Tampereen yliopisto

Miikka Toivola

KATSAUS JAVASCRIPT-SOVELLUSKEHYKSIIN

Informaatioteknologian ja viestinnän tiedekunta
Kandidaattitutkielma
Toukokuu 2021

TIIVISTELMÄ

Miikka Toivola: Katsaus JavaScript-sovelluskehyksiin
Kandidaattitutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Toukokuu 2021

Tämä tutkielma on kirjallisuuskatsaus frontend-kehityksestä ja siinä käytettävistä JavaScript-sovelluskehyksistä. Frontend-kehitys on sovelluskehityksen osa-alue, joka keskittyy sovelluksien käyttöliittymien kehittämiseen. Tutkielmassa esiteltyjä teknologioita käytetään työpöytä-, web- ja mobiilisovelluksissa.

Erilaisia frontend-kehityksessä käytettäviä JavaScript-sovelluskehyksiä on runsaasti. Tässä tutkielmassa esitellään kolme kehystä: Vue, React ja Angular. Valitut kehykset ovat avoimen lähdekoodin projekteja, joita voi käyttää vapaasti kaikenlaisissa projekteissa. Tämä oli yksi tarkasteltavien kehysten valintaperuste. Lisäksi kaikki kolme kehystä ovat erittäin suosittuja kehittäjien keskuudessa. Vue.js:llä on tähtiä GitHubissa 183 000, kun Reactilla on 168 000 ja Angularilla 72 000.

Tutkielman tavoitteena on esitellä sovelluskehykset ja frontend-kehityksen konsepteja yleisesti ja tätä kautta helpottaa sovelluskehyksen valintaa projekteissa. Kehyksiä tarkastellaan keskeisten ominaisuuksien sekä esimerkkikoodien kautta, eikä kehyksiä ole tarkoitus laittaa paremmuusjärjestykseen.

Tutkielmaa tehdessä ei selvinnyt selvää syytä, miksi kannattaisi valita projektiin jokin kehys toisen sijaan. Ainoastaan Vuen ja Reactin käyttämä virtuaalinen dokumenttiobjektimalli näytti tuovan selkeää etua suorituskykyä ajatellen.

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla

Avainsanat: frontend, sovelluskehys, JavaScript, Vue, Angular, React

Sisällysluettelo

1 Johdanto

Sovelluskehitys on yksi nykypäivän parhaiten työllistävä teollisuuden ala ja alan ammattilaisista on pulaa. Yksi syy tähän voi olla, että käytössä olevia teknologioita on valtava määrä ja niiden kaikkien hallitseminen voi olla hankalaa. Yritykset rekrytoivatkin nyt usein *asiakassovelluksien* (frontend) ja *taustasovelluksien* (backend) kehittäjiä pelkästään näihin tehtäviin. Tässä tutkielmassa tarkastellaan sovelluskehitystä asiakassovellusten näkökulmasta.

Tutkielmaan valikoin tarkasteltavaksi kolme suosittua frontend-*sovelluskehystä* (framework) Vue.js, React ja Angular. Kaikkien edellä mainittujen kehysten lähdekoodi on kirjoitettu JavaScriptillä tai TypeScrptillä ja niillä ohjelmoitavat sovellukset kirjoitetaan samoilla kielillä. JavaScript on yleisin frontend-sovelluskehityksessä käytetty ohjelmointikieli (Gizas et al., 2012).

Tutkielman toisessa luvussa tarkastellaan JavaScriptiä ja sen ominaisuuksia. Kolmannessa luvussa tutustutaan yleisesti frontend-kehitykseen sekä siinä käytettäviin tekniikoihin ja teknologioihin. Neljännessä luvussa esitellään aiemmin mainitut kolme frontend sovelluskehystä: Vue.js, React ja Angular. Viidennessä luvussa tehdään yhteenveto tässä tutkielmassa esitellyistä sovelluskehyksistä ja pohditaan minkälaisissa projekteissa niitä voi hyödyntää.

2 JavaScript

JavaScript on ohjelmointikieli, jonka kehitti Netscape vuonna 1995 Netscape Navigator selaimelle kevyemmäksi vaihtoehdoksi Java-appleteille. Kielen avulla verkkosivuille voidaan lisätä interaktiivisuutta sekä animaatioita (Fruhlinger, 2019). Kieli on tulkattava, joka tarkoittaa sitä, että erillinen tulkkiohjelma, esimerkiksi V8, käsittelee ja suorittaa ohjelmakoodin. JavaScriptin standardoinnista vastaa European Computer Manufacturers Association (ECMA, 2021). JavaScriptin standardoitua versioita kutsutaankin nimellä ECMAScript. (Fruhlinger, 2019)

JavaScriptin alkuvaiheissa oli yleistä, että selaimet eivät sallineet sen suorittamista oletuksena, mutta nykyään se tekisi monista verkkosivuista käyttökelvottomia. JavaScript onkin nykyään sovelluskehittäjien keskuudessa suosituin opiskeltava ohjelmointikieli. Se on helposti lähestyttävä ja helppo oppia. (Fruhlinger, 2019)

Tänä päivänä JavaScriptillä voidaan tehdä sovelluksia verkkosivulle, mobiililaitteille ja palvelimille. Myös perinteiset työpöytäsovellukset voidaan kirjoittaa JavaScriptillä. JavaScript-ohjelmia kehitetään usein yhdessä HTML- ja CSS-teknologioiden kanssa.

Mobiililaitteiden natiiviominaisuuden kuten kamera ja tiedostonhallinta. saadaan käyttöön käyttämällä paketoijasovelluksia kuten Apache Cordova.

JavaScript on heikosti tyypitetty kieli, joka tarkoittaa, että muuttujia voidaan vertailla keskenään tyypistä riippumatta. Esimerkiksi merkkijonoa ”1” voidaan verrata lukuun 1, ja tulos palauttaa totuusarvon tosi (koodiesimerkki 1).

```
var a = "1"
var b = 1

console.log(a == b) // true
```

Koodiesimerkki 1. Muuttujien vertailu.

Tarkkaa vertailua käyttämällä vältytään virheiltä. Koodiesimerkissä 2 tehdään sama vertailu kuin koodiesimerkissä 1, mutta tällä kertaa käytetään tarkkaa vertailua.

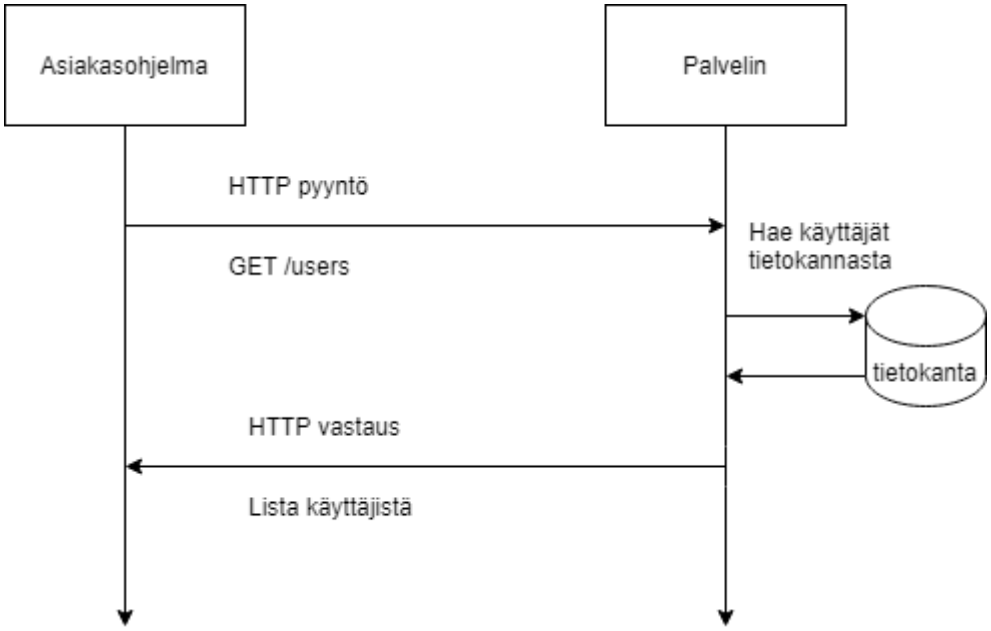
```
var a = "1"
var b = 1

console.log(a === b) // false
```

Koodiesimerkki 2. Muuttujien tarkkavertailu.

3 Frontend -kehitys

Frontend-kehityksellä tarkoitetaan asiakassovelluksen ja sen käyttöliittymän toteuttamista esimerkiksi verkkosivulle tai mobiililaitteelle. Käyttöliittymä on se osa sovelluksesta, jonka järjestelmän käyttäjä näkee ja jonka kanssa käyttäjä on vuorovaikutuksessa. Käyttöliittymä koostuu yleensä painikkeista, syötekentistä, graafisista elementeistä kuten kuvista.

Laajemmissa järjestelmissä asiakassovellus keskustelee yleensä taustasovelluksen kanssa, joka vastaa esimerkiksi käsiteltävän datan pysyvästä tallennuksesta. Taustasovellus sijaitsee palvelimella. Tämä asiakassovelluksen ja palvelimen välinen keskustelu välitetään tyypillisesti HTTP-protokollaa käyttäen. Asiakassovellus lähettää palvelimelle *pyynnön* (request), johon palvelin *vastaa* (response) (Danielyan, 2001). Kuvassa 1 on kuvattu yhden pyynnön ja vastauksen välisen keskustelun vaiheet, jossa asiakassovellus pyytää listaa käyttäjistä palvelimelta.



Kuva 1. HTTP-pyyntö ja HTTP-vastaus.

3.1 Single page application (SPA)

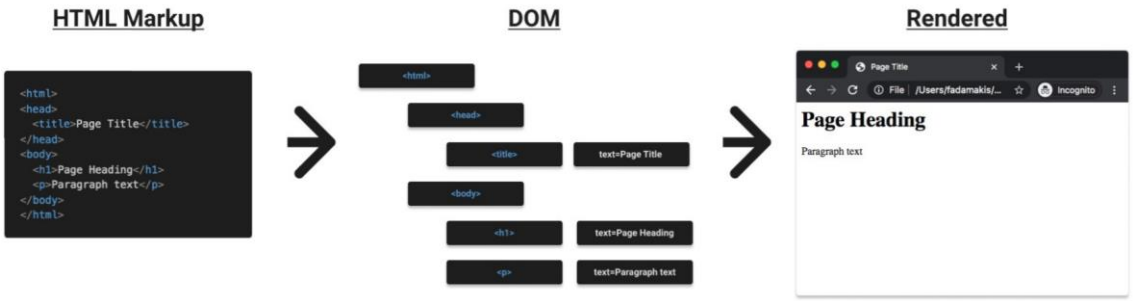
Modernit web-sovellukset toteutetaan usein *yhden sivun sovelluksina* (single page application, SPA). SPA-sovellukset luovat sivut tai sivujen osan dynaamisesti asiakassovelluksessa. Tämä toteutustapa poikkeaa huomattavasti perinteisestä tavasta, jossa palvelin joko lähettää käyttäjälle staattisesti luodun sivun tai tuottaa sen dynaamisesti taustasovelluksessa. Flanagan (2011) mukaan SPA on sovellus, joka tarvitsee vain yhden sivulatauksen palvelimelta. Joidenkin SPA-sovelluksien ei tarvitse keskustella palvelimen kanssa ensimmäisen sivulautauksen jälkeen kertaakaan.

3.2 AJAX

AJAX (Asynchronous JavaScript and XML) on SPA-sovelluksissa käytetty tekniikka, joka mahdollistaa asiakassovelluksen ja palvelimen välisen keskustelun ilman, että koko sivua tarvitsee ladata uudelleen. Tämä parantaa asiakassovelluksen eli verkkosivun tai mobiilisovelluksen käytettävyyttä. Asiakassovelluksessa voidaan esimerkiksi päivittää jokin tietty osa ja silti sovelluksen muut osat ovat käytettävissä. Ilman AJAX-tekniikkaa sovelluksen pitäisi odottaa vastausta palvelimelta, ja tällöin myös sovelluksen muut osat olisivat poissa käytöstä.

3.3 Dokumenttiobjektimalli ja virtuaalinen dokumenttiobjektimalli

Dokumenttiobjektimalli (document object model, DOM) on rajapinta, joka kuvaa HTML-dokumentin puurakenteena, jossa jokainen solmu edustaa osaa dokumentista (Adamakis 2020). Kuvassa 1 on HTML-dokumentin DOM-rakenne sekä selaimen käyttäjälle *muuntama* (render) sivu.



Kuva 2. Dokumenttiobjektimalli (Adamakis, 2020).

Virtuaalinen dokumenttiobjektimalli (virtual document object model) esittää oikean DOM:n JavaScript-objekteina. Kuvassa 2 nähdään esimerkki yhdestä virtuaalisesta DOM-solmusta.

Kuva 3. Virtuaalinen DOM-solmu (Adamakis, 2020).

Virtuaalinen DOM seuraa sivulla tapahtuvia muutoksia ja päivittää tarvittavat muutokset oikeaan DOM:iin. Tämä tapa on huomattavasti nopeampaa kuin elementin etsiminen oikeasta DOM:sta ja sen päivittäminen. (Vuejs.org, 2020)

4 JavaScript-sovelluskehykset

JavaScript-sovelluskehykset ovat käytännöllisiä, kun toteutetaan laajoja sovelluksia. Ne määrittelevät oman korkeamman tason rajapinnan (application programming interface,

API). Kehysten käyttö pakottaa sovelluskehittäjän käyttämään kehyksen määrittelemää rajapintaa, mikä tekee kehittämisestä johdonmukaista, koska kaikkien kehittäjien on kirjoitettava sovellusta samalla tavalla. Kehysten avulla kehittäjä pystyy toteuttamaan toimintoja kirjoittamalla vähemmän koodia sekä ratkaisemaan monia selainten välisiä yhteensopivuusongelmia sekä tietoturva- ja saavutettavuusongelmia. (Flanagan, 2011)

Seuraavaksi esitellään kolme JavaScript-sovelluskehystä: Vue, React ja Angular. Valitut kehykset ovat avoimen lähdekoodin projekteja, joita voi käyttää vapaasti kaikenlaisissa projekteissa. Lisäksi kaikki kolme kehystä ovat erittäin suosittuja kehittäjien keskuudessa. Kehyksiä tarkastellaan keskeisten ominaisuuksien sekä esimerkkikoodien kautta, eikä niitä ole tarkoitus laittaa paremmuusjärjestykseen.

4.1 Vue.js

Vue.js on progressiivinen sovelluskehys, joka on helppo integroida muiden kirjastojen kanssa tai lisätä vanhoihin projekteihin. Sitä käytetään myös modernien SPA-sovellusten kehitykseen. (Vuejs.org, 2021)

Vue.js:n on kehittänyt entinen Googlen Angular-kehittäjä Evan You. Youn tavoite oli tehdä kevyt sovelluskehys, joka sisältäisi AngularJS:n parhaat ominaisuudet (Kyoreva 2017). Vue.js:n ensimmäinen versio julkaistiin helmikuussa 2014. Vue.js:n suosio kasvoi nopeasti ja tätä kirjoittaessa sillä on enemmän tähtiä GitHubissa kuin Angularilla ja Reactilla.

4.1.1 Ominaisuudet

Kuten aikaisemmin mainittiin Vue.js-sovellukset voivat olla pieniä yhden toiminnon suorittavia ohjelmia tai suurempia sovelluskokonaisuuksia. Vue.js-sovellukset ovat hyvin suorituskykyisiä, koska sovelluskehys käyttää virtuaalista dokumenttiobjektimallia (Adamakis, 2020). Sovelluksen koosta riippumatta Vue.js rakentuu erilaisista komponenteista. Komponentti voi olla esimerkiksi yksittäinen nappi tai useamman komponentin yhteen kokoava kokonaisuus.

Rakennettaessa suurempia sovelluksia tarvitaan yleensä myös *tilanhallintaa* (state management) ja *reititystä* (routing). Tilanhallintaan voidaan esimerkiksi tallentaa tieto käyttäjän onnistuneesta kirjautumisesta tai tieto käyttäjän valitsemasta teemasta sovelluksessa. Vue.js tilanhallinta-kirjasto on nimeltään Vuex.

Navigointi sovelluksessa tapahtuu usein *URL:ia* (uniform resource locator) vaihtamalla. URL:n muutoksia seuraa sovelluskehyksissä yleensä reititin-komponentti,

joka lataa URL:a vastaavan *näkymän* (view). Vue.js-sovelluksissa reitittimenä käytetään erillistä Vue-router-reititinkirjastoa.

Muita Vue.js-sovelluksien kehityksessä käytettäviä työkaluja ovat muun muassa Vue-cli, joka helpottaa projektien luomista ja ylläpitämistä sekä Vue-devtools-*selainlaajennus* (browser extension), joka tekee virheiden etsinnästä helpompaa.

4.1.2 Esimerkkisovellus

Katsotaan seuraavaksi esimerkki yksinkertaisesta Vue.js-sovelluksesta.

```
// index.html
<!DOCTYPE html>
<html>
   <head>

<title>Esimerkkisovellus</title>
   </head>
   <body>
     <div id="app"></div>
   </body>
</html>
---------------------
//main.js
import Vue from 'vue'
import App from './App'

new Vue({
    render: (h) => h(App),
}).$mount('#app')

---------------------
```

```
//App.vue
<template>
  <div>
    <h1>{{ hello }}</h1>
  </div>
</template>
<script>
export default {
  name: 'App',
    data() {
      hello: 'Hello World'
    }
}
</script>
<style>
   .hello {
     font-size: 36px;
   }
</style>
```

Koodiesimerkki 3. Vue.js sovellus.

Koodiesimerkissä 3 on kolme tiedostoa: index.html, main.js ja App.vue. Selain lataa ensimmäiseksi index.html-tiedoston, joka lataa main.js-tiedoston. Main.js-tiedostossa luodaan uusi Vue-instanssi, jolle annetaan asetuksissa App-komponentti, joka on tuotu

main.js:lle import-komennolla. Lopuksi kerrotaan mihin kohtaan index.html instanssi luodaan. Tämä tehdään komennolla .$mount('#app'), joka luo sovelluksen index.html-tiedostossa olevaan div-elementtiin, jonka id-attribuutti on app.

App-komponentti sisältää kolme osaa: template, script ja style. Template-osassa määritellään komponentin rakenteellinen ulkoasu. Script-osa sisältää komponentin sovelluslogiikan ja muuttujat. Style-osassa voidaan tehdä tarkempia tyylimäärittelyitä.

4.2 React

React on Facebookin ylläpitämä JavaScript-kirjasto. Sen kehittäjänä pidetään Jordan Walkea, joka vuonna 2011 kehitti FaxJS:n, Reactin prototyypin. Vuonna 2013 React julkaistiin avoimen lähdekoodin (open source) projektina JS ConfUS -tapahtumassa. (Hámori 2018)

Kuten Vuejs, myös React käyttää virtuaalista dokumenttiobjektimallia, joka takaa sillä toteutetuille sovelluksille hyvän suorituskyvyn.

4.2.1 Deklaratiivinen ohjelmointitapa

React ja Vuejs käyttävät *deklaratiivistä* (declarative) ohjelmointiparadigmaa. Vastakohtana tälle on perinteisempi *imperatiivinen* (imperative) paradigma, jota käytetään muun muassa Javalla, C++:lla ja C:llä ohjelmoitaessa. Deklaratiivisen ja imperatiivisen paradigman ero voitaisiin selittää seuraavasti: deklaratiivisessa tavassa tietokoneelle kerrotaan mitä halutaan, välittämättä siitä, miten haluttu toiminto on toteutettu ja imperatiivisessa tavassa tietokoneelle annetaan tarkat ohjeet, miten toiminto pitää suorittaa. Ian Mundyn (2017) koodiesimerkissä 4 on kuvattu imperatiivinen toteutus, jossa ruudulle lisätään nappi ja koodiesimerkissä 5 deklaratiivinen toteutus samasta toiminnosta.

```
const container = document.getElementById('container');
const btn = document.createElement('button');
btn.className = 'btn red';
btn.onclick = function(event) {
  if (this.classList.contains('red')) {
    this.classList.remove('red');
    this.classList.add('blue');
  } else {
    this.classList.remove('blue');
    this.classList.add('red');
  }
};
container.appendChild(btn);
```

Koodiesimerkki 4. Imperatiivinen tapa (Mundy, 2017)

```
class Button extends React.Component{
  this.state = { color: 'red' }
  handleChange = () => {
    const color = this.state.color === 'red' ? 'blue' : 'red';
    this.setState({ color });
  }
  render() {
     return (<div>
       <button
         className=`btn ${this.state.color}`
         onClick={this.handleChange}>
       </button>
     </div>);
  }
}
```

Koodiesimerkki 5. Deklaratiivinen tapa (Mundy, 2017)

Koodiesimerkissä 4 nähdään, kuinka sovellukselle on annettu tarkat ohjeet: ensin tehdään *säiliö*-elementti (container), jonka jälkeen tehdään nappi. Tämän jälkeen napille ohjelmoidaan toiminnot sitä painettaessa ja lopuksi nappi lisätään säiliö-elementtiin.

Näiden kahden koodin ero on pieni, mutta jälkimmäisessä esimerkissä ei muokata DOM-elementtiä ollenkaan, vaan määritellään, mitä käyttäjälle näytetään perustuen sovelluksen tilaan. (Mundy, 2017)

4.2.2 JSX

React-sovellukset kirjoitetaan yleensä *JSX:llä.* JSX on JavaScriptin syntaksin laajennus, joka mahdollistaa JavaScriptin kirjoittamisen merkintäkielen sekaan. JavaScript-lausekkeet kirjoitetaan käyttämällä aaltosulkeita ”{}” merkintäkielen sisällä, näitä voivat olla muun muassa yksittäisen muuttujan arvon esittäminen tai funktiokutsun tuloksen esittäminen käyttöliittymässä. (reactjs.org, 2021)

Katsotaan seuraavaksi koodiesimerkkiä 6, jossa on käytetty JSX:ää. Esimerkissä esitellään muuttuja name, jonka arvo tulostetaan <h1> elementin sisään käyttäen aaltosulkeita.

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;
```

Koodiesimerkki 6. JSX (reactjs.org, 2021).

4.2.3 Esimerkkikomponentti

Seuraavissa koodiesimerkeissä on esitetty yksinkertainen React-komponentti, joka tulostaa näytölle tekstin ”Hello World”. Reactissa komponentti voidaan kirjoittaa funktionaalista tapaa tai luokkaesitystä käyttäen. Esimerkissä 7 on käytetty funktionaalista React-komponenttia ja esimerkissä 8 on sama komponentti kirjoitettuna luokkaa käyttäen.

```
import React from "react";

function Hello() {
    return <h1>Hello World</h1>;
}
```

Koodiesimerkki 7. Funktionaalinen React-komponentti.

```
import React from "react";

class Hello extends React.Component {
  render() {
    return <h1>Hello World</h1>;
  }
}
```

Koodiesimerkki 8. React-luokkakomponentti.

4.3 Angular

Angular on kehitysalusta, joka sisältää muun muassa komponenttipohjaisen sovelluskehyksen, kokoelman integroituja kirjastoja sekä erilaisia kehitystyökaluja, jotka helpottavat esimerkiksi sovelluksien testausta ja ylläpitoa. Sen ympärille on kehittynyt 1,7 miljoonan kehittäjän yhteisö. (Angular.io, 2021)

Angularin kehitti alun perin Googlen työntekijä Misko Hevery omana sivuprojektinaan. Tämä sivuprojekti julkaistiin vuonna 2010 nimellä AngularJS, jota kutsutaan myös Angularin versioksi 1. Vuonna 2014 Googlen kehitystiimi päätti kirjoittaa Angularin uudelleen, josta myöhemmin tuli nykypäivänä käytössä olevan Angularin ensimmäinen versio Angular 2.0. (Gavigan, 2018)

Angular on tämän tutkielman sovelluskehyksistä ainoa, joka on kirjoitettu TypeScriptillä. TypeScript perustuu JavaScriptiin ja se sisältää JavaScriptin toiminnallisuudet sekä niiden lisäksi esimerkiksi staattisen muuttujien tyypityksen. TypeScript on avoimen lähdekoodin projekti ja sen kehityksestä vastaa Microsoft. (typescriptlang.com, 2021)

4.3.1 Esimerkkisovellus

Koodiesimerkissä 9 on esitetty Angular-komponentti, joka tulostaa näytölle Hello World tekstin. Muista tutkielman kehyksistä poiketen, Angular komponentti jaetaan yleensä useampaan tiedostoon. Esimerkissä app.compnent.ts-tiedosto sisältää komponentin sovelluslogiikan ja mahdolliset riippuvuudet. Selector-ominaisuus kertoo mihin kohtaa DOM:ia komponetti tulee asettaa. TemplateUrl- ja styleUrls ominaisuudet määrittelevät, mistä tiedostosta HTML-rakenne ja CSS tyylit ladataan, tässä esimerkissä ne sijaitsevat samassa hakemistossa app.component.ts-tiedoston kanssa.

```
//app.component.ts
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  message: string = 'Hello World';
}

// app.component.html
<div>
     <h1 class=”title”> {{message}} </h1>
</div>

//app.component.css
.title {
  color: red;
}
```

Koodiesimerkki 9. Angular-komponentti.

5 Yhteenveto

Tässä tutkielmassa tarkasteltiin frontend-sovelluskehitystä ja sen merkittäviä konsepteja yleisesti ja hyvin pintapuolisesti. Lisäksi esiteltiin kolme suosittua sovelluskehystä Vue, React ja Angular. Tutkielma tehtiin kirjallisuuskatsauksena eri lähteitä käyttäen. Aiheesta löytyy kohtuullisesti aineistoa ja hyvien sekä ajantasaisten kirjoitusten hakeminen oli ajoittain hankalaa.

Tutkielman tarkoitus oli lisätä tietoisuutta aiheesta ja mahdollisesti helpottaa sovelluskehyksen valintaa uutta projektia aloitettaessa. Kolmesta esitellystä kehyksestä Vue näyttäisi olevan tällä hetkellä suosituin kehys kehittäjien keskuudessa. Vuella on tähtiä GitHubissa 183 000, kun Reactilla on 168 000 ja Angularilla 72 000. Pelkästään GitHub- tähtien perusteella kehystä ei kuitenkaan kannata valita. Vue ja React sopivat hyvin kaiken kokoisiin projekteihin, kun taas Angular on suunnattu enemmän suurempiin

projekteihin. Kaikista kolmesta kehyksestä löytyy nykyään myös TypeScript-tuki, joten sekään ei yksinään ole peruste kehyksen valinnalle.

Tutkielmaa tehdessä ei selvinnyt selvää syytä, miksi kannattaisi valita projektiin jokin kehys toisen sijaan. Ainoastaan Vuen ja Reactin käyttämä virtuaalinen DOM näytti tuovan selkeää etua suorituskykyä ajatellen.

Lähdeluettelo

Adamakis, F. (2020). Vue Virtual Dom. *https://medium.com/js-dojo/vue-virtual-dom-13af62d2be41*. (Haettu 15.2.2021)

Angular.io (2021). The modern web developer's platform. *https://angular.io/* (Haettu 9.2.2021)

Danielyan, E. (2001, 05). Understanding internet protocols: HTTP and HTTPS. *Inside Solaris, 7*,11-13.

ECMA (2021). European Computer Manufacturers Association. *https://www.ecma- international.org/* (haettu 19.5.2021)

Flanagan, D. (2011). JavaScript: The Definitive Guide, 6th Edition. O'Reilly Media, Inc., 2011. Print.

Fruhlinger, J. (2019). What is JavaScript? JavaScript and ECMAScript, explained. InfoWorld.Com, *https://www.infoworld.com/article/3441178/what-is-javascript-the-full-stack-programming-language.html* (Haettu 3.3.2021)

Gavigan D. (2018). The History Of Angular. *https://medium.com/the-startup-lab-blog/the-history-of-angular-3e36f7e828c7* (Haettu 26.3.2021)

Gizas A., Christodoulou S., Papatheodorou T. (2012). Comparative evaluation of JavaScript frameworks. In Proceedings of the 21st International Conference on World Wide Web (WWW '12 Companion). Association for Computing Machinery, New York, NY, USA, 513–514.

Hámori, F. (2018). The History of React.js on a Timeline. https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/ (Haettu 15.2.2021)

Kyoreva, K. (2017). State of the art JavaScript application development with vue.js. *Sofia: International Conference on Application of Information and Communication Technology and Statistics and Economy and Education (ICAICTSEE).* 567

Mundy, I. (2017). Declarative vs Imperative Programming. https://codeburst.io/declarative-vs-imperative-programming-a8a7c93d9ad2 (Haettu 15.2.2021)

Reactjs.org (2021). A JavaScript library for building user interfaces. *https://reactjs.org/* (Haettu 16.2.2021)

TypeScript (2021). Typed JavaScript at Any Scale. *https://typescriptlang.org/* (Haettu 10.2.2021)

Vuejs.org (2021). The Progressive JavaScript Framework. *https://vuejs.org/*. (Haettu 10.2.2021)