Tampereen ammattikorkeakoulu

REACT-KOMPONETTIEN JAKAMINEN SOVELLUSTEN KESKEN

Yksityiset Node.js-paketit

Sami Pitkänen

OPINNÄYTETYÖ
Marraskuu 2021

Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

PITKÄNEN, SAMI:
REACT-KOMPONETTIEN JAKAMINEN SOVELLUSTEN KESKEN
Yksityiset Node.js-paketit

Opinnäytetyö 34 sivua, joista liitteitä 1 sivua
Marraskuu 2021

Verkkosovelluksia kehitetään joka päivä yhä useampaan tarkoitukseen, joten koodia syntyy paljon ja sama kehittäjä voi olla vastuussa useasta eri verkkosovelluksesta. Esimerkiksi samaan sovellusverkkoon kuuluvien verkkosovellusten välillä saattaa olla paljon samanlaisia komponentteja, kuten kirjautumisnäkymiä tai alatunnisteita. Tällaisen koodin kopioiminen sovelluksesta toiseen ei kuitenkaan ole kestävä ratkaisu, sillä löydetyt ongelmat ja tehdyt muutokset tulee yhtäkkiä muistaa tehdä useaan eri sovellukseen.

Tässä opinnäytetyössä luodaan React-verkkosovelluksen osista eli komponenteista Node.js-paketteja, jolla kyseistä komponenttia voidaan käyttää toisessa sovelluksessa, sekä esitetään kaksi eri tapaa jakaa ja hallinnoida näitä komponentteja. Lisäksi työssä käydään läpi modulaarista verkkokehitystä ja selitetään, miksi verkkosovellus on hyvä kehittää modulaarisesti.

Asiasanat: node.js, npm, react, modulaarinen, koodin jakaminen, verdaccio

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Business Information Systems
Software Development

PITKÄNEN, SAMI:
SHARING REACT-COMPONENTS BETWEEN APPLICATIONS
Using and maintaining private Node.js-packages

Bachelor's thesis 34 pages, appendices 1 pages.
November 2021

As technology and the processing power of our devices improve, we start developing web applications for more and more purposes, and especially large webs of web-applications could eventually suffer some overlap in the need for code and function. Components of web-applications such as Login views or footers could easily be reused with minor changes, but simply copying these files over would create a gap in maintainability. Suddenly any bug fixes or changes made to one such component should be made to all of them, which can be an extremely time-consuming task the more that piece of code was re-used.

This thesis will explain how to create and maintain Node.js-packages from pieces or ’components’ of your React-software, which can be used to move that component from one web-application to another and presents two different ways of privately sharing these packages between applications to re-use that piece of code. We will also explain modular development, as well as how and why web-applications should be built modularly.

Key words: node.js, npm, react, modular, code sharing, verdaccio

SISÄLLYS

1 JOHDANTO

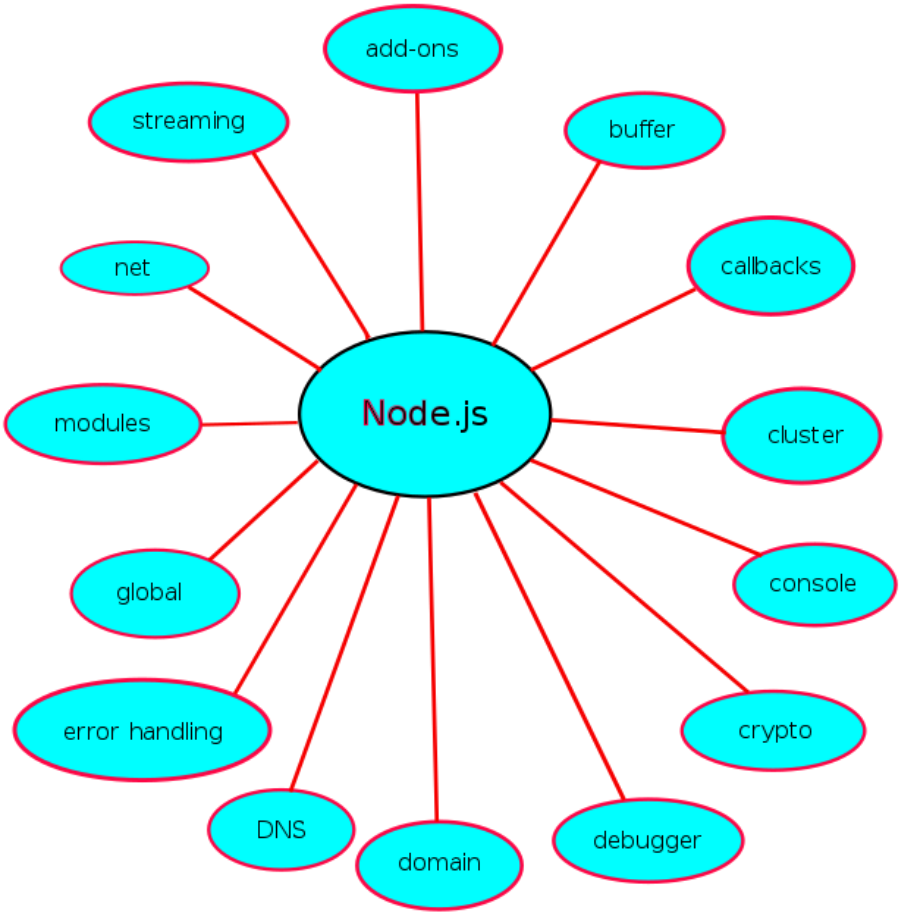
Internetistä voi löytää lyhyellä vilkaisulla monenlaisia eri web-sovelluksia. Verkkopeleistä lääkärin ajanvaraukseen, ennen asennusta vaativia sovelluksia voidaan siirtää verkkoon verkkosovelluksiksi. Näihin verkkosovelluksiin pääsee käsiksi jokaisella asiakkaan päätelaitteella, kuten kannettavilla tietokoneilla tai mobiililaitteella, joten asiakkaan on helppo jatkaa sovelluksen käyttöä ilman minkäänlaista asennuspakkoa. Monet konsultointiin keskittyvät kehittäjäyritykset kehittävät ja lisensoivat tällaisia sovelluksia päivätyökseen, joten sovelluksia syntyy monta ja lähdekoodia on paljon. Mitä jos tätä koodia halutaan jakaa sovellusten kesken? Syitä tähän voivat esimerkiksi olla tarve samanlaiselle “pohjalle” sovelluksen taustaksi, jotta sovellukset näyttävät yhtenäisiltä, tai kenties halutaan rakentaa useampi sovellus eri tason käyttäjille, mutta osa näkymistä tulisi näkyä useammassa sovelluksessa.

Koodia jakaessa yksinkertaisin ratkaisu on kopioida tiedostot tai koodi sovelluksesta toiseen, mutta tämä ei ole kestävä ratkaisu. Jos alkuperäisestä koodista löytyy bugi tai sitä halutaan muuttaa, muutokset tulee muistaa kopioida kaikkiin alkuperäistä koodia käyttäviin sovelluksiin. Tämä saattaa helposti unohtua, jolloin sovellusten jaettu koodi alkaa erota muista ja ajan myötä ongelmien etsimisestä ja korjaamisesta saattaa tulla hidasta ja vaivalloista koodien eroavaisuuksien vuoksi. Tämän ongelman ratkaisemiseksi tuli löytää keino tallentaa jaettu koodi ja sen muutokset paikkaan, josta sitä voi käyttää jokainen sitä tarvitseva sovellus.

Tässä opinnäytetyössä rakennettiin yksinkertainen sovellusverkko, jossa hyödynnettiin Node.js-ajoympäristöä ja yksityisiä, sovelluksen osista luotuja Node.js-paketteja koodin jakamiseksi sovellusten välillä. Jotta koodin jakaminen olisi mahdollisimman helppoa, sovellus rakennettiin modulaarisesti, eli sen osat kehitettiin erilleen omiin tiedostopuihinsa.

2 NODE.JS

Node.js tai Node on vuonna 2009 julkaistu avoimen lähdekoodin ajoympäristö JavaScript-koodin suorittamiseen. Node.js suorittaa JavaScript-koodia palvelimella käyttäjän selaimen sijaan, mikä säästää käyttäjän resursseja ja mahdollistaa täysin JavaScript-pohjaisen full-stack sovelluksen luomisen. Aktiivisen kehittäjäyhteisön lisäksi Node.js-sovelluksilla on käytössään valtava kokoelma kolmannen osapuolen julkaisemia Node.js-paketteja, jotka helpottavat sovelluksen kehittämistä tarjoamalla esimerkiksi funktioita matemaattisten yhtälöiden laskemiseen tai jopa täysin käyttövalmiita UI-komponentteja. Noden tärkeimpiä ominaisuuksia kuvataan alla kuvassa 1.



KUVA 1. Node.js tärkeät ominaisuudet (Sae1962, Wikimedia commons).

2.1 Node package manager

Node package manager eli yleisemmin npm on maailman suurin ohjelmakirjasto, joka sisältää yli 1.7 miljoonaa julkista ja ilmaisessa käytössä olevaa Node.js-pakettia, joita voi käyttää ilman rekisteröitymistä (replicate.npmjs, 12.10.2021). Yli 11 miljoonaa kehittäjää käyttää npm-kirjastoa ympäri maailmaa (npmjs.com, 19.10.2021). Esimerkiksi monet avoimen lähdekoodin kehittäjät käyttävät npm-kirjastoa työnsä jakamiseen. Ilmaisella npm-versiolla voi julkaista rajattoman määrän julkisia koodipaketteja, mutta yksityisten pakettien julkaisemiseksi tarvitaan sen maksullinen 'Pro' versio.

2.1.1 Package.json

Jokaisella Node.js-paketilla on package.json-tiedosto, joka määrittelee kaiken pakettiin liittyvän metadatan, kuten nimen, version ja lisenssin. Tämä tiedosto luodaan kehityksen alussa ja sitä päivitetään jatkuvasti.

```
{
  "name": "thesis",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.11.4",
    "@testing-library/react": "^11.1.0",
    "@testing-library/user-event": "^12.1.10",
    "react": "^17.0.2",
    "react-dom": "^17.0.2",
    "react-scripts": "4.0.3",
    "web-vitals": "^1.0.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  }
}
```

KUVA 2. Uusi package.json-tiedosto.

Kuten yllä olevasta kuvasta 2 voi nähdä, paketin käyttämät moduulit listataan package.json-tiedostoon riippuvuuksiksi eli “dependencies”. Tämän avulla npm osaa asentaa oikeat paketit sovelluksen ajoa varten. Kun komento “npm install” ajetaan, npm tutkii riippuvuudet läpi ja asentaa ne yksi kerrallaan, käyden samalla läpi asennettujen pakettien omat package.json tiedostot ja asentamalla vastaavasti niiden tarvitsemat riippuvuudet.

Riippuvuudet voivat olla tavallisten riippuvuuksien lisäksi kehitysriippuvuuksia eli “dev dependencies” tai vertaisriippuvuuksia eli “peer dependencies”. Kehitysriippuvuudet asennetaan vain, kun ohjelma ajetaan kehitystilassa, joten ne ovat esimerkiksi sovelluksen testaamiseen liittyviä riippuvuuksia, joita ei tarvita julkaisuvaiheessa. Paketin vertaisriippuvuuksia puolestaan ei asenneta automaattisesti, vaan pakettia käyttävän koodin tulee lisätä vertaisriippuvuus omaksi riippuvaisuudekseen, jotta pakettia voi käyttää. Vertaisriippuvuuksia käytetään, kun tietystä kirjastosta saa olla koko projektissa vain 1 versio.

2.1.2 Package-lock.json

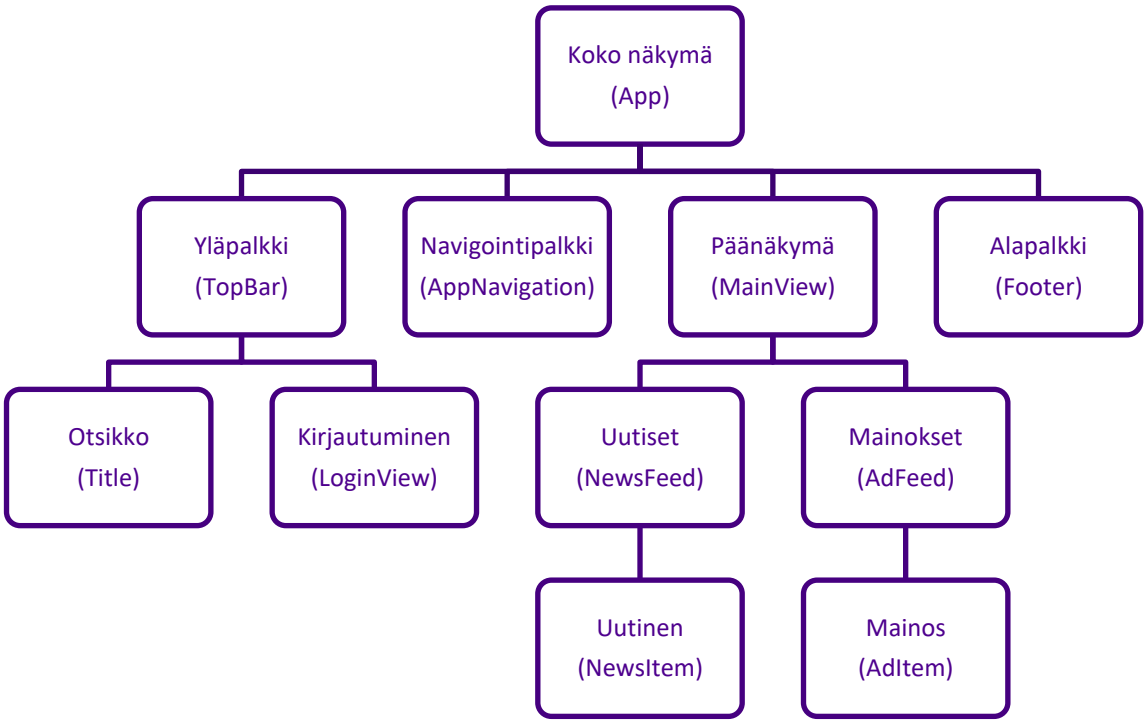
Kun Node.js-sovelluksen riippuvuudet asennetaan ensimmäistä kertaa, sille luodaan package-lock.json -tiedosto. Tämä tiedosto lukitsee asennettujen pakettien versionumerot, jotta sovellus näyttäisi samalta ja toimisi samalla tavalla sovelluksen julkaisuvaiheessa. Ajamalla komennon “npm ci” npm tarkistaa package-lock.json -tiedoston ja asentaa juuri siinä listatut versiot sovelluksen riippuvuuksista. Tämä tiedosto luodaan automaattisesti eikä sitä tule koskaan muuttaa manuaalisesti.

3 MODULAARINEN KEHITYS

3.1 Modulaarisuuden periaatteet

Jotta koodia voitaisiin jakaa verkkosovellusten välillä mahdollisimman kivuttomasti, koko sovellus on hyvä kehittää modulaariseksi alusta alkaen. Modulaarisuus tarkoittaa sitä, että verkkosovelluksen toiminnallisuus hajotetaan pienemmiksi ”moduuleiksi”, joista jokainen on vastuussa tietystä osasta verkkosivua.

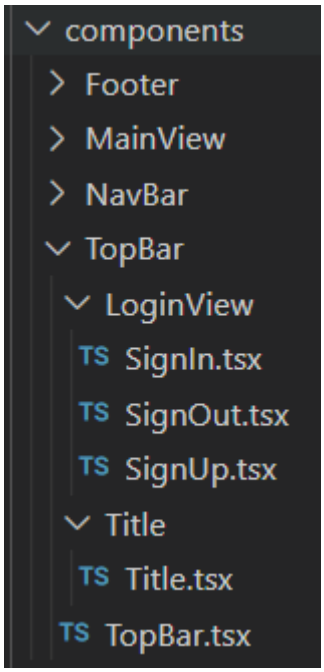
Ajatellaan esimerkiksi uutissivuston etusivua. Verkkosivulle luodaan ”pohja”, joka määrittelee sivun asettelun ja jokaiselle moduulille annettavan tilan. Nämä moduulit puolestaan saattavat jakaa oman tilansa alemman tason moduuleille ja niin edelleen. Esimerkiksi ”Uutiset”-moduuli sisältäisi listan uutisia, mutta jokainen yksittäinen uutinen käyttäisi samaa moduulia, jolle annetaan uutisen kuva ja otsikko. Yksinkertaisen uutissivuston etusivun rakennetta on kuvattu alla kaaviossa 1.



KAAVIO 1. Suunnitelma uutissivuston etusivusta modulaarisena.

3.2 Modulaarinen tiedostorakenne

Hyvin suunniteltu ja toteutettu modulaarinen komponentti on eritelty omiin tiedostoihinsa, jotka sisältävät kaiken komponentin tarvitseman koodin, kuten mahdolliset tietokanta- tai rajapintakutsut, tyylit ja käytetyt grafiikat.



KUVA 3. Esimerkki modulaarisesti rakennetun uutissivuston komponenteista.

Yllä oleva kuva 3 kuvaa kaaviossa 1 suunniteltuun uutissovellukseen kuuluvien komponenttien tiedostorakennetta. ”TopBar”-komponentin kansiossa ovat myös ”LoginView”- ja ”Title”-komponentit, jotka sijoittuvat verkkosovelluksessa ”TopBar”-komponentin sisälle.

3.3 Modulaarisuuden hyödyt

3.3.1 Jaettu vastuu sovelluksen toimivuudesta

Kun jokainen moduuli on vastuussa omasta osastaan koodia, ongelmien aiheuttajat on helpompi jäljittää ja korjata. Turhaksi käynyt moduuli on myös paljon helpompi poistaa koodista samaan tiedostoon liitettyyn koodiin verrattuna.

3.3.2 Helpompi lukea ja navigoida

Hyvin suunnitellun modulaarisen sovelluksen tiedostorakenne on selkeämpi ja tietystä toiminnallisuudesta vastaava koodi on paljon helpompaa löytää. Koodin määrä pysyy lisäksi pienempänä, joten koodi on luettavampaa. Luettavuus on erityisen tärkeätä kehittäjätiimeissä.

3.3.3 Uudelleenkäytettävää koodia

Moduuleja voi helposti käyttää uudelleen ja muuntaa eri tarkoitukseen saman sovelluksen sisällä, joten toistuvan koodin määrä vähenee. Toistuva koodi on suuri uhka sovelluksen huollettavuudelle.

4 DEVELOPMENT STACK

Tässä kappaleessa kuvataan kappaleessa 4 kehitettyyn verkkosovellukseen käytetyt teknologiat. Kehitys tehtiin JavaScript-ohjelmointikielellä Visual Studio Code -koodieditoria käyttäen.

4.1 React.js

React.js tai yleisemmin React on JavaScript-kirjasto ja tapa luoda sovelluksen käyttöliittymä. React pilkkoo verkkosivun pienemmiksi palasiksi eli komponenteiksi. Nämä komponentit ovat omia JavaScript-funktioitaan, jotka palauttavat näkyvää osuutta kuvaavaa koodia. React pinoaa nämä näkyvät osuudet ja näyttää tämän järjestyksen käyttäjälle. React oli käytetyin web-infrastruktuuri vuoden 2021 alussa (Stack Overflow, Toukokuu 2021).

4.2 TypeScript

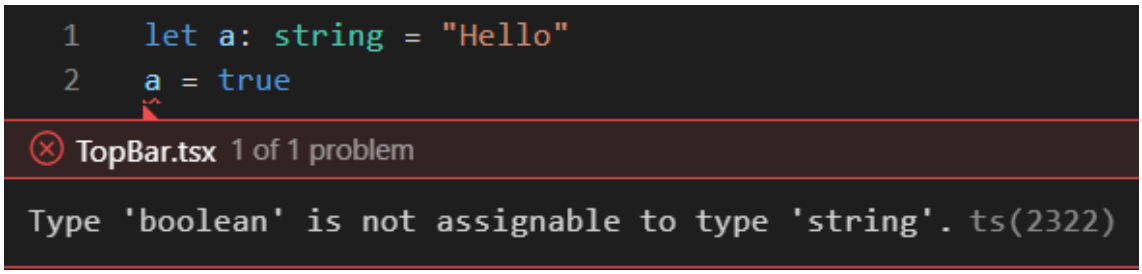
TypeScript on Microsoftin vuonna 2012 julkaisema ”laajennus” JavaScriptille. TypeScript on kuin oma ohjelmointikielensä, joka muuntaa itsensä JavaScriptiksi ajon aikana, joten se toimii kaikilla laitteilla ja selaimilla kuin normaali JavaScript-koodi.

TypeScriptin hyöty on sen kyky vahtia kehittäjän koodia ja huomata yleisimpiä virheitä erityisesti muuttujien käytössä. Ainoa ero JavaScriptin ja TypeScriptin koodisyntaksin välillä on muuttujien ”tyypitys”. Tyypityksellä kehittäjä voi antaa jokaiselle muuttujalle muuttujatyypin, kuten ”string” (merkkijono), ”boolean” (totuusarvo) tai ”number” (numero). Tällaisiin muuttujiin voi tyypin määrityksen jälkeen asettaa vain sen tyypin arvoja, jolloin TypeScript tietää, mitä ominaisuuksia ja funktioita kyseisellä arvolla on. Tuntemattoman tyypin muuttujille voidaan antaa ”any”-tyyppi, jolloin TypeScript ei yritä ymmärtää muuttujan sisältöä. Esimerkkejä tyypityksestä kuvassa 4.

```
const a: string = "Hello"
const b: number = 100
const c: boolean = true
const d: any = {
  name: "Jacob"
}
```

KUVA 4. TypeScript-tyypitys.

Jos tietyn tyypin muuttujalle yritetään antaa sopimaton arvo, TypeScript ei suostu kääntämään ohjelmaa. Suosituimmat koodieditorit osaavat havaita ja korostaa tällaiset ongelmat jo ennen ohjelman kääntämistä. Kuvassa 5 on tästä esimerkki.

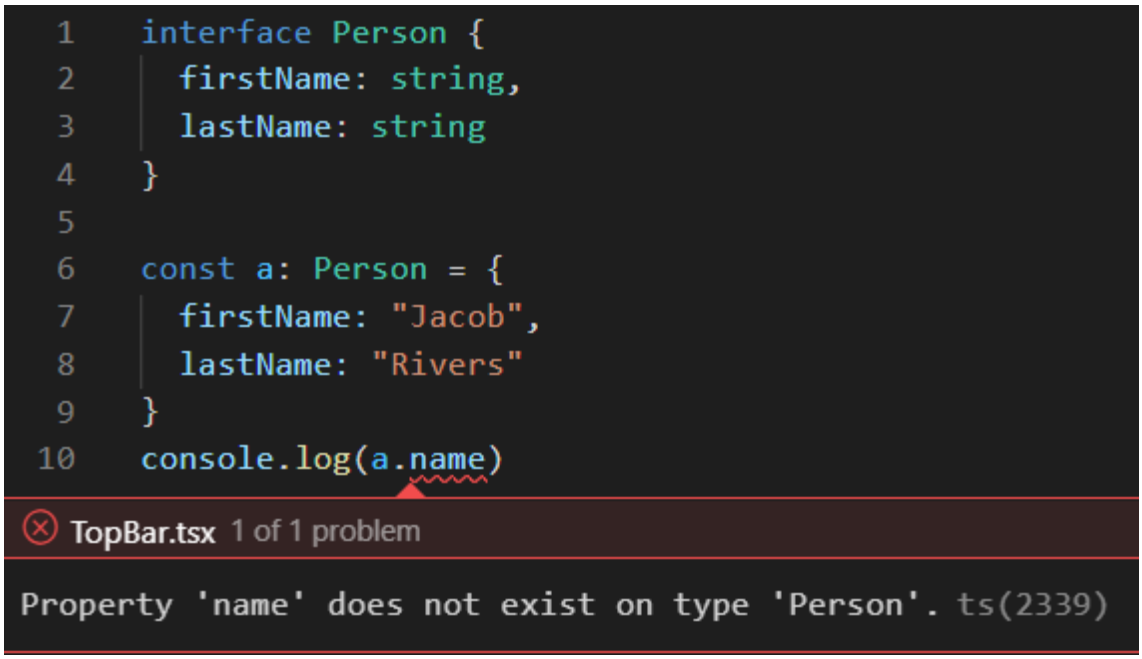


KUVA 5. TypeScriptin varoitus väärän tyypin arvosta.

Tyyppejä voidaan myös luoda itse ”interface”-objektien avulla. Omien tyyppien luominen ja käyttö on aina parempi vaihtoehto kuin ”any”-tyypin käyttäminen, jotta TypeScript osaa varoittaa vääränlaisista arvoista tai vääristä funktiokutsuista. Alla esimerkit ”any”-tyypin käytön aiheuttamasta virheestä ja samasta koodista itse määritetyllä tyypillä.

```
1  const a: any = {
2    firstName: "Jacob",
3    lastName: "Rivers"
4  }
5  console.log(a.name)
6
```

KUVA 6. TypeScript ei osaa varoittaa ”any”-tyypin muuttujien väärinkäytöstä.



KUVA 7. TypeScript osaa varoittaa itse määritetyn tyypin muuttujan väärinkäytöstä.

4.3 Create React App

Create React App on virallinen tapa luoda helposti ylläpidettävä React-applikaatio sekunneissa. Create React App lisää sovellukseen valmiiksi konfiguroidut Webpack- ja Babel-työkalut, mutta näiden työkalujen asetukset on piilotettu, jotta kehittäjän ei tarvitsisi huolehtia niistä.

Mikäli näitä asetuksia halutaan muuttaa manuaalisesti, kehittäjän täytyy irrottaa koko sovellus Create React App:in hallinnasta, jolloin kaikki piilotetut tiedostot tulevat näkyviin, eikä niitä ylläpidetä enää automaattisesti. Irrottaminen on pysyvä muutos, eikä sitä voi perua.

Create React App tarjoaa myös monenlaisia pohjia sovellukselle. Näitä pohjia käyttämällä luotuun sovellukseen asennetaan valmiiksi erilaisia luokkakirjastoja. Yksi tällaisista pohjista asentaa valmiiksi konfiguroidun TypeScriptin sovelluksen pohjaksi.

4.4 MUI

Ennen tunnettu nimellä Material-UI, MUI on suuri ja ilmainen UI-kirjasto, joka tarjoaa käyttäjälle kestäviä ja helposti muokattavia UI-komponentteja React-sovelluksiin. Nämä komponentit helpottavat UI-kehittäjän työtä ja tekevät lopputuloksesta helposti ammattimaisen näköisen.

MUI-komponentit on kehitetty “mobile first” -periaatteella, eli mobiilinäkymä luodaan ensin ja näkymää skaalataan ylöspäin suuremmille näytöille. Tämä varmistaa, että oikein käytettyinä komponentit näyttävät hyvältä sekä mobiililaitteilla että tietokoneilla.

4.5 Express

Express on minimalistinen ja taipuisa rajapinta Node.js applikaatioille. Expressin avulla sovelluksen API eli ohjelmointirajapinta voidaan luoda helposti ja nopeasti. Express-rajapinta kirjoitetaan JavaScriptillä.

4.6 Verdaccio

Verdaccio on yksinkertainen tapa luoda oma paikallinen ja yksityinen npm-kirjasto. Tähän kirjastoon voidaan julkaista rajattomasti omia Node.js-paketteja. Verdaccion avulla voidaan myös julkaista omia versiota julkisista Node.js-paketeista yksityiseen käyttöön.

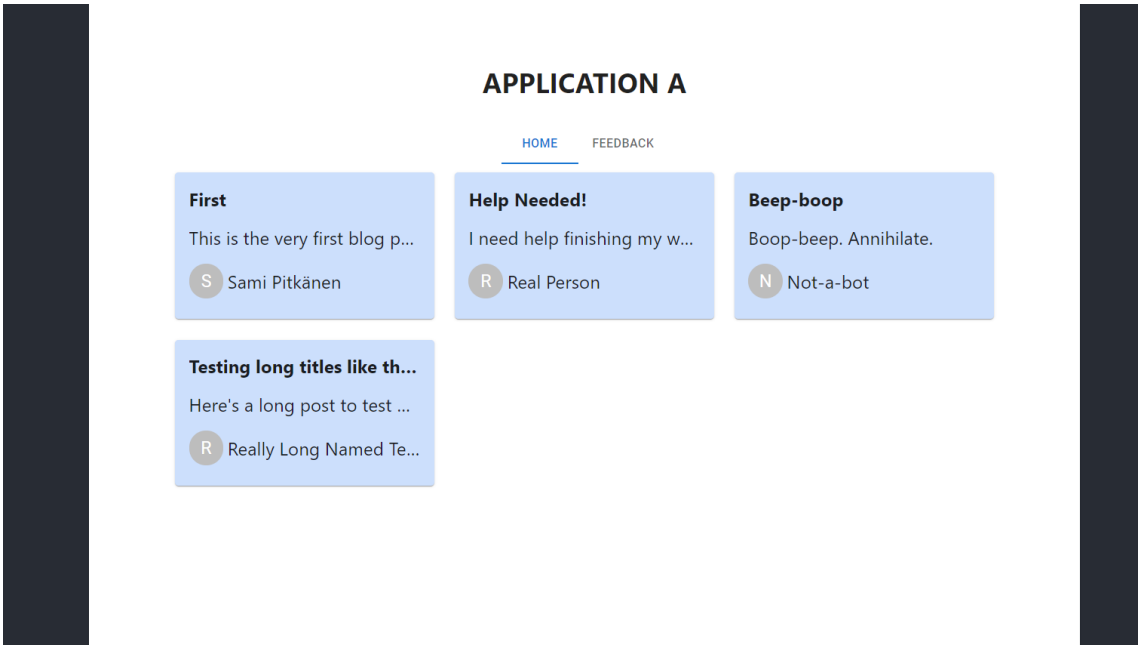
5 KEHITYSTYÖ

5.1 Tavoite

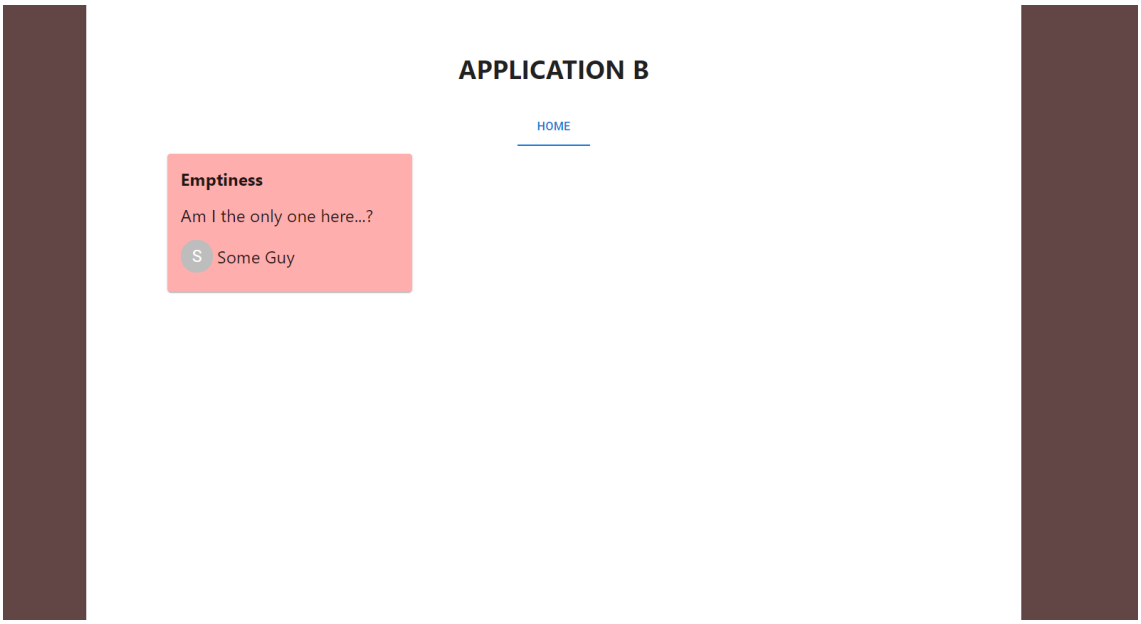
Tätä kappaletta varten kehitettiin 2 yksinkertaista verkkosovellusta. Nämä sovellukset löytyvät GitHubista, ja linkit löytyvät Liitteet-osiosta. Tässä kappaleessa käydään läpi eri tapoja uudelleen käyttää komponentteja näiden sovellusten välillä.

5.2 Alkutilanne

Aluksi luotiin 2 applikaatiota, applikaatio A ja applikaatio B, sekä erillisen serverin EXPRESS, jonka ainoa vastuu on vastaanottaa ja säilyttää palautetta verkkosivusta. Alla kuvat applikaatioista.

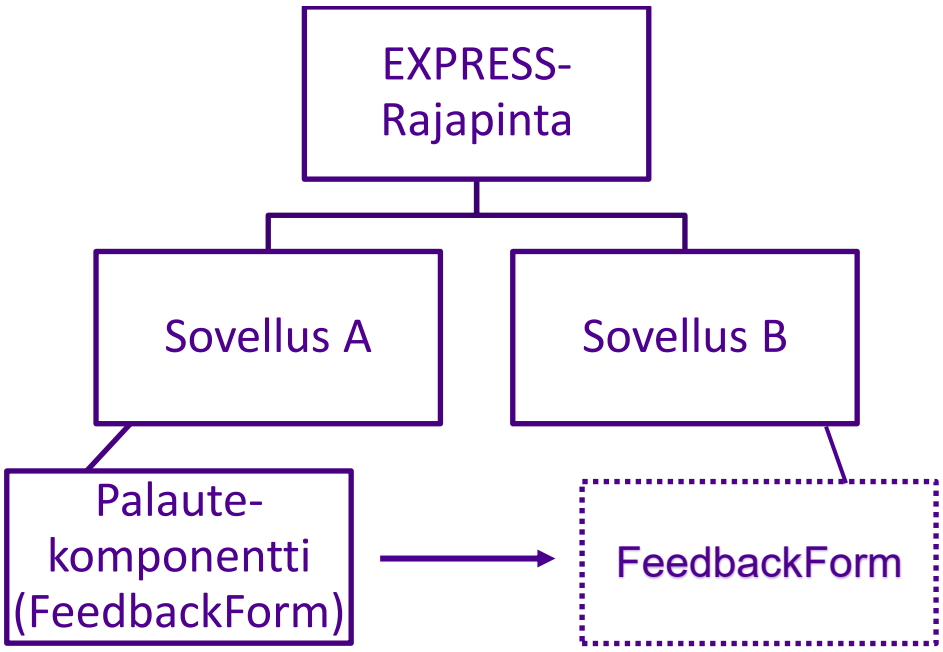


KUVA 8. Applikaatio A.

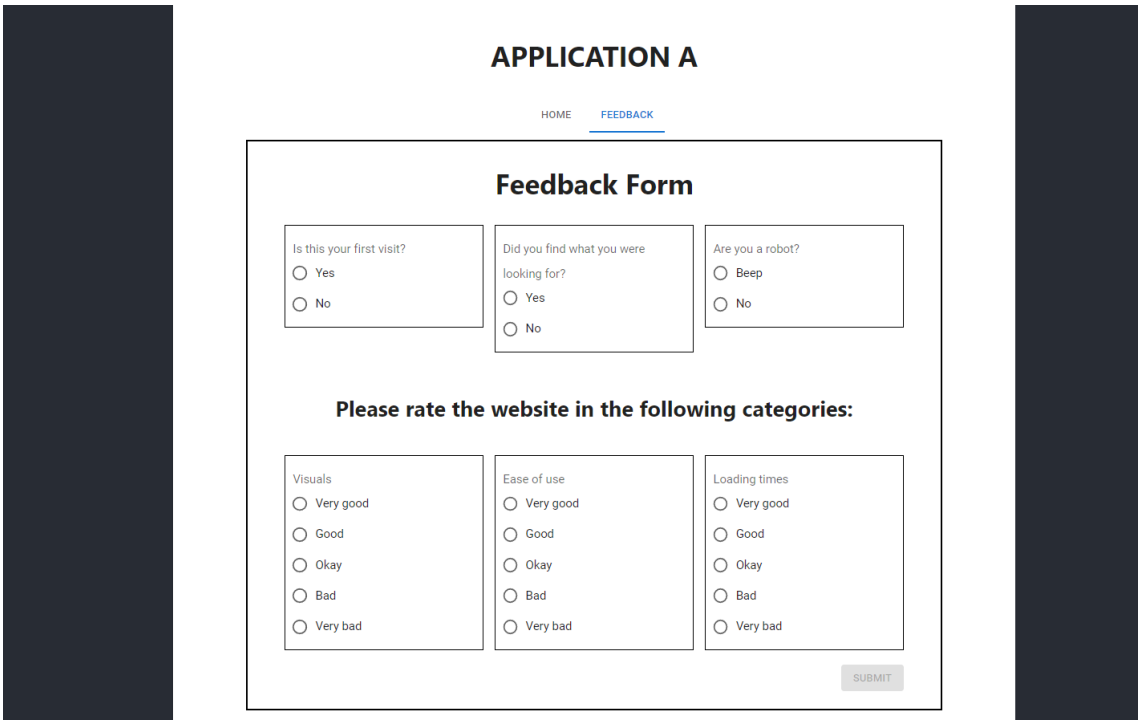


KUVA 9. Applikaatio B.

Applikaatiolla A on modulaarisena komponenttina palautesivu, jossa käyttäjä voi jättää palautetta sivuston ulkonäöstä ja yleisestä toimivuudesta. Tämä komponentti tuli jakaa sovelluksesta toiseen tavalla, joka tekee ongelmanratkaisusta ja jatkokehityksestä mahdollisimman helppoa ja nopeaa. Tämä tilanne on kuvattu alla kaaviossa 2. Palautesivun voi nähdä alta kuvasta 10.



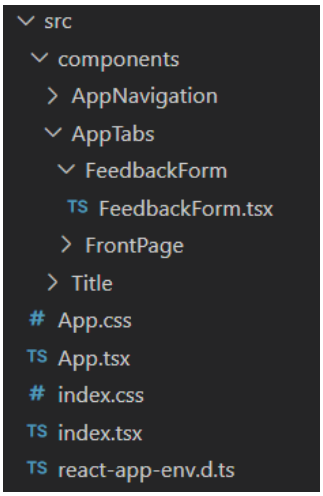
KAAVIO 2. Kaavio komponentin jakamisesta toiseen sovellukseen.



KUVA 10. Applikaatio A:n jaettava palautesivu.

5.3 Paketin oma package.json

Koska palautesivu kehitettiin modulaarisesti, sen koodi on jo erillään muusta koodista, kuten kuvattu alla kuvassa 11. Jotta tämä tiedosto voitiin paketoida, luotiin Feedback-kansiolle oma package.json-tiedosto. Tämä tapahtui helposti avaamalla Feedback-kansio terminaalissa ja ajamalla komennon “npm init -y”. Uusi package.json-tiedosto näkyy kuvassa 12.



KUVA 11. Applikaatio A tiedostorakenne.

```
{
  "name": "feedbackform",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

KUVA 12. ”npm init -y” -komennon luoma package.json-tiedosto.

5.4 TypeScriptin kääntäminen

Tätä vaihetta ei tarvita, jos komponentti rakennettiin alusta alkaen JavaScriptillä. Harjoituksen vuoksi komponentit rakennettiin kuitenkin TypeSciptiä käyttäen, joten ne tuli kääntää JavaScriptiksi ennen julkaisua. Tähän käytettiin TypeScriptin omaa kääntäjää.

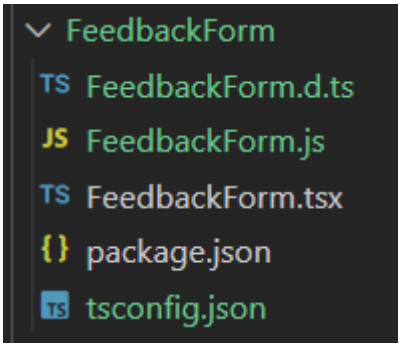
Ensimmäiseksi luotiin tsconfig.json-tiedosto. Tämän tiedoston kaikkia asetuksia ei käydä tässä läpi, mutta on tärkeää ymmärtää muutama sääntö, joita käytetään koodin kääntämiseksi.

```
{
  "compilerOptions": {
    "target": "es5",
    "lib": [
      "dom",
      "dom.iterable",
      "esnext"
    ],
    "allowJs": true,
    "skipLibCheck": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "noFallthroughCasesInSwitch": true,
    "module": "esnext",
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": false,
    "declaration": true,
    "jsx": "react-jsx"
  },
}
```

KUVA 13. FeedbackForm-komponentin tsconfig.json-tiedosto.

Kuvassa 13 näkyvistä asetuksista tärkeimmät ovat ”noEmit”- ja ”decleration” -säännöt. Jos ”noEmit”-sääntö on tosi, TypeScriptin kääntäjä ei luo uutta JavaScript-tiedostoa, vaan vain tarkistaa tiedostot tyyppivirheiden varalta. Se siis asetettiin epätodeksi. ”declaration”-sääntö taas asetettiin todeksi, sillä sen avulla TypeScript ymmärsi luoda kääntöprosessin aikana JavaScript-tiedostosta erillisen FeedbackForm.d.ts-tyyppitiedoston, jonka avulla komponenttia käyttävät kehittäjät voivat edelleen käyttää komponentin TypeScript-omaisia ominaisuuksia.

Näiden muutosten jälkeen tuli enää ajaa ”npx -p typescript tsc” -komento. Komento ei tulostanut mitään, mikä on hyvä merkki, ja nyt komponentin kansioon ilmestyi kaksi uutta tiedostoa. Tässä tapauksessa nämä tiedostot olivat ”FeedbackForm.js” ja ”FeedbackForm.d.ts”. Uusi tiedostorakenne alla kuvassa 14.



KUVA 14. Komponentin tiedostorakenne TypeScript-kääntämisen jälkeen.

5.5 Palautesivun paketointi

Seuraavaksi aikaisemmin osassa 5.3 luotua package.json-tiedostoa muutettiin niin, että ”main”-kenttään asetettiin tie tiedostoon, josta komponentin ”exportataan” kuten alla kuvassa 15.

```
import React from 'react';

// Luodaan komponentti.
const CreatedComponent = () => {
  return (
    <p>Hello</p>
  );
};

// Sallitaan pääsy komponenttiin tiedoston ulkopuolelta.
export default CreatedComponent;
```

KUVA 15. Yksinkertaisen komponentin ”exporttaaminen”.

”Exporttaaminen” paljastaa kyseisen komponentin muiden komponenttien käyttöön saman projektin tai komponentista luotua moduulia käyttävän projektin sisällä. Tämä FeedbackForm-komponentti ”exportattiin” FeedbackForm.js-tiedostossa, joten ”main”-kentän arvoksi asetettiin ”FeedbackForm.js”. Huomioi, että tähän ei käy TypeScript-tiedosto. Myös muita tietoja kuten komponentin kuvaus ja lisenssi on hyvä lisätä, mutta ne eivät olleet tarpeellisia tämän esimerkin kohdalla.

Jotta komponentista luotu paketti toimisi toisessa sovelluksessa suoraan sen asentamisen jälkeen, tulee Nodelle antaa ohjeet kaikkien npm-kirjastosta ladattujen pakettien asentamiseksi, joita komponentti käyttää. FeedbackForm-komponentin käyttämät moduulit näkyvät alla kuvassa 16.

```
import React, { useEffect } from 'react';
import {
  Button,
  FormControlLabel,
  FormLabel,
  Grid,
  Radio,
  RadioGroup
} from '@mui/material';
import { useState } from 'react';
import styled from '@emotion/styled';
import { validate } from 'validate.js';
```

KUVA 16. FeedbackForm-komponentin käyttämät moduulit.

Näiden pohjalta oli helppo lisätä tarvittavat moduulit komponentin package.json-tiedostoon. Kuvassa 17 näkyy paketin julkaisuvalmis package.json-tiedosto.

```
{
  "name": "@sami-p/feedbackform",
  "version": "1.0.0",
  "description": "",
  "main": "FeedbackForm.js",
  "dependencies": {
    "@emotion/react": "^11.5.0",
    "@emotion/styled": "^11.3.0",
    "@mui/material": "^5.1.0",
    "react": "^17.0.2",
    "validate.js": "^0.13.1"
  },
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

KUVA 17. Julkaisuvalmis package.json-tiedosto.

Ennen julkaisuvaihetta luotiin lisäksi .npmignore-tiedosto, joka ohjeisti Nodea ohittamaan tietyt tiedostot julkaisuvaiheessa. Tässä tapauksessa paketin ei tullut sisältää alkuperäistä TypeScript-tiedostoa, joten käskemme Nodea ohittamaan sen ja aikaisemmin luodun tsconfig.json-tiedoston. Esimerkki tiedostosta näkyy alla kuvassa18.

```
FeedbackForm.tsx
*.tgz
tsconfig.json
```

KUVA 18. .npmignore-tiedosto.

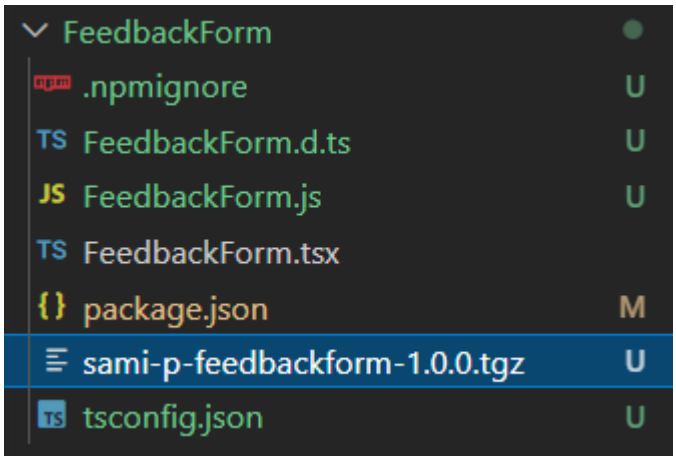
5.6 Paketin jakaminen sovellusta toiseen

Nyt paketti oli valmis jaettavaksi. Seuraavaksi käydään läpi kaksi erilaista tapaa paketin jakamiseen. Näistä tavoista jälkimmäinen on parempi suurille organisaatioille, tai jos pakettia halutaan käyttää useampaan kuin kahteen sovellukseen.

5.6.1 Tapa 1: npm pack

”npm pack” -komentoa käytetään usein pakettien testaukseen ennen julkaisua, mutta se toimii myös yksityiseen komponenttien jakamiseen omien sovellusten välillä. Normaalisti pakettia julkaistessa npm-kirjastoon Node ottaa komponentin tietoja kuten nimen ja versionumeron sen package.json-tiedostosta ja luo paketista .tgz-tiedoston. Tämä tiedosto sitten tallennetaan npm-kirjastoon muiden käyttäjien ladattavaksi. ”npm pack” -komento toimii samalla tavalla, mutta kirjastoon tallentamisen sijaan .tgz-tiedosto luodaankin projektikansioon kehittäjän laitteelle.

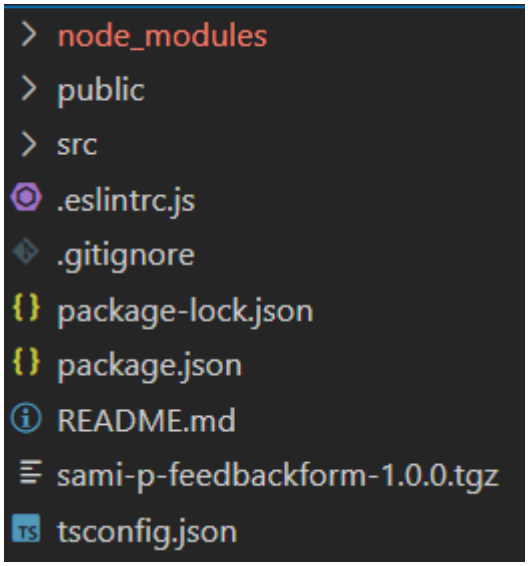
Komennon ajaminen FeedbackForm-komponentin kansiossa loi valmiiksi tällaisen tiedoston, kuten kuvasta 19 voidaan nähdä.



KUVA 19. ”npm pack” -komennon luoma .tgz-tiedosto.

Luotu tiedosto voidaan avata paketinhallintaohjelmilla kuten 7-Zip, ja sen sisältä löytyvät kaikki komponentin tiedostot. Tämä valmis .tgz-paketti sitten siirrettiin

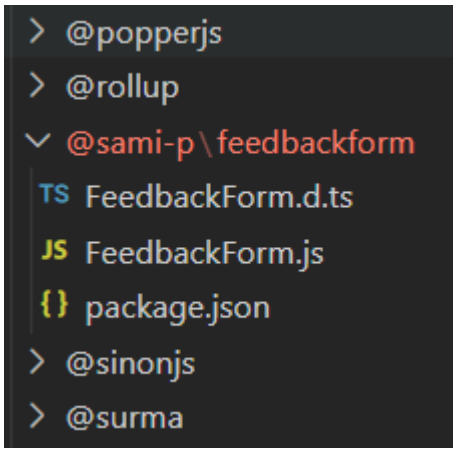
manuaalisesti Sovelluksen B juurihakemistoon ja asennettiin sieltä komennolla ”npm install ./sami-p-feedbackform-1.0.0.tgz”. Asennettu paketti ilmestyi sitten käytettäväksi node_modules-kansioon, aivan kuin tavallinen moduuli.



KUVA 20. Applikaation B tiedostorakenne.

```
apps\thesis-typescript-b> npm i ./sami-p-feedbackform-1.0.0.tgz
```

KUVA 21. Paketin asennuskomento.



KUVA 22. Asennettu paketti ”node_modules”-kansiossa.

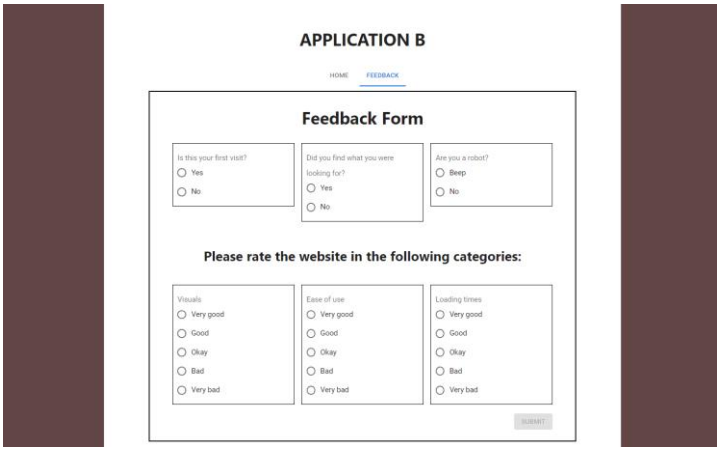
Siirrettyä komponenttia voitiin sitten käyttää uudessa sovelluksessa aivan kuin tavallista moduulia. Tämän projektin tapauksessa tämä tapahtui AppNavigation.tsx-tiedostossa. Jotta luotua komponenttia voitaisiin käyttää Applikaation B

komponenteissa, se tulee ”importata” (tuoda koodiin muuttujana tai funktiona) kuten alla kuvassa 23.

```
import styled from '@emotion/styled';
import { Box } from '@mui/system';
import FeedbackForm from '@sami-p/feedbackform';
```

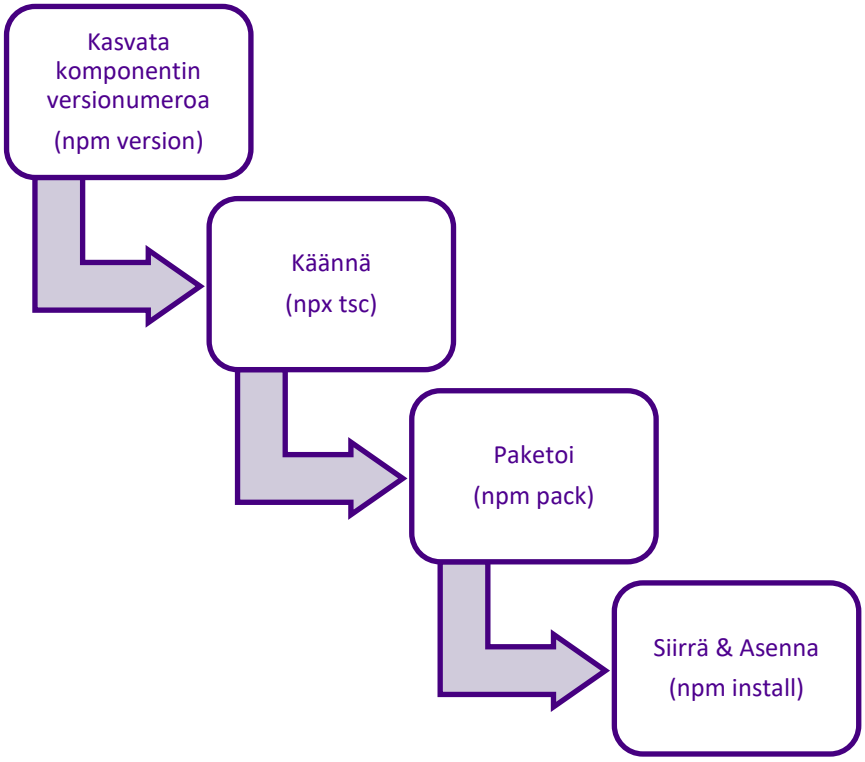
KUVA 23. Asennetun paketin kutsuminen koodissa.

Komponentin lisäämisen tuloksena palautesivu näkyi myös applikaatiossa B, kuten kuvasta 24 voi nähdä.



KUVA 24. Palautesivu applikaatiossa B.

Tulevaisuudessa palautesivun muutoksen tehdään alkuperäiseen applikaatio A:n FeedbackForm.tsx-tiedostoon, jonka jälkeen askeleet toistuvat seuraavassa järjestyksessä:



KAAVIO 3. Tapa 1 - Askeleet alkuperäisen komponentin muutosten jälkeen.

5.6.2 Tapa 2: Verdaccio

Kirjastoja yksityisen npm-paketin julkaisemiseen löytyy monia, kuten esimerkiksi npm:n oma kirjasto maksullisella Pro-käyttäjällä tai Amazon Web Services -pilvipalvelujen CodeArtifact. Tässä esimerkissä käytettiin kuitenkin kappaleessa 4 mainittua Verdaccio-pakettia.

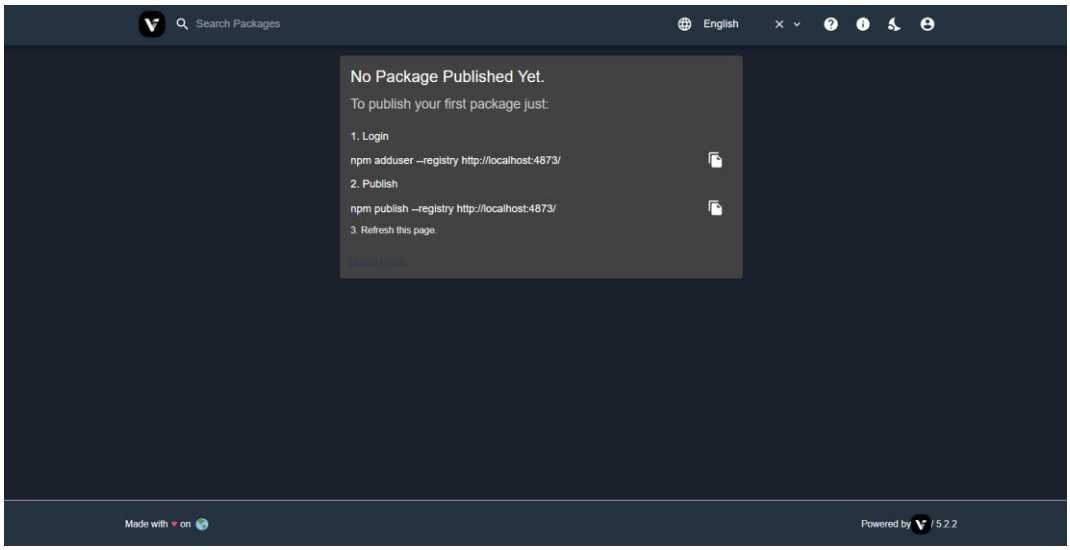
Ensimmäiseksi luotiin uusi projektikansio ja se nimettiin ”verdaccio”. Seuraavaksi tälle uudelle projektille luotiin oma package.json-tiedosto ajamalla komento ”npm init -y”. Tämän jälkeen Verdaccio-paketti voitiin asentaa käyttämällä ”npm install verdaccio” -komentoa. Ennen Verdaccion ajamista tuli vielä luoda uusi config.yaml-tiedosto, joka sisälsi Verdaccion asetukset. Tässä projektissa käytettiin Verdaccion dokumentaatiosta löytyvää oletuskonfiguraatiota, joka näkyy myös alla kuvassa 25.

```yaml
storage: ./storage
auth:
  htpasswd:
    file: ./htpasswd
uplinks:
  npmjs:
    url: https://registry.npmjs.org/
packages:
  "@*/*":
    access: $all
    publish: $authenticated
    proxy: npmjs
  "**":
    proxy: npmjs
logs:
  - { type: stdout, format: pretty, level: http }
```

KUVA 25. Oletusarvoinen config.yaml-tiedosto.

Pakettien julkaisemiseksi tuli Verdacciolle luoda valtuutettu käyttäjä. Tämä tehtiin luomalla salattu käyttäjänimi-salasana merkkijono käyttäen Gary Stevensin Htpasswd Generaattoria. Projektin juureen luotiin htpasswd-tiedosto (ei loppupäätettä), jonka sisään kopioitiin generoitu merkkijono.

Nyt Verdaccio oli valmis ajettavaksi. Jotta se osasi käyttää oikeaa konfiguraatiotiedostoa, tuli se osoittaa luotuun config.yaml-tiedostoon komentolinjan parametrinä. Komento Verdaccion käynnistämiseen oli siis tässä tapauksessa ”verdaccio --config ./config.yaml”. Verdaccio käynnistyi ja avasi paikallisen portin 4873. Kyseinen portti avattiin verkkoselaimessa osoitteella ”http://localhost:4873”. Verkkoselaimeen aukesi Verdaccion käyttöliittymä, joka näkyy alla kuvassa 26.



KUVA 26. Verdaccion käyttöliittymä.

Seuraavaksi alkuperäinen FeedbackForm-komponentti applikaatiosta A voitiin julkaista. Tätä varten piti komponentin package.json-tiedostoon ensin lisätä uusi asetus, joka varmistaa, että paketti julkaistaan Verdaccioon, ei npm-kirjastoon. Päivitetty package.json-tiedosto näkyy alla kuvassa 27, ja lisätyt rivit on reunustettu vihreällä.

```
  "main": "FeedbackForm.js",
  "publishConfig": {
    "registry": "http://localhost:4873"
  },
  "dependencies": {
    "@emotion/react": "^11.5.0",
    "@emotion/styled": "^11.3.0",
    "@mui/material": "^5.1.0",
```

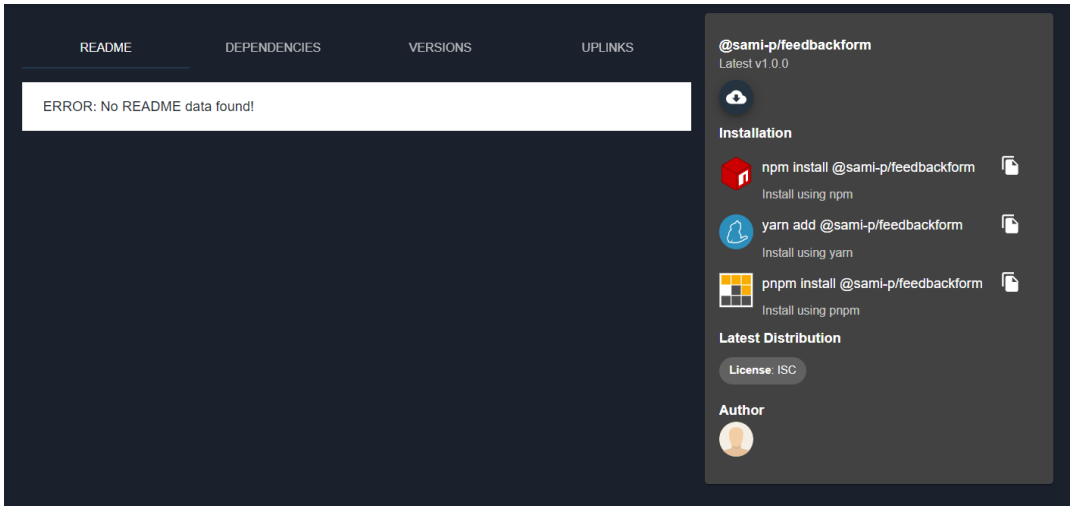
KUVA 27. package.json-tiedostoon lisätty sääntö.

Seuraavaksi aikaisemmin luodulle käyttäjälle kirjauduttiin komentolinjaa käyttäen ajamalla komento ”npm adduser --registry http://localhost:4873/” ja käyttämällä generaattorissa käytettyjä käyttäjätunnusta ja salasanaa. Nyt paketti voitiin viimein julkaista, joten prosessi käynnistettiin komennolla ”npm publish”. Komennon tulos näkyy alla kuvassa 28.

```
npm notice
npm notice 📦  @sami-p/feedbackform@1.0.0
npm notice === Tarball Contents ===
npm notice 192B   FeedbackForm.d.ts
npm notice 10.3kB FeedbackForm.js
npm notice 483B   package.json
npm notice === Tarball Details ===
npm notice name:          @sami-p/feedbackform
npm notice version:       1.0.0
npm notice filename:      @sami-p/feedbackform-1.0.0.tgz
npm notice package size:  3.1 kB
npm notice unpacked size: 10.9 kB
npm notice shasum:        605590a25aa2dcef8fb4c66e87a0756a237007eb
npm notice integrity:     sha512-Somef6frI14v+[...]lneWvInUYOwkA==
npm notice total files:   3
npm notice
+ @sami-p/feedbackform@1.0.0
```

KUVA 28. FeedbackForm-komponentin julkaisemisen tulos.

Nyt kun paketti oli julkaistu, se tuli näkyviin Verdaccion käyttöliittymässä. Paketin avaamalla voitiin tarkastella sen tietoja, kuten riippuvuuksia, versionumeroita ja lisenssiä. Paketin tietoja näkyy kuvassa 29.



KUVA 29. FeedbackForm-komponentti Verdaccion käyttöliittymässä.

Seuraavaksi tämä paketti piti saada käyttöön applikaatiossa B. Jotta applikaatio B osasi hakea julkaistun paketin oikeasta kirjastosta, tuli applikaatio B:n juureen luoda uusi .npmrc-tiedosto, jonka ainoaksi riviksi asetettiin ”registry = http://localhost:4873”. Tämä ohjeisti Nodea ohjaamaan kaikki asennuskomennot tähän uuteen kirjastoon npm-kirjaston sijaan. Mikäli haettua pakettia ei löydy Verdaccion kirjastosta, se osaa ohjata Noden toistamaan haun npm-kirjastossa. Luotu .npmrc-tiedosto alla kuvassa 30.

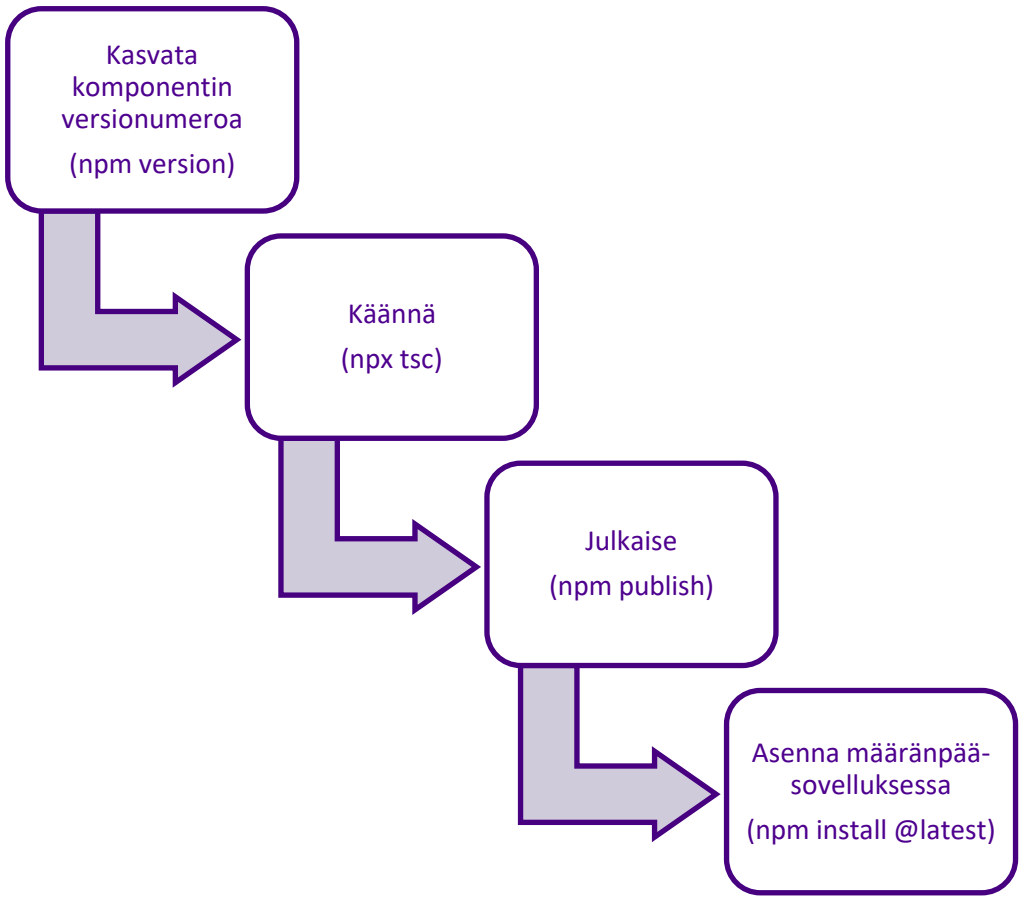
KUVA 30. Luotu .npmrc-tiedosto.

Enää tuli vain ajaa komento ”npm install @sami-p/feedbackform” applikaatio B:n juuressa. Node osasi hakea ja asentaa julkaistun paketin Verdaccion kirjastosta. Paketti ilmestyi myös applikaatio B:n package.json-tiedostoon, kuten kuvasta 31 voi nähdä.

```
"@emotion/styled": "^11.3.0",
"@mui/material": "^5.1.0",
"@sami-p/feedbackform": "^1.0.0",
"@testing-library/jest-dom": "^5.11.4",
"@testing-library/react": "^11.1.0",
```

KUVA 31. Julkaistu komponentti applikaatio B:n package.json-tiedostossa.

Nyt kun sovellukset ja Ferdaccio oli konfiguroitu yksityisten komponenttien käyttöön, uusien komponenttien julkaisu ja ylläpito oli helpompaa. Tällaisessa projektissa tulevaisuuden muutokset palautesivuun tehdään alkuperäiseen FeedbackForm.tsx-tiedostoon, jonka jälkeen toteutetaan askeleet kaavion 4 kuvaamassa järjestyksessä.



KAAVIO 4. Tapa 2 - Askeleet alkuperäisen komponentin muutosten jälkeen.

6 POHDINTA

Tämän työn idea lähti toimeksiannosta, jossa minun piti keksiä ratkaisu jatkuvasti kasvavan koodimäärän hillitsemiseen. Toimeksiantajalla oli usean sovelluksen sovellusverkko, jossa eri sovellukset on tarkoitettu eri tason käyttäjille. Sovellukset jakoivat paljon koodia, ja toimeksiantaja halusi sieventää koodikantaa ja löytää keinon uudelleen käyttää näkymiä sovellusverkon sisällä mahdollisimman kestävällä tavalla.

Tätä seurasi kuukausien pituinen projekti, jossa yhdistelin koodikantoja kuin palapelin paloja ja löysin viimein keinon pitää ne yhdessä: Node.js-paketit. Lisää aikaa kului parhaiden tapojen oppimiseen ja testaamiseen pakettien kanssa, ja lopulta sain aikaan ratkaisun, johon olen edelleen tyytyväinen. Vaikka tutkin asiaa todella pitkällä aikavälillä, opin silti paljon uutta tämän työn kirjoittamisen aikana. Haluankin kiittää toimeksiantajaa joustavuudesta ja kärsivällisyydestä minun ja työni suhteen.

LÄHTEET

Thapaliya, I. 2016. Web and Cross-Platform Mobile application sharing same code base using modern web technologies. Luettu 11.10.2021. https://www.theseus.fi/bitstream/handle/10024/118278/Thapaliya_Ishwor.pdf?sequence=1&isAllowed=y

Bhammarker, R. 2017. Understanding npm dependency resolution. Luettu 12.10.2021. https://medium.com/learnwithrahul/understanding-npm-dependency-resolution-84a24180901b

replicate.npmjs.com. n.d. 'total_rows'-laskuri. Viitattu 12.10.2021. https://replicate.npmjs.com/_all_docs

Denicola, D. 2013. Peer Dependencies. Luettu 12.10.2021. https://blog.domenic.me/peer-dependencies/

Palmer, T. 2019. npm Peer Dependencies. Luettu 12.10.2021. https://in-depth.dev/posts/1187/npm-peer-dependencies

flavioscopes.com. 2019. What are peer dependencies in a Node module? Luettu 12.10.2021. https://flaviocopes.com/npm-peer-dependencies/

Ström, C. 2019. React pähkinänkuoressa. Luettu 12.10.2021. https://joinex.fi/react-pahkinankuoressa/

insights.stackoverflow.com. 2021. Stack Overflow Developer Survey 2021. Viitattu 12.10.2021. https://insights.stackoverflow.com/survey/2021#most-popular-technologies-webframe

docs.npmjs.com. n.d. package.json. Luettu 12.10.2021. https://docs.npmjs.com/cli/v7/configuring-npm/package-json

heynode.com. n.d. What Is package.json? Luettu 12.10.2021. https://heynode.com/tutorial/what-packagejson/

Ombui, V. 2021. How to Set up a Node.js Express Server for React. Luettu 13.10.2021. https://www.section.io/engineering-education/how-to-setup-nodejs-express-for-react/

expressjs.com. n.d. Express-dokumentaatio. Luettu 15.10.2021. https://expressjs.com

create-react-app.dev. n.d. Create React App -dokumentaatio. Luettu 15.10.2021. https://create-react-app.dev/docs/getting-started

learn.co. n.d. React Modular Code. Luettu 19.19.2021. https://learn.co/lessons/react-modular-code

Schwane, J. 2017. Writing Modular JavaScript — Pt 1. Luettu 10.11.2021. https://medium.com/@jrschwane/writing-modular-javascript-pt-1-b42a3bd23685

typescriptlang.org. n.d. Why TypeScript. Luettu 10.11.2021. https://www.typescriptlang.org/why-create-typescript

Vitullo, C. 2018. Testing npm packages before publishing. Luettu 14.11.2021. https://medium.com/@vcarl/problems-with-npm-link-and-an-alternative-4dbdd3e66811

dasmikko. n.d. Verdaccio documentation. Luettu 15.11.2021. https://verdaccio.org/docs/what-is-verdaccio

LIITTEET

Liite 1. Projektin GitHub.