking about servers or clusters. Verkkoaineisto. Amazon Web Services. <https://aws.amazon.com/lambda/>. Luettu 17.3.2023.

38 Lambda function scaling. Verkkoaineisto. Amazon Web Services. <https://docs.aws.amazon.com/lambda/latest/dg/lambda-concurrency.html>. Luettu 17.3.2023.

39 AWS Lambda Pricing. Verkkoaineisto. Amazon Web Services. <https://aws.amazon.com/lambda/pricing/>. Luettu 17.3.2023.

40 Amazon API Gateway. Verkkoaineisto. Amazon Web Services. <https://aws.amazon.com/api-gateway/>. Luettu 17.3.2023.

41 Krüger, Gerhard & Lane, Charles. 2023. How to Write a Software Requirements Specification (SRS Document). Verkkoaineisto. Perforce Software. <https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document>. 17.1.2023. Luettu 24.2.2023.

42 REST. 2023. Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Glossary/REST>. Päivitetty 22.2.2023. Luettu 25.3.2023.

43 An overview of HTTP. 2023. Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>. Päivitetty 3.3.2023. Luettu 5.3.2023.

44 What is OAuth 2.0? 2023. Verkkoaineisto. Auth0.com. <https://auth0.com/intro-to-iam/what-is-oauth-2>. Luettu 14.4.2023.

45 Introducing JSON. Verkkoaineisto. JSON org. <https://www.json.org/json-en.html>. Luettu 25.3.2023.