Opinnäytetyö

Tradenomi (YAMK), Ohjelmistotekniikka ja ICT

2023

Tiina Vienonen-Jokiniemi

CDR SELAUS NÄKYMÄN KÄYTTÖLIITTYMÄUUDISTUS

– React versus Vue AngularJSn korvaajana

TURKU AMK
TURKU UNIVERSITY OF
APPLIED SCIENCES

Opinnäytetyö | Tiivistelmä

Turun ammattikorkeakoulu

Tradenomi(YAMK), Ohjelmistotekniikka ja ICT

2023 | 51 sivua

Tiina Vienonen-Jokiniemi

CDR selausnäkymän käyttöliittymäuudistus

- React versus Vue, AngularJsn korvaajana

Tämän opinnäytetyön tarkoituksena oli selvittää React ja Vue webteknologioiden soveltuvuutta teleliikennetietojen keräämiseen ja analysointiin tarkoitetun sovelluksen käyttöliittymäuudistukselle. Tämä työ on osa isompaa järjestelmän päivitysprojektia ja toteutettiin yhteistyössä suomalaisen teleoperaattorin ja CGI Suomi Oy:n kanssa.

Opinnäytetyö rakentuu kolmesta osasta. Ensimmäisessä osuudessa perehdytään teorian kautta React ja Vue webteknologioihin, kun taas toisessa osassa teknologioihin tutustutaan sovelluksen näkökulmasta, toteuttamalla CDR selausnäkymä nykyisestä sovelluksesta. Kolmannessa osiossa on keskitytty tarkemmin käytettyjen webteknologioiden eroavaisuuksiin ja siihen, miten ne eroavat nyt käytössä olevasta AngularJS webkehyksestä.

Tutkimuksen aikana toteutettiin kaksi toimivaa käyttöliittymää. Sovellusratkaisujen perusteella kumpikin teknologioista sopii korvaamaan nykyisen AngularJS käyttöliittymän, mutta varsinaiseen teknologiavalintaan vaikuttaa enemmän sovelluskehysten oppimiskäyrä, koodin luettavuus ja ylläpidettävyys sekä dokumentaatio ja avun saatavuus.

Asiasanat:

React, Vue, Käyttöliittymä, modernit webteknologiat

Master’s Thesis | Abstract

Turku University of Applied Sciences

Master’s Degree Programme in Software Engineering and ICT

2023 | 51 pages

Tiina Vienonen-Jokiniemi

User interface upgrade for CDR Search view

- React versus Vue as replacement for AngularJS

The aim of the present Master’s thesis is to investigate the applicability of React and Vue web technologies for the telecommunication analysis system. The study is part of the bigger system upgrade project, and it was implemented in cooperation with a Finnish telecommunication operator and CGI Suomi Oy.

The thesis consists of three parts. In the first part, React and Vue web technologies are discussed through theory, while in the second part, the technologies are introduced from the perspective of the application, by implementing a CDR browsing view of the current application. The third part focuses more on the differences between the selected web technologies and how they differ from the currently used AngularJS web framework.

During the research, two functional user interfaces were implemented. Based on the application solutions, both technologies are suitable for replacing the current AngularJS user interface, but the actual choice of the technology is more influenced by the learning curve of the application frameworks, the readability and maintainability of the code, as well as the availability of documentation and help.

Keywords:

React, Vue, User interface, modern web technologies

Sisältö

Käytetyt lyhenteet tai sanasto **7**

1 Johdanto **8**

1.1 Tutkimuksen rajaus ja rakenne 9

2 Nykytilanne **11**

2.1 Käyttöliittymä 11

2.2 Haasteet 12

3 React.js **13**

3.1.1 CLI 13

3.1.2 Komponentit 14

3.1.3 Syntaksi & JSX 15

3.1.4 DOM 16

3.1.5 Tilanhallinta 16

3.1.6 Router 17

4 Vue.js **18**

4.1.1 CLI 18

4.1.2 Komponentit 19

4.1.3 Syntaksi 20

4.1.4 DOM 22

4.1.5 Tilanhallinta 22

4.1.6 Router 23

5 CDR selaus näkymän uudistaminen **24**

5.1 React toteutus 24

5.1.1 Projektin luominen 25

5.1.2 syntaksi 25

5.1.3 Komponentit 26

5.1.4 Tilanhallinta 28

5.1.5 Router 31

5.2 Vue toteutus 32
5.2.1 projektin luominen 32
5.2.2 Syntaksi 33
5.2.3 Komponentit 35
5.2.4 Tilanhallinta 36
5.2.5 Router 38

6 Vertailu 40
6.1 Vue ja React versus AngularJS 40
6.1.1 Syntaksi 40
6.1.2 Komponentit 40
6.1.3 Tilanhallinta 42
6.1.4 Router 42
6.2 Vuen hyvät ja huonot puolet 43
6.2.1 Vuen hyvät puolet 43
6.2.2 Vuen huonot puolet 43
6.3 Reactin hyvät ja huonot puolet 44
6.3.1 Reactin hyvät puolet 44
6.3.2 Reactin huonot puolet 45

7 Yhteenveto 46

Lähteet 47

Kuvat

Kuva 1 CDR Selaus 11
Kuva 2 uusi selausnäkymä react 24
Kuva 3 create-react-app 25
Kuva 4 react syntaksi 26
Kuva 5 react komponentit 26
Kuva 6 react sortingForm 27
Kuva 7 redux store 28

Kuva 8 reducer 29
Kuva 9 useState ja Redux 30
Kuva 10 react router 31
Kuva 11 uusi cdrselausnäkymä vue 32
Kuva 12 vue CLI 32
Kuva 13 syntaksi 33
Kuva 14 reaktiivisuus & komponentin lisääminen 34
Kuva 15 luodut komponentit 35
Kuva 16 SortingForm komponentti 35
Kuva 17 CustomSelect komponentti 36
Kuva 18 SortingFormStore 37
Kuva 19 storen käyttöönotto 37
Kuva 20 Vue Router 38
Kuva 21 Router käyttöönotto 39

Käytetyt lyhenteet tai sanasto

Lyhenne	Lyhenteen selitys (Lähdeviite)
CDR	Call Detail Record, on yksityiskohtainen tietue puhelun tiedoista. (Technopedia. n.d)
MVC-arkkitehtuuri	Model-view-controller, on ohjelmistosuunnittelussa käytetty malli, joka korostaa ohjelmiston liiketoimintalogiikan ja näytön erottamista toisistaan.

1 Johdanto

Modernit web-kehykset kehittyvät jatkuvasti, mikä tarkoittaa, että vanhat teknologiat saavat siirtyä uusien tieltä. Tämä lisää painetta uudistaa pitkän elinkaaren omaavia sovelluksia, sillä vanhentuneet teknologiat eivät pysty enää vastaamaan kasvaviin vaatimuksiin ja niillä kehittämisestä tulee hitaampaa. Vanhojen teknologioiden parissa työskennellessä tulee lisäksi olla tietoinen siitä, että mikäli teknologian tuki lakkautetaan kokonaan, ei mahdollisia haavoittuvuuksia enää korjata, jolloin sovellukset voivat jäädä haavoittuviksi. (Penttinen, N. 2022)

Tässä opinnäytetyössä käsiteltävä järjestelmä on osa teleliikennetietojen keräämiseen ja analysointiin kehitettyä sovelluskokonaisuutta, joka on kehitetty jo vanhentunutta AngularJS sovelluskehystä käyttämällä. Nykyinen ratkaisu on tullut tiensä päähän, jonka vuoksi sovelluskehyksen päivittäminen uudempaan on tullut ajankohtaiseksi. Kuten Nino Penttisen työstä ” Frontend-teknologiamigraatio AngularJS:stä Reactiin” käy ilmi, sovelluskehysmigraatio on lähes aina pitkä ja työläs prosessi, joka vaatii selvitystyötä ja suunnitelmallisuutta, jotta nykyinen liiketoimintalogiikka saadaan parhaalla mahdollisella tavalla siirrettyä, ilman ylimääräisiä kustannuksia. Tämä oli myös juurisyynä sille, että tässä opinnäytetyössä tehtävä esiselvitys haluttiin toteuttaa.

Yleisesti sovelluskehyksellä tarkoitetaan eräänlaista sovellusrunkoa, jonka päälle kehitettävät ohjelmat rakennetaan. Ne voivat sisältää sovelluskirjastoja, erilaisia funktioita ja objekteja sekä uudelleenkäytettäviä koodikokonaisuuksia, jotka on kirjoitettu sovelluskehyksen yleisiä toimintoja silmällä pitäen. Sovelluskehyksellä voidaan myös viitata sovelluskirjastoon. Näiden kahden merkittävän ero on siinä, että sovelluskehys pyrkii tarjoamaan kaiken täydellisen sovelluksen kehittämiseen, kun sovelluskirjasto keskittyy usein vain tiettyyn osaan sovelluksesta. (Kemppainen, H. 2022)

Tähän opinnäytetyöhön valittiin kaksi erityyppistä sovelluskehystä. React on yksi tämän päivän käytetyimmistä sovelluskehyksistä ja sijoittunut tutkimuksissa

kärkisijoille. Se on myös vahvasti suosiossa organisaation sisällä. Toiseksi sovelluskehykseksi valikoitui Vue, joka on viime vuosina noussut haastamaan Angularia. Angular suljettiin tutkimuksen ulkopuolelle, sillä sen suosio on ollut pitkään laskusuunnassa, joten sovellusmigraatiosta saatavan hyödyn ei nähdä konkretisoituvan. (StackOverflow, 2022)

Tässä opinnäytetyössä tehtävä selvitys on osa laajempaa käyttöliittymäuudistusprojektia, ja sen tarkoituksena on kartoittaa ja analysoida modernien sovelluskehysten toimivuutta AngularJS sovelluskehyksen korvaajana.

1.1 Tutkimuksen rajaus ja rakenne

Tämä opinnäytetyö tarkastelee valittuja sovelluskehyksiä kahdesta eri näkökulmasta, jakaen sen kolmeen osaan. Ensimmäisessä vaiheessa teknologioihin perehdytään teorian kautta, jossa syvennytään sovelluskehysten pääominaisuuksiin ja toimintaperiaatteisiin. Sovelluskehyksen ymmärtäminen on tärkeää, jotta sen soveltuvuutta nykyisen järjestelmän toimintalogiikkaan voidaan analysoida ja toisaalta, että ymmärrys sovelluskehyksen toimintalogiikasta on olemassa, jotta sitä osataan hyödyntää parhaalla mahdollisella tavalla.

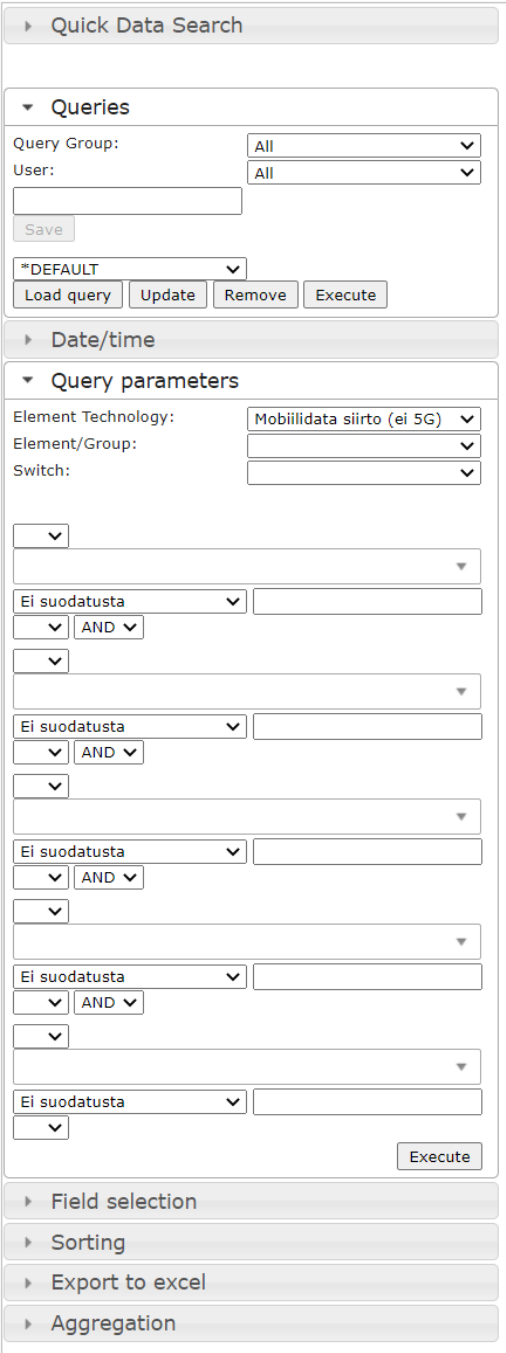
Toisessa osassa sovelluskehyksiä lähestytään käytännön kautta. Tässä opinnäytetyössä tämä toteutetaan niin, että kumpaakin teknologiaa hyödyntämällä luodaan CDR selausnäkymä nykyisestä järjestelmästä. Käytännön toteutuksen tarkoituksena on saada syvällisempi käsitys sovelluskehysten toimintalogiikasta ja luoda käsitys projektin rakenteesta ja luettavuudesta.

Kolmannessa osassa sovelluskehyksiä vertaillaan olemassa olevaan AngularJS:ään ja toisaalta perehdytään tarkemmin valittujen sovelluskehysten hyviin ja huonoihin puoliin. Tämän kappaleen tarkoituksena on saavuttaa lopullinen ymmärrys siitä, mihin sovelluskehykset kykenevät ja mitkä ovat mahdolliset esteet niiden hyödyntämiselle.

Koska kyseessä on järjestelmä, joka koostuu useista sovelluskokonaisuuksista, ja tämän opinnäytetyön on tarkoitus toimia esiselvityksenä mahdolliselle teknologiavalinnalle, on tekninen toteutus rajattu niin, että se koskee yhtä AngularJS teknologialla toteutettua järjestelmän osaa. Selvityksen tarkoituksena ei ole muuttaa olemassa olevan käyttöliittymän toimintalogiikkaa tai niihin kytköksissä olevia taustajärjestelmiä, eikä integraatiota taustajärjestelmiin tulla tässä vaiheessa toteuttamaan.

2 Nykytilanne

Kuvassa 1 on näkymä nykyisestä CDR selaus käyttöliittymästä. Se on rakennettu CDR tietueiden hakemiseen ja näyttämiseen.



Kuva 1 CDR Selaus

2.1 Käyttöliittymä

Nykyinen käyttöliittymä jakautuu kahteen osaan, jossa Quick Search osuus mahdollistaa nopean CDR selauksen, kun taas Advanced Search mahdollistaa monimutkaisempien hakujen tekemisen annettujen kriteerien mukaisesti.

Advanced Search käyttöliittymä muodostuu seitsemästä hakua tarkentavasta osiosta, joiden kaikkien valinnat huomioidaan lopullisessa backend kutsussa, vaikkei käyttäjä olisi tehnyt muutoksia kuin yhteen osioon. Tämä tekee käyttöliittymästä hyvin teknisen, eli käyttäjän tulee tietää tarkasti mitä on hakemassa, jotta saa vastauksena haluamansa CDR tietueet.

Date/time osion valinnat mahdollistavat haun rajauksen tiettyyn ajanhetkeen. Nykyisellään varsinaisina rajauksia sallitulle haun aikavälille ei käyttöliittymän puolella ole.

Query parameters valinnat määrittelevät haettavan CDR tietueen elementin tai elementtiryhmän, ja hakua voidaan rajata

elementtikohtaisin and ja or ehdoin. Näistä muodostuu backend kutsua rajaavia SQL hakuehtoja.

Field selection osiossa hakua voidaan rajata koskemaan elementtikohtaisesti valittuja kenttiä, kun taas Sorting puolestaan mahdollistaa haun suodattamisen elementtikohtaisten kenttien mukaan.

Käyttöliittymän Export to excel toiminnallisuus mahdollistaa valikoitujen rivien viemisen Exceliin jatkokäsittelyä varten.

Aggregation on toiminnallisuus, joka antaa mahdollisuuden kiinteän verkon elementtien liikennetietojen syykoodien luokitteluun ja analysointiin.

2.2 Haasteet

Vaikka sovellus itsessään on toimiva, sen ongelmaksi ylläpidon näkökulmasta muodostuu sen vesiputousmaisuus. Kun yksi muutos aiheuttaa muutoksia useammassa paikassa, tällöin muutoksen hallittavuus särkyy. Jos kokonaisuus ei ole ehyt, sovellus ei toimi ja toisaalta taas pienikin muutos voi aiheuttaa epätoivotun ketjureaktion, jossa aiheutetaan erilaista toimintaa kuin on alun perin ajateltu. Tämä vaikeuttaa uusien ratkaisujen toteuttamista, sillä pienikin muutos voi olla useiden päivien työ, jotta voidaan varmistua siitä, että halutun muutoksen lisäksi vanhat toiminnallisuudet toimivat kuten pitääkin.

Haastetta lisää sovelluksen rakenne, jossa komponentteja ei ole pilkottu tarpeeksi pieniin osiin, niin että niiden muokkaaminen toisistaan irrallaan tai useamman kehittäjän toimesta onnistuisi samanaikaisesti. Tämä tarkoittaa myös sitä, että nykyisellään kaikki testaaminen on manuaalista työtä, sillä testejä ei pysty luotettavasti tekemään osio kerrallaan.

3 React.js

React.js-kehys on Facebookin kehittämä avoimen lähdekoodin JavaScript-kehys ja on yksi suosituimpia käyttöliittymäsuunnittelussa käytettyjä kirjastoja. Se sai alkunsa vuonna 2011 kun ohjelmistokehittäjä Jordan Walke pyrki ratkaisemaan haasteen, jossa lopputuloksena piti olla dynaaminen, nopea ja reagoivampi käyttöliittymä, suorituskykyä unohtamatta. React yksinkertaisti tätä kehitysprosessia tarjoamalla jäsennellymmän tavan rakentaa dynaamisia ja interaktiivisia käyttöliittymiä uudelleen käytettävistä komponenteista. (Herbert, D. 2022)

Reactin ensisijainen rooli sovelluksessa on käsitellä sovelluksen näkymäkerrosta. Se koostuu kaikista niistä toiminnoista, jotka ovat suoraan vuorovaikutuksessa käyttäjän kanssa. Sen sijaan, että koko käyttöliittymää käsiteltäisiin yhtenä kokonaisuutena, React.js pyrkii erottamaan monimutkaiset käyttöliittymät yksittäisiksi uudelleenkäytettäviksi komponenteiksi. Tällä tavoin se mahdollistaa JavaScriptin nopeuden ja tehokkuuden hyödyntämisen yhdessä uuden tehokkaamman DOM-manipulointimenetelmän kanssa, jolloin lopputuloksena saadaan dynaamisia ja interaktiivisia verkkosovelluksia.

ReactJS on sovelluskehyksenä keskittynyt siis puhtaasti webelementtien tulkintaan, jonka vuoksi ReactJS sovellus vaatii lisäkirjastoja, joiden avulla voidaan luoda edellytykset tärkeille toiminnallisuuksille, kuten web-sivujen reititykselle ja tilanhallinnalle. (Herbert, D. 2022)

3.1.1 CLI

Uusi projekti on helppoa luoda käyttämällä Reactin CLI (Command Line Interface) työkalua Create React App. Se ei vaadi erillistä asentamista, mutta käytettävä komento riippuu siitä, minkä pakettimanagerin valitsee. Itsessään se ei käsittele taustalogiikkaa tai tietokantoja; se vain luo käyttöliittymän rakennusprosessin, joten sitä voidaan käyttää minkä tahansa taustajärjestelmän

kanssa. Komennon mukana asentuu kehitysympäristö sekä automaattisesti konfiguroidut versiot babelista ja webpackista. (React. n.d)

Webpack ja Babel ovat työkaluja, jotka mahdollistavat JavaScript sovelluksien optimoinnin. Webpack niputtaa kaikki käytetyt moduulit, ja tuottaa niistä yhden tiedoston. Tämä tapa pienentää JavaScript tiedostojen määrää, mikä taas lisää yleistä tehokkuutta. Babel puolestaan on syntaksin muunnin. Se kääntää uusimman JavaScriptin(ES6+) vanhemmaksi JavaScript koodiksi, mikä tekee koodista yhteensopivan kaikkien selaimien kanssa, riippumatta käytetystä JavaScript versiosta. (Dissanayake, N. 2022)

3.1.2 Komponentit

React on tehokas ja joustava JavaScript-kirjasto, jolla voidaan rakentaa monimutkaisia käyttöliittymiä pienistä, eristetyistä komponenteista. Komponentit sallivat käyttöliittymän jakamisen itsenäisiin, uudelleenkäytettäviin palasiin, jossa niitä voidaan tarkastella omina, eristyksissä olevina kokonaisuuksinaan. (React. n.d)

Toiminnalliset komponentit ovat yleisimpiä komponentteja, joita Reactissa käytetään ja ne ovat yksinkertaisimmillaan JavaScript-funktioita. Nämä toiminnot voivat vastaanottaa tai olla vastaanottamatta dataa parametreina ja niiltä puuttuu huomattava määrä ominaisuuksia verrattuna luokkapohjaisiin komponentteihin. Toiminnallisissa komponenteissa palautusarvo on JSX-koodi, joka renderöidään DOM-puuhun.

Luokkapohjaiset komponentit ovat yksinkertaisia luokkia, jotka koostuvat useista funktioista, jotka lisäävät toimintoja sovellukseen. Kaikki luokkapohjaiset komponentit ovat ReactJS:n komponenttiluokan lapsiluokkia. Luokkapohjaisten komponenttien pääominaisuus, joka erottaa ne toiminnallisista komponenteista, on se, että niillä on pääsy elinkaaren toimintoihin, kuten componentWillMount(), componentDidMount(), componentWillReceiveProps(), componentWillUpdate(), shouldComponentUpdate(), render() ja komponentWillUnmount(); Näitä

elinkaarifunktioita kutsutaan elinkaaren eri vaiheissa, ja niitä käytetään moniin tarkoituksiin, kuten tilan muuttamiseen. (GeeksForGeeks. 2021.)

Toiminnallisten ja luokkapohjaisten komponenttien välistä eroavaisuutta on pyritty pienentämään ReactJS-konseptilla nimeltä "koukut "(eng. hooks). Koukut ovat erikoistoimintoja, jotka mahdollistavat ReactJS-ominaisuuksien käytön toiminnallisissa komponenteissa. ReactJS:llä on pääsy erityiseen koukkuun nimeltä useState(), jonka avulla voidaan luoda illuusio tilan kanssa työskentelystä toiminnallisissa komponenteissa. UseState()-funktiota käytetään vain yhden tilamuuttujan alustamiseen, joten useamman muuttujan tapauksessa tarvitaan useita useState()-määrityksiä. Ensimmäinen palautettu arvo on tilamuuttujan alkuarvo, kun taas toinen palautettu arvo on viittaus funktioon, joka päivittää sen. Kun tilamuuttuja on päivitettävä, se voidaan tehdä kutsumalla päivitystoimintoa ja päivittämällä tilamuuttuja suoraan sen sisällä.

Toiminnallisilla komponenteilla ei ole pääsyä elinkaaritoimintoihin, kuten luokkapohjaisilla komponenteilla, koska elinkaarifunktiot on määritettävä luokan rajojen sisällä. Jos elinkaarifunktioita on käytettävä toiminnallisten komponenttien kanssa, on käytettävä erityistä React-koukkua nimeltä useEffect(). On syytä huomata, että useEffect() ei ole tarkka kopio elinkaarifunktioista – se toimii ja käyttäytyy hieman eri tavalla. Sen voidaan ajatella olevan yhdistelmä componentDidMount(), componentDidUpdate() ja componentnWillUnmount() metodeita. (GeeksForGeeks. 2021.; Kagga, J. 2018. ;React. n.d)

3.1.3 Syntaksi & JSX

ReactJS käyttää HTML syntaksin sijaan JSX:ää. JSX syntaksia voidaan kirjoittaa suoraan Javascriptin sekaan toisinkuin HTML: ää, mutta tällainen Javascript pitää kuitenkin kääntää erillisellä kääntäjällä. Tällaisia ovat esimerkiksi Babel, jonka päätehtävä on kääntää modernit Javascript versiot sellaiseen muotoon, että selaimet osaavat niitä käyttää. HTML ja JSX poikkeavat formaatiltaan toisistaan jonkin verran, muun muassa attribuuttien

nimien osalta, joka estää HTML-tiedoston toiminnan sellaisenaan JSX formaatissa. (React. n.d; Javapoint. n.d)

3.1.4 DOM

Document object model eli DOM on selaimen websivusta luoma malli, jonka pohjimmaisena tarkoituksena on toimia rajapintana websivuille. Se on objektipohjainen kuvaus alkuperäisestä HTML-dokumentista eli DOMin tehtävänä on muuttaa HTML dokumentti objektimalliin, jolloin sitä voidaan käyttää ohjelmien kanssa. DOM sallii ohjelmien lukea ja muuttaa sivun sisältöä, tyyliä ja rakennetta, joten se ei aina ole täysin sama kuin sen alkuperäisenä lähteenä ollut HTML dokumentti. (KS, A. 2022; Javapoint. n.d)

3.1.5 Tilanhallinta

React komponenteilla on sisäänrakennettu tilaobjekti, joka pitää sisällään sellaista dataa, joka on pysyvää komponenttien renderöinnin välillä. Tämä tila muuttuu vuorovaikutuksessa sovelluksen kanssa, jolloin käyttöliittymä näyttää erilaiselta, koska se edustaa uutta tilaa vanhan sijaan.

React komponentit hallitsevat tilaansa sisäisesti, mutta tämä komponenttien kesken jaettujen tilojen hallinnan monimutkaisuus korostuu mitä isommasta sovelluksesta on kyse. Jos useampi tila muuttuu samaan aikaan, on vaikea löytää missä kohtaa asiat menevät pieleen. Tämän vuoksi sovelluksessa tulisi olla erillinen tilanhallinta. (Gupta, V. n.d.)

React-Redux on Redux kirjastoa hyödyntävä tilanhallintajärjestelmä. Se on eräänlainen keskusmuisti, joka sisältää sovelluksen kaikki tilat. Komponentit pääsevät kiinni näihin tiloihin ilman, että niitä tarvitsee lähettää komponentilta toiselle. Se rakentuu kolmesta palasesta, jotka ovat actions, reducer ja store. (Gupta, V. n.d.)

- Actions eli toiminnot lähettävät dataa sovelluksesta storeen komennolla store.dispatch()
- Reducer määrittää, kuinka sovelluksen tila muuttuu suhteessa storeen lähetettyihin toimintoihin.
- Store säilyttää sovelluksen tilan ja sitä voidaan käyttää, päivittää ja hallinnoida sen kuuntelijoita erilaisin apumenetelmin.

3.1.6 Router

React Router on vakio kirjasto, jota käytetään sivujen reitittämiseen sovelluksessa. Se mahdollistaa navigoinnin sovelluksen eri komponenttinäkymien välillä, samalla huolehtien siitä, että käyttöliittymä pysyy synkronoituna URL osoitteen kanssa. (Geeksforgeeks. n.d)

React Router pääkomponentit ovat

- BrowserRouter on toteutus, joka käyttää HTML5 historyAPI ratkaisuja, joilla pidetään käyttöliittymä synkronoituna URL-osoitteen kanssa.
- Switch mahdollistaa reitityksen parhaan osuman mukaan, ilman että kuljettaisiin järjestelmällisesti osoitteesta toiseen.
- Route eli reitti on näytettävä komponentti, jonka polku vastaa URL osoitetta
- Link komponenteilla luodaan linkkejä eri reiteille ja näiden avulla toteutetaan navigointi sovelluksen ympärille. Link komponentti vastaa HTML-ankkuritunnistetta.

4 Vue.js

Evan You kehitti Vuen sivuprojektina muiden töidensä ohella. Se sai alkunsa, kun AngularJS alkoi tuntua käytössä turhan raskaalta kaikkine ominaisuuksineen, vaikka se muutoin sisälsi toimiviakin ratkaisuja, kuten DOM käsittelyn. You alkoi kehittää omaa kirjastoaan siten, että hän sisällytti omaansa palasia Angularista ja jätti pois niitä ominaisuuksia, joista ei pitänyt. Myöhempiin versioihin palasia on otettu myös Reactista. Ensimmäinen Vue versio julkaistiin vuonna 2014. (Cromwell. 2017)

Vue on avoimen lähdekoodin progressiivinen kehys, joka on suunniteltu asteittain käyttöönotettavaksi, koska ydinkirjasto on keskittynyt vain näkymäkerroksen ympärille. VueJS:ää käytetään ensisijaisesti verkkorajapintojen ja yksisivuisten sovellusten rakentamiseen. Vue tekee tämän käyttämällä perinteistä MVC-arkkitehtuuria tarkastellakseen sovelluksen tai verkkosivuston käyttöliittymää käyttämällä sen ydinkirjastoa oletusnäkymäkerroksena. Se on joustava siinä mielessä, että se voi toimia myös komponenttipohjaisen arkkitehtuurin kanssa – aivan kuten React. Yksinkertaisesti sanottuna se on näkymiin keskittynyt kirjasto. Näkymä on olennaisin osa kaikkea järjestelmän sisällä tapahtuvaa, ja kaikki tiedot validoidaan vain, jos se on vuorovaikutuksessa näkymien kanssa oikein. Vue mahdollistaa koodin osien muokkaamisen pitämällä tiedot tieto-objektissa, kun päivitykset vaikuttavat muihin linkitettyihin osiin reaaliajassa. (Winter, R. 2022)

4.1.1 CLI

Vue CLI työkalua voidaan käyttää komentokehotteen kautta tai erillisen graafisen Vue UI käyttöliittymän kautta. Työkaluina molemmat tarjoavat samat toiminnallisuudet. Uusia käyttäjiä ohjataan käyttämään vue CLIn sijaan create-vue:ta. Vue CLI perustuu webpackiin, kun taas create-vue pohjautuu Viteen. Vite tukee useimpia Vue CLI -projekteista löydetyistä konfiguroiduista käytännöistä heti valmiina ja tarjoaa huomattavasti paremman

kehityskokemuksen erittäin nopean käynnistyksensä ja moduulien vaihtonopeuden ansiosta. Toisin kuin Vue CLI, itsessään create-vue on vain rakennusteline: se luo esikonfiguroidun projektipohjan valituille ominaisuuksille ja delegoi loput Vitelle. Tällä tavalla rakennetut projektit voivat hyödyntää suoraan Vite-laajennusten ekosysteemiä, joka on Rollup-yhteensopiva. (Create vue. n.d)

4.1.2 Komponentit

Komponenttien avulla voimme jakaa käyttöliittymän itsenäisiin ja uudelleenkäytettäviin osiin ja ajatella jokaista osaa erikseen. Vue toteuttaa oman komponenttimallinsa, jonka avulla mukautettu sisältö ja logiikka voidaan kapseloida jokaiseen komponenttiin. Vue toimii hienosti natiiviverkkokomponenttien kanssa. Komponentteja voidaan käyttää uudelleen niin monta kertaa kuin halutaan tai niitä voidaan käyttää toisessa komponentissa, mikä tekee siitä alikomponentin. (Vue.js. n.d)

Globaalit komponentit on rekisteröity sovelluksen laajuisesti, ja niitä voidaan käyttää missä tahansa ilman, että niitä tarvitsee viedä tai tuoda tiedostoon, johon ne sisältyvät. Kun taas lokaalit komponentit luodaan lokaalisti, ja niitä voidaan käyttää vain siellä missä ne on luotu.

Kun luodaan suuria verkkosovelluksia, joissa on erilaisia toimintoja, on parasta jakaa sovellus yksittäisiin tiedostokomponentteihin, eli Single-File-komponentteihin (SFC). Tämä .vue päätteinen tiedostomuoto on erityinen siitä, että se auttaa kapseloimaan komponentin mallin, logiikan ja tyylin yhdeksi tiedostoksi.

Jokainen .vue-tiedosto koostuu kolmen tyyppisistä ylemmän tason kielilohkoista: <template>, <script> ja <style> sekä vaihtoehtoisesti muista lohkoista. Muita lohkoja voidaan sisällyttää *.vue-tiedostoon projektikohtaisia tarpeita varten, kuten esimerkiksi <docs>-lohko.

Jokainen *.vue-tiedosto voi sisältää enintään yhden ylätason <template>-lohkon kerrallaan, jonka sisältö puretaan ja välitetään @vue/compiler-domiin, jossa ne esikäännetään JavaScript-renderöintifunktioiksi ja liitetään vietyyn komponenttiin sen renderöintivaihtoehtona. (Vue.js. n.d)

Tämän lisäksi *.vue-tiedosto ei voi myöskään sisältää kuin enintään yhden <script>-lohkon kerrallaan (pois lukien <script setup>). Skriptit suoritetaan aina ECMAScript (lyhyesti ESM) -moduulina, joka on JavaScript-standardi, jonka tarkoituksena on kapseloida JavaScript-koodit uudelleenkäyttöä varten. (Roy, G. 2022)

<style>-tageja puolestaan voi olla useampi, ja niitä voidaan määrittää scoped ja module attribuutein, joilla voidaan helpottaa tyylin kapselointia sen hetkisen komponentin ympärille. (Vue.js. n.d)

4.1.3 Syntaksi

Vue käyttää HTML-pohjaista template syntaksia, jonka avulla hahmotettu DOM sidotaan deklaratiivisesti taustalla ilmenevien komponenttien tietoihin. Kaikki Vue templaatit ovat syntaksiltaan kelvollisia HTML-koodeja, joten ne voidaan jäsentää niille tarkoitetuin selaimin ja jäsentimin. Vue kokoaa mallit optimoiduksi Javascript-koodiksi, joka yhdessä reaktiivisen järjestelmän kanssa selvittää renderöitävien komponenttien vähimmäismäärän ja käyttää mahdollisimman vähän DOM manipulaatiota sovelluksen tilan muuttuessa. Vuen kanssa on mahdollista hyödyntää JSX syntaksia, mutta sillä ei ole samanlaista käännösajan optimointia kuin templaateilla. (Vue.js. n.d)

Vue sisältää sisäänrakennettuja direktiivejä, joita käytetään templaten sisällä olevassa HTML-koodissa. Ne ovat kuin HTML-attribuutteja ja ne alkavat aina v-alkuliitteellä. Direktiivien avulla voidaan kertoa ohjelmalle, mitä DOM-elementille pitää tehdä, ilman että tarvitsee kirjoittaa monimutkaista koodia asian ilmaisemiseksi. (Copes 2018)

Vue sisältää useita sisäänrakennettuja direktiivejä. Tällaisia ovat muun muassa (Vue.js. n.d):

- v-text, jota käytetään tuplakaarisulkeiden sijaan, ja se korvaa elementin sisällä olevan sisällön. Tuplakaarisulkeita tulee käyttää silloin, kun vain osaa tekstinsisällöstä halutaan päivittää.
- v-html sisältö lisätään tavallisessa HTML muodossa, eikä sen Vue-template syntaksia käsitellä.
- v-show direktiivin avulla voidaan määritellä, milloin elementti näytetään ja milloin ei.
- v-if, v-else ja v-else-if käytetään kuten ehtolauseita ja kun ehto muuttuu, elementti ja sen sisältämät direktiivit tai komponentit tuhotaan ja luodaan uudelleen.
- v-for direktiiviä käytetään silloin kun halutaan tulostaa elementti tai template block useita kertoja lähdetietojen perusteella, esimerkkinä listan sisällön tulostaminen.
- v-on lisää elementille event listenerin, joka kuuntelee tapahtumia, esimerkiksi kun syöttökentän sisältö muuttuu tai kun elementtiä klikataan
- v-bind sitoo dynaamisesti yhden tai useamman attribuutin. Kun attribuutit sidotaan keskenään, Vue tarkistaa oletusarvoisesti onko elementin avain määritelty ominaisuudeksi ja jos on, Vue asettaa arvon DOM-ominaisuutena attribuutin sijaan.
- v-model luo kahdensuuntaisen sidoksen lomakkeen syöttöelementille tai komponentille, jolloin komponentin tila muuttuu automaattisesti, kun syötetty arvo muuttuu.
- v-once renderöi elementin ja komponentin vain kertaalleen. Seuraavalla renderöintikerralla niitä käsitellään staattisena sisältönä.
- v-memo toimii kuten v-once mutta sillä poikkeuksella, että renderöinti skipataan vain silloin kun muistissa oleva renderöinti vastaa uutta renderöintiä, missä mikään arvo ei ole muuttunut.

4.1.4 DOM

Vuen DOM on vastaava kuin Reactissa, joka oli DOM manipulaation edelläkävijä. Virtuaalinen DOM on kuitenkin enemmän malli kuin tietty tekniikka, joten se voidaan toteuttaa monella erillä tapaa. (Vue.js. n.d)

Vue jakaa Virtuaalisen DOMin kolmeen osaan

- Compile vaihe voidaan tehdä joko etukäteen tai lennossa käyttämällä ajon aikaista piirturia. Tässä vaiheessa kaikki Vue-templatet käännetään renderöintifunktioiksi, jotka palauttavat niitä vastaavat virtuaaliset DOM-puut.
- Mount on ajon aikainen piirturi, joka voi kävellä virtuaalista DOM-puuta ja rakentaa siitä todellisen DOM-puun.
- Patch sovittaa tehdyt muutokset niin, että ajon aikainen piirturi pystyy vertaamaan eroja ja soveltamaan havaitut muutokset yhteen todellisen DOMin kanssa.

4.1.5 Tilanhallinta

Vue komponentti hallitsee omaa reaktiivista tilaansa. Se koostuu seuraavista osista (Vue.js. n.d)

- State eli tila, joka on sovelluksen todellinen sen hetkinen tila.
- View eli näkymä, joka on sovelluksen deklaratiivinen tila
- Actions eli toiminnot, jotka ovat tapoja, joilla tilaa voidaan muuttaa näkymistä saatuihin syötteisiin.

Mutta kuten Reactissa, ongelmia alkaa muodostua silloin, kun useampi komponentti jakaa samaa tilaa. Tämän vuoksi myös Vuessa tulee käyttää tilanhallintaan tarkoitettuja kirjastoja.

Pinia on Vuelle kehitetty kirjasto, jonka avulla voidaan jakaa tila komponenttien/sivujen kesken. Sen kehittäminen sai alkunsa vuonna 2019, kun

lähdettiin hahmottelemaan ratkaisua Vuessa jo olevan compositionAPI ratkaisun pohjalta. Pinia toimii sellaisenaan Vue2 ja Vue3 versioiden kanssa, eikä se vaadi compositionAPIn käyttöä toimiakseen. Verrattuna Vuexiin, joka on toinen hyvin käytetty tilanhallinnan kirjasto, Pinia tarjoaa yksinkertaisemman rajapinnan, joka tukee suoraan TypeScriptiä. (Pinia. n.d.)

4.1.6 Router

Vue Router auttaa yhdistämään selaimen URL-osoitteen/historian ja Vuen komponentit, sallien tietyt polut renderöidä kaikki siihen liittyvät näkymät. (Maribojoc, M. 2020)

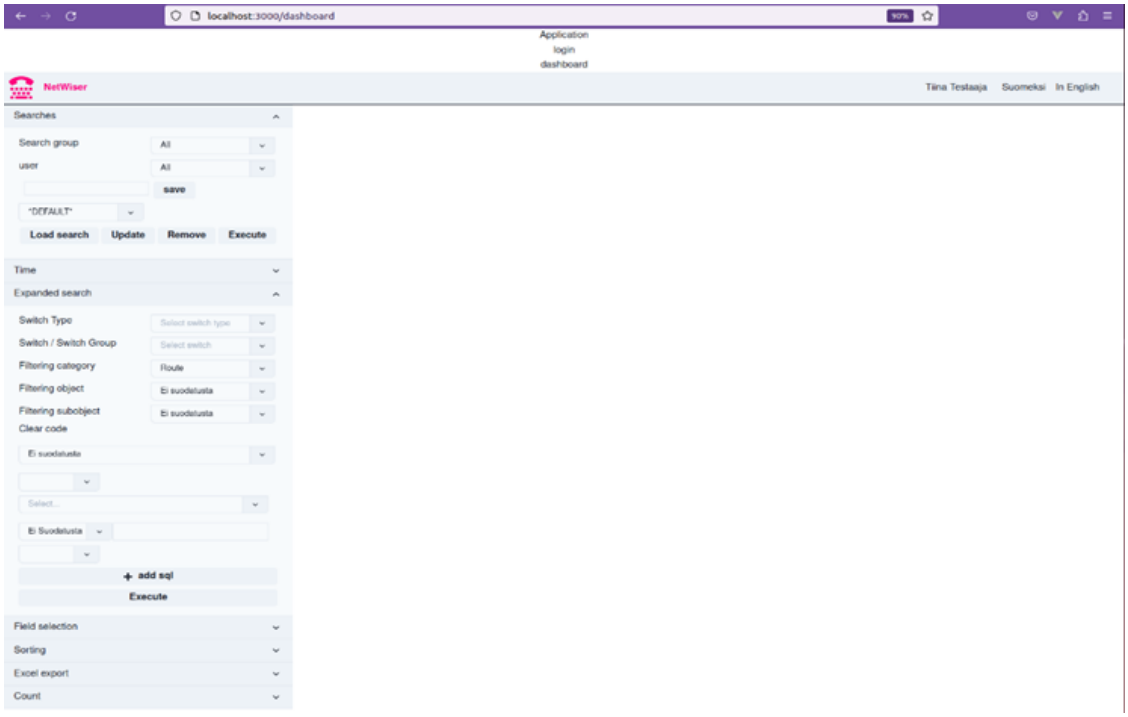
Vue Router pääkomponentit ovat (Vue Router. n.d)

- Router-link mahdollistaa URL-osoitteen muuttamisen ilman, että sivua ladataan uudelleen, sekä hoitaa URL-osoitteen luomisen ja sen koodauksen.
- Router-view näyttää URL-osoitetta vastaavan osan.

5 CDR selaus näkymän uudistaminen

Tässä kappaleessa rakennetaan CDR selaus näkymä hyödyntäen Vue ja React teknologioita.

5.1 React toteutus



Kuva 2 uusi selausnäkymä react

Kuva 2 esittää lopputulosta, joka saatiin aikaiseksi React projektin aikana. React projektissa tarkoitus oli toteuttaa käyttöliittymä ilman backendiä ja toteuttaa muutamia teoriaosuudessa käsiteltyjä asioita, kuten tilanhallinta ja reititys. Tässä kappaleessa on tarkoitus käydä teoria lyhyesti käytännössä.

5.1.1 Projektin luominen

```
tintti@osboxes:~$ yarn create react-app nw-react-app
yarn create v1.22.19
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

success Installed "create-react-app@5.0.1" with binaries:
      - create-react-app

Creating a new React app in /home/tintti/nw-react-app.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
```

Kuva 3 create-react-app

React sovellus luodaan komennolla yarn create react-app, jonka perässä on haluttu sovelluksen nimi. Kuten kuvassa 3, nw-react-app on luotu komennolla *yarn create react-app nw-react-app*. Create react-app on CLI työkalu, jonka mukana tulevat kaikki rakennustyökalut ja tiedostot, jotka ovat valmiiksi määritelty ja valmiita käyttöön. Create react app ei asenna automaattisesti router sovelluskirjastoa tai tilanhallinnassa käytettävää reduxia. Nämä on asennettava erikseen omilla komennoillaan. React routerin asennus onnistuu komennolla. *yarn add react-router-dom*, joka lisää react-router-dom kirjaston mukaan pakettiin. Redux asennettaan komennolla *yarn add @reduxjs/toolkit react-redux*

5.1.2 syntaksi

JSX on Javascriptin syntakstilaajennus, joka tuottaa Reactin elementtejä. React itsessään ei vaadi sen käyttöä, mutta se on havaittu toimivan hyvänä visuaalisena työkaluna, kun työskennellään käyttöliittymän kanssa Javascript koodin sisällä. JSX syntaksin avulla React pystyy käsittelemään koodia paremmin ja luomaan näin ollen hyödyllisemmät virheilmoitukset ongelmatilanteissa. React ei erottele teknologioita toisistaan, niin kuin esimerkiksi Angularissa tehdään, vaan hyödyntää löyhästi kytkettyjä komponentteja, jotka sisältävät sekä logiikan että merkinnät. Koska JSX on lähempänä JavaScriptiä kuin HTML:ää, React DOM käyttää camelCase-ominaisuuksien nimeämiskäytäntöä HTML-attribuuttien nimien sijaan. (React. n.d)

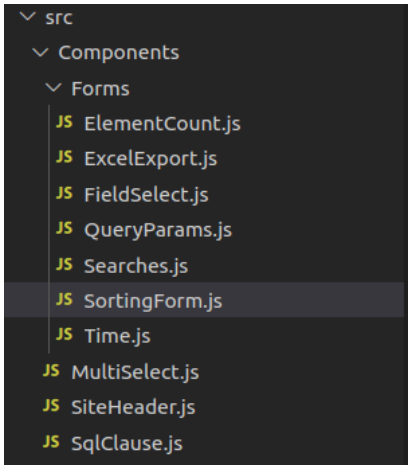
```
src > Pages > JS Dashboard.js > ...
1  import React from 'react';
2  import SiteHeader from '../Components/SiteHeader';
3  import FormView from './FormView';
4
5  export default function Dashboard() {
6    return(
7      <>
8
9      <SiteHeader />
10     <FormView />
11
12     </>
13   );
14 }
```

Kuva 4 react syntaksi

JSX merkintöjen kohdalla tulee muistaa seuraavat säännöt:

- Palauta aina yksi juurielementti. Juurielementti voi olla esimerkiksi .
    - Kuten kuvassa 4, kaksi tuotua komponenttia on kääritty yhden tyhjän <> tagin sisään, jota kutsutaan tässä tapauksessa juurielementiksi. Tyhjiä tagejä kutsutaan Fragmenteiksi, joilla voidaan ryhmitellä asioita, jättämättä kuitenkaan jälkeä selaimen HTML puuhun.
- Sulje kaikki tagit
    - kuvassa 4, <SiteHeader /> on suljettu lisäämällä kauttaviiva itsesulkeutuvan tagin merkiksi.
- Käytä CamelCase muotoa nimeämisessä
    - JSX kääntyy Javascriptiksi, jolloin attribuuttien nimistä muodostuu avaimia Javascript objekteille. Komponenteissa nämä attribuutit luetaan usein muuttujiksi, mutta Javascript asettaa rajoituksia muuttujien nimiin. Tämän vuoksi Reactissa on suositeltavaa käyttää camelCase kirjaimia. Kuten kuvan 4 SiteHeader, jossa Header on kirjoitettu isolla alkukirjaimella.

(react, beta. n.d)



Kuva 5 react komponentit

5.1.3 Komponentit

React-sovelluksissa on kolme yleisintä nimeämiskäytäntöä:

1) Camel case tiedoston nimille ja pascal case komponentin nimille
2) Kebab case tiedoston nimille ja

pascal case komponentin nimille

3) Pascal case tiedoston ja komponentin nimille (Bado B, 2022)

Noudatettiinpa mitä tahansa nimeämiskäytäntöä, komponentin nimet ovat aina pascal case muotoisia, eli ne alkavat aina isolla alkukirjaimella. Komponenteille annettu nimi tulee olla selkeä ja ainutlaatuinen sovelluksessa, jotta ne olisi helpompi löytää ja jotta vältytään mahdollisilta sekaannuksilta. Samoin komponentit kannattaa järjestellä sovelluksessa niin, että komponentit on jaettu omiin osioihin, esimerkiksi kuvassa 5, kaikki lomakekomponentit löytyvät kansiosta Forms.

```
src > Components > Forms > JS SortingForm.js > SortingForm
1  import { FormControl, Button, FormLabel, Box } from '@chakra-ui/react';
2  import React from 'react'
3  import MultiSelect from "../MultiSelect";
4
5
6  function SortingForm() {
7
8
9      const options = [
10         { value: "duration[s] NET", label: "duration[s] NET"},
11         { value: "APN", label: "APN"},
12         { value: "Closure code", label: "Closure code"}
13       ];
14
15     return(
16         <>
17         <form>
18             <FormControl>
19             <Box pl="2" pr="2" >
20             <FormLabel w="200px" mt="2" htmlFor="sortFilter">Filter</FormLabel>
21             <MultiSelect
22                 options={options}
23                 width={"415px"}
24                 name={"sortFilter"}
25                 closeMenuOnSelect={false}
26                 size={"sm"}
27                 id="sortFilter"
28                 isMulti={true}
29             />
30             </Box>
31             </FormControl>
32             <Button w="415px" m="0.5" maxH={"30px"}>Execute</Button>
33         </form>
34         </>
35     )
36
37 }
38
39
40 export default SortingForm
```

Kuva 6 react sortingForm

SortingForm(kuva 6) on yksi monista komponenteista. Se koostuu importeista, luokkafunktiosta ja exportista. Importit ovat staattisia tuontiviittauksia, joita käytetään tuomaan lukumuotoisia live-sidoksia, jotka jokin toinen moduuli vie. Ominaispiirre näissä on, että sidoksen vienyt moduuli hoitaa päivittämisen, kun taas tuova moduuli ei voi määrittää niitä uudelleen. Jotta voidaan käyttää JSXn hyödyntämää React.createElement()-funktiota täytyy react tuoda reactista tuomalla se käyttäen Import React from ’react’. Tämä mahdollistaa sen, että jokainen JSX luo React-komponentin JSX-muuntajalla. Jos tuontia ei tehdä, jokainen kutsuttu React.createElement()- funktio on määrittelemätön.

Sovellukseen on valittu käyttöliittymäkirjastoksi chakra, josta tuodaan kirjastosta lomakkeelle siinä halutut elementit, kuten Button, Box, ja FormLabel. Nämä ovat chakran elementtejä, jotka tyylittelevät HTML-elementtejä. Box renderöi elementin, button <button> elementin ja FormLabel puolestaan on kokoava elementti, joka toimii HTML labelin tavoin, mutta asettaa automaattisesti htmlFor määrityksen lapsielementille. Kuvassa 6 lapsielementiksi on määritelty MultiSelect, joka on kustomoitu select komponentti, joka tuodaan omalla import lausekkeella. (chakra-ui. n.d)

SortingForm itsessään luo ainoastaan options listan, joka määritellään render funktiossa MultiSelect komponentin parametriksi. Reactin render funktion tarkoituksena on esittää sille määritelty HTML-koodi määriteltyjen HTML-elementtitunnisteiden sisällä. Export puolestaan huolehtii siitä, että komponenttia voidaan kutsua tiedoston ulkopuolelta.

5.1.4 Tilanhallinta

Reactissa tilanhallintaa voidaan lähestyä kahdella eri tapaa. React mahdollistaa komponenttien tilan seuraamisen useState koukun avulla, kun taas Redux on koko sovelluksen kattava store, joka säilyttää muuttujien tilan keskitetysti, jolloin tilaa voidaan käsitellä mistä päin sovellusta tahansa, komponenttihierarkiasta riippumatta.

Kuvassa 7 luodaan kaksi reduceria sisältävä store. ConfigureStore on createStore funktion johdannainen, joka lisää automaattisesti tärkeimmät perustoiminnallisuudet storen luomiseksi. ConfigureStore ottaa parametrikseen sliceReducer objektin, esim {counter: counterReducer, switczh: switczhReducer}, jolloin configureStore luo automaattisesti juuri

```
src > app > JS store.js > default
1  import { configureStore } from '@reduxjs/toolkit'
2  import counterReducer from '../features/counter/counterSlice'
3  import switczhReducer from '../features/queryparam/queryParamsSlice'
4
5  export default configureStore({
6    reducer: {counter: counterReducer, switczh: switczhReducer}
7  })
```

Kuva 7 redux store

reducerin välittämällä objektin combinedReducer toiminnallisuuteen, joka huolehtii siitä, että jokaista lapsi reduceria kutsutaan ja niistä luodaan yksi koko storen kattava reducer. (Redux-toolkit.js .n.d)

```
src > features > queryparam > queryParamsSlice.js > ...
1   import { createSlice } from '@reduxjs/toolkit'
2
3   export const switczhSlice = createSlice({
4     name: 'switczh',
5     initialState: {
6       value: ""
7     },
8     reducers: {
9       setSwitczh: (state, action) => {
10        state.value = action.payload
11        return state
12      }
13    }
14  })
15
16  export const { setSwitczh } = switczhSlice.actions
17
18  export default switczhSlice.reducer
```

Kuva 8 reducer

CreateSlice (kuva8) on funktio, joka hyväksyy alkutilan, reducer objektin ja Sliceobjektin nimen, ja luo automaattisesti toimintoja ja toimintatyyppejä, jotka vastaavat tilaa ja sen reducereita.

Name on String muotoinen nimi, joka annetaan tälle tilan osalle. Generoidut toiminnot käyttävät tätä nimeä prefixinä. InitialState puolestaan on tilakohtainen alkuarvo. Toisin sanoen tätä voidaan käyttää asettamaan tilalle jokin arvo, jota käytetään, kun alustetaan tilaa storesta saadulla arvolla. Kuten kuvassa 9 haettu switczh. Reducer on objekti, joka sisältää funktioita, jotka on tarkoitettu käsittelemään tiettyä toimintotyyppiä. Kuten kuvassa 8, reducer saa arvokseen funktion setSwitczh, joka puolestaan ottaa parametreikseen tilan ja actionin, hyödyntäen näitä tietoja päivittäessä storessa olevan muuttujan tilaa.

UseState() koukkua voidaan käyttää luomaan komponentin lokaali tila. Kuten kuvassa 9, tämä tehdään muodostamalla muuttuja, jossa switczh on tila ja changeSwitczh on funktio, jolla tilaa voidaan päivittää. UseState() funktiolle voidaan asettaa oletusarvo sulkeiden sisään. Switczh saa esimerkissä oletusarvoksi redux storesta haetun arvon selectedSwitch. Tilaa voidaan tämän jälkeen muuttaa kutsumalla asetusfunktiota changeSwitczh, joka muokkaa muuttujan tilaa komponentissa. Tulee kuitenkin huomata, että changeSwitczh ei päivitä kuin lokaalin tilan. Jos muutoksella halutaan päivittää myös storessa

olevan muuttujan tilaa, se tehdään kutsumalla storen päivitysfunktiota redux dispatch komennolla.

Kun tila haetaan redux storesta, käytetään siihen useSelector funktiota. Tämä funktio toimii valitsimena, jolla redux-storesta voidaan valita haluttu tila. useSelector(state => state.switczh.value) toimii siis siten, että funktio saa parametrikseen koko tilan, ja tästä tilasta haetaan switczh muuttujan arvo.

Storessa olevaa tilaa hallitaan dispatch metodilla, joka on ainoa tapa päivittää storessa olevan arvon tilaa. Päivittäminen tapahtuu kutsumalla dispatch komennon sisällä reducerissa määriteltyä toiminto-objektia. Eli kun selectedSwitch muuttujan arvoa halutaan päivittää, kutsutaan reducerissa(kuva8) olevaa setSwitczh dispatch komennon sisällä ja toiminto-objektille välitetään haluttu arvo. dispatch(setSwitczh(switczh)) päivittää näin ollen storen vastaamaan lokaalin muuttujan switczh arvoa.

```
src > Components > Forms > QueryParams.js > QueryParams > useEffect() callback > shortened
4   import {Select} from "chakra-react-select";
5
6   import { useSelector, useDispatch} from 'react-redux'
7   import MultiSelect from "../MultiSelect";
8   import SqlClause from "../SqlClause";
9   import {
10      setSwitczh
11    } from '../../features/queryparam/queryParamsSlice';
12
13  function QueryParams({typeOptions, switchOptions, filteringOptions, filteringObjOptions}) {
14
15      const selectedSwitch = useSelector(state => state.switczh.value)
16      const dispatch = useDispatch();
17      const [type, setType] = useState("DATA")
18      const [types, setTypes] = useState(typeOptions)
19      const [fillops, setFillops] = useState(filteringObjOptions)
20      const [switczhes, changeSwitches] = useState(switchOptions)
21      const [switczh, changeSwitch] = useState(selectedSwitch)
22
23      console.log("selected from store " + selectedSwitch.toString())
24
25      useEffect(() => {
26
27          if(type  === "" || type !== "DATA"){
28              changeSwitches(switchOptions)
29          } else {
30              const datatypes = [{value: "GGSN4", label: "GGSN4"}]
31              changeSwitches(datatypes)
32          }
33
34          console.log(switczh.toString())
35          if(switczh  !== "" && switczh !== "GGSN4") {
36              const updated = filteringObjOptions;
37              updated.push({value: "APN", label: "APN"})
38              setFillops(updated)
39          } else {
40              const shortened = fillops.splice(0,1)
41              setFillops(shortened)
42          }
43      }, [switczh, type]);
44
```

Kuva 9 useState ja Redux

5.1.5 Router

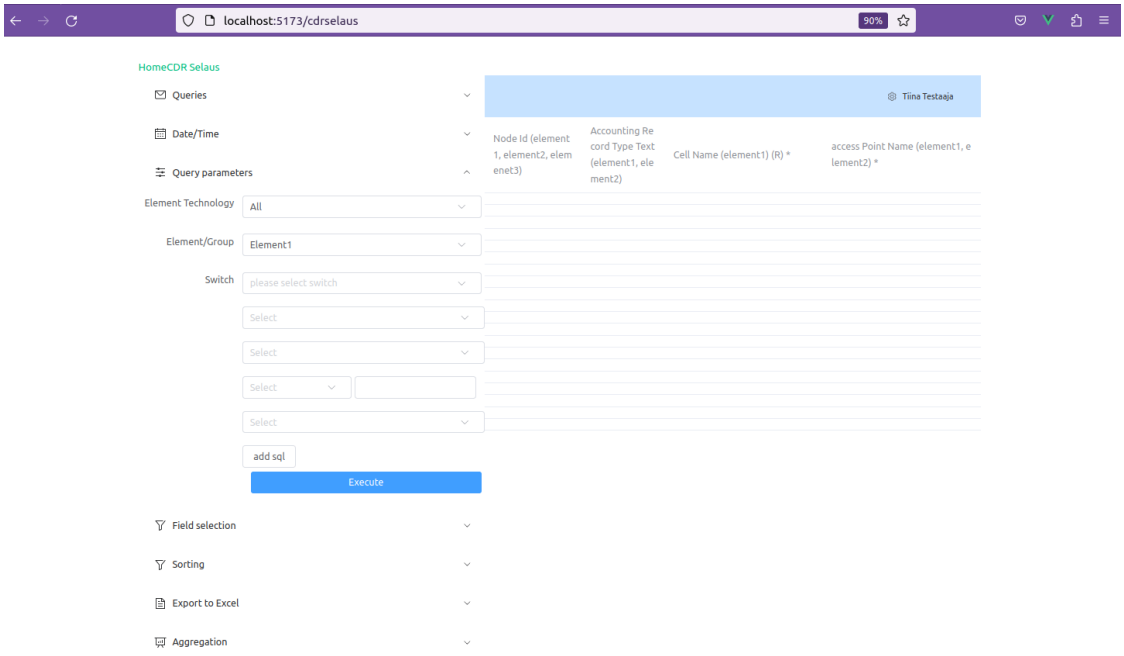
```
src > JS App.js > ...
1  import './App.css';
2  import LoginPage from './Pages/LoginPage';
3  import { BrowserRouter as Router, Routes, Route, Link } from "react-router-dom";
4  import Dashboard from './Pages/Dashboard';
5
6
7  function App() {
8    return (
9      <div className="App">
10
11         <div className="wrapper">
12       <h1>Application</h1>
13       <Router>
14         <div>
15           <ul>
16             <li>
17               <Link to="/login">login</Link>
18             </li>
19             <li>
20               <Link to="/dashboard">dashboard</Link>
21             </li>
22           </ul>
23         </div>
24
25         <Routes>
26           <Route path="/login" element={<LoginPage />}>
27           </Route>
28           <Route path="/dashboard" element={<Dashboard />}>
29           </Route>
30         </Routes>
31         </Router>
32
33     </div>
34     </div>
35   );
36 }
37
38 export default App;
```

Kuva 10 react router

Kuvassa 10 React Router otetaan käyttöön tuomalla BrowserRouter nimellä Router App nimiseen luokkaan.

<Router> tallentaa nykyisen sijainnin selaimen osoitepalkkiin puhtaiden URL-osoitteiden avulla ja navigoi selaimen sisäisen historiapinon avulla. Routers saa sisäänsä <Routes> tagin, joka huolehtii kaikista <Route> reiteistä. Aina kun sijainti muuttuu, <Routes> käy läpi kaikki alareitit löytääkseen parhaan vastaavuuden ja hahmottaa kyseisen haaran käyttöliittymästä. <Route> on routerin tärkein elementti, sillä ne yhdistävät komponentit, tietojen lataamisen ja datamutaatiot oikeisiin reitteihin. Kuten kuvassa 10, Route saa arvoikseen path tiedon, joka vastaa reitin urlia eli /dashboard ja elementin, joka vastaa ladattavasta komponentista, jonne reitti ohjautuu. Tässä tapauksessa komponentin Dashboard. (React Router. n.d)

5.2 Vue toteutus



Kuva 11 uusi cdrselausnäkymä vue

Kuva 11 esittää lopputulosta, joka saatiin aikaiseksi tämän projektin aikana. Vue projektissa tarkoitus oli toteuttaa käyttöliittymä ilman backendiä ja toteuttaa muutamia teoriaosuudessa käsiteltyjä asioita, kuten tilanhallinta ja reititys. Tässä kappaleessa on tarkoitus käydä teoria lyhyesti käytännössä.

5.2.1 projektin luominen

```
tintti@osboxes:~$ npm create vue@3

Vue.js - The Progressive JavaScript Framework

✔ Project name: … nw-vue-frontend
✔ Add TypeScript? … No / Yes
✔ Add JSX Support? … No / Yes
✔ Add Vue Router for Single Page Application development? … No / Yes

✔ Add Pinia for state management? … No / Yes
✔ Add Vitest for Unit Testing? … No / Yes
✔ Add an End-to-End Testing Solution? › No
✔ Add ESLint for code quality? … No / Yes

Scaffolding project in /home/tintti/nw-vue-frontend...

Done. Now run:

  cd nw-vue-frontend
  npm install
  npm run dev
```

Kuva 12 vue CLI

Projektin luominen onnistuu yksinkertaisimmillaan komennolla npm create vue@3

Tämä suorittaa create-vue komennon, joka perustuu Viteen. Luonti aloitetaan nimeämällä projekti, jonka

jälkeen työkalu esittää kysymyksiä valinnaisista ominaisuuksista, jotka voidaan halutessa liittää projektiin. Tällaisia ominaisuuksia ovat muun muassa TypeScript, pinia ja useat testauksen tueksi tarvittavat työkalut. Kuva 12 näyttää, että sovellukselle on valittu JSX tuki, Router ja pinia. Lopuksi projekti luodaan valittuun paikkaan sille annetulla nimellä.

5.2.2 Syntaksi

```
src > components > TheWelcome.vue > ...
1   <script setup>
2
3   </script>
4
5
6   <template>
7   <div>
8     <h1>CDR selaus</h1>
9     </div>
10  </template>
11
12  <style>
13  h1  {
14    color: cadetblue;
15  }
16  </style>
```

Kuva 13 syntaksi

TheWelcome.vue (kuva 13) osoittaa millainen vuen Single File komponentti voi yksinkertaisimmillaan olla. <template>-, <script>- ja <style>-lohkot kapseloivat ja sijoittavat samaan tiedostoon komponentin näkymän, logiikan ja tyylin.

<script setup> tagin sisällä oleva koodi käännetään komponentin setup()-funktion sisällöksi.

Tämä tarkoittaa, että toisin kuin normaali <script>, joka suoritetaan vain kerran, kun komponentti tuodaan ensimmäisen kerran, <script setup> -koodin sisällä oleva koodi suoritetaan aina, kun komponentin esiintymä luodaan. <script setup> kohdassa ilmoitettuja ylätason sidoksia kuten muuttujat, funktiomääritykset ja importit voidaan käyttää suoraan <template> mallissa. (Vue.js basic syntax. n.d)

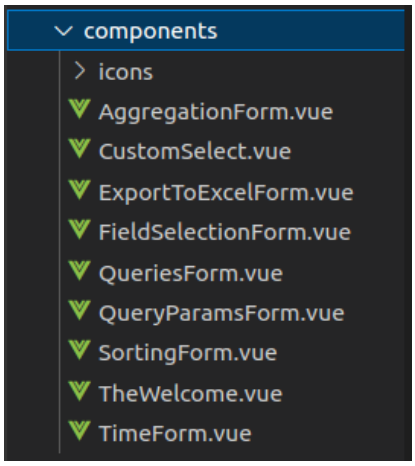
```
> components > QueriesForm.vue > {} script setup
<script setup>
import { useQueriesFormStore } from '@/stores/queriesFormStore'
import { storeToRefs } from 'pinia'
import CustomSelect from './CustomSelect.vue';
import {ref}  from 'vue'

const label = ref("queries")

const store = useQueriesFormStore()

const { userSelect, query, queryTxt, defaultQ, userOptions, queryOptions, queryGroupOptions } = storeToRefs(store)

</script>

<template>
    <el-form label-width="200px">
      {{ label }}
      <el-form-item label="Query group:">
        <CustomSelect v-model="query" :options="queryGroupOptions" placeholder="please select query" />
      </el-form-item>
> <el-form-item label="User:">…
      </el-form-item>
> <el-form-item label="">…
      </el-form-item>
> <el-form-item label="">…
      </el-form-item>
> <el-form-item>…
      </el-form-item>
    </el-form>


  </template>
```

Kuva 14 reaktiivisuus & komponentin lisääminen

Reaktiivinen tila on luotava erikseen Reactivity API:n avulla. Se tehdään tuomalla ref ja liittämällä ref funktio haluttuun objektiin (kuva 14). Samoin kuin setup()-funktiosta palautetut arvot, viitteet puretaan automaattisesti, kun niihin viitataan <template> malleissa. Ref() toimii ottamalla sisäisen arvon ja palauttamalla reaktiivisen ja muuttuvan ref-objektin, jossa yksi ominaisuus osoittaa sisäiseen arvoon. Ref objektit ovat muuttuvia, eli niille voidaan määrittää uusia arvoja. Ne ovat myös reaktiivisia, eli kaikkia toimintoja, jotka viittaavat objektin .value arvoon, seurataan ja niitä koskevat kirjoitustoiminnot laukaisevat niissä muutoksia. (Vue.js reactivity core. n. d)

<script setup> -alueen arvoja voidaan käyttää myös suoraan mukautettuina komponenttitunnisteen niminä. Kuvassa 14 voidaan ajatella, että CustomSelect on viitattavana muuttujana.

5.2.3 Komponentit



Kuva 15 luodut komponentit

Sovellus koostuu yhdeksästä komponentista, kuvan 15 mukaisesti. Jokainen nykyisen sovelluksen osa-alue on pilkottu omaksi komponentikseen, jotta sitä voidaan hallita ja testata omana kokonaisuutenaan, irrallaan muista.

Vue ohjeistaa nimeämään komponentit kuvaavalla nimellä, mieluiten kaksiosaisesti, jolloin luettavuus isoissa projekteissa säilyy.

```
src > components > SortingForm.vue > ...
1   <script setup>
2   import { useSortingFormStore } from '@/stores/sortingFormStore'
3   import { storeToRefs } from 'pinia';
4   import CustomSelect from './CustomSelect.vue';
5
6   const store = useSortingFormStore()
7   const {sorting, options } = storeToRefs(store)
8
9   </script>
10
11  <template>
12
13      <div>
14          <el-form  label-width="200px">
15              <el-form-item >
16                  <CustomSelect v-model="sorting" :options="options" multiple />
17              </el-form-item>
18              <el-form-item>
19                  <el-button style="width: 100%; margin-right: 5px;margin-top: 5px;" type="primary" @click="store.handleSubmit">Execute</el-button>
20              </el-form-item>
21          </el-form>
22
23          <p> Sorting items are {{ sorting }}</p>
24      </div>
25  </template>
```

Kuva 16 SortingForm komponentti

Kuvassa 16 on esitelty yksi lomakekomponenteista. Komponentti sisältää yhden <script> ja yhden <template> tagin. Script osuudessa komponentille alustetaan store, käyttämällä lomakkeelle omistettua useSortingFormStore() funktiota. Tämä store muuttuja sisältää tiedon kaikista sortingFormStoressa olevista objekteista ja niiden arvoista. SortingForm komponentti tarvitsee toimiakseen sorting ja options nimiset muuttujat, joiden arvot haetaan lomakekohtaisesta storesta, käyttämällä storeToRefs funktiota. StoreToRefs funktiota käytetään, jotta muuttujat säilyttävät reaktiivisuuden storen ulkopuolella. Sorting muuttujaa käytetään <template> osiossa, kustomoidun select komponentin arvona, johon se on määritelty direktiivillä v-model. Samalla kustomoidulle valintalistalle eli

sortingForm komponentin lapsikomponentille annetaan parametrina muuttuja options.

```
src > components > CustomSelect.vue > ...
1   <script>
2   export default {
3     props: [ 'options']
4   }
5   </script>
6
7
8   <template>
9           <el-select
10        collapse-tags
11        collapse-tags-tooltip
12        style="width: 100%; margin-right: 5px;margin-top: 5px;"
13      >
14        <el-option
15          v-for="item in options"
16          :key="item.value"
17          :label="item.label"
18          :value="item.value"
19        />
20      </el-select>
21  </template>
22
```

Kuva 17 CustomSelect komponentti

CustomSelect (kuva 17) saa parent komponentilta propseilla listan. Propsit muodostavat aina yhdensuuntaisen sidoksen aliominaisuuden ja pääominaisuuden välille, mikä tarkoittaa, että kun lista päivittyy parent komponentilla, muutokset valuvat alaspäin, mutta ei koskaan päinvastoin. Tällä tavoin estetään lapsikomponentteja päivittämästä vahingossa vanhemman tilaa, mikä voisi sotkea sovelluksen tiedonkulkua.

Lista asetetaan komponentin <template> tagin sisällä, v-for direktiivin avulla, valintalistaelementin arvo listaksi. V-for direktiiviä käytetään tässä tapauksessa muodostamaan luettelo options listan kohteista. V-for-direktiivi muodostetaan syntaksilla item in options, jossa options on lähdetaulukko ja item on alias iteroitavalle taulukkoelementille. (Vue.js, n. d.)

5.2.4 Tilanhallinta

Store luodaan käyttämällä defineStore()-komentoa, joka vaatii ensimmäisenä parametrina uniikin nimen. Tätä nimeä kutsutaan myös storen ID tiedoksi, joka on välttämätön, sillä Pinia käyttää tätä uniikkia id tietoa storen yhdistämiseksi selaimen kehitystyökaluihin. Palautetun funktion nimeämisessä käytetään use-alkuliitettä, joka tekee sen käytöstä idiomaattista. DefineStore() komennon toiseksi argumentiksi voidaan liittää joko Setup-funktio tai Options-objekti.

```
src > stores > sortingFormStore.js > useSortingFormStore > defineStore('sortingFormStore') callback > watch() callback
1   import { ref } from 'vue'
2   import { defineStore } from 'pinia'
3   import { useFormStore } from '@/stores/formStore'
4   import { storeToRefs } from 'pinia';
5   import {watch} from 'vue';
6
7   export const useSortingFormStore = defineStore('sortingFormStore', () => {
8
9   const store = useFormStore()
10
11  const {switchValue } = storeToRefs(store)
12
13  const sorting = ref([])
14
15 > const options = ref([…
28    ])
29
30 >  const options2 = […
43    ]
44
45 >  const options3 = […
58    ]
59
60      watch(switchValue, (newValue) => {
61        if(newValue === "Element1") {
62          options.value = options2
63        } else if(newValue !== "Element1") {
64          options.value = options3
65        }
66      });
67
68      function handleSubmit() {
69        console.log("submit!")
70      }
71
72      return {sorting, options, handleSubmit}
73  })
```

Kuva 18 SortingFormStore

Kuvassa 18 on luotu sortingFormStore, joka muodostetaan Setup funktion avulla. Setup funktiossa ref() muuttujat muodostavat tila ominaisuudet, computed() muuttujat muodostavat getterit ja functions() muuttujat muodostavat toiminnot. Setup funktio on joustavampi kuin Options objekti ja mahdollistaa tarkkailijoiden(watchers) käytön storen sisällä. Watch on tarkkailija, joka on vuen toiminnallisuus, jolla voidaan seurata jonkun olemassa olevan muuttujan tilan muuttumista ja päivittää vastaavasti sellaisia arvoja, joita tästä tilanmuutoksesta aiheutuu. Esimerkiksi sortingFromStoressa seurataan toiselta storelta saatavaa switchValue muuttujan arvoa. Kun toisella lomakkeella käydään muuttamassa swichValueta ja sen tila storessa muuttuu, watch toiminto havaitsee päivityksen ja laukaisee watch funktion. Nyt watch funktiossa katsotaan minkä arvon muuttuja saa, ja muuttuneen arvon perusteella päivitetään options muuttujan arvoa.

```
<script setup>
import { useSortingFormStore } from '@/stores/sortingFormStore'
import { storeToRefs } from 'pinia';
import CustomSelect from './CustomSelect.vue';

const store = useSortingFormStore()
const {sorting, options } = storeToRefs(store)

</script>
```

Kuva 19 storen käyttöönotto

Storea ei luoda ennen kuin use…Store() funktiota kutsutaan komponentin script tagissa. Kuten kuvassa 19, store luodaan kutsumalla useSortingFormStore funktiota. Luonnin jälkeen mitä tahansa storen ominaisuutta voidaan käyttää suoraan storesta. Jotta storesta poimittavat ominaisuudet säilyttäisivät reaktiivisuuden, tulee muuttujia kutsua hyödyntämällä storeToRefs() -komentoa. Se luo viitteitä jokaiselle reaktiiviselle

ominaisuudelle, joka on hyödyllistä erityisesti silloin kun mitään erillistä toimintoa ei kutsuta. Kuvassa 19 tätä hyödynnetään, kun kutsutaan sorting ja options muuttujia storesta.

5.2.5 Router

```
src > router > index.js > router > routes > name
1   import { createRouter, createWebHistory } from 'vue-router'
2   import HomeView from '../views/HomeView.vue'
3
4   const router = createRouter({
5     history: createWebHistory(import.meta.env.BASE_URL),
6     routes: [
7       {
8         path: '/',
9         name: 'home',
10        component: HomeView
11      },
12      {
13        path: '/cdrselaus',
14        name: 'cdrselaus',
15        component: () => import('../views/AboutView.vue')
16      }
17    ]
18  })
19
20  export default router
21
```

Kuva 20 Vue Router

Kuvassa 20 on luotu reititinobjekti, joka kutsuu createRouter funktiota, jolle on välitetty historia-avainarvot ja reittitaulukko parametreina. Funktio createRouter rakentaa itse vue router objektin, kun taas funktiolla createWebHistory mahdollistetaan käyttäjän historian luominen, jolla käyttäjä voidaan palauttaa edellisellä kerralla haettuun osoitteeseen. Kuten kuvassa 20 huomataan, itse reitit luodaan routes nimiseen listaan omina objekteinaan ja ne saavat seuraavat tiedot:

- path eli osoite, johon polku viittaa
- name eli valinnainen nimi, jota halutaan käytettävän linkin nimenä kyseiselle reitille
- component eli mikä komponentti ladataan, kun kyseistä reittiä kutsutaan

Komponentti voidaan määritellä objektille kahdella tapaa. Kuten kuvassa 20, reitin cdselaus komponentti on määritelty funktion kautta, kun taas home reitille on määritelty komponentiksi suoraan HomeView. Eroavaisuus liittyy latausteknikkaan. Lazy load eli hidas lataus tarkoittaa, että sovellus ei lataa tiettyjä reittejä ennen kuin käyttäjä klikkaa kyseistä linkkiä, joka ohjautuu kyseiselle reitille. Vue routerissa hidas lataus toteutetaan funktion kautta, niin, että tuonti lauseketta ei lisätä routerin yläosaan, vaan haku tapahtuu dynaamisella tuonti lausekkeella. Tämä lauseke hakee vaaditun tiedoston, jota reitillä käytetään.

```
src > App.vue > {} template > body
1  <script setup>
2  import { RouterLink, RouterView } from 'vue-router'
3
4  </script>
5
6  <template>
7
8          <nav>
9            <RouterLink to="/">Home</RouterLink>
10           <RouterLink to="/cdrselaus">CDR Selaus</RouterLink>
11         </nav>
12         <body >
13         <RouterView />
14         </body>
15 </template>
16
17
```

Kuva 21 Router käyttöönotto

Routerin käyttöönotto tehdään App.vue komponentissa. Router-Link objektit toimivat linkkeinä, joilla määritellään osoite, johon halutaan siirtyä. Kuten kuvassa 21, komponentille on luotu kaksi linkkiä, josta kumpikin osoittaa polkuihin, jotka määriteltiin routerissa (kuva 20). Huomaa, että tavallisten a-tunnisteiden sijaan käytetään mukautettua router-link linkkiä linkkien luomiseen. Tämä mahdollistaa sen, että Vue Router voi muuttaa URL-osoitetta lataamatta sivua uudelleen, hoitaa URL-osoitteen luomisen sekä sen koodauksen. Router-view näyttää URL-osoitetta vastaavan osan. Se voidaan sijoittaa minne tahansa sovelluksen asettelusta riippuen.

6 Vertailu

6.1 Vue ja React versus AngularJS

AngulasJS on sovelluskehys, joka tarjoaa oletuksena suuren määrän vaihtoehtoja ja ominaisuuksia, joiden avulla projektin pystyy aloittamaan ilman, että tarvitsee tehdä suuria määriä päätöksiä projektin alussa. ReactJS:ää, VueJS:ää ja AngularJS:ää vertaillessa on kuitenkin muistettava, että ReactJS ja VueJS ovat vain Javascript-kirjastoja, joten niihin on lisättävä ulkoisia komponenttikirjastoja, jotta ominaisuudet vastaisivat AngularJS:n ominaisuuksia.

6.1.1 Syntaksi

AngularJS on toiminut Vuen inspiraationa. Tämä selittää näiden kahden syntaksin samankaltaisuuden. Kummassakin mallipohjat kirjoitetaan HTML:llä, joihin sisällytetään sovelluskohtaisia elementtejä ja attribuutteja. AngularJS:ssä, tällaisia ovat esimerkiksi direktiivit. AngularissaJS:ssä direktiivit ovat DOM-elementin merkkejä, joilla voidaan ohjata HTML-kääntäjää liittämään tiettyä toimintaa DOM-elementtiin tai muuttamaan kyseistä DOM-elementtiä. Sekä standardeja että erityisiä direktiivejä voidaan käyttää sitomaan DOM-elementtejä AngularJS-sovelluksiin. Myös Vue käyttää direktiivejä – vertaa esimerkiksi v-if vs ng-if. (Vue. n.d)

React ei tarjoa jakoa malleihin ja direktiiveihin tai mallilogiikkaan. Mallin logiikka on kirjoitettava itse. Reactissa kaikki mallit ja käyttöliittymä tulkitaan uudelleen aina, kun tapahtuma tapahtuu. (rishabhsoft, n.d)

6.1.2 Komponentit

Vuessa on selkeämpi ero direktiivien ja komponenttien välillä. Direktiivit on tarkoitettu vain kapseloimaan DOM-manipulaatioita, kun taas komponentit ovat

itsenäisiä yksiköitä, joilla on oma näkemyksensä ja tietologiikkansa. AngularJS:ssä direktiivit tekevät kaiken ja komponentit ovat vain tietyntyyppisiä direktiivejä. (Vue. n.d)

AngularJS on tunnettu sen kaksisuuntaisesta datasidonnasta, joka mahdollistaa mallin ja näkymän tietojen automaattisen synkronoinnin. Se tarkoittaa, että jos sovellukselle annetaan uusi arvo, se johtaa sekä näkymän että mallin päivitykseen. Lisäksi se auttaa kirjoittamaan vähemmän peruskoodia, jolla sisällytetään sovellukseen komponenttien väliset vuorovaikutukset. Kaksisuuntaisella tiedonsidontamenetelmällä on kuitenkin negatiivinen vaikutus suorituskykyyn. (Bednarz, C. 2022)

Vuessa ei oletuksena käytetä kahdensuuntaista datasidontaa, vaikka tuki on olemassa, joka näin ollen auttaa Vueta skaalauksen ja suorituskyvyn suhteen. Vuen lähestymistapa eroaa suuresti AngularJS:stä, sillä Vue luottaa omaan ekosysteemiinsä, jolla se laajentaa toimintojaan. Se tekee siitä skaalautuvan ja mukautuvan, joka tarkoittaa, että Vue on mahdollista ottaa käyttöön niin pienissä osin kuin se on sovelluksen kannalta tarpeellista. Vue tarjoaa myös laajennuskirjastoja, joiden avulla se sopii hyvin erilaisiin tarpeisiin. (Vue. n.d)

AngularJS on tässä mielessä kankeampi, sillä se odottaa, että sen omia ominaisuuksia käytetään kaikkiin tarpeisiin ja se määrittelee hyvin tarkasti millainen sovelluksen rakenteen tulisi olla. Alussa tällainen mielivaltainen rakenne voi olla hyvä asia, mutta myöhemmin vaatimuksien ja laajuuden muuttuessa, se voi aiheuttaa haasteita. (Bednarz, C. 2022)

ReactJS tukee yksisuuntaista sidontaa. Riippuvuuksien toteutus auttaa singulaarisuuden erottumaan luokista ja virheriskit voidaan minimoida. Myös yksisuuntainen tietovirta ReactJS:ssä auttaa hallitsemaan monimutkaisuutta, joten on paljon helpompaa korjata suurten React-sovellusten itsenäisiä mutta samanlaisia komponentteja kuin suurempia AngularJS-sovelluksia. (rishabhsoft, n.d)

6.1.3 Tilanhallinta

AngularJS hoitaa kaiken itse ja siinä on suurin osa ominaisuuksista sisäänrakennettuna ilman ulkoisten resurssien tarvetta. AngularJS:ssä tilaa voidaan hallita useilla tavoilla, mutta on tärkeää hallita tilaa jäsennellyllä ja ylläpidettävällä tavalla, koska sillä voi olla merkittävä vaikutus AngularJS-sovelluksen yleiseen käyttäytymiseen ja suorituskykyyn. On suositeltavaa käyttää serviceä tai factorya $scopen sijaan sovelluksen tilan hallintaan, koska ne ovat paremmin uudelleenkäytettäviä ja testattavia. (AngularJS. n.d)

Vuessa ja Reactissa jokainen komponentti hallitsee omaa reaktiivista tilaansa, ja näiden tilaa voidaan hallita manuaalisesti sovelluksen läpi. Isoissa sovelluksissa on kuitenkin järkevämpää turvautua erillisiin tilanhallinnan kirjastoihin, jossa tilaa hallitaan koko sovelluksen laajuisesti storesta käsin.

6.1.4 Router

AngularJS tukee SPA:ta käyttämällä reititysmoduulia ngRoute. Tämä reititysmoduuli toimii URL-osoitteen perusteella. Kun käyttäjä pyytää tiettyä URL-osoitetta, reitityskone kaappaa kyseisen URL-osoitteen ja hahmottaa näkymän määritettyjen reitityssääntöjen perusteella. (AngularJS. n.d)

Sekä Vue, että React tukevat SPA:ta, mutta tarvitsevat parhaan toteutuksen takaamiseksi kolmannen osapuolen kirjastoja.

6.2 Vuen hyvät ja huonot puolet

Vue.js:n keskeinen kilpailuetu on nopea suorituskyky ja helppo ylläpito. Oppimiskäyrä on myös lyhyempi kuin Reactissa. Tämä tarkoittaa sitä, että kehittäjän tarvitsee käyttää vähemmän aikaa oppimiseen ja kirjoittamiseen. Vueta käytetään yleisimmin pienissä ja keskisuurissa projekteissa.

6.2.1 Vuen hyvät puolet

- Vuella on kattava listaus työkaluja ja kirjastoja (Vue.js official CLI, Development Tools, Vue Loader, Vue Router), joilla pääsee nopeasti kehitystyön alkuun.
- Vuen parhaina puolina pidetään sen joustavuutta ja yksinkertaisuutta. Kaikki mitä kehittäjä tarvitsee aloittaakseen, on perustiedot HTML:stä, CSS:stä ja JavaScriptistä.
- Vuen dokumentaatio on erinomainen, ja se on hyvin jäsennelty ja kattaa kaikki mahdolliset aiheet, ja se kuvaa tarkasti kaiken asennuksesta aina syvemmälle, kuten kaksisuuntaiseen tiedon sitomiseen ja sovelluksen skaalaukseen asti.
- Vue soveltuu isoihin ja pieniin sovelluksiin, ja se voidaan integroida mukaan niin pieninä paloina kuin se on projektin kannalta tarpeellista.
- Vue on todella kevyt kirjasto, joka tekee siitä nopean ja skaalautuvan.
- Vuen yksinkertainen syntaksi ja hyvä dokumentaatio takaavat sen, että oppimiskynnys on pieni.

(altexsoft. 2022; Stoyko, T. 2022; Neuhaus,J. 2017)

6.2.2 Vuen huonot puolet

- Vaikka Vuella on osaava ja aktiivinen yhteisö, se on pieni.
- Vuella ei ole samanlaista kattavuutta lisäosien kanssa kuin esimerkiksi Reactilla.

- Vuen kohdalla kielimuuria pidetään edelleen yhtenä suurena kompastuskivenä, sillä valtaosa Vuen käyttäjistä tulee edelleen Kiinasta. Tämän vuoksi kehittäjien on vaikea löytää ongelmille nopeasti ratkaisuja, vaikka dokumentaatio onkin hyvä.
- Vueta voidaan pitää liian joustavana, joka mahdollistaa huonon koodin ja monimutkaiset ratkaisut, jotka näin ollen näkyvät haasteena ylläpidettävyydessä.
- Vue osaajia on markkinoilla vähän, sillä Vue on suhteellisen nuori teknologia. Kokeneiden Vue osaajien löytäminen rajoittaa uusien Vue projektien aloittamista isoissa yrityksissä.
- Vueta käytetään edelleen suhteellisen pienissä projekteissa eikä se nauti suurten yritysten suosiota.

(altexsoft. 2022; Stoyko, T. 2022; Neuhaus,J. 2017)

6.3 Reactin hyvät ja huonot puolet

Reactin tärkein valttikortti on rikas ekosysteemi ja suosio. Se on suosituin JS-kehys, jossa on tuhansia ilmaisia työkaluja, kirjastoja ja koulutusresursseja. React.js:llä on kaikki välineet suuriin projekteihin.

6.3.1 Reactin hyvät puolet

- Reactissa päivitysten hallinta on helppoa kehittäjille, koska kaikki komponentit ovat eristettyjä, eivätkä yhden muutokset siirry muille.
- Reactin käyttämä JSX syntaksi koetaan hyvänä, sillä yksi tiedosto sisältää sekä markupit että logiikan, ilman erillistä tarvetta opetella erilaisia HTML-direktiivejä.
- React mahdollistaa datan ja sen esitystavan erottelemisen
- React on helppo ottaa käyttöön ja se on tehty suhteellisen yksinkertaiseksi ymmärtää.

- Koska React on vain kirjasto, sen oppiminen on nopeampaa, sillä sen mukana ei tule niin paljon oletusasetuksia.
- React toimii hyvin myös monimutkaisten operaatioiden ja tietokanta kutsujen kanssa
- React huolehtii, että suorituskyky on optimoitu
- React osaajia on markkinoilla paljon eikä Reactin suosion nähdä hiipuvan lähitulevaisuudessa, mikä takaa kehittäjille valoisat työmarkkinat.
- React soveltuu monelle eri alustalle, myös mobiiliin.
- Yhdensuuntainen datavirtaus

(altexsoft. 2020; Stoyko, T. 2022; Neuhaus,J. 2017)

6.3.2 Reactin huonot puolet

- React on vain sovelluskirjasto, ei sovelluskehys. Se tarvitsee useita lisäkirjastoja toimivan sovelluksen rakentamiseksi.
- React ei toteuta MVC- tai MVVM -arkkitehtuuria toisin kuin Angular ja Vue, vaan keskittyy vain näkymän hallintaan. Käyttäjät joutuvat itse valitsemaan minkä arkkitehtuurin he haluavat rakentaa.
- Reactiin on olemassa useita lisäkirjastoja, mutta suuri määrä tekee kehittäjälle ongelmaksi valita juuri omaan projektiin sopivin kirjasto.
- React kehittyy niin nopeaa tahtia, että sen dokumentaatio laahaa jäljessä, jolloin uuden kehittäjän on vaikea löytää ajantasaista tietoa.
- Koska lisäkirjastojen tarve sovelluksessa on suuri, menee suurin osa ajasta niiden opiskeluun
- JSX voidaan nähdä oppimista hidastavana tekijänä, koska se poikkeaa Angularin ja Vuen syntaksista.

(altexsoft. 2020; Stoyko, T. 2022; Neuhaus,J. 2017)

7 Yhteenveto

Tämän opinnäytetyön tarkoituksena on tutkia ja selvittää Vue ja React sovelluskehysten toimivuutta AngularJSn korvaajana. Tutkimus on toteutettu jakamalla se kolmeen osaan. Ensimmäisessä osassa käydään läpi sovelluskehysten teoriaa ja yleisiä piirteitä. Toisessa osassa sovelluskehyksiä lähestytään käytännön näkökulmasta, jossa toteutuksen lähtökohtina on sovellettu teoriaosuudessa läpikäytyjä osa-alueita. Viimeisessä osassa on tehty tutkimuksen kannalta oleellista vertailua sovelluskehysten hyvistä ja huonoista puolista sekä selvitetty niiden eroavaisuuksia.

Tutkimuksen aikana toteutetut sovellusratkaisut osoittavat, että kummatkin sovelluskehykset taipuvat nykyisen käyttöliittymän asettamiin vaatimuksiin. Yksistään sen perusteella ei voida siis päätellä kumpi teknologia soveltuu käyttöliittymäuudistukseen paremmin. Sovelluskehyksen valintaan vaikuttaa ennemmin kehittäjien kyky sisäistää uusi teknologia, koodin luettavuus, dokumentaation ja avun saatavuus sekä näkemys ylläpidettävyydestä. Vuen samankaltaisuus AngularJSn kanssa nopeuttaa oppimista, mutta heikosti saatavilla olevat esimerkit ja apu tekivät kehitystyöstä hitaampaa. Reactin oppimiskäyrä on jyrkempi, mutta kehityksen tueksi löytyi paljon esimerkkejä ja usein kysyttyjä kysymyksiä.

Jatkokehitysehdotuksena on lisätä taustajärjestelmien liittäminen nyt tehtyihin ratkaisuihin, esimerkiksi toteuttamalla jokin nykyisen käyttöliittymän käyttötapauksista. Tämä auttaa luomaan paremman käsityksen sovelluksen ja taustajärjestelmien yhdistämisen monimutkaisuudesta ja mahdollisista rajoitteista. Lisäksi sovellusratkaisut voidaan jalkauttaa kehitystiimille, jolloin saadaan tarkempi käsitys teknologioiden ylläpidettävyyden ja luettavuuden näkökulmista.

Selvityksen seuraavissa vaiheissa tulee pohtia tarkemmin mitä sovelluskehyksen päivittämisellä halutaan saavuttaa ja millaisella aikavälillä, millainen sovellusarkkitehtuuri halutaan toteuttaa ja miten valittu teknologia istuu haluttuun arkkitehtuurimalliin.

Lähteet

Abiola, F. n.d. What are Vue components?. Blogi-kirjoitus. Luettu 12/2022.

https://www.educative.io/answers/what-are-vue-components

Altexsoft. 2022. The Good and the Bad of Vue.js Framework Programming. Blogi-kirjoitus. Luettu 01/2023.

https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/

Altexsoft. 2020. The Good and the Bad of ReactJS and React Native. Blogi-kirjoitus. Luettu 01/2023

https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-reactjs-and-react-native/

AngularJS. n.d. AngularJS dokumentaatio. luettu 03/2023

https://docs.angularjs.org/guide/component

Bado, B. 2022. Naming Conventions in React JS. Blogi-kirjoitus. Luettu 02/2023.

https://www.upbeatcode.com/react/react-naming-conventions/

Bednarz, C. 2022. Commparing AngularJS vs. VueJS: Key Similarities and Differences. Blogi-kirjoitus. Luettu 03/2023

https://www.openlogic.com/blog/angularjs-vs-vuejs

Chakra-ui. n.d. Chakra-ui dokumentaatio. Luettu 03/2023.

https://chakra-ui.com/docs/components

Create Vue. n.d. Github-dokumentaatio. Luettu 12/2022.

https://github.com/vuejs/create-vue

Copes, F. 2018. The Vue Handbook: a thorough introduction to Vue.js. Blogi-kirjoitus. Luettu 01/2023

https://www.freecodecamp.org/news/the-vue-handbook-a-thorough-introduction-to-vue-js-1e86835d8446

Cromwell, V. 2017. Between the Wires: An interview with Vue.js creator Evan You. Blogi-kirjoitus. Luettu 12/2022

https://medium.com/free-code-camp/between-the-wires-an-interview-with-vue-js-creator-evan-you-e383cbf57cc4

Daityari, S. 2023. Angular vs React vs Vue: Which Framework to Choose. Blogi-kirjoitus. Luettu 01/2023

https://www.codeinwp.com/blog/angular-vs-vue-vs-react/

Dissanayake, N. 2022. Why and How to Use Webpack and Babel with Vanilla JS. Blogi-kirjoitus. Luettu 01/2023.

https://www.syncfusion.com/blogs/post/why-and-how-to-use-webpack-and-babel-with-vanilla-js.aspx

GeeksForGeeks. 2021. ReactJS components. ReactJS tutoriaali. Luettu 12/2022.

https://www.geeksforgeeks.org/reactjs-components/

GeeksForGeeks. 2021. ReactJS components. ReactJS tutoriaali. Luettu 12/2022.

https://www.geeksforgeeks.org/reactjs-router/

Gupta, V. n.d. React state management. Blogi-kirjoitus. Luettu 12/2022.

https://blog.loginradius.com/engineering/react-state-management

Herbert, D. 2022. What is React.js? (Uses, Examples, & More). Blogi-kirjoitus. Luettu 12/2022.

https://blog.hubspot.com/website/react-js

Javapoint. n.d. ReactJS tutoriaali. Luettu 12/2022.

https://www.javatpoint.com/react-jsx

Javapoint. n.d. ReactJS tutoriaali. Luettu 12/2022.

https://www.javatpoint.com/what-is-dom-in-react

Kagga, J. 2018. Understanding React components. Blogi-kirjoitus. Luettu 12/2022.

https://medium.com/the-andela-way/understanding-react-components-37f841c1f3bb

Kelly, D. 2021. How to Structure a large scale Vue.js Application. Luettu 12/2022.

https://vueschool.io/articles/vuejs-tutorials/how-to-structure-a-large-scale-vue-js-application/

Kemppainen, H. 2022. Modernit verkkosovelluskehykset ja kirjastot. Opinnäytetyö. Luettu 05/2023

https://www.theseus.fi/bitstream/handle/10024/780084/Kemppainen_Henri.pdf?sequence=3&isAllowed=y

KS, A. 2022. React.js Basics – The DOM, Components, and Declarative Views Explained. Blogi-kirjoitus. Luettu 12/2022.

https://www.freecodecamp.org/news/reactjs-basics-dom-components-declarative-views/

Maribojoc, M. 2020. A Closer Look at Vue Router. Luettu 12/2022.

https://vuejsdevelopers.com/2020/01/27/closer-look-at-vue-router/

Neuhaus, J. 2017. Angular vs. React vs. Vue: A 2017 comparison. Blogi-kirjoitus. Luettu 01/2023

https://medium.com/pixelpassion/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176

Penttinen, N. 2022. Frontend-teknologiamigraatio AngularJS:stä Reactiin. Opinnäytetyö. Luettu 05/2023

https://www.theseus.fi/bitstream/handle/10024/748827/Penttinen_Nino.pdf?sequence=2&isAllowed=y

Pinia. n.d. Pinia-dokumentaatio. Luettu 12/2022.

https://pinia.vuejs.org/introduction.html#a-more-realistic-example

React, beta. n.d React-dokumentaatio. Luettu 03/2023

https://beta.reactjs.org/learn/writing-markup-with-jsx

React. n.d. React-dokumentaatio. Luettu 12/2022.

https://reactjs.org/docs/introducing-jsx.html

React. n.d. React-dokumentaatio. Luettu 12/2022.

https://reactjs.org/docs/components-and-props.html

React. n.d. React-dokumentaatio. Luettu 12/2022.

https://reactjs.org/docs/create-a-new-react-app.html

Redux-toolkit.js. n.d. Redux-dokumentaatio. Luettu 03/2023

https://redux-toolkit.js.org/api/

Rishabhsoft. 2019. AngularJS Vs ReactJS: Know Which is the Best for Your Project. Blogi-kirjoitus. Luettu 02/23

https://www.rishabhsoft.com/blog/reactjs-vs-angularjs

Roy, G. 2022. CommonJS vs ES Modules in Node.js - A Detailed Comparison. Blogi-kirjoitus. Luettu 01/2023

https://www.knowledgehut.com/blog/web-development/commonjs-vs-es-modules

StackOverflow Developer Survey. 2022. Verkkosivu. Luettu 05/2023.

https://survey.stackoverflow.co/2022/#most-popular-technologies-webframe

Stoyko, T. 2022. Angular Vs React Vs Vue: The Main Differences And Use Cases. Blogi-kirjoitus. Luettu 01/2023

https://incora.software/insights/react-vs-angular-vs-vue

Technopedia. n.d. Luettu 12/2022.

https://www.techopedia.com/definition/25004/call-detail-record-cdr

Vue Router. n.d. Vue Router-dokumentaatio. Luettu 12/2022.

https://router.vuejs.org/guide/)

Vue.js. n.d. Vue.js-dokumentaatio. Luettu 12/2022.

https://vuejs.org/guide/essentials/template-syntax.html

Vue.js. n.d. Vue.js-dokumentaatio. Luettu 12/2022.

https://vuejs.org/guide/extras/rendering-mechanism.html

Vue.js. n.d. Vue.js-dokumentaatio. Luettu 12/2022.

https://vuejs.org/guide/scaling-up/state-management.html#what-is-state-management

Wadhwani, P. 2023. Why Use Angular in 2023? [For Entrepreneurs & Programmers]. Blogi-kirjoitus. Luettu 01/2023

https://www.bacancytechnology.com/blog/why-use-angular

Wadhwani, P. 2023. React vs Angular 2023: Which Front-end Framework You Should Choose?. Blogi-kirjoitus. Luettu 01/2023

https://www.bacancytechnology.com/blog/react-vs-angular

Winter, R. 2022. What Is Vue.js? (Uses + 5 Website Examples). Blogi-kirjoitus. Luettu 12/2022.

https://blog.hubspot.com/website/vue-js