PHP-POHJAISEN VERKKOSOVELLUKSEN PÄIVITTÄMINEN JAVASCRIPT-ALUSTALLE

Lappeenrannan–Lahden teknillinen yliopisto LUT

Tietotekniikan kandidaatintyö

2024

Tuomas Mustakallio

Tarkastaja(t):

Yliopisto-opettaja (TkT) Erno Vanhala

TIIVISTELMÄ

Lappeenrannan–Lahden teknillinen yliopisto LUT
LUT Teknis-luonnontieteellinen
Tietotekniikka

Tuomas Mustakallio

PHP-pohjaisen verkkosovelluksen päivittäminen JavaScript-alustalle

Tietotekniikan kandidaatintyö

2024

31 sivua, 8 kuvaa

Tarkastaja(t): Yliopisto-opettaja (TkT) Erno Vanhala

Avainsanat: Verkkosovellus, Migraatio, JavaScript, Typescript, NestJS, React

Vanhojen ohjelmistojen päivitys on yleinen ongelma ohjelmistoyrityksissä. Varsinkin sivut voivat vanhentua nopeasti, jos niitä ei päivitetä. Tämä työ seuraa päivitysprojektia, jossa vanhentunut PHP-pohjainen verkkosovellus päivitetään JavaScript-alustalle. Projektissa tehdään täysin uusi verkkosovellus vanhan PHP-sivun ominaisuuksien pohjalta

Työssä tarkastellaan vanhaa ohjelmistoa, jonka jälkeen esitellään uudet teknologiat, jotka päivitykselle on valittu. Tämän jälkeen päivitysprosessia analysoidaan ja tarkastellaan mitä työkaluja siihen on käytetty. Lopuksi arvioidaan, miten palvelun päivitys on onnistunut ja miten se vertautuu vanhaan.

ABSTRACT

Lappeenranta–Lahti University of Technology LUT
School of Engineering Science
Software Engineering

Tuomas Mustakallio

Updating a PHP-based web application to a JavaScript base

Bachelor's thesis

2024

31 pages, 8 figures

Examiners: University lecturer (D.Sc.) Erno Vanhala

Keywords: Web application, Migration, JavaScript, Typescript, NestJS, React

Updating old software is a common problem in software companies. Especially websites can quickly become outdated if not updated. This work involves following an update project where an outdated PHP-based web application is being updated to a JavaScript platform. The project involves creating an entirely new web application based on the features of the old PHP page.

The work involves examining the old software, followed by introducing the new technologies chosen for the update. Then, the update process is analyzed, and the tools used for it are examined. Finally, an assessment is made of how successful the service update has been and how it compares to the old one.

KIITOKSET/ ACKNOWLEDGEMENTS

Haluan kiittää ystäviäni ja läheisiäni, jotka kannustivat minua tämän työn kanssa. Haluan kiittää työn ohjaajaa Erno Vanhalaa kärsivällisyydestä ja neuvoista tämän työn kanssa. Lopuksi kiittäisin myös esimiestäni, että hän kannusti minua valitsemaan tämän aiheen.

LYHENNELUETTELO

HTML	Hypertekstin merkintäkieli (HyperText Markup Language)
HTTP	Hypertekstin siirtoprotokolla (HyperText Transfer Protocol)
AWS	Pilvipalvelualustan tarjoaja (Amazon Web Services)
JSX	Syntaksilisä JavaScript-ohjelmointikieleen (JavaScript Syntax Extension)
TypeORM	TypeScript kielelle Objekti-relaatiokartoitus työkalu (TypeScript Object-Relational Mapper)
GraphQL	Tietojen kysely- ja käsittelykieli (Graph Query Language)
SQL	Standardoitu kyselykieli relaatiotietokannoille (Standard Query Language)
PHP	Ohjelmointikieli (PHP: Hypertext Processor)
DevOps	Kehityksen, testaamisen ja ylläpidon automatisointi (Development and Operations)
CI/CD	Jatkuva ohjelmistointegraatio ja jatkuva jakelu (Continuous Integration and Continuous Delivery)
S3	Yksinkertainen varastointipalvelu (Simple Storage Service)
EC2	Elastinen laskentapilvi (Elastic Compute Cloud)
ECR	Elastinen konttirekisteri (Elastic Container Registry)
ECS	Elastinen konttipalvelu (Elastic Container Service)

Sisällysluettelo

1 Johdanto

Internet on täynnä vanhoja verkkosovelluksia, joita ihmiset käyttävät säännöllisesti ja joiden palveluita he tarvitsevat. Kuitenkin web-teknologioiden kehittyessä näiden ohjelmistojen rakenne ja tekniikka vanhenee ja samalla vaikeutuu myös niiden ylläpito ja päivittäminen. Myös uusien teknologioiden lisääminen vanhoihin sivuihin on vaikeampaa, jos niille ei enää tarjota samanlaista tukea. Tämä johtaa lopulta pisteeseen, missä vanha ohjelmisto on tarpeellista päivittää uudempaan teknologiaan, jotta sen toiminta jatkuu ja se pysyy käytettävänä. Tämä kandidaatintutkielma keskittyy tällaisten verkkosovelluksien uudelleen rakentamiseen nykyaikaisemmalla teknologialla. Työssä tarkastellaan päivittämiseen johtavia syitä, mahdollisia moderneja teknologiavaihtoehtoja ja päivitysprosessia. Työn esimerkkinä käytetään verkkosovelluksen päivitysprojektia, joka tehtiin Visma Consulting yritykselle (nyk. *t*woday).

1.1 Tausta

Projektin tavoitteena on päivittää verkkosovellus, joka on tarkoitettu vammaisavustuksen löytämiseen. Palvelu on käytössä aluekohtaisesti, joten sen sisältö tulee olla aluekohtaista. Palvelun päätoiminnallisuus on auttaa vammaisavustajia ja vammaisavustusta tarvitsevia löytämään toisensa työilmoitusten kautta ja lähettämään viestiä toisilleen. Palvelussa on myös ylläpitotoiminto palveluntarjoajalle, josta he voivat hyväksyä ja hallita käyttäjiä. Tämä tarkoittaa, että palvelun toteuttamisessa piti huomioida, että sivun käytettävyys pysyy samana esimerkiksi käyttäessä näytönlukijaa.

Vanha palvelu on saman yrityksen aiemmin toteuttama, joka on rakennettu PHP:llä (PHP: Hypertext Preprocessor) ja sitä voi hallinnoida Drupalin kautta. Projektissa ei kuitenkaan käytetty vanhaa lähdekoodia ollenkaan. Uutta palvelua lähdettiin kehittämään sen tarjoamien toiminnallisuuksien perusteella ja käyttäjäpolut haluttiin pitää mahdollisimman samanlaisena. Päätös vanhan applikaation päivittämiseen nousi yrityksen linjauksesta, jossa päätettiin, että yritys ei ylläpidä ohjelmistoja, jotka ovat kehitetty PHP-kielellä ja haluaa

päivittää ne modernimpaan tekniikkaan. Päätös johtui siitä, että PHP-osaaminen oli vähäistä yrityksen sisällä, koska teknologia alkaa olla vanhaa. Palvelu alkoi myös olla aika vanha, joka tarkoitti, että sillä oli suurempi riski tietoturva-aukoille, joten päivitys todettiin tarpeelliseksi.

1.2 Työn tavoitteet ja tutkimusmenetelmä

Työn haasteena on valita oikeat työkalut ja resurssit, jotka sopivat täyttämään projektin vaatimukset ja tekevät siitä paremmin laajennettavan tulevaisuudessa. Toinen haaste on tietenkin itse verkkosovelluksen toteutus. Työ alkaa olemassa olevan verkkosovelluksen määrittelyllä. Mitä ominaisuuksia se sisältää ja millainen käyttäjäkunta sillä on. Kun ymmärretään minkälaisia toiminnallisuuksia pitää toteuttaa ja tiedetään, kuinka suuri käyttäjämäärä on ja millaisia käyttäjiä he ovat voidaan oikea web-teknologia valita. Kun teknologiapino on valittu, analysoidaan, kuinka hyvin se oikeasti toimii verrattuna vanhaan palveluun ja ovatko alussa määritellyt tavoitteet toteutettu.

Esimerkkinä käytetyssä verkkosovelluksessa käytetään palvelinpuolella Node.js-pohjaista Nest.js-sovelluskehystä ja tietokannan hallintaan käytetään TypeORM:ia (TypeScript Object-Relational-Mapping eli TypeScript Objekti-Relaatiokartoitus). Käyttöliittymä puolella taas käytetään React-sovelluskehystä ja näiden välisenä rajapintana käytetään GraphQL-rajapintaa (Graph Query Language eli Tietojen kysely- ja käsittelykieli), joka sidotaan yhteen Apollo-palvelinrakenteella. Koko verkkosovellus on kirjoitettu TypeScriptillä.

Työ seuraa tätä päivitysprosessia ja analysoi sen tuloksia. Tavoitteena on ensin tutkia, miten tällainen päivitystyö toteutetaan edellä mainituilla teknologioilla ja työkaluilla. Tämän jälkeen arvioidaan kuinka hyvin uusi palvelu vastaa sille asetettuja vaatimuksia ja miten se vertaa vanhaan palveluun. Työn tutkimuskysymyksiksi voidaan siis määritellä:

1) Miten valitut teknologiat sopivat päivitykseen?
2) Miten päivitetty palvelu vertautuu vanhaan?

Työn tarkoitus on rajata tutkimus palvelinpuolen(backend) ja käyttöliittymän(frontend) kehitys työkaluihin ja niillä uuden verkkosivun toteuttamiseen. Aluksi työhön oli tarkoitus sisällyttää myös pilvipalveluilla toteutettu verkkoinfrastruktuuri ja tietokantamalli, mutta tämä

tekisi työstä liian suuren, joten pilvipalvelu valitseminen uuteen verkkosivuun käydään hyvin pintatasolla tutkimuksessa.

Tutkimusmenetelmänä käytetään case-tutkimusmallia, jossa esimerkkinä toimii edellä mainittu päivitysprojekti. Työssä ei ole tavoitteena selvittää, mikä olisi paras mahdollinen tapa tällaiselle projektille, vaan työssä tutkitaan miten valitut työkalut ja teknologiat sopivat päivitystarpeisiin.

1.3 Työn rakenne

Tämä työ seuraa normaalia kandidaatintyön rakennetta. Toisessa luvussa käsitellään työhön liittyvää aiempaa tutkimusta ja samanlaisia tutkimuksia samalla tutkimusmenetelmällä. Kolmannessa kappaleessa tarkastellaan ensin vanhan palvelun arkkitehtuuria. Tämän jälkeen esitellään tekniset vaatimukset uudelle palvelulle ja millä teknologioilla se kehitetään. Kun vaatimukset ovat selvät dokumentoidaan, kuinka päivitys tehdään. Neljännessä kappaleessa analysoidaan uutta palvelua, miten sen kehitys onnistui, miten se täytti vaatimukset ja miten se vertautuu vanhaan palveluun. Viidennessä luvussa arvioidaan miten työ vastasi edellä mainittuihin tutkimuskysymyksiin ja miten työtä voisi jatkaa eteenpäin.

2 Tutkimukset aiheesta

Vanhat ohjelmistot ja niiden päivittäminen on varsin yleinen ongelma tietotekniikan alan yrityksillä ja aiheesta on tehty paljon tutkimuksia. PHP on ollut pitkään erittäin suosittu ohjelmointikieli, mutta nykyään TIOBE-indeksin, RedMonk-sijoituksen ja Github-tilastojen perusteella Javascript on ohittanut sen suosiossa [1]–[3]. Nämä Javascript-ohjelmat yleisimmin pyörivät Node.js-ajoympäristössä. Tutkimuksessa ensin tutkitaan hyötyjä Node.js ja PHP:n välillä ja mitä sovelluskehyksiä kannattaa käyttää sen kanssa.

Node.js on tutkitusti nopeampi I/O-operaatiossa (input/output, suomeksi kirjoitus- ja lukuoperaatio) ja resurssien käytössä kuin PHP yhdistettynä yleisin käytettyyn Apache-palvelinohjelmaan, mutta Node.js on hitaampi staattisten tiedostojen käsittelyssä sisäänrakennetulla HTTP-palvelimella (Hypertext Transfer Protocol eli hypertekstin siirtoprotokolla). Tärkeintä kuitenkin varsinkin tässä verkkosovelluksessa on nopea ja helposti käytettävä käyttäjäkokemus sivun tietokannan kanssa, johon tarvitaan nopea yhteys palvelinpäähän. [4]

Rajapinnan suunnittelussa voidaan verrata REST- ja GraphQL-rajapintoja. Vaikka REST-rajapinta on yleisempi, on tutkittu, että GraphQL sopii paremmin rajapintakutsuihin, jotka sisältävät monta parametria. GraphQL on myös tutkitusti helpompi kehittää, vaikka ohjelmistokehittäjillä ei olisi kokemusta kyseisen teknologian kanssa. [5]

Toinen tutkimus myös löysi, että GraphQL vähentää JSON-dokumenttien kenttien määrää 94 % verrattuna REST-rajapintaan ja vähentää tavujen määrää 99 % [6]. Tämä ero kannattaa ottaa huomioon varsinkin esimerkki verkkosovelluksessa, jossa tiedonsiirto on todella suuri osa verkkosovelluksen toimintaa.

Tällaista päivitysprosesseista on myös tehty ennenkin tutkimuksia. Yhdessä tutkimuksessa tarkasteltiin vanhentuneen verkkosovelluksen päivittämistä Angular-kirjastoon, joka on myös JavaScript-pohjainen. Siinä tultiin johtopäätöksiin, että päivityksessä koodin standardisointi ja ylläpidettävyys on erittäin tärkeää jatkuvalle kehitykselle ja laajennus kannattaa olla mahdollisimman helppoa. [7] Toinen tutkimus myös painotti mobiilinäkymien tukemista uusissa verkkopalveluissa ja sitä, että uuden verkkosivu käyttäjäpolut kannattaa toteuttaa tutkimalla käyttäjiä. [8]

3 Tekninen toteutus ja analyysi

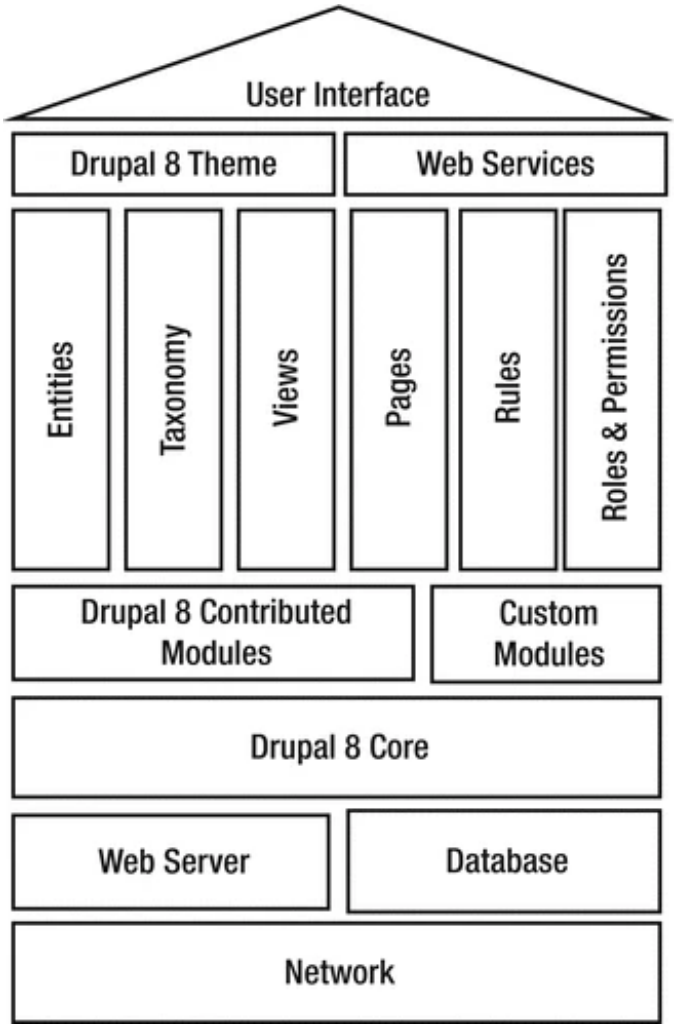
Tässä kappaleessa käsitellään palvelun teknistä toteutusta. Ensin tarkastellaan vanhan toteutuksen arkkitehtuuria, jonka jälkeen kuvataan uuden palvelun suunniteltua arkkitehtuuria, mikropalveluita ja sen ylläpitomallia. Tämän jälkeen tarkastellaan tarkemmin palvelinpuolen ja käyttöliittymäpuolen valittuja teknologioita, mitä hyötyä niistä on ja miten ne toimivat toistensa kanssa.

3.1 Vanha palvelu

Vanha palvelu on kirjoitettu PHP-kielellä. PHP on palvelinpuolella käytettävä skriptauskieli, jolla tehdään enimmäkseen dynaamisia nettisivuja ja se on avointa lähdekoodia. PHP-syntaksi on samanlaista kuin esim. C-kielessä. Verkkosivuja kehittäessä PHP käsitellään verkkopalvelimella, joka palauttaa luodun HTML-vastauksen prosessoidun koodin perusteella.

Palvelu on tehty Drupal-sisällönhallintajärjestelmän päälle. Se toimii kehyksenä verkkosivun hallitsemiselle. Drupal tarjoaa erilaisia sisällönhallinta työkaluja palvelun ja sen sivun hallitsemiseen, sillä voi hallita käyttäjiä ja se tarjoaa erilaisia moduuleja, joilla sivun toiminnallisuuksia voi laajentaa. Drupalilla on oma käyttöliittymä, jonka kautta näitä voi helposti käyttää. Tietokantana käytössä on MySQL, joka on relationaalinen tietokanta ja se on myös avointa lähdekoodia. Se on yleisesti käytetty ja on nopea tiedon haussa. MySQL käyttää SQL-kieltä (Structured Query Language), joka on standardisoitu kyselykieli, mikä mahdollistaa tiedonhaun kannasta.

Tämä applikaatio ja sen tietokanta ovat Linux-koneella, jossa on erilaisia cron-ohjelmia eli ajastettuja ohjelmia sähköpostien lähettämiselle ja käyttäjähallintaan. Alla tyypillinen Drupal sivun arkkitehtuuri.



Kuva 1. Drupal-arkkitehtuuri. [9]

Vanhan palvelun hallinta tehdään enimmäkseen Drupalin oman käyttöliittymän kautta. Sen kautta voidaan tehdä tarpeellisia muutoksia käyttäjätietoihin tai muuhun sisäiseen tietoon ja sen kautta pystyy muokkaamaan ulkoisten sähköpostiviestien pohjia. Vanhassa palvelussa aluekohtainen jako on tehty tekemällä vanhasta palvelusta eri sivut eri alueille, joka tarkoittaa, että niillä on myös omat tietokantansa. Vanhassa palvelussa sivut ja tietokannat ovat eroteltu myös eri palvelimille alueiden mukaan, mutta ne käyttävät kuitenkin samaa koodikantaa verkkosivuihin.

3.2 Arkkitehtuuri ja suunnittelu

Uudessa palvelussa palvelinpuoli ja käyttöliittymäpuoli tehdään erikseen. Käyttöliittymänpuoli toimii staattisena verkkosivuna, joka lähettää pyyntöjä palvelinpuolelle, joka sitten käyttää tietokantaa, käyttäjänhallintaa ja autentikointia ja lähettää viestejä sähköpostilla tai tekstiviestitse. Uutena ominaisuutena sivuun tulee tekstiviestilähetys, joka on tarkoitettu käyttäjätietojen lähettämiseen uusille käyttäjille, jotka pääkäyttäjä luo, jos henkilö ei itse pysty luomaan käyttäjää tai hänellä ei ole sähköpostia.

Palvelinpuolelle tulevien pyyntöjen rajapinta toteutetaan GraphQL:n ja Apollon avulla. Pyynnön mukaan palvelin autentikoi käyttäjän ja vahvista hänen käyttämän session, jonka kanssa käyttäjä voi hakea tietoa kirjautuneena. Kirjautuminen tapahtuu palvelinpuolen päässä AWS (Amazon Web Services) Cognito-käyttäjänhallinta työkalua käyttäen, jonka avulla käyttäjiä voidaan hallita ja jossa käyttäjänimi ja salana pidetään erillään. Kun käyttäjä on autentikoitu hän voi lähettää pyyntöjä sivun tiedoista, jotka haetaan tietokannasta.

Tietokantana käytetään PostgreSQL-tietokantaa, joka on relationaalinen tietokanta ja käyttää SQL-hakukieltä. Tämä myös auttaa vanhojen tietojen migraatiossa uuteen palveluun, kun tietokannan rakenne on samanlainen. Tietokantaan taas tallennetaan kaikki muu tieto käyttäjistä, heidän ilmoituksistaan, sivulla lähetettävistä viesteistä ja alueista, jossa käyttäjät ovat.

3.2.1 AWS Cloud

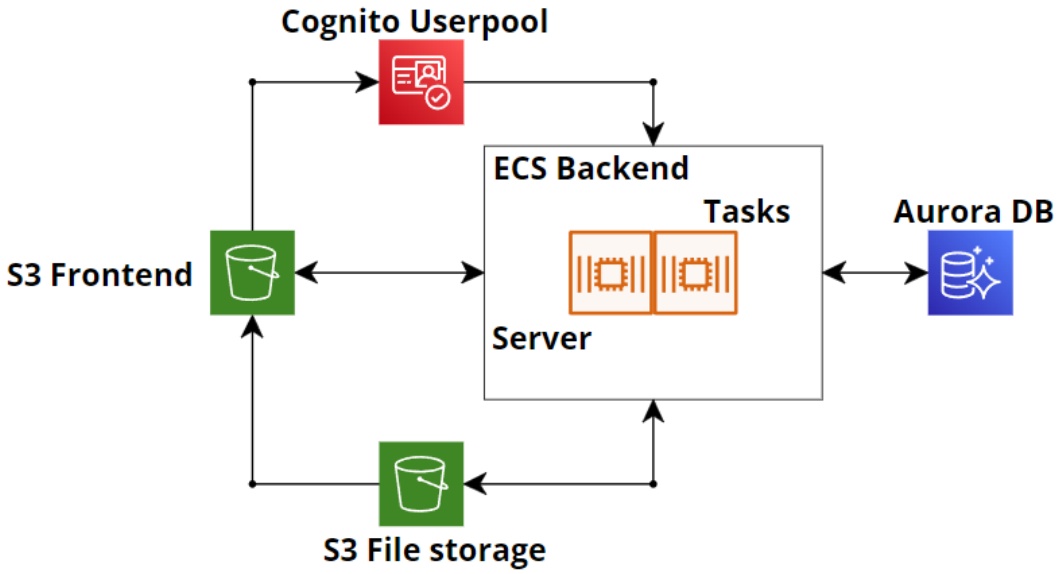
Infrastruktuurin rakentamisessa tälle palvelulle käytetään pilviratkaisuja. Pilvipalveluiden tarjoajaksi valittiin AWS. Pilvipalvelut ovat tietojenkäsittelypalveluita, joita käytetään tarvittaessa asiakkaan vaatimusten mukaan. Tämä mahdollistaa, että palvelun kehittäjien ei itse tarvitse huolehtia fyysisestä infrastruktuurista esim. palvelinkoneista tai datan säilyttämisestä vaan pilvipalvelusta vuokrataan näitä resursseja. [10] Amazon Web Services lyhyemmin AWS omaa eniten näitä palveluita kaikista pilvipalveluiden tarjoajasta. AWS keskittyy enimmiten siihen, että heidän palveluitansa voi vuokrata millä hetkellä vain, ne ovat jaettu ympäri maapalloa, jotta palvelut ovat lähellä ja nopeasti saatavilla ja ne skaalautuvat nopeasti, mutta niistä laskutetaan vain se mitä asiakas käyttää. [11] Tärkeimmät pilvityökalut,

joita tähän palveluun käytettiin, olivat S3, EC2, ECR, ECS, Cognito ja Aurora, jotka avataan seuraavaksi.

S3 (Simple Storage Service, suomeksi yksinkertainen varastointipalvelu) on objektien varastointi palvelu, jota käytetään palvelussa käyttöliittymän ylläpitoon ja profiilikuvien ja viestien liitetiedostojen säilömiseen. S3 tarjoaa turvallisen tavan säilyttää tätä dataa ja sen käyttöä voidaan rajoittaa. S3-palvelussa objektien säilömiseen käytetään resursseja, joita kutsutaan nimellä ”Bucket” eli Ämpäri. Kaksi muuta S3-resurssia on tarkoitettu tiedostojen säilyttämiselle. Profiilikuvat ja liitetiedostot ovat kahdessa eri resurssissa, ja niitä voi käyttää vain luomalla hetkellisen linkin, jonka kautta tiettyyn tiedostoon pääsee käsiksi.

Palvelinpuolen sovellus on taas jaettu kahteen osaan ja ne ovat ”kontitettu”. Kontilla tarkoitetaan ohjelman virtualisointia moneksi kevyeksi itsenäiseksi järjestelmäksi [12]. Tämä kontti pyörii ECS-palvelussa (Elastic Container Service suomeksi Elastinen konttipalvelu) ja sitä ylläpidetään ECR (Elastic Container Registry suom. Elastinen konttirekisteri) kautta. Yksi sovellus on rajapintaa varten, joka käsittelee käyttäjän pyynnöt ja toinen sovellus on järjestelmän tehtäviä varten, kuten sähköpostien lähettämiseen.

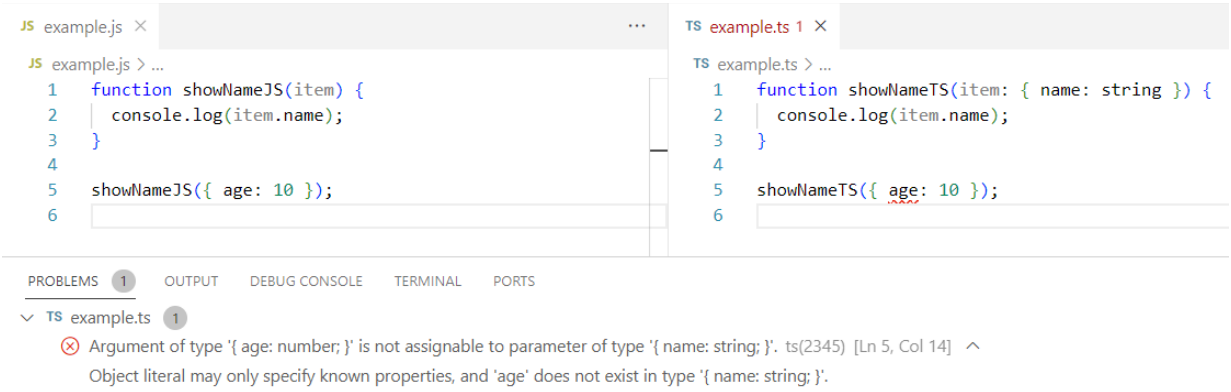
Cognito-käyttäjänhallinta järjestelmää taas käytetään käyttäjätietojen säilyttämiseen ja verifikointiin. Se tarjoaa oman kirjautumisen, rekisteröitymisen ja OAuth2 autentikoinnin, josta käyttäjät kirjautuvat omilla tunnuksillaan. Salasanat pysyvät pelkästään Cognito-palvelussa, eikä niitä säilytetä tietokannassa. Cognito myös hoitaa käyttäjien luonnin ja tiedon lähettämisen sähköpostin tai SMS-viestien kautta. Muut tiedot säilytetään PostgreSQL-tietokannassa, jota ylläpidetään Aurora-palvelun kautta. Aurora on tietokannan säilytyspalvelu relationaalisille tietokannoille. Alla yksinkertaistettu kuvaaja tästä arkkitehtuurista.



Kuva 2. Yksinkertaistettu AWS-arkkitehtuuri.

3.3 TypeScript

TypeScript oli valittu ohjelmointikieli koko palvelulle, sillä kehitettiin palvelin ja käyttöliittymä. TypeScript oli sopiva valinta projektille JavaScript-kielen yleisyyden vuoksi yrityksessä ja koska se oli kehittäjille tuttu kieli. TypeScript on Javascript-kielen syntaktinen laajennus, joka kääntyy ennen ohjelman suoritusta JavaScriptiksi. TypeScript lisää tyypityksen koodiin, jonka se tarkastaa ennen ohjelman ajoa ja huomauttaa, jos ohjelmassa on tyypitysvirheitä. Se lisää myös muita työkaluja ohjelmoinnin parantamiseen, kuten määritystiedostot ja rajapintaluokat.

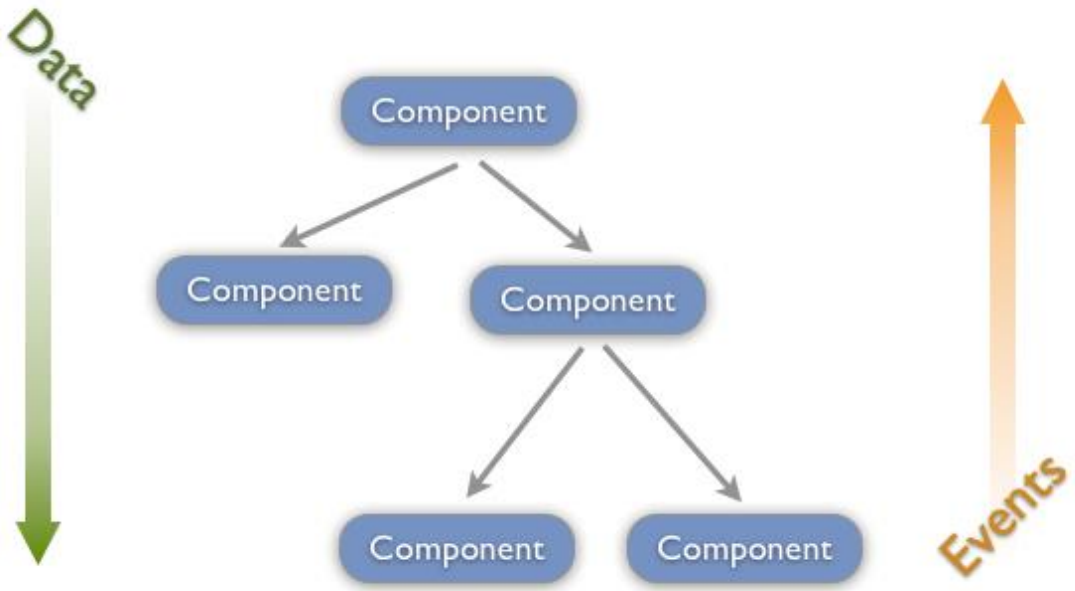


Kuva 3. Funktion kutsuminen väärällä tyypillä JavaScriptissä ja TypeScriptissä.

Koska Typescript on laajennus JavaScriptin päälle, se toimii JavaScript kirjastojen kanssa ja sen syntaksi on sama. TypeScript kuitenkin lisää turvallisuutta ohjelmansuoritukseen pakottamalla tyypityksen koodiin. Typescript tarjoaa myös tuen React-kehykselle, jota käytetään käyttöliittymän tekemisessä, NestJS-kehykselle, jota käytetään palvelimen suunnittelussa ja Apollon GraphQL-hallintatyökalun kanssa, jota käytetään rajapinnan kehittämisessä.

3.4 React

Käyttöliittymän kehitys tehtiin kokonaan Reactin avulla. React on JavaScript-kirjasto, joka on tarkoitettu dynaamisten käyttöliittymien kehittämiseen ja sen on Facebookin kehittämä. Kirjasto on avointa lähdekoodia ja sille on laaja valikoima kirjastoja, joita sen kanssa voi hyödyntää. Reactin filosofia perustuu komponenttien renderöimiseen. React-komponentit ovat uudelleenkäytettäviä ja niiden avulla sivun pystyy jakamaan omiin komponentteihinsa. Reactissa myös ominaista on sen tilan hallinta komponenteissa, joka tapahtuu ”Hook”-funktioiden avulla. Nämä funktiot tarjoavat tavan hallita komponentin tilanhallintaa ilman erillistä luokkaa sille [13]. Tiedon siirto näissä komponenteissa liikkuu ensin alas ”props”-muuttujien kautta ja takaisin ylös komponenttitapahtumien kautta. React-komponentit toimivat siis ottamalla tietoa ylempää tiedostorakenteesta, jonka jälkeen ne renderöivät eli palauttavat JSX- tai TSX-elementin, joka on syntaksiltaan samanlaista kuin HTML, mutta siinä on komponentissa käytettyä funktionaalisuutta. Tämä sitten muutoksesta lähettää tapahtumia ylöspäin tiedostorakenteessa. React valittiin myös sen yleistyneen käytön myötä yrityksen sisällä ja kehittäjät olivat myös ennen tehneet töitä Reactin parissa.



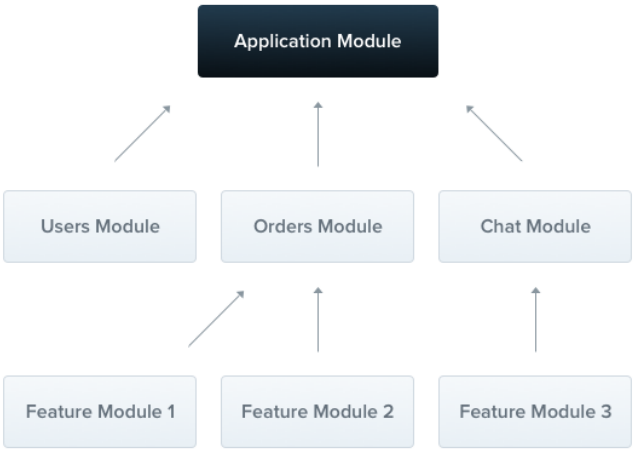
Kuva 4. React yksisuuntainen tietovirta [14].

3.4.1 React-kirjastot

Kuten mainittu, Reactilla on paljon kirjastoja, joita sen kanssa voi käyttää. Tässä projektissa käytettiin monia, mutta eniten käytettyjä näistä olivat MUI-materials, Formik ja React Router. MUI-materials kirjastoa käytettiin sen valmiiden UI-komponenttien takia. MUI-tarjoaa laajan valikoiman UI-komponentteja, joita käytetään HTML-elementtien sijaan. Näitä on helppo muokata, koska ne näyttävät esteettisemmiltä, kuin standardi HTML-elementit ja ne auttavat käytettävyyden kanssa. Formik on kirjasto, jota käytetään erilaisten kyselyjen kanssa. Tätä voi hyödyntää esim. käyttäjätietojen ja ilmoitusten kanssa ja Formikilla on myös oma validointi ja tiedon hallintaan työkaluja. Kolmas käytetyimmistä kirjastoista on React Router, jolla hallitaan URL-osoitetta ja se hallitsee komponentteja sen mukaan. Reactin kanssa oli myös käytössä Apollo- ja GraphQL-kirjastot, joita käytettiin API-kutsujen hallinointiin.

3.5 NestJS

Palvelinpuolen kehittämiseen valittiin NestJS Typescript toteutuksella. NestJS on sovelluskehys Node.js-palvelinsovelluksiin. Se käyttää hyväkseen Express http-palvelinkehystä oletuksena pohjalla, mutta sen kanssa voi käyttää myös muita palvelinkehyksiä. Tässä projektissa käytettiin Apollo-palvelinpohjaa, koska se tukee GraphQL-rajapintaa. NestJS valittiin siksi, koska se tukee tasaisesti myös Typescript-kieltä, se tarjoaa valmiin tuen GraphQL-rajapintaan Apollo-palvelimen kanssa ja se tarjoaa sopivan arkkitehtuuripohjan, jotta palvelua on helppo myös laajentaa. NestJS toimii moduulipohjaisella arkkitehtuurilla. Tämä tarkoittaa, että sovellus jakautuu moduuleihin, jotka jaetaan asiakohtaisesti. Jokaisella omalla sovelluksella on oma juurimoduuli, jossa konfiguroidaan tietokanta, määritellään alemmat moduulit ja hoidetaan käyttäjän autentikointi.



Kuva 5. NestJS palvelinarkkitehtuuri [7].

Sen alla olevat moduulit koostuvat sen omasta juurimoduulista ja Resolver ja Service ohjelmista. Moduuleissa on myös omat objektiluokat GraphQL-kutsuja varten. Resolver-ohjelma käsittelee GraphQL-kutsuja, jota palvelimelle tulee ja ohjaa ne oikeaan palveluun. Service-ohjelma sisältää nämä palvelufunktiot, jotka ovat taas yhteydessä tietokantaan ja käsittelevät sen tiedon kutsun mukaisesti.

NestJS mahdollistaa myös monorepo arkkitehtuurin, joka tarkoittaa, että yhdessä tietovarastossa voi olla enemmän kuin yksi sovellus. Tätä käytettiin myös projektissa hyväksi rakentamalla monorepo, jossa on kaksi sovellusta palvelin ja palveluviestit. Tämä mahdollistaa,

että molemmat sovellukset voivat käyttää samoja entiteettejä ja molempia voi testata samalla. [16]

3.5.1 GraphQL ja Apollo

Palvelinpuolen rajapintakieleksi valittiin GraphQL. GraphQL pyrkii erottumaan normaalista REST-rajapinnasta sillä, että GraphQL antaa käyttäjän rajata palvelimelta tulevan tiedon attribuutteja myöten. GraphQL kuvaa rajapintaa omalla GraphQL-schemalla, jossa käytetään GraphQL omaa SDL-kieltä (Schema Definition Language). Rajapinta toimii määrittelemällä ensin sille omat tyypit ja attribuutit eli kentät. Tätä tyyppijärjestelmää käytetään tiedon käsittelyyn. Sillä tehdään rajapintapyynnöt ja tieto palautetaan niitä käyttämällä. Tämän erottuvuuden takia GraphQL valittiin REST-rajapinnan yli. Kun GraphQL palvelu on käynnissä palvelimella, käyttäjä voi tehdä rajapintapyyntöjä sinne yleisesti URL-osoitteen kautta. Pyynnön tullessa palvelu ensin validoi, että pyynnössä käytetyt tyypit ovat oikein ja kun pyyntö on validoitupalvelu suorittaa siihen annetun funktion ja palauttaa tuloksen. Pyyntöjä on kahdenlaisia nimeltään ”query” eli kysely ja ”mutation” eli muutos. Kysely on tarkoitettu kutsuihin, jossa sille annettu funktio vain hakee tietoa ja muutoskyselyissä tietoa halutaan muokata, lisätä tai poistaa. Alla esimerkki kysely, jossa pyydetään tietyllä tunnisteella objektia, josta halutaan kentät ”id”, ”name” ja ”height”.

Kuva 6. Yksinkertainen GraphQL-kutsu [11, Queries and Mutations]

GraphQL-pohjaisen palvelun hallintaan otettiin käyttöön Apollon palvelin ja käyttöliittymäkirjastot. Apollon palvelin ajureita ja kirjastoa käytettiin palvelimen ajamiseen ja GraphQL-kyselyjen vastaanottoon ja niihin vastaamiseen. Käyttöliittymän puolella Apollon käyttöliittymäkirjastoa käytettiin Reactin kanssa tukemaan palvelinkutsuja. Tämän avulla käyttöliittymässä saadaan generoitua valmiit Typescript-tyypit käyttöliittymäohjelmaan palvelinpuolella luoduista GraphQL-olioista, jotka varmistavat, että kutsuissa käytetään oikeanlaisia

tyyppejä. Apollon kirjasto tarjoaa myös omat tilanhallintafunktiot GraphQL-kyselyiden tekoon ja niiden tulosten käsittelyyn generoitujen tyyppien avulla.

3.5.2 TypeORM

NestJS kanssa käytettiin myös TypeORM kirjastoa. TypeORM on ORM (Object-relational mapping tool, suom. Objekti-relaatiokartoitustyökalu). ORM-työkaluja käytetään tiedon muuntamiseen relaatiotietokannasta (SQL) olio-ohjelmointikieleen objekteiksi. Tässä tapauksessa Typescript-kielelle. Tämä tarkoittaa, että tietokannan taulut ja niiden relaatiot voidaan kirjoittaa suoraan Typescriptillä, josta ne voi luoda yhdistettyyn tietokantaan ilman, että tarvitsee kirjoittaa SQL-käskyjä. Alla esimerkki Typescriptillä määritellystä taulusta TypeORM avulla. Entity tarkoittaa taulua.

```
import { Entity, PrimaryGeneratedColumn, Column } from "typeorm"

@Entity()
export class User {
    @PrimaryGeneratedColumn()
    id: number

    @Column()
    firstName: string

    @Column()
    lastName: string

    @Column()
    age: number
}
```

Kuva 7. TypeORM Typescript objekti SQL-taulusta [18]

Näitä luotuja tauluja ja niissä olevaa tietoa voi sitten hallita Repository-objektin avulla. Repository on objekti, joka on taulukohtainen ja sillä on erilaisia metodeja, jotka vastaavat erilaisia SQL-kyselyjä, mutta nekin voidaan toteuttaa kooditasolta käsin ilman SQL-kyselyn

kirjoittamista. TypeORM tarjoaa migraatio mahdollisuuden. Jos tauluihin tulee muutoksia, niin käyttämällä TypeORM CLI-työkalua (Command Line Interface suom. Komentorivikäyttöliittymä) TypeORM hoitaa migraation myös tietokannan puolella ja tallentaa vanhan pohjan muistiin.

3.6 PostgreSQL

Tietokannaksi valikoitui siis SQL-relaatiotietokanta. Tietokannan hallintajärjestelmäksi valikoitui PostgreSQL. PostgreSQL on erittäin suosittu avoimen lähdekoodin relaatiotietokannan hallintajärjestelmä ja vuonna 2022 se oli Stackoverflow-sivun tekemän kyselyn mukaan suosituin tietokantajärjestelmä ohjelmistokehittäjien keskuudessa [19]. PostgreSQL tarjoaa oman käyttöliittymän tietokannan hallinnointiin ja sen ylläpitämiseen. PostgreSQL tukee SQL-standardikyselykieltä ja sitä pystyy laajentamaan erilaisilla funktioilla ja data tyypeillä.[20] PostgreSQL sopi hyvin palvelun uudeksi tietokannaksi, koska vanhassa palvelussa oli myös käytössä SQL-pohjainen tietokanta, joten tietomigraatio helpottui sitä kautta.

3.7 Kehitysprosessi

Palvelun kehitysprosessi alkoi vaatimusmäärittelyistä. Vanhan palvelun käyttäjäpolkuja määriteltiin ja sen tarvittavia ominaisuuksia. Näiden pohjalta tehtiin tietokanta malli ja suunniteltiin palvelimen rajapintaa. Näiden jälkeen tehtiin edellä mainitut teknologiavalinnat niiden sopivuuden ja yrityksen teknologiasuosituksien mukaan. Kun teknologiavalinnat oli tehty käyttöliittymää, alettiin suunnittelemaan visuaalisesti ja sen käytettävyyttä. Samalla projektille tehtiin Gitlab-tietovarasto. Tietovarastoon alustettiin sekä NestJS-palvelinsovellus ja React-käyttöliittymä. Koska tietokantaa ja rajapintaa oli mallinnettu valmiiksi, sen kehitys voitiin aloittaa.

Palvelimen kehitys tapahtui ensin kirjoittamalla tietokantataulut TypeORM-objekteilla ja rajapintamallit tehtiin ”koodi ensin”-lähestymisellä eli ne kirjoitettiin Typescript-objekteina, josta NestJS-palvelimen käynnistyessä generoi rajapinnan GraphQL-scheman [11, GraphQL]. Kehityksessä käytettiin lokaalia tietokantaa ja rajapintaa oli käynnissä paikallisella palvelimella, jota pystyi käyttämään Apollon sandbox-käyttöliittymän kautta, joka

varmistaa oikean SDL-formatoinnin ja näyttää GraphQL-scheman kutsuja tehdessä. Myöhemmin NestJS jaettiin monorepo-arkkitehtuuriin, johon kehitettiin myös palvelun viestien lähettämiseen erillinen ohjelma.

Palvelun AWS-infrastruktuurin kehitys tehtiin melkein kokonaan käyttämällä Terraform-ohjelmistotyökalua. Terraform on infrastruktuurin hallintaohjelma, joka mahdollistaa pilviresurssien varustamisen ja konfiguroimisen deklaratiivisella koodilla, joka kirjoitetaan käyttäen HCL-kieltä (HashiCorp Configuration Language suom. HashiCorp Konfiguraatiokieli). Terraform käyttää deklaratiivista koodia kuvatakseen enemmänkin mikä ohjelman haluttu lopputulos on, jonka sitten Terraform luo pilvialustalle. [21]

```
File: chapter3\infra\iteration1\main.tf
19: resource "aws_instance" "apache2_server" {
20:   ami           = "ami-00399ec92321828f5"
21:   instance_type = "t2.micro"
22:   user_data = << EOF
23:         #!/bin/bash
24:         sudo apt update -y
25:         sudo apt install apache2 -y
26:     EOF
27:   tags = {
28:     env = "dev"
29:   }
30: }
```

Kuva 8. Esimerkki EC2-koneen luomisesta Terraformilla. [21]

Kun käyttöliittymän alustava visuaalinen ilme saatiin valmiiksi, alettiin työstämään myös React-käyttöliittymää samalla. Käyttöliittymään näkymistä tehtiin ensin Figma-suunnittelutyökalulla visuaalisia esimerkkinäkymiä, jonka jälkeen ne toteutettiin Reactilla toiminnallisuuksien kanssa.

Kun React-sovellukseen saatiin tehtyä alustavia näkymiä, palvelut alustettiin AWS-koneille Terraformin avulla. NestJS-palvelin alustettiin ECS-konttiin ja React-käyttöliittymä alustettiin S3-säilöön. PostgreSQL-tietokanta alustettiin Aurora DB-tietokantavarastoon. AWS-alustalle alustettiin ensin kehitys käyttöön tarkoitetut koneet ja tietokanta, jolla palvelua

pystyi testata kokonaisuudessaan. Myöhemmin, kun palvelu alkoi olla käyttövalmiina, alustettiin myös tuotantokoneet alueiden omille palveluille.

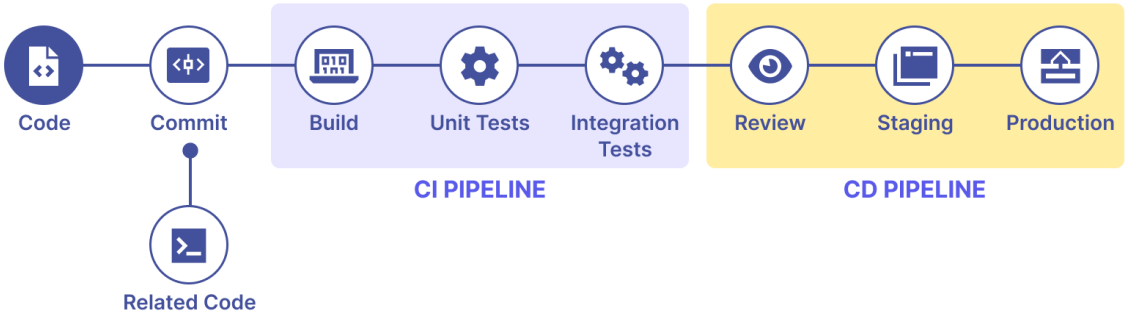
Käyttäjähallintaan otettiin käyttöön Cognito-palvelu, jonka kautta otettiin myös käyttöön sen oma käyttöliittymä kirjautumis- ja rekisteröitymissivulle. Kirjautuminen ja rekisteröityminen tehtiin tekemällä ensin kirjautumispyyntöpalvelimelle, joka ohjasi käyttäjän Cognito-kirjautumissivulle. Cognito hoiti siellä käyttäjän kirjautumisen autentikoinnin tai sen luomisen, auktorisoi käyttäjän session palvelimella, jonka jälkeen palvelin uudelleenohjasi käyttäjän palvelun kotisivulle tai profiilinluontiin. Käyttäjän luominen onnistui myös ylläpitäjän toimesta Reactin kautta, jos käyttäjä ei itse pystynyt sitä tekemään.

3.7.1 CI/CD

Kehityksessä edettiin Agile-filosofian mukaisesti sprinteissä eli toistuvissa muutaman viikon ajanjaksoissa, jossa projektia kehitetään ja testataan jatkuvasti. Tämä tarkoittaa, että yhdenkin päivän sisällä kehittäjien on pystyttävä saamaan muutoksensa näkyville tietovarastoon, testattavaksi, arvioitavaksi ja lopuksi vielä kehitysympäristöön. [22] Jotta tämä ongelma saadaan ratkaistua pitää ottaa käyttöön CI/CD-työnkulku. CI/CD tarkoitta jatkuvaa kehitystä ja jatkuvaa käyttöönottoa (eng. Continuous Integration / Continuous Deployment). Tätä varten Gitlab-tietovarastoon konfiguroitiin putkisto, jotta kehittäjien lokaalit muutokset saatiin mahdollisimman automatisoidusti näkyviin.

Gitlab tarjoaa valmiit työkalut tämän putkiston kehittämistä varten. Gitlab tarjoaa tähän Runner-agentit, putkiston ja muuttujavaraston. Runner-agentit hoitavat yksittäiset työt, jotka voivat olla esimerkiksi sovelluksen pyörittäminen, testaaminen tai alustaminen pilveen. Putkisto koostuu näistä töistä, jotka ovat jaettu tasoittain. Tasot ovat yleisimmin koontiversiotaso, testaustaso ja julkaisutaso. Muuttujissa voidaan määritellä esim. AWS-pilven osoite. Nämä konfiguroidaan YML-tiedostoon, jonka jälkeen koodimuutokset menevät tällaista reittiä. Jos työ epäonnistuu tai ei mene läpi koodikatselmoinnista muilta kehittäjiltä niin se menee takaisin iteroitavaksi aiemmalle tasolle. Kehityksessä edettiin ensin puskemalla koodimuutokset omaan versionhallintahaaraan, jonka jälkeen se koottiin, suoritettiin automaattiset testit, jotka olivat kirjoitettu ja koodimuutokset käytettiin katselmoinnin läpi muiden

kehittäjien puolesta. Kun nämä kaikki vaiheet olivat käyty läpi, uusi muutos voitiin alustaa pilveen. [23]



Kuva 9. Esimerkki CI/CD-työnkulun putkistosta. [24]

3.7.2 Testaus

Testit toteutettiin käyttäen Jest-kehystä. Jest tarjoaa valmiin kirjaston yksikkötesteille ja End-to-end-testeille, jolla voidaan testata React-komponentteja ja NestJS API-funktioita. Projektissa testattiin enimmäkseen palvelinpuolen funktioita ja niitä käytettiin itsenäisesti ja CI/CD-putkistossa. Yksikkötestit ovat funktioita, jotka testaavat yksittäistä komponenttia. Testille annetaan alkuarvo ja kerrotaan mitä komponentin lopputila tulisi olla. End-to-end testit taas testaavat sovellusta kokonaisuudessaan. NestJS ja React tarjoavat valmiin integraation Jest-työkaluille, joten testien kirjoittaminen onnistui hyvin sillä. Testejä kehitettiin koko kehitysprosessin ajan. Yleisesti uuden toiminnallisuuden tullessa sille kirjoitettiin myös testit. Tällä saatiin varmistettua, että palvelu toimi oikein ennen sen viemistä pilveen ja työtaakka keveni, koska testit saatiin automatisoitua. Lopuksi palvelua testattiin vielä normaalilla käytöllä kehittäjien puolesta.

4 Lopputulos

Uusi verkkosovellus oli valmis tuotantoon, kun se täytti kaikki vaatimuksensa ja sitä oli testattu laajalti. Tämä tarkoitti sitä, että uudesta palvelusta puuttui enää vanhojen käyttäjien migraatio ja se oli valmis käyttöön. Tämä kappale käsittelee ensin, miten migraatiovalmius saatiin vanhasta Drupal-palvelusta, jonka jälkeen käydään läpi mitä haasteita päivitysprosessissa oli ja analysoidaan päivitysprosessia ja uutta palvelua alussa määritettyjen tutkimuskysymysten perusteella.

4.1 Migraatio vanhasta palvelusta

Migraatiosta haluttiin tehdä mahdollisimman vaivaton niin kehittäjille, kuin käyttäjille. Tätä auttoi paljon se, että vanha ja uusi tietokanta on SQL-relaatiotietokanta, joten ne toimivat samalla SQL-kyselykielellä. Tämä tarkoitti, että kun molempien palveluiden taulut tiedettiin, niin niiden siirto onnistui yhdellä SQL-skriptillä. Jotta käyttäjät kuitenkin saatiin kokonaisuudessaan siirrettyä uuteen palveluun, ne piti myös lisätä AWS Cognito-käyttäjähallintaan. Tämä onnistui myös simppelillä Python-skriptillä, jolla käyttäjät lisättiin vanhasta palvelusta CSV-tiedostoon (Comma-separated values, suom. pilkulla erotellut arvot), jonka kautta ne pystyi importoida suoraan Cognitoon. Kun uudet käyttäjätiedot olivat lisätty palvelu lähetti sähköpostin tai käyttäjille tekstiviesti, jonka kautta he pystyivät kirjautumaan ja tutustumaan uuteen palveluun.

4.2 Implementaatiohaasteet

Kehitysprosessiin kuului myös omia haasteitaan. Käytettävyys oli tärkeä kohta käyttöliittymää suunnitellessa varsinkin, koska merkittävä osa käyttäjistä käyttää palvelua esimerkiksi näytönlukijan kautta. Varsinkin toiminnallisuuksia kehittäessä piti varmistaa, että palvelussa on tarpeeksi kontrastia, suurennetut näkymät pitivät toimia yhtä hyvin ja verkkosivua piti pystyä käyttämään näytönlukijalla mahdollisimman esteettömästi.

Sähköpostittomien käyttäjien autentikointi ja heidän tavoittamisensa oli myös haaste palvelussa, kun palvelun kirjautumisprosessia ja käyttäjähallintaa kehitettiin. Palvelussa sähköpostit olivat tärkeä osa, koska niillä pystyttiin tavoittamaan käyttäjät, mutta kaikilla käyttäjillä ei kuitenkaan ollut omaa sähköpostia. Tämä myös hankaloitti kirjautumisvarmuuden lisäämistä, koska käyttäjän antama sähköposti haluttiin varmistaa. Lopuksi palvelussa tehtiin rajaus, että käyttäjällä pitää olla joko sähköpostite tai puhelinnumero, jotta käyttäjän voi tehdä.

Haasteena oli myös ylläpitonäkymien parantaminen. Palvelussa ylläpitäjällä piti olla mahdollisuus tehdä kaikki mitä normaalin käyttäjäkin pystyi. Tämä oli tarpeellista, koska jotkut käyttäjistä käyttivät palvelua vain ylläpitäjien kautta, joko puhelimen kautta tai paikan päällä.

4.3 Toteutuksen arviointi ja vertaus

Vaikka päivitysprosessissa oli myös omia haasteitaan, uudet teknologiat kuitenkin tarjosivat myös ratkaisun niille. Toteutusta pitää kuitenkin arvioida jotenkin. Kun alussa tarkasteltiin aiempaa tutkimusta, todettiin että uudelle versiolle ylläpidettävyys ja laajennusmahdollisuudet ovat tärkeitä. Tärkeää on myös tarkastella päivityksessä, miten uusi palvelu vertautuu vanhaan.

Ylläpidettävyys saatiin varmistettua projektissa valitsemalla aluksi koko palvelulle yrityksessä yleistynyt kieli TypeScript. Myös kirjastot, joita projektiin käytettiin etenkin React on todella yleinen. Uutta palvelua ohjelmoitaessa oli myös aina tärkeää, että koodi on mahdollisimman hyvin standardisoitua ja itseään dokumentoivaa. Erillistä dokumentaatiota myös kirjoitettiin projektille, jotta uuden kehittäjän on helppo ylläpitää sitä. Uusi verkkosovellus on myös laajennettava. NestJS tarjoaa valmiin arkkitehtuurin, johon on helppo lisätä uusia moduuleja tai jopa uusia sovelluksia. AWS myös tarjoaa hyvät laajennusmahdollisuudet, jos käyttäjämäärät nousevat niin pilvikoneidenkin määrä nousee.

Miten uusi palvelu sitten vertautuu vanhaan? Uusi palvelu on paljon turvallisempi käyttää, koska uudet käyttäjät varmistetaan kaksiosaisellavarmistuksella, joko sähköpostin tai puhelinnumeron kautta. Tämä suojaa palvelua valekäyttäjiltä. Myös tietoturvariskejä on vähemmän. Palvelu on alustettu AWS-pilvipalveluun, joka on hyvin suosittu infrastruktuurin

tarjoaja, joka antaa varmuutta infrastruktuurin tietoturvaan. Kaikki palvelut ja resurssit ovat tiukan tietoturvamuurin ja varmennuksen takana, joka suojaa resursseja ja käyttäjätietoja. Palvelussa käytetyt kirjastot eivät myöskään ole vanhentuneita, joka vähentää koodikannan tietoturvariskejä ja koodikantaa on kehitetty ajankohtaisten tietoturvamenetelmien mukaisesti.

Uuden version käyttöliittymä on myös helppokäyttöisempi, koska siinä keskityttiin alusta lähtien käyttäjäläheiseen suunnitteluun tutkimalla vanhan palvelun käyttäjiä. Uudessa palvelussa on myös uusia ominaisuuksia käyttäjän hakuun ja profiilin muokkaukseen, mutta se pitää silti sisällään vanhan palvelun tarvittavat ominaisuudet.

5 Yhteenveto

Tässä työssä tarkasteltiin päivitysprosessia vanhentuneesta Drupal-verkkopalvelusta uuteen JavaScript-pohjaiseen verkkopalveluun. Työssä tarkasteltiin ensin vanhan palvelun teknologiaa, jonka jälkeen tarkasteltiin uuteen palveluun valittuja teknologioita. Työssä tarkasteltiin erikseen uutta määriteltyä pilvi-infrastruktuuria AWS-palvelussa ja uuden palvelun rakennetta. Työssä käytiin tarkemmin vielä läpi sen palvelin- ja käyttöliittymäohjelmia. Näihin käytettiin JavaScript-pohjaisia kirjastoja kuten NestJS ja React, mutta ne toteutettiin TypeScript-kielen avulla. Työssä ominaista oli myös GraphQL-rajapinta, joka valittiin yleisemmän REST-rajapinnan yli. Tarkasteltiin myös, miten kehitysprosessi onnistui näillä teknologioilla ja mitä kehitystyökaluja siihen käytettiin kuten CI/CD työkalut ja testaustavat.

Uusi palvelu toteutettiin onnistuneesti ja sen voi nyt päivittää vanhasta palvelusta käyttäjämigraation kanssa. Uusi palvelu täyttää sille määritellyt vaatimukset ja sitä on testattu laajasti. Alussa tälle työlle määriteltiin kaksi tutkimuskysymystä. Nämä olivat:

1) Miten valitut teknologiat sopivat päivitykseen?

2) Miten päivitetty palvelu vertautuu vanhaan?

Työn toteutuksen arvioinnin jälkeen voidaan todeta, että valitut teknologiat sopivat hyvin päivitykseen, koska niillä saavutettiin haluttu palvelu. Uutta palvelua on helppo ylläpitää, koska se on tehty yleisellä ohjelmointikielellä ja koodi on standardisoitua. Sitä pystyy laajentamaan NestJS-arkkitehtuurin avulla helposti. Uusi palvelu myös vertautuu hyvin vanhaan. Se suoriutuu paremmin kuin vanha palvelu, koska se on turvallisempi ja helppokäyttöisempi. Uusi verkkosivu pitää sisällään kaikki vanhan palvelun ominaisuudet, mutta tuo myös uusia ominaisuuksia, jotka helpottavat palvelun käyttöä.

Lopputulos esittää siis onnistuneen päivitystyön sille annetuilla teknologioilla. Työ myös tarjoaa esittelyn teknologioista, joita päivitykseen valittiin ja antaa esimerkin käytetystä päivitystavasta. Jatkossa olisi mielenkiintoista selvittää miten valitut teknologiat vertaavat muihin TypeScript-pohjaisiin kirjastoihin, miten palvelu toimisi REST-rajapinnalla tai miten päivitysprosessi onnistuisi, jos uudessa palvelussa ei olisi käytössä relaatiotietokanta. Verkkopalvelussa oli myös paljon työtä, joten päivitysprosessin automatisoinnin tutkiminen ja

sen lisäämisen mahdollisuuksien selvittäminen voisi olla arvokasta tulevaisuudessa tuleville päivitysprojekteille.

Lähteet

[1] ‘TIOBE Index for January 2024’, Jan. 2024, Accessed: Jan. 07, 2024. [Online]. Available: https://www.tiobe.com/tiobe-index/

[2] ‘The RedMonk Programming Language Rankings: January 2023’, May 2023, Accessed: Jan. 07, 2024. [Online]. Available: https://redmonk.com/sogrady/2023/05/16/language-rankings-1-23/

[3] ‘The top programming languages’, 2022, Accessed: Jan. 07, 2024. [Online]. Available: https://octoverse.github.com/2022/top-programming-languages

[4] I. K. Chaniotis, K.-I. D. Kyriakou, and N. D. Tselikas, ‘Is Node.js a viable option for building modern web applications? A performance evaluation study’, *Computing*, vol. 97, no. 10, pp. 1023–1044, Oct. 2015, doi: 10.1007/s00607-014-0394-9.

[5] G. Brito and M. T. Valente, ‘REST vs GraphQL: A Controlled Experiment’. arXiv, Mar. 10, 2020. Accessed: Nov. 24, 2022. [Online]. Available: http://arxiv.org/abs/2003.04761

[6] G. Brito, T. Mombach, and M. T. Valente, ‘Migrating to GraphQL: A Practical Assessment’, in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Hangzhou, China: IEEE, Feb. 2019, pp. 140–150. doi: 10.1109/SANER.2019.8667986.

[7] B. Verhaeghe *et al.*, ‘From GWT to Angular: An Experiment Report on Migrating a Legacy Web Application’, *IEEE Softw.*, vol. 39, no. 4, pp. 76–83, Jul. 2022, doi: 10.1109/MS.2021.3101249.

[8] V. Cajas, M. Urbieta, G. Rossi, and Y. Rybarczyk, ‘Migrating legacy Web applications’, *Clust. Comput.*, vol. 24, no. 2, pp. 1033–1049, Jun. 2021, doi: 10.1007/s10586-020-03147-6.

[9] T. Tomlinson, *Enterprise Drupal 8 development: for advanced projects and large development teams*. Berkeley, California: Apress, 2017.

[10] S. Ranger, ‘What is cloud computing? Everything you need to know about the cloud explained’, Feb. 2022, Accessed: Dec. 25, 2022. [Online]. Available: https://www.zdnet.com/article/what-is-cloud-computing-everything-you-need-to-know-about-the-cloud/

[11] O. Mustafa, ‘Overview of Amazon Web Services’, in *A Complete Guide to DevOps with AWS: Deploy, Build, and Scale Services with AWS Tools and Techniques*, O. Mustafa, Ed., Berkeley, CA: Apress, 2023, pp. 1–35. doi: 10.1007/978-1-4842-9303-4_1.

[12] N. Rudnäs, ‘Ohjelmien kontitus ja sen hyödyt’, Dec. 2018, Accessed: Dec. 23, 2023. [Online]. Available: https://lehti.seamk.fi/alykkaat-ja-energiatehokkaat-jarjestelmat/ohjelmien-kontitus-ja-sen-hyodyt/

[13] ‘React Documentation: Introducing Hooks’. 2023. Accessed: Dec. 25, 2023. [Online]. Available: https://legacy.reactjs.org/docs/hooks-intro.html

[14] D. Gosai, ‘React State vs Props: Introduction & Differences’, Jun. 2022, Accessed: Dec. 19, 2023. [Online]. Available: https://bosctechlabs.com/react-state-vs-props-introduction-differences/

[15] ‘NestJS Documentation’. 2023. Accessed: Dec. 19, 2023. [Online]. Available: https://docs.nestjs.com/

[16] K. Mysliwiec, ‘Announcing NestJS Monorepos and new CLI commands.’, Announcing NestJS Monorepos and new CLI commands. Accessed: Dec. 19, 2023. [Online]. Available: https://trilon.io/blog/announcing-nestjs-monorepos-and-new-commands

[17] ‘GraphQL Documentation’. 2023. Accessed: Dec. 19, 2023. [Online]. Available: https://graphql.org/learn/

[18] ‘TypeORM Documentation’. 2023. Accessed: Dec. 19, 2023. [Online]. Available: https://typeorm.io/

[19] ‘2022 Developer Survey by StackOverflow’. Accessed: Dec. 20, 2023. [Online]. Available: https://survey.stackoverflow.co/2022/#most-popular-technologies-database-prof

[20] ‘PostgreSQL 16.1 Documentation’. The PostgreSQL Global Development Group, 2023. Accessed: Dec. 20, 2023. [Online]. Available: https://www.postgresql.org/files/documentation/pdf/16/postgresql-16-A4.pdf

[21] R. Salecha, ‘Introduction to Terraform’, in *Practical GitOps: Infrastructure Management Using Terraform, AWS, and GitHub Actions*, R. Salecha, Ed., Berkeley, CA: Apress, 2023, pp. 67–122. doi: 10.1007/978-1-4842-8673-9_3.

[22] P. P. Dingare, ‘Understanding CI/CD’, in *CI/CD Pipeline Using Jenkins Unleashed: Solutions While Setting Up CI/CD Processes*, P. P. Dingare, Ed., Berkeley, CA: Apress, 2022, pp. 1–7. doi: 10.1007/978-1-4842-7508-5_1.

[23] ‘Gitlab documentation’. 2023. Accessed: Jan. 01, 2024. [Online]. Available: https://docs.gitlab.com/ee/ci/index.html

[24] ‘What is CI/CD? Continuous Integration & Continuous Delivery’, 2023, Accessed: Dec. 29, 2023. [Online]. Available: https://katalon.com/resources-center/blog/ci-cd-introduction