VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Krista Rintala

REACTJS SOVELLUSTUOTANNOSSA

Case Task list

Liiketalous
2023

VAASAN AMMATTIKORKEAKOULU
Tietojenkäsittely

TIIVISTELMÄ

Tekijä	Krista Rintala
Opinnäytetyön nimi	ReactJS sovellustuotannossa
Vuosi	2023
Kieli	suomi
Sivumäärä	33
Ohjaaja	Päivi Rajala

ReactJS on nykyään yksi suosituimmista JavaScript-kirjastoista, jota käytetään laajalti verkkosovellusten kehittämisessä. Tämän opinnäytetyön tavoitteena on tutkia ReactJS-kirjaston keskeisiä ominaisuuksia ja sen käyttöä. Teoksessa selvitetään Reactin peruskonsepteja, kuten komponenttipohjaista lähestymistapaa ja niiden toimintaa, virtuaalista DOM-mallia, JSX-syntaksia ja tilan hallintaa. Opinnäytetyössä esitellään myös Reactin käyttöönottoa ja käytännönosan kautta komponenttien toimintaa ja niiden välisiä suhteita.

Käytännön osassa luodaan yksinkertainen tehtävälista-sovellus, jossa käytetään funktiokomponentteja ja tilan hallintaan React Hookseja. Tehtävälista-sovelluksessa voidaan lisätä, muokata, poistaa ja merkitä tehtäviä suoritetuksi.

Opinnäytetyön lopputuloksena syntyi tietopaketti Reactin keskeisistä ominaisuuksista ja sen käytöstä sovelluksen toteutuksessa. Lisäksi tuloksena oli tehtävälistasovelluksen demo-versio, jossa hyödynnettiin ReactJS-kirjastoa käytännön sovelluksessa.

Avainsanat React, JSX, Komponentti, Tila

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Business Information Technology

ABSTRACT

Author	Krista Rintala
Title	ReactJS Application Development
Year	2023
Language	Finnish
Pages	33
Name of Supervisor	Päivi Rajala

ReactJS is one of the most popular JavaScript-libraries, which is used broadly in app development. The goal of this thesis was to explore the main features of ReactJS and how to use it. The thesis investigated the main concepts of ReactJS like component-based approach, virtual DOM model, JSX-syntax, and state management. The thesis also presents the introduction of ReactJS, and how components work and as well their relations through the practical section of the thesis.

In the practical section of the bachelor's thesis a simple task list-application was created, in which functional components were used and for the state management React Hooks were used. In the task list application, it is possible to add, edit or delete tasks and as well mark them as completed.

The end result of this thesis delivered an information package of the main features of ReactJS and its usage in application development. Another result was also the task list application demo-version which utilized ReactJS library in a practical application.

Keywords React, JSX, Component, State

SISÄLLYS

KUVALUETTELO

1 JOHDANTO

React, jota kutsutaan myös nimellä ReactJS tai React.js on nykyään yksi suosituimmista JavaScript-kirjastoista, jota käytetään laajalti verkkosovellusten kehittämisessä. React helpottaa interaktiivisten ja reaktiivisten käyttöliittymien rakentamista ja hallintaa pilkkomalla käyttöliittymän uudelleen käytettäviin osiin.

Opinnäytetyön aihe valittiin sen ajankohtaisuuden vuoksi sekä omasta kiinnostuksesta oppia aiheesta. Opinnäytetyön tavoitteena on tutustua ja esitellä Reactin keskeiset ominaisuudet ja luoda yksinkertainen käyttöliittymä, jonka tarkoituksena on selventää teoriaosuudessa läpikäytyjä käsitteitä. Opinnäytetyön tuloksena syntyy tietopaketti Reactin keskeisistä ominaisuuksista ja sen käytöstä.

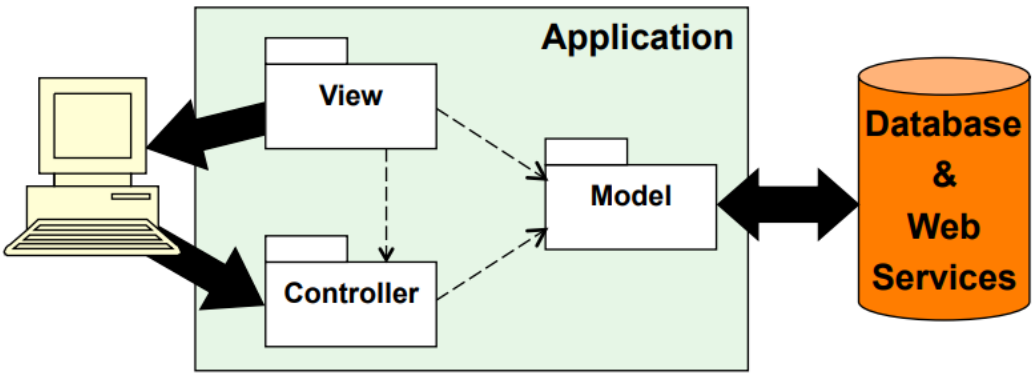
Tämä opinnäytetyö on empiirinen, laadullinen tapaustutkimus, joka toteutetaan projektina. Opinnäytetyössä vastataan tutkimuskysymykseen, miten rakennetaan sovellus käyttäen ReactJS-kirjastoa ja mitkä ovat sen keskeiset ominaisuudet?

Luvussa kaksi kerrotaan mikä React on ja sen hyödyistä. Luvussa kolme käsitellään Reactin keskeisiä ominaisuuksia, kuten virtuaalista DOM-mallia, komponentteja ja JSX-syntaksia. Luvussa neljä luodaan tehtävälista-sovellus, jossa tarkastellaan komponenttien toimintaa ja luvussa viisi käsitellään opinnäytetyön tuloksena syntynyttä tehtävälistasovellusta. Luku kuusi sisältää yhteenvedon.

2 REACTJS

React on Facebookin (nykyisin Meta) kehittämä ja ylläpitämä avoimen lähdekoodin front-end Javascript-kirjasto käyttöliittymien luomiseen. React kehitettiin vuonna 2011 ja on sittemmin noussut yhdeksi suosituimmista Javascript-kirjastoista. React sai alkunsa, kun Facebook Ads-sovellus sai jatkuvasti lisää ominaisuuksia ja kehittäjät alkoivat kohtaamaan ongelmia koodin hallinnan kanssa. (Schwarzmüller 2022; Hámori 2022.)

React vastaa näkymäosaa (view) Model-View-Controller (MVC) arkkitehtuurissa (kuva 1). Se ottaa vastaan tietoa ja esittää sen käyttäjälle. React sovellus koostuu uudelleen käytettävistä komponenteista, joita pystyy yhdistämään. Tämä uudelleen käytettävä koodi helpottaa sovelluksen kehitystä ja ylläpitoa. React käyttää virtuaalista DOM-mallia päivittääkseen DOMin. Virtuaalisen DOM-mallin käyttäminen on nopeampaa, sillä se päivittää vain tarvittavat muutokset DOMiin. (JavaTpoint 2023a; Minnick 2022.)



Kuva 1. Model-View-Controller arkkitehtuuri (Helsingin yliopisto 2023).

Reactia käyttää monet suuret yritykset kuten Facebook, Instagram, Reddit ja Netflix. (Minnick 2022.)

2.1 Deklaratiivisuus

React käyttää deklaratiivista lähetymistapaa imperatiivisen sijaan. Reactin deklaratiivinen lähestymistapa helpottaa monimutkaisten interaktiivisten ja reaktiivisten käyttöliittymien rakentamista ja ylläpitoa, sillä deklaratiivisessa tavassa ei tarvitse määrittää kaikkia yksittäisiä vaiheita. Sopivat DOM-ohjeet jäävät siis Reactin päätettäväksi. Imperatiivisessa tavassa täytyy määritellä ja kirjoittaa jokainen vaihe mitä selaimen tulee tehdä. (Schwarzmüller 2022.)

Minnick 2022 kuvaa imperatiivisen ohjelmoinnin esimerkin kautta, jossa halutaan ohjeilmoida robotti tekemään voileivän. Vaiheet tulisi määritellä seuraavilla tavalla:

- 1. Hae leipää.
- 1a. Käytä visuaalista sensoria paikantaaksesi leivän.
- 1aa. Jos leipää löytyy, liiku sitä kohti.
- 1ab. Jos leipää ei löydy, palaa ensimmäiseen vaiheeseen.
- 1b. Käytä tarttumisvartta avataaksesi leipäpussin.

Deklaratiivisessa ohjelmoinnissa taas tarvitsee vain kertoa mitä pitää tehdä, eikä miten tehdään. Kyseinen voilepää valmistava robotti tietäisi voileivän valmistusvaiheet, joten sitä ohjelmoidessa tarvitsisi kertoa vain, että ”tee voileipä, joka näyttää tältä”. Deklaratiivisessa lähestymistavassa ei siis tarvitse huolehtia teknisistä yksityiskohdista, vaan kehittäjä voi keskittyä haluttuun lopputilaan. (Minnick 2022; Schwarzmüller 2022.)

2.2 Yhden sivun sovellus

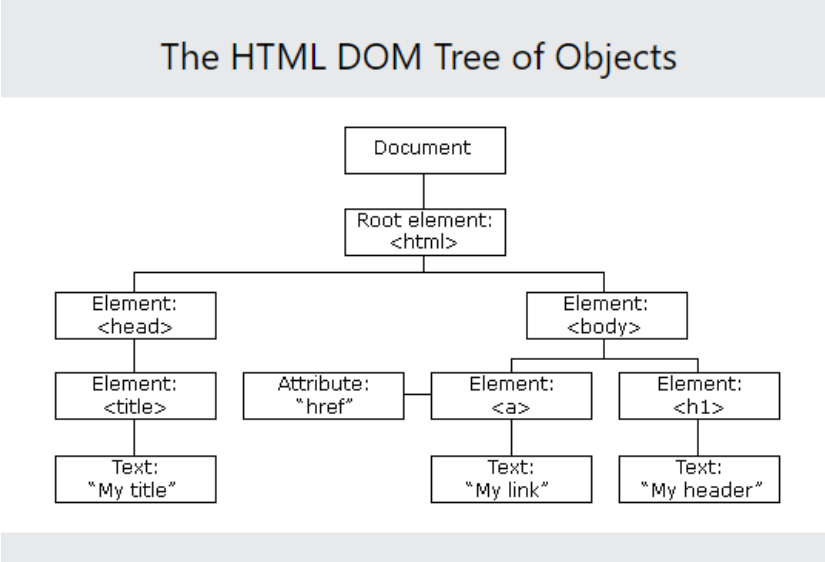
Yhden sivun sovellus eli Single Page Application (SPA) on ohjelmointitapa, jossa kaikki tiedot ja toiminnot ladataan kerralla selaimeen yhtenä dokumenttina. Näitä kaikkia tietoja ja toimintoja ei kuitenkaan näytetä käyttäjälle kerralla. Kun sivulla navigoidaan, selain uudelleen renderöi sisällön käyttäjälle päivittämättä sivustoa. SPA tekee käyttökokemuksesta paljon sulavampaa verrattaen monen sivun

sovellukseen (Multi Page Application), jossa navigoidessa sivu latautuu aina uudestaan. Reactilla on mahdollista rakentaa yhden- ja monen sivun sovelluksia. (Bolmér 2021; Halme 2018; Mozilla 2023b.)

3 REACTIN KESKEISET OMINAISUUDET

3.1 Virtuaalinen DOM

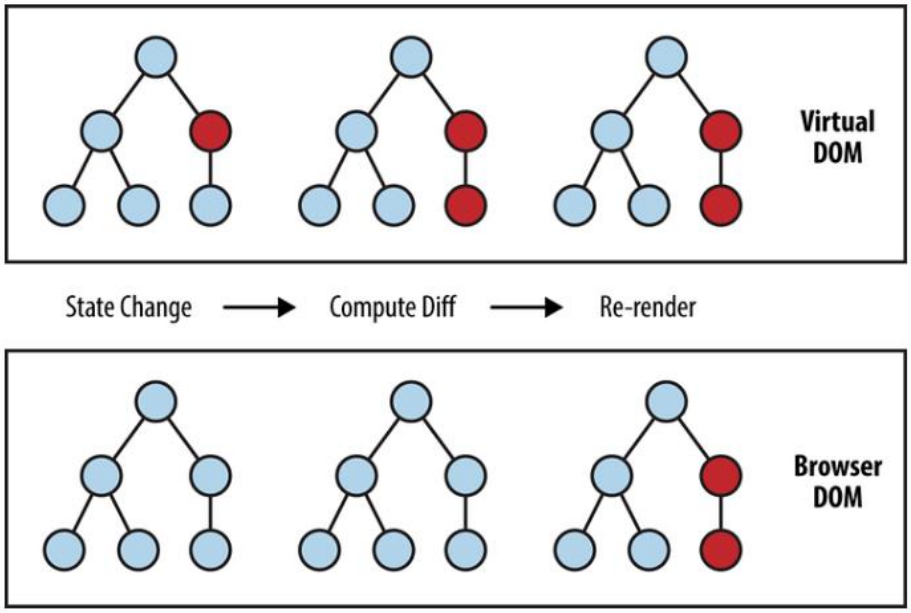
Document Object model (DOM) on objektipohjainen web-dokumenteille käytettävä ohjelmointirajapinta (Application programming interface, API). DOM on selaimen sisäinen esitys verkkosivusta. Kun verkkosivu ladataan, selain luo tästä DOM-puun (kuva 2), jossa määritellään HTML-elementit objekteiksi. Tämän avulla ohjelmointikielet pystyvät vuorovaikuttamaan sivun kanssa antaen ohjelmointikielille mahdollisuuden muokata sivun rakennetta, sisältöä ja tyyliä. (Mozilla 2023a; W3schools 2023a.)



Kuva 2. HTML DOM-puu (W3schools 2023a).

Aina kun käyttöliittymän tila muuttuu, DOM päivitetään vastaamaan käyttöliittymän muutosta. Muutosten päivittäminen DOMiin on hidasta ja tehotonta, etenkin jos kyseessä on monimutkainen käyttöliittymä. Tämä johtuu siitä, että DOM saattaa suorittaa tarpeettomia päivityksiä tarpeellisten lisäksi. (Minnick 2022; Mardan 2017; JavaTpoint 2023a.)

Tehostaakseen käyttöliittymien toimintaa React käyttää Virtuaalista DOM-mallia (VDOM). Virtuaalisessa DOM-mallissa React rakentaa virtuaalisen DOM-puun, joka sijaitsee muistissa. Muutosten tapahtuessa Virtuaalinen DOM vertaa itseään oikeaan DOM-malliin ja päivittää vain tarpeelliset muutokset. (Narayn 2022; Mardan 2017.)

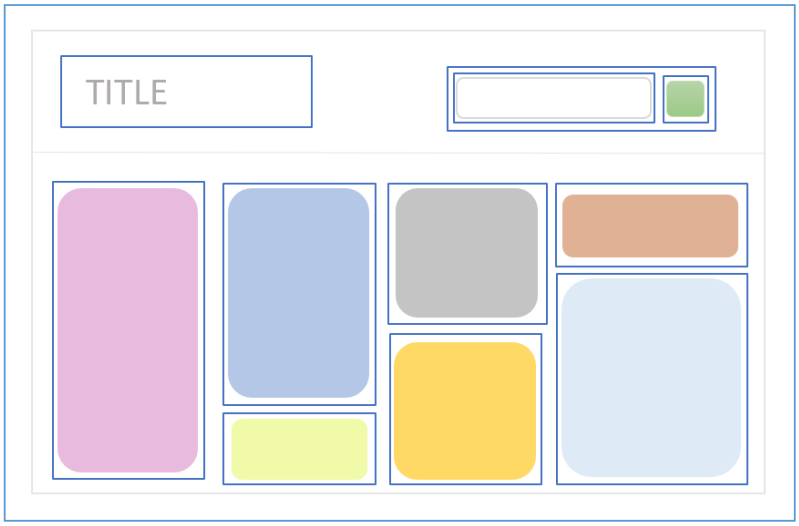


Kuva 3. Virtuaalinen DOM (Ravichandran 2019).

Tarkemmin kuvailtuna, aina kun käyttöliittymän komponentin tila muuttuu, React luo uuden Virtuaalisen DOM-puun ja vertaa eroja uuden ja edellisen Virtuaalisen DOMin välillä. Tämän jälkeen Virtuaalinen DOM laskelmoi parhaimman tavan tehdä muutokset oikeaan DOMiin, jonka jälkeen React päivittää vain ne objektit jotka ovat muuttuneet oikeaan DOMiin. Kyseistä prosessia kutsutaan nimellä differentiation (kuva 3). (Ravichandran 2019.)

3.2 Komponentit

React-komponentit ovat itsenäisiä ja uudelleen käytettäviä rakennuspalikoita, joita yhdistämällä muodostuu lopullinen käyttöliittymä. Komponentti voi olla yksinkertainen painike tai monimutkaisempi navigointipalkki, joka koostuu useasta painikkeesta. Kuvassa 4, rajatut osat kuvastavat yhtä komponenttia sivulla. (Minnick 2022.)



Kuva 4. Kuvaus komponenteista sivulla.

Komponentit, jotka renderöidään toisten komponenttien sisällä kutsutaan lapsikomponenteiksi (child component), kun taas komponentit joiden sisällä nämä renderöidään kutsutaan emokomponentiksi (parent component). Lapsikomponentti voi myös toimia emokomponenttina toiselle lapsikomponentille. Juurikomponentti on tämän hierarkian ylimmällä tasolla ja se renderöidään Domiin. Näin ollen se ei voi toimia lapsikomponenttina. (Meta Open Source 2023; Minnick 2022; Schwarzmüller 2022.)

Komponentin voi yhdistää käyttämällä export default JavaScript syntaksia komponentin sisällä, joka mahdollistaa viennin toisiin komponentteihin. Tämän jälkeen komponentti pystyy sisällyttämään toiseen import toiminnolla. Tällöin komponenteille syntyy lapsi-vanhempi suhde (kuva 5). (Meta Open Source 2023b.)

React komponentti voi olla joko luokkakomponentti (Class component) tai funktiokomponentti (Function component). Luokkakomponentti ohjaa komponentin logiikan toteutusta ja tilan muutoksia. Sillä on myös pääsy elinkaarimetodin eri vaiheisiin. Funktiokomponentti on yksinkertaisesti JavaScript-funktio, joka ei itsessään sisällä tila tai elinkaarimetodi ominaisuuksia. Funktiokomponentti vain vastaanottaa ja palauttaa tietoa DOMiin renderöitäväksi. Tämä kuitenkin muuttui 16.8 versiopäivityksen myötä, jossa otettiin käyttöön uusi ominaisuus nimeltä Hooks. Hooks mahdollistaa tila- ja elinkaarimetodien käyttämisen funktiokomponenteissa. (Abiodun 2021; Minnick 2022.)

```
import Gallery from './Gallery.js';

function App() {
  return (
    <Gallery />
  );
}
export default App;
```

```
function Gallery() {
  return (
   <img
      src="https://i.imgur.com/QIrZWGIs.jpg"
      alt="Alan L. Hart"
    />
  );
}
export default Gallery;
```

Kuva 5. Komponenttien yhdistäminen (Meta Open Source 2023b).

Komponentit käyttävät return-metodia palauttamaan sen, mitä halutaan käyttöliittymään. Jos elementit, joita palautetaan eivät ole samalla rivillä, tulee ne kääriä return-metodin jälkeen kaarisulkeisiin. (Meta Open Source 2023e.)

3.2.1 Properties

Properties eli props on objekti, joka tekee komponenteista uudelleen käytettäviä. Propseja käytetään tiedonvälitykseen emo- ja lapsikomponenttien välillä. Propsin arvo voi olla mikä tahansa JavaScript-lauseke tai tietotyyppi ja se on vain luettavissa olevaa tietoa. Komponentit voivat vastaanottaa rajattoman määrän propseja ja ne ovat muuttumattomia, joten niitä ei voi muokata komponentin sisällä. (Minnick 2022; javaTpoint 2023c.)

```
function ParentComponent() {
  return <ChildComponent name="Pekka" />;
}

const ChildComponent = (props) => {
  return <p>{props.name}</p>;
};
```

Kuva 6. Esimerkki tiedon välityksestä.

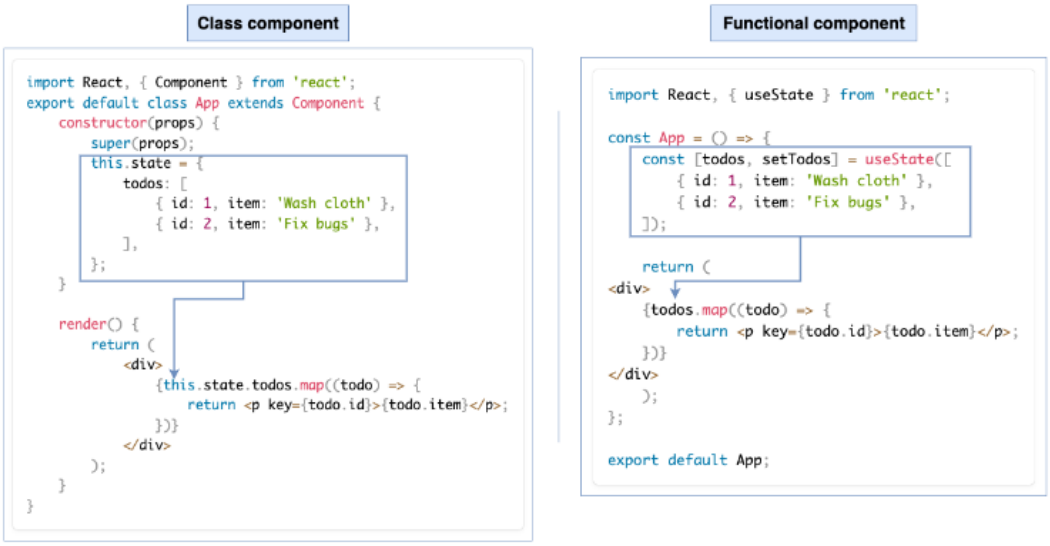
Kuvassa 6 on esimerkki siitä miten voidaan välittää tietoa käyttäen propseja. Emokomponenttiin on sisällytetty lapsikomponentti, jossa ”name” on määritetty propsiksi ja sen arvo on ”Pekka”. Tämän jälkeen propsi välitetään lapsikomponentille ja lopuksi välitetään elementille, joka lopulta renderöidään. (Eygi 2020.)

3.2.2 Komponentin tila

Tila (State) on komponentin sisäänrakennettu objekti, johon varastoidaan tietoa komponentista. Tilan avulla komponentit voivat luoda ja hallita omia tietojaan sisäisesti. Komponentin tila voi muuttua ajan myötä, esimerkiksi käyttäjän antaman syötteen tai järjestelmä tapahtuman myötä.

Komponentit jotka sisältävät tilan, renderöidään tilan tietojen perusteella. Kun tila muuttuu, React renderöi uudelleen muutoksen välittömästi DOMiin. React ei päivitä kuitenkaan koko DOMia vaan vain komponentin, jonka tila on muuttunut. (W3schools 2023c; javaTpoint 2023d; Eygi 2020.)

Luokkakomponentissa tilan määrittämiseksi täytyy ensin määrittää komponentin alkutila käyttäen this.state-ominaisuutta. Komponentin tilan pystyy päivittämään setState()-metodilla, kun metodia kutsutaan käyttöliittymä päivittyy. Funktiokomponenteissa tilaa hallitaan React Hooksien avulla. Se mahdollistaa tilan määrittämisen ilman että tarvitsee luoda erillistä luokkakomponenttia. Kuvassa 7 nähdään luokkakomponentin ja Hooksia käyttävän funktiokomponentin tilanmäärittelyn ero. (JavaTpoint 2023d.)



Kuva 7. Luokka- ja funktiokomponentin tilan määrittely (Olawanle 2022).

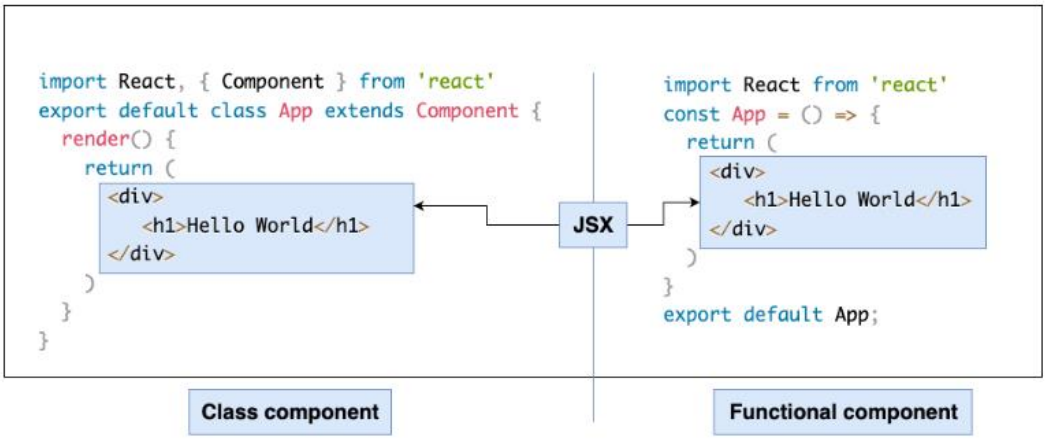
Funktiokomponentissa tila määritellään useState()-Hookilla. Kuvassa 7 ensin tilalle annetaan arvo ”todos” ja sen jälkeen ”setTodos” funktio, jota kutumalla tilan

pystyy päivittämään. Näiden kahden elementin nimen valitseminen on täysin kehittäjän päätettävissä. (Schwarzmüller 2022.)

3.3 JSX-syntaksi

JSX on JavaScriptin syntaksilaajennus, jota käytetään komponenttien luontiin Reactissa. Sen avulla määritetään miltä käyttöliittymä tulisi näyttää. React ei itsessään vaadi JSX:n käyttöä, mutta useimmiten kehittäjät käyttävät sitä, sillä kokevat sen käytön hyödylliseksi. (Meta Open Source 2023d; Joinex 2019.)

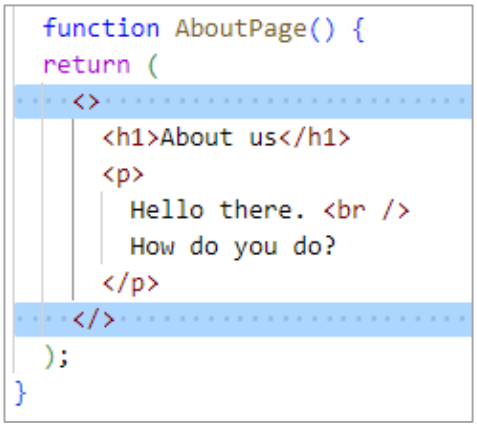
Pitkän aikaa kehittäjät ovat pitäneet sivun tyylin, sisällön ja logiikan omissa erillisissä tiedostoissa, Reactissa nämä yhdistyvät JSX:n avulla yhdessä samassa paikassa Reactin komponenteissa. JSX siis mahdollistaa HTML-tyyppisen kielen kirjoittamisen suoraan JavaScriptiin. Vaikka JSX muistuttaa hyvin paljon HTML-kieltä, on se tapa kirjoittaa JavaScriptia (kuva 8).



Kuva 8. Luokka- ja funktiokomponentti JSX (Olawanle 2022).

JSX on säännöiltään tiukempi kuin HTML, se vaatii esimerkiksi elementin sulkemisen ja jos komponenttiin halutaan sisällyttää enemmän kuin yksi elementti, tulee ne kääriä kontti elementtiin (container element) (kuva 9). JSX-

syntaksissa käytetään ”className” ominaisuutta, kun taas HTMLssä ”class” ominaisuutta. (React Docs 2023; Meta Open Source 2023d; Houston Inc.; JavaTpoint 2023b.)



Kuva 9. JSX kontti elementti.

JSX-syntaksilla pystyy kirjoittamaan lausekkeita, jotka voivat olla React-muuttujia, merkkijonoja, numeroita tai mikä tahansa muu kelvollinen JavaScript-lauseke. Lausekkeet kirjoitetaan aaltosulkeiden sisään. (Meta open Source 2023d; W3schools 2023e.)

Selain ei osaa lukea moderneja JavaScript kieliä kuten JSX:ä, joten se tulee kääntää selaimelle luettavaan muotoon käyttämällä Babel-kääntäjää. Babel muuntaa JSX-koodin selaimelle ymmärettäväksi JavaScript versioksi. (Narayn 2022.)

3.4 Tapahtumakäsittelijät

React mahdollistaa tapahtumakäsittelijöiden (event handlers) lisäämisen JSX-syntaksiin. Ne ovat omia funktioita, jotka aktivoituvat vastauksena interaktioon kuten hiiren klikkaukseen, lomakkeen tekstikenttään keskittämiseen tai näppäinpainallukseen. Tapahtumien käsittely antaa käyttäjälle mahdollisuuden vuorovaikuttaa sivun kanssa. Reactin tapahtuman käsittely on samankaltainen

kuin HTMLssä, mutta pienillä muutoksilla. React käyttää tapahtumakäsittelyssä camelCase-syntaksia, kun taas HTML lowercase-syntaksia. (Meta Open Source 2023a.)

```
export default function Button() {
  function handleClick() {
    alert('You clicked me!');
  }

  return (
    <button onClick={handleClick}>
      Click me
    </button>
  );
}
```

Kuva 10. Tapahtumakäsittelijän lisääminen (Meta Open Source 2023).

Tapahtumakäsittelijän lisäämisessä täytyy ensin määritellä funktio ja välittää se propsina sopivalle elementille. Kuvassa 10, komponentti renderöi painikkeen, jota klikkaamalla ilmestyy viesti ”You clicked me!”. Komponentissa määritellään ensin funktio nimeltä handleClick, jonka jälkeen se välitetään <button> elementtiin. Tapahtumakäsittelijä funktiot alkavat nimellä ”handle”, jonka jälkeen tulee tapahtuman nimi. (Meta Open Source 2023c.)

3.4.1 Komponentin elinkaari

React komponentilla on kolme elinkaarta, jotka ovat komponentin synty (mounting), päivittäminen (updating) ja tuhoutuminen (unmounting). Maranan (2022) luettelee elinkaarien merkitykset seuraavanlaisesti:

- **Mounting –** Synty tapahtuu kun komponentti luodaan ja sijoitetaan DOMiin. Syntyä kutsutaan alustavaksi renderöinniksi ja se voi tapahtua vain kerran.
- **Updating –** Päivitys on vaihe, jossa komponentti renderöidään uudelleen tai päivitetään. Tämä tapahtuu silloin kun komponentin tila taikka propsit päivitetään.
- **Unmounting –** Tuhoutuminen tapahtuu kun komponentti poistetaan DOMista. (Maranan 2022.)

Luokkakomponentissa elinkaaren vaiheisiin voi kutsua eri elinkaaritoimintoja. Elinkaaritoimintoja joita voi määrittää komponentin synty vaiheessa ovat constructor, render, getDerivedStateFromProps ja componentDidMount. Constructor-metodia käytetään yleisemmin tapahtumakäsittelijöiden sitomiseen ja komponentin tilan alustamiseen. Render on ainoa luokkakomponentille vaadittu metodi. Se palauttaa yleensä JSX-syntaksin ja lopulta renderöidään. getDerivedStateFromProps-metodia kutsutaan ennen kuin elementit renderöidään DOMiin. Viimeinen elinkaarimetodi, jota kutsutaan elinkaaren synty vaiheessa on ComponentDidMount. Kyseisellä metodilla voi lisätä sivuvaikutuksia, kuten komponentin tilan päivitys.

Päivitys vaiheen metodeja ovat getDerivedStateFromProps, shouldComponentUpdate, render, getSnapshotBeforeUpdate ja componentDidUpdate. ShouldComponentUpdate-metodin avulla voi määrittää, että pitäisikö Reactin jatkaa renderöintiä vai ei. Pääsy komponentin aikaisempaan tilaan ja propseihin ennen päivittämistä on mahdollista getSnapshotBeforeUpdate-metodilla. ComponentDidUpdate-metodia kutsutaan, kun komponentti on päivitetty DOMiin.

Komponentin tuhoutumis vaiheen metodi on componentWillUnmount. Tätä metodia kutsutaan, kun komponentti poistetaan DOMista. (Maranan 2022; W3schools 2023.)

Funktiokomponenteissa käytetään useEffect Hookia, joka antaa mahdollisuuden suorittaa sivuvaikutuksia. UseEffect Hook ei sisällä luokkakomponentin elinkaaritoimintoja, vaan on vastine componentDidMount-, componentDidUpdate- ja componentWillUnmount-metodeille. (Maranan 2022.)

3.5 React ja tyyliohjekieli CSS

Tyylien kirjoittamiseen käyttäen CSS-tyyliohjekieltä Reactissa on monta eri tapaa. Tapoja ovat esimerkiksi Inline Styling ja CSS stylesheet. Inline Styling tavassa kirjoitetaan tyyli suoraan elementtiin kahden aaltosulkeen sisään (kuva 11).

```
<h1 style={{ color: "Green" }}>About us</h1>
```

Kuva 11. Inline Styling.

CSS stylesheet tapaa käytettäessä luodaan erillinen tiedosto, johon määritellään CSS tyylit. Tiedosto tallennetaan .css-päätteellä ja tuodaan sovellukseen. (W3schools 2023d.)

Reactille on myös saatavilla komponenttikirjastoja, jotka auttavat nopeuttamaan ohjelmistojen kehitystä. Komponenttikirjastot sisältävät valmiita komponentteja, kuten painikkeita, kaavioita, taulukoita, karttoja tai värejä. Kyseisiä komponentteja kehittäjä voi muokata oman tarpeensa mukaan. Komponenttikirjastoja ovat muun muassa Material-UI (MUI), React Bootstrap, Ant Desing (Ant) ja Chakra UI. (Acharya 2023.)

4 TEHTÄVÄLISTASOVELLUKSEN TOTEUTUS

Tässä kappaleessa luodaan yksinkertainen tehtävälistasovellus, jonka kautta tutustutaan Reactin käyttöön ja sen keskeisiin ominaisuuksiin. Tehtävälista pitää sisällään tehtävän lisäys-, muokkaus- ja poisto-toiminnot. Lisäksi tehtävän tulee pystyä merkitsemään tehdyksi, klikkaamalla tehtävän tekstiä.

Verkkosovelluksen ohjelmointiin käytettiin Visual Studio Codea, Node.js JavaScript-tulkkia ja npm-paketinhallintaa. Toteutuksessa käytettiin funktiokomponentteja ja siitä käyvät ilmi komponenttien hierarkia, sekä React useState Hooksien ja propsien käyttö.

4.1 Reactin käyttöönotto

Reactin käyttöönotto aloitettiin asentamalla Node.js ja npm-paketinhallinta. Seuraavaksi luotiin uusi kansio nimeltä ”Opinnäyte”, johon siirrytään terminaalissa cd-komennolla. Tämän jälkeen terminaalissa suoritetaan komento, npx create-react-app task-list-app, joka on nopea tapa aloittaa React-projekti. Komento luo React projektin ja konfiguroi sekä asentaa tarvittavat paketit automaattisesti ja antaa projektin nimeksi task-list-app (kuva 12).

```
npx create-react-app task-list-app
```

Kuva 12. React projektin luonti komento.

Tämän jälkeen React sovellus käynnistetään terminaalissa siirtymällä cd task-list-app komennolla projektin kansioon ja suorittamalla siellä npm start. Sovellus aukeaa selainikkunaan osoitteella localhost:3000.

4.2 Tehtävälistan toteutus

Sovellus koostuu TaskForm-, Task- ja TaskList-komponenteista, sekä App-juurikomponentista. TaskForm-komponentti (kuva 13) vastaa tehtävien lisäämisestä ja muokkaamisesta. Se sisältää lomakkeen ja tekstikentän, johon syötetään uusi tehtävä ja sitä kautta lisätään tehtävälistaan.

TaskForm.js

src > components > TaskForm.js > ...

```
1  import React, { useState } from "react";
2
3  function TaskForm(props) {
4    const [value, setValue] = useState(props.edit ? props.edit.value : "");
5
6    const handleChange = (e) => {
7      setValue(e.target.value);
8    };
9
10   const handleSubmit = (e) => {
11     e.preventDefault();
12
13     props.onSubmit({
14       id: Math.floor(Math.random() * 10000),
15       text: value,
16     });
17     setValue("");
18   };
19
20   return (
21     <form onSubmit={handleSubmit} className="task-form">
22       {props.edit ? (
23         <>
24           <input
25             placeholder="Muokkaa tehtävä"
26             value={value}
27             onChange={handleChange}
28             name="text"
29             className="task-input edit"
30           />
31           <button className="task-button edit">Päivitä</button>
32         </>
33       ) : (
34         <>
35           <input
36             placeholder="Lisää uusi tehtävä"
37             value={value}
38             onChange={handleChange}
39             name="text"
40             className="task-input"
41           />
42           <button className="task-button">Lisää</button>
43         </>
44       )}
45     </form>
46   );
47 }
48
49 export default TaskForm;
```

Kuva 13. TaskForm-komponentti.

TaskForm-komponentille asetettiin tila syöttökenttää varten. Sen tila on asetettu useState-hookilla, jonka alkuarvoksi määritellään TaskList emokomponentilta (kuva 16) propseina tuotu muokattavan tehtävän nimi tai tyhjä merkkijono. TaskForm päivittää tilan käyttämällä setValue-funktiota. Komponentti vastaanottaa onSubmit-funktion propseina. HandleSubmit tapahtumakäsittelijä käsittelee lomakkeen lähettämistä ja kutsuu onSubmit-funktiota tehtävän lisäyksessä tai muokkauksessa Task- ja TaskList-komponenteilta.

Task.js

src > components > Task.js > ...

```
import React, { useState } from "react";
import TaskForm from "./TaskForm";
import { AiOutlineDelete } from "react-icons/ai";
import { AiOutlineEdit } from "react-icons/ai";

const Task = ({ tasks, completeTask, removeTask, updateTask }) => {
  const [edit, setEdit] = useState({
    id: null,
    value: "",
  });

  const submitUpdate = (value) => {
    updateTask(edit.id, value);
    setEdit({
      id: null,
      value: "",
    });
  };

  if (edit.id) {
    return <TaskForm edit={edit} onSubmit={submitUpdate} />;
  }

  return tasks.map((task, index) => (
    <div
      className={task.isComplete ? "task-row complete" : "task-row"}
      key={index}
    >
      <div className="task" key={task.id} onClick={() => completeTask(task.id)}>
        {task.text}
      </div>
      <div className="icons">
        <AiOutlineEdit
          onClick={() => setEdit({ id: task.id, value: task.text })}
          className="edit-icon"
        />
        <AiOutlineDelete
          onClick={() => removeTask(task.id)}
          className="delete-icon"
        />
      </div>
    </div>
  ));
};

export default Task;
```

Kuva 14. Task-komponentti.

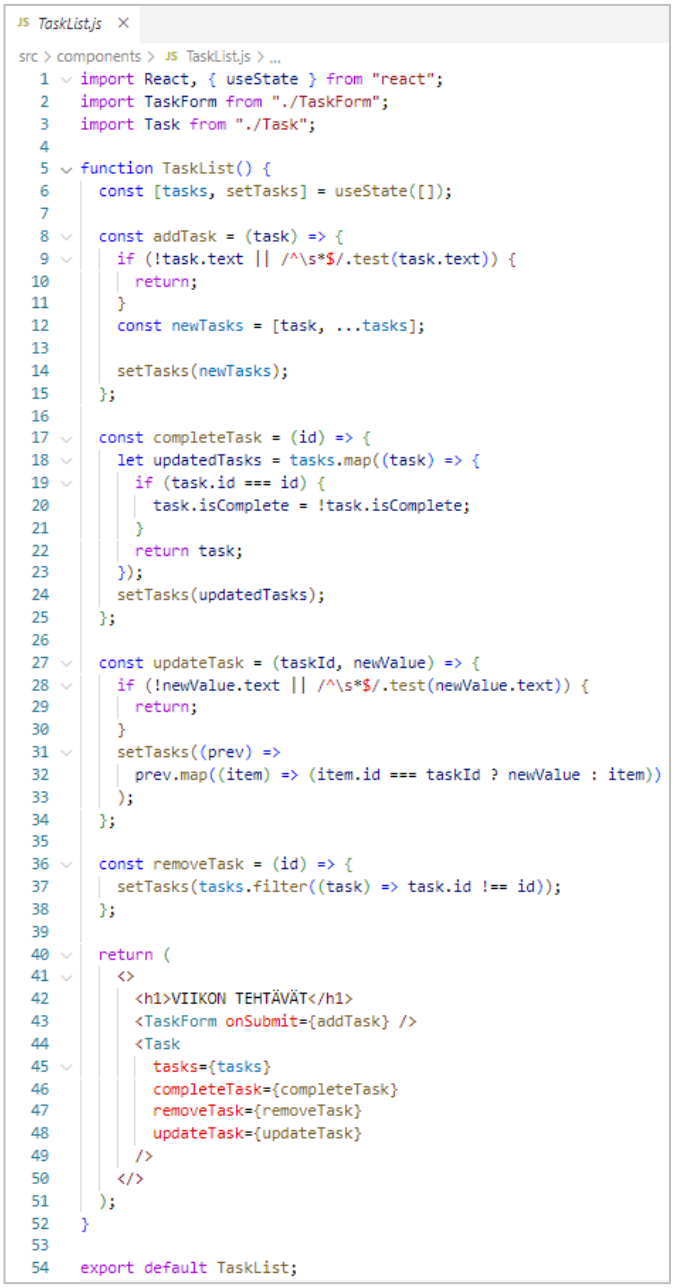
Task-komponentille (kuva 14) tuodaan emokomponentilta propseina luodut tehtävät ja tehtävien suoritus, muokkaus ja poisto toiminnot käsitelläkseen näitä toimintoja. Komponentti vastaa yksittäisen tehtävän näyttämisestä tehtävälistassa sekä niiden muokkaamisesta ja poistamisesta. Komponentti käyttää kunkin tehtävän näyttämiseen .map-metodia. Jokaisella tehtävällä on omat muokkaus- ja poistopainikkeet. Muokkaus- ja poistopainikkeita varten asennettiin React-icons paketti, josta löytyi sopivat kuvakkeet painikkeille (kuva 15).

```
npm install react-icons --save
```

Kuva 15. React Icons asennus.

Kun tehtävää klikataan komponentti välittää tiedon emokomponentille (kuva 16) ja asettaa tehtävän suoritetuksi. Task-komponentti käyttää useState-hookia hallitaakseen muokkaustilan tilaa. Siihen määritellyt id ja value edustavat tehtävän tunnistetta ja tehtävän tekstiä. Komponentin tila on määritelty niin, että jos muokkaus painiketta klikataan, TaskForm-komponentin muokkaus tekstikenttä renderöidään listan sijaan. Tehtävän muokkaamisen jälkeen Task-komponentti palauttaa setEdit-funktiolla komponentin alkutilaan.

TaskList-komponentti (kuva 16) hallitsee tehtävälistan tilaa ja siihen on sisällytetty Task ja TaskForm komponentit, joten se toimii näiden kahden komponentin emokomponenttina. Komponentille on myös määritelty tila useState-hookilla, jonka alkuarvo on taulukko.



```
JS TaskList.js
src > components > JS TaskList.js > ...
1  import React, { useState } from "react";
2  import TaskForm from "./TaskForm";
3  import Task from "./Task";
4
5  function TaskList() {
6    const [tasks, setTasks] = useState([]);
7
8    const addTask = (task) => {
9      if (!task.text || /^\s*$/.test(task.text)) {
10       return;
11     }
12     const newTasks = [task, ...tasks];
13
14     setTasks(newTasks);
15   };
16
17   const completeTask = (id) => {
18     let updatedTasks = tasks.map((task) => {
19       if (task.id === id) {
20         task.isComplete = !task.isComplete;
21       }
22       return task;
23     });
24     setTasks(updatedTasks);
25   };
26
27   const updateTask = (taskId, newValue) => {
28     if (!newValue.text || /^\s*$/.test(newValue.text)) {
29       return;
30     }
31     setTasks((prev) =>
32       prev.map((item) => (item.id === taskId ? newValue : item))
33     );
34   };
35
36   const removeTask = (id) => {
37     setTasks(tasks.filter((task) => task.id !== id));
38   };
39
40   return (
41     <>
42       <h1>VIIKON TEHTÄVÄT</h1>
43       <TaskForm onSubmit={addTask} />
44       <Task
45         tasks={tasks}
46         completeTask={completeTask}
47         removeTask={removeTask}
48         updateTask={updateTask}
49       />
50     </>
51   );
52 }
53
54 export default TaskList;
```

Kuva 16. TaskList-komponentti.

TaskList sisältää tehtävien lisäys, muokkaus, poisto ja suoritus toiminnot. AddTask-funktio lisää uuden tehtävän tasks-taulukkoon, updateTask-funktio päivittää olemassa olevan tehtävän tasks-taulukossa, removeTask-funktio poistaa tehtävän taulukosta tehtävän idn perusteella ja completeTask-funktio vaihtaa

tehtävän tilan suoritetuksi/ei suoritetuksi idn perusteella. Nämä neljä funktiota hallitsevat tilan päivitystä käyttämällä setTasks-funktiota.

TaskList renderöi TaskForm komponentin, jolle addTask lisäys toiminto viedään propsina. TaskList näyttää kaikki tehtävät luomalla yksittäisen Task-komponentin kutakin tehtävää kohden. TaskList antaa tasks tilan ja tehtävän suoritus, poisto ja muokkaus funktiot Task-komponentille propseina. TaskList sisällytetään App-komponenttiin, joka on sovelluksen ylinkomponentti (kuva 17).

App.js

src > App.js > ...

```
1   import React from "react";
2   import "./App.css";
3   import TaskList from "./components/TaskList";
4
5   function App() {
6     return (
7       <div className="task-app">
8         <TaskList />
9       </div>
10    );
11  }
12
13  export default App;
```

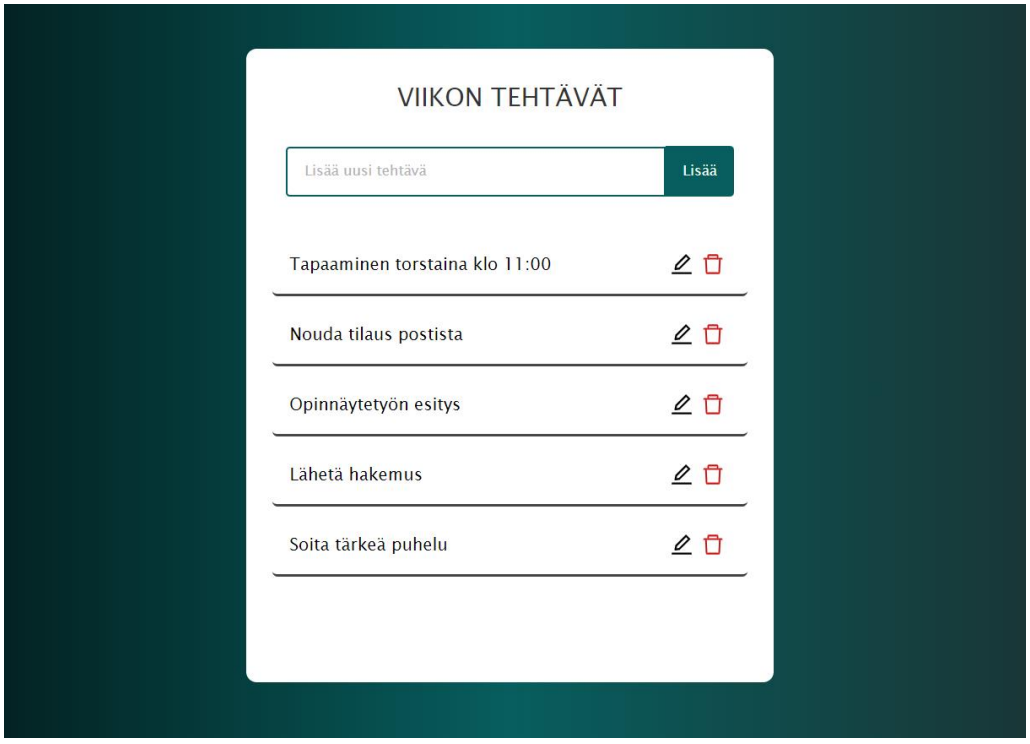
Kuva 17. Juurikomponentti.

Tehtävälistan ulkoasua varten luotiin erillinen App.css tiedosto, johon määriteltiin tyylit. App.css tiedosto tuodaan juurikomponentille, niin kuin kuvassa 17 nähdään App-komponentin ylimässä tasossa.

5 VALMIS TEHTÄVÄLISTASOVELLUS

Tässä luvussa esitellään valmis tehtävälistasovellus ja sovelluksen käyttöliittymä. Lisäksi analysoidaan, saavutettiinko sovellukselle asetetut tavoitteet?

Tehtävälistasovelluksen rakentamisessa käytettiin useita toiminnallisuuksia, kuten tehtävien lisääminen, muokkaaminen, poistaminen ja merkitseminen suoritetuksi. Komponenttien väliseen tietojen välitykseen käytettiin props-ominaisuutta ja tilan hallintaan Hookseja. Tuloksena syntyi kolme funktiokomponenttia, joiden toiminnallisuudet ja tilan hallinta sekä tietojen välitys toimivat odotetusti. Tehtävälistasovelluksesta tuli yksinkertainen ja selkeä.

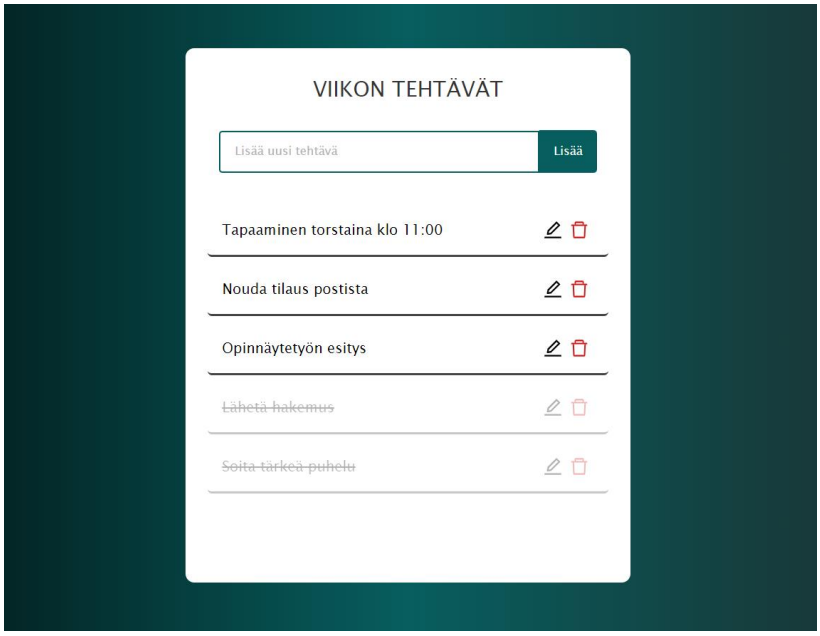


Kuva 18. Tehtävälistasovellus käyttöliittymä.

Kuvassa 18 esitetään tehtävälistasovelluksen etusivu. Tehtävän pystyy syöttämään tekstikenttään ja lisäämään tehtävälistaan. Tehtävät näkyvät

listattuna tekstikentän alapuolella ja uusin luotu tehtävä ilmestyy listassa aina ensinmäiseksi. Jokainen tehtävä sisältää omat muokkaus ja poisto painikkeet.

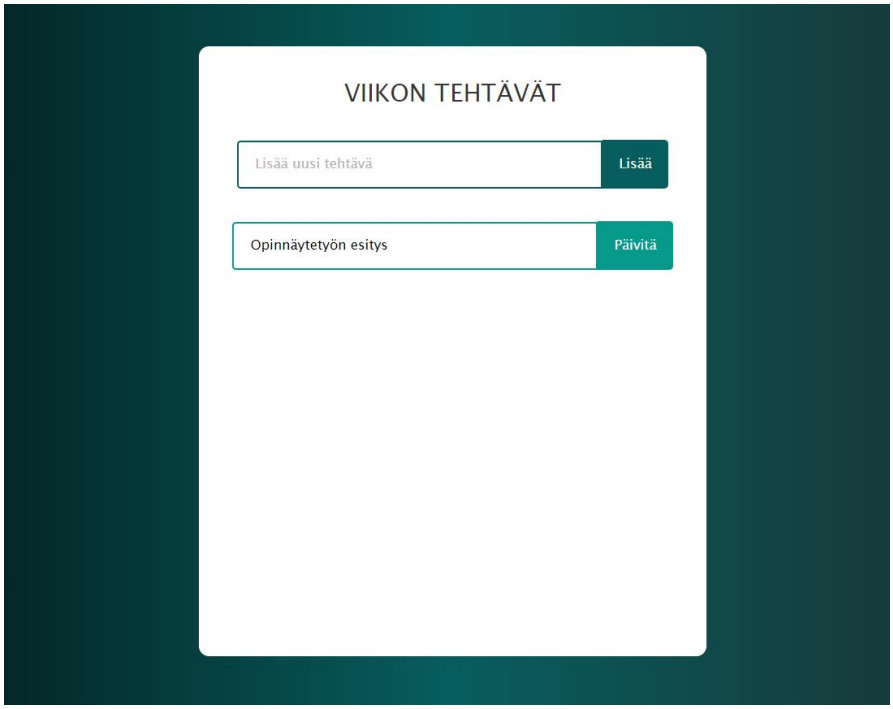
Kuvassa 19 tehtäviä on merkattuna suoritetuksi. Suoritetut tehtävät ovat yliviivattu ja näkyvät sumeasti. Tehtävät merkitään suoritetuiksi klikkaamalla tehtävän tekstiä ja ne voidaan palauttaa aktiiviseksi uudelleen klikkaamalla.



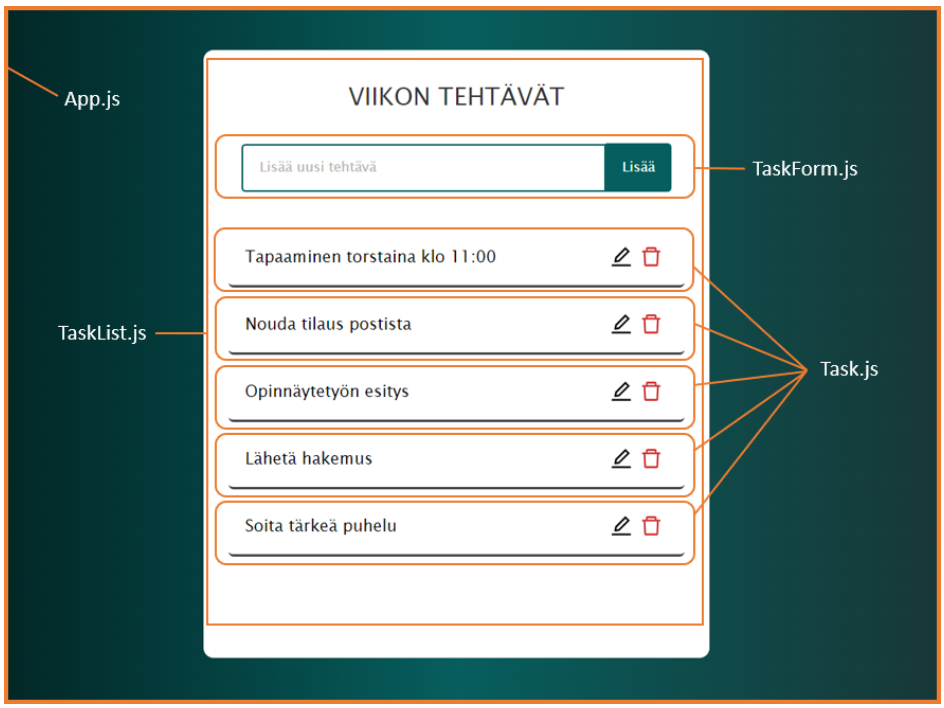
Kuva 19. Tehtävien merkitseminen suoritetuksi.

Kun tehtävän muokkaus kuvaketta klikataan, listan tilalle ilmestyy tekstikenttä muokkausta varten (kuva 20). Sen tila on asetettu niin että muokattava tehtävä näkyy tekstikentässä. Kun tehtävä on muutettu ja päivitä painiketta klikattu näkyy tehtävä muokattuna listassa. Tehtävän poisto onnistuu klikkaamalla punaista roskakorikuvaketta, jolloin tehtävä poistuu listasta.

Kuvassa 21 nähdään kuinka App-, TaskList-, Task- ja TaskForm-komponentit muodostavat tehtävälistasovelluksen.



Kuva 20. Tehtävien muokkaus.



Kuva 21. Komponentit tehtävälistasovelluksessa.

6 YHTEENVETO

Opinnäytetyön tavoitteena oli tutustua Reactin keskeisiin ominaisuuksiin ja luoda yksinkertainen sovellus, joka selventäisi teoriaosuudessa käytyjä käsitteitä. Työssä tarkasteltiin ydinkonsepteja, kuten komponentteja, tilaa, propseja ja tapahtumakäsittelyä. Opinnäytetyön tavoitteet täyttyivät ja työn tuloksena syntyi tietopaketti Reactin keskeisistä ominaisuuksista. Työ soveltuu hyvin Reactista kiinnostuneille. Haasteellista opinnäytetyössä oli joidenkin monimutkaisten käsitteiden lähteiden löytäminen.

Projektin tuloksena syntyi toimiva sovellus, jolla on helppo hallita tehtäviä. Reactin komponenttipohjaisuus, propsit, JSX ja tilan hallinta Hookseilla tarjosi miellyttävän ja tehokkaan tavan rakentaa käyttöliittymän. Sovellus vastasi asetettuja vaatimuksia ja toimi odotetusti.

Opinnäytetyö vastasi esitettyyn tutkimuskysymykseen hyvin. Reactin käyttö ja sen pääpiirteet tulivat työssä ilmi. Etenkin se, että Reactilla on tehokasta kehittää interaktiivisia käyttöliittymiä sen komponentti pohjaisuuden vuoksi, sillä se helpottaa sovelluksen eri osien hallintaa ja ylläpitoa. Työssä havaittiin, että React on tehokas ja todella suosittu kirjasto moderneissa verkkosovelluksissa.

LÄHTEET

Abiodun, D. 2021. React Class Component vs. Functional Component: What's the Difference. Progress Telerik-blogi. Viitattu 15.4.2023. https://www.telerik.com/blogs/react-class-component-vs-functional-component-how-choose-whats-difference.

Acharya, D. 2023. React UI Components Libraries: Our Top Picks for 2023. Kinsta-blogi. Viitattu 18.4.2023. https://kinsta.com/blog/react-components-library/.

Bolmér, P. 2021. How To Implement a Single Page Application Using React Router. ProgrammingPercy. Viitattu 15.4.2023. https://programmingpercy.tech/blog/how-to-implement-a-single-page-application-using-react-router/.

Eygi, C. 2020. React.js for Beginners-Props and State Explained. FreeCodeCamp. Viitattu 15.4.2023. https://www.freecodecamp.org/news/react-js-for-beginners-props-state-explained/.

Halme, A. 2018. Mikä on Single Page App ja mihin sitä käytetään?. City Dev Labs. Viitattu 15.4.2023. https://citydevlabs.fi/single-page-app/.

Hámori, F. 2022. The History of React.js on a Timeline. RisingStack-blogi. Viitattu 15.4.2023. https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/.

Helsingin yliopisto. 2023. 582101 – Ohjelmistotekniikan menetelmät, arkkitehtuuria ja rajapintoja. Viitattu 16.4.2023. https://www.cs.helsinki.fi/u/pohjalai/ke08/otm/slides/otm_08-arkkitehtuurisuunnittelua.pdf.

Houston Inc. JSX. Full Stack Open. Viitattu. 15.4.2023. https://fullstackopen.com/osa1/reactin_alkeet#jsx.

JavaTpoint. 2023a. React Introduction. Viitattu 15.4.2023. https://www.javatpoint.com/react-introduction

JavaTpoint. 2023b. React JSX. Viitattu 15.4.2023. https://www.javatpoint.com/react-jsx.

JavaTpoint. 2023c. React Props. Viitattu 15.4.2023. https://www.javatpoint.com/react-props.

JavaTpoint. 2023d. React State. Viitattu 15.4.2023. https://www.javatpoint.com/react-state.

Maranan, M. 2022. The React lifecycle: methods and hooks explained. Retool. Viitattu 15.4.2023. https://retool.com/blog/the-react-lifecycle-methods-and-hooks-explained/.

Mardan, A. 2017. React Quickly: Painless web apps with React, JSX, Redux, and GraphQL. E-kirja. Manning Publications. Viitattu 15.4.2023. https://learning.oreilly.com/library/view/react-quickly-painless/9781617293344/.

Meta Open Source 2023a. Adding Interactivity. https://react.dev/learn/adding-interactivity.

Meta Open Source. 2023b. Importing and Exporting Components. Viitattu 15.4.2023. https://react.dev/learn/importing-and-exporting-components.

Meta Open Source. 2023c. Responding to Events. Viitattu 24.4.2023. https://react.dev/learn/responding-to-events.

Meta Open Source. 2023d. Writing Markup with JSX. Viitattu 15.4.2023. https://react.dev/learn/writing-markup-with-jsx.

Meta Open Source. 2023e. Your First Component. Viitattu 15.4.2023. https://react.dev/learn/your-first-component.

Minnick, C. 2022. Beginning ReactJS Foundations Building User Interfaces with ReactJS. E-kirja. Wiley. Viitattu 15.4.2023. https://learning.oreilly.com/library/view/beginning-reactjs-foundations/9781119685548/.

Mozilla. 2023a. Introduction to the DOM. Mdn web docs. Viitattu 15.4.2023. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction.

Mozilla. 2023b. SPA (single-Page application). Mdn web docs. Viitattu 15.4.2023. https://developer.mozilla.org/en-US/docs/Glossary/SPA.

Narayn, H. 2022. Just React!: Learn React the React way. E-kirja. Apress. Viitattu 15.4.2023. https://learning.oreilly.com/library/view/just-react-learn/9781484282946/.

Olawanle, J. 2022. Rules of React Hooks. CoderPad. Viitattu 16.4.2023. https://coderpad.io/blog/development/rules-of-react-hooks/.

Ravichandran, A. 2019. React Virtual DOM Explained in Simple English. Medium-artikkeli. Viitattu 15.4.2023. https://adhithiravi.medium.com/react-virtual-dom-explained-in-simple-english-fc2d0b277bc5.

React. 2023. Introducing JSX. Viitattu 8.3.2023. https://legacy.reactjs.org/docs/introducing-jsx.html.

Schwarzmüller, M. 2022. React Key Concepts. E-kirja. Packt Publishing Ltd. Viitattu 15.4.2023. https://learning.oreilly.com/library/view/react-key-concepts/9781803234502/.

Ström, C. 2019. React pähkinänkuoressa. Joinex-blogi. Viitattu 15.4.2023. https://joinex.fi/react-pahkinankuoressa/.

W3schools. 2023a. JavaScript HTML DOM. Viitattu 15.4.2023. https://www.w3schools.com/js/js_htmldom.asp.

W3schools. 2023b. React lifecycle. Viitattu 24.4.2023. https://www.w3schools.com/react/react_lifecycle.asp.

W3schools. 2023c. React State. Viitattu 15.4.2023. https://www.w3schools.com/react/react_state.asp.

W3schools. 2023d. Styling React Using CSS. Viitattu 15.4.2023. https://www.w3schools.com/react/react_css_styling.asp.

W3schools. 2023e. React JSX. Viitattu 15.4.2023. https://www.w3schools.com/react/react_jsx.asp.