jamk

# Svelte ensimmäisenä frontend-teknologiana

## Case: JAMK Frontend-perusteet opintojakso

Siina Kaakinen

Opinnäytetyö, AMK
Toukokuu 2021
Liiketalouden ala
Tradenomi (AMK), tietojenkäsittely

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

**Kuvailulehti**

**Kaakinen, Siina**

**Svelte ensimmäisenä frontend-teknologiana. Case: JAMK Frontend-perusteet opintojakso**

Jyväskylä: Jyväskylän ammattikorkeakoulu. Toukokuu 2021, 50 sivua

Liiketalouden ala, tietojenkäsittelyn tutkinto-ohjelma, opinnäytetyö AMK

Julkaisun kieli: suomi

Verkkojulkaisulupa myönnetty: Kyllä

**Tiivistelmä**

Jyväskylän ammattikorkeakoulun tietojenkäsittelyn tutkinto-ohjelma vaihtoi vuoden 2020 opetussuunnitelmauudistuksessa ensimmäisen opetettavan frameworkin Angularista Svelteksi. Syynä vaihtoon oli oletus, että Svelte olisi helpompi ja näin sopivampi ensimmäiseksi frontend-teknologiaksi kuin Angular. Tutkimuksella haluttiin selvittää, pitääkö oletus paikkaansa. Lisäksi haluttiin selvittää mitkä tekijät vaikuttavat siihen, tuntuuko frameworkin oppiminen helpolta.

Tutkimus oli kvalitatiivinen tapaustutkimus, jossa aineistoa kerättiin kyselyillä, haastatteluilla ja kokoamalla ulkopuolisista lähteistä kattavat taustatiedot vertailukehikoksi. Tutkimuksen tietoperustassa esiteltiin frontend-kehittämisen yleisimmät termit ja frameworkit ja vertailtiin muita frameworkejä Svelteen tilastojen pohjalta.

Lopputuloksena saatiin varmistus tilaajan hypoteesille siitä, että Svelte koetaan Angularia helpommaksi teknologiaksi aloittelevalle kehittäjälle. Svelten käyttöönotto ja syntaksi olivat selkeämpiä kuin Angularissa. Svelten käyttö koettiin intuitiivisemmaksi ja opiskelijat kokivat, että Sveltellä heidän pohjatietonsa olivat riittävät frameworkin opiskeluun. Angularilla oli Svelteä paremmat urakehitysmahdollisuudet ja parempi dokumentaatio ja tuki.

Tuloksia ei voi yleistää, mutta ne toimivat hyvänä taustatutkimuksena jatkotutkimuksia ajatellen. Jatkokehitystä voi tehdä muilla frameworkeillä, jotta löydettäisiin optimaalisin framework aloittelijoille. Tutkimusta voi myös syventää toistamalla tutkimus muilla aloittelevilla opiskelijoilla ja näin saada selvyys täsmäävätkö opiskelijoiden kokemukset ja mielipiteet tämän tutkimuksen opiskelijoiden tulosten kanssa.

**Avainsanat (asiasanat)**

Svelte, Angular, ohjelmistokehitys, JavaScript, frontend, framework

**Muut tiedot (salassa pidettävät liitteet)**

jamk

**Description**

**Kaakinen, Siina**

**Svelte as the first frontend technology. Case: JAMK course Frontend basics**

Jyväskylä: JAMK University of Applied Sciences, May 2021, 50 pages

Bachelor of Business Administration. Degree programme in Business Information Technology. Bachelor's Thesis.

Permission for web publication: Yes

Language of publication: Finnish

**Abstract**

The degree program in business information technology at Jyväskylä University of Applied Sciences changed the first framework to be taught from Angular to Svelte in the 2020 curriculum reform. The reason for the switch was the assumption that Svelte would be easier and thus more suitable as a first frontend technology than Angular. The aim of the study was to find out whether the assumption is correct. In addition, the aim was to find out what factors influence whether learning the framework feels easy.

The study was a qualitative case study in which material was collected through surveys, interviews, and by compiling comprehensive background information from external sources into a reference framework. The knowledge base of the study presented the most common terms and frameworks for frontend development and compared other frameworks in Svelte, based on statistics.

As a result, confirmation was obtained for the subscriber's hypothesis that Svelte is perceived as an easier technology by Angular for a novice developer. Svelte's introduction and syntax were clearer than in Angular. The use of Svelte was perceived to be more intuitive and the students felt that at Svelte their basic knowledge was sufficient for studying framework. Angular had better career development opportunities and better documentation and support than Svelte.

The results cannot be generalized, but they serve as a good background study for further research. Further development can be done with other frameworks to find the optimal framework for beginners. The study can also be deepened by repeating the study with other novice students to gain clarity on whether the students' experiences and opinions match the results of the students in this study.

**Keywords/tags (subjects)**

Svelte, Angular, software development, JavaScript, frontend, framework

**Miscellaneous (Confidential information)**

**Sisältö**

**1 Johdanto ... 3**
**2 Opinnäytetyön tavoitteet ja menetelmät ... 4**
2.1 Tavoitteet ja tutkimuskysymykset ... 4
2.2 Tutkimusote ja aineiston keruumenetelmät ... 4
2.3 Kyselylomakkeet ... 5
2.4 Teemahaastattelut ... 5
2.5 Vertailukehikko ... 7
**3 Frontend-kehittämisen termistö ... 8**
3.1 Frontend & framework ... 8
3.2 DOM ja virtuaalinen DOM ... 9
3.3 SPA ... 10
**4 Frameworkit ja niiden vertailu ... 11**
4.1 Frameworkien esittely ... 11
4.2 Frameworkien vertailu ... 12
4.3 Projektien rakenteiden vertailu ... 14
4.4 Käyttökokemuksia eri frameworkeistä ... 18
**5 Tulokset ja johtopäätökset ... 22**
5.1 Frontend-perusteet -opintojakson kyselyjen tulokset ... 22
5.2 Teemahaastattelujen tulokset ... 25
5.3 Täydennetty vertailukehikko ... 28
**6 Pohdinta ... 31**
6.1 Johtopäätökset ... 31
6.2 Kehittämis- ja jatkotutkimusehdotukset ... 32
6.3 Tutkimuksen eettisyys ja luotettavuus ... 32
**Lähteet ... 34**
**Liitteet ... 37**
Liite 1. Frontend-perusteet opintojakson lähtötasokysely ... 37
Liite 2. Frontend-perusteet opintojakson loppukysely ... 38
Liite 3. Opinnäytetyön teemahaastattelumateriaali ... 40

**Kuviot**

Kuvio 1. HTML DOM esimerkki ... 9
Kuvio 2. Eri frameworkien tiedoston siirtokoko kilotavuina ... 13

Kuvio 3. Eri frameworkeillä luodun saman sovelluksen koodirivien määrä ..... 14
Kuvio 4. React-projektin rakenne. ..... 15
Kuvio 5. Angular-projketin rakenne. ..... 16
Kuvio 6. Vue-projektin rakenne. ..... 17
Kuvio 7. Svelte-projektin rakenne. ..... 18
Kuvio 8. Kehittäjien tyytyväisyys frontend frameworkeihin vuosina 2019-2020 ..... 19
Kuvio 9. Frontend frameworkien saamat tähdet Github-yhteisössä maaliskuussa 2021 ..... 20
Kuvio 10. Kehittäjien kiinnostus frontend frameworkkeja kohtaan vuosina 2019-2020 ..... 21
Kuvio 11. Lähtötasokyselyn jakautuminen kysyttäessä voisivatko he kuvitella työskentelevänsä ohjelmistokehittäjänä. ..... 23
Kuvio 12. Opintojakson lopussa kysyttäessä ohjelmistokehittämisesta uravaihtoehtona. ..... 25

**Taulukot**

Taulukko 1. Loppukyselystä kootut arvosanat opiskelijoiden motivaatiosta opintojaksolle, opintojakson haastavuudesta ja opiskelijoiden Svelten käyttöön. ..... 24
Taulukko 2. Angularin ja Svelten vertailukehikko täytettynä. ..... 28

# 1 Johdanto

Työelämässä tarve osaaville ohjelmistokehittäjille on suuri. On puhuttu paljon “koodaripulasta” ja siitä, kuinka tarvetta osaaville ohjelmoijille on enemmän, kuin tekijöitä on tarjolla (Korhonen, 2020). Uusia sovelluskehittäjiä tarvitaan yhtä lailla, koska tulevaisuuden senior kehittäjä on tämän päivän junior kehittäjä. Ohjelmistokehitystä on kuitenkin laajuutensa takia haastava oppia. Yksi syy siihen on, että ohjelmistokehityksessä on useita mahdollisia käytettäviä teknologioita, joten aloittelevan kehittäjän voi olla vaikea hahmottaa mistä kannattaisi aloittaa. Jos alku on vaikea, kiinnostusta ohjelmistokehittämiseen ei synny ja opiskelijat erikoistuvat muuhun opinnoissaan.

Frontend-kehittäminen on ohjelmistokehityksen osa, jossa kehitetään käyttöliittymää. Se on ohjelmoinnin perusteiden jälkeen yleensä ensimmäinen askel ohjelmistokehitykseen syventyessä. Ohjelmistokehitystä opetellessa, Jyväskylän ammattikorkeakoulussa (JAMK) opetellaan pohjatietoina HTML, CSS ja JavaScript kielet. Seuraavaksi opetellaan frontend-framework, jossa pääsee soveltamaan aiemmin oppimiaan kieliä. Tästä syystä frontendin perusteiden oppiminen ja ymmärtäminen on merkittävä askel muusta ohjelmistokehityksestä kiinnostumiseen. Frontend-teknologioista JAMK siirtyi opettamaan Svelteä Angularin sijaan lukuvuotena 2020–2021. Svelten oletettiin olevan aloittelijalle helpompi oppia. Näin kynnys päästä sisälle ohjelmistokehityksen maailmaan olisi pienempi ja se parantaisi opiskelijoiden kiinnostusta ohjelmistokehitystä kohtaan.

Svelte on melko uusi framework, minkä takia siitä ei ole tehty vielä paljoa tutkimusta. Oletus Svelten helppoudesta on luotu muiden kehittäjien mielipiteiden pohjalta, jotka ovat jo ammattilaisia. Svelten helppoutta ei ole kuitenkaan tutkittu aloittelevilla ohjelmoijilla. Uuden frameworkin oppiminen aloittelijana ei ole sama asia kuin oppia se silloin, kun on jo pohjatietoja muista frameworkeistä. Siksi opinnäytetyö pyrkii selvittämään pitääkö hypoteesi Svelten helppoudesta paikkansa, kun on kyse aloittelevasta ohjelmistokehittäjästä.

Opinnäytetyön tilaaja on JAMK ja opinnäytetyön materiaalin pohjalta luotiin JAMKin tietojenkäsittelyn tutkinto-ohjelman Frontend-perusteet-opintojakson opetusmateriaalia. Kyseinen opintojakso järjestettiin nyt ensimmäistä kertaa opetussuunnitelman muutoksen jälkeen eli opintojakson opiskelija ovat ensimmäiset, jotka opiskelevan ensimmäisenä frontend-teknologianaan Svelten eivätkä Angularia. Tässä työssä paneudutaan sekä Svelteä opiskelleiden opiskelijoiden kokemuksiin, että verrataan Svelten oppimista Angularin oppimiseen.

# 2 Opinnäytetyön tavoitteet ja menetelmät

## 2.1 Tavoitteet ja tutkimuskysymykset

Työn tavoitteena on selvittää, kokevatko ensimmäisen vuoden tietojenkäsittelyn opiskelijat Svelten helpoksi ensimmäiseksi frontend-teknologiaksi. Tavoitteella pyritään auttamaan JAMKia parantamaan opiskelijoiden oppimiskokemusta ja kannustaa opiskelijoita jatkamaan kehittymistä ohjelmistokehittämisen parissa. Opinnäytetyötä on rajattu poistamalla aiheen pedagoginen tarkastelu. Opinnäytetyössä ei siis oteta kantaa opetustapoihin. Työn empiirisessä osiossa keskitytään vertailemaan vain Svelteä ja Angularia keskenään eikä oteta kantaa muihin frameworkeihin.

Työssä esitellään frontend-kehittämisen yleiset termit ja esitellään yleisimmät frameworkit. Empiirisessä osiossa selvitetään mitkä asiat saavat uuden teknologian oppimisen tuntumaan helpolta tai vaikealta. Lisäksi Svelteä opiskelleiden opiskelijoiden kokemuksia verrataan Angularilla frontendin perusteet oppineiden opiskelijoiden kokemuksiin. Työssä vertaillaan Svelteä ja Angularia keskenään sekä teorian pohjalta, että haastattelemalla opiskelijoita. Vertailussa hyödynnetään useita tiedonkeruumenetelmiä, joista tarkemmin myöhemmin tässä luvussa.

Opinnäytetyön tutkimuskysymykset ovat:

- Mikä tekee frameworkista helpon oppia?
- Soveltuuko Svelte ensimmäiseksi frontend-teknologiaksi aloittelevalle kehittäjälle?
- Onko Svelte helpompi oppia kuin Angular?

## 2.2 Tutkimusote ja aineiston keruumenetelmät

Opinnäytetyön tutkimusote on kvalitatiivista eli laadullista tutkimusta. Tutkimuksen alussa on hypoteesi, jonka opinnäytetyö pyrkii todistamaan joko oikeaksi tai vääräksi. Tässä työssä hypoteesi on oletus siitä, että Svelte on helppo ensimmäinen frontend-teknologia. Tutkimuksen käytössä on vain yksi havaintoyksikkö, jota voidaan hyödyntää tutkimuksessa. Tällä tarkoitetaan, että käytössä

on vain yhden opintojakson opiskelijoiden kokemukset. Tästä syystä tulokset eivät ole yleistettävissä laajemmin muihin vastaaviin tai tuleviin opintojaksoihin. Tämä on tyypillistä laadullisissa tutkimuksissa. (Kananen, 24–25, 2008.)

Koska havainnointiyksikkö oli pieni, tämä opinnäytetyö on myös tapaustutkimus. Tapaustutkimus pyrkii tuottamaan mahdollisimman paljon tietoa yksittäisistä tapauksista, jotka eivät ole suoraan yleistettävissä sellaisenaan. Saadusta aineistosta muodostuu kuitenkin kokonaisuus, jota voidaan tarkastella. Tutkimusprosessi kuvataan työssä niin hyvin, että kuka vain voi halutessaan toistaa tutkimuksen. Tapaustutkimukselle tyypilliseen tapaan, tässä työssä hyödynnettiin useita aineistonkeruumenetelmiä, jotka kuvataan seuraavaksi. (Saarela-Kinnunen & Eskola, 189–190, 2010.)

## 2.3 Kyselylomakkeet

Ensimmäisenä aineistonkeruumenetelmänä olivat kyselylomakkeet. Kyselyt toteutettiin sähköisesti Microsoft Forms-työkalun avulla ja ne pidettiin lyhyinä. Suurimpaan osaan kysymyksistä oli valmiit vastausvaihtoehdot, joista valita. Kysymyksiä luotaessa pyrittiin välttämään monitulkintaisuutta avaamalla kysymyksiä mahdollisimman hyvin. Käytetyt asteikot selitettiin auki jokaisen kysymyksen kohdalla. (Järvinen & Järvinen, 155–158, 2000.)

Kyselyillä haluttiin seurata Frontend-perusteet opintojakson oppimiskokemuksia, jotta saataisiin selkeämpi mielikuva siitä, kuinka opiskelijat kokivat perusteiden ja Svelten oppimisen. Opiskelijoita pyydettiin täyttämään opintojakson alussa ja lopussa kyselylomakkeet. Lähtötasokyselyllä (Liite 1) selvitettiin opiskelijoiden lähtökohdat ja kiinnostus opintojakson aiheisiin. Loppukyselyllä (Liite 2) selvitettiin opiskelijoiden kokemuksia opintojaksosta ja kasvoiko heidän kiinnostuksensa aiheeseen. Kyselylomakkeiden tuloksia on tarkoitus hyödyntää yhtenä osana, kun päätellään vastaus toiseen tutkimuskysymykseen: soveltuuko Svelte ensimmäiseksi frontend-teknologiaksi aloittelevalle kehittäjälle.

## 2.4 Teemahaastattelut

Toinen aineistonhankintamenetelmä oli teemahaastattelut. Haastatteluprosessi eteni Kanasen (73–74, 2008) ohjeistuksen mukaisesti. Kysymykset olivat avoimina ja haastattelutilanne keskustelumaisena. Haastattelut toteutettiin yksilöhaastatteluina ja haastateltaville kerrottiin etukäteen

mitä teemoja haastattelussa käsitellään. Haastattelut nauhoitettiin haastateltavien luvalla. Nauhoitteet litteroitiin ja litteroidut tiedostot käytiin läpi varmistaen, ettei niissä ole mitään, mistä voisi tunnistaa haastateltavan. Litteroinnista myös siistittiin pois puheen sisältämät takeltelut, tauot ja täytesanat. Opinnäytetyössä oltiin ensisijaisesti kiinnostuneita puheen sisällöstä, jolloin litteroidun aineiston siistiminen selkeyttää pääkohtien löytämistä (Kallinen, T & Kinnunen, T. N/d.). Litteroitu aineisto teemoitettiin ja tulokset löytyvät tämän työn luvusta 5.2.

Haastattelu valittiin aineiston keruumenetelmäksi, koska haluttiin vertailla kokemuksia frameworkien oppimisessa. Opinnäytetyön painopiste on selvittää ensimmäisen frontend-teknologian kokemuksia, mutta vertailun vuoksi tarvittiin myös kokemuksia Angularilla oppimisesta. Jokainen voi oppia ensimmäisen frontend-teknologian vain kerran, mutta jos on oppinut vain yhden frontend frameworkin, kuinka voi tietää onko oma oppimisprosessi ollut helppo vai vaikea muihin frameworkeihin verrattuna? Jotta tähän saataisiin varmistus, valittiin muutama opiskelija vertailemaan Svelteä ja Angularia keskenään teemahaastatteluihin.

Haastatteluiden avulla Svelteä oppineiden kokemuksia voitaisiin verrata Angularia oppineiden kokemuksiin. Haastatteluihin valittiin yhteensä kuusi henkilöä, joista puolet olivat oppineet Svelten ensimmäisenä frontend-teknologianaan ja toinen puoli Angularin. Haastateltavat valittiin sen pohjalta, että he olivat käyneet peruskurssin omasta frameworkista eli he eivät olleet jättäneet kurssia kesken. Haastattelut keskittyivät haastateltavien oppimiskokemuksiin ensimmäisestä frameworkista. Haastatteluja varten luodut kysymykset ja vertailtava koodi ovat liitteessä 3.

Haastattelu oli jaettu kolmeen osioon. Ensimmäinen osio käsitteli haastateltavan lähtökohtia peruskurssiin ja frontendin perusteiden oppimiseen. Toisessa osiossa keskusteltiin frameworkista ja sen oppimiskokemuksista. Kolmannessa osiossa vertailtiin Svelteä ja Angularia keskenään. Tätä osiota varten oli luotu kaksi käyttöliittymältään samanlaista projektia, joista toinen oli luotu Sveltellä ja toinen Angularilla. Haastateltavien kanssa katsottiin, kuinka samat asiat oli luotu kussakin frameworkissä ja mitä ajatuksia niistä heräsi.

## 2.5 Vertailukehikko

Kolmas aineistonkeruumenetelmä kokosi kaiken kootun aineiston yhteen. Tarkoitus oli etsiä tietyt piirteet, joiden valossa aineistoa tarkastellaan. Kahden aiemman aineistonkeruumenetelmän havainnoista tutkittiin millä tavalla valitut piirteet niissä esiintyivät. Sen lisäksi tutkittiin myös muita lähteitä ja koottiin niistä kokemuksia ja havaintoja, jotka sopivat valittuihin piirteisiin. Vertailukehikossa piirteet kootaan taulukkoon riviotsikoiksi ja frameworkit laitetaan sarakeotsikoiksi. Jokaisen piirteen kohdalla merkataan, sopiiko se paremmin Sveltelle vai Angularille. Luvussa 5.3. on koottuna valmis vertailukehikko (taulukko 2).

Jotta voidaan vastata kysymykseen, onko Svelte helpompi oppia kuin Angular, täytyy vastata ensimmäiseen tutkimuskysymykseen: mikä tekee frameworkista helpon oppia. Kirjaimellisesti, jos jokin on helppoa, se on vaivatonta ja vähän taitoa vaativaa (helppo, N/d). Frontend-kehittämisessä on monia piirteitä, jotka vaikuttavat siihen, tuntuuko oppiminen vasta-alkajasta helpolta vai ei. Samoja helppouden piirteitä voi huomata kaikkia ohjelmointikieliä opetellessa. Vertailukehikkoa varten koottiin eri lähteistä yleisimpiä ja toistuvimpia piirteitä, jotka tekevät ohjelmoinnin oppimisesta helpompaa. Toistuvat teemat on esitelty alla.

**Käyttöönotto:** Jos uuden teknologian käyttöönotto on haastavaa, into oppia voi kadota jo ennen kuin on päästy aloittamaan edes itse ohjelmointia (Robinsonin, n/d). Käyttöönoton mukavuutta voidaan arvioida muun muassa sillä vaatiiko aloittaminen paljon asentamista, tai kauanko aloittaminen kestää.

**Dokumentaatio ja tuki:** Jos ohjelmointityössä kohtaa ongelman, on kriittistä löytääkö ongelmaan apua helposti joko teknologian dokumentaatioista tai erilaisista yhteisöistä. Sekä Bhat (2019), Abi-Nakhoul (n/d) että Robinson (n/d) ovat tästä yhtä mieltä. Robinson huomioi vielä erikseen, että virheenkorjauksen, eli debugauksen, tulisi olla helppoa. Tässä teknologian omat virheilmoitukset ja yhteensopivuudet linttereiden kanssa ovat merkityksellisiä.

**Syntaksi**: Kirjoitetun koodin selkeys nousi esille monissa listauksissa, kun puhuttiin aloittelijalle sopivasta ohjelmoinnista. Bhatin (2019), Abi-Nakhoulin (n/d) ja Robinsonin (n/d) listauksissa luokiteltiin tärkeäksi, että kirjoitettu koodi näyttää hyvältä, on yksinkertaista ja luettavaa. Bhat lisää vielä, että myös koodin vähyys on etu.

**Uramahdollisuudet:** Kun ohjelmointikielen on oppinut, koetaan tärkeäksi, että oppiminen johtaa työpaikan saamiseen ja hyvään urakehitykseen. Hyvät urakehitysmahdollisuudet johtavat usein myös hyvään palkkatasoon. (Bhat, 2019)

**Intuitiivisuus:** Intuitiivisuudella tarkoitetaan sitä, kun jokin on vaistomaisesti tai välittömästi tajuttu (intuitiivinen, n/d). Frameworkien kohdalla sillä voidaan tarkoittaa esimerkiksi sitä, kuinka helposti frameworkin logiikka lähtee selkenemään ensikertalaiselle tai kuinka helpolta oppiminen tuntuu. Bhat (2019) ja Robinson (n/d) molemmat korostavat miten helppo oppimiskokemus tekee teknologiasta paremman.

**Pohjatietojen hyödyntäminen:** JAMKissa opetetaan tietyt perusteet pohjatiedoksi ennen kuin aletaan opiskelemaan frontend-perusteita tai frameworkejä. Aloittelijan kannalta on parempi, jos olemassa olevia pohjatietoja pystyy hyödyntämään mahdollisimman samanlaisina kuin mitä ne on oppinut. Siksi kiinnostaa pystyykö pohjatietoja hyödyntämään suoraan frameworkissä?

# 3 Frontend-kehittämisen termistö

## 3.1 Frontend & framework

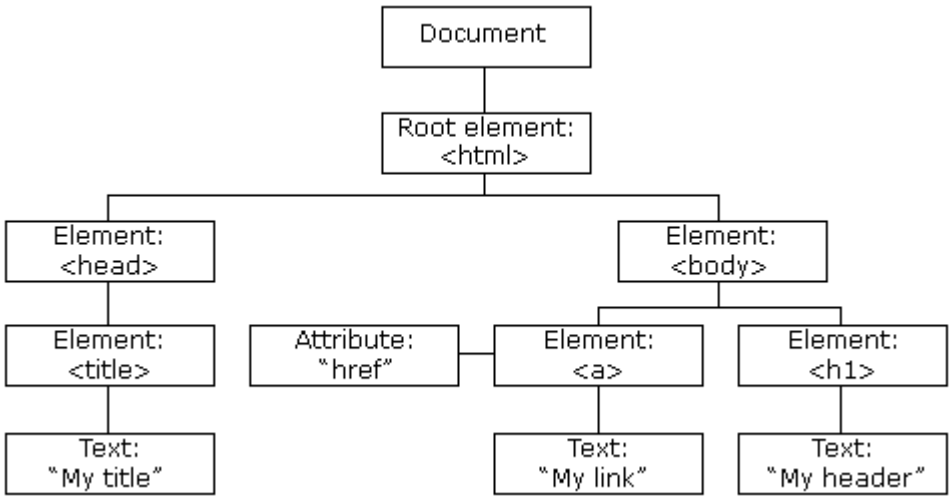
Ohjelmistokehityksessä käytetään nimitystä frontend käyttäjälle näkyvästä osuudesta. Frontend-kehittäminen on siis käyttöliittymän kehittämisestä. Kehitystyö tehdään yhdistämällä kolmea ohjelmointikieltä: HTML, CSS ja JavaScript. (What is a front-end developer, 2018.) Näistä HTML-kielellä luodaan sivuston/sovelluksen rakenne, CSS:llä muokataan sen ulkonäköä ja JavaScriptillä sivustolle/sovellukselle saadaan toiminnallisuuksia. (Web Technologies Employed by Front-End Developers, 2018.)

Framework on ohjelmistokehys, jonka tarkoitus on luoda koodille valmis rakenne ja kokoelma valmiita koodikirjastoja, tukiohjelmia, työkaluja jne. Framework jäsentelee koodia rakenteen mukaan ja se vaatii myös ohjelmoijaa kirjoittamaan koodia frameworkin rakenteeseen sopivaksi. Rakenne kuitenkin helpottaa ohjelmointityötä helpottamalla koodin uudelleen käyttämisessä sovelluksessa tai sivustolla. Myös frameworkien valmiit toiminnallisuudet helpottavat tiettyjen dynaamisten ja reaktiivisten toimintojen ohjelmointia. Frameworkeja on paljon erilaisia, mutta frontend-frameworkit ovat usein JavaScript-pohjaisia. (Morris, N/d.)

Frameworkeissä komponenttipohjaisuus mahdollistaa koodin uudelleen käyttämisen. Frameworkin sisältö kootaan komponenteista, jotka sisältävät aina tietyn osa koodia. Komponentit voivat koostua omista pienemmistä komponenteista. Esimerkiksi sovelluksen painikkeilla voi olla oma komponentti, jota voi hyödyntää kaikkialla sovelluksessa. Painikkeen koodi on kirjoitettu vain kerran painikkeen komponenttiin ja tätä komponenttia kutsutaan aina tarvittaessa muista komponenteista. Komponenttipohjaisuus vähentää kirjoitettavan koodin määrää ja helpottaa koodin ylläpidettävyyttä (Sandeep, 2015).

## 3.2 DOM ja virtuaalinen DOM

DOM on lyhenne sanoista Document Object Model (suom. dokumenttioliomalli). DOM on HTML- ja XML-dokumenttien ohjelmointirajapinta, jonka avulla määritellään mainituille dokumenteille tietty hierarkkinen rakenne. Rakenne kuvataan usein puumaisena, mutta se ei ole ainoa rakenne DOMille. Rakenne koostuu objekteista, joita pystytään muokkaamaan ja joiden välillä tieto liikkuu vapaasti. Kuviosta 1. näkee esimerkin HTML DOMista, jossa on HTML-dokumentille tyypillinen puumaisen hierarkia. Hierarkian päällä on dokumenttiolio ja sen alla on dokumentin muut elementit. DOM on luotu yhtenäiseksi standardiksi, joka toimisi kaikissa ympäristöissä, sekä kaikilla selaimilla ja ohjelmointikielillä. (Robie, 1998)




Kuvio 1. HTML DOM esimerkki (JavaScript HTML DOM, n/d).

Toivainen (2020) tuo esille pro gradu -tutkielmassaan, että vaikka DOM mahdollistaa tehokkaan elementtien määrittelyn ja hallinnan, sen ongelmana on virhealttius ja raskaus. Virhealttius johtuu siitä, että DOM on monimutkainen ja huonosti tunnettu, eikä sitä ymmärretä täysin. Raskaus taas johtuu siitä, että aina kun johonkin elementtiin tehdään muutos, alkaa hidas päivitysprosessi. Tästä syystä DOMiin halutaan tehdä mahdollisimman hallittuja ja pieniä muutoksia. Tässä frameworkit ovat auttaneet paljon. (Toivainen, 2020)

Koska DOMin muokkaaminen vaatii sitä enemmän suorituskykyä, mitä enemmän muutoksia tehdään, on luotu virtuaalinen DOM. Siinä alkuperäisestä DOMista luodaan kopio keskusmuistiin JavaScript oliona, johon muutokset tehdään. Muokkausten jälkeen muutokset tuodaan alkuperäiseen DOMiin kohdennettuina ja optimoituina. Virtuaalista DOMia hyödynnetään mm. Reactissa ja Vuessa. (Aderinokun, 2018) Harris (2018) kuitenkin tuo esille, että vaikka virtuaalinen DOM helpottaa ohjelmointia, koska ei tarvitse miettiä suorituskykyä tai tilasiirtymiä (transitioita), se ei tarkoita, ettei samanlaisiin ohjelmointimalleihin päästäisi myös ilman virtuaalista DOMia.

## 3.3 SPA

SPA on lyhenne termistä Single Page Application eli yhden sivun sovellus. SPA-sovellukset toimivat selaimessa. Selaimeen ladataan koko sovellus, jolloin sovelluksen sivuja ei tarvitse hakea palvelimelta kesken sovelluksen käytön. Vain näytettävä tieto vaihtuu sovellusta käytettäessä. Tämä nopeuttaa ja helpottaa sovelluksen käyttöä. Tosin SPA-sovellusten heikkoutena on ensimmäisen latauksen hitaus, jossa koko sovellus ladataan selaimelle. JavaScript pohjaiset frameworkit rakentavat SPA-sovelluksia. (Skólski, 2016)

Klassinen sovellus on nimeltään Multi Page Application (MPA) eli monisivuinen sovellus. Näissä, aina kun vaihdetaan sivua, joudutaan lataamaan koko sivu uudestaan. Sovelluksen lopullinen käyttötarkoitus määrittää onko järkevämpää käyttää SPA vai MPA sovelluskehitystä. (Skólski, 2016)

# 4 Frameworkit ja niiden vertailu

## 4.1 Frameworkien esittely

**React**

Tällä hetkellä käytetyin frontend framework on React (Greif & Benitte, 2020). React kehitettiin vuonna 2013 Facebookin toimesta (Gackenheimer, 2015). Tarkalleen ottaen React on kirjasto (library). Siinä missä moni framework tarjoaa suoraan sisäänrakennettuna tiettyjä ominaisuuksia ja toimintoja, React hyödyntää eri toimintoihin eri paikoista saatavia kokonaisuuksia ja kokoaa ne yhteen kirjastoksi (Baer, n/d).

Reactissa elementit kirjoitetaan JSX:llä, joka on JavaScriptin syntaksin laajennus. JSX auttaa sekoittamaan JavaScriptin HTML: ajan ja tekemään käyttöliittymästä sillä tavalla reaktiivisempaa. JSX: n kanssa React-komponentti kirjoitetaan kuin se olisi mukautettu HTML-tagi. React-komponentit kirjoitetaan yleensä JSX:llä, vaikka on mahdollista kirjoittaa React ilman sitä. (Baer, n/d) Reactissa hyödynnetään virtuaalista DOM:a (Gackenheimer, 2015).

**Angular**

Yleisimmistä frameworkeistä vanhin, Angular, on uudelleen luotu framework AngularJS:stä. Alkuperäisen AngularJS:n kehitti Google vuonna 2010 ja vuonna 2016 se muovautui nykyiseksi Angulariksi. Vaihdon myötä vaihdettiin myös JavaScriptin käyttö TypeScriptiksi. (Angular vs React vs Vue vs Svelte: A Comparison of the Top Frontend Frameworks., 2020.)

Angularilla on tiukin arkkitehtuuri mitä tulee frameworkin käyttöön. Se ei siis mukaudu kehittäjien tarpeisiin samalla tavalla kuin muut frameworkit, mutta tarjoaa hyvin valmiita ja pitkälle hiottuja toimintoja kehittäjien käyttöön. (Angular vs React vs Vue vs Svelte: A Comparison of the Top Frontend Frameworks., 2020.)

**Vue**

Vue on avoimeen lähdekoodiin perustuva, laajan kehitystiimin aikaansaannos, joka kehitettiin vuonna 2014. Sen laittoi aluille Evan You, joka oli aiemmin työskennellyt Googlelle AngularJS:n parissa. (Angular vs React vs Vue vs Svelte: A Comparison of the Top Frontend Frameworks., 2020.)

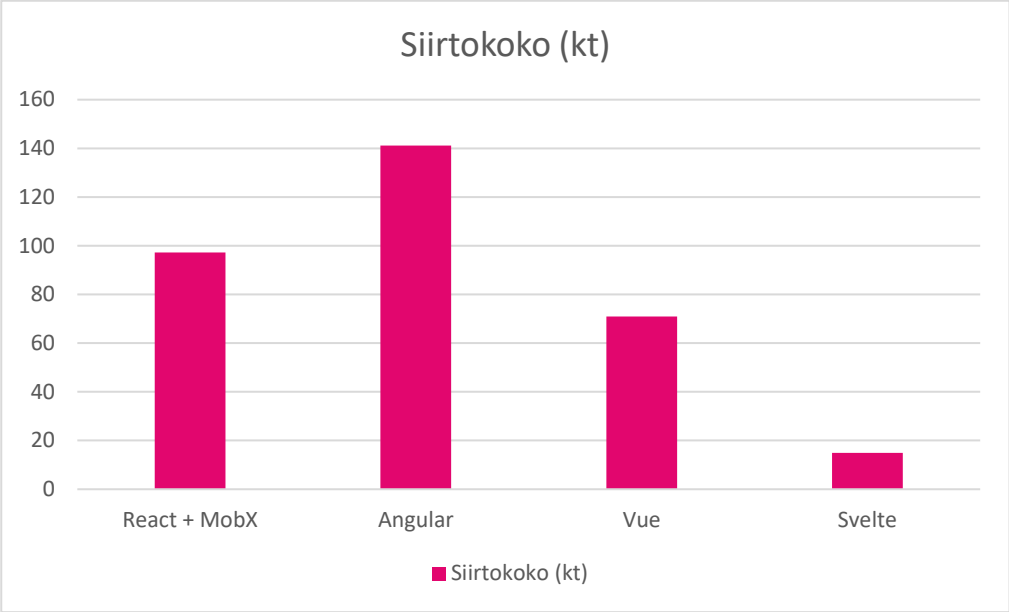
Vuen arkkitehtuuri on Reactin tavoin kehittäjien tarpeeseen mukautuva ja siksi kehittäjät pitävät siitä. Sivut luodaan tavallisella HTML ja JavaScript yhdistelmällä ja frameworkiin on lisätty toiminnallisuuksia, jotka helpottavat monien ominaisuuksien luomista. Tästä syystä Vue on noussut viime vuosina suosituimmaksi frameworkiksi. (Angular vs React vs Vue vs Svelte: A Comparison of the Top Frontend Frameworks., 2020.)

**Svelte**

Svelte on vuonna 2016 kehitetty ilmainen komponenttipohjainen avoimen lähdekoodin framework (Harris, 2020). Muista frameworkeistä poiketen, Svelte on kääntäjä (compiler) eli Svelte kääntää koodin pieniksi JavaScript-moduuleiksi. Tämä tekee sovelluksesta nopeamman ja kevyemmän, koska suoritusaikaa selaimessa ei tarvita. Svelte on luotu seuraamaan staattisesti mitä sovellus tarvitsee milläkin hetkellä ja se kääntää vain sillä hetkellä tarvittavan koodin sovelluksen käyttöön. (Harris, 2016)

## 4.2 Frameworkien vertailu

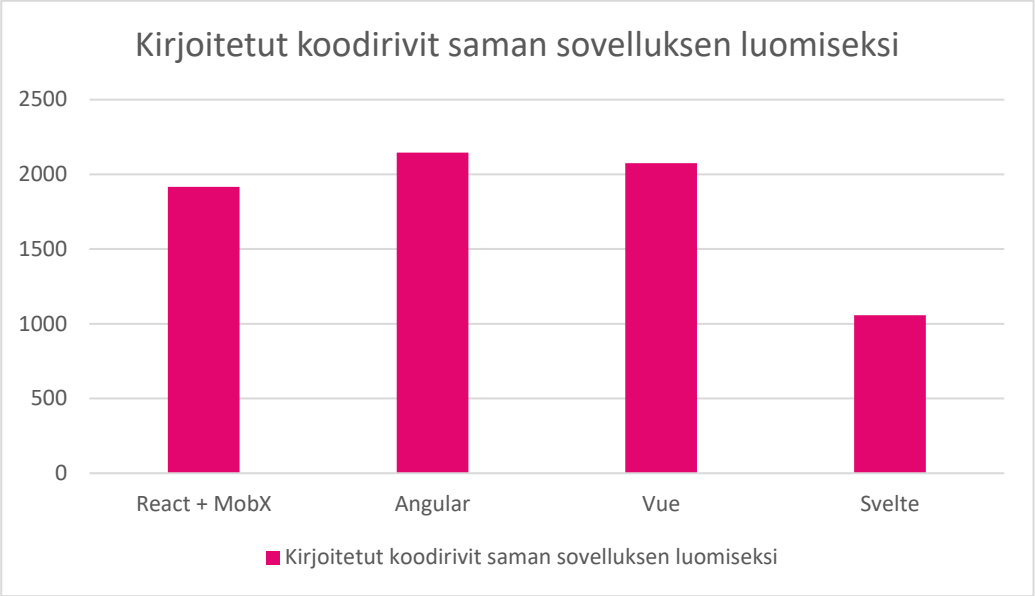
Jo se, että Svelte on kääntäjä, erottaa sen muista yleisesti käytetyistä frameworkeistä ja kirjastoista: Reactista, Angularista ja Vuesta. Muissa frameworkeissä selaimelle toimitetaan halutun koodin lisäksi myös frameworkin oma koodirunko. Svelten käännettyyn koodiin ei lisätä tätä ylimääräistä framework-koodia. Sen ansiosta Svelte-sovellukset ovat selvästi pienempiä kuin muiden frameworkien sovelluksista. (Schwarzmüller, 2019) Ero sovelluksien koossa on selvästi nähtävissä mm. sovelluksen siirtokoosta. Eri frameworkien siirtokoiden erot näkee kuviosta 2.




Kuvio 2. Eri frameworkien tiedoston siirtokoko kilotavuina. (Schae, 2020)

Svelten nopeus selittyy sillä, että Svelte luo koodista nippuja (bundle). Nipuissa on vain suorittamiseen tarvittava koodi ja koska mitään ylimääräistä ei viedä selaimeen, sovellus lataa ja toimii nopeasti. Tästä samasta syystä myös koodimuutokset voidaan tehdä päivittämällä alkuperäistä DOMia eikä Sveltessä tarvitse luoda virtuaalista DOMia kuten Reactissa ja Vuessa. (Harris, 2020)

Svelteä kehittäessä, kehittäjä Harris pyrki myös pääsemään minimaaliseen koodin määrään, koska mitä vähemmän koodia joudutaan kirjoittamaan, sitä pienempi riski on virheille (Harris, 2019). Tämä näkyy myös Schaen (2020) vertaillessa eri frameworkien koodimäärää saman sovelluksen toteuttamiseksi. Schaen luvuista on koottu kuvion 3 pylväsdiagrammi.




Kuvio 3. Eri frameworkeillä luodun saman sovelluksen koodirivien määrä. (Schae, 2020)

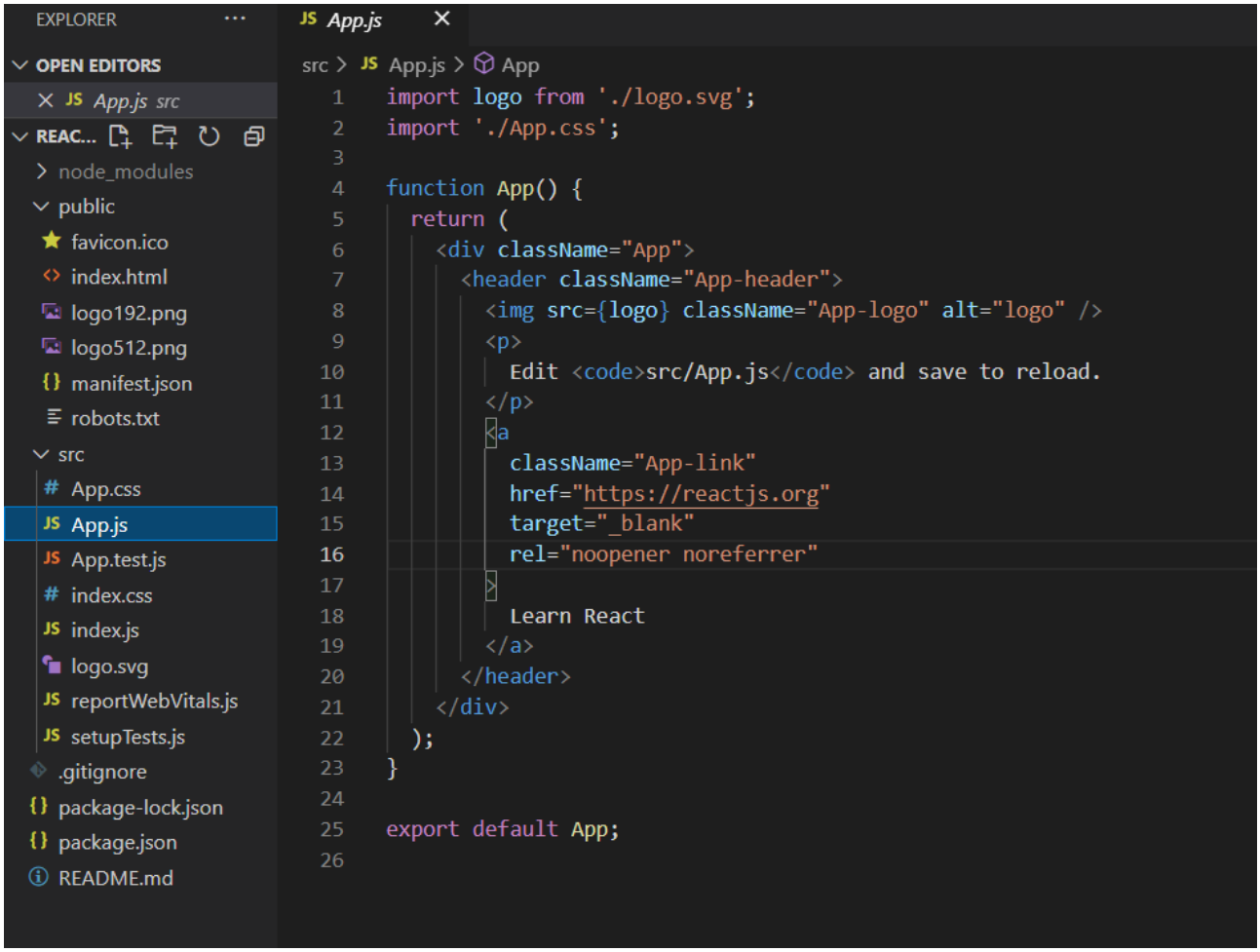
Svelteen on rakennettu sisään paljon toimintoja, joihin muissa frameworkeissä tarvitaan erillisiä kirjastoja (Harris, 2020). Svelte ei kuitenkaan sisällä paljoa ylimääräisiä ominaisuuksia, toisin kuin muut frameworkit. Nämä ominaisuudet ovat frameworkeihin sisäänrakennettuja ja ne muun muassa parantavat sovelluksen toimimista eri selainympäristöissä, parantaen sovelluksen käyttömukavuutta. (Schwarzmüller, 2019)

## 4.3 Projektien rakenteiden vertailu

Yhteisenä tekijänä kaikilla eri framework projekteilla on, että niissä on kaikissa src-kansio, jonka sisään kootaan komponentit ja muut koodit. Kaikissa projekteissa on myös index.html-tiedosto. Koska frameworkit tuottavat SPA-sovelluksia, tämä index.html toimii koko projektin HTML-runkona, jonka sisälle toimitetaan komponentit sitä mukaan, kun niitä on tarkoitus näyttää käyttöliittymässä. Vaikka toimintaperiaate on lähtökohtaisesti sama, eri frameworkien projektien rakenne eroaa toisistaan selvästi. Näitä eroavaisuuksia vertaillaan tässä alaluvussa.

Kuviossa 4 on Reactin vastaluotu projekti, jossa näkyy projektin rakenne vasemmalla ja juurikomponentin sisältö oikealla. Siitä voi nähdä, ettei React-projektilla ole paljon tiedostoja työstämättömässä projektissa. Projektilla on melko löyhä tiedostorakenne eli React ei tarjoa valmiina mitään

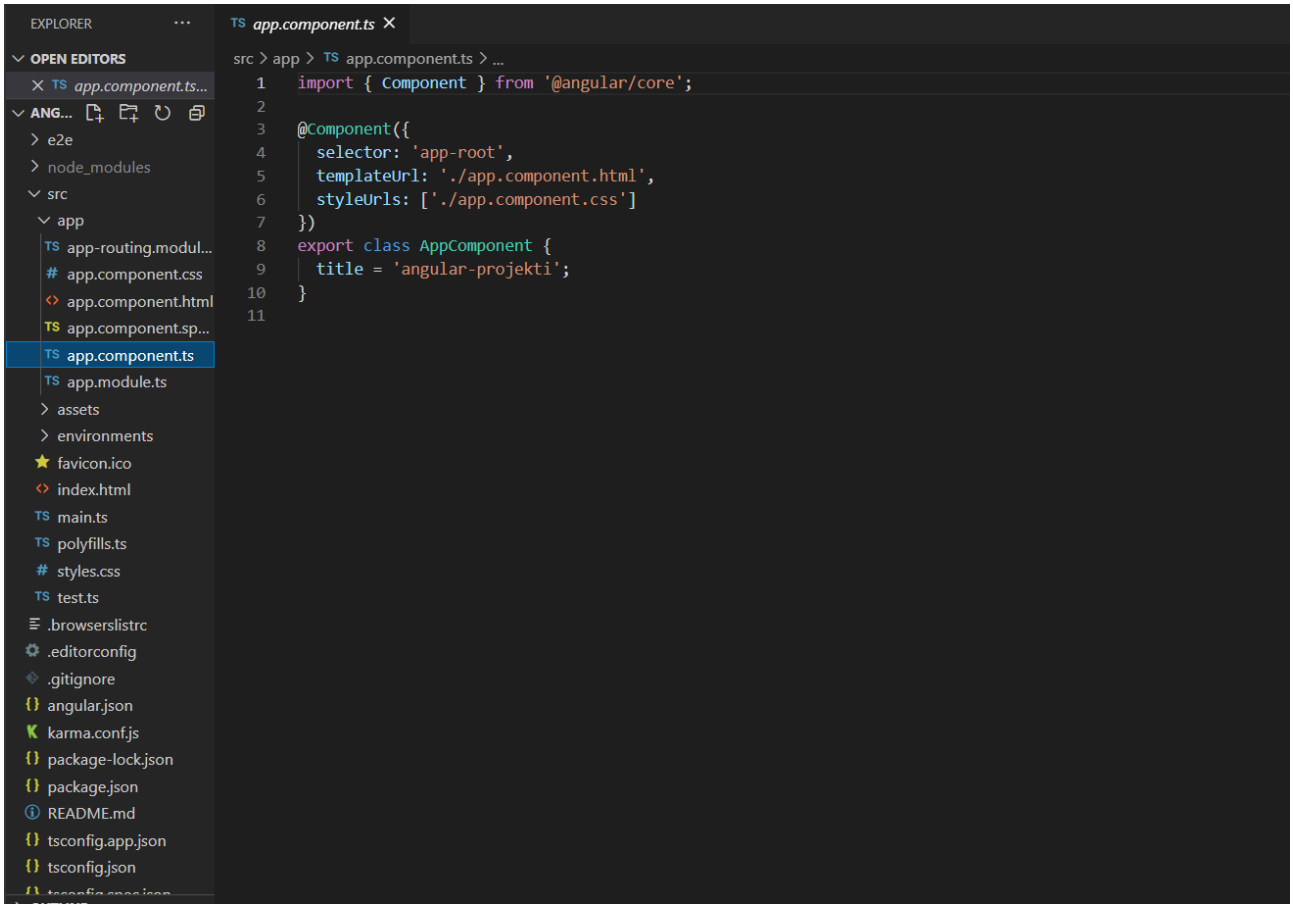
tiettyä kansiorakennetta, johon tulevat tiedostot tulisi luoda. React-komponentit ovat .js-päätteisiä. Komponenttirakenteesta huomaa, kuinka Reactissa hyödynnetään JSX:ää ja siksi HTML:n näköinen koodi on laitettu JavaScript-funktion sisälle. React-komponenttien muotoilut ovat erillisessä CSS-tiedostossa, joka tuodaan komponentille importin avulla ja React tarjoaa myös testitiedoston komponenttien tarkistamiseen. (Components and Props, n/d.)




Kuvio 4. React-projektin rakenne.

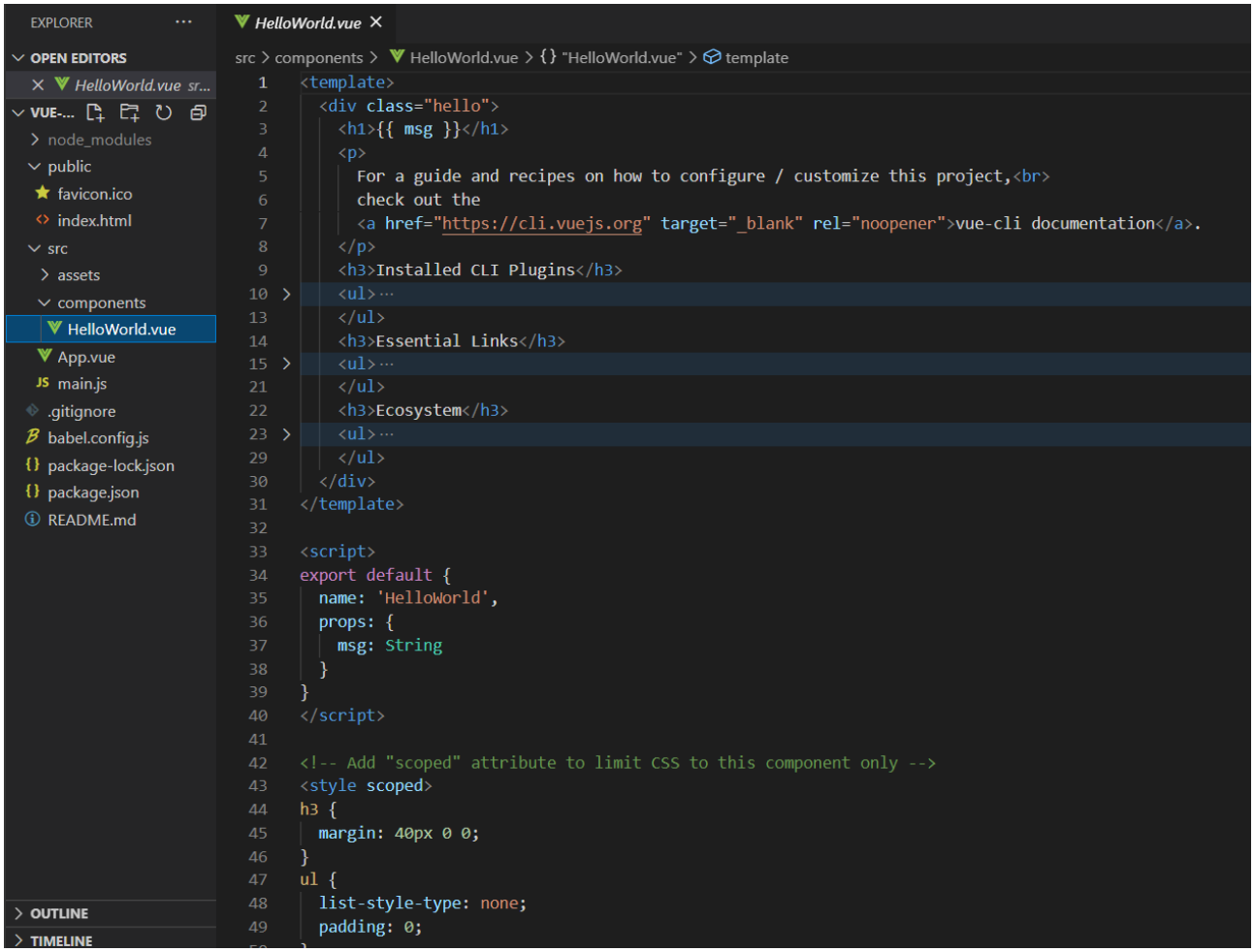
Angular-projektissa on selvästi kaikista eniten sisältöä. Kuviosta 5 näkee, kuinka Angular-projektilla on selvä kansiorakenne. Kuviossa näkyy projektin rakenne vasemmalla ja juurikomponentin TypeScript-tiedoston sisältö oikealla. Kansiot kuten app, assets ja environments ovat valmiiksi annettu. Komponentit kootaan omaan app-kansioon, jossa on valmiina myös routing-moduuli, joka vastaa sovelluksessa navigoimisesta. Jokainen komponentti koostuu neljästä tiedostosta: TypeScript-tiedostosta (app.component.ts), HTML-tiedostosta (app.component.html), testitiedostosta

(app.component.spec.ts) ja CSS-tiedostosta (app.component.css). Komponentin tiedostot tunnistavat nimessä olevasta component-osasta ja jokaisen komponentin tiedostot kootaan yhteen @Component-objektin sisälle kyseisen komponentin TypeScript-tiedostoon. Angular-komponentit kootaan usein omiksi kansioikseen, jotta tiedostomäärät pysyisivät hahmotettavina, kun komponenttien määrä kasvaa. (What is Angular? n/d.)



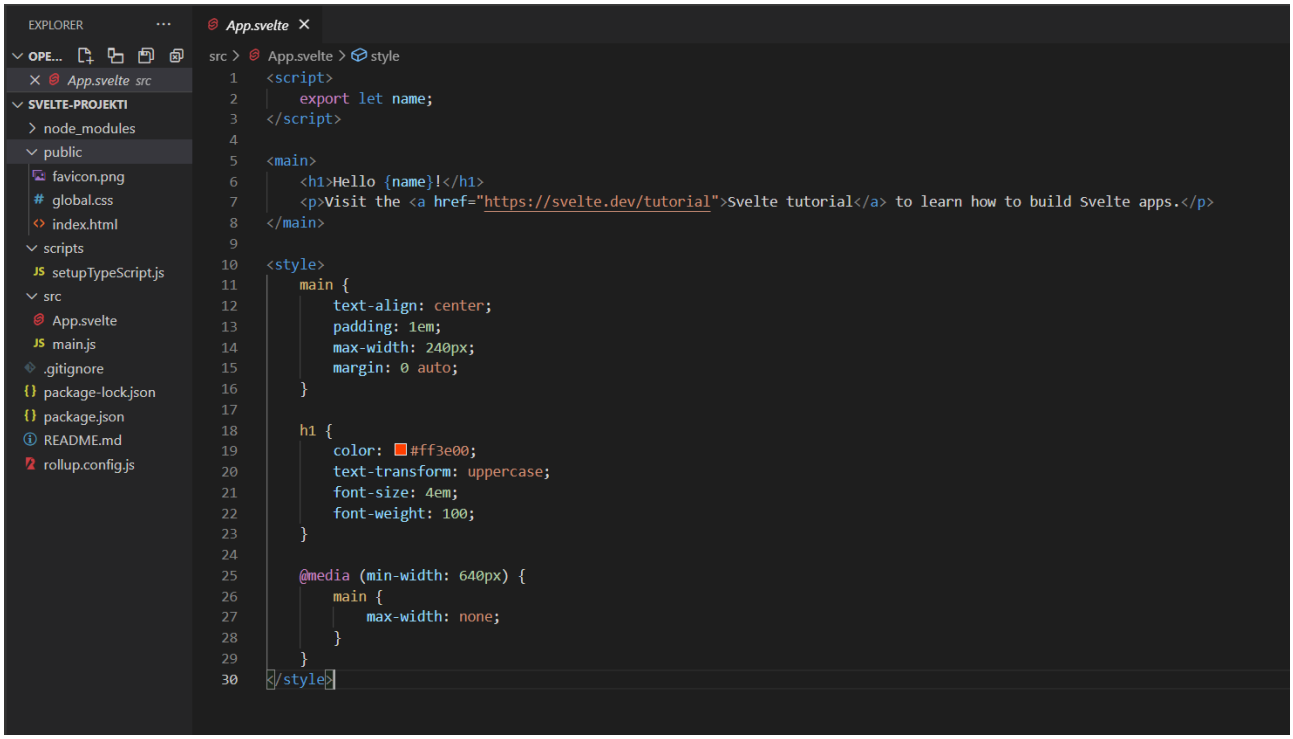

Kuvio 5. Angular-projketin rakenne.

Kuviosta 6 näkee, että Vue-projektissa on selvästi vähemmän sisältöä kuin kahdessa aiemmin esitellyssä projektissa. Kuten kuviosta näkee, rakenteellisesti Vuessa on pieni rakenne. Vue antaa projektilleen valmiina components- ja assets-kansiot. Vuen komponentit ovat vain yksi .vue-päätteinen tiedosto, jossa on kaikki mitä kyseinen komponentti tarvitsee. Kuten kuviosta 6 huomaa, komponentin rakenne alkaa HTML-osiolla, joka on <template> tagien sisällä. Sitä seuraa JavaScript-osuus, joka on <script> tagien sisällä ja lopulta kyseisen komponentin CSS-osio on <style> tagien sisällä. (Introduction, n/d.)




Kuvio 6. Vue-projektin rakenne.

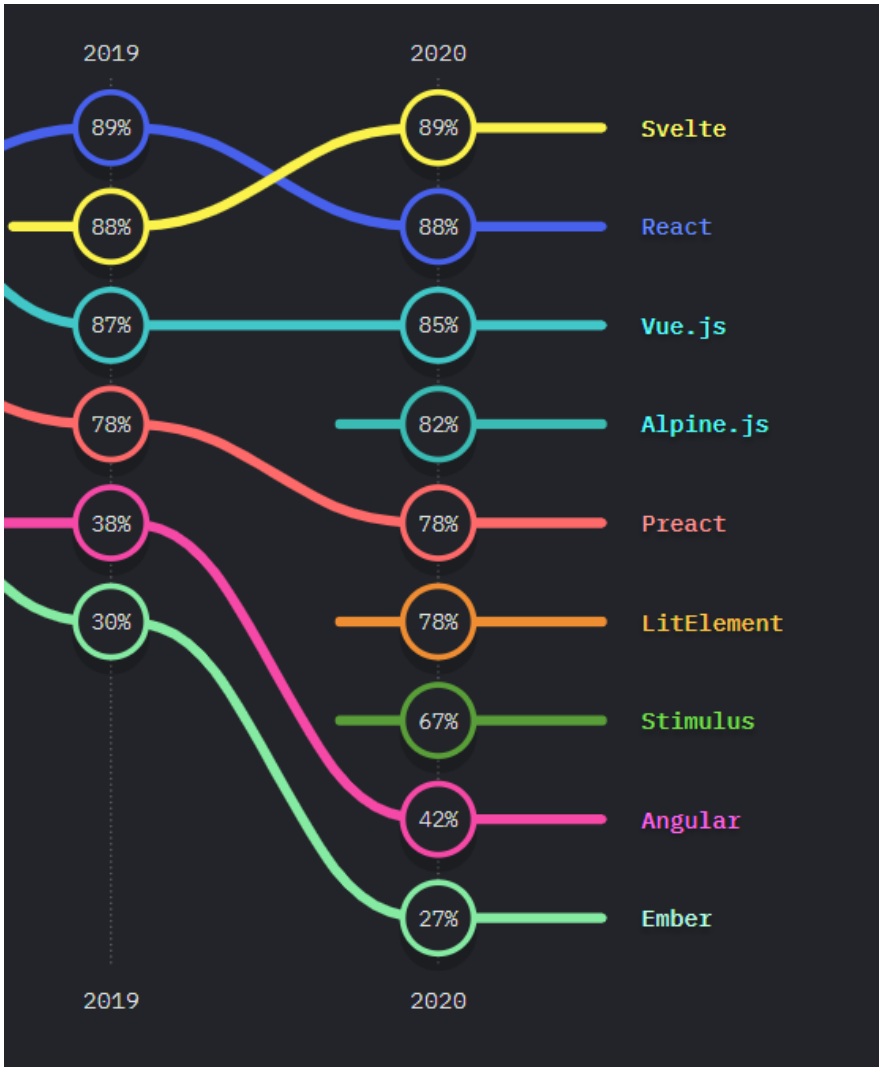
Kuten aiemmin tuli ilmi, Sveltessä pyritään minimalistiseen koodimäärään. Tämä näkyy myös Svelte-projektin rakenteessa. Vuen tavoin, Svelte antaa vain vähän valmiita tiedostoja. Kuten kuvi-osta 7 näkee, Svelte-projektissa ei ole juurikaan valmiiksi annettua kansiorakennetta. Komponentit ovat .svelte-päätteisiä tiedostoja, joissa on Vuen tavoin kaikki komponentin rakenteet samassa tie-dostossa. HTML-osio on kuvion 7 esimerkissä <main> tagien sisällä, mutta mikä tahansa HTML-tagi toimii yhtä lailla. JavaScript-osuus on <script> tagien sisällä ja CSS-osio on <style> tagien sisällä. (Component format, n/d.)




Kuvio 7. Svelte-projektin rakenne.

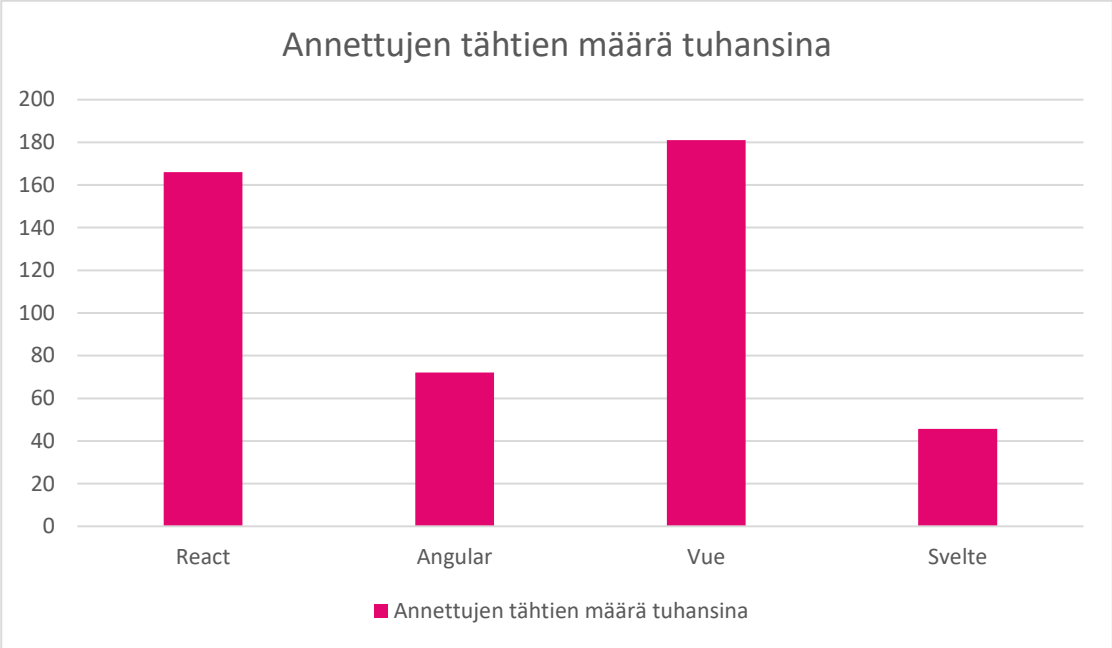
## 4.4 Käyttökokemuksia eri frameworkeistä

State of JS-sivusto selvittää kehittäjien tyytyväisyyttä eri frameworkeihin ja Svelte on ollut sivuston vertailussa mukana vuodesta 2019 lähtien. Ensimmäisenä vuotenaan Svelte oli listalla toisena. React oli tyytyväisyydessä ensimmäisenä ja Vue kolmas. Angular oli viides. Vuonna 2020 Svelte oli noussut tyytyväisyydessä ensimmäiseksi tyytyväisyysprosentilla 89 %. React oli toisena (88 %), Vue yhä kolmantena 85 %. Angular oli tippunut listalla kahdeksanneksi (42 %) (kuvio 8). (Greif & Benitte, 2019.)



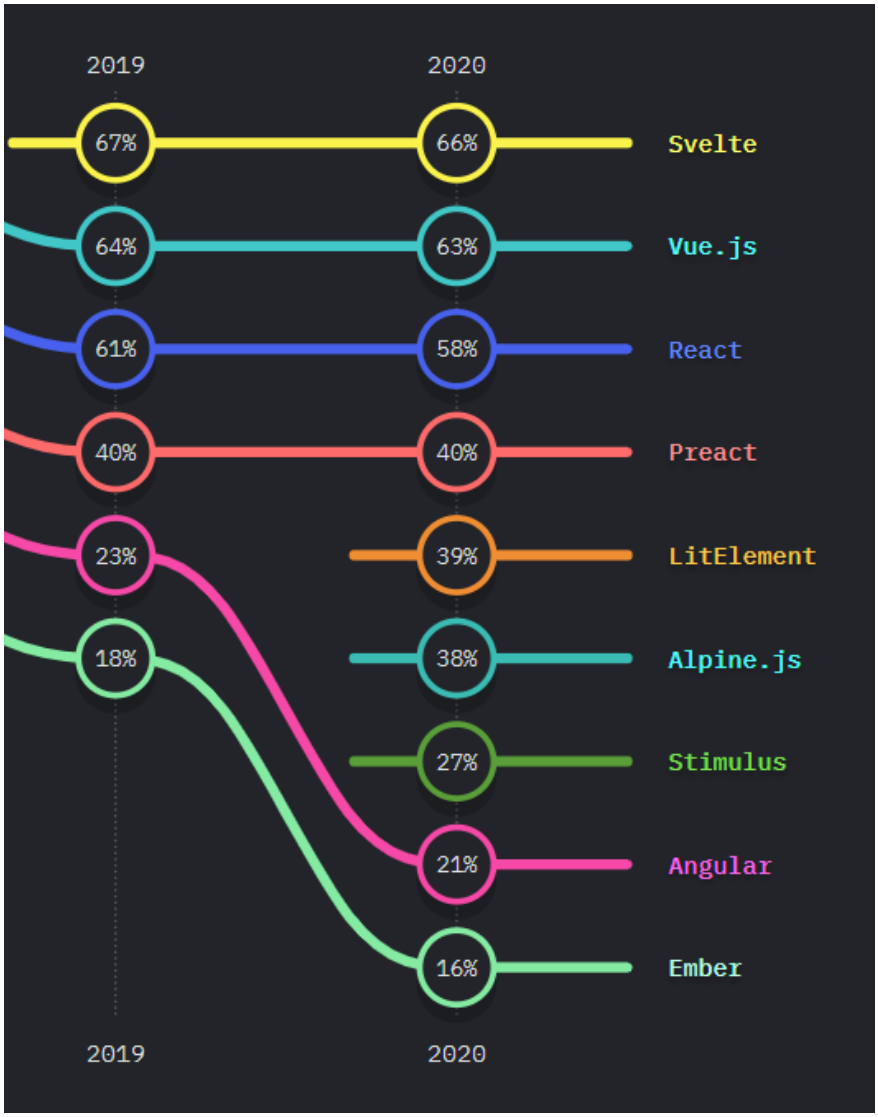

Kuvio 8. Kehittäjien tyytyväisyys frontend frameworkeihin vuosina 2019-2020 (Greif & Benitte, 2020).

Toinen mistä huomaa, että Svelte on otettu hyvin vastaan, on GitHub-yhteisön antamat tähtimäärät. Kuviosta 9 näkee, että Vue on GitHubissa kehittäjien suosiossa ja kuinka Svelte hätyyttelee pian Angularia. (facebook/react, angular/angular, vuejs/vue & sveltejs/svelte, 2020.)




Kuvio 9. Frontend frameworkien saamat tähdet Github-yhteisössä maaliskuussa 2021 (facebook/react, angular/angular, vuejs/vue & sveltejs/svelte, 2021).

State of JS-sivusto listaa Svelten vuonna 2020 kiinnostavimmaksi JavaScript frameworkiksi. Toisena oli Vue, kolmantena React ja Angular kahdeksas (kuvio 10). Vuonna 2019 Svelte pääsi listalle ensimmäisen kerran ja nousi kiinnostavimmaksi frameworkiksi, pitäen paikkansa myös seuraavana vuonna. Samalta sivulta näkee myös, että Svelte on myös yleisiä frameworkejä paljon huonommin tunnettu. Siinä missä sekä React, Angular ja Vue saivat 100 % tunnettavuuden, Svelten tunnettavuus oli 86 %. (Greif & Benitte, 2020.)




Kuvio 10. Kehittäjien kiinnostus frontend frameworkkeja kohtaan vuosina 2019-2020 (Greif & Benitte, 2020).

Schwarzmüller (2019) toi videollaan esille, että vanhemmilla frameworkeillä kuten Reactilla ja Angularilla on jo hyvin vakiintunut paikka yrityksien sovelluskehityksessä. Koska Svelte on vielä niin uusi framework, se ei ole vielä lyönyt läpi yritysmaailmassa samalla tavalla. (Schwarzmüller, 2019). Tämä sama näkyi myös State of JS-sivustolla, jossa Svelten käyttöprosentti on 15 %, kun se on Reactilla 80 %, Angularilla 56 % ja Vuella 49 % (Greif & Benitte, 2020).

# 5 Tulokset ja johtopäätökset

## 5.1 Frontend-perusteet -opintojakson kyselyjen tulokset

### Lähtötasokyselyn tulokset

Lähtötasokyselyyn vastasi 40 opiskelijaa. 93 % vastaajista oli ensimmäisen vuoden opiskelijoita. Kenelläkään vastaajista ei ollut aiempaa kokemusta frameworkeistä ja 15 % vastaajista kertoi, että heillä on joitain pohjatietoja frontend-kehittämisestä. Pohjatietoja kysyessä vastaajat saivat itse kuvailla omia pohjatietojaan. Kaikki kokivat osaavansa HTML:n ja CSS:n perusteet joko perustasolla tai hyvin. JavaScriptin suhteen ei oltu niin varmoja. Vastaajista 13 eli noin kolmasosa arvioi JavaScript osaamisensa perustasoa huonommaksi.

Opinnäytetyön kannalta haluttiin erityisesti seurata, kuinka opiskelijoiden kiinnostus frontend-kehittämiseen kehittyy opintojakson aikana. Kuten opinnäytetyön tarkoituksessa kuvattiin, Frontend-perusteet opintojakson toivotaan helpottavan opiskelijoiden hakeutumista ohjelmistokehityksen pariin. Tästä syystä yksi lähtötasokyselyn kysymyksistä koski vastaajien kiinnostusta ohjelmistokehittämistä kohtaan. Opintojakson alussa 70 % vastaajista voisi kuvitella työskentelevänsä ohjelmistokehittäjänä. Loput eivät joko näe itseään ohjelmistokehittäjinä tai eivät osanneet sanoa (kuvio 11). 25 % vastasi, ettei näe ohjelmistokehitystä vaihtoehtona itselleen ja 5 % ei osannut sanoa.


Kuvio 11. Lähtötasokyselyn jakautuminen kysyttäessä voisivatko he kuvitella työskentelevänsä ohjelmistokehittäjänä.

**Loppukyselyn tulokset**

Loppukyselyyn vastanneiden opiskelijoiden määrä oli selvästi pienempi kuin lähtötasokyselyyn vastanneiden määrä. Loppukyselyyn vastasi 21 opiskelijaa. Heistä 20 eli 95 % koki oppineensa frontend-kehittämisen perusteet opintojakson aikana. Opiskelijat saivat arvioida oman motivaationsa opintojaksolle, opintojakson haastavuuden ja oman Svelten käyttönsä asteikolla 1–5, joista 1 on huonoin ja 5 on paras. Arvosanojen keskiarvot ja mediaanit on koottu taulukkoon 1.

Taulukko 1. Loppukyselystä kootut arvosanat opiskelijoiden motivaatiosta opintojaksolle, opintojakson haastavuudesta ja opiskelijoiden Svelten käyttöön.

| | **Keskiarvo** | **Mediaani** |
|---|---|---|
| Opiskelijoiden motivaatio | 3,62 | 4 |
| Opintojakson haastavuus | 3,62 | 4 |
| Svelten käyttö | 3,19 | 3 |

Kun kysyttiin mikä opintojaksolla tuntui haastavalta, osa vastasi, ettei kokenut minkään tietyn asian olleen haastavaa ja että opintojakson asiat olivat sisäistettävissä. Niistä vastauksissa, joissa haasteita löytyi, toistuvia teemoja olivat:

- Tiedon liikuttaminen kuten propsien ja storen käyttö ja komponenttien välinen tiedonsiirto
- Komponenttien käyttö kuten miten ne toimivat yhteen ja mitä sisältöä laitetaan millekin komponentille
- JavaScriptin käyttö
- Vähäiset ohjeet netistä, jolloin soveltaminen tai omien kokeiluiden teko tuntui vaikealta
- Vian diagnosoiminen silloin kun ainoa tieto virheestä on se, ettei sovellus toimi

Koska haluttiin seurata, kuinka opiskelijoiden kiinnostus frontend-kehittämiseen kehittyy opintojakson aikana, myös loppukyselyssä toistettiin sama kysymys kuin lähtötasokyselyssä (kuvio 12). Opintojakson lopussa 57 % voisi yhä kuvitella ohjelmistokehitystä tulevaksi uravaihtoehdoksi. Määrä oli hieman tippunut lähtötasokyselystä. ”En osaa sanoa”-vastauksien osuus oli kasvanut ja kaikista vastauksista niitä oli lopussa 33 %. 10 % ei ole kiinnostunut ohjelmistokehityksestä. Tuloksiin voi vaikuttaa se, että vastaajamäärä on eri kuin lähtötasokyselyssä. Ei voida tietää, onko opiskelijoiden mielipide oikeasti muuttunut opintojakson aikana, vai eikö lähtötasokyselyssä tietyllä

tavalla vastanneet ole enää vastanneet niin aktiivisesti loppukyselyssä. Noin kolmasosa jättäytyi pois opintojaksolta, mikä selittää vähentyneen vastaajamäärän.


Kuvio 12. Opintojakson lopussa kysyttäessä ohjelmistokehittämisesta uravaihtoehtona.

## 5.2 Teemahaastattelujen tulokset

**Lähtökohdat kurssille ja frontendin perusteiden oppiminen**

Kaikilla haastateltavilla oli samanlaiset lähtökohdat ennen peruskurssin alkua. Jokainen oli käynyt jonkin ohjelmoinnin peruskurssin sekä HTML:n ja CSS:n läpikäyvän kurssin. Kaikki haastateltavat kokivat olleensa aloittelijoita oman peruskurssinsa alussa ja motivoituneita kurssin suorittamiseen, koska tiesivät, että kurssista on hyötyä heidän tulevaisuutensa ja työelämän kannalta. Angularia opiskelleet (ryhmä A) olivat hieman varautuneempia omaan peruskurssiinsa, koska olivat kuulleet ennestään, että kurssi olisi haastava. Ryhmästä A vain osa oli tutustunut JavaScriptiin ennen peruskurssin alkua. Nämä henkilöt olivat oppineet ohjelmoinnin perusteet C#:lla. Heille kuitenkin esiteltiin nopeasti JavaScript ennen frontend-perusteiden opettamista.

Opittuaan perusteet, opiskelijoiden motivaatio kurssia kohtaan, joko kasvoi tai laski sen mukaan, kuinka vaikeiksi he olivat perusteet kokeneet. Kaikki haastateltavat mainitsivat, että etenkin sovelluskehittämisen hahmottaminen isossa kuvassa oli haastavinta ja aikaa vievintä. Osa koki, että he

eivät ehtineet hahmottaa kaikkea isossa kuvassa ennen kuin aiheessa siirryttiin jo eteenpäin. Se aiheutti paniikkia ja tunteita asioiden sekavuudesta. Haastateltavista ryhmä A:n keskuudessa oli enemmän negatiivisia tuntemuksia. Tästä syystä haastateltavista vain Angularin puolella oli henkilöitä, jotka kokivat, etteivät innostuneet frontend-kehittämisestä heti perusteiden myötä. Into frontend-kehittämiseen saattoi löytyä vasta myöhemmässä vaiheessa kurssia tai kurssin jälkeen. Kaikki haastateltavat kokivat, että frontendin perusteet ovat opittavissa. Se motivoi monia oppimaan haasteista huolimatta.

**Angularin ja Svelten oppiminen**

Selkeimmät erot oppimiskokemuksissa tulivat, kun alettiin puhua frameworkin oppimisesta. Svelteä opiskelleet (ryhmä S) kokivat pohjatietojensa riittäviksi, kun taas ryhmä A:ssa koettiin, että heidän pohjatietonsa riittivät vain alkuun. Esimerkiksi haastateltavat kokivat Angularin projektirakenteen sekavaksi ja TypeScriptin käytön vaikeaksi. Tästä syystä Angularin logiikka ei tuntunut aukeavan helposti haastateltaville. Moni kertoi haastatteluryhmä A:Sta, että olivat ymmärtäneet Angularin logiikan vasta kurssin jälkeen. Sen sijaan ryhmässä S kaikki olivat hahmottaneet Svelten logiikan peruskurssin aikana. Kun puhuttiin frameworkien intuitiivisuudesta, frameworkin logiikan hahmottamisella oli suuri vaikutus siihen, kokivatko haastateltavat frameworkin intuitiiviseksi.

Ryhmä S:n haastateltavat löysivät nopeammin tuttuja piirteitä Svelten logiikan ja oman logiikkansa väliltä. Ryhmän S haastateltavat sanoivat, että joutuivat kyllä näkemään vaivaa Svelten oppimisessa ja ymmärtämisessä. Tämä käy yhteen loppukyselyn tuloksen kanssa, jossa Svelten osaamisen arvosanan keskiarvoksi arvioitiin 3,19/5 (taulukko 1). Etenkin virhetilanteissa, jossa toimimattomuus johtuu merkkivirheestä, he olisivat kaivanneet Svelten ohjelmointiympäristöltä enemmän apua tai vihjeitä siitä missä vika on. Ryhmässä S koettiin muuten Svelteen liittyvän tuen ja dokumentaation riittäviksi perusteita opiskellessaan. Vain paikoin Svelteen liittyvä tuki ei riittänyt ja opiskelija ei löytänyt suoraan Svelteen liittyvää ohjetta jonkin ongelman ratkaisuun. Kaiken kaikkiaan Svelte jätti positiivisia tunteita haastateltaviin, vaikka asioiden oppimiseen kaivattiinkin enemmän aikaa.

Ryhmästä A kaikki ilmaisivat, ettei Angular ole intuitiivinen tai helppo oppia. Osa haastateltavista kertoi, että vaikka he oppivat Angularin lopulta hyvin, he tiedostavat sen vaativuuden. Oppiminen

vei kauan aikaa ja logiikka avautui pakottamalla vain pienissä palasissa kerrallaan. Angularia kuvailtiin muun muassa julmaksi ja vaativaksi. Jos tarvitsi tukea tai dokumentaatiota, Angularista sitä oli jopa liikaa. Angularista on paljon versioita, joista vanhimmat eivät käy enää yhteen uusimpien kanssa, joten uusimman tiedon etsiminen vei paljon aikaa. Angularin tarjoamia omia tutoriaaleja ei koettu aloittelijaystävällisiksi.

**Ristiinvertailu Sveltelle**

Kun Svelteä esiteltiin ryhmälle A vastaanotto oli positiivinen. Svelte koettiin mielenkiintoiseksi ja loogiseksi. Siitä tunnisti heti frameworkin piirteet ja sen syntaksit olivat ymmärrettäviä. Svelten projektirakenne tuntui selkeältä, koska siinä oli vähemmän tiedostoja kuin Angularin projektirakenteessa. Haastateltavat ilmaisivat, että mieluummin alussa vähän tiedostoja, joista tietää mitä ne tekevät, kuin paljon toimintoja, joita ei ymmärrä tai osaa käyttää aloittelijana. Kompaktius tuli esille muun muassa komponentti rakennetta vertaillessa. Siinä missä Angularissa on neljä tiedostoa per komponentti, Sveltessä on vain yksi.

Svelte sai kritiikkiä siitä, että tiettyjen käyttöliittymäelementtien kanssa joutuu näkemään enemmän vaivaa kuin Angularissa. Angularissa on helppo hyödyntää esimerkiksi Angular Materialia, joka nopeuttaa ja selkeyttää näiden käyttöliittymäelementtien tekoa ja etenkin muotoilua. Lisäksi haastatteluissa nousi esille, että jos voi ja pystyy oppimaan Angularin ensimmäisenä frontend-teknologiana, se on työelämän kannalta hyödyllisempi kuin Svelte. Mutta jos Angularin kanssa oli haasteita, haastateltavat olisivat mielellään halunneet oppia perusteet ensin Sveltellä ja vaihtaa sitten Angulariin. 2/3 haastateltavista koki Svelten helpommaksi ja olisi halunnut aloittaa sovelluskehitysopintonsa sillä.

**Ristiinvertailu Angularille**

Kun Angular esiteltiin ryhmälle S, mielikuva oli sekavuus. Pienen tutustumisen jälkeen, frameworkien piirteet alkoivat hahmottua Angularista. Kommenteista kävi ilmi, että haastateltavat uskovat Angularissa olevan enemmän haltuun otettavaa, mutta ei se mahdottomalta vaikuttanut. Haastateltavat uskoivat, että Angular on ennemmin pidemmälle kehittyneen ohjelmoijan teknologia kuin aloittelijan. Tästä syystä Angularia ei olisi halunnut ensimmäiseksi frontend-teknologiakseen kukaan haastateltavista. Kaikki pitivät Svelteä myös paljon helpompana kuin Angularia.

Angular sai paljon kritiikkiä tiedostojen määrässä. Se, että komponentti on pilkottu neljään pienempään tiedostoon, ei tuntunut datan käsittelyn kannalta niin selkeältä kuin Svelten yksitiedostoinen komponentti. Komponentin pilkkominen toimii kuulemma isommissa projekteissa paremmin, kun yhdessä näkymässä on vähemmän koodia, mutta se ei ole ajankohtaista vielä aloittelijatasolla. Angularin toimintojen, kuten *ngIf ja data binding, ei koettu erottuvan niin hyvin kuin Svelten. Ne katosivat helposti HTML:n sekaan. Myös ajatus TypeScriptin käytöstä kuitenkin hirvitti haastateltavia.

## 5.3 Täydennetty vertailukehikko

Taulukko 2. Angularin ja Svelten vertailukehikko täytettynä.

| | **Svelte** | **Angular** |
|---|---|---|
| Käyttöönotto | X | |
| Dokumentaatio ja tuki | | X |
| Syntaksi | X | |
| Uramahdollisuudet | | X |
| Intuitiivisuus | X | |
| Pohjatietojen hyödyntäminen | X | |

Taulukosta 2 voi nähdä täydennetyn vertailukehikon. Arvo X on annettu sille frameworkille, jolla kyseessä oleva ominaisuus on vahvempi tai parempi. Ominaisuudet ovat samat, joiden valinta perusteltiin luvussa 2.5. Perustelut valinnoille käydään läpi seuraavaksi.

**Käyttöönotto**

Sekä Angularin että Svelten ympäristö vaatii Node.js:n asennuksen. Jotta Angular projekti saadaan alustettua ja otettua käyttöön, vaaditaan Angular CLI:n asennus (Setting up the local environment and workspace, N/d). Tämän jälkeen voitiin luoda Angular-projekti. Sen luomiseen kului noin kuusi minuuttia. Ajanotto pysäytettiin siksi aikaa, kun vastattiin lisäkysymyksiin. Lisäkysymyksinä olivat, halutaanko lisätä routing-ominaisuus, ja mitä stylesheet formatia projektissa halutaan käyttää. Svelten ympäristö vaatii terminaalin hallintaohjelman, kuten Git:n, asennuksen ja sen jälkeen voitiin luoda Svelte-projekti hyödyntäen Svelten omaa templatea (Getting started with Svelte, N/d). Projektin luomiseen kului viisi sekuntia samalla tietokoneella, joten Svelte oli tässä parempi nopeutensa takia.

**Dokumentaatio ja tuki**

Sekä Angular, että Svelte tarjoavat dokumentaatiosivut kehittäjille, vaikka molemmissa on parantamisen varaa. Teemahaastatteluissa nousi esille, ettei Angularin tutoriaali ole aloittelijalle helppo. Loppukyselyssä taas mainittiin, että koska Sveltestä löytyy vähän ohjeita, omat oppitunneista poikkeavat kokeilut eivät onnistuneet. Dokumentaation määrästä kertookin paljon se, että kun haki 19.4.2021 Googlesta hakusanalla ”Angular tutorial”, hakutuloksia saatiin noin 71 100 000 kpl. Hakusanalla ”Svelte tutorial” saatiin noin 724 000 tulosta. Tämä on selvä osoitus siitä, että Angularilla on pidemmän historiansa ansiosta laajat kehittäjäyhteisöt.

Vaikka Angularin hakutulos on selvästi parempi kuin Svelten, kaikki tulokset eivät ole enää ajan tasalla. Syynä tähän on Angularin useat versiot. Angularin uusin version on 11 (Angular versioning and releases, N/d) ja Svelten 3 (sveltejs/svelte, N/d). Tästä syystä Svelte on ehtinyt muuttua vähemmän ja moni sen luoduista materiaaleista on yhä käyttökelpoisia. Lisäksi Angular julkaisee uusia versioita tiiviimmin kuin Svelte. Esimerkiksi vuoden 2020 aikana Angular julkaisi versiot 9, 10 ja 11 (Angular versioning and releases, N/d). Svelten viimeisin versio on vuodelta 2019 ja sitä edellinen versio julkaistiin vuotta aikaisemmin vuonna 2018 (sveltejs/svelte, N/d). Svelten suppea tuki nousee kuitenkin esille, kun puhutaan Svelten heikkouksista (Dhaduk, 2021).

Tämä ominaisuus menee lopulta selvästi Angularille. Sen lisäksi, että Angularille on tarjolla laajemmin tukea, myös teemahaastatteluissa ja loppukyselyssä nousi esille, kuinka huonosti Svelte auttoi kehittäjää virhetilanteissa. Muun muassa virhetilanteissa auttavat lintterit eivät toimineet Sveltessä niin hyvin mitä ne toimivat Angularissa.

**Syntaksi**

Teemahaastatteluissa nousi selvästi esille, kuinka Svelte tuntui selkeämmältä niin projektirakenteensa suhteen kuin syntaksinsa osalta. Svelten omat koodilohkot olivat selkeitä sisäistää ja erottuivat hyvin HTML-koodista. Angularin vastaavat ominaisuudet olivat havaittavissa koodista, mutta ne eivät olleet yhtä selkeitä kuin Svelten. Lisäksi kuten jo frameworkejä vertaillessa kävi ilmi, Sveltessä kirjoitetaan selvästi vähemmän koodia kuin muissa frameworkeissä (Schae, 2020). Näistä syistä hyvä syntaksi on selvästi Svelten vahvuus.

**Uramahdollisuudet**

Kuten Schwarzmüller (2019) kertoi videollaan, Svelte on uutuutensa takia vielä hyvin harvinainen yritysmaailmassa. Aiemmin tehdyssä frameworkien vertailussa Angular taas osoitti olevansa hyvin vakiintunut yrityskäytössä. Käytännön esimerkki tästä on 22.4.2021. tehty työnhakuvertailu, jossa Oikotie.fi-sivuston työnhakuun laitettiin hakusanaksi Svelte ja saatiin yksi täsmäävä hakutulos. Sen sijaan hakusanalla Angular saatiin 58 hakutulosta. Myös haastatteluissa kävi ilmi, että Angularilla perusteensa oppineista osa toi ilmi, että kokee Angularin työelämälähtöisyyden suurena plussana. Näistä syistä Angular on selvästi ylivoimainen uramahdollisuuksien suhteen.

**Intuitiivisuus**

Osa haastateltavista ilmaisi suoraan, ettei Angular ole intuitiivinen käyttää tai oppia. Angularilla perusteensa opiskelleet opiskelijat kertoivat, että Angularin logiikka oli auennut heille kaikille vasta opintojakson jälkeen. Osalle logiikka ei ollut vielä haastatteluhetkenäkään selvinnyt. Svelteä ensimmäisenä frameworkinä oppineet eivät kokeneet päässeensä helpolla Svelten kanssa, mutta heille kaikille frameworkin logiikka oli lopulta auennut opintojakson aikana.

Myös ulkopuoliset lähteet pitävät Svelteä hyvin intuitiivisena. Esimerkiksi kirjailija ja ohjelmistokehittäjä Houssein Djirdeh pitää Svelteä erittäin intuitiivisena frameworkinä. Hänellä on kokemusta niin Angularista kuin Sveltestä ja hän kertoo podcast-haastattelussa, että olisi itse halunnut oppia

frameworkien perusteet Sveltellä. Djirdeh pitää Svelten kehittäjäkokemusta hyvänä. (Performance on the web with Houssein Djirdeh, 2020.) Samoin Syntax.fm-podcastin Wes Bos ja Scott Tolinski pitävät Svelteä vaivattomana muihin frameworkeihin verrattuna (Hasty Treat - Wes & Scott Look At Svelte 3, 2019).

**Pohjatietojen hyödyntäminen**

Kuten frameworkejä vertaillessa kävi ilmi, Angularissa JavaScript osaamista tulee soveltaa TypeScriptiin (Angular vs React vs Vue vs Svelte: A Comparison of the Top Frontend Frameworks, 2020.), joten se vaatii enemmän pohjatietoja kuin Svelte. Svelte hyödyntää suoraan JavaScriptiä, jolloin myös JavaScript osaaminen tai osaamattomuus näkyy suoraan frameworkiä kehittäessä. TypeScriptin käyttö aloittelijatasolla koettiin myös teemahaastatteluissa negatiiviseksi asiaksi. Angularin ensimmäisenä frontend-teknologiana oppineet opiskelijat eivät kokeneet pohjatietojensa riittävän niin hyvin mitä Sveltellä oppineet kokivat.

HTML:n osalta Djirdeh tuo podcast-haastattelussaan esille, että Svelten HTML-keskeisyys parantaa kehittäjäkokemusta selvästi (Performance on the web with Houssein Djirdeh, 2020). Saman huomion tekevät myös Bos ja Tolinski podcastissään ja kehuvat Svelteä siitä, kuinka siinä voi suoraan hyödyntää olemassa olevia pohjatietoja (Hasty Treat - Wes & Scott Look At Svelte 3, 2019). Sveltellä pääsee siis paremmin hyödyntämään pohjatietojaan kuin Angularilla.

# 6 Pohdinta

## 6.1 Johtopäätökset

Vertailukehikon helppouden piirteillä tuloksista voidaan vetää yhteen, että Svelte soveltuu Angularia paremmin ensimmäiseksi frontend-teknologiaksi aloittelevalle ohjelmoijalle. Vasta-alkajat kokevat Svelten yleisesti selkeämmäksi ja sen takia helpommaksi frameworkiksi. Tutkimuksen tilaajan hypoteesi Svelten helppoudesta piti paikkansa tässä tapaustutkimuksessa.

Angularilla on omat vahvuutensa, mutta niitä pääsee paremmin hyödyntämään, kun on jo kehittyneempi ohjelmoija. Toisaalta edistyneemmät kehittäjät pitävät myös Sveltestä enemmän kuin An-

gularista. Tämä käy ilmi, kun verrattiin frameworkien käyttäjäkokemuksia. State of JS-sivuston tilastojen mukaan Svelteen ollaan tyytyväisempiä kuin Angulariin. Saman sivuston tilastojen mukaan Svelte myös kiinnostaa kehittäjiä paljon enemmän kuin Angular. (Greif & Benitte, 2019.)

Opinnäytetyön tavoitteet saavutettiin. Tutkimuskysymyksiin vastattiin ja tilaajan hypoteesi saatiin varmistettua. Opiskelijat kokivat Svelten sopivaksi ensimmäiseksi frameworkiksi ja kokivat oppineensa frontendin perusteet. Kokonaan Svelte ei poistanut frontendin oppimisen haastavuutta ja tällä havaintoyksiköllä opintojakso horjutti hieman suoraa mielenkiintoa ohjelmistokehitystä kohtaan. Loppukyselystä ilmeni, että opintojakson jälkeen myös niiden opiskelijoiden määrä, jotka vastasivat, ettei heitä kiinnosta ohjelmistokehitys uravaihtoehtona, oli pienentynyt. Lähtötasokyselyn ja loppukyselyn tuloksia ei kuitenkaan voi vertailla suoraan keskenään, koska loppukyselyssä vastaajia on lähes puolet vähemmän kuin lähtötasokyselyssä.

## 6.2 Kehittämis- ja jatkotutkimusehdotukset

Tutkimuksen selvin kehittämiskohde on teemahaastatteluissa. Angularia opiskelleet haastateltavat olivat kaikki suorittaneet opintojakson, jossa opetettiin kerralla sekä JavaScript sekä frontend- ja backend-perusteet. Tällaisen ison kokonaisuuden oppimiskokemus on erilainen kuin pelkän frontend-perusteiden. JAMKilla on myös tarjota opintojakso, jossa opiskellaan vain frontendin perusteet Angularilla. Tämän opintojakson opiskelijoiden lähtökohdat olisivat saattaneet muistuttaa enemmän Svelteä opiskelevien opiskelijoiden lähtökohtia ja tästä syystä he olisivat saattaneet sopia paremmin tutkimuksen haastateltaviksi Angularin osalta.

Tutkimuksen aineistonkeruumenetelmissä otettiin kantaa Svelten helppouteen vain suhteessa Angulariin. Se ei kuitenkaan tarkoita vielä, että Svelte on optimaalisin framework ensimmäiseksi frontend-teknologiaksi. Jatkotutkimuskohteeksi sopisi hyvin myös muiden frameworkien vastaava tutkiminen. Esimerkiksi Vue näytti frameworkien projekteja verratessa hyvin samanlaiselta kuin Svelte ja tunnettavuutensa puolesta React on myös potentiaalinen tutkittava.

## 6.3 Tutkimuksen eettisyys ja luotettavuus

Opinnäytetyön eettisyyden takaamiseksi tutkimuksessa pyrittiin etenemään oikeaoppisesti noudattaa tutkimuseettisiä ohjeistuksia. Koska haastateltavat ja seurattavan opintojakson opiskelijat

ovat JAMKin opiskelijoita, täytyi tutkimusta varten laatia JAMKille tutkimuslupahakemus. Lupahakemus hyväksyttiin. Opinnäytetyössä ei kerätty henkilötietoja haastateltavilta eikä haastateltavia ei voi tunnistaa opinnäytetyöstä. Opiskelijat pidettiin informoituina siitä mihin käyttöön heidän antamiaan haastatteluja tai täyttämiään lomakkeita käytetään. Kerättyä aineistoa varten luotiin aineistonhallintasuunnitelma.

Aineistonhankintamenetelmänä haastattelut ovat aina haaste. Haastateltavia on tässä tutkimuksessa hyvin rajallinen määrä, mikä vaikuttaa tutkimuksen luotettavuuteen. Ei voida olla varmoja, että jos haastattelut olisi tehty eri henkilöille, tulokset olisivat samanlaiset. Tulokset pohjautuvat nyt näiden tiettyjen opiskelijoiden mielipiteisiin ja kokemuksiin, mutta niitä ei voi yleistää koskemaan kaikkia opiskelijoita. Tutkimus on kuitenkin pyritty toteuttamaan niin, että se olisi tarvittaessa toistettavissa. Mahdollisten toistojen myötä mielipiteiden määrä kasvaa ja luotettavuus paranisi.

Tutkimuksessa haluttiin saada kommentteja myös Angularia ensimmäisenä frontend-teknologianaan opiskelleilta opiskelijoilta. Heidän kohdallaan asetelma on kuitenkin hieman erilainen sillä he ovat suorittaneet oman peruskurssinsa jo reilu vuosi ennen haastattelija. Sen sijaan Svelteä opiskelleita haastateltiin heti kun peruskurssin kontaktitunnit olivat päättyneet. Osa Angularia opiskelleista haastateltavista oli ehtinyt kehittää taitojaan peruskurssin jälkeen tai tutustua muihin frameworkeihin. Tästä syystä heidän suhtautumisensa Angulariin on saattanut muuttua siitä mitä se oli heti peruskurssin jälkeen. Tästä syystä suhtautuminen muihin frameworkeihin saattaa olla jo erilaista.

Lähteitä on monipuolisesti, mutta kaikki eivät ole tieteellisesti katsottuna luotettavia. Aiempien Svelte-tutkimusten vähyys pakotti etsimään lähteitä myös tieteellisten tutkimusten ulkopuolelta. Tästä syystä lähdemateriaalissa on aineistoa, jonka ei voida olettaa olevan täysin riippumatonta tai puolueetonta. Tutkimuksessa oltiin kiinnostuneita mielipiteistä ja niitä pyrittiin löytämään mahdollisimman monipuolisesti, mutta nämä mielipiteet ovat aina subjektiivisia.

# Lähteet

Abi-Nakhoul, N. N/d. Most Fun Programming Language to learn. Viitattu 22.4.2021. https://www.lighthouselabs.ca/en/blog/most-fun-programming-language-to-learn.

Aderinokun, I. 2018. Understanding the Virtual DOM. 24.12.2020. Viitattu 17.12.2020. https://bitsofco.de/understanding-the-virtual-dom/.

angular/angular. Github.com. 2021. Viitattu 30.3.2021. https://github.com/angular/angular/stargazers.

Angular versioning and releases. N/d. Viitattu 19.4.2021. https://angular.io/guide/releases.

Angular vs React vs Vue vs Svelte: A Comparison of the Top Frontend Frameworks. 8.10.2020. Viitattu 9.3.2021. https://www.icicletech.com/blog/angular-react-vue-svelte-comparison.

Baer, E. n/d. What React Is and Why It Matters. n/d. Viitattu 9.3. 2021. https://www.oreilly.com/library/view/what-react-is/9781491996744/ch01.html.

Bhat, S. 2019. 6 Reasons Why Python is Beginner Friendly. Viitattu 22.4.2021. https://medium.com/@sangaddeshreevatsa/6-reasons-why-python-is-beginner-friendly-e63c6eddf765.

Component format. n/d. Viitattu 29.3.2021. https://svelte.dev/docs.

Components and Props. n/d. Viitattu 29.3.2021. https://reactjs.org/docs/components-and-props.html.

Dhaduk, H. 2021. Best Frontend Frameworks of 2021 for Web Development. 5.1.2021. Viitattu 20.4.2021. https://www.simform.com/best-frontend-frameworks/.

facebook/react. Github.com. 2021. Viitattu 30.3.2021. https://github.com/facebook/react/stargazers.

Gackenheimer, C. 2015. Introduction to React. Apress. Viitattu 9.3.2021. https://link.springer.com/chapter/10.1007/978-1-4842-1245-5_1.

Getting started with Svelte. N/d. Viitattu 19.4.2021. https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Svelte_getting_started.

Greif, S & Benitte, R. 2020. Front-end frameworks. N/d. Viitattu 15.1.2021. https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/.

Harris, R. 2016. Frameworks without the framework: why didn't we think of this sooner? 26.11.2016. Viitattu 21.12.2020. https://svelte.dev/blog/frameworks-without-the-framework.

Harris, R. 2018. Virtual DOM is pure overhead. 27.12.2018. Viitattu 17.12.2020. https://svelte.dev/blog/virtual-dom-is-pure-overhead#Where_does_the_overhead_come_from.

Harris, R. 2020. Why Svelte? Frontend Masters–sivustolla. Video. Julkaistu 15.7.2020. Viitattu 21.12.2020. https://frontendmasters.com/courses/svelte/why-svelte/.

Harris, R. 2019. Write less code. 20.4.2019. Viitattu 14.1.2021. https://svelte.dev/blog/write-less-code.

Hasty Treat - Wes & Scott Look At Svelte 3. 2019. Jakso 173. Podcast-ohjelma, julkaistu 26.8.2019. Syntax.fm -verkkosivu. Viitattu 20.4.2021. https://toppodcast.com/podcast_feeds/svelte-radio/.

helppo. N/d. Virtuaalinen sanakirja. Viitattu 19.4.2021. https://www.suomisanakirja.fi/helppo.

Introduction. n/d. Viitattu 29.3.2021. https://vuejs.org/v2/guide/.

intuitiivinen. N/d. Virtuaalinen sanakirja. Viitattu 20.4.2021. https://www.suomisanakirja.fi/intuitiivinen.

JavaScript HTML DOM. Viitattu 17.12.2020 https://www.w3schools.com/js/js_htmldom.asp.

Järvinen, P & Järvinen, A. 2000. Tutkimustyön metodeista. Tampere: Opinpajan kirja.

Kallinen, Timo & Kinnunen, Taina. N/d. Laadullisen tutkimuksen verkkokäsikirja. Tampere: Yhteiskuntatieteellinen tietoarkisto. Viitattu 15.4.2021. https://www.fsd.tuni.fi/fi/palvelut/menetelmaopetus/kvali/laadullisen-tutkimuksen-prosessi/.

Kananen, J. 2008. Kvali. Kvalitatiivisen tutkimuksen teoria ja käytänteet. Jyväskylän ammattikorkeakoulun julkaisuja-sarja. Jyväskylä: Jyväskylä Yliopistopaino.

Korhonen, E. 2020. Koodaripulakyselyn päähavainnot. Viitattu 19.2.2021. https://www.koodaripula.com/2020/04/06/main-results.html.

Morris, S. N/d. Tech 101: What Is a JavaScript Framework? Here's Everything You Need to Know. N/d Viitattu 15.1.2021. https://skillcrush.com/blog/what-is-a-javascript-framework/.

Performance on the web with Houssein Djirdeh. 2020. Svelte Radio. Podcast-ohjelma, julkaistu 15.9.2020. Top podcast -verkkopalvelu. Viitattu 20.4.2021. https://toppodcast.com/podcast_feeds/svelte-radio/.

Potluck - CSS × Angular × Dev job preparation × Svelte × File organization × Gear × More!. 2019. Jakso 148. Podcast-ohjelma, julkaistu 29.5.2019. Syntax.fm -verkkosivu. Viitattu 20.4.2021. https://toppodcast.com/podcast_feeds/svelte-radio/.

Robie, J. 1998. What is the Document Object Model? 19.7.1998. Viitattu 17.12.2020. https://www.w3.org/TR/WD-DOM/introduction.html.

Robinson, S. N/d. Why Beginners Should Learn Python. Viitattu 22.4.2021. https://stackabuse.com/why-beginners-should-learn-python/.

Saarela-Kinnunen, M & Eskola, J. 2010. Tapaus ja tutkimus = tapaustutkimus? Julkaisussa Ikkunoita tutkimusmetodeihin 1. Metodin valinta ja aineiston keruu: virikkeitä aloittelevalle tutkijalle. Kolmas uudistettu ja täydennetty painos. Jyväskylä: PS-kustannus.

Sandeep, Kumar Partel. 2015. E-kirja. Learning Web Component Development. Packt Publishing. Viitattu 19.3.2021. https://scholar.google.com/

Schae, J. 2020. A RealWorld Comparison of Front-End Frameworks 2020. 9.3.2020. Viitattu 30.3.2021. https://medium.com/dailyjs/a-realworld-comparison-of-front-end-frameworks-2020-4e50655fe4c1.

Schwarzmüller, M. 2019. Video. Svelte vs React vs Angular vs Vue. YouTube-videopalvelu. Julkaistu 15.5.2019. Viitattu 27.12.2020. https://youtu.be/DZyWNS4fVE0.

Setting up the local environment and workspace. N/d. Viitattu 19.4.2021. https://angular.io/guide/setup-local.

Skólski, P. 2016. Single-page application vs. multiple-page application. 1.12.2016. Viitattu 9.1.2021. https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58.

sveltejs/svelte. Github.com. 2021. Viitattu 30.3.2021. https://github.com/sveltejs/svelte/stargazers.

sveltejs/svelte. Github.com. N/d. Viitattu 19.4.2021. https://github.com/sveltejs/svelte/releases.

Toivanen, A. 2020. Dokumenttioliomalli selainpohjaisena sovelluskehysten hahmonnuskohteena. Pro gradu. Helsingin yliopisto, Matemaattis-luonnontieteellinen tiedekunta. Viitattu 11.2.2021. http://urn.fi/URN:NBN:fi:hulib-202012084724.

vuejs/vue. Github.com. 2021. Viitattu 30.3.2021. https://github.com/vuejs/vue/stargazers.

Web Technologies Employed by Front-End Developers. 2018. Artikkeli Frontend Masters –sivustolla. Viitattu 27.12.2020. https://frontendmasters.com/books/front-end-handbook/2018/practice/tech-employed-by-fd.html.

What is a front-end developer? 2018. Viitattu 27.12.2020. https://frontendmasters.com/books/front-end-handbook/2018/what-is-aFD.html.

What is Angular? n/d. Viitattu 29.3.2021. https://angular.io/guide/what-is-angular.

# Liitteet

## Liite 1. Frontend-perusteet opintojakson lähtötasokysely

**Frontend-perusteiden lähtötasokysely**

Kurssin pohjalta tehtävän opinnäytetyön kysely, jossa selvitetään kurssille osallistujien taustatiedot ja kiinnostus frontend-kehittämisestä. Kysely on anonyymi ja tietoja käsitellään luottamuksellisesti. Annettuja tietoja käytetään vain tässä opinnäytetyössä.

* Pakollinen

1. Monettako vuotta opiskelet? *

- 1.
- 2.
- 3. tai enemmän

2. Onko sinulla jo jotain pohjatietoja frontend-kehittämisestä? *

- Kyllä
- Ei

3. Oletko käyttänyt jotain Frontend frameworkiä aiemmin? *

- En ole käyttänyt mitään aiemmin
- React
- Angular
- Vue
- Svelte
- Muu

4. Voisitko kuvitella ohjelmistokehittämistä tulevana uravaihtoehtona *

- Kyllä
- En
- En osaa sanoa

5. Kuvaa muutamalla sanalla lähtöosaamisesi kielistä: HTML, CSS ja JavaScript *

## Liite 2. Frontend-perusteet opintojakson loppukysely

**Frontend-perusteiden loppukysely**

Kurssin pohjalta tehtävän opinnäytetyön kysely, jossa selvitetään kurssille osallistujien kokemuksia kurssista ja kiinnostusta frontend-kehittämisestä. Kysely on anonyymi ja tietoja käsitellään luottamuksellisesti. Annettuja tietoja käytetään vain tässä opinnäytetyössä.
* Pakollinen

1. Koetko oppineesi frontend-kehittämisen perusteet tällä opintojaksolla? *

Kyllä
En

2. Asteikolla 1–5. jossa 1 on huono ja 5 on hyvä, kuinka hyvin koet osaavasi toteuttaa frontend-sovelluksia Sveltellä? *

1 2 3 4 5

3. Asteikolla 1–5, jossa 1 on huonoin ja 5 on paras, kuinka motivoitunut olit oppimaan opintojakson asiat? *

1 2 3 4 5

4. Asteikolla 1–5, jossa 1 on helppo ja 5 on vaikea, kuinka vaikeaksi koit kurssin? *

1 2 3 4 5

5. Mikä tuntui haastavalta ymmärtää tällä opintojaksolla? *

6. Voisitko kuvitella ohjelmistokehittämistä tulevana uravaihtoehtona *

Kyllä

En

En osaa sanoa

## Liite 3. Opinnäytetyön teemahaastattelumateriaali

**Opinnäytetyön haastattelumateriaali**

1. Frontend perusteiden helppous

- Muistatko omat lähtökohtasi ennen peruskurssia?
- Millaiset odotukset sinulla oli kurssista?
- Millaisia tunteita frontend-perusteiden opiskelu herätti?
- Mikä frontendin perusteissa oli haastavinta hahmottaa/ymmärtää?
- Innostuitko frontend-kehittämisestä perusteiden myötä?

2. Opittu framework

- Millä frameworkilla opit ensimmäistä kertaa rakentamaan frontendia?
- Pohjatietona olit opiskellut HTML:n, CSS:n ja JavaScriptin. Pystyitkö mielestäsi hyödyntämään näitä suoraan frameworkia opiskellessasi? Oliko pohjatiedot riittävät frameworkin opiskeluun?
- Osaatko arvoida kuinka kauan sinulla kesti, ennen kuin frameworkin logiikka aukeni? (Esimerkiksi missä vaiheessa kurssia vai kurssin jälkeen?)
- Kuinka intuitiivisena pidit oppimaasi frameworkia? (Eli kuinka vaistonvaraisesti, automaattisesti frameworkin logiikka aukeni? Oliko se helppo/selkeä ottaa käyttöön? Oliko omien projektien kehittäminen luontevaa?)
- Löysitkö helposti ohjeita, tukea ja dokumentaatiota tarvittaessa?

3. Frameworkien erot

*Esitellään kaksi saman käyttöliittymänäkymän tuottavaa sovellusta. Toinen on koodattu Sveltellä ja toinen Angularilla. Tarkoituksena on, että käymme yhdessö läpi näiden frameworkien logiikkaa ja kertoisit minulle mitä ajatuksia ne herättävät.*

- Mitä mieltä olet eroista:
    - Komponenteissa?
    - Projektin rakenteessa?
    - Datan käsittelyssä (if-lohkot, forEach-kohkot, databinding jne.)?
- Omien kokemustesi ja näkemäsi perusteella, kumman koet olevan helpompi?
- Kummalla olisit halunnut oppia frontendin alkeet?

Kuva luodusta käyttöliittymäesimerkistä


**Juurikomponentti**

Angular HTML

```
<!--Tämä on juurikomponentin template-osuus-->
<main>
    <h1>Pääkomponentti</h1>
    <!--Näin kutsutaan muita komponentteja-->
<app-input-komponentti></app-input-komponentti>
<app-kortti-komponentti></app-kortti-komponentti>
<button id="popUp" (click)="popup()">Avaa pop up</button>
</main>
```

Angular TS

```
// Juurikomponentin TypeScript-osuus
// Importattu Angular material helpompaan popupn luomiseen

import { Component, OnInit } from '@angular/core';
import { MatDialog } from "@angular/material/dialog";
import { PopUpKomponenttiComponent } from './pop-up-komponentti/pop-up-kompo-
nentti.component';
```

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit{

  // Materialin dialogi otetaan käyttöön constructorissa
  constructor(private dialog: MatDialog) { }

  ngOnInit(): void {
  }

  title = 'angular-esimerkki';

  // Funktio, joka avaa PopUpKomponentin
   popup() {
    this.dialog.open(PopUpKomponenttiComponent);
  }
}
```

Angular CSS

```
/*Juurikomponentin tyylitiedosto*/

main {
  text-align: center;
  padding: 3em;
  max-width: 240px;
  margin: 0 auto;
  background-color: lightyellow;
}

h1 {
  color: #ad1414;
  text-transform: uppercase;
  font-size: 4em;
  font-weight: 100;
}

@media (min-width: 640px) {
  main {
    max-width: none;
  }
}

#popUp {
  position: fixed;
  left: 45%;
```

```
  top: 60%;
}
```

Svelte

```
<!--Tämä on juurikomponentti-->
<script>
  import KorttiKomponentti from "./KorttiKomponentti.svelte";
  import InputKomponentti from "./InputKomponentti.svelte";
  import PopUpKomponentti from "./PopUpKomponentti.svelte";

  let ikkuna = false;

  let valittuTervehdys = "";

  const tervehdykset = [
    "Tervehdys! Olet upea!",
    "Moi! Hyvältä näyttää!",
    "Terve! Osaat hyvin!",
    "Hei! Päivä parani, kun sinä saavuit paikalle.",
    "Moikka!! Sinua on aina kiva nähdä!",
    "Tere! Olet taitava!",
    "Heippa! Olen sinusta ylpeä!",
  ];

  function randomTervehdys() {
    valittuTervehdys =
      tervehdykset[Math.floor(Math.random() * tervehdykset.length)];
  }
</script>

<main>
  <h1>Pääkomponentti</h1>
  <!--Näin kutsutaan muita komponentteja-->
  <KorttiKomponentti />
  <InputKomponentti />
  <button
    id="popUp"
    on:click={() => {
      ikkuna = true;
      randomTervehdys();
    }}>Avaa pop up</button
  >

  {#if ikkuna}
    <PopUpKomponentti>
      <h1 class="modal-header" slot="header">Pop up-komponentti</h1>
      <p class="modal-text">
        {valittuTervehdys}
      </p>
```

```
      <div slot="footer">
        <button class="ok" on:click={() => (ikkuna = false)}>Ok</button>
      </div>
    </PopUpKomponentti>
  {/if}
</main>

<style>
  main {
    text-align: center;
    padding: 1em;
    max-width: 240px;
    margin: 0 auto;
    background-color: lightyellow;
  }

  h1 {
    color: #ad1414;
    text-transform: uppercase;
    font-size: 4em;
    font-weight: 100;
  }

  @media (min-width: 640px) {
    main {
      max-width: none;
    }

    #popUp {
      position: fixed;
      left: 45%;
      top: 80%;
    }

    .modal-header {
      text-transform: none;
      color: black;
      font-size: 20px;
      font-weight: bold;
      text-align: left;
      padding-left: 5%;
    }

    .modal-text {
      text-align: left;
      padding-left: 5%;
    }

    .ok {
      margin-left: -60%;
```

```
      margin-bottom: 5%;
    }
  }
</style>
```

**Korttikomponentti**

Angular HTML

```
<div class="kortti">
    <h2>Kortti-komponentti</h2>
    <p>Tästä napista painamalla näet loput kortin sisällöstä:</p>
    <button (click)="nayta()">Näytä sisältö</button>

    <!--If-lohko toteutetaan *ngIf:n avulla-->
<div *ngIf="naytaSisalto">
    <h3>Tadaa!</h3>
    <p>Laita tämä sisältö piiloon tästä napista:</p>
    <button (click)="piilota()">Piilota sisältö</button>
</div>
</div>
```

Angular TS

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

@Component({
  selector: 'app-kortti-komponentti',
  templateUrl: './kortti-komponentti.component.html',
  styleUrls: ['./kortti-komponentti.component.css']
})
export class KorttiKomponenttiComponent implements OnInit {
  naytaSisalto = false;

  constructor() { }

  ngOnInit(): void {
  }

  nayta() {
    this.naytaSisalto = true;
  }

  piilota() {
    this.naytaSisalto = false;
  }

}
```

Angular CSS

```
.kortti {
  background-color: aqua;
  width: 500px;
  height: 300px;
  padding: 1%;
  margin-bottom: 3%;
  float: left;
  margin-left: 9%;
}
```

Svelte

```
<script>
  let naytaSisalto = false;
</script>

<div class="kortti">
  <h2>Kortti-komponentti</h2>
  <p>Tästä napista painamalla näet loput kortin sisällöstä:</p>
  <button on:click={() => (naytaSisalto = true)}>Näytä sisältö</button>

  <!--If-lohkoja varten luotu If-lohko-->
  {#if naytaSisalto}
    <h3>Tadaa!</h3>
    <p>Laita tämä sisältö piiloon tästä napista:</p>
    <button on:click={() => (naytaSisalto = false)}>Piilota sisältö</button>
  {/if}
</div>

<style>
  .kortti {
    background-color: aqua;
    width: 500px;
    height: 300px;
    padding: 1%;
    margin-bottom: 3%;
    float: left;
    margin-left: 9%;
  }
</style>
```

**Input-komponentti**

Angular HTML

```
<div class="input">
    <h2>Input-komponentti</h2>
    <p>Kirjoita kenttään mitä haluat ja tulosta se korttiin.</p>
```

```
    <!--Input kentän sisältö on määritelty kirjoitus-muuttujan sisällöksi-->
    <input #kirjoitus (keyup.enter)="tallennus(kirjoitus.value)">

    <!--Lähetetään clickillä sisältä typeScriptin puolelle-->
    <button (click)="tallennus(kirjoitus.value)">Tulosta</button>

    <!--ForEach-lohko toteutetaan *ngFor:n avulla-->
    <p *ngFor="let kirjoitus of kirjoitukset">{{kirjoitus}}</p>
</div>
```

Angular TS

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

@Component({
  selector: 'app-input-komponentti',
  templateUrl: './input-komponentti.component.html',
  styleUrls: ['./input-komponentti.component.css']
})
export class InputKomponenttiComponent implements OnInit {

  // Muuttujille voi määritellä minkä tyyppistä sisältö ne sisältävät
  kirjoitukset: string[] = [];
  kirjoitus;

  constructor() { }

  ngOnInit(): void {
  }

  tallennus = (kirjoitus: string) => {
    if (kirjoitus) {
     this.kirjoitukset.push(kirjoitus);
    }
  };
}
```

Angular CSS

```
.input {
  background-color: palevioletred;
  width: 500px;
  height: 300px;
  padding: 1%;
  margin-bottom: 3%;
  float: right;
  margin-right: 9%;
}
```

Svelte

```
<script>
  let kirjoitukset = [];
  let kirjoitus;

  function tyhjaaInput() {
    kirjoitus = "";
  }

  const tallennus = (kirjoitus) => {
    let uusiKirjoitus = kirjoitus;
    kirjoitukset = [...kirjoitukset, uusiKirjoitus];
    tyhjaaInput();
  };
</script>

<div class="input">
  <h2>Input-komponentti</h2>
  <p>Kirjoita kenttään mitä haluat ja tulosta se korttiin.</p>

  <!--Data binding-->
  <input type="text" bind:value={kirjoitus} />
  <button on:click={tallennus(kirjoitus)}>Tulosta</button>

  <!--ForEach on toteutettu each-lohkolla-->
  {#each kirjoitukset as kirjoitus}
    <p>{kirjoitus}</p>
  {/each}
</div>

<style>
  .input {
    background-color: palevioletred;
    width: 500px;
    height: 300px;
    padding: 1%;
    margin-bottom: 3%;
    float: right;
    margin-right: 9%;
  }
</style>
```

**Pop-up-komponentti**

Angular HTML

```
<div class="mat-dialog-content">
    <div class="otsikko">
```

```
        <h2 class="mat-dialog-title">Pop up-komponentti</h2>
        <p>
            <!--Databind typeScript-puolen muuttujaan-->
            {{ valittuTervehdys }}
        </p>
    </div>
    <div class="nappi">
        <button mat-dialog-close>OK</button>
    </div>
</div>
```

Angular TS

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

@Component({
  selector: 'app-pop-up-komponentti',
  templateUrl: './pop-up-komponentti.component.html',
  styleUrls: ['./pop-up-komponentti.component.css']
})
export class PopUpKomponenttiComponent implements OnInit {

  // Muuttujat
  valittuTervehdys;

  tervehdykset = [
    "Tervehdys! Olet upea!",
    "Moi! Hyvältä näyttää!",
    "Terve! Osaat hyvin!",
    "Hei! Päivä parani, kun sinä saavuit paikalle.",
    "Moikka!! Sinua on aina kiva nähdä!",
    "Tere! Olet taitava!",
    "Heippa! Olen sinusta ylpeä!",
  ];

  constructor() { }

  // Elinkaari-funktio, jonka sisältö ajetaan heti komponentin käynnistyttyä
  ngOnInit(): void {
    this.randomTervehdys();
  }

  randomTervehdys() {
    this.valittuTervehdys = this.tervehdykset[
      Math.floor(Math.random() * this.tervehdykset.length)
    ];
  }
}
```

Angular CSS

Muotoilut saatiin Angular Materialin avulla

Svelte

```
<script>
</script>

<!--Data siirtämistä komponenttikutsusta komponenttii slot:n avulla-->
<div class="modal">
  <header>
    <slot name="header" />
  </header>
  <slot />
  <footer>
    <slot name="footer">
      <button on:click>Sulje</button>
    </slot>
  </footer>
</div>

<style>
  .modal {
    position: fixed;
    top: 50%;
    left: 35%;
    width: 25%;
    height: auto;
    background: white;
    border-radius: 5px;
    border: 1px solid black;
  }

  header {
    border-bottom: 1px solid #ccc;
  }
</style>
```