Joose Sippola

JavaScriptin historia ja kehittyminen

Tietotekniikan kandidaatintutkielma

14. joulukuuta 2022

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Joose Sippola

Yhteystiedot: joose.j.sippola@student.jyu.fi

Ohjaaja: Jonne Itkonen

Työn nimi: JavaScriptin historia ja kehittyminen

Title in English: History and development of JavaScript

Työ: Kandidaatintutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 20+0

Tiivistelmä: Tämä kandidaatintutkielma tutustuu ja pohtii tapahtumia sekä syitä, miten JavaScript on kehittynyt yhdeksi suurimmista ohjelmointikielistä. Tavoitteena on syventyä niihin lähtökohtiin mistä JavaScript on saanut alkunsa, ja saada kuva niistä JavaScriptin kehitysaskeleista, jotka ovat johtaneet JavaScriptin suureen suosioon.

Avainsanat: JavaScript, ECMAScript, Web-kehitys, kandidaatintutkielmat

Abstract:

This bachelor's thesis explores and reflects on the events and reasons for how JavaScript has developed into one of the largest programming languages. The goal is to get deeper into the starting points from which JavaScript got its start, and to get a picture of the development steps in JavaScript which have led to the great popularity of JavaScript.

Keywords: JavaScript, ECMAScript, Web-development, Bachelor's Theses

Termiluettelo

Dynaaminen	Verkkosivujen kontekstissa dynaamisuudella tyypillisesti tarkoitetaan sisältöä tai toimintaa, joka muuttuu.
ECMAScript	Standardi, jolla pyritään pitämään eri selaimilla yhtenäinen tuki JavaScriptille.
ES*x* ja ECMAScript*x*	ECMAScriptin versio. ES*x*-nimeämisestä luovuttiin ES6:n jälkeen, jolloin siirryttiin käyttämään version nimenä version julkaisuvuotta, esimerkiksi ECMAScript2020.
Edusta	Engl. *Front-end*. Järjestelmän käyttäjälle näkyvä osa, jonka kanssa käyttäjä on vuorovaikutuksessa.
Kehys	Engl. *Framework*. Kokoelma tapoja tai sääntöjä kirjoittaa ohjelmakoodia.
Kirjasto	Engl. *Library*. Kokoelma valmiiksi tehtyjä ohjelmia, joita voi hyödyntää omissa ohjelmissa.
Komponentti	Osa jonkin järjestelmän rakennetta.
Plugin	Ohjelmistokomponentti, jonka voi liittää valmiiseen ohjelmaan.
Skriptaus	Skriptaus on järjestelmän muuttamista tai muokkaamista. Skriptauskieliä ei tyypillisesti erikseen käännetä, vaan komennot käännetään konekäskyiksi ajon aikana.
Staattinen	Verkkosivujen kontekstissa staattisuudella tarkoitetaan muuttumattomuutta.
Tausta	Engl. *Back-end*. Järjestelmän taustalla tehty ohjelmalogiikkaa ja tiedon käsittely.

Kuviot

Taulukot

Sisällys

1 Johdanto

Kandidaattitutkielman tutkimusaiheena on JavaScript-ohjelmointikielen historia ja kehittyminen. JavaScript julkaistiin vuonna 1995, mutta tässä ajassa se on kokenut paljon päivityksiä. On mielenkiintoista verrata mitä tarkoitusta varten JavaScript on alkujaan tehty, ja mitä kaikkea sillä nykypäivänä tehdään. Kandidaattitutkielman tarkoitus on pohtia sitä, miten JavaScript on kummallisesta alustaan huolimatta yksi maailman käytetyimmistä ohjelmointikielistä[1], jolla kehitetään niin pieniä kuin valtaviakin projekteja, vaikka sitä ei siihen ole alun perin suunniteltu.

Ohjelmistokehityksen alalla tulee ja menee koko ajan uusia trendejä, ja monesti työkaluissa on paljon valinnanvaraa. Työpöytäsovelluksille, kännykkäsovelluksille ja sulautetuille järjestelmille on lukuisia eri ohjelmointikieliä, joilla kehitystä tehdään. Joukosta erottuu kuitenkin yksi osasto, verkkosovellukset. Verkkosovelluksissa, jotka ajetaan selaimessa, käytetään lähes poikkeuksetta JavaScriptiä, ja on käytetty koko *web 2.0*[2]-aikakauden ajan.

Aluksi perehdytään JavaScriptin lähtökohtiin, sen varhaiseen käyttöön ja syntymiseen. Sen jälkeen pohditaan mikä sen on pitänyt erittäin suosittuna kielenä tähän päivään asti, huomioiden sen alkutarkoitus. Lopussa esitellään JavaScriptin nykykäyttöä ja pohditaan JavaScriptin suosion syitä.

1. Vuonna 2022 Stackoverflow -sivuston vuosittaisessa kyselyssä oli kymmenes vuosi, jossa JavaScript oli suosituin ohjelmointikieli. Myös JavaScriptistä jalostettu TypeScript oli kymmenen suosituimman joukossa (*Stack overflow developer surveys* 2011-2022). Myös muissa listoissa, kuten esimerkiksi PYPL (Popularity of Programming Language) indeksi tai TIOBE (The Importance Of Being Earnest) indeksi, joissa tutkitaan suosituimpia ohjelmointikieliä, JavaScript löytyy lähes poikkeuksetta kymmenen parhaan joukosta, mutta koska mittareita tutkimuksille on monenlaisia, ei yksiselitteistä vastausta suosituimmasta kielestä ole (*PYPL popularity of Programming Language index* 2022; *Tiobe index* 2022).

2. O'reilley määrittelee web 2.0 olevan verkko alustana, joka kattaa kaikki laitteet. Web 2.0 sovellukset ovat ne, jotka hyödyntävät alustan luontaisia etuja, kuten ohjelmistojen toimittaminen jatkuvasti päivittyvänä palveluna, joka paranee mitä enemmän ihmiset käytttävät sitä, saamalla ja käsittelemällä tietoa monesta lähteestä, mukaan lukien yksittäisiltä käyttäjiltä, samalla välittämällä omaa tietoaan ja palveluitaan muodossa, joka mahdollistaa muiden käsittelevän sitä. Tämä luo verkoston niin sanotun osallistumiisarkkitehtuurin avulla, ja tarjoaa *web 1.0*:llaa kattavamman käyttökokemuksen. (O'reilly 2005).

2 Alkuperäinen JavaScript

A. Wirfs-Brockin ja B. Eichin artikkelissa JavaScriptin ensimmäisestä kahdestakymmenestä vuodesta kerrotaan, miten JavaScriptin ensimmäisen version takana oli vain yksi kehittäjä, Brendan Eich. Alun perin Mocha-nimellä tunnettu ensimmäinen versio toteutettiin vain kymmenessä päivässä ja pienillä vaatimuksilla, sillä sen perimmäinen idea oli olla helppokäyttöinen skriptauskieli, jolla saada dynaamisuutta verkkosivuille. Verkkosivut olivat tähän asti olleet pelkällä merkintäkielillä tehtyjä, joten interaktiiviset toiminnot olivat todella rajoitettuja. Alun perin vain NetScape Navigator -selain tuki skriptausta, ennen kuin tähän aikaan nopeasti kasvava Microsoft kehitti oman selaimensa sekä komentokielensä, JScript. (Wirfs-Brock ja Eich 2020).

2.1 Selaimille yhtenäinen kieli, ECMAScript

Verkkosivujen määrän ja koon eksponentiaalinen kasvu, ja kahden samankaltaisen selainten komentokielten myötä huomattiin selvä tarve standardoida selainten tuki. Tämä johti ECMAScriptiin. (Dao 2020; Huberman ja Adamic 1999). ECMA eli *European Computer Manufacturers Association* on voittoa tavoittelematon organisaatio, jonka tarkoituksena on tietokonejärjestelmien standardointi (Dao 2020). Se on organisaatio, joka ylläpitää vuonna 1997 julkaistua ECMA-262 spesifikaatiota. Tämä tunnetaan paremmin nimellä ECMAScript. Se on JavaScript standardi, jolla pyritään varmistamaan verkkosivujen yhteensopivuus eri selainten välillä. ECMAScriptin alkuperäinen spesifikaatio perustuu sen hetkiseen JavaScriptiin, eli versioon 1.0 (*ECMA - MDN Web Docs glossary: definitions of web-related terms: MDN* 2022).

JavaScriptin ensimmäisen version C. Severance kuvailee olleen hyvä siihen mihin se oli tarkoitettukin, eli yksinkertaiseen skriptaukseen. Selainten sisäinen JavaScript-ohjelmointi nousi odottamattoman nopeasti selainten perusominaisuudeksi, jolloin nopean toteutuksen jättämät ongelmat ilmenivät, kun sovellukset kasvoivat koko ajan suuremmiksi, eikä JavaScriptiä oltu varsinaisesti suunniteltu mihinkään suurempaan. JavaScript oli jo juurtunut niin perusteelliseksi selainten osaksi, että kokonaan uuteen kieleen siirtyminen olisi ollut mo-

nimutkaista, vaikeaa ja riskialtista. Tässä vaiheessa tilanne oli kuitenkin yhteensopivuuden kannalta hyvä selainten kesken ECMAScript-standardin johdosta, ja ECMAScriptin avulla päivityksiä kieleen saatiin toimitettua yhtenäisesti. (Severance 2012).

2.2 Yhteisön vaikutukset

Yksi mielenkiintoinen asia JavaScriptissä on, miten sille on niin monia erilaisia lisätyökaluja. B. Frankstonin artikkelissa JavaScriptin ekosysteemistä hän kertoo, miten JavaScriptillä nykyään tehdään monimutkaisia ja vaativia projekteja, mikä on mahdollista osaksi kieleen tehtyjen päivitysten ansiosta, mutta vähintään yhtä suuressa osassa on JavaScriptin ulkoiset lisäosat, kuten kirjastot ja kehykset. Kielen luomisvaiheen ongelmia ei voi kokonaan kitkeä pois, sillä kielen jatkuvuutta tulee ylläpitää, ja vanhempien ohjelmien on toimittava myös ohjelmointikielten uudemmilla versioilla. Erilaisilla sovelluskehyksillä ja kirjastoilla jää kuitenkin vapaat kädet korjata ongelmia tai tehostaa toimenpiteitä.

Frankstron kertoo JavaScriptin menestyksen olleen todellinen yllätys. Verkkosivut ovat *web 2.0* -aikakaudella kehittyneet olemaan pikemminkin kokonaisia ohjelmia, kuin vain klassisia staattisia sivustoja. Frankstronin mukaan kieli esimerkiksi mahdollistaa monen tyylistä ohjelmointia ja todella joustavaa kehitystä. Nämä, kuten monet muutkin tekijät, ovat saaneet kehittäjät jatkokehittämään JavaScriptiä luomalla omia lisätyökaluja kielen avuksi. Frankstronin mukaan JavaScript on edelleen kehittymisvaiheessa, ja nähtäväksi jää, mikä on JavaScriptin seuraava läpimurto. (Frankston 2020). Jeff Atwoodin[1] blogissa esitelty humoristinen Principle of least power[2]:iin perustuva Atwoodin laki kertoo, että kaikki sovellukset, jotka voidaan kirjoittaa JavaScriptillä, tullaan lopulta kirjoittamaan JavaScriptillä. Tämä tuntuu koko ajan tulevan todemmaksi, vaikka alkujaan olikin vain vitsi (Atwood 2007).

1. Jeff Atwood on toinen StackOverflow-sivuston perustajista.

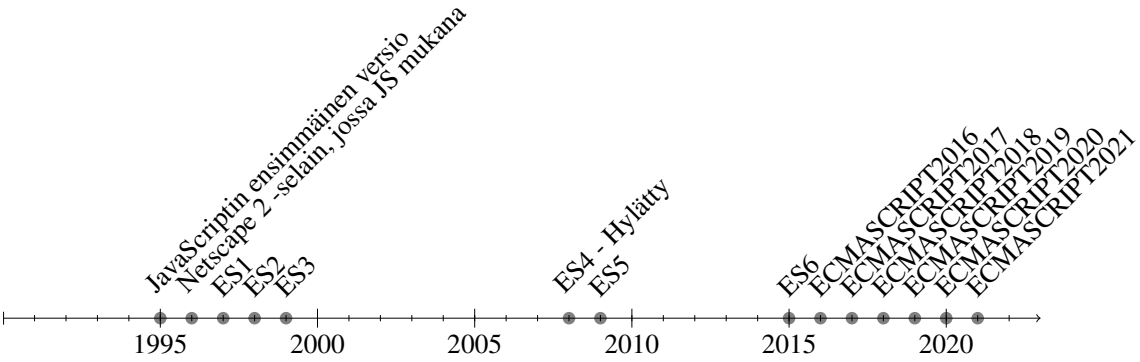
2. *Principle of least power* tai *Rule of least power* on ohjelmistojen suunnitteluperiaate, jonka mukaan tehtävän tekemiseen tulisi valita vähiten tehokas työkalu ongelman ratkaisemiseen. Tämän T. Berners-Lee perustelee dokumentissaan sillä, että mitä vähemmän tehokas työkalu, sitä enemmän sen sisältämällä datalla voidaan tehdä (Berners-Lee 1998).

3 JavaScriptin kehitys

Wirfs-Brock ja Eich kertovat artikkelissaan, miten kehittäjät olivat jo toivoneet JavaScriptille ominaisuuksia, joita muut kielet tarjosivat, sillä verkkosovellukset olivat alkaneet olemaan vaativampia *web 2.0*-aikakauden takia. *Web 2.0*-aikakauden sanotaan alkaneen noin 2005, eli ES3- ja ES5-versioiden välissä. Tällöin verkkosovellusten määrä ja koko kasvoi, ja erilaisten ominaisuuksien tarvetta paikattiin joko sovelluskohtaisilla ratkaisuilla tai ulkoisilla kirjastoilla ja kehyksillä. P. Gardner, A. Maffeis ja G. Smith vertailevat tutkielmassaan, miten siirtymä ES3:sta ES5:teen johti monien sovellusten korjauksiin. ES5 lisäsi ominaisuuksia ja optimointia, joten vaikka vanhan version mukaiset sovellukset eivät hajonneet, on uusien versioiden ominaisuuksien käyttöönotto tärkeää ylläpidon ja sovellusten tehokkuuden kannalta. (Wirfs-Brock ja Eich 2020; Gardner, Maffeis ja Smith 2012).

F. Kereki pohtii artikkelissaan JavaScriptin asioita, jotka houkuttelevat kehittäjiä käyttämään JavaScriptiä. Hyviksi puoliksi hän listaa esimerkiksi mahdollisuuden kehittää edustaa ja taustaa samalla kielellä, laaja tuki ja työkalut sekä jatkuvan kehityksen (Kereki 2015). JavaScript on laajojen yhteisöprojektien avulla kehittynyt verkkosovellusten ohjelmointikielestä mahdolliseksi käyttää myös muun muassa palvelinten, mobiilisovellusten ja sulautettujen järjestelmien ohjelmointiin. Alaluvuissa tutustumme joihinkin tärkeisiin kehityksiin, mitä JavaScriptille on tullut.

3.1 JavaScript ja ECMAScriptin päivitykset



Kuvio 1: ECMAScript-versioiden aikajana

(Wirfs-Brock ja Eich 2020; w3schools 2022).

ECMAScriptin päivityksistä suurimpia muutoksia listataan taulukossa 1 ja aikajana kuvassa 1. Wirfs-Brockin ja Eich artikkelissaan kertoo, että vuoteen 2000 mennessä Eich oli siirtynyt jo muihin projekteihin, ja ECMAScriptin ylläpitäminen oli jäänyt ECMA:n vastuulle. ECMA:lla oli ollut vaikeuksia päästä sopuun siitä, miten standardin päivittäminen toteutetaan, ja mitä siitä ylipäätään päivitetään. JavaScriptillä oli jo alettu tekemään suuremman ja suurempia sovelluksia, ja kehittäjät toivoivat laajempia ominaisuuksia, että haluttuja sovelluksia pystyttäisiin tehdä ilman, että heidän täytyisi itse tehdä sovelluskohtaisia ratkaisuja ominaisuuksille. ES4-versio jäi kokonaan julkaisematta, ja osa ominaisuuksista, jotka lopulta haluttiin lisätä, liitettiin osaksi ES5:ttä. Kymmenen vuotta ES3:a myöhemmin ES5 vihdoin julkaistiin. ES5 on muutoksien määrästä mitattuna suurin päivitys. Tämä aiheutti paljon sovelluksia, jotka eivät enää olleet yhteensopivia uusimman ECMAScriptin version kanssa.

ECMA:lla oli päivityksissä monia tavoitteita, kuten yhteensopivuuden varmistaminen, kielen nykyaikaistaminen ja päivitysten nopea aikataulu. Seitsemän vuotta ES5:den jälkeen, vuonna 2015, julkaistiin ES6. Tämän päivityksen Eich kertoo olevan perusta tänäkin päivänä jatkuvalle JavaScriptin kehitykselle. Tästä vuodesta eteenpäin ECMAScriptin päivitykset ovat olleet kooltaan pienempiä, mutta niitä tulee vuosittain. Nykyään JavaScriptissä ei puhuta varsinaisesti versioista, vaan tyypillisesti selainten JavaScript-moottorit tukee jotain ECMA:n versiota täysin, sekä mahdollisesti joitain ominaisuuksia uudemmista spesifikaatioista. (Wirfs-Brock ja Eich 2020).

Taulukko 1: ECMAScript päivitykset

ES1	Alkuperäinen julkaisu
ES2	Toimituksellisia muutoksia
ES3	Lisätty säännölliset lausekkeet (engl. regular expressions) Lisätty try ja catch -lohkot Lisätty poikkeus olio Lisätty switch -lauseke Lisätty useita metodeita merkkijonoille ja taulukoille Lisätty while -silmukka
ES4	Ei julkaistu
ES5	Lisätty "strict mode" Lisätty JSON(JavaScript Object Notation) tuki Lisättty useita iteraatio metodeita taulukoille
ES6	Lisätty let ja const avainsanat Lisätty oletusarvot parametreihin Lisätty useita metodeita taulukoille ja merkkijonoille Lisätty Promise-oliot Lisätty lambda funktiot Lisätty olioille vaihtoehtoinen Class -syntaksi
ECMAScript 2016 [1]	Lisätty potenssi operaattori (**) Lisätty Array.includes() -metodi
ECMAScript 2017	Lisätty merkkijonoille metodeita Lisätty metodeita merkkijonoille Lisätty asynkroniset funktiot

1. ES6 jälkeen ECMA vaihtoi nimeämiskäytännön olemaan ECMAScript, jonka perässä vuosiluku.

ECMAScript 2018	Lisätty silmukoille asynkroninen iterointi Lisätty Promise-oliolle finally()-metodi Lisätty uusia ominaisuuksia säännöllisiin lausekkeisiin
ECMAScript 2019	Lisätty metodeita merkkijonoille, olioille ja taulukoille Vaihdettu try-lohkon catch-osan parametrin välitys valinnaiseksi
ECMAScript 2020	Lisätty BigInt-tietotyyppi Lisätty tyhjän liitto operaattori (??) Lisätty valinnainen ketjutus operaattori (?)
ECMAScript 2021	Lisätty numeerinen erotinmerkki (_) Lisätty metodeita merkkijonoille ja Promise-oloille

(Wirfs-Brock ja Eich 2020; w3schools 2022)

3.2 JavaScriptistä johdetut kielet

JavaScriptille on lukuisia sitä laajentavia kieliä. Näille kielille on ominaista, että ne ovat suunniteltu siten, että ne käännetään(*transpile*) JavaScriptiksi. Syitä, miksi tällaisia kieliä, jotka lopulta kääntyy JavaScriptiksi, käytetään, on esimerkiksi mielekkäämpi syntaksi, yhteensopivuus muiden työkalujen kanssa tai lisäominaisuudet. Seuraavissa alaluvuissa käsitellään joitain tunnettuja JavaScriptistä johdetuista kielistä.

3.2.1 TypeScript

TypeScriptistä kertovassa kirjassaan N. Rozentals kertoo TypeScriptin olevan Microsoftin suunnittelema avoimen lähdekoodin ohjelmointikieli. TypeScript käännetään tavalliseksi JavaScriptiksi, joten käytännön hyödyt TypeScriptin käyttämisessä on kehitysvaiheessa, esimerkiksi puhtaamman, vähemmän virhealttiin koodin kirjoittamisessa. TypeScriptillä on säädettävissä, minkä ECMAScript-version mukaiseksi JavaScriptiksi TypeScriptillä kirjoitettu tiedosto käännetään. TypeScript lisää kehityksen avuksi muun muassa staattisen tyypityksen, luokkien yksityiset- sekä julkiset muuttujat ja vaihtoehtoista syntaksia joihinkin tehtäviin. Nämä helpottavat rakentamaan laajoja, kokonaisia ohjelmia JavaScriptillä (Rozentals 2015). Myös JavaScriptillä kirjoitettuja kirjastoja pystyy käyttämään suoraan, sekä niihin saa myös tyypitykset mukaan ylimääräisellä *.d*-tiedostolla.

JavaScriptin *prototype*-pohjaisen rakenteen takia JavaScript-luokkien tekeminen on ollut melko sotkuista. Tähän monet kielet, jotka käännetään JavaScriptiksi, ovat kehittäneet oman ratkaisun syntaksista. TypeScriptin olio-ohjelmoinnin syntaksi on alusta alkaen muistuttanut vahvasti C-kieliperheen jäseniä (esimerkiksi *Java, C#, C++*). JavaScriptiin lisättiin luokkien luominen TypeScriptistä sekä muista kielistä tutummalla *Class*-syntaksilla ES6:ssa (Leprohon 2017). Tämä tosin vain jäljittelee luokkia, ja taustalla JavaScript edelleen käyttää prototyyppi -rakennetta.

3.2.2 CoffeeScript

CoffeeScriptin syntaksiin on saatu inspiraatiota *Rubystä* ja *Pythonista*, sekä se toteuttaa monia ominaisuuksiaja käytänteitä näistä kielistä. Näistä kielistä lainattuja käytänteitä on esimerkiksi funktiokutsujen kirjoittaminen ilman sulkuja, ehtolauseiden määrittely lausekkeen päätteeksi ja sanalliset vertailuoperaattorit. CoffeeScript pyrkii ottamaan JavaScriptin hyvät puolet lyhyempään ja helpommin luettavaan syntaksiin. (Ashkenas ym. 2009). Myös CoffeeScriptillä pystyy käyttämään JavaScriptillä kirjoitettuja kirjastoja, sekä se käännetään lopuksi JavaScriptiksi.

3.2.3 PureScript

J. Dupal esittelee tutkielmassaan PureScriptin olevan vahvasti tyypitetty, Haskell-kielestä vaikutteita saanut puhtaasti funktionaalinen ohjelmointikieli, joka käännetään JavaScriptiksi. PureScriptin on alun perin suunnitellut Phil Freeman vuonna 2013, josta on sittemmin tullut avoimen lähdekoodin yhteisöprojekti. Myös Haskell-koodeille on työkaluja[2], joilla koodin pystyy kääntämään JavaScriptiksi, mutta Freeman ei ollut näihin tyytyväinen, vaan tahtoi kääntyneen JavaScript-koodin olevan selvempää ja lukukelpoista. Tästä PureScript projekti sai alkunsa. (Dupal 2016).

3.2.4 Dart

Dart on Googlen kehittämä web-ohjelmointikieli, joka seuraa C-kieliperheen tyyppisiä käytänteitä. A. Hassan kertoo Dart-kieltä käsittelevässä artikkelissaan tällä tähdättävän siihen, että se on tuttu valtaosalle kehittäjistä, helppo käyttää ja projektit pysyvät skaalautuvina.

2. On olemassa monia työkaluja, joilla voi kääntää jonkin ohjelmointikielen lähdekoodia muille ohjelmointikielelle. Niitä emme käsittele tässä tutkielmassa tarkemmin.

Dartin alkuperäinen idea oli lisätä Google Chrome -selaimeen Dart-moottori, jolloin Dart toimisi JavaScriptin kanssa rinnakkain omana kielenään, mutta tämä suunnitelma on päätetty olla toteuttamatta. Dart-moottoria käytetään joissain määrin Dartin ajamiseen selaimen ulkopuolella, mutta selaimille Dart-koodi käännetään JavaScriptiksi. (Hassan 2020).

3.2.5 JS++

JS++ on kieli, joka laajentaa JavaScriptiä. Kaikki JavaScript ohjelmakoodi toimii itsenään JS++ sisällä, sekä JS++ tuo lisäominaisuuksia, esimerkiksi mahdollisuuden tyypittää muuttujia. Tämän tarkoituksena on auttaa huomaamaan ongelmatilanteita kehittämisvaiheessa. (*The JS++ Type System* 2022).

3.3 Kirjastot ja kehykset

JavaScriptille on lukuisia kirjastoja ja kehyksiä kehityksen avuksi. Näiden perimmäisenä tarkoituksena on helpottaa kehitysprosessia. Kirjastot ja kehykset ovat tyypillisesti suunniteltu jotain spesifiä ongelmaa ajatellen, ja ne esimerkiksi vähentävät kirjoitettavaa ohjelmistokoodin määrää tai tekee siitä selvempää.

Johnson R. määrittää kehykset siten, että ne ovat koko järjestelmän tai sen osan uudelleenkäytettävä malli, jota edustaa joukko abstrakteja luokkia ja tapa, jolla niiden esiintymät ovat vuorovaikutuksessa. Hänen mukaansa kuitenkaan kehyksille ei varsinaisesti ole tarkkaa yleistynyttä määritelmää. Kirjastot ovat myös koodipaketteja, joiden tarkoituksena on helpottaa tai nopeuttaa kehittämistä. Erona kirjastolla ja kehyksellä on, että kirjastoja voi tyypillisesti liittää ja käyttää oman ohjelmakoodin kanssa, mutta kehykset tyypillisesti tarjoavat tavallisesta ohjelmakoodista eroavan, abstraktimman tavan kirjoittaa joko koko ohjelman tai osan ohjelmaa. (Johnson 1997).

3.3.1 Node.js

Node.js on JavaScript ajoympäristö, joka mahdollistaa JavaScript-koodin ajamisen selaimen ulkopuolella (Heller 2017). Sitä käytetään siis paljon palvelinten ohjelmointiin JavaScriptillä. Node sisältää myös oman paketinhallintajärjestelmänsä *Node Packet Manager* eli NPM. Pakettien määrästä laskettuna se on suurin paketinhallintajärjestelmä tällä hetkellä, ja 27.11.2022 se sisältää yli 1 300 000 eri koodipakettia. (*What is NPM?* 2022).

3.3.2 jQuery

JQuery on JavaScript kirjasto, jonka MZ. Ahmed kertoo artikkelissaan yksinkertaistavan HTML-dokumentin muokkaamista, kuten esimerkiksi läpikäyntiä, tapahtumienkäsittelyä ja animointia. JQuery on avoimen lähdekoodin projekti, mikä on eniten käytetty JavaScript -kirjasto kautta aikojen. Ahmed selittää JQueryn suosiota esimerkiksi sillä, että sen saa yhdistettyä mukaan muihin kehyksiin, se on resurssivaatimuksiltaan hyvin kevyt ja se mahdollistaa tehokkaampaa kehitystä tavalliseen JavaScriptiin verrattuna. JQueryyn voi myös helposti liittää lisäominaisuuksia asennettavilla *plugineilla*. (Ahmed 2014). Daon mukaan JQuery on helmikuussa 2020 ollut käytössä 74.4% suosituimmista 10 000 000:sta verkkosivusta (Dao 2020).

3.3.3 Käyttöliittymäkehykset

Käyttöliittymäkehykset ovat kehyksiä, joita käytetään verkkosovelluksen edustan tekemiseen. Nämä kehykset tarjoavat omat tavat rakentaa käyttöliittymiä, ja ne sisältävät usein myös työkaluja sovelluksen tilan hallintaan, verkkopyyntöjen tekemiseen ja muiden sovelluskohtaisten tehtävien suorittamiseen.

B. Satromin mukaan käyttöliittymäkehysten käyttö tehostaa käyttöliittymien tekemistä huomattavasti. Ne tarjoavat kehyskohtaisen rakenteen ja käytäntöjä koodin järjestämiseen, mikä voi helpottaa sen ymmärtämistä ja ylläpitoa. Ne tyypillisesti myös tarjoavat joukon valmiita komponentteja, sekä mahdollisuuden käyttää uudelleen jo tehtyjä komponentteja. (Satrom 2018).

JavaScriptille on saatavilla monia erilaisia käyttöliittymäkehyksiä. Käytetyimpiä ovat muun muassa React, Angular ja Vue.js. Kaikilla näillä on omat niin sanotut viitekehykset sovellusten rakentamiselle, ja ne voivat olla vahvoja eri osa-alueilla. Satrom esittelee esimerkiksi Reactin joustavaksi, Angularin sopivaksi suurille sovelluksille ja Vuen nopeimmaksi oppia. (Satrom 2018).

3.3.4 Sulautetut järjestelmät

Kuten sanottu, Node.js mahdollisti JavaScriptin suorittamisen myös selainten ulkopuolella. Node.js käytöstä sulautetuissa järjestelmissä kertovassa kirjassa P. Mulder ja K. Bresman kertovat, miksi myös JavaScript on varteen otettava vaihtoehto sulautetuissa järjestelmissä. Heidän mukaansa vaikuttavat tekijät eivät ole niinkään kielen tehokkuus, vaan ihmisten laaja

osaaminen ja JavaScriptin ympärillä olevan ekosysteemin suuruus, sekä se, että JavaScript on hyvin monikäyttöinen. JavaScript on myös *tarpeeksi* tehokas, vaikka matalan tason kielten, kuten C ja C++, ohjelmia pystyisi optimoimaan paremmin. Suurimmaksi huonoksi puoleksi Mulder ja Bresman ajattelevat Node.js ajoympäristön asennuksen vaatiman muistin. (Mulder ja Breseman 2016).

4 Pohdinta

JavaScriptin suurelle suosiolle on myös selittäviä tekijöitä, jotka eivät suoranaisesti ole olleet tietoisia päätöksiä, vaan pikemminkin sattumaa. Wirfs-Brockin ja Eiching artikkelissa Eich myöntää itsekin, että JavaScript tehtiin vähillä odotuksilla, lähinnä ajatellen aloittelevia ja osa-aikaisia ohjelmoijia. Myös kritisoijia on aina ollut. Konferenssikeskustelussa hän muistelee, mitä JavaScriptistä on epäilty: JavaScriptillä ei voi rakentaa ominaisuuksiltaan rikkaita verkkosovelluksia, se ei voi olla nopea, sitä ei saa korjattua... Mutta aina JavaScript on osoittanut epäilyt vääriksi. (Wirfs-Brock ja Eich 2020)

Wirfs-Brock ja Eich listaavat monia kysymyksiä siitä, mitä olisi voinut käydä, jos vain jotain olisi tehty toisin, kuten esimerkiksi: Mitä jos NetScape olisi palkannut jonkun muun kuin Eichin tekemään heidän selaimellensa oman skriptauskielen? Mitä jos ECMAScriptiä ei olisi? Mitä jos Microsoft ei olisi kehittänyt JScriptiä, vaan pyrkinyt yhdistämään esimerkiksi Visual Basic -kielen selaimeensa? (Wirfs-Brock ja Eich 2020). Verkkoselainten ja verkon rakenne voisi pienestäkin käänteestä olla erilainen, mutta kaikki mitä on tapahtunut, sattumaa tai ei, on johtanut siihen, mitä JavaScript on tänä päivänä.

JavaScriptin suosiota selittää myös sen monipuolisuus. Se vastaa moniin, erilaisiinkin tarpeisiin. JavaScript tulee sisäänrakennettuna suurimmissa osissa selaimia, joten yksinkertaisen kehitysympäristön saa käyttöönsä avaamalla selaimessa JavaScript-konsolin ja kokeilemalla. Sen parissa on helppo aloittaa, mutta lukuisten yhteisöprojektien avulla sen käyttötapaukset ovat lähes rajattomat.

5 Yhteenveto

Voidaan sanoa, että JavaScript on ollut oikeassa paikassa oikeaan aikaan. Vaikka JavaScriptin ensimmäinen versio kehitettiin vain kymmenessä päivässä, oli se sen ajan tarpeisiin riittävä. Tästä edelleen kehitys on ottanut suuria askelia eteenpäin, ja muun muassa valtaosa selaimista tukee jonkin version ECMAScriptiä.

Ensimmäisen version puutteista ja odotuksista huolimatta JavaScript on nykyään yksi käytetyimmistä ohjelmointikielistä, ellei käytetyin. JavaScriptin ympärille on syntynyt valtava ekosysteemi, josta voisi sanoa löytyvän jokaiselle jotain.

On monia JavaScriptistä jalostettuja kieliä, jotka voivat esimerkiksi yksinkertaistaa, selventää ja helpottaa kehittämistä lisäämällä ominaisuuksia perus JavaScriptiin. Tällaiset JavaScriptin laajennokset yleensä vaativat lisäkäännösaskeleen, jossa koodi käännetään JavaScript-koodiksi. Tämä mahdollistaa monen eri tyylisen koodin kirjoittamista, ja helpottaa entisestään JavaScriptin parissa työskentelyä, varsinkin jos on taustaa muilla kielillä.

Erilaiset projektit vaativat usein ohjelmointikieliltä hyvin vaihtelevia ominaisuuksia. Node.js ajoympäristön avulla JavaScriptiä pystyy ajaa myös selaimen ulkopuolella, joka mahdollistaa JavaScript-kehittämisen selaimen ulkopuolella. Tämä avaa oven JavaScript *Full stack*-kehitykselle, eli kehittäjä pystyy samalla kielellä kehittämään edustaa ja taustaa. Node.js projekteissa riippuvuuksien hallitsemiseen on sisäänrakennettu paketinhallintajärjestelmä npm, joka tekee riippuvuuksien asennuksesta, kustomoinnista ja käyttämisestä vaivatonta.

Kaiken kaikkiaan huomioiden JavaScriptin lähtökohdat, JavaScriptin paikka verkossa on vakuuttava, ja se on tähän asti pysynyt hyvin mukana verkon kasvussa. Tulevaisuudessa jää nähtäväksi, pystyykö JavaScript jatkaa tarjota kehittäjille, verkolle ja verkon ympärillä oleville teknologioille riittäviä työkaluja tarpeisiin.

Lähteet

Ahmed, Md Zeeshan. 2014. “Which one is better-JavaScript or jQuery”. *International Journal of Computer Science and Mobile Computing* 3 (6): 193–207.

Ashkenas, Jeremy, ym. 2009. “CoffeeScript”. *Retrieved* 4 (16): 2015.

Atwood, Jeff. 2007. *The principle of least power,* heinäkuu. https://blog.codinghorror.com/the-principle-of-least-power/.

Berners-Lee, Tim. 1998. *Principles of design.* https://www.w3.org/DesignIssues/Principles.html.

Dao, Chau. 2020. “The Nature And Evolution Of JavaScript”.

Dupal, Jan. 2016. *Concurrency support for PureScript.*

Frankston, Bob. 2020. “The JavaScript Ecosystem”. *IEEE Consumer Electronics Magazine* 9 (6): 84–89. https://doi.org/10.1109/MCE.2020.3009457.

Gardner, Philippa Anne, Sergio Maffeis ja Gareth David Smith. 2012. “Towards a program logic for JavaScript”. Teoksessa *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages,* 31–44. https://doi.org/10.1145/2103621.2103663.

Hassan, AfafMirghani. 2020. “JAVA and DART programming languages: Conceptual comparison”. *Indonesian Journal of Electrical Engineering and Computer Science* 17 (2): 845–849. https://doi.org/10.11591/ijeecs.v17.i2.pp845-849.

Heller, Martin. 2017. “What is Node. js? The JavaScript runtime explained”. *InfoWorld.*

Huberman, Bernardo A, ja Lada A Adamic. 1999. “Growth dynamics of the world-wide web”. *Nature* 401 (6749): 131–131. https://doi.org/10.48550/arXiv.1405.0749.

PYPL popularity of Programming Language index. 2022, joulukuu. https://pypl.github.io/PYPL.html.

Johnson, Ralph E. 1997. “Components, frameworks, patterns”. Teoksessa *Proceedings of the 1997 symposium on Software reusability,* 10–17. https://doi.org/10.1145/258366.258378.

Kereki, Federico. 2015. “JavaScript all the way down”. *Linux Journal* 2015 (250): 1.

Leprohon, Marc-André. 2017. “ECMAScript 6 and the evolution of JavaScript: A deeper look into the language’s new features”.

ECMA - MDN Web Docs glossary: definitions of web-related terms: MDN. 2022. https://developer.mozilla.org/en-US/docs/Glossary/ECMA.

Mulder, Patrick, ja Kelsey Breseman. 2016. *Node. js for Embedded Systems: Using Web Technologies to Build Connected Devices.* "O’Reilly Media, Inc.".

O’reilly, Tim. 2005. *Web 2.0: compact definition.*

The JS++ Type System. 2022. https://www.onux.com/jspp/tutorials/type-system.

Rozentals, Nathan. 2015. *Mastering TypeScript.* Packt Publishing.

Satrom, Brandon. 2018. “Choosing the Right JavaScript Framework for Your Next Web Application”. *Prog./Kendo UI* 34.

Severance, Charles R. 2012. “JavaScript: Designing a Language in 10 Days”. *Computer* 45:7–8. https://doi.org/10.1109/MC.2012.57.

Stack overflow developer surveys. 2011-2022. https://insights.stackoverflow.com/survey.

Tiobe index. 2022, kesäkuu. https://www.tiobe.com/tiobe-index/.

w3schools. 2022. https://www.w3schools.com/js/js_versions.asp.

What is NPM? 2022. https://www.w3schools.com/whatis/whatis_npm.asp.

Wirfs-Brock, Allen, ja Brendan Eich. 2020. “JavaScript: the first 20 years”. *Proceedings of the ACM on Programming Languages* 4 (HOPL): 1–189. https://doi.org/10.1145/3386327.