Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

SAMULI LÖF

Javascript UI kirjastot

TIETOTEKNIIKAN KOULUTUSOHJELMA
2020

Tekijä(t)
Löf, Samuli

Julkaisun laji
Opinnäytetyö, AMK

Päivämäärä
Toukokuu 2020

Sivumäärä
30

Julkaisun kieli
Suomi

Julkaisun nimi
Javascript UI kirjastot

Tutkinto-ohjelma
Tietotekniikan koulutusohjelma

Tiivistelmä
Opinnäytetyössä tutkittiin kolmea eri Javascript käyttöliittymä kirjastoa. Nämä ovat React, Angular ja Vue. Valinta tehtiin suosion mukaan.

Tutkinta tehtiin luomalla pieni sivusto jokaisella kirjastolla. Jokainen sivusto tehtiin samanlaiseksi käyttäen samoja tyylejä ja samanlaista tietorakennetta. Tavoitteena oli löytää eroavaisuuksia kirjastojen toiminnassa, millä on yksinkertaisinta tehdä sivuston osat. Työssä selvitetään myös mihin kirjastoon on helpoin yhdistää valmiiksi tehdyt tyylit ja datakoodit.

Tekeminen aloitettiin etsimällä kirjaston sivuilta ohjeet miten luoda projekti lokaaliin tiedostoon. Tämän jälkeen tehtiin sivu näyttämään esiluotuja elementtejä. Viimeisenä lisättiin lomake uuden elementin luomiseen.

Muutamia eroavaisuuksia löytyi. Reactissa oli vaikeinta liittää tyylit sivuston osiin. Se ei tukenut komponenttikohtaisia tyylejä suoraan, vaan olisi tarvinnut toisen kirjaston. Angularissa tietorakenne toimi eri tavalla. Koodeja ei voinut käyttää suoraan, vaan ne piti muokata Angularin muotoon. Vuessa taas kehittäjäystävällisyys ei ollut samalla tasolla kuin muissa. Kirjoittaessa Vue koodia editori ei osannut kertoa mitkä muuttujat ja funktiot ovat sallittuja missäkin tilanteessa.

Kaikilla oli siis omat heikot puolensa. Loppupäätelmänä React oli hyvänä keskitienä yksinkertaisuuden ja kehittäjäystävällisyyden välillä. Työkalut kertoivat suoremmin, jos jokin on väärin ja mikä sallittua. Projektirakenne oli myös paljon yksinkertaisempi kuin Angularissa

Javascript, HTML, CSS, Ohjelmointi, sovelluskehykset

Author(s)
Löf. Samuli

Type of Publication
Bachelor's thesis

Date
May 2020

Number of pages
30

Language of publication:
Finnish

Title of publication
Javascript UI libraries

Degree programme
Information Technology

Abstract
Three different Javascript interface libraries were compared in the study. These three were React, Angular ja Vue. They were chosen according to popularity.

Research was made by creating a small website with each library. Every website was made to work and look the same way, using the same styles and data layout. The goal was to find differences between the libraries, what is simplest to do with which. Also, which library is easiest to use with existing code.

The study was started by finding the directions on how to start a local project on each library's own website. After this a page to show already made items was created. Lastly a form to add new items to the page was added.

A few differences were found. In React it was the hardest to add styles in a good way. You could not add styles into a component without an additional library. Angular then did not work with data code from outside. It had to be remade to fit into the mould. Vue was not as good in developer friendliness. When writing Vue code, the editor could not tell all mistakes or which variable and function could be used where.

Every library had its weaknesses. As a conclusion React is a good middle point between simplicity and developer friendliness. The tools worked better telling what is wrong and what can be used where. The project structure was much simpler than Angular.

Javascript, HTML, CSS, World Wide Web

LYHENTEET

JS	Javascript. Selaimessa käytetty skriptikieli
CSS	Cascading Style Sheets. Selaimessa käytettävät tyylit
HTML	Hypertext Markup Language. Kuvauskieli verkkosivuille
UI	User Interface. Käyttöliittymä
NPM	Node Package Manager. Pakettien hallinta ohjelma
CLI	Command Line Interface. Komentorivi ohjelma
SRC	Source. Lähdekoodi
DB	Database. Tietokanta
JSON	Javascript Object Notation. Muoto säilyttää tietoa

SISÄLLYS

1 JOHDANTO

Javascript UI kirjastot ovat työkaluja sivujen luomiseen. Ne helpottavat lisäämään, muuttamaan ja poistamaan sivuilta osia sen käyttämisen aikana (Stepnov, 2019). Kirjastot sisältävät tapoja jakaa sivu komponentteihin, joita yhdistelemällä lopputulos valmistuu. Eri kirjastot päättävät enemmän projektin rakenteesta kuin toiset. On tärkeää käyttää jotain kirjastoa, jotta projekti pysyy organisoituna ja hallittavana.

Komponentti tuo uudelleenkäytettäviä osia verkkosivuihin. Yhden osan koodi on myös selvästi yhdessä paikassa eikä leviteltynä moniin eri tiedostoihin. Sivun tilan hallinta on vaikeaa, jos mikä vaan osa koodia voi koskea mihin vaan. Käyttöliittymän pitäminen samassa tilassa kuin data on vaikeaa (Gimeno, 2018). Ilman kirjastoja tämän toteuttaminen vaatii paljon enemmän koodia sekä on helposti hajoavaa. Kirjastot ovat ratkaisu tähän mahdollistaen yhä monipuolisempien ja suurempien verkkosivujen luomisen sekä ylläpitämisen.

Verkkosivuja tehtäessä mietitään eroavaisuuksia kirjastojen välillä. Otetaan selvää kuinka helppoa sekä selkeää minkäkin toiminnallisuuden tekeminen on eri kirjastoilla. Jokainen testiprojekti aloitetaan etsimällä kirjaston omilta sivuilta ohje projektin aloittamiseen ja seurataan sitä. Pitää myös huomioida kehitysympäristön helppous, mikä tukee parhaiten virheiden tunnistamista suoraan editorissa, mikä muuttujien täydennystä.

Tässä työssä tehdään yksinkertainen sivu muutamalla eri javascript kirjastolla. Sivuksi valittiin muistiinpano ohjelma, joka onkin näissä suosittu esimerkki ja demo sivu.

2 VALITUT KIRJASTOT

Kirjastot valittiin niiden suosion pohjalta. Selvästi käytetyimmät ovat React, Angular ja Vue (inVerita, 2020)

2.1 React

React on Facebookin vuonna 2013 julkaisema kirjasto. Tällä hetkellä se on suosituin kirjastoista, ja siihen löytyy myös suurin määrä yhteensopivia muita paketteja. React välittää ainoastaan elementtien lisäämisestä sivulle, joten kaikki tiedon hallinta ja hakeminen tehdään muilla kirjastoilla.

React toi virtuaalisen sivun mukaan kirjastomaailmaan. Tässä tekniikassa ensin tallennetaan sivu muistiin, ja muutokset sisältöön tehdään ylikirjoittamalla vanha sivu. React sitten laskee, mikä on oikeasti muuttunut, ja päivittää näkyvän sivun vastaamaan muistissa olevaa.

JSX on myös yksi tärkeä ominaisuus. Se on lisäosa Javascriptiin, joka tuo HTML tyyppisen syntaksin suoraan javascriptiin. Ilman tätä React koodin kirjoittaminen vaatisi paljon käsin kirjoitettavia funktiokutsuja.

2.2 Angular

Angular on Googlen kehittämä kirjasto, joka on vanhimpia käyttöliittymäkirjastoja. Tämän vanhempi versio AngularJs julkaistiin alun perin vuonna 2010. AngularJs siis on vanhin tässä käsiteltävistä, ja yksi vanhimmista käyttöliittymä kirjastoista (Wanyoike, 2018).

2016 julkaistu Angular on uusi versio, joka korjasi paljon puutteita aiemmasta versiosta. Käyttöön tulivat Reactin tapaan komponentit. Typescript, eli Microsoftin kehittämä tyypitetty javascript, otettiin käyttöön. Työssä käytetään uutta versiota Angularista.

2.3 Vue

Evan You teki Vuen inspiroituessaan Angularista, kun hän oli töissä Googlella. Tämä onkin ainut näistä kolmesta, joka ei ole minkään yrityksen hallinnassa. Alun perin Vue julkaistiin vuonna 2014, ja se on pikkuhiljaa kerännyt lisää suosiota. Vaikka Vue ei ole Euroopassa ja Amerikassa yhtä suuri kuin React ja Angular, niin Kiinassa Vue on erittäin käytetty. Evan saikin Kiinasta paljon tukea Vuen kehittämiseen (Cromwell, 2017).
Evan ajatteli Angularin olevan liian suuri ja päättävän liian paljon projektista. Tämän takia Vue:sta tehtiin pieni, yksinkertainen ja mukautuva. Vuessa saa valita käyttääkö JSX:ää, Typescriptiä, html templateja, funktiota tai mitä haluaakaan.

Kun React ei tuo itsessään mitään muuta kuin alustan komponenteilla, ja Angular puolestaan tuo kaiken mahdollisen, niin Vue tuo virallisesti tuettuna muutaman tärkeimmän lisäpaketin. Esimerkiksi sivun tilan hallinnoiminen on yksi näitä.

3 YHTEINEN OSA

Jokainen kirjasto vaatii tyylit ja tietorakenteen. Määritellään ne jokaiselle samoiksi, ja lisätään ne projekteihin niiden kirjastojen käyttämällä tavalla.

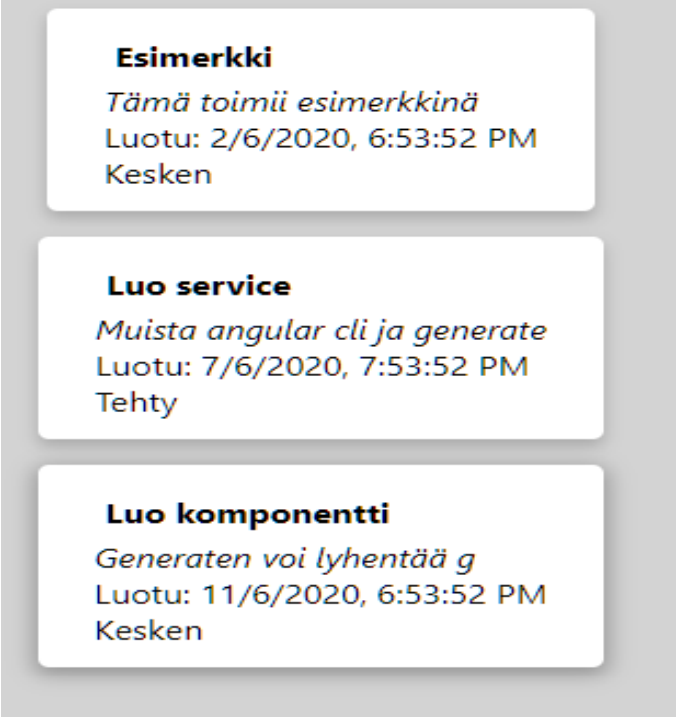
3.1 Tyylit

Tyylit tehdään tavallisella CSS:llä, ja siihen lisätään tyylit, joita kukin kirjasto käyttää. Tyylit tarvitaan korteille, missä näytetään muistiinpanot. CSS ohjeita löytyy W3 School sivulta.

W3 sivulla olevaan kortti-demoon (W3Schools, 2020) lisätään myös marginaalit, jotta kortit ovat keskitetty ja etteivät ne ole aivan vierekkäin. Lisätään padding, jotta tekstit eivät ole aivan reunoissa kiinni. Muutetaan kortin taustaväri valkoiseksi ja yleinen taustaväri harmaaksi. Lopputulos on esitetty kuvissa 1 ja 2.

```
.card {
  /* Add shadows to create the "card" effect */
  box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2);
  transition: 0.3s;
  width: fit-content;
  padding: 10px;
  background-color: white;
  border-radius: 5px; /* 5px rounded corners */

  margin-left: auto;
  margin-right: auto;
  margin-bottom: 15px;
}

/* On mouse-over, add a deeper shadow */
.card:hover {
  box-shadow: 0px 0px 16px 5px rgba(0, 0, 0, 0.5);
}

/* Add some padding inside the card container */
.container {
  padding: 2px 16px;
}

h4 {
  margin: 5px;
  padding: 0px;
}
```

Kuva 1. Tyylit



Kuva 2. Tyylitetyt elementit

3.2 Data

Muistiinpanot tulevat tarvitsemaan otsikon, kuvauksen ja päivämäärän. Tarvitaan myös id, jotta voimme pitää yllä uniikkia listaa. Json muodossa tämä tulee näyttämään Kuva 3 mukaiselta.

```
[
  {
    "id": 1,
    "title": "Esimerkki",
    "description": "Tämä toimii esimerkkinä",
    "createdAt": "2020-02-06T16:53:52.178Z",
    "done": false
  }
]
```

Kuva 3. Esimerkki todo data

Niinpä JSDoc tyyppinä tämä näyttää Kuva 4 näköiseltä

```
/**
 * Define TodoItem type
 * @typedef {object} TodoItem
 * @property {number} id - an ID
 * @property {string} title - Title of the item
 * @property {string} description - Description of the item
 * @property {Date} createdAt - Date when this item was created
 * @property {boolean} done - Whether this todo item is complete
 */
```

Kuva 4. Todo tyyppi

Tyypit tekevät editorin käytöstä mukavampaa. Ilman tyyppejä editori ei tietäisi mikä on sallittua missäkin tilanteessa. Lisäämällä tämän toiminnallisuuden ja käyttämällä tyyppiä oikeissa paikoissa editori voi kertoa onko koodi oikein ja auttaa kirjoittamaan oikeat kentät. Tämä nopeuttaa kehitystä ja auttaa uudelleennimeämään muuttujia myöhemmin. Tyypit vähentävät bugeja noin 15 %:lla (Zheng Gao, 2017).

JSDoc on yksi tapa luoda tyyppejä normaaliin javascriptiin. Siinä tehdään tietyn muotoisia kommentteja, joita tekstieditori ja työkalut osaavat käyttää.

Data tallennetaan Local Storageen (MDN web docs, 2020), jotta ei tarvita tietokantaa tai ulkoista riippuvuutta. Local Storage on tallennustilaa selaimessa, jossa sivut voivat pitää dataa. Jokaisella sivulla on oma muistinsa. Local storage toimii tallentamalla tekstiä nimellä, ja myöhemmin samalla nimellä voi hakea tallennetun tekstin. Tulemme tallentamaan Todo datan muutettuna JSON-muotoiseen tekstiin.

Local storagen käyttämiseen luodaan kaksi apufunktiota. Toinen hakee kaikki, ja toinen tallentaa kaikki. Kuvassa 5 on esitetty koodit tähän toimintoon.

```
const db = window.localStorage;
const key = 'todo_data_json';
/**
 * Returns all current todo items or an empty array
 * @returns {TodoItem[]} All todo items
 */
export const GetAll = () => {
  /** @type {TodoItem[]} */
  const all = JSON.parse(db.getItem(key)) || [];
  return all.map(x => ({
    ...x,
    createdAt: new Date(x.createdAt),
  }));
};
/**
 * Saves the parameter as todo items
 * @param {TodoItem[]} all All todo items
 */
export const Save = all => {
  db.setItem(key, JSON.stringify(all));
};
```

Kuva 5. Local Storage funktiot

Näitä käyttämällä voidaan luoda muut funktiot poistamaan, lisäämään ja muuttamaan muistiinpanoja. Esimerkiksi uuden muistiinpanon lisääminen onnistuu kuva 6 näyttämällä tavalla

```
/**
 * Adds the new item to db
 * @param {types.TodoItem} item Item to add
 */
export const AddItem = item => {
  const all = GetAll();
  all.push(item);
  Save(all);
};
```

Kuva 6. Funktion listauksen lisäämiseksi tietokantaan

4 PROJEKTIN TEKO

Tehdään jokaiselle alustalle projekti alustan sivuilta löytyvän ohjeen mukaan. Vaatimuksina on asentaa projekti kansioon tietokoneella sekä kehitysympäristö hot-reloading ominaisuudella. Tämä tarkoittaa, että sivu päivittyy itsestään muutoksien jälkeen.

Jokainen kirjasto tulee vaatimaan Nodejs:n ja npm:n asennuksen. Nodejs on ohjelma, joka mahdollistaa javascript ohjelmien ajon selaimen ulkopuolella. Npm on pakettien hallinta ohjelma Nodelle. Näiden asennus onnistuu nodejs.org osoitteesta (Node, 2020).

4.1 Angular

Angularin ohjeet (Angular, 2020) ohjaavat ensin käyttämään StackBlitz sivua. Tutkimalla sivua löytyy teksti ”In actual development you will typically use the Angular CLI”, josta painaessa aukeaa Angular CLI:stä muutaman rivin selvitys. Se on komentorivi ohjelma, jolla hallitaan ja ajetaan Angular projektia. Tämän alla on toinen linkki ”Local Environment Setup”, josta viimein projektin teko ohjeet.

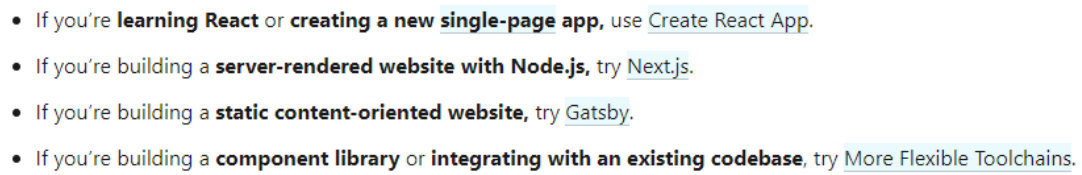
Ensin asennetaan angular/cli sovellus npm pakettienhallinnasta. Tämä mahdollistaa komennon ng komentorivillä. Uusi projekti luodaan ajamalla ”ng new app”. Luomisessa kysytään lisätäänkö routing paketit. Esimerkkiprojektissa ei tulla käyttämään tätä ominaisuutta, joten vastataan ei. Tyylikirjastoista on valittavana CSS, SCSS, Sass, Less ja Stylus. Valitaan CSS, sillä se on näistä yksinkertaisin.

Projekti ajetaan kirjoittamalla komentoriville ”ng serve –open”. Tämä avaa myös projektin selaimeen, jossa on linkkejä Angularin dokumentaatioon ja ohjeisiin, jotka ohjeistavat miten jatkaa.

4.2 React

Reactin sivut (React, 2020) on pitkä lista eri ohjeita, miten lisätä React sivulle ja mistä aloittaa oppiminen. Ensin sanotaan, että ”simple HTML page with script tags might

still be the best option”, mutta heti seuraava kappale sanoo, että toisenlainen pohja antaa koko React ekosysteemin. Linkin takana sanotaankin, että ensin mainittu on helpoin tapa lisätä jo olemassa olevaan sivuun, kun taas koko pohja antaa paljon etuja kuten virheiden huomaaminen aikaisemmin.



Kuva 7. Lista suositelluista ohjelmista React projektiin (ReactJs, 2020)

Harjoitusprojektissa opetallaan Reactia ja luodaan single-page app, joten projektissa valitaan kuvassa 7 listatuista tavoista ensimmäinen. Projekti luodaan kirjoittamalla komentoriville ”npx create-react-app my-app”. Npx on ohjelma ajaa npm paketteja asentamatta niitä. Mitään asetuksia ei kysytty, vaan create-react-app teki suoraan projektin.

Create-react-app on Facebookin ylläpitämä paketti, joka pitää sisällään kaiken tarvittavan projektin tekemiseksi. Ilman tätä pitäisi erikseen asentaa monta eri vaiheen pakettia muokkaamaan koodia, lataamaan sivun uudestaan, ajamaan testejä, kertomaan yleisistä virheistä sekä muuta. Tämä yksi paketti helpottaa ylläpitoa, mutta jos haluaa muokata sisältyvien pakettien asetuksia, niin pitää jakaa create-react-app sen osiin ja ylläpitää näitä osia käsin.

Projekti käynnistyy kirjoittamalla ”npm start”. Tämän jälkeen aukeaa sivu, joka kertoo mitä tiedostoa muokata ja antaa linkin Reactin sivuille. Lopputuloksena on selvästi pienempi sivu kuin Angularin projektissa.

4.3 Vue

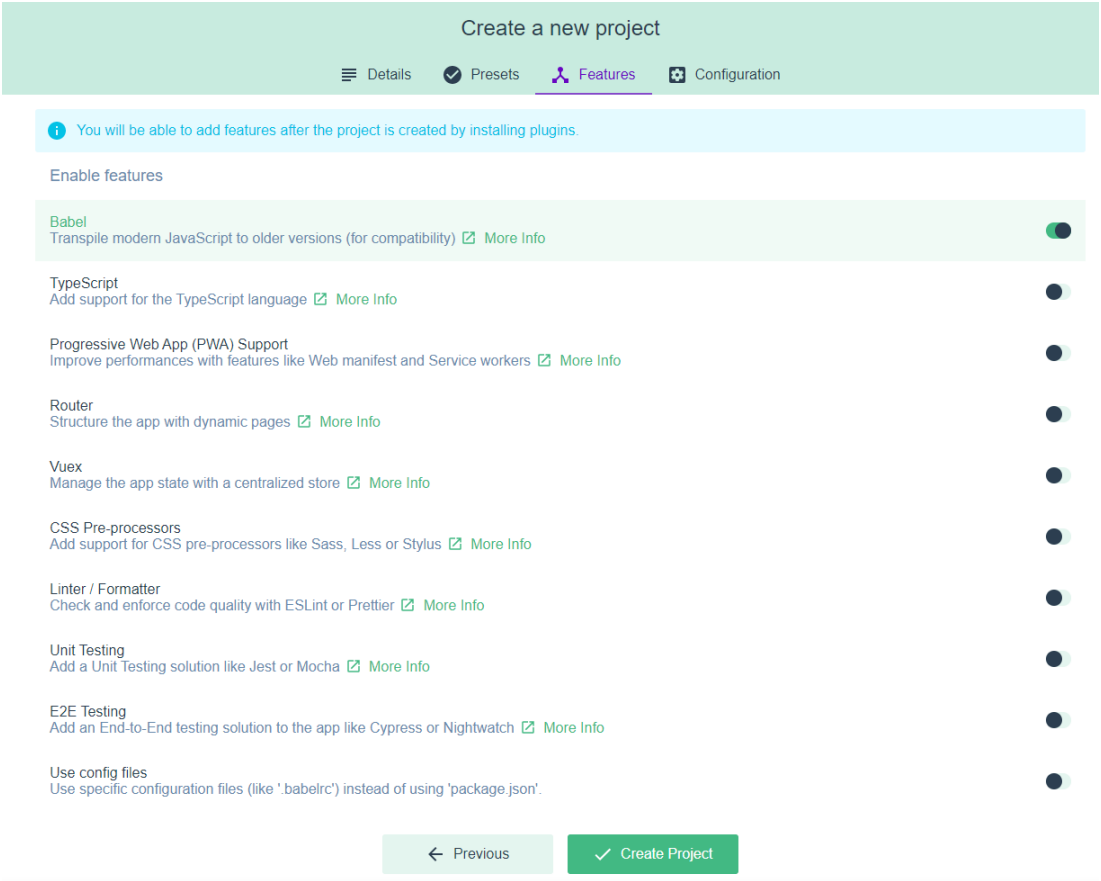
Vuen sivut (VueJs, 2020) ohjaavat tuttuun tapaan selaimessa toimivaan esimerkkiin. Sivut tarjoavat myös html tiedoston, johon on lisätty vue.js skripti. Muutaman linkin kautta tulee vastaan pohja moderniin työtapaan.

Vuen tapa on Angularin tyyliin komentorivi ohjelma. Tällä ei kuitenkaan hallita koko projektia. Ominaisuuksiin kuuluu uuden projektin luominen, yhden prototyyppi tiedoston ajaminen sekä graafinen UI projektin hallitsemiseen.

Uuden projektin luonti onnistuu asentamalla ohjelma ajamalla ”npm install -g @vue/cli” ja sen jälkeen ”vue create my-app”

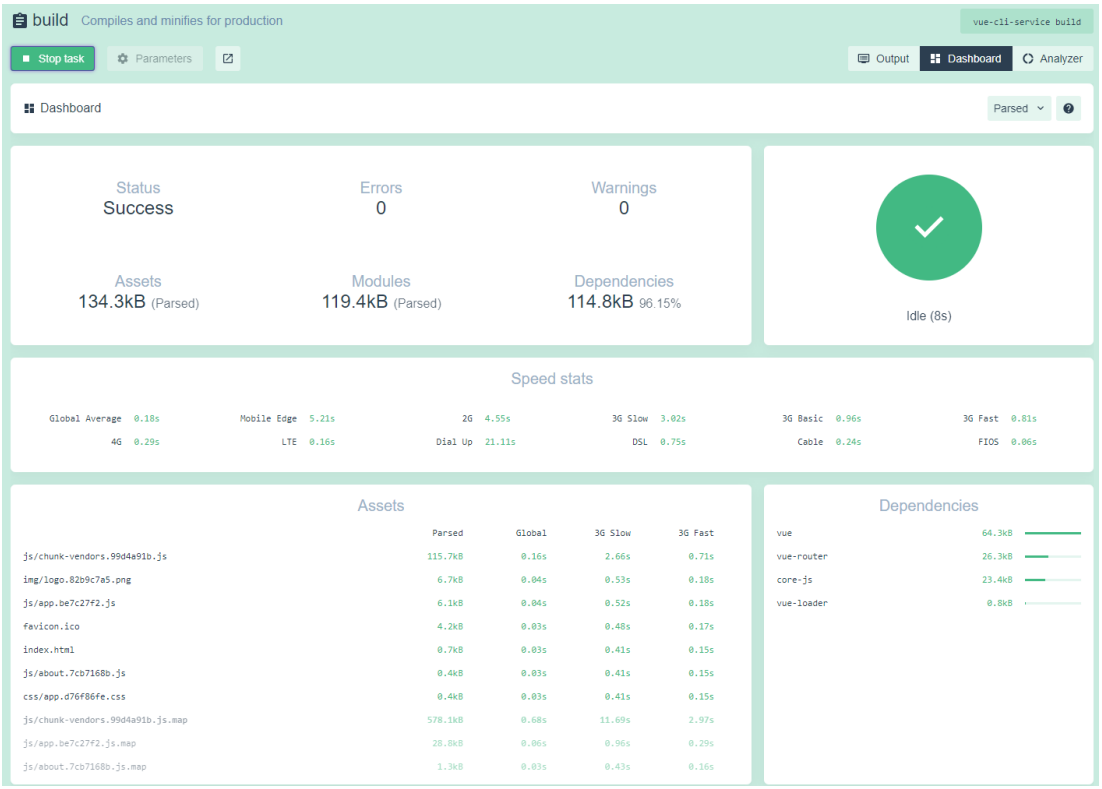
Vue tarjoaa monta eri asetusta projektia luodessa kuten routerin ja tilanhallinta kirjaston. Vue myös kysyy, mikä linter ja formatter asennetaan. Linter etsii virheitä koodista ja formatter asettelee koodin tyyliohjeen mukaan ilman ohjelmoijan työtä. Testiprojektissa valittiin ESLint + Prettier, mitkä hoitavat nuo molemmat halutut toiminnot.

Vaihtoehtona on ”vue ui” joka tarjoaa selaimessa toimivan käyttöliittymän myös projektin luontiin. Hiirellä voi naputella aiemmat vaihtoehdot sekä lukea lisää jokaisesta valinnasta kuten Kuvassa 7.



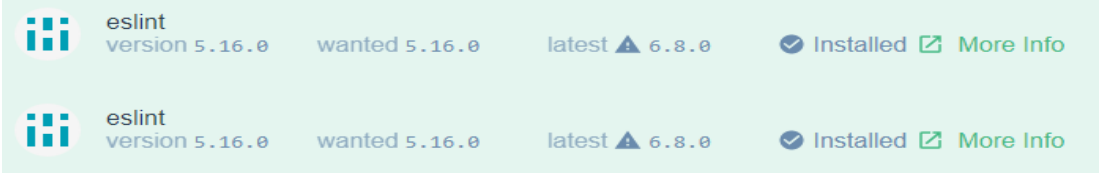
Kuva 7. Toiminto Vue UI projektin luomiseen

Projekti ajetaan kirjoittamalla ”npm run serve” yleisen ”npm start” sijaan. Tämä on helppo vaihtaa package.json tiedostossa haluttuun muotoon. Tämä jälkeen aukeaa minimaalinen sivu, jossa on linkkejä dokumentaatioon ja yleisiin lisäpaketteihin.



Kuva 8. Vue ui projektista

Kuvassa 8 oleva Vue UI näyttää mielenkiintoista dataa projektista. Esimerkiksi voidaan nähdä kuinka kauan sivun lataamiseen menisi eri liittymänopeuksilla, mitkä tiedostot ja ulkopuoliset kirjastot vievät tuosta eniten eli mihin kohdistaa optimointi. Tämä UI toimii kuitenkin hitaasti muutaman välilehden vaihdon jälkeen.
Hyödyllinen osa voi myös olla Plugins ja Dependencies. Ne näyttävät selvästi, mitä kirjastoja on asennettuna ja mitkä haluavat päivityksiä. Ainoastaan pienen versionmuutoksen päivitykset kuitenkin toimivat oikein, esimerkiksi versiosta 5.1.3 versioon 5.10.0. Yrittäessä ladata kirjaston uusi versio, asentuu vain vanha versio uudestaan kuten kuvassa 9.



Kuva 9. Paketti kahteen kertaan

5 MUISTIINPANON NÄYTTÄMINEN

Tässä osiossa luodaan kortti näyttämään muistiinpano. Tämä tehdään omana komponenttina, joka voidaan lisätä silmukassa jokaiselle listaukselle pääkomponentissa. Muistiinpanoon kuuluu otsikko, kuvaus sekä päivämäärä. Harjoitusprojektin tekemisen testaaminen aloitetaan Angularilla.

5.1 Angular

Angularissa ulkoisen datan kanssa pidetään yhteyttä service-arkkitehtuurin kautta. Luodaan uusi service kirjoittamalla komentoriville ”ng g service todo-service”. Angular käyttää typescriptiä, joka on kuin javascript, mutta sitä on parannettu tyypeillä ja lisäominaisuuksilla. Niinpä luodaan aiemmin tehdyn esimerkin data koodit typescript muotoon. Tämä tarkoittaa JSDoc kommenttien muuttamista interfaceksi kuvan 10 tavalla.

```
  /**
   * Updates item in db
   */
  UpdateItem = (item: TodoItem) => {
    let all = this.GetAll();
    all = all.map(x => (x.id !== item.id ? x : { ...x, ...item }));
    this.Save(all);
  };
}

interface TodoItem {
  id: number;
  title: string;
  description: string;
  createdAt: Date;
  done: boolean;
}
```

Kuva 10. Typescript tyyppi määrittely

AppComponent on pääkomponentti, joka tulee näyttämään sivun sisällön. Toteutetaan OnInit rajapinta komponentille lisäämällä ”Implements OnInit” niin saadaan alustuskoodia sivulle. Tässä alustuksessa haemme tallennetun datan taulukkoon.

Seuraavaksi luodaan komponentti ajamalla ”ng g component todo-list-item”. Tämä luo komponentin kansioon todo-list-item. Samalla saadaan erillinen tiedosto tyyleille, html:lle ja logiikalle. Tavoitteena on lähettää tälle komponentille yksi TodoItem, mikä onnistuu Angularin @Input dekoraatiolla. Tämä mahdollistaa html:ssä datan antamisen tägille.

Seuraavaksi poistetaan AppComponentin html tiedostosta kaikki sisältö ja lisätään <app-todo-list-item [todoItem]="item" *ngFor="let item of todoItems">. Nimen ympäröiminen hakasuluilla tarkoittaa, että sisällä oleva muuttuja tulee komponentin Typescript osasta. *ngFor tarkoittaa, että toistetaan sama tagi jokaiselle todoItems taulukon riville.

Sitten lisätään TodoListItemille html-sisältö. Muuttujia näytetään html:n keskellä ympäröimällä tämä {{}} merkkien sisälle. Esimerkiksi muistiinpanon otsikko saadaan näkyviin kirjoittamalla {{todoItem.title}}.

5.2 React

React-toteutuksessa luodaan kansio src/api ja siirretään sinne aikaisemmin luodut data koodit. Tällöin niitä voidaan käyttää suoraan, sillä React toimii yhteen muiden datakirjastojen kanssa. Kansiorakenne on myös yksinkertaisempi, koska projektissa on tiedostoja vähemmän.

App.js toimii pääkomponenttina. Tyhjennetään ensin kaikki sen sisällä oleva demokoodi, ja haetaan sitten data kyseiseen komponenttiin. React funktiokomponenteissa ei saa muokata ohjelman tilaa suoraan, vaan pitää käyttää tiettyä funktiota. Tässä tapauksessa tämä on React.UseEffect(), jota React kutsuu aina komponentin päivittyessä. Toisena parametrinä voi antaa taulukon, ja Effect ajetaan vain, jos arvot tämän taulukon sisällä ovat muuttuneet. Niinpä UseEffect on kuin Angularin OnInit ja OnChange yhdessä. Kuva 11:ssa on toteutettu haluttu useEffect funktio.

```
function App() {
  const [items, setItems] = React.useState(
    /**@type {import('./api/todotype').TodoItem[]}*/ ([])
  );

  // Alustetaan komponentin käynnistyessä
  React.useEffect(() => {
    DemoInit();
    const all = GetAll();
    setItems(all);
  }, []);
  return items.map(todoItem => <TodoListItem todoItem={todoItem} />);
}
```

Kuva 11. Reactin UseEffect

Typescriptin sijaan JSDoc käyttäminen vie enemmän tilaa koska tässä import lauseke tulee useState funktion sisälle. Typescriptissä se olisi ollut <TodoItem[]> lisäys useState funktiokutsuun.

Tyhjä taulukko UseEffectin parametrinä tarkoittaa, että se ajetaan vain komponenttia luodessa. Toiminta on vastaava kuin OnInit Angularissa.

Reactissa ei ole erillistä html template tiedostoa. Sen sijaan myös html kirjoitetaan javascriptin sisään käyttäen JSX formaattia. Ohjelman kääntövaiheessa se muutetaan javascriptiksi, createElement() funktioiksi.

Komponenteille annetaan parametrejä syntaksilla key={value} Angularissa olleen [key]=”value” sijaan. Ero ei ole iso. JSX kanssa voi kuitenkin sulkea tagin suoraan kirjoittamalla />. Normaalissa html tämä on mahdollista vain harvojen tagien kanssa. Rivien toistaminen onnistuu käyttämällä normaalia javascript silmukkaa. Funktio map toistaa sille annetun rivin jokaiselle taulukon arvolle.

```
/**
 * Renders a single todo item into a card
 * @param {{todoItem: import('./api/todotype').TodoItem}} props
 */
const TodoListItem = props => {
  const { todoItem } = props;

  return (
    <div className="card">
      <div className="container">
        <h4>{todoItem.title} </h4>
        <span style={{ fontStyle: 'italic' }}>{todoItem.description} </span>
        <div>Luotu: {todoItem.createdAt.toLocaleString()}</div>
        <div>{todoItem.done ? 'Tehty' : 'Kesken'}</div>
      </div>
    </div>
  );
};
```

Kuva 12. Reactin komponentti

Komponentti ottaa vastaan muuttujia suoraan funktion parametrissä, kuten kuvassa 12 olevasta props muuttujasta nähdään. Tyypitys on tosin paljon työläämpää kuin Angularissa, jossa riitti yksinkertainen todoItem: TodoItem. Reactissa toteutuksesta tuli pitkä kommenttirivi. Itse html on lähes samanlaista. Muuttujien ympärillä on vain yhdet {}, sekä CSS vaatii erikoisformaatin. Merkinnästä font-style sijaan poistetaan väliviiva ja käytetään isoa kirjainta, ja tekstin sijaan se pitää ympyröidä kaksilla aaltosuluilla. Ensimmäiset aaltosulut tekevät siitä parametrin, toiset sulut tekevät json objektin.

5.3 Vue

Aloitetaan taas siirtämällä ensin aiemmat db tiedostot src/api kansioon. Luodaan components kansioon TodoListItem.vue. Vue käyttää omaa tiedostopäätettä, vue, sillä yhdessä tiedostossa on kolme eri osioita. <template> tägien sisään html, <script> tägien sisään koodia ja <style> tägien sisään css. Jakaminen on helppoa. Lisätään skripti osioon todoItem muuttuja.

Html kirjoittaessa huomataan, että autocomplete-toiminto ei kuitenkaan toimi. Ainoastaan muuttujan nimi täydentyy, mutta muuttujan kenttiä ei tarjota vaihtoehdoiksi. Editori osaa ilmoittaa, jos jotain on väärin, mutta ei osaa antaa oikeaa vaihtoehtoa.

```
<template>
  <div class="card">
    <div class="container">
      <h4>{{ todoItem.titl }}</h4>
      <span style="fontStyle: 'italic'">{{ todoItem.description }} </span>
      <div>Luotu: {{ todoItem.createdAt.toLocaleString() }}</div>
      <div>{{ todoItem.done ? 'Tehty' : 'Kesken' }}</div>
    </div>
  </div>
</template>

<script>
export default {
  props: {
    /**@type{{new():import('../api/todotype').TodoItem)}} todoItem */
    todoItem: {
      type: Object,
      required: true,
    },
  },
};
</script>
```

Kuva 13. Esimerkki väärinkirjoitetusta muuttujasta

Kuvassa 13 näkyy väärinkirjoitettu titl, minkä indikointi on parempi kuin ei mitään. Tyyppien määrittäminen vaatii myös erikoisen new(): syntaksin. Typescript testattiin ja todettiin sen toimivan hieman paremmin, mutta sekään ei toiminut täydellisesti. Muiden komponenttien autoimport-toiminto ei myöskään onnistunut.

```
<template>
  <div id="app">
    <TodoListItem
      v-for="item in items"
      :key="item.id"
      :todoItem="item"
    ></TodoListItem>
  </div>
</template>

<script>
import TodoListItem from './components/TodoListItem';
import { GetAll, DemoInit } from './api/localStorage';

export default {
  name: 'App',
  data: () => ({
    items: /** @type {import('./api/todotype').TodoItem[]} */ ([]),
  }),
  components: {
    TodoListItem,
  },
  created() {
    DemoInit();
    this.items = GetAll();
  },
};
</script>
```

Kuva 14. Vue komponentti

Muille komponenteille parametrien antaminen onnistuu kaksoispisteellä, kuten kuvassa 14 on esitetty. Erikoisfunktiot alkavat v- tekstillä. OnInit-funktion sijasta käytetään created() funktiota. JSDoc tyyppien automaattinen korjaus ei myöskään toimi. Vue vaikuttaa olevan lähempänä täysin dynaamista kuin muut kirjastot.

Omituista on, että components muuttujaan pitää kirjoittaa kaikki komponentit, joita aikoo käyttää templatessa. Tässäkin editori osaa ilmoittaa, jos tätä komponenttia ei ole käytetty missään, mutta ei osaa täydentää sitä listaan itsestään. Editori ei lisäksi löydä tiedostoa, jossa komponentti on määritetty.

Tyylit saa kohdistettua yhteen komponenttiin vain lisäämällä sanan scoped tiedoston style osioon.

6 UUDEN MUISTIINPANON LISÄÄMINEN

Uuden muistiinpanon tekeminen tapahtuu <form> komponentilla. Lomakkeen OnSubmit metodi tulee kutsumaan javascript funktiota, joka lisää tämän listauksen tietokantaan sekä käyttöliittymään.

On olemassa ohjattuja ja ei ohjattuja lomakkeita (Arinich, 2016). Ohjattu tarkoittaa, että jokainen muutos ajaa javascriptia, jolloin kentän tieto on joka hetkellä tallennettuna muuttujaan. Ei ohjattu tarkoittaa, että kentän päivittäminen on selaimen vastuulla ja ainoastaan lähetettäessä lomake tieto haetaan kentistä. Ohjatulla on hyvänä puolena, että on helpompi muuttaa, tarkistaa tai tyhjentää kentän arvo. Testiprojektissa tehdään ohjattuja kenttiä.

6.1 Angular

Angularissa on kaksi eri tapaa luoda ohjattuja lomakkeita, jotka ovat Template ja Reactive lomakkeet. Template lomakkeissa lisätään html tagiin attribuutti, mikä linkittyy muuttujaan luokassa. Reactive lomakkeessa tehdään objekti luokkaan ja käytetään sen nimiä html tägeissä. Getting Started ohjeessa käytetään Reactive lomakkeita, joten tällä perusteella testiprojektissa käytetään myös niitä.

Moduuliin tarvitsee lisätä FormsModule ja ReactiveFormsModule, jotta angular osaa löytää oikeat attribuutit. Luodaan uusi komponentti nimellä add-new-item.

```
export class AddNewItemComponent {
  constructor(private formBuilder: FormBuilder) {
    this.addnewForm = this.formBuilder.group({
      title: "",
      description: ""
    });
  }
  addnewForm: FormGroup;
  @Output() onCreateNew: EventEmitter<addNewItem> = new EventEmitter();

  onSubmit(customerData) {
    this.addnewForm.reset();
    this.onCreateNew.emit({
      title: customerData.title,
      desc: customerData.description
    });
  }
}
```

Kuva 15. Angular output komponentti

Luomme lomakkeen koodin puolelle formBuilder.group funktiolla kuten kuvassa 15. Tälle annetaan kenttien nimet ja vapaaehtoisesti validointi-koodit. Output muuttujalla lähetetään tietoa tämän komponentin yläpuolella olevalle komponentille, että jotain on tehty. Esimerkin tapauksessa koodi suoritetaan, jos on painettu Lisää-näppäintä.

FormBuilder ei ota tyyppiä parametrinä, joten menetämme tyyppien tarkistuksen, ellei niitä toisteta uudestaan. Testiprojektissa jätetään tämä tekemättä.
Lomakkeen palautus alkuperäistilaan on erittäin helppoa, kutsumalla suoraan addnew-Form.reset().

```
<form [formGroup]="addnewForm" (ngSubmit)="onSubmit(addnewForm.value)">
    <div>
        <label for="name">
            Title
        </label>
        <input id="title" type="text" formControlName="title">
    </div>
```

Kuva 16. Angular eventin vastaanottaminen

Html on muuten normaalin lomakkeen tyylistä niin kuin kuvasta 16 nähdään, paitsi siihen on lisätty angularin attribuutit formGroup ja ngSubmit lomakkeelle. Ensimmäinen liittää tämän komponentissa esiteltyyn muuttujaan. Toinen liittää nappulan funktioon. Tämän lisäksi jokaiselle tekstikentälle annetaan formControlName, jotta angular tietää mihin muuttujaan tämän kentän teksti lisätään.

Pääkomponentissa tämä käsitellään antamalla komponentille attribuutti (onCreateNew)=”addNewItem($event)”. Sulut attribuutin ympärillä tarkoittavat tapahtumaa. Merkintä $event on taikasana, jolla saadaan tapahtuman parametrit eteenpäin ja addNewItem on pääkomponenttiin tehty funktio.

6.2 React

Reactissa ohjatut kentät tallennetaan komponentin tilaan. Jokaiselle kentälle tehdään oma muuttuja UseState kutsulla. Tieto uuden muistiinpanon luomisesta lähetetään eteenpäin aiemminkin käytetyllä props parametrilla. Reactissa propsit voivat olla myös funktioita.

Seuraavaksi tehdään uusi komponentti, joka ottaa propseina funktion, jota kutsutaan kun uusi listaus tehdään.

```
<form
  onSubmit={e => {
    e.preventDefault();
    onSubmit();
  }}
>
  <h4>Luo uusi</h4>
  <div>
    <label>
      Title: <input type="text" id="title-input" value={title} onChange={e => setTitle(e.target.value)} />
    </label>
  </div>
```

Kuva 17. React lomakkeen html

Kuvassa 17 lomake tehdään antamalla form tagille onSubmit attribuutti. Ensin pitää ajaa preventDefault, jotta sivu ei uudelleenohjaudu ja lataa itseään uudestaan. Kenttä

tehdään ohjatuksi lisäämällä input tagille value ja onChange attribuutit. Funktiossa onChange voidaan tehdä validointia.

Esimerkissä muutetaan myös ohjelman aloitusta. Muistiinpanojen alustamisessa ladataan kovakoodatut testi muistiinpanot vain, jos ei ole tallennettu yhtäkään muistiinpanoa.

```
20    /**
21     * createdCb
22     * @param {import('./api/todotype').TodoItem} todo
23     */
24    const createdCb = todo => {
25      todo = AddItem(todo);
26
27      setItems(p => p.concat(todo));
28    };
29    return (
30      <div>
31        <AddNewForm createdCb={createdCb} />
32        {items.map(todoItem => (
33          <TodoListItem key={todoItem.id} todoItem={todoItem} />
34        ))}
35      </div>
36    );
37  }
38
```

Kuva 18. React eventin vastaanottaminen

Kuvassa 18 pääkomponenttiin lisätään funktio uuden muistiinpanon vastaanottamiseksi lomakkeelta. Tiedon päivittämiseksi ajetaan setItems funktio, jossa käytetään funktiota sen sijaan että käytettäisiin suoraan vanhaa tilaa. Näin vältetään mahdollinen virhe, jos yritetään lisätä kaksi muistiinpanoa täysin samaan aikaan. On mahdollista, että toisen muistiinpanon lisäämisen aikaan ensimmäistä ei ole tallennettu tilaan, ja näin toinen yliajaisi sen.

6.3 Vue

Vuessa tehdään hieman samalla tavalla kuin Angularin Template tapa. Input tageihin lisätään kuvassa 19 v-model attribuutti, jolla tagi linkitetään muuttujaan komponentissa.

```
<form v-on:submit.prevent="formSubmitted">
  <div>
    <label for="name">
      Title
    </label>
    <input id="title" type="text" v-model.trim="title" />
  </div>
```

Kuva 19. Vue lomakkeen html

Muuten Vue näyttää aika samanlaiselle kuin Angular. Vuessa on muutama sisäänrakennettu lisäys tapahtumille. Tässä käytetty prevent ajaa preventDefault koodin tapahtumalle suoraan, joten komponentin funktiossa ei tarvitse miettiä sitä. Käytetty trim poistaa välilyönnit syötetyn tekstin ympäriltä.

Itse eventin lähettäminen pääkomponentille on myös aika samanlaista, mutta pieniä eroja esiintyy. Vue:ssa ei ole omaa Output muuttujaa, vaan tässä käytetään $emit globaalia funktiota, jolle annetaan tekstillä eventin nimi. Tämän nimen pitää olla kirjoitettu pienin kirjaimin. Pääkomponentti voi sitten kuunnella tätä samalla v-on attribuutilla.

Huonona puolena tässä on, ettei ole minkäänlaista tarkistusta tekovaiheessa onko kuunneltu oikeaa funktiota ja oikeilla parametreilla. Reactissa tyypit varmistivat, että on oikeat parametrit ja menevät oikealla nimellä. Angularissa myös nimi on varmistettu, mutta parametrien tyypit eivät välity automaattisesti.

7 TUTKITTUJEN TEKNIIKOIDEN ARVIOINTI

Verratuissa tekniikoissa ei ole yhtäkään selvästi ylitse muiden. Kaikilla saatiin tehtyä likimain yhtä tehokkaasti haluttu kokonaisuus, mutta eri osat veivät enemmän koodausaikaa. Annetaan jokaiselle tekniikalle pisteet yhdestä kolmeen ominaisuuksista yksinkertaisuus, monipuolisuus ja työkalut. Yksinkertaisuus kertoo, kuinka helppoa on aloittaa kirjaston käyttö sovelluksissa. Monipuolisuus on, kuinka helppoa on käyttää eri toiminnallisuuksia ja liittää muita kirjastoja yhteen. Työkalut kertovat, miten hyvin tekstieditori tukee koodaamista.

	Yksinkertaisuus	Monipuolisuus	Työkalut
React	2	3	1
Angular	3	2	2
Vue	1	1	3

8 JOHTOPÄÄTÖKSET

8.1 Yksinkertaisuus

Yksinkertaisuudessa Vue saa parhaat pisteet. Se saa pisteet yksinkertaisesta projektin rakenteesta sekä normaalista css:n ja html:n syntaksista. Vuessa on kuitenkin ylimääräistä ylläpidettävää tarvittavien komponenttien lisäämiseksi taulukkoon. Reactissa tätä ei tarvinnut tehdä.

React tulee seuraavana, sillä html piti kirjoittaa javascriptin sisään. Tällä tulee ajoittain epäselvää koodia.

Angularin projektirakenne on paljon monimutkaisempi kuin kummallakaan toisella. Komponentit pitää lisätä moduuliin, jolloin ei näe suoraan mikä komponentti käyttää mitäkin ominaisuutta. Tämä on myös ylimääräinen osa luotaessa uusia komponentteja.

8.2 Monipuolisuus

Vue saa eniten pisteitä myös monipuolisuudessa, sillä se otti vastaan tyylit ja tietorakenteen ongelmitta.

Angular jäi pisteissä hieman jälkeen, sillä tyylit kyllä toimivat suoraan, mutta tietorakenne ei. Ne vaativat oman servicen ja moduuliin lisäämisen.

React tulee tässä osiossa viimeiseksi, sillä sekä tyyliä että html-merkintöjä ei voi käyttää suoraan. Css vaatii joko oman kirjaston, joka ei sekään mahdollista selvää syntaksia, tai enemmän organisointia tyyliluokkien kanssa.

8.3 Työkalut

Työkaluissa React on selvästi huipulla. Editori osaa kertoa suoraan, mitä mikäkin komponentti ja funktio ottaa vastaan attribuutteina ja parametreina. Ei ole mahdollista

antaa väärän muotoisia funktioita tai muuttujia. Työkalut myös kertovat heti, jos jokin tieto on antamatta komponentille.

Angular on tässä asiassa melkein yhtä hyvä. Siinä editori kyllä kertoo minkä nimisiä muuttujia, funktioita ja komponentteja voi käyttää missäkin, mutta tyypit eivät mene läpi komponenttirajojen automaattisesti. Komponentti ei myöskään voi pakottaa attribuutteja asetettaviksi.

Vue on tässä kohdassa heikoin. Editori ei osaa kertoa sallittuja nimiä eikä tyyppejä eri osissa koodia. Myös editoria auttavien tyyppien kirjoittaminen eri osille komponenttia on paljon vaikeampaa kuin muilla.

8.4 Lopputulos

Vue	React	Angular
1,67	2	2,33

Lopputuloksena sijoituksien keskiarvoissa Vue on ensimmäinen, vaikka sen kirjoittaminen ei kuitenkaan ole yhtä tuettua kuin muilla. Varsinkin suuremmissa projekteissa työkaluille annetaan suurempi painoarvo, sillä ne helpottavat ylläpidettävyyttä. Reactia on huomattavasti mukavampi kirjoittaa, sillä editori osasi täydentää käytettyjä funktioita ja lisätä halutut komponentit tiedostoon itsestään. Angularillakin on hyvät puolensa. Siinä on helpompi käyttää normaalia html merkintöjä kuin Reactissa sekä se omaa paremman tuen sen kirjoittamiseen editorissa kuin Vue.

LÄHTEET

Angular. (13. 5 2020). Noudettu osoitteesta Getting Started: https://angular.io/start

Arinich, G. (7. 11 2016). *Goshakk*. Noudettu osoitteesta Controlled and uncontrolled form inputs in React don't have to be complicated: https://goshakkk.name/controlled-vs-uncontrolled-inputs-react/

Cromwell, V. (30. 5 2017). *FreeCodeCamp*. Noudettu osoitteesta Between the Wires: An interview with Vue.js creator Evan You: https://www.freecodecamp.org/news/between-the-wires-an-interview-with-vue-js-creator-evan-you-e383cbf57cc4/

Gimeno, A. (27. 3 2018). *Medium*. Noudettu osoitteesta The deepest reason why modern JavaScript frameworks exist: https://medium.com/dailyjs/the-deepest-reason-why-modern-javascript-frameworks-exist-933b86ebc445

inVerita. (31. 3 2020). *becominghuman*. Noudettu osoitteesta Most popular JavaScript frameworks in 2020: https://becominghuman.ai/most-popular-javascript-frameworks-in-2020-e60534b14b36

MDN web docs. (13. 5 2020). Noudettu osoitteesta Local Storage: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

Node. (13. 5 2020). Noudettu osoitteesta https://nodejs.org/en/

React. (5. 13 2020). Noudettu osoitteesta Getting Started: https://reactjs.org/docs/getting-started.html

ReactJs. (5. 13 2020). Noudettu osoitteesta Create a New React App: https://reactjs.org/docs/create-a-new-react-app.html

Stepnov, E. (17. 10 2019). *Flatlogic*. Noudettu osoitteesta UI JAVASCRIPT FRAMEWORKS AND LIBRARIES FOR WEB DEVELOPMENT: https://flatlogic.com/blog/ui-javascript-frameworks-and-libraries-for-web-development/

W3Schools. (5. 13 2020). Noudettu osoitteesta How TO - Cards: https://www.w3schools.com/howto/howto_css_cards.asp

Wanyoike, M. (16. 10 2018). *LogRocket*. Noudettu osoitteesta History of front-end frameworks: https://blog.logrocket.com/history-of-frontend-frameworks/

VueJs. (5. 13 2020). Noudettu osoitteesta Guide: https://vuejs.org/v2/guide/

Zheng Gao, C. B. (2017). *To Type or Not to Type: Quantifying Detectable Bugs in JavaScript.* IEEE/ACM 39th International Conference on Software Engineering (ICSE).

Angular. (13. 5 2020). Noudettu osoitteesta Getting Started: https://angular.io/start

Arinich, G. (7. 11 2016). *Goshakk*. Noudettu osoitteesta Controlled and uncontrolled form inputs in React don't have to be complicated: https://goshakkk.name/controlled-vs-uncontrolled-inputs-react/

Cromwell, V. (30. 5 2017). *FreeCodeCamp*. Noudettu osoitteesta Between the Wires: An interview with Vue.js creator Evan You: https://www.freecodecamp.org/news/between-the-wires-an-interview-with-vue-js-creator-evan-you-e383cbf57cc4/

Gimeno, A. (27. 3 2018). *Medium*. Noudettu osoitteesta The deepest reason why modern JavaScript frameworks exist: https://medium.com/dailyjs/the-deepest-reason-why-modern-javascript-frameworks-exist-933b86ebc445

inVerita. (31. 3 2020). *becominghuman*. Noudettu osoitteesta Most popular JavaScript frameworks in 2020: https://becominghuman.ai/most-popular-javascript-frameworks-in-2020-e60534b14b36

MDN web docs. (13. 5 2020). Noudettu osoitteesta Local Storage: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

Node. (13. 5 2020). Noudettu osoitteesta https://nodejs.org/en/

React. (5. 13 2020). Noudettu osoitteesta Getting Started: https://reactjs.org/docs/getting-started.html

ReactJs. (5. 13 2020). Noudettu osoitteesta Create a New React App: https://reactjs.org/docs/create-a-new-react-app.html

Stepnov, E. (17. 10 2019). *Flatlogic*. Noudettu osoitteesta UI JAVASCRIPT FRAMEWORKS AND LIBRARIES FOR WEB DEVELOPMENT: https://flatlogic.com/blog/ui-javascript-frameworks-and-libraries-for-web-development/

W3Schools. (5. 13 2020). Noudettu osoitteesta How TO - Cards: https://www.w3schools.com/howto/howto_css_cards.asp

Wanyoike, M. (16. 10 2018). *LogRocket*. Noudettu osoitteesta History of front-end frameworks: https://blog.logrocket.com/history-of-frontend-frameworks/

VueJs. (5. 13 2020). Noudettu osoitteesta Guide: https://vuejs.org/v2/guide/

Zheng Gao, C. B. (2017). *To Type or Not to Type: Quantifying Detectable Bugs in JavaScript.* IEEE/ACM 39th International Conference on Software Engineering (ICSE).