OULUN AMMATTIKORKEAKOULU

Toni Miettinen

JAVASCRIPT JA SIITÄ SYNTYNEET ALUSTAT

JAVASCRIPT JA SIITÄ SYNTYNEET ALUSTAT

Toni Miettinen
Opinnäytetyö
Syksy 2021
Tietojenkäsittely
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma

Tekijä(t): Toni Miettinen
Opinnäytetyön nimi: JavaScript ja siitä syntyneet alustat
Työn ohjaaja(t): Teppo Räisänen
Työn valmistumislukukausi ja -vuosi: Syksy 2021 Sivumäärä: 38

Tämän opinnäytetyön tarkoituksena on tutkia JavaScriptin historiaa sekä sen eri käyttötarkoituksia. Työn tavoitteena on tutustua tarkemmin kieleen itsessään ja siitä syntyneisiin eri kehyksiin ja kirjastoihin, sekä mihin eri käyttötarkoituksiin niitä käytetään. Lisäksi katsotaan millä lailla nämä eroavat toisistaan, sekä miten ne käytännössä toimivat. Työn aihe on omavalintainen ja valittu oman mielenkiinnon pohjalta.

Toteutuksen aikana käytetään ohjelmoijille tarkoitettua tekstieditoria Visual Studio Code. Kaikkien alustojen rinnalla toimii myös tyylittely kehys Bootstrap, jota hyödynnetään sovellusten tyylittelyssä. Historia osiossa on käytetty lähteinä useita eri artikkeleja, osa ollen internetin digitaalisesta ”Wayback Machine” arkistosta.

Vertailun aikana kävi ilmi, että kaikki käsitellyt kehykset ovat hyviä vaihtoehtoja kehittäjien moniin eri tarpeisiin. Näistä haettuja eroavaisuuksia tehdyn esimerkki sovelluksen perusteella löytyi riittävä määrä, joista suurin osa liittyi komponenttien rakenteiden koostumuksiin, ja miten osa niiden funktioista toimi.

Kirjastojen osalta koin JQueryn olevan hyödyllinen, helposti käytettävä ja projektiin tuotava alusta. Toisen valitun kirjaston, BackboneJS, toisaalta vaikka vaikuttikin kätevältä, on jäänyt ajan myötä jälkeen muita käytännöllisiä ohjelmia joita käytetään ohjelmoinnin parissa.

Asiasanat: JavaScript, Visual Studio Code, Kehys, Kirjasto, Bootstrap, Komponentti, Wayback Machine

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Business Information Systems

Author(s): Toni Miettinen
Title of thesis: JavaScript and platforms born from it
Supervisor(s): Teppo Räisänen
Term and year when the thesis was submitted: Fall 2021 Number of pages: 38

The purpose of this thesis is to explore into the history of JavaScript and look into it's different intended uses. Goal is to become familiar with the language itself and with it's frameworks and libraries. In addition we will look how they differ from each other and how they basically function. The topic of the work was self selected and was chosen based on own interest on the topic.

During the course of implementation, we will be using a source code editor known as Visual Studio Code that is meant for programmers. Alongside with each framework and library, we'll be making a use of a styling framework called Bootstrap, which is used purely for the purpose of styling the programs. In history chapter, there have been used a few articles as sources, some of being from internet's digital archive, Wayback Machine.

While performing comparisons, it became apparent, that all of the processed frameworks are good choices for developers' many needs. Implemantation ended up showing a number of differences between them, most of being component and structure related and how some of their functions worked.

As far as the libraries go, JQuery was the most useful, with it being easiest to use and easiest to import into the project. Other chosen library, BackboneJS, although seeming competent, has fallen a bit behind from the modern tools that are alongside programming.

Keywords:

JavaScript, Visual Studio Code, Framework, Library, Bootstrap, Component, Wayback Machine

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena on perehtyä JavaScript kieleen, tutkia sen historiaa ja oppia ymmärtämään siitä syntyneitä lukuisia eri työkaluja. JavaScript (tai pelkästään JS) on suosittu ja käytetty oliopohjainen komentosarjakieli, jonka tehtävänä on tuoda toiminnallisuutta web-ympäristöihin sekä mahdollistaa useita eri interaktioita käyttäjien kanssa. Tehtyjen tutkimuksien mukaan noin 95% nykypäivän sivuista käyttää kieltä jossain määrin.

Kappaleet on jaettu neljään osaan alkaen historiasta, missä käydään JavaScriptin aikajana alusta loppuun, jonka jälkeen tehdään katsaus sen eri työkaluihin kappaleessa JavaScriptin perijät. Tavoitteena on saada työkaluista yleisymmärrys, ei varsinaisesti miten ne toimivat. Kappaleessa käsiteltäviin kehyksiin ja kirjastoihin perehdytään tarkemmin neljännessä kappaleessa, missä tuotetaan esimerkki sovellus, millä pyritään etsimään näiden väliset mahdolliset erot. Työn päätteeksi tehdään yleiskatsaus miten projekti eteni ja mikä oli työn lopputulos.

Työssä huomioitavaa ovat monet englannista suomeksi itse käännetyt termit, moni ollen tapauksia missä virallisia käännöksiä ei ollut saatavilla. Näiden kanssa on jätetty sijalle englannin kielen termi, jotta lukijalle ei synny hämmennystä mitä kyseinen termi tarkoittaa.

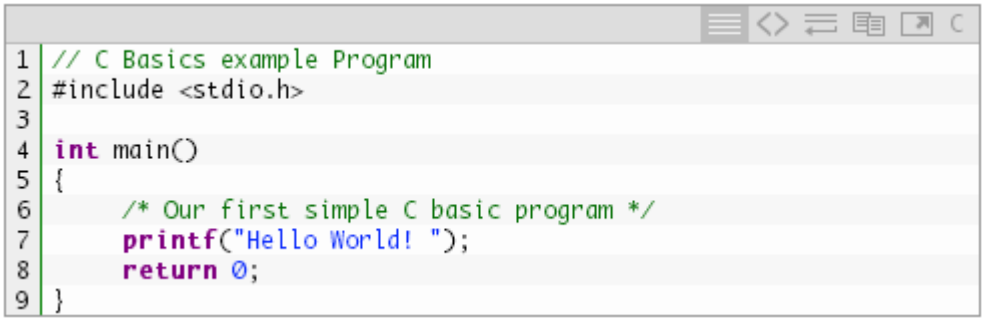
2 HISTORIA

1990-luvulla selaimet olivat vielä varhaisvaiheessa. World Wide Web (tai WWW) oli vasta syntynyt, ja yritykset pyrkivät etsimään ratkaisuja sen hyödyntämiseksi. Ensimmäisenä pääsi vauhtiin Marc Andreessenin kehittämä ”Mosaic” (1993), ensimmäinen graafinen WWW-selain. Andreessenin yritys “Mosaic Communications Corporation” siirtyi selaimesta kuitenkin pian eteenpäin, nimeämällä itsensä ”Netscape Communications Corporationiksi” (tai lyhyesti Netscape), sekä aloittamalla alusta uuden selaimen kanssa nimeltä ”Netscape Navigator” (1994). Seuraavana vuonna markkinoille astui uusi kilpailija, tunnettu tietokoneiden valmistaja Microsoft selaimellaan ”Internet Explorer”. (Peyrott 2017.)

2.1 Netscape

Vuonna 1995 Netscape ja Java-kielen kehittäjä, Sun Microsystems aloittivat yhteistyön, jossa tavoitteena oli tuoda Java selain käyttöön. Tämä idea päätettiin jättää lopulta kuitenkin välistä, sillä tuon aikainen yleinen mielipide oli, että Java ei soveltunut skriptaajille, harrastajille ja suunnittelijoille. Java koettiin ”liian isoksi sekä yritteliääksi”, joten päädyttiin ratkaisuun, jossa Java tuotaisiin ammattilaisten käyttöön. Kumppaniksi päätettiin luoda kieli, joka hoitaisi Javan puolesta pienemmät skriptaus tehtävät. (Peyrott 2017.)

Projektin kehitykseen valittiin ohjelmoija Brendan Eich, jonka tehtävänä oli luoda kieli, jonka syntaksi muistuttaisi Javan omaa, joka vuorostaan muistuttaa C-kieltä (Katso kuvio 1). Kielen luomisen sijaan harkittiin myös valmiita vaihtoehtoja: Python, Tcl sekä Scheme. Nämä kuitenkin erosivat liikaa Javasta. (Peyrott 2017.)



```c
// C Basics example Program
#include <stdio.h>

int main()
{
     /* Our first simple C basic program */
     printf("Hello World! ");
     return 0;
}
```

KUVIO 1. Tyypillinen C-Kielen syntaksi

Hän sai ensimmäisen prototyypin valmiiksi kymmenessä päivässä. Suunnittelun aikana, Eich otti vaikutteita Self- ja Scheme ohjelmointikielistä, ja pyrki pitämään syntaksin samankaltaisena kuin Javan. Hän antoi kielelle nimeksi ”Mocha”. Vaikkakin Mocha säilyi kielen sisäisenä nimenä, julkinen nimi vaihtui uudelleen pari kertaa, ensiksi LiveScript ja myöhemmin julkaisun lähestyessä lopulta JavaScript. (Peyrott 2017.)

Vuoden 1995 lopulla sopimus vietiin loppuun ja tehtiin kaupat. Sopimuksessa Mocha sai lopullisen nimensä ja sovittiin, että se hoitaisi pienet pääteohjelma tehtävät selaimessa, jolloin isommat tehtävät jäisivät Javalle. (Peyrott 2017.)

Syksyllä vuotta myöhemmin, Eich kävi Mochan läpi uudelleen, tällä kertaa siistiäkseen suurimman osan mahdollisista teknisistä ongelmista pois. Kielen moottorin uusin versio sai nimen ”SpiderMonkey”, nimi joka tähänkin päivään asti on pysynyt samana JavaScriptin moottorissa selaimella Firefox. (Peyrott 2017.)

JavaScriptin saapuessa markkinoille heräsi Netscapen kilpailijan Microsoftin halu luoda kielestä oma versio. Elokuussa vuonna 1996 tämä toteutui ja heidän uusin selain versionsa Internet Explorer 3.0 toi mukanaan kielen ”JScript”. (Peyrott 2017.)

2.2 ECMAScript

Vuoden lopulla pohdittiin, miten JavaScriptin kanssa kannattaisi jatkaa eteenpäin. Kielen standartisoiminen nähtiin lupaavana valintana, joten Netscape piti tapaamisen marraskuun 21 ja 22 päivinä

tunnetun standardointiorganisaation ECMA kanssa. Palaverin lopputulos oli luoda JavaScriptista alan standardi sekä luoda sille komitea. (Netscape 1996.)

Komitean ensimmäisiin tehtäviin kuului standardille nimen valitseminen. JavaScriptiksi sitä ei voitu kutsua tavaramerkki syistä, joten pitkän keskustelun jälkeen päädyttiin nimeen ECMAScript. (Peyrott 2017.)

Ensimmäinen ECMAScript standardi perustui Netscape Navigatorin versioon 4. Siitä puuttui monia tärkeitä ominaisuuksia, kuten säännölliset lausekkeet (regular expressions), JSON, poikkeukset (exceptions) sekä tärkeitä metodeja builtin objekteja varten. Tästä huolimatta uusi standardi suoriutui selaimilla huomattavasti paremmin. Versio julkaistiin kesällä 1997. (Peyrott 2017.)

1998 vuoden puolivälissä julkaistiin standardin toinen versio ECMAScript 2 (ES2). Kyseessä ei ollut suuri päivitys, sillä kieleen itseensä ei tuotu mitään muutoksia, vain korjauksia ECMA ja ISO JavaScript standardin välisiin epäjohdonmukaisuuksiin. (Peyrott 2017.)

Jälleen vuotta myöhemmin sai standardi uuden version, tällä kertaa pysyvämmän version ECMAScript 3:sen (ES3:sen). Tämä standardi toi mukanaan ison määrän muutoksia, kuten säännölliset lausekkeet (Regular expressions), do-while loop, poikkeukset (exceptions) ja try-catch ominaisuuden, lisää built-in funktioita string ja array -tyyppejä varten, muotoilu numeeriselle tuotolle, (formatting for numeric output), in ja instanceof -operaattorit sekä parannuksia virheiden käsittelyyn. Standardi sai paljon suosiota, levisi laajalti ja oli tuettu suosituimmilla selaimilla monien vuosien ajan. (ECMA-International 2019.)

Neljännen standardin ES4 työstäminen alkoi heti kolmannen julkaistua vuoden 1999 lopulla. Kehitys ei ollut kuitenkaan helppoa, koska komitea ei päässyt yhteisymmärrykseen standardin suunnasta. Työ standardia kohtaan lopetettiin hetkellisesti vuonna 2003 ja jatkettiin uudelleen 2 vuotta myöhemmin JavaScriptin uuden suosion ansiosta, mikä johtui mahdollisesti uudesta tekniikasta, nimeltä AJAX sekä funktiosta XMLHttpRequest. (Peyrott 2017.)

Suunnitelmana oli tuoda suuri määrä uusia ominaisuuksia, kuten luokat (classes), käyttöliittymät (interfaces), nimiavaruudet (namespaces), paketit (packages), valinnaiset tyyppi merkinnät (optional type annotations), valinnaiset staattiset tyyppi tarkistukset (optional static type checking), ra-

kennetyypit (structural types), tyyppimääritykset (type definitions), monimenetelmät (multimethods), parametrisoidut tyypit (parameterized types), kunnolliset häntärekursiot (proper tail calls tai PTC), iteraattorit (iterators), generaattorit (generators), itsetarkastelu (introspection), tyyppejä erottelevat poikkeusten käsittelijät (type discriminating exception handlers), jatkuvat sidokset (constant bindings), oikea lohkojen näkyvyys (proper block scoping), tuhoaminen (destructuring), suppeat funktiolausekkeet sekä lopulta parannuksia taulukon käsittelyyn (array comprehensions). (Peyrott 2017.)

Kehitykseen tuli kuitenkin lopulta suuri este, koska komitean jäsenet olivat eriä mieltä kielen polusta. Jäsenet jakautuivat kahteen vastakkaiseen ryhmään, ES4 ja ES3.1 -kannattajiin. Version 4 kannattajat koostuivat komitean enimmistöstä Adobe, Google, Mozilla ja Opera. Versioon 3.1 kuului Microsoft sekä Yahoon edustaja Doug Crockford. ES3.1 oli Crockfordin kehittämä idea, jossa ECMAScript pidettäisiin yksinkertaisena: Ei uutta syntaksia, pelkästään käytännöllisiä muutoksia. (Peyrott 2017.)

Ryhmät työstivät tämän jälkeen versioitaan yli vuoden ajan. Tuloksena oli yksi viimeistelty versio, ES3.1, ES4 jääden kesken. Seurauksena oli pitää palaveri, jossa komitea tulisi lopulta yhteisymmärrykseen standardin kulusta. Kokous nimeltä ”Harmony” pidettiin Oslossa heinäkuun 2008 lopulla. Johtopäätöksinä olivat seuraavat: 1. Keskitytään version 3.1 edistämiseen kaikkien osapuolten täydellä yhteistyöllä, 2. Tehdään yhteistyötä seuraavan version kanssa, joka tulee sisältämään versiota 4 vaatimattomampia syntaktisia laajennuksia, 3. Osa version 4 ehdotuksista ei sovellu hyvin nettiä kohden, joten ne ovat pysyvästi pois neuvottelupöydältä (packages, namespaces ja early binding), 4. Muut tavoitteet ja ideat versiosta 4 muotoillaan sopivammaksi pitääkseen yhteisymmärryksen komiteassa (classes). (Eich 2008.)

Vuosi myöhemmin, ES3.1 julkaistiin uusien muutoksien kanssa ja sai myös uuden nimen ECMAScript 5 (ES5). Versio 4 nähtiin jo olemassa olevana ECMAScriptin varianttina, minkä takia tehtiin päätös hypätä sen yli suoraan seuraavaan versioon. ES5 tuomiin ominaisuuksiin kuului lukija- ja asettajametodit (getters and setters), taulukoiden ja objektien päättäminen pilkkuun (trailing commas in array and object literals), varatut sanat ominaisuuden niminä (reserved word as property names), uusia objekti metodeja (new object methods), uusia taulukko metodeja (new array methods), uusia päiväys metodeja (new date methods), funktio bind, JSON, muuttumattomat globaalit

objektit (immutable global objects), rajoitettu tila (strict mode) sekä muita pieniä muutoksia. Seuraava päivitys 5.1 saapui 2 vuotta myöhemmin. Päivityksessä käytiin läpi standardin mahdolliset epäselvyydet, mutta ei uusia ominaisuuksia. (Peyrott 2017.)

ECMAScript 6 (ES6) julkaistiin vuonna 2015 ja toi suuren määrän uusia muutoksia. Näistä osa oli peruutetun neljännen standardin lupaamia ominaisuuksia. Kaikki aiemmin lykätyt syntaktiset muutokset käytiin läpi tällä kertaa komitean yhteisymmärryksellä. (Peyrott 2017.)

Kuudennen standardin uusiin ominaisuuksiin kuului tehokkaammat muuttujan määritykset let ja const (let and const bindings), nuoli funktiot ja this avainsana (arrow functions and lexical this), luokat (classes), objektien käsittelyn parannuksia (object literal improvements: computed keys, shorter method definitions), tekstipohjat (template strings), lupaukset (promises), generaattorit, iteraatiot, iteraattorit, for...of toistorakenne (generators, iterables, iterators and for..of), oletus argumentit funktioita ja rest operaattoria varten (default arguments for functions and the rest operator), laajennus syntaksi (spread syntax), tuhoaminen (destructuring), moduuli syntaksi (module syntax), uusia kokoelmia Set, Map, WeakSet, WeakMap (new collections: Set, Map, WeakSet, WeakMap), proxyt, symbolit (symbols), tyypitetyt taulukot (typed arrays), tuki sisään rakennettujen luokkien aliluokille (support for subclassing built-ins), taattu optimointi (guaranteed tail-call optimization), yksinkertaisempi unicode tuki(simpler unicode support) sekä lopuksi binaari ja octa desimaali muotoiset muuttujat (binary and octal literals). (Peyrott 2017.)

Standardien nimeäminen vaihdettiin uuteen tapaan, jossa jokainen versio vuodesta 2015 lähtien nimetään nykyisen vuoden mukaan, joten ECMAScript 6 vaihdettiin nimeen ECMAScript 2015. Tämän lisäksi standardien julkaisu prosessi sai uuden aikataulun, jossa uusi päivitys tulee joka vuosi.

ES2016 koostui enimmäkseen bugien korjauksista, toimituksellisista korjauksista sekä muista parannuksista. Tämän lisäksi kehitettiin monia ohjelmisto työkaluja standardin tukemiseksi kuten Ecmarkup, Ecmarkdown ja Grammarkdown. Tuki uudelle eksponentointi operaattorille ja uusi metodi array.prototype taulukolle nimeltä includes. (ECMA-International 2019.)

ES2017 toi mukanaan Async funktiot (Async functions), jaetun muistin (shared memory), atomics objektin pienien kieli- ja kirjasto parannusten kanssa, bugi korjauksia ja toimituksellisia päivityksiä. Objekti sai myös pieniä lisäyksiä staattisten metodejen muodossa: Object.values, Object.entries ja Object.getOwnPropertyDescriptors. (ECMA-International 2019.)

ES2018 toi tuen asynkroniselle iteroinnille (asynchronous iteration) AsyncIterator protokollan ja async generaattoreiden kautta. Päivitys sisälsi myös neljä uutta säännöllistä lauseke (regular expression) ominaisuutta dotAll flag, named capture groups, Unicode property escapes ja look-behind assertions. Lopuksi se toi mukanaan parametrin nimeltä “rest parameter” ja spread operaattori tuen objektin ominaisuuksia varten. (ECMA-International 2019.)

ES2019 sisältää uusia built-in funktioita kuten flat ja flatMap Array.prototype taulukossa flattening arrayta varten, Object.fromEntries mikä kääntää Object.entries palautuvan arvon uudeksi objektiksi sekä alternatiiviset nimet String.prototype.trimLeft ja -trimRight built-ineille, nimeltään trimStart ja trimEnd. Lisäksi päivityksessä nähdään muutamia pieniä päivityksiä syntaksissa ja semantiikassa. (ECMA-International 2019.)

ES2020 esittää uuden metodin matchAll; import() syntaksi joka tuo asynkronisesti moduulit dynaamisen määrittäjän kanssa; uusi alkeellinen numero BigInt jota käytetään sattumanvaraisten tarkkuuslukujen kanssa (arbitrary precions integers); uusi promise yhdistäjä Promise.allSettled joka ei aiheuta oikosulkuja; globalThis, mikä antaa vapauden kutsua this arvoa globaalisesti; export * as ns from ’module’ syntaksi moduleiden sisällä käyttöä varten, lisättiin for-in standardin mukainen toistorakenne; import.meta; kaksi uutta syntaksi ominaisuutta, jotka auttavat työskentelyä null arvojen parissa, nulllish coalescing, value selection operator ja optional chaining. (ECMA-International 2022.)

ES2021 sisältää metodin replaceAll mitä käytetään stringejä varten; Promise.any promise yhdistäjän; uuden virhe tyypin ”AggregateError”, mikä esittää useita virheitä kerralla; uudet loogisten tehtävien operaattorit (??=, &&=, ||=); WeakRef millä referoidaan objektia säilyttämättä sitä roskakokoelmasta; FinalizationRegistry; Erottimet (separators) numeerisille literaaleille (1_000); parannuksia Array.prototype.sort metodiin. (ECMA-International 2022.)

JavaScriptin pitkän historian aikana syntyi kielestä monen tyyppistä sovellusta, näitä ollen esimerkiksi ohjelmointikielet, sovelluskehykset, kirjastot ja ohjelmistokehykset. Näiden lisäksi kehitettiin palvelin puolen ohjelmia ja muita avustavia työkaluja. Seuraavassa osiossa käydään läpi näitä kyseisiä ohjelmia ja katsotaan mistä ne saivat alkunsa.

3 JAVASCRIPTIN PERIJÄT

JavaScriptin pitkän kehityksen aikana käyttäjät törmäsivät usein pisteeseen, jossa iski halu kehittää kieltä omin tahoin eteenpäin. Syinä saattoi olla kielen nykyiset rajoitukset tai oman työn helpottaminen ja nopeuttaminen.

Tässä osiossa käydään läpi ohjelmia, jotka saivat alkunsa JavaScriptista. Ohjelmille tehdään katsaus, jossa kerrotaan niistä yleisesti. Tutkittavat ohjelmat on valikoitu suosion perusteella.

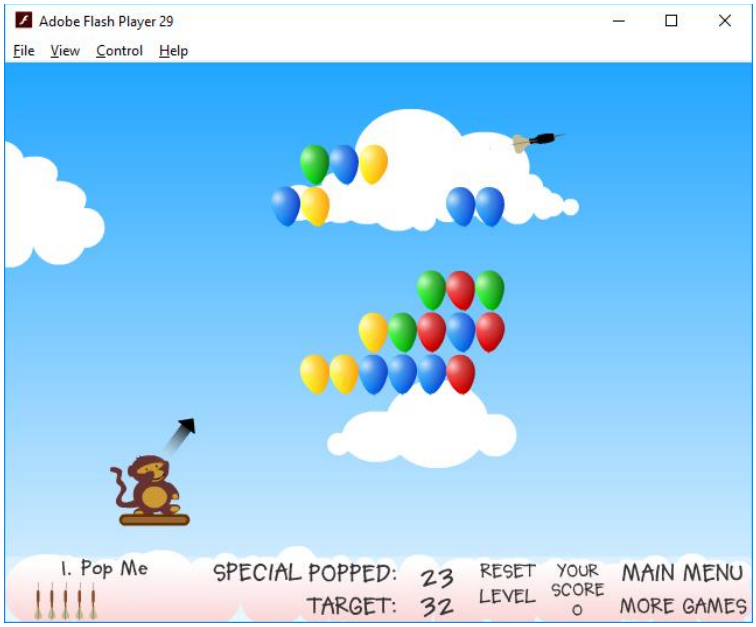
3.1 Ohjelmointikielet

Tässä osiossa tutkitaan JavaScriptista vaikutteita ottaneita kieliä. Tutkittavat kielet muistuttavat hyvin paljon JavaScriptia, mutta sisältävät omia ominaisuuksiaan, jotka tekevät kielestä omankaltaisensa.

3.1.1 ActionScript

Julkaistiin ensimmäistä kertaa vuonna 1998 ohjelmistoyrityksen Macromedian alla. Kieli sai vaikutteita Apple yrityksen HyperCard hypermediasovelluksen ohjelmistokielestä HyperTalk sekä ECMAScriptin ensimmäisestä versiosta.

Kielen alkuperäisenä tarkoituksena oli käsitellä yrityksen kehitysympäristön ”Flash Player” vektori animaatioita. Myöhempien versioiden saapuessa kieli alkoi ottaa suurempia rooleja selaimilla. Näihin tehtäviin kuului muun muassa selain-pelien (Katso kuvio 2), videoiden ja audion -käsittelyä.



KUVIO 2. Flash Playerilla pyörivä peli Bloons

ActionScript 2 otti monia peruutetun ECMAScript neljännen standardin lupaamia ominaisuuksia osaksi kieltänsä. Pian julkaisun jälkeen kehittäjän yrityksen omistajuus siirtyi Adobe Systems yritykselle.

Viimeisin versio 3 julkaistiin vuonna 2006 ja on tähänkin päivään asti ollut käytössä useissa Adoben käyttämissä ohjelmistoissa.

3.1.2 AssemblyScript

AssemblyScript tai lyhyesti AS, on melko uusi kieli, joka julkaistiin vuonna 2017. Kielen pohjana toimii Microsoftin TypeScript ja on suunniteltu WebAssembly ohjelmointikieltä varten käyttäen apuna Binaryen kääntäjää (compiler). (AssemblyScript 2021.)

Kielen arkkitehtuuri eroaa JavaScriptin virtuaalikoneesta siinä piirteessä, että se kääntää (compile) ohjelman etuajassa, kuten C-kääntäjä (AssemblyScript 2021).

Ominaisuuksista lyhyesti kuvailtuna kieli tarjoaa WebAssemblyn tyypit, matalan tason sisään rakennetut WebAssemblyn ominaisuudet, oman JavaScript kirjaston sekä pienen muistin hallinnan ja roskien keruu ajonajan (AssemblyScript 2021).

3.1.3 CoffeeScript

Rubyn, Pythonin ja Haskelin inspiroima kieli CoffeeScript julkaistiin vuoden 2009 lopulla. Sen tehtävänä on tuoda esille JavaScriptin parhaat puolet yksinkertaisin tavoin helpottaen JavaScriptin käyttöä käyttäjille. (CoffeeScript 2020.)

Kieli on saanut vuosittaisia päivityksiä, joista viimeisin saapui vuoden 2020 alkupuolella. Version 2.0 julkaistua vuonna 2017 päivitys toi kielen modernille JavaScriptin kaudelle kääntäen ES2015 ja uudempien standardejen syntakseja. (CoffeeScript 2020.)

3.1.4 TypeScript

Vuonna 2012 Microsoftin julkaisema avoimen lähdekoodin kieli, joka rakentaa JavaScriptin päälle lisäten staattisia tyyppi määritteitä. Nämä tyypit tarjoavat tavan kuvailla objektia, paremman dokumentaation sekä antavat TypeScriptin vahvistaa kirjoitetun koodin virheiden varalta. (TypeScript 2021.)

Kieli itsessään on varsin avoin ja vapaa, sillä normaali (toimiva) JavaScript koodi toimii myös TypeScriptissa. Lisäksi koodi muuntuu JS koodiksi TS tai Babel JS -kääntäjän kautta. Kirjoitetut koodit pyörivät siellä missä JavaScriptikin; selaimilla, Node.JS ajoympäristössä ja älypuhelimien sovelluksissa. (TypeScript 2021.)

3.2 Sovelluskehykset

Tässä osiossa tehdään pieni yleiskatsaus sovelluskehyksiin. Vertailu osiossa katsotaan tarkemmin niiden rakenteisiin ja ominaisuuksiin.

3.2.1 Angular

Googlen Angular tiimin kehittämä sovellusten suunnittelu kehys Angular (tai Angular 2+) julkaistiin vuonna 2016. Kehyksen pohjana toimii TypeScript ja on käytettävissä kaikilla alustoilla. Kehys on tiimin toinen, täysin uudelleen kirjoitettu versio Angular.js kehyksestä. (Angular 2021.)

Mobiilisovellusten työskentelyssä voi käyttää alustalle tarkoitettuja ohjelmistokehyksiä, kuten Cordovaa, Ionicia ja NativeScriptia. Työpöydän sovelluksiin riittää Angular itsessään. Halutessaan voi käyttää myös sillekin tarkoitettuja ohjelmistokehyksiä, kuten Electron.js. (Angular 2021.)

Sovellukset on suunniteltu ”yksi sivuisiksi”, mikä poistaa kuormikkeen, jossa käyttäjä joutuu lataamaan uusia sivuja sivujen vaihdossa (Angular 2021).

3.2.2 Angular.js

Googlen alkuperäinen Angular sovelluskehys, joka julkaistiin vuonna 2010.

Kehys on ollut vuodesta 2018 LTS (Long term support) tilassa, mikä tarkoittaa sitä että kehystä tuetaan vuoden 2021 loppuun asti. Tämän päätyttyä, loppuu kehyksen tukeminen kehittäjien osalta. Kehittäjät suosittelevat siirtymään heidän toiseen kehykseensä, Angular 2+. (Angular.js 2020.)

3.2.3 Vue.js

Evan Youn sovelluskehys Vue.js (2014) käyttää pohjana TypeScriptia ja on työkalu käyttöliittymien ja yksisivuisten sovellusten luontia varten. Kehyksen ytimen kirjasto keskittyy pelkästään näkymiin, mikä tekee siitä erittäin adoptiivisen. (Vue.js 2021.)

Kehys on tarkoitettu PC sovellusten kehittämistä varten, mutta mobiileja varten löytyy myös sujuvia ratkaisuja, kuten Ionicin Capacitor ja NativeScript ohjelmistokehykset (Vue.js 2021).

3.2.4 React.js

Facebookin ja yhteisön kehittämä ReactJS sai alkuperäisen julkaisunsa vuonna 2013. Kehys on tarkoitettu selainten parissa työskentelyyn, mutta mobiilikin onnistuu React Native (2015) sovelluskehyksen kautta. (React 2021.)

3.3 Kirjastot

Tässä osiossa tehdään pieni yleiskatsaus kirjastoihin. Kuten kehyksissä, myös vertailu osiossa katsotaan tarkemmin niiden rakenteisiin ja ominaisuuksiin.

3.3.1 Backbone.js

CoffeeScriptin kehittäjä, Jeremy Ashkenas, julkaisi alustasta riippumattoman (cross-platform) JavaScript kirjaston Backbone.js ensimmäistä kertaa vuonna 2010. Kirjastolla on muutamia vaatimuksia toimiakseen täysin voimin, näitä ollen JS, Underscore.js sekä JQuery -kirjastot. (Backbone.js 2021.)

3.3.2 JQuery

JQuery tiimin JS kirjasto (2006) luotiin tarkoituksena helpottamaan kehittäjien front-end työskentelyä. Kuten Backbone.js, JQuery pyörii myös kaikilla alustoilla. (JQuery 2021.)

3.4 Muuta

Aiempien mainittujen lisäksi on vielä käytävä läpi sovelluskehyksissä usein käytetyt ohjelmistokehykset sekä tiettyihin tehtäviin tarkoitetut ohjelmat.

3.4.1 Ohjelmistokehykset

Monet ohjelmistokehykset ovat luotu tarkoituksena laajentaa sovelluskehyksien rajoja, osa tuoden saatavuuden uusille alustoille.

Mobiilisovelluksiin keskittyviä kehyksiä ovat muun muassa Apache Cordova ja sitä pohjana käyttävä Ionic sekä React.js sovelluskehyksestä syntynyt React Native.

Tietokone kehyksien määrä ei ole suuri, sillä sovelluskehykset kattavat jo monien kehittäjien tarpeet. Kehyksistä helpostikin suosituin on OpenJs Foundationin Electron.js, jota on esimerkiksi käytetty suositussa Discord pikaviestintä sovelluksessa ja ohjelmoijille tutussa Visual Studio Code tekstieditorissa.

3.4.2 Node.js

Aiemmin mainittu Electron.js kehyksen kehittäjä OpenJs Foundation kehittää myös suosittua alustastariippumatonta ajoympäristöä, Node.js (2009) ohjelmaa. Sen tarkoituksena on rakentaa skaalautuvia verkko-sovelluksia. (Node.js 2021.)

Useita yhteyksiä voidaan suorittaa yhtä aikaa. Jokaisen yhteyden kanssa syötetään ”callback” funktio takaisin. Tehtävien loppuessa ajoympäristö menee lepotilaan, kunnes sitä taas tarvitaan uudelleen. (Node.js 2021.)

Se on saanut vaikutteita Rubyn Event Machinesta ja Pythonin Twistedistä (Node.js 2021).

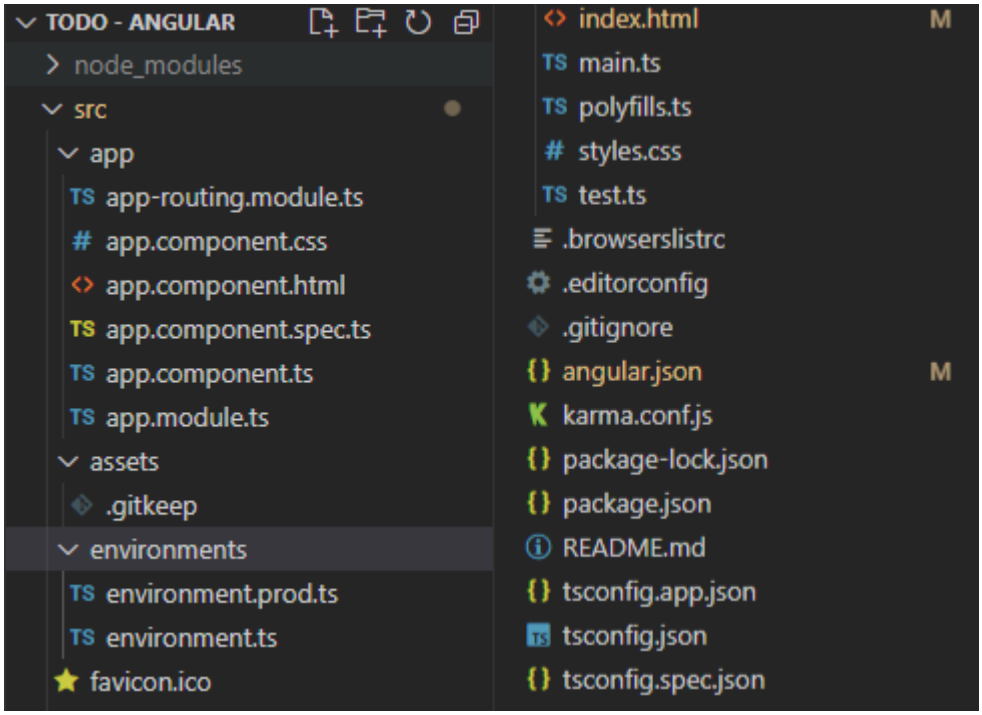
4 VERTAILU

Tässä osiossa perehdytään tarkemmin valittujen kehyksien ja kirjastojen ominaisuuksiin sekä että rakenteisiin. Tämän jälkeen tehdään ”to-do” tai suomeksi ”ostoslista”, missä vertaillaan toisiansa keskenään ja katsotaan niiden erovaisuudet sekä mahdolliset haasteet mitä työskentelimisen aikana tuli vastaan. Tavoitteeksi otin tehdä ostoslistoista mahdollisimman samannäköiset.

4.1 Käsiteltävät kehykset ja niiden rakenteet

Ensimmäinen kehys minkä valitsin oli Angular. Valinnan perusteena oli suurin osin sen keräämä suosio sekä oma käyttökokemus. Kehyksen rakenne on mielestäni helposti ymmärrettävä (Katso kuvio 3). Se koostuu seuraavista:

Angularin kokoonpano (config) tiedostot koostuvat JSON tiedostoista (angular.json, package.json, package-lock.json, tsconfig.json, tslint.json), editoijien configista, README dokumentista, githubin gitignoresta, noden asentamista npm paketeista (node_modules).



KUVIO 3. Angularin rakenne

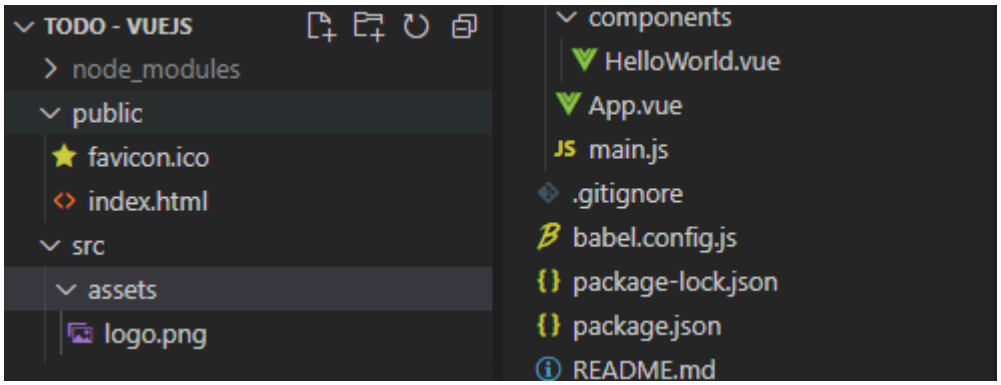
Projektin työskentely tapahtuu suurin osin src kansiossa, joka sisältää sovelluksen tiedostot. Näihin kuuluvat kansiot app, assets, ja environments sekä useita eri tiedostomuotoja, kuten favicon.ico (kuvake), index.html (mikä toimii sovelluksen oletus sivuna), main.ts (mikä on sovelluksen ”sisäänkäynti”), polyfills.ts (tuo polyfill skriptit selain tuelle), styles.sass (luettelee projektin css tiedostot) sekä test.ts, joka toimii yksikkö testauksen sisäänkäyntinä.

App kansio pitää sisällään viisi tiedostoa, jotka määrittävät eri asioita AppComponentille: app.component.ts (määrittää logiikan), app.component.html (määrittää html templaten), app.component.css (määrittää css pohjan), app.component.spec.ts (määrittää reitin yksikkö testaukselle), app.module.ts (määrittää root moduulin, kertoo kehykselle miten sovellus kasataan. Oletuksena määrittää vain AppComponentin. Uudet luodut komponentit pitää luoda myös tänne).

Assets kansio on kuvia ja muita tiedostoja varten. Tiedosto sisältää jo valmiiksi .gitkeep tiedoston. Environments kansio sisältää luonnin kokoonpano (build configurations) vaihtoehdot eri alustoja varten. Oletuksena sieltä löytyy nimetön vakio kehitys- ja tuotantoympäristö.

Toinen kehys, jonka valitsin on VueJS. Olen pitänyt tätä silmällä jo jonkin aikaa, koska monet sillä luodut sivut näyttävät visuaalisesti erittäin hyvän näköisiltä, varsinkin animaatiot mitä monilta sivuilta löytyy.

Rakenteeltaan Vue on yksinkertaisempi kuin Angular (Katso kuvio 4). Se koostuu node_moduuleista, public ja src kansioista, githubin gitignoresta, babel.config.js tiedostosta, kahdesta json tiedostosta (package-lock ja package) sekä readme tekstitiedostosta. Public kansio sisältää kuvakkeen sekä index.html tiedostoston, joka tyypillisesti toimii aloitussivuna.

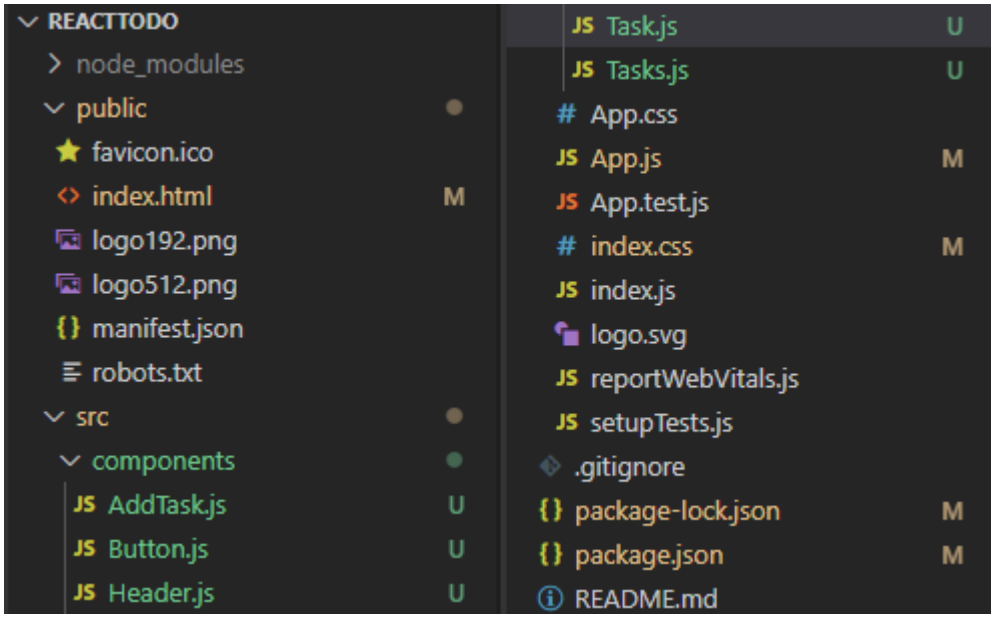


KUVIO 4. Vuen rakenne

Työskentely tapahtuu suurin osin src kansion sisällä, joka sisältää assets ja components kansiot sekä App.vue ja main.js tiedostot.

Kolmas valittu kehys on ReactJS. Omalta osalta siitä ei ole aiempaa käyttökokemusta, mutta olen siitä aiemmin kuullut Angularin opiskelun aikana sekä että se on Facebookin kehittämä. Suurin syy miksi kehys on käsiteltävänä on se, että se on kerännyt suurta suosiota sekä se että aiemmin mainittu kehittäjä on yhteyksissä siihen, mikä herättää oman mielenkiintoni kehystä kohtaan.

Projekti sisältää (Katso kuvio 5.) tavalliset kansiot (public ja src) sekä tiedostot (gitignore, package.json ja readme). Public kansio pitää sisällään favicon kuvakkeen, pari kuvaa logosta, index.html ja manifest.json tiedostot sekä mysteerisen robots tekstitiedoston.



KUVIO 5. Reactin rakenne

4.2 Käsiteltävät kirjastot ja niiden rakenteet

Toiminnallisuudeltaan kirjastot poikkeavat kehyksistä, sillä käyttäjä voi halutessaan joko käyttää CDN-linkkejä jotka tuovat kirjastot suoraan netistä tai ladata ne paikalliseen projektin kansioon.

Ensimmäinen valitsemani kirjasto oli tietenkin JQuery, sillä se on kehittäjille tutuin ja käytetyin kirjasto. CDN-linkkejä on minun tapauksessa vain yksi, JQuery.min.js. Kuten kaikkien kyseisten linkkien kanssa, se sijoitetaan <body> tagin loppuun.

Toinen valittu kirjasto on BackboneJS, joka on rakenteeltaan mielestäni paljon monimutkaisempi kuin JQuery.

4.3 Toteutus

Aloitin ensin kehyksistä, sillä oletuksena oli, että ne veisivät enemmän aikaa kuin kirjastot. Näistä ensimmäiseksi lähdin liikkeelle Angularin kanssa, koska uskoin sen sujuvan omalta osin parhaiten aiemman kokemuksen perusteella.

Angular

Työskentely tapahtui enimmäkseen komponenttejen sisällä, missä Typescript ja Html tiedostot olivat tärkeimmät. Toisin kuin seuraavien kehyksien kanssa, pyrin työskentelemään vain oletus komponentin ”app” nimisen tiedoston sisällä.

Angularilla komponentteja luodessa pitää muistaa pari asiaa; Komponentin luokan vienti (export class) typescript tiedoston sisällä (Katso kuvio 7.) ja sen vastaan ottaminen moduulin sisällä sekä sen ”julkistaminen” (Katso kuvio 6.).

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Kuvio 6. Sovelluksen moduuli tiedosto

TypeScript tiedostossa (Katso kuvio 7.) tuodaan ensin vaadittu komponentti luokka core kansiosta, jonka jälkeen sille syötetään selectori (hakee index.html tiedostosta body tagin sisältä app-rootin), templateUrl (html tiedosto) ja styleUrls (tyylitiedosto). Komponentin määritettyä alkaa sen oikea työstäminen AppComponent luokan sisällä, missä kuvasta huomaakin, olen lisännyt kaksi muuttujaa sekä kolme eriä metodia: addTask, removeTask ja toggleClass.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Todo List';
  list: any[]=[];

  addTask(item:string)
  {
    this.list.push(
      {
        id: this.list.length,
        name: item,
        active: false,
      })
    console.warn(this.list);

  }
  removeTask(id:number) {
    console.warn(id);
    this.list=this.list.filter(item=>item.id!==id);
  }

  toggleClass(item) {
    item.active = !item.active;
  }
}

```

KUVIO 7. App komponentin TypeScript tiedosto

Html tiedosto (Katso kuvio 8) puolestaan sisältää tagin, jolla on luokka container. Tämä tagi sisältää teksti syöttökentän, joka ottaa vastaan mitä tahansa merkkiä. Tehtävät sijoittuvat rivin neljätoista sisälle, missä Angularin *ngFor toimii taulokoiden ja sen vastaavijen toistajana, käy listan läpi syötetyistä tehtävistä ja tuo ne käyttäjälle näkyville.

Muuta huomioitavaa on hakasulkeiden sisällä oleva ngClass, joka on kehyksen ehdollinen luokan sitoja (conditional class binding). Tämä tekee mitä nimi viittaakin. Jos ehdot ovat kohdillaan, elementti saa määritetyn luokan, tässä tapauksessa tummemman taustan sekä punainen vasemman seinän.

```
<div class="container">
        <img alt="Angular logo" src="./assets/angularlogo.png" class="logo">
        <div class="container card shadow p-3 mb-5 bg-white rounded">
            <h2 class="text-center">TODO Angular</h2>
            <div class="form-control">
                <h5 class="pl-2">Tehtävä</h5>
                <div class="row">
                    <div class="col-10">
                        <input type="text" placeholder="Lisää tehtävä" #task />
                    </div>
                    <div class="col-2">
                        <i class="fas fa-plus-circle fa-2x pt-2 ps-3" role="button" (click)="addTask(task.value)"></i>
                    </div>
                </div>
            </div>
            <div *ngFor="let item of list; index as i" class="pt-2">
                <div class="row card shadow rounded p-4" (click)="toggleClass(item)" [ngClass]="{'active': item.active}">
                    <h3 class="taskheading">Tehtävä {{ i + 1}}
                        <i (click)="removeTask(item.id)" class="fas fa-times float-end fa-xs"></i>
                    </h3>
                    <div class="col-12 display:inline-block">
                        <p class="pt-3">{{ item.name }}</p>
                    </div>
                </div>
            </div>
        </div>
</div>
```

KUVIO 8. App komponentin Html tiedosto

Ulkopuolisia lisäyksiä näkymien visuaaliseen parantamiseen toin tietenkin Bootstrapin sekä myös fontawesome kuvake-kokoelman. Angular sisältää oman kokoelmansa, mutta koin fontawesomen olevan kattavampi.

Työn lopputulos (Katso kuvio 9.) oli mielestäni tyydyttävä. Kuten muidenkin kehyksien kanssa, pyrin sisällyttämään lisäyksen, poiston sekä tehtyjen tehtävien merkkauksen.

KUVIO 9. Todo Angular

Rakenteeltaan sovellus on yksinkertaisempi kuin muiden, mutta tästä huolimatta Angular on muita kehyksiä huomattavasti suurempi sen sisäänrakennettujen omaisuuksien vuoksi. Kooltaan se on reilut 70% kehykset kansion koosta (sisältää tätä opinnäytetyötä varten tehdyt eri kehyksien sovellukset), mikä on yhden gigatavun kokoinen. Suuri koko vaikuttaa myös mahdollisesti ainakin tämän projektin osalta muita kehyksiä hieman pitempään palvelimen käynnistämisaikaan.

Vaikka aiempaa kokemusta kehyksestä löytyi, asioiden mieleen tuominen vei kutakuinkin yhtä paljon aikaa mitä toisten kehyksien ensikertaa oppiminen vei. Tästä huolimatta sanoisin kehyksen olevan keskivertoisen helppo oppia, vaikkakin kehyksen suuri koko saattaakin aluksi tuntua hieman musertavalta.

Vue

Seuraavan kehyksen, VueJS:in, koin olevan jonkin verran helpompi ja yksinkertaisempi. Kuten aiemmista rakenteiden kuvailusta huomaa, VueJS ei pidä juuri mitään sisällään noden asentamien tiedostojen lisäksi.

Toisin kuin Angular, Vue ei lajittele komponentteja kolmeen eri tiedostoon (html, typescript ja css), vaan pyrkii sisällyttämään kaikki yhden tiedoston sisällä, joita kutsutaan nimellä Vue.

Vue tiedostot sisältävät tyypillisesti ensin templatet, mitkä vastaavat näkymien sisällöstä. Tämän jälkeen jonossa ovat skriptit ja lopuksi tyylittelyt (Katso kuvio 10.). Skripteissä tuodaan (import) ensin mahdolliset komponentit ja sitten viedään (export) kyseessä oleva komponentti tarvittavineen tietoineen, jotta sitä voidaan kutsua muuallakin. Näiden lisäksi tarvittavat metodit mitä sovellus vaatii voidaan myös suorittaa täällä.

```vue
<template>
  <img alt="Vue logo" src="./assets/logo.png" class="logo">
  <div class="container">
    <div className="container card shadow p-3 mb-5 bg-white rounded">
      <Header title="TODO VueJS" />
      <AddTask @add-task="addTask" />
      <Tasks
      @toggle-reminder="toggleReminder"
      @delete-task="deleteTask"
      :tasks="tasks" />
      </div>
  </div>
</template>

<script>
import Header from './components/Header'
import Tasks from './components/Tasks'
import AddTask from './components/AddTask'

export default {
  name: 'App',
  components: {
    Header,
    Tasks,
    AddTask,
  },
  data() {
    return {
      tasks: [],
    }
  },
  methods: {
    addTask(task) {
      this.tasks = [...this.tasks, task]
    },
    deleteTask(id) {
      if(confirm('Are you sure?')) {
        this.tasks = this.tasks.filter((task) => task.id !== id)
      }
    },
    toggleReminder(id) {
      this.tasks = this.tasks.map((task) => task.id === id ? {...task, reminder: !task.reminder} : task)
    },
  },
  created() {
    this.tasks = [
    ]
  }
}
</script>

<style>
@import url('https://fonts.googleapis.com/css2?family=Poppins:wght@300;400&display=swap');
* {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}
body {
  font-family: 'Poppins', sans-serif;
}
.container {
  max-width: 500px;
  margin: 30px auto;
  overflow: auto;
```

KUVIO 10. App.vue tuo komponentit yhteen

Tehtävälistan yksittäiset tehtävät (Katso kuvio 11.) päädyin muotoilemaan tiedostossa task ja kokonaisuudessaan tiedostossa tasks. Komponentteja luodessa huomaamani paras ominaisuus oli se, että niissä tehdyt tyylittelyt rajoittuvat niiden sisälle, mikä helpottaa omalta osalta niiden muotoilua. Näihin liittyen Angularista tuttu luokan sitoja tekee paluun, mutta kulkee nimellä :class.

```vue
<template>
    <div @dblclick="$emit('toggle-reminder', task.id)" :class="[task.reminder ? 'reminder active' : '', 'task card shadow rounded']">
      <h4>Tehtävä
        <i @click="$emit('delete-task', task.id)" class="fas fa-times"></i>
      </h4>
      <p>
        {{task.text}}
      </p>
    </div>
</template>

<script>
export default {
    name: 'Task',
    props: {
        task: Object,
    },
}
</script>

<style scope>
.task {
  background: #f4f4f4;
  margin: 5px;
  padding: 10px 20px;
  cursor: pointer;
}
```

KUVIO 11. Task komponentti antaa toiminnallisuuden jokaiselle tehtävälle

Tehtävät sijoittuvat tagin sisälle, jossa niille annetaan :key attribuuttiin identiteettinsä, jonka se saa task.id:stä sekä käydään lista läpi sen sisällä olevista tehtävistä v-for direktiivillä. Identiteetin ja muut tiedot se saa tehtävää luodessa AddTask komponentin (Katso kuvio 12.) sisällä.

```vue
<template>
  <form @submit="onSubmit" class="add-form">
    <div class="form-control">
      <h5 class="pl-2">TODO Tehtävä</h5>
      <div class="row">
          <div class="col-10">
            <input type="text" v-model="text" name="text" placeholder="Lisää tehtävä" />
          </div>
          <div class="col-2">
            <i class="fas fa-plus-circle fa-2x pt-2"></i>
          </div>
      </div>
    </div>
  </form>
</template>

export default {
    name: 'AddTask',
    data() {
        return {
            text: '',
            reminder: false
        }
    },
    methods: {
        onSubmit(e) {
            e.preventDefault()

            if(!this.text) {
                alert('Lisää tehtävä')
                return
            }

            const newTask = {
                id: Math.floor(Math.random() * 100000),
                text: this.text,
                reminder: this.reminder
            }

            this.$emit('add-task', newTask)
            this.text = ''
        }
    }
}
```

KUVIO 12. AddTask komponentti

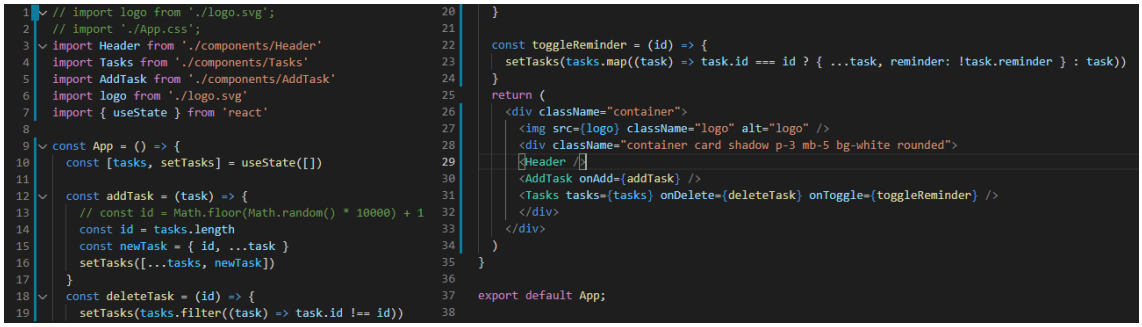
Lopputulos (Katso kuvio 13.) täytti suurin osin odotukseni.

KUVIO 13. Todo VueJS

React

Kolmas ja viimeinen kehys, ReactJS, oli mielestäni hieman hankalampi ja monimutkaisempi kuin Vue, mutta oli helpompi opetella kuin Angular. Kooltaan kehys kulkee kahden mainitun kehyksen välillä, mutta kuten Vue, on sekin silti huomattavasti kevyempi kuin Angular. Työskentely tapahtui taas yhden tyyppisten komponentti tiedostojen sisällä. Näiden tiedostojen tyyppi on JavaScript. Projekti sisältää myös yhden laajennuksen nimeltä React Redux, joka auttaa käyttöliittymän rakentamisessa.

Kuten muidenkin kehyksien kanssa, React myös sisältää app komponentin (Katso kuvio 14.), missä kutsutaan muita luotuja komponentteja. Tämän projektin tapauksessa päädyin sisällyttämään komponenttien sisällöt nuoli funktioiden sisällä, jonka ”rafce” komento (Visual Studio laajennus) kätevästi luo valmiiksi



```
// import logo from './logo.svg';
// import './App.css';
import Header from './components/Header'
import Tasks from './components/Tasks'
import AddTask from './components/AddTask'
import logo from './logo.svg'
import { useState } from 'react'

const App = () => {
  const [tasks, setTasks] = useState([])

  const addTask = (task) => {
    // const id = Math.floor(Math.random() * 10000) + 1
    const id = tasks.length
    const newTask = { id, ...task }
    setTasks([...tasks, newTask])
  }
  const deleteTask = (id) => {
    setTasks(tasks.filter((task) => task.id !== id))
  }

  const toggleReminder = (id) => {
    setTasks(tasks.map((task) => task.id === id ? { ...task, reminder: !task.reminder } : task))
  }
  return (
    <div className="container">
      <img src={logo} className="logo" alt="logo" />
      <div className="container card shadow p-3 mb-5 bg-white rounded">
      <Header />
      <AddTask onAdd={addTask} />
      <Tasks tasks={tasks} onDelete={deleteTask} onToggle={toggleReminder} />
      </div>
    </div>
  )
}

export default App;
```

KUVIO 14. Reactin App komponentti

Luodut metodit ovat jälleen tutut addTask, deleteTask sekä toggleReminder, joille on myös annettu const deklaraatiot. Tehtävät saavat identiteettinsä nykyisen listan mukaan. ToggleReminderissa on käytössä jälleen luokan sitoja.

```
import { useState } from "react"
import { FaPlusCircle } from 'react-icons/fa'

const AddTask = ({ onAdd }) => {
    const [text, setText] = useState('')
    const [reminder, setReminder] = useState('false')
    const onSubmit = (e) => {
        e.preventDefault()
        if(!text) {
            alert('Lisää tehtävä')
            return
        }

        onAdd({text, reminder})

        setText('')
        setReminder(false)
    }

    <form className="add-form" onSubmit={onSubmit}>
        <div className="form-control">
            <h5 className="pl-2">Tehtävä</h5>
            <div className="row">
                <div className="col-10">
                    <input type="text" placeholder="Lisää tehtävä"
                    value={text} onChange={(e) => setText(e.target.value)} />
                </div>
                <div className="col-2">
                    <FaPlusCircle className="centerfy" size={30} type="submit" value="Tallenna"
                    style={{color: '#c1c1c1', cursor: 'pointer' }}></FaPlusCircle>
                </div>
            </div>
        </div>
    </form>
  )
}

export default AddTask
```

KUVIO 15. Tehtävät lisätään AddTask komponentissa

Elementtien tyylittelyistä huomaa, että ne poikkeavat muista kehyksistä hieman. Class on nimetty uudelleen nimeksi className. Syynä voi olla Reactin käyttämässä JSX laajennuksessa, joka muuttaa kielen syntaksia erin tavoin.

Tyylittelyihin liittyen Bootstrap kirjaston sisällyttäminen ei ole helpoin tehtävä, sillä tyylittelyt eivät aina välttämättä lataudu. Tämä johtuu siitä miten kehyksen DOM toimii, joka päivittää vain sivun tietyt osat, jossa muutokset tapahtuvat.

Kehyksen kirjastossa löytyy myös oma arsenaali ikoneita. Angularista poiketen niiden pitää olla aaltosulkeiden sisällä (Katso kuvio 16.).

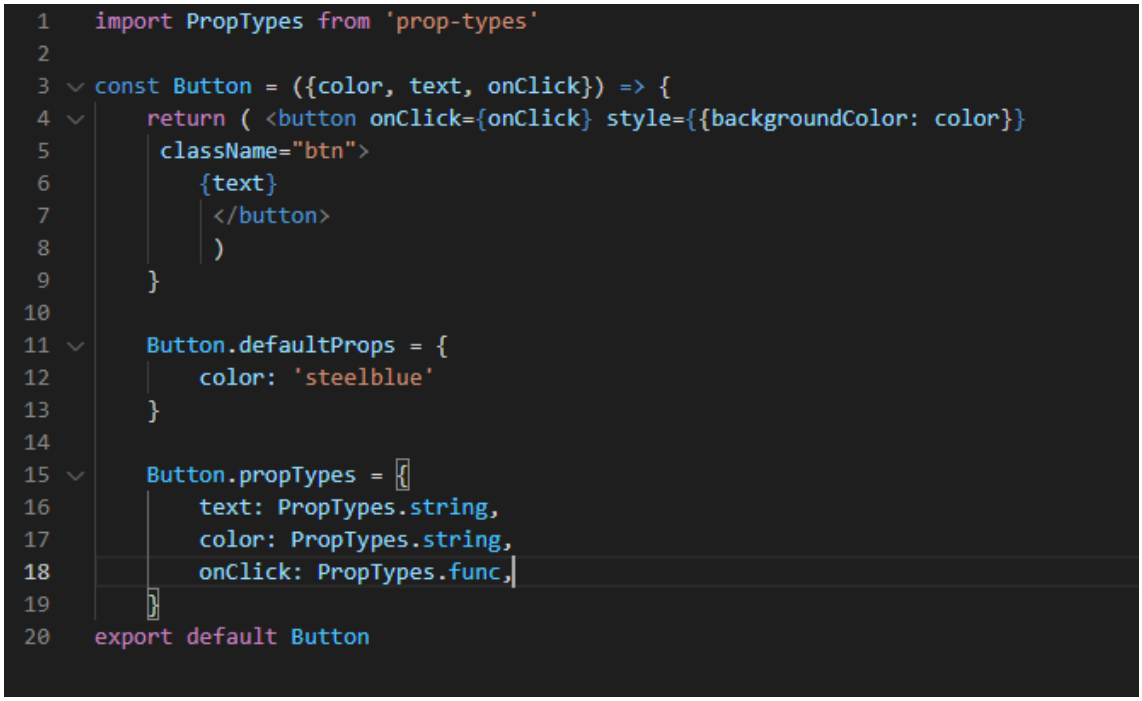
```
import { FaTimes } from 'react-icons/fa'

const Task = ({ task, onDelete, onToggle }) => {
    return (
        <div className={`task ${task.reminder ? 'reminder' : ''} card shadow rounded`} onDoubleClick={() => onToggle(task.id)}>
            <h3>Tehtävä {task.id + 1} <FaTimes style={{color: 'red', cursor: 'pointer' }} onClick={() => onDelete(task.id)}/></h3>
            <p>{task.text}</p>
        </div>
    )
}

export default Task
```

KUVIO 16. Tehtävän komponentti, jossa on käytössä kehyksen ikoni

Komponentteihin on mahdollista syöttää eri määrityksiä sen sisältämään ”propTypes” (Katso kuvio 17.) ominaisuuteen. Näitä määrityksiä voi muun muassa olla mitä tekstityyppejä tai funktioita se ottaa vastaan. Projektin tapauksessa tehtyjä määrityksiä löytyy ylätunnisteessa sekä napeissa.



```
import PropTypes from 'prop-types'

const Button = ({color, text, onClick}) => {
    return ( <button onClick={onClick} style={{backgroundColor: color}}
     className="btn">
        {text}
        </button>
        )
    }

    Button.defaultProps = {
        color: 'steelblue'
    }

    Button.propTypes = {
        text: PropTypes.string,
        color: PropTypes.string,
        onClick: PropTypes.func,
    }
export default Button
```

KUVIO 17. Esimerkki propTypes ominaisuuden käytöstä

Lopputulos (Katso kuvio 18.) oli jälleen tyydyttävä. Kaikki ominaisuudet toimivat kuin pitääkin.

KUVIO 18. Todo ReactJS

Kirjastot

Seuraavaksi lyhyesti miten kirjastojen tapauksessa meni.

JQuery

Sovelluksen toteutustapa poikkesi suuresti kirjastojen tai enemmänkin ”kirjaston” osalta kehyksiin verrattuna. Ensimmäisen kirjaston JQueryn, kanssa työskentely oli helppoa sen yksinkertaisen rakenteensa vuoksi. Melkeinpä kaikki koodi sijoittui yhden Html tiedoston (Katso kuvio 19.) sisälle tyylittelyt jääden omaan tyyli tiedostoonsa.

```
<div class="container">
    <img src="assets/jqueryicon.png" class="logo">
    <div class="container card shadow p-3 mb-5 bg-white rounded">
        <h1>TODO JQuery</h1>
        <div class="form-control">
            <h5 class="pl-2">Tehtävä</h5>
            <div class="row">
                <div class="col-10">
                    <input type="text" id="input" placeholder="Lisää tehtävä">
                </div>
                <div class="col-2">
                    <i class="fas fa-plus-circle fa-2x pt-2"></i>
                </div>
            </div>
        </div>
        <ul></ul>
    </div>
</div>
```

KUVIO 19. Containerin rakenne

Html tiedosto pitää sisällään bootstrapin tagin, jolla on container luokka, joka pitää sisällään käyttäjälle näkyvät tehtävät. Skripti tagi sijoitetaan ennen <body> tagin loppua ja sisältää tehtävien lisäyksen. Tehtävien teossa käytetään muutamaa JQuery funktiota, kuten change(), append() ja toggleClass().

Sovelluksen lopputulos on kutakuinkin sama kuin muiden, mutta poikkeaa hieman toiminnallisuudeltaan, sillä tehtäviä poistatteassa ne eivät katoa kokonaan, vaan häivyttyvät pois käyttäjän ruudulta luoden illuusion poistosta.

```
32      <script type="text/javascript">
33          $(document).ready(function() {
34
35              $('#input').change(function() {
36                  var input = $(this).val();
37                  $('ul').append(
38                      // Card Starts
39                      '<div class="task card shadow rounded" id="task">'
40                      +
41                          // First Row starts
42                          '<div class="row">'
43                      +
44                              '<div class="col-10">'
45                      +
46                                  '<h4>' + 'Tehtävä' + '</h4>'
47                      +
48                              '</div>'
49                      +
50                              '<div class="col-2">'
51                      +
52                                  '<i class="fas fa-times fa-lg"></i>'
53                      +
54                              '</div>'
55                      +
56                          '</div>'
57                          // First Row Ends
```

KUVIO 20. Skripti 1

```
                    +
                        // Second Row Starts
                        '<div class="row">'
                    +
                            '<div class="col-12">' + input + '</div>'
                    +
                        '</div>'
                        // Second Row Ends
                    +
                    '</div>'
                    // Card Ends
                                                );
                $(this).val('');
            });

            $('ul').on('click', '.fa-times', function(){
                $(this).parent().parent().parent('div').fadeOut(200);
            });
            $('ul').on('click', '.fa-check', function(){
                $(this).parent('div').toggleClass('checked');
            });
        });
    </script>
```

KUVIO 21. Skripti 2

KUVIO 22. Todo JQuery

Koin toteutuksen olevan helpompi kuin kehyksien tapauksessa, mutta huomasin myös kuinka rajoitettua työskentely oli yhden tiedoston sisällä sekä että kuinka käteviä komponentit sovellusten toteuttamisessa oikeasti ovat.

BackboneJS

Kaikista käsiteltyistä alustoista koin tämän olevan suurin haaste. Backbone poikkeaa JQuerysta monin tavoin. Se on MVC (Malli-Näkymä-Kontrolleri tai Model-View-Controller) pohjainen alusta, vaatii muita kirjastoja toimiakseen, sisältämä kokoelma ei ole yhtä laaja sekä sen suosio on hiipunut huomattavasti. Viimeinen näistä oli yksi syistä ongelmiini, sillä nykyinen versio poikkeaa monen vuoden takaisista yhteisön tarjoamista ohjeistuksista, mikä hankaloitti oppimista. Toinen oli MVC-pohjaisten alustojen kokemuksen puute, josta tämä jo valmiiksi poikkesikin jossain osin.

Rakenne koostuu malleista, näkymistä, kokoelmista (collection) ja routereista. Näistä malli toimii kuten muutkin tutut MVC-mallit. Tehtävänä on käsitellä sisäisten taulukkojen attribuutteja ja muuttaa dataa sen muuttuessa.

```
var Todo = Backbone.Model.extend({

defaults: function() {
      return {
        title: "empty todo...",
        order: Todos.nextOrder(),
        done: false
      };
    },

      toggle: function() {
      this.save({done: !this.get("done")});
    }

  });
```

KUVIO 23. Esimerkki Github käyttäjän Jeromegn luomasta TODO sovelluksen mallista

Näkymä (View) ei täsmää kuitenkaan tuttua MVC-nimen pitäjää, vaan sen rooli kulkee lähempänä sen vastaavaa kontrolleria. Tehtävikseen se omaksuu muutosten tarkkailun, käyttöliittymän renderöinnin, käyttäjien interaktion hoitamisen sekä kerätyn tiedon syöttöjen toimittamisen malleihin. Syynä nimeämiseen on luultavasti se, että se säilyttää myös osan näkymille kuuluvista tehtävistä, kuten sivujen renderöinnin ja käyttöliittymän kanssa työskentelyn.

```
var TodoView = Backbone.View.extend({

    tagName:  "li",

    template: _.template($('#item-template').html()),

    events: {
      "click .toggle"   : "toggleDone",
      "dblclick .view"  : "edit",
      "click a.destroy" : "clear",
      "keypress .edit"  : "updateOnEnter",
      "blur .edit"      : "close"
    },

    initialize: function() {
      this.listenTo(this.model, 'change', this.render);
      this.listenTo(this.model, 'destroy', this.remove);
    },

    render: function() {
      this.$el.html(this.template(this.model.toJSON()));
      this.$el.toggleClass('done', this.model.get('done'));
      this.input = this.$('.edit');
      return this;
    },

    toggleDone: function() {
      this.model.toggle();
    },

    edit: function() {
      this.$el.addClass("editing");
      this.input.focus();
    },

    close: function() {
      var value = this.input.val();
      if (!value) {
        this.clear();
      } else {
        this.model.save({title: value});
        this.$el.removeClass("editing");
      }
    },

    updateOnEnter: function(e) {
      if (e.keyCode == 13) this.close();
    },

    clear: function() {
      this.model.destroy();
    }

});
```

KUVIO 24. Esimerkki Github käyttäjän Jeromegn luomasta TODO sovelluksen näkymästä

Kokoelmat hoitavat malleja, jotka ovat yhteyksissä toisiinsa ja avustavat lataamisessa sekä tallentavat uusia malleja palvelimelle.

```
var TodoList = Backbone.Collection.extend({

  model: Todo,

  localStorage: new Backbone.LocalStorage("todos-backbone"),

  done: function() {
    return this.where({done: true});
  },

  remaining: function() {
    return this.where({done: false});
  },

  nextOrder: function() {
    if (!this.length) return 1;
    return this.last().get('order') + 1;
  },

  comparator: 'order'

});

var Todos = new TodoList;
```

KUVIO 25. Esimerkki Github käyttäjän Jeromegn luomasta TODO sovelluksen kokoelmasta

Työn lopputulos oli sovellus, mitä en valitettavasti saanut toteutettua.

5 POHDINTA

Silmäiltyäni JavaScriptin historiaan ja siitä syntyneisiin eri ohjelmiin sekä tutkittua kehyksien ja kirjastojen erot, siirrytään viimeiseen osioon missä käydään läpi matkan varrella kohdatut haasteet ja mielenpäällä olevat asiat sekä mikä oli edellisen vertailun lopputulos.

Ensimmäinen huomattava kehyksien ero minkä totesin jo heti alkuun, oli projektien koot, Angular ollen ylivoimaisesti suurin. Tämä voi olla hyvä tai huono asia kehittäjille, sillä suuri määrä tuo enemmän ominaisuuksia, mutta sen opetteleminen vie myös enemmän aikaa. Kooltaan Vue ja React ovat puolestaan hyvin saman kokoisia ja ovat helposti opeteltavissa. Angular eroaa muista myös siinä määrin, että sen rakenne tuntuu paljon rajoittavaisemmalta.

Komponentit käyttäytyivät suurimmin osin samalla lailla, eroten joissain määrin. Esimerkkeinä ollen miten jotkut funktiot toimivat, miten ne yhdistyivät toisiinsa, tyylittelyjen käyttäytymiset sekä miten html tagien käsittely hoidettiin. Angularin komponenteissa käytetty Typescript voi olla käytännöllisempi kehittäjille kuin muiden käyttämä JavaScript ja myös toisin päin. Samoin myös Reactin käyttämä JSX (JavaScript Syntax Extension), joka korvaa tyypilliset Html tiedostot. Näistä koin JavaScriptin ja Html:n olevan helpoimmat, mutta tämä johtunee niillä kerätystä aiemmasta kokemuksesta.

Kehyksien opetteleminen ei ollut kovin haastavaa, sillä jo yhden opeteltua, muutkin alkoivat sujua näppärästi. Helpoin opetella oli mielestäni Vue sen yksinkertaisuuden vuoksi ja hankalin Angular. Syynä tähän voi tosin olla että käsitellyistä kehyksistä Angular oli ensimmäinen sekä myös johdantoni komponentteihin ja kehyksien kanssa työskentelyyn.

Kirjastojen osalta en oppinut paljoa uutta, sillä toinen kehyksistä, JQuery oli tuttu jo entuudestaan ja BackboneJS ei tuottanut paljoa tulosta. Tästä huolimatta opin silti ymmärtämään miten senkin kuuluisia toimia. Kirjastoista helppo käyttöisin oli JQuery sen yksinkertaisen projektiin sisällyttämisen ja helppo käyttöisten funktioiden ansiosta.

Muutamia asioita mitä tekisin erillä lailla ovat kirjasto osio, mihin olisin saanut hakea muita vertailtavaksi Backbonen sijaan sekä kehyksien osalta olisin voinut mielestäni tehdä joko hieman monimutkaisemman sovelluksen, joka toisi esille enemmän kehyksien ominaisuuksia tai 2 erilaista sovellusta, jotka pyrkivisivät tekemään eri asioita.

Opinnäytetyöstä yleisesti sanoen aikataulu pysyi suurimmin osin kohdallaan, alkoi huhtikuun lopulla ja määränpäänä syyskuun loppu. Työskentely lähestulkoon pysähtyi kesäkuun ajalle sekä hidastui jossain määrin elokuusta alkaen, näihin syynä ollen muut opiskelut olevan päällekkäin. Työtä tehdessä opin paljon JavaScriptin taustasta sekä miten kehykset koostuvat ja miten niitä kuuluisi käyttää. Tekstiä kirjoittaessa huomasin myös oppineeni käyttämään paremmin Word kirjoitus työkalua, mitä en osannut odottaa ennen työn aloittamista.

LÄHTEET

Peyrott, Sebastian 2017. A Brief History of JavaScript. Auth0 16.1.2017. Hakupäivä 3.5.2021. https://auth0.com/blog/a-brief-history-of-javascript/

ECMA-International 2019. ECMA-262, 10th edition. ECMA-International 10.6.2019. Hakupäivä 4.5.2021 https://262.ecma-international.org/10.0/

O'Reilly, Reid & Valentine, 2013. JavaScript Programmer's Reference. Apress. https://learning.oreilly.com/library/view/javascript-programmers-reference/9781430246299/?ar/?orpq&email=%5Eu

Netscape 1996. Industry leaders to advance standardization of netscape's at standards body meeting. Netscape Communications Corporation 15.11.1996. Hakupäivä 5.5.2021. https://web.archive.org/web/19981203070212/http:/cgi.netscape.com/newsref/pr/newsrelease289.html

Eich, Brendan 2008. ECMAScript Harmony. Mozilla 13.8.2008. Hakupäivä 12.5.2021 https://mail.mozilla.org/pipermail/es-discuss/2008-August/003400.html

ECMA-International 2009. Ecma International approves major revision of ECMAScript. ECMA-International 15.12.2009. Hakupäivä 12.5.2021 https://www.ecma-international.org/news/ecma-international-approves-major-revision-of-ecmascript/

ECMA-International 2021. ECMA 2022 Language Specification. Tc39 12.5.2021. Hakupäivä 18.5.2021 https://tc39.es/ecma262/

AssemblyScript 2021. The AssemblyScriptBook. AssemblyScript 2021. Hakupäivä 26.5.2021 https://www.assemblyscript.org/introduction.html

TypeScript 2021. What is TypeScript? TypeScript 2021. Hakupäivä 28.5.2021 https://www.typescriptlang.org/

Angular 2021. Features. Angular 2021. Hakupäivä 28.5.2021
https://angular.io/features

Angularjs 2020. Version support status. Angular 2020. Hakupäivä 28.5.2021
https://docs.angularjs.org/misc/version-support-status

Angularjs 2020. Version Support Status. Angularjs 2020. Hakupäivä 28.5.2021
https://angularjs.org/

Vue.js 2021. Guide. Vue.js 2021. Hakupäivä 6.6.2021
https://vuejs.org/v2/guide/

Backbone.js 2021. Getting Started. Backbone.js 2021. Hakupäivä 9.6.2021
https://backbonejs.org/#

JQuery 2021. JQuery API. JQuery 2021. Hakupäivä 13.6.2021
https://api.jquery.com/

React 2021. Getting Started. React 2021. Hakupäivä 13.6.2021
https://reactjs.org/docs/getting-started.html

Node.js 2021. About. Node.js 2021. Hakupäivä 27.6.2021
https://nodejs.org/en/about/