Tampereen ammattikorkeakoulu

# Svelte- ja Angular-Sovelluskehysten vertailu

Saku Tynjälä

Opinnäytetyö
Marraskuu 2021
Tietojenkäsittely
Ohjelmistotuotanto

**TIIVISTELMÄ**

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

TYNJÄLÄ, SAKU:
Svelte- ja Angular-Sovelluskehysten vertailu

Opinnäytetyö 45 sivua
Marraskuu 2021

---

Opinnäytetyössä tarkastellaan JavaScriptiä ja sen johdannaiskieltä TypeScriptiä sekä Angular- ja Svelte sovelluskehyksiä ja niiden eroavaisuuksia. Vertailussa on tarkoituksena tuoda esille kehysten välisiä eroavaisuuksia ja yhtäläisyyksiä sekä antaa lukijalle käsitys siitä mitä sovelluskehyksillä pystyy tekemään.

JavaScript on 1990-luvulla kehitetty ohjelmointikieli, joka toi interaktiivisuutta verkkosivustoille. Vuosien varrella se on kehittynyt myös serverin puolella käytettäväksi kieleksi. Sen avulla on myös kehitetty Node.js ajoympäristö ja monia muita front- ja backend-kehyksiä.

Angular on yksi tunnetuimmista Googlen kehittämistä, Angular JS-sovelluskehykseen pohjautuvista kehyksistä. Svelte on suhteellisen tuore kehys, jonka suosio on noussut vuonna 2019 Svelte 3 -version julkaisun myötä. Molemmat kehykset ovat komponenttipohjaisia.

Suurin ero kehysten välillä on projektirakenteissa ja siinä, kuinka ne manipuloivat verkkosivulla näkyvää asiakirjaobjektimallia. Angularin komponentti koostuu kolmesta eri tiedostosta: logiikka, käyttöliittymä ja tyylit. Svelten komponentin logiikka, käyttöliittymä ja tyylit ovat samassa tiedostossa. Angular on TypeScript pohjainen sovelluskehys, kun taas Svelte on alun perin JavaScript-pohjainen, mutta nykyään Svelte tukee myös Typescriptiä laajennuksien avulla. Angular päivittää sivua virtuaalisen asiakirjaobjektimallin avulla, jolla vertaillaan ja toteutetaan muutoksia verkkosivun oikeaan asiakirjaobjektimalliin, Svelte muokkaa kirurgisesti suoraan verkkosivun asiakirjaobjektimallia.

---

Asiasanat: angular, svelte, javascript, typescript, asiakirjaobjektimalli

## ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Software Production

TYNJÄLÄ, SAKU:
Comparison of frontend frameworks Svelte and Angular

Bachelor's thesis 45 pages
November 2021

---

This thesis explores the concept of JavaScript, its derivative programming language TypeScript and frontend frameworks Angular and Svelte how these differ from each other. Purpose in comparison is to discover difference and similarity of these frameworks.

Scripting language JavaScript was developed throughout in the 1990s, which gave interactivity in the websites. Over the years JavaScript has evolved vastly, to operate as well in server side and to runtime environments. Furthermore, many and various front- and backend frameworks.

Angular is one of the most known frameworks that is based on AngularJS framework. Svelte is a newcomer in the frontend frameworks which popularity has been in the lift since year 2019 Svelte 3 – version release. Both frameworks are component-based frameworks.

The dissimilarity between the frameworks is most notable when it comes to project architecture and how these manipulate document object model of webpages. Angular divides all its components into three file, UI, styles, and logic. Svelte on the other hand keeps all the code that is related to component in the same file. Angular is a TypeScript based framework, whereas Svelte is JavaScript-based, however it supports TypeScript with help of extensions. Angular updates webpages with help of virtual document object model, which is used to compare and execute changes in real document object model. Svelte writes code that surgically updates the document object model of the web pages.

---

Key words: angular, svelte, javascript, typescript, document object model

**SISÄLLYS**

## LYHENTEET JA TERMIT

| | |
|---|---|
| CSS | Tyyliohjeiden laji (Cascading Style Sheet). |
| DOM | Asiakirjaobjektimalli, puunomainen rakenne kuvaama HTML- tai XML-dokumenttia (Document Object Model). |
| Frontend | Viittaa selainpuolen ohjelmointiin, joka luo toiminnallisuuksia verkkoselaimelle, jonka käyttäjät näkevät. |
| Framework | Sovelluskehys, joka tarjoaa valmiiksi tuotettuja toiminnallisuuksia ja nopeuttaa uusien sovelluksien kehitystä. |
| HTML | Hypertekstin merkkauskieli (Hypertect Markup Language). |
| JS | Ohjelmointikieli, jolla luodaan toiminnallisuutta verkkosivuille (JavaScript). |
| NPM | Pakettimanageri lisäosien, kirjastojen ja riippuvuuksien asentamiseen projektille (Node Package Manager). |
| TS | Ohjelmointikieli, joka rakentuu JavaScript ohjelmointikielen varaan (TypeScript). |

# JOHDANTO

Verkkosovellusten ja -sivujen tärkeys on korostunut huimasti asioiden siirtyessä verkkoon. Uusien verkkosivujen ratkaisut kehitetään nykyään poikkeuksetta dynaamisiksi, vaikkakin staattisia verkkosivuja on vielä olemassa. JavaScript ja siitä johdannaiset ohjelmointikielet ovat tärkeässä asemassa verkkokehityksessä, koska uusia verkkosivuja kehittäessä, on käytössä usein näillä ohjelmointikielillä käytettävä sovelluskehys helpottamassa kehittäjien työtä.

Verkkosivujen kehittäminen tulisi olla mahdollisimman sujuvaa ja helppoa. Koska ohjelmointialalla teknologiat, tekniikat ja toimintatavat kehittyvät nopealla vauhdilla, voi hyvän sovelluskehyksen löytäminen verkkosivujen kehittämiseen tuntua vaivalloiselta ja vaikealta. Uuden sovelluskehyksen oppiminen vie aina aikaa ja resursseja, koska täytyy oppia uusia toimintatapoja ja parhaita käytänteitä. Monet yritykset haluavat pysyä ajan hermoilla uusien verkkosivujen kehityksessä käytettävissä olevien sovelluskehyksien käytössä, mutta uuden teknologian käyttöönotto tulisi aina olla harkittu päätös

Opinnäytetyön tavoite on hakea vastausta siihen, miten Svelte-sovelluskehys eroaa jo tunnettuun asemaan päässyt Angular-sovelluskehyksen kanssa. Työn tarkoituksena ei ole arvioida sovelluskehyksiä parhaimmasta huonompaan, vaan ottaa selvää onko Svelte varteenotettava sovelluskehys ja vakiinnuttaako se asemansa yhtenä käytetyimpänä sovelluskehyksenä.

Opinnäytetyöllä ei ollut toimeksiantajaa, vaan se tuotettiin omaksi eduksi. Opinnäytetyötä saa hyödyntää kuka tahansa aiheesta kiinnostunut.

# 1 JAVASCRIPT

JavaScriptin kehitti vuonna 1995 Brendan Eich ollessaan insinöörinä Netscapella. Alkujaan kieli oli saamassa nimen LiveScript, mutta nimettiin uudelleen. Toisin kuin useimmissa muissa ohjelmointikielissä, JavaScriptillä ei ole syöte- tai tulostekäsitettä. Se on suunniteltu toimimaan komentosarjakielenä isäntäympäristössä, ja isäntäympäristön tehtävänä on tarjota mekanismeja kommunikoimiseksi ulkomaailman kanssa. Yleisin isäntäympäristö on selain (Doshi, L. 2021).

JavaScript on kevyt, tulkittu, objektipohjainen ohjelmointikieli, joka sisältää ensiluokkaisia toimintoja, ja se parhaiten tunnetaan verkkosivujen komentosarjakielenä, sitä myös käytetään monissa muissa ympäristöissä kuin selainympäristöissä. JavaScript on prototyyppipohjainen, usean paradigman skriptikieli, joka toimii dynaamisesti ja tukee oliopohjaista, välttämättömiä ja toiminnallisia ohjelmointityylejä (MDN Web Docs. n.d).

Toisin kuin yleisen väärinkäsityksen mukaan, JavaScript ei ole ”Tulkittu Java”. Lyhyesti sanottuna JavaScript on dynaaminen skriptikieli, joka tukee prototyyppipohjaisia objektirakennetta. Perussyntaksi on tarkoituksellisesti samankaltaista kuin Java ja C++ ohjelmointikielet, mikä vähentää kielen oppimiseen tarvittavia uusien käsitteiden määrää. Kielen konstruktiot, kuten ehtolauseet, for-silmukka, while-silmukka ja vaihtolauseet sekä try-catch-lohkot toimivat samalla tavalla tai hyvin samankaltaisesti kuin yllä mainituilla kielillä (MDN Web Docs. n.d).

## 1.1 Ominaisuudet

Ensisijainen esimerkki JavaScriptistä on sen kyky lisätä interaktiivisuutta verkkosivulle. Tämä toimii, koska tulkki on upotettu verkkoselaimeen, joten sen toimimiseksi ei tarvitse lisätä mitään ohjelmistoja. (Ferguson, R. 2019)

Tämä on tehnyt JavaScriptistä helposti saatavilla olevan kielen, koska sen käyttämiseen tarvitsee vain tekstieditorin ja selaimen. Verkkoselaimen puolella siihen voidaan lisätä vuorovaikutteisuutta, kuten painikkeiden painalluksiin vastaamista ja lomakkeen sisällön vahvistamista. Sen avulla voi myös hyödyntää selaimeen

rakennettuja sovellusohjelmointirajapintoja. Esimerkkinä tästä on geolokaatio paikannusominaisuuksien lisääminen verkkosivustolle. (Ferguson, R. 2019)

Toinen käyttötapa on JavaScriptin suorittaminen palvelimella käyttämällä Node.js -ympäristöä. Esimerkki palvelinpuolen JavaScriptin käyttötavasta on sen kyky tehdä esimerkiksi pyyntöjä tietokannasta ja vastata http-pyyntöihin ja luoda tiedostoja (Ferguson, R. 2019).

JavaScript on löyhästi tyypitetty ja dynaaminen kieli. Tämä tarkoittaa sitä, että JavaScript on erittäin mukautuva ohjelmointikieli. Riippuen käyttötarkoituksesta, tämä voi olla joko huono tai hyvä asia (Ferguson, R. 2019).

## 1.2 TypeScript

TypeScript on JavaScript kielellä käännettävä ohjelmointikieli, jonka Microsoft julkaisi avoimen lähdekoodin projektina vuonna 2012. TypeScriptillä kirjoitettu ohjelma käännetään ensiksi JavaScript kielelle ja sitten se voidaan suorittaa selaimessa tai erillisessä JavaScript moottorilla (Fain, Y. & Moiseev, A. 2020).

TypeScript on staattisesti tyypitetty eli siihen voidaan julistaa muuttuja ja muut6 tietorakenteet tietyntyyppisiksi, kuten merkkijonoiksi tai totuusarvoiksi, ja TypeScript tarkistaa niiden arvojen oikeellisuuden. Tämä ei ole mahdollista JavaScriptissä, joka on dynaamisesti tyypitetty kieli (Davis, B. 2021).

## 1.3 Node.JS

Node on avoimen lähdekoodin JavaScript-ympäristö, jonka avulla voit suorittaa JavaScriptiä palvelinpuolella. Tämän avulla on mahdollista käyttää samaa kieltä ohjelmakoodin suorittamiseen sekä frontend sekä backend puolella (Ferguson, R. 2019).

Node käyttää kokoelmia moduuleja. Moduulit muodostavat ydintoiminnallisuudet Node:lle ja mahdollistavat esimerkiksi tiedostojärjestelmien, verkkoprotokollien, Binääridatan käytön ja mahdollisuuden keskustella tietokannan kanssa (Ferguson, R. 2019).

Molemmat Node sekä JavaScript, jotka suoritetaan selaimen sisällä, toimivat täsmälleen samalla moottorilla. Sitä kutsutaan V8 JavaScript -suorituskoneeksi. Se on avoimen lähdekoodin ympäristö, joka kääntää kirjoitetun JavaScript -koodin ja kääntää sen nopeasti konekoodiksi. Sen ansiosta Node.js toimii nopeasti (Mead, A. 2018).

Konekoodi on matalan tason koodia, jota tietokoneet voivat käyttää suoraan ilman tulkitsemista. Koneet osaavat vain käyttää tietyntyyppistä ohjelmakoodia, esimerkiksi ne eivät voi suorittaa JavaScript- tai PHP- koodia ilman, että ne olisi ensin muunnettu matalan tason koodiksi (Mead, A. 2018).

### 1.4 Node Package Manager

Node.js sisältää oletuspakettihallinta työkalun nimeltään Node Package Manager (NPM), jonka avulla käyttäjä voi komentorivin kautta asentaa ilmaisia sekä maksullisia laajennuksia online-tietokannasta, joka tunnetaan nimellä NPM-rekisteri (Beniwal, R. 2021).

Node Package Manager:n antaa mahdollisuuden asentaa laajennuspaketteja sekä rekisteröimään paketteja rekisteriin. Lisäksi npm integroituu myös npm -rekisteriin. Npm -rekisteri on verkkopalvelu, joka isännöi tällaisia paketteja. Npm -vakio rekisteriä ylläpidetään osoitteessa https://npmjs.org. Se on käytettävissä avoimessa internetissä, ja sitä ylläpitää Joyent, organisaatio, joka tukee Node.js kehitystä. Paketteja voi julkaista npm -rekisterissä, ja se löytyy nimellä npm -rekisterisivustolta. Esimerkiksi kirjailija on julkaissut paketin nimeltä "express". Paketin voi hankkia suorittamalla ”npm install express” tietokoneella komentokehotteessa tai käymällä osoitteessa http://npmjs.org/package/express (Ali, J. 2013).

### 1.5 DOM

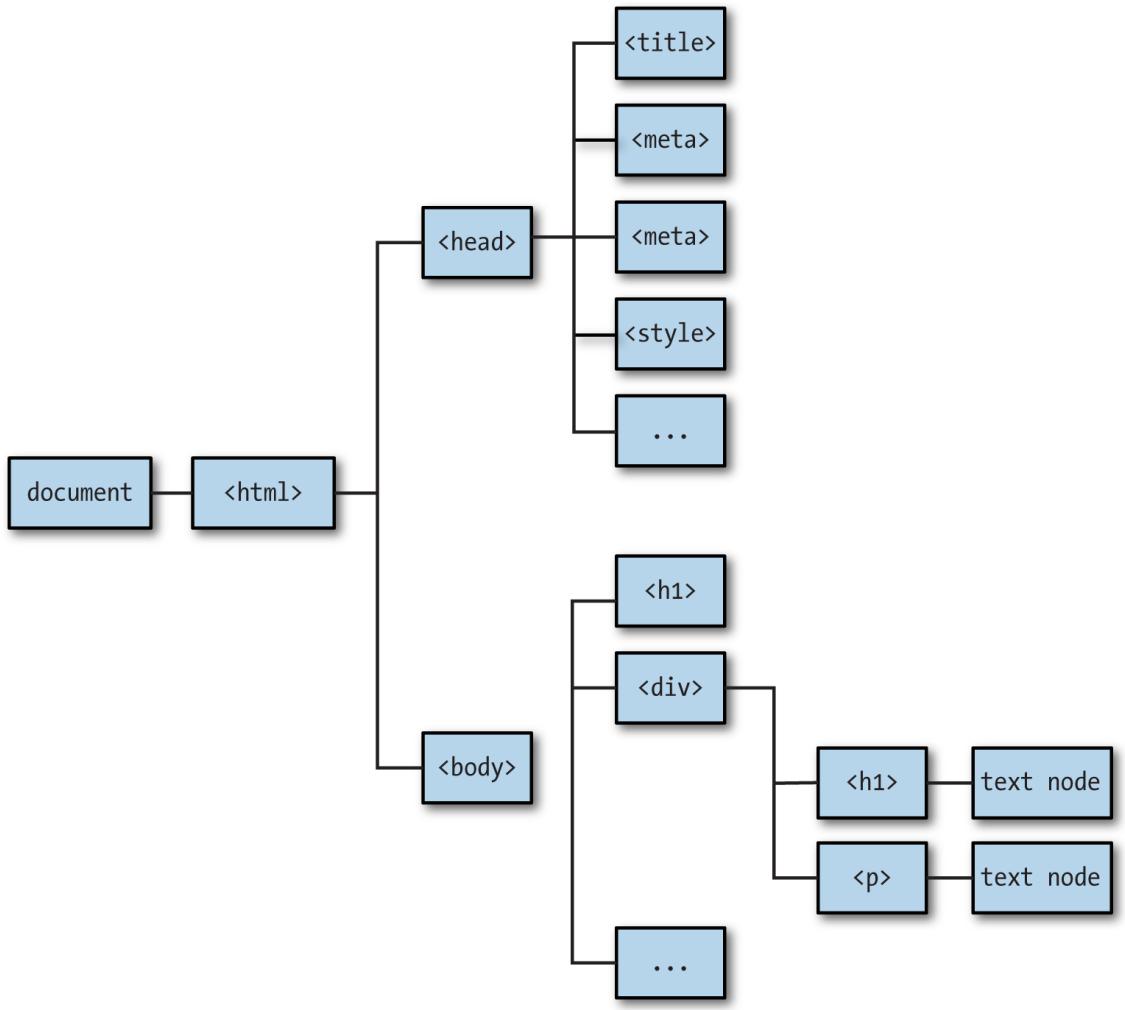
DOM on lyhennys Document Object Model:sta eli asiakirjaobjektimalli. Se on verkkosivusi hierarkkinen esitys (Sheth, K. 2021).

Asiakirjaobjektimalli on ohjelmointirajapinta HTML- ja XML-asiakirjoille. Se määrittää asiakirjojen loogisen rakenteen ja tavan, jolla asiakirjaa käsitellään ja manipuloidaan. DOM-spesifikaatiossa termiä "asiakirja" käytetään laajassa merkityksessä - yhä useammin XML-muotoa käytetään monien erilaisten tietojen esittämisessä, jotka voidaan tallentaa eri järjestelmiin, ja suuri osa tästä nähtäisiin perinteisesti tietoina eikä asiakirjoina. Siitä huolimatta XML esittää nämä tiedot asiakirjoina, ja asiakirjaobjektimallia voidaan käyttää näiden tietojen hallintaan (Robie, J. n.d).

Asiakirjaobjektimallin avulla ohjelmoijat voivat luoda ja rakentaa asiakirjoja, navigoida niiden rakenteessa ja lisätä, muokata tai poistaa elementtejä ja sisältöä. Kaikkiin HTML- tai XML-asiakirjoihin sisältyviin tietoihin voidaan päästä käsiksi, niitä voidaan muuttaa, poistaa tai lisätä käyttämällä asiakirjaobjektimallia muutamaa poikkeusta lukuun ottamatta - erityisesti sisäisen ja ulkoisen osajoukon asiakirjaobjektimallin -rajapintoja ei ole vielä määritetty. (Robie, J. n.d).

Asiakirjaobjektimalli on tapa kuvata HTML-asiakirjan rakennetta, ja on selaimen kanssa toimimisen ydin (Brown, T. 2016).

Käsitteellisesti asiakirjaobjektimalli on puu (Kuva 1). Puu koostuu solmuista: jokaisella solmulla on ylätaso (paitsi juurisolmulla) ja useampia tai ei yhtään alisolmuja. Juurisolmu on asiakirja, ja se koostuu yhdestä lapsesta, joka on <html> -elementti. <html> -elementillä on puolestaan kaksi lasta: <head> -elementti ja <body> -elementti (Kuva 1) (Brown, T. 2016).




KUVA 1. DOM puu

Yllä olevan kuvan asiakirjaobjektimalli näyttäisi html ohjelmakoodina alla olevan kuvan mukaisesti (Kuva 2).

```
<!doctype html>
<html>
    <head>
        <title></title>
        <meta charset="utf-8">
        <meta charset="utf-8">
        <style></style>
    </head>
    <body>
        <h1></h1>
        <div>
            <h1>text node</h1>
            <p>text node</p>
        </div>
    </body>
</html>
```

KUVA 2. Esimerkki html koodista

Jokainen asiakirjaobjektimallin -puun solmu (mukaan lukien itse asiakirja) on solmuluokan esiintymä. Solmuobjekteilla on parentNode- ja childNodes -ominaisuudet sekä tunnistavat ominaisuudet, kuten nimi ja tyyppi (Brown, T. 2016).

Virtuaalinen asiakirjaobjektimalli yksinkertaisesti sanottuna on vain kopio alkuperäisestä asiakirjaobjektimallista, joka säilytetään muistissa ja synkronoidaan todellisen asiakirjaobjektimallin kanssa kirjastojen, kuten ReactDOM:n, kanssa. Tätä prosessia kutsutaan sovinnoksi (Sheth, K. 2021).

Virtuaalisella asiakirjaobjektimallilla on samat ominaisuudet kuin oikealla asiakirjaobjektimallilla, mutta siltä puuttuu valta muuttaa näytön sisältöä suoraan (Sheth, K. 2021).

Virtuaalista asiakirjaobjektimallia voi ajatella koneen suunnitelmana, suunnitelmaan tehdyt muutokset eivät heijastele itse laitetta (Sheth, K. 2021).

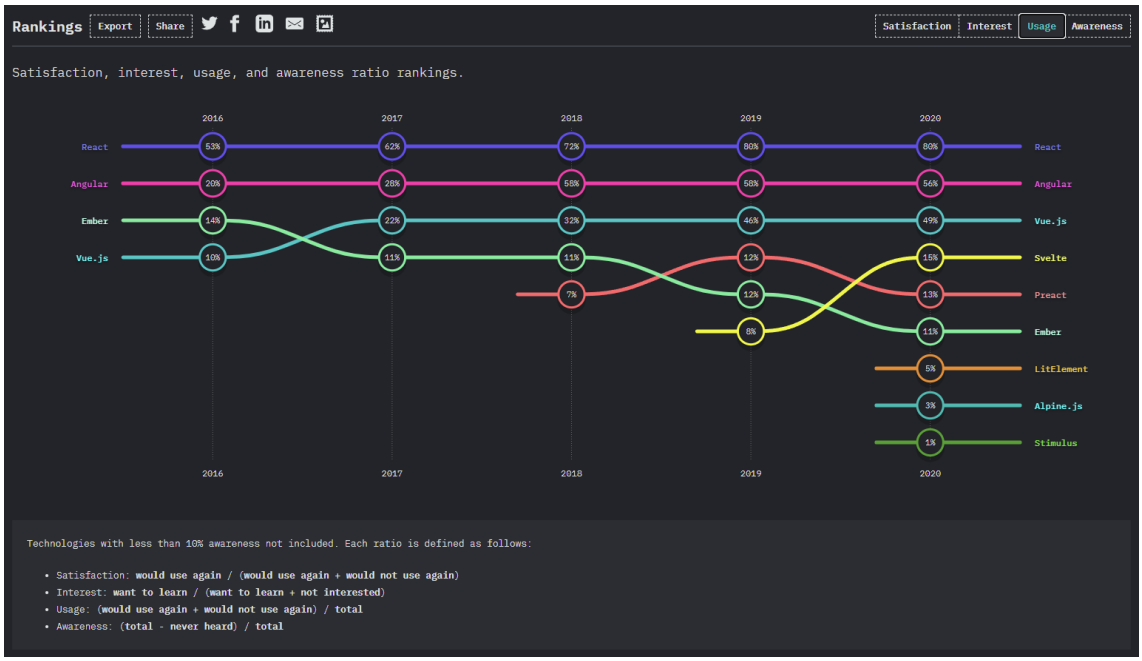
Google pyrkii kehittämään toisenlaista lähestymistapaa inkrementaalisen asiakirjaobjektimallin kanssa. Sillä on sama perusta kuin virtuaalisella asiakirjaobjektimallilla, mutta eri lähestymistapa. Sen sijaan, että se rakentaa esityksen asiakirjaobjektimalli -puusta muistiin, se käyttää todellista asiakirjaobjektimallia vertailussa uusia puita vastaan (Terradillos, P. 2015).

On tärkeä ymmärtää, että virtuaalinen asiakirjaobjektimalli ei ole ominaisuus. Se on keino päästä lopputulokseen tai päämäärään, joka on deklaratiivinen, tilan ohjaama käyttöliittymän kehittäminen. Virtuaalinen asiakirjaobjektimalli on hyödyllinen, koska sen avulla voit rakentaa sovelluksia ajattelematta tilamuutoksia, ja suorituskyky on yleensä riittävän hyvä. Tämä tarkoittaa vähemmän viallista koodia ja enemmän aikaa luoviin tehtäviin työläiden sijaan. Mutta käy ilmi, että samanlaisen ohjelmointimallin voi saavuttaa käyttämättä virtuaalista asiakirjaobjektimallia, Svelten avulla tämä on mahdollista (Harris, R. 2018).

# 2 FRONTEND-KEHYKSET

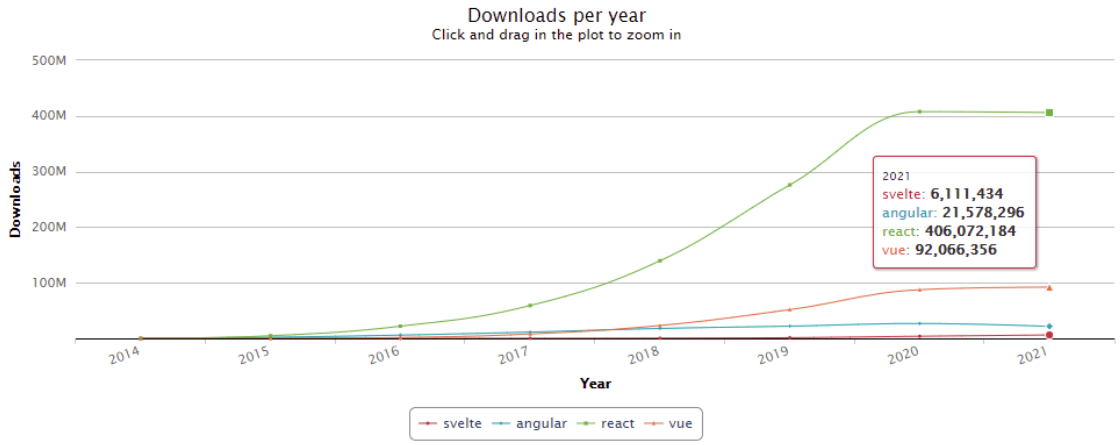
## 2.1 Suosio kehysten välillä

Frontend-sovelluskehyksiä löytyy käytettäväksi hyvinkin suuri määrä, mutta vuoden 2020 StateofJS tuottaman kyselyn mukaan neljä käytetyintä frontend-sovelluskehystä olivat React, Angular, Vue ja Svelte (Kuva 3). Syy Svelten nousu neljän käytetyimmän frontend-sovelluskehyksen joukkoon on sille vuonna 2019 Svelte Versio 3 julkaisu.



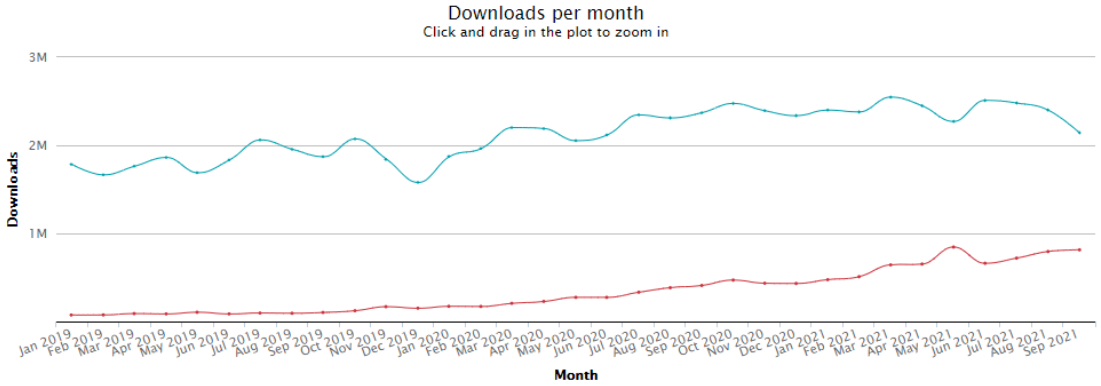

KUVA 3. StateofJS sovelluskehyksien käytöstä vuonna 2020.

Käytetyimpien frontend-sovelluskehyksien keskuudessa eniten käytetyin on React. Vue ohitti Angulari: vuonna 2017 latauksien määrässä NPM-rekisterissä ja on selkeästi ylittänyt Angularin latausten suhteen (Kuva 4). Svelte ei vielä huomattavasti erotu latausten määrässä, mutta jos sitä vertailee Angular:n kanssa, on sitä ladattu jo noin neljännes Angular:n latauksiin verrattuna vuoden 2021 aikana.




KUVA 4. Lataukset vuosittain react, vue, angular, svelte. (NPM-stats.com 2014–2021)

Angular:n latausten määrä on hyvinkin ailahtelevaa kuukausien välillä (Kuva 5), kun Svelte:n latausten määrä on melko tasaisesti noussut vuodesta 2019 lähtien kun sen versio 3 julkaistiin.



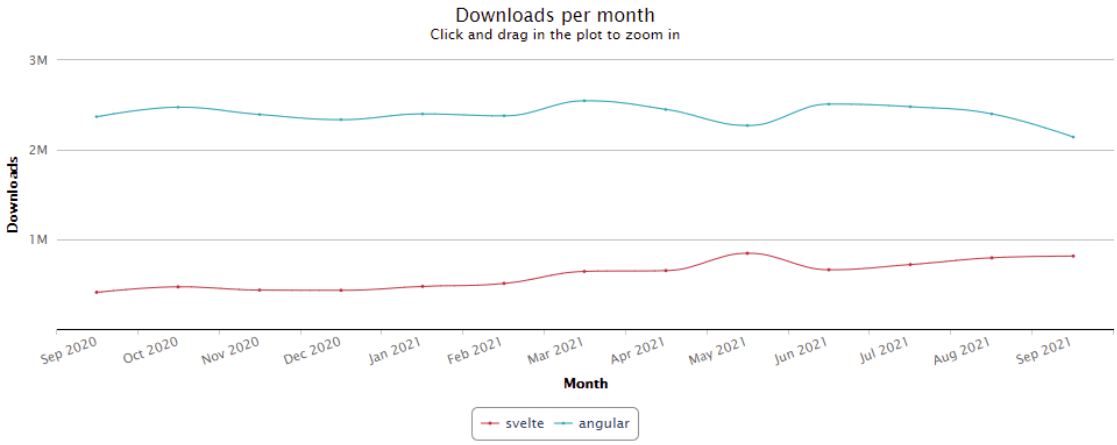

KUVA 5. Lataukset vuosittain angular ja svelte (NPM-stats-com 2019–2021)

Jos tarkastellaan vuodesta 2019 lähtien latausten määrää (Taulukko 1) näiden kahden ohjelmointikehyksen välillä on Svelte vielä melko tuntematon frontend-sovelluskehys.

TAULUKKO 1. Lataukset angular ja svelte (NPM-stats-com 2019–2021)

| **Paketti** | **Lataukset** |
|---|---|
| Angular | 70,178,519 |
| Svelte | 11,147,045 |

Viimeisen vuoden aikana Angular:n latausten määrä on pysynyt hyvinkin vakaana, kun taas Svelte:n latausten määrä on melkein kaksinkertaistunut (Kuva 6).



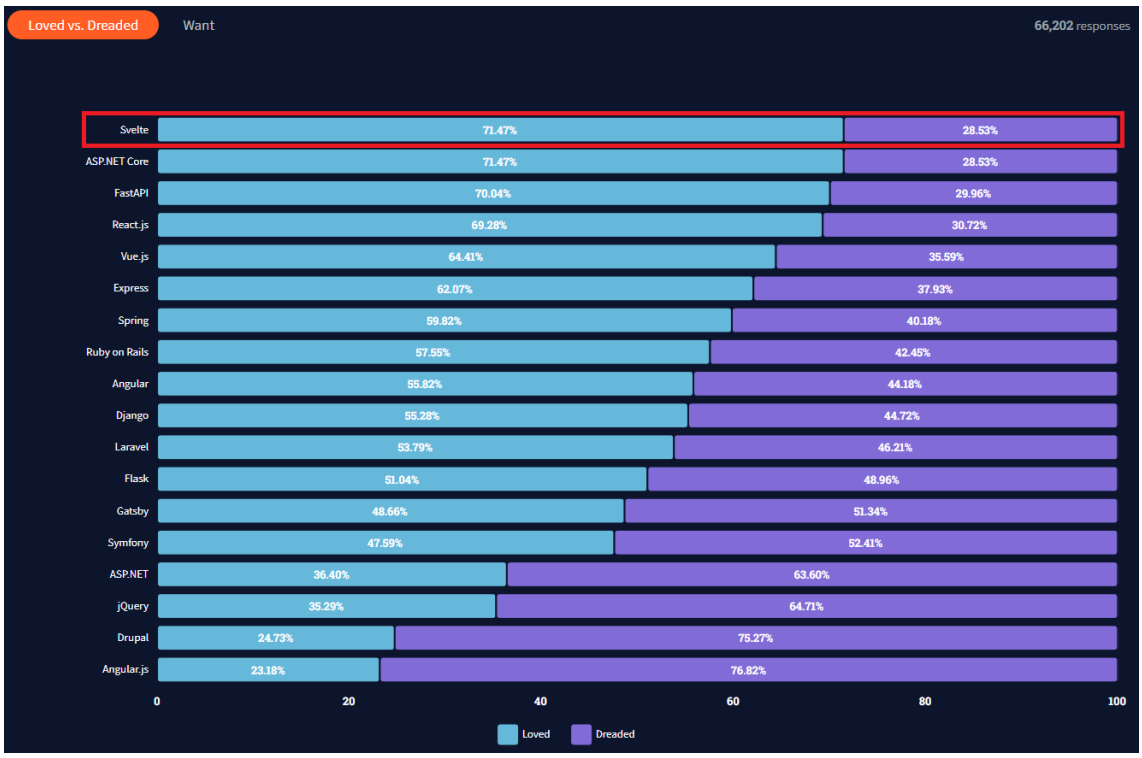

KUVA 6. Lataukset vuosittain angular ja svelte (NPM-stats.com 2020-2021)

Svelte on jo määrällisesti viimeisen vuoden aikana ladattu neljännes Angular:n latausmäärästä (Talukko 2).

TAULUKKO 2. Lataukset angular ja svelte (NPM-stats-com 2020–2021)

| **Paketti** | **Lataukset** |
|---|---|
| Angular | 31,151,735 |
| Svelte | 7,856,984 |

Vaikka Svelte:n latausmäärät ja käyttäjämäärät eivät vielä yllä Angular:n, React:n tai Vue:n tasolle pääsi se Stack Overflown:n vuonna 2021 tuottamassa kyselyssä (Kuva 6) rakastetuimmaksi web-sovelluskehityskehykseksi.




KUVA 6. Stack overflow:n vuoden 2021 kysely (2021 Developer Survey. n.d).

Stack Overflow:n markkinointiviestinnän varatoimitusjohtaja Khalid El Khatib uskoo, että todennäköisesti Svelte:n nopea nousu huipulle johtuu siitä, koska sen käytön aloittaminen on helppoa, se on avoimen lähdekoodin kehys ja sillä on kääntäjäkehys, joka on tehokkaampi kuin muut (Kernes, S. 2021),

## 2.2 Angular

Angular on avoimen lähdekoodin Javascript-kirjasto, jota sponsoroi ja ylläpitää Google. Angular:n avulla on rakennettu joitakin suurimpia ja monimutkaisia web sovelluksia (Freeman, A. 2020).

Angular on alusta ja kehys yksisivuisten käyttäjäsovellusten rakentamiseen HTML- ja TypeScript-tekniikan avulla. Angular on kirjoitettu TypeScriptillä. Se toteuttaa ydin- ja valinnaisia toiminnallisuuksia joukkona TypeScript -kirjastoja, joita voidaan tuoda sovelluksiin (Angular docs. n.d).

Angular hyödyntää palvelinpuolen kehittämisen parhaita puolia ja käyttää niitä HTML:n parantamiseen selaimessa ja luo perustan, joka tekee monipuolisten sovellusten rakentamisesta yksinkertaisempaa ja helpompaa. Angular applikaatiot on rakennettu selkeän suunnittelumallin ympärille, joka korostaa sellaisten sovellusten luomista, jotka ovat:

- **Laajennettavia:** On helppo selvittää, kuinka monimutkainen Angular – applikaatio toimii, kun ymmärtää Angular:n. perusteet se merkitsee, että voidaan helposti parantaa applikaatioita tuottamalla uusia ja käytännöllisiä toiminnallisuuksia käyttäjäkunnalle.
- **Ylläpidettäviä:** Angular applikaatioista on helppo etsiä ohjelmointivirheitä ja korjata niitä, mikä tarkoittaa, että pitkän ajan ylläpito on helppoa.
- **Testattavia:** Angularilla on hyvä tuki yksikkö- ja päästä-päähän-testaukseen, mikä tarkoittaa, että vikoja voi löytää ja korjata ennen kuin käyttäjät sen löytävät niitä.
- **Vakioitu:** Angular perustuu verkkoselaimen luontaisiin ominaisuuksiin, joiden avulla voidaan rakentaa standardien mukaisia verkkosovelluksia, jotka hyödyntävät uusinta HTML-koodia ja ominaisuuksia (Freeman, A. 2020).

Angular toimii niin, että sillä voi laajentaa HTML:ää, Angular-sovellukset ilmaisevat toiminnallisuuksia mukautettujen elementtien avulla, ja Angular-sovellus voi tuottaa HTML-asiakirjan, joka sisältää yhdistelmän vakio- ja mukautettuja merkintöjä (Freeman, A. 2020).

Komponentit ovat perusrakennus palikoita Angular-sovelluksissa. Ne kontrolloivat eri osia verkkosivulla, joita kutsutaan näkymiksi, kuten lista tuotteita tai rekisteröitymis- lomake. Angular-sovellus koostuu komponenttipuista, jotka voivat olla vuorovaikutuksessa toistensa kanssa (Bampakos, A. & Deeleman, P. 2020).

Angular-sovelluksen arkkitehtuuri perustuu komponentteihin. Jokainen Angular-komponentti voi kommunikoida ja vuorovaikuttaa yhteen tai useampaan komponenttiin komponenttipuussa (Bampakos, A. & Deeleman, P. 2020).

Angular-sovellukset ovat modulaarisia, Angular sisältää oman modulaarisuusjärjestelmän nimeltään NgModules. NgModules ovat säiliöitä yhtenäiselle koodilohkolle, jotka on varattu sovellusalueelle, työnkululle tai läheisesti liittyville ominaisuuksille. Ne voivat sisältää komponentteja, palveluntarjoajia ja muita kooditiedostoja, joiden laajuuden määrittelee sisältävä NgModule. Ne voivat tuoda toimintoja, jotka viedään muista NgModuuleista, ja vievät valittuja toimintoja käyttöön muille NgModuuleille (Angular docs. n.d).

## 2.3 Svelte

Verrattuna Angular- ja React frontend -kehyksiin Svelte on suhteellisen tuore kehys. Sen loi alun perin vuonna 2016 ohjelmistokehittäjä ja visuaalinen toimittaja Rich Harris (Segala, A. 2020).

Version 3 julkaisu huhtikuussa 2019 oli Svelte:n tärkeä hetki, koska se uudelleenkirjoitettiin kokonaan, joka esitteli uuden yksinkertaisemman syntaksin Svelte -komponenttien kirjoittamiseen. Svelten kolmannen julkaisun myötä kehys hyväksyttiin laajalti käyttöön (Segala, A. 2020).

Svelte on komponenttikehys kuten React tai Vue, mutta tärkeällä erolla. Perinteisten kehysten avulla voidaan kirjoittaa deklaratiivisen tilapohjaista koodia, mutta selaimen on tehtävä ylimääräistä työtä näiden deklaratiivisten rakenteiden muuntamiseksi DOM-operaatioiksi käyttämällä tekniikoita, jotka heikentävää kehyksen suorituskykyä ja työllistävää liikaa automaattisia roskien keräilijöitä (Harris. R. 2019).

Sen sijaan Svelte toimii verkkosivun rakennusaikana ja muuntaa komponentit erittäin tehokkaasti pakolliseksi koodiksi, joka muokkaa suoraan asiakirjaobjektimallia. Tämän seurauksena voidaan kirjoittaa kunnianhimoisia sovelluksia, joilla on erinomaiset suorituskykyominaisuudet (Harris. R. 2019).

Svelte:n avulla voidaan rakentaa yksittäisiä, uudelleenkäytettäviä komponentteja kaikenlaisiin projekteihin, mukaan lukien suurempiin sovelluksiin, jotka on kirjoitettu Angular-, React-, Vue- tai muilla kehyksillä. Tai sen avulla voidaan rakentaa kokonaisia verkkosovelluksia (Segala, A. 2020).

Useimmissa vertailuissa Svelte 3 erottuu kaikkien kehysten joukosta kyvystään tuottaa pienempiä koodipaketteja, jotka toimivat nopeammin selaimessa verrattuna Angular tai React sovelluskehyksiin (Segala, A. 2020).

# 3 VERKKOKEHYSTEN RAKENNE JA KÄYTTÖ

## 3.1 Angular

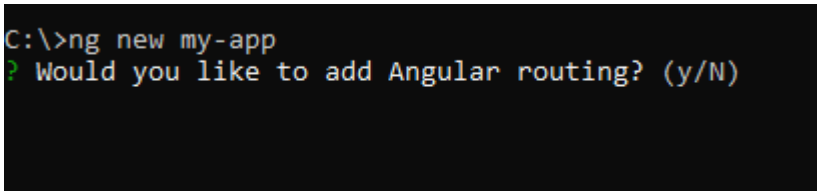
Angular sovelluskehyksen käyttöä varten tulee olla Node.js asennettuna laitteella. Komentokehoitteen kautta, node package manageria käyttäen voi asentaa Angular CLI:n, jonka avulla voi luoda projekteja, generoida applikaatioita ja kirjasto koodia, sekä toteuttaa erinäköisiä kehitystehtäviä kuten testauksia, ohjelmakoodin niputtamista ja julkaista projekteja.

Komennolla ”npm install -g @angular/cli” Angular CLI asentuu laitteelle globaalina pakettina node package managerin avulla.

### 3.1.1 Projektin luonti

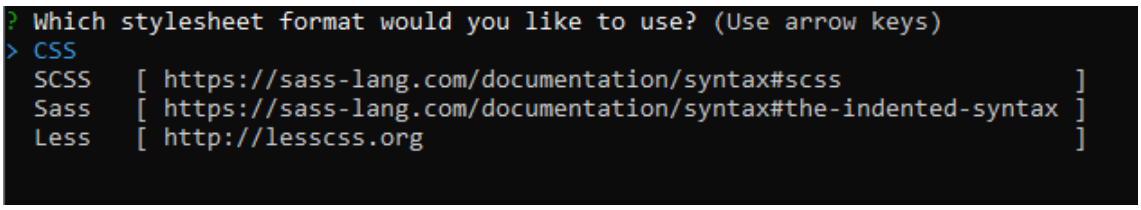
Uuden Angular projekti on helpoin luoda Angular CLI:n avulla komennolla ”ng new my-app”. ”Ng new” on Angular CLI:n komento uuden projektin luomiseen ja ”my-app” komennolle tarjottu nimi projektille.

Uutta projektia luodessa kysyy Angular CLI asetuksia kuten lisätäänkö Angular routing moduuli projektiin (Kuva 7) jos alkuperäisessä ”ng new” komennossa ei annettu asetusta tälle.



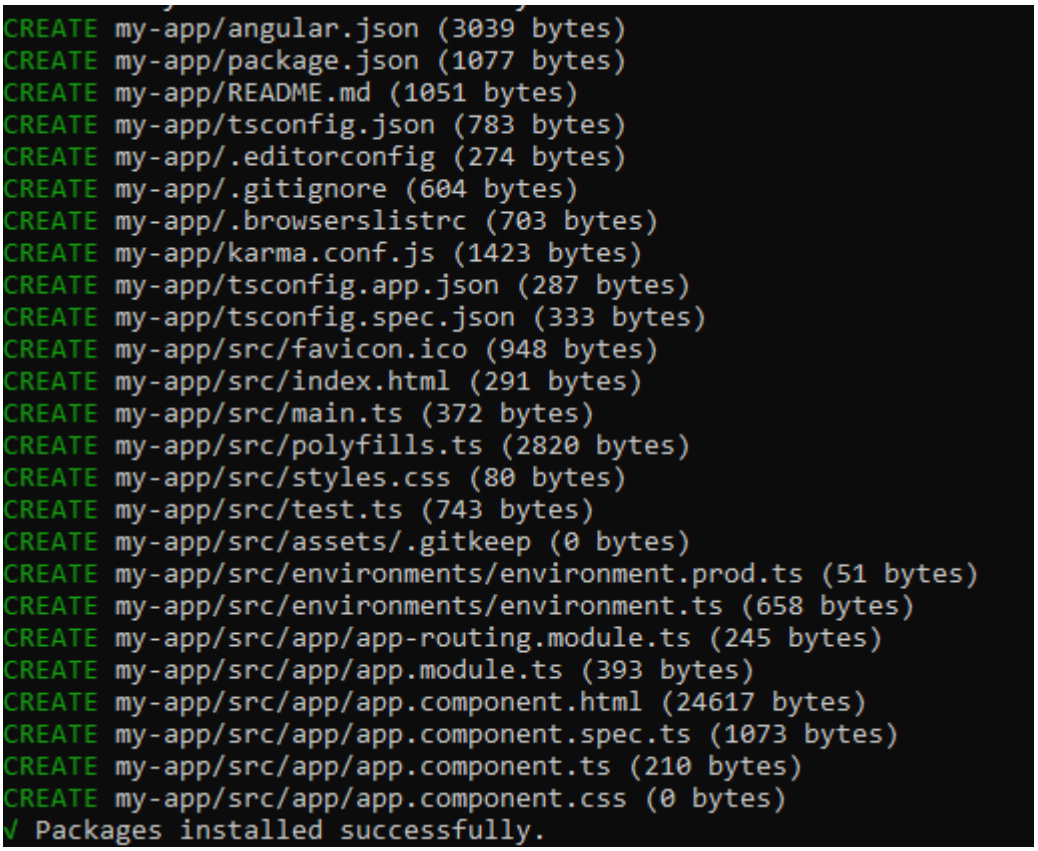

KUVA 7. Angular cli:n kysely lisätäänkö angular routing moduuli projektiin.

Angular CLI kysyy mitä tyylitiedoston formaattia projektissa käytetään (Kuva 8) jos sitä ei annettu alkuperäisessä ”ng new” komennon asetuksissa.



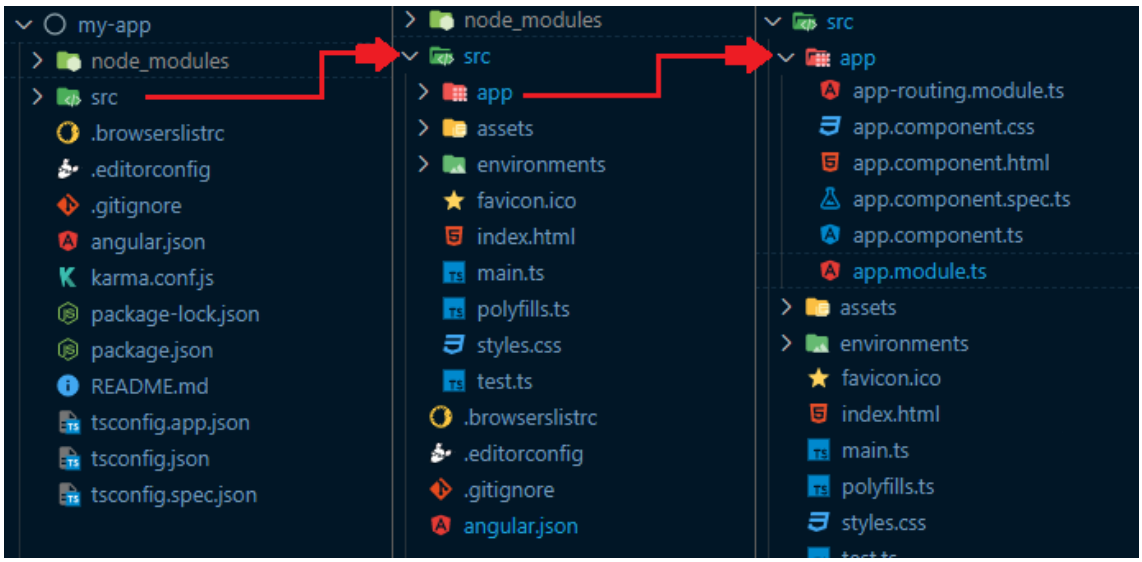

KUVA 8. Angular cli:n kysely mitä tyylitiedoston formaattia projektissa käytetään.

Angular CLI:n asentaa tarpeelliset Angular npm paketit ja muut riippuvuudet uudelle projektille, joka voi kestää muutaman minuutin (Kuva 9).




KUVA 9. Tuloste uuden projektin npm pakettien asennuksen etenemisestä

Tämän jälkeen Angular CLI on luonut kuvan 10 mukaisen Angular projektin, joka sisältää npm paketit ja riippuvuudet sen käynnistämistä varten.

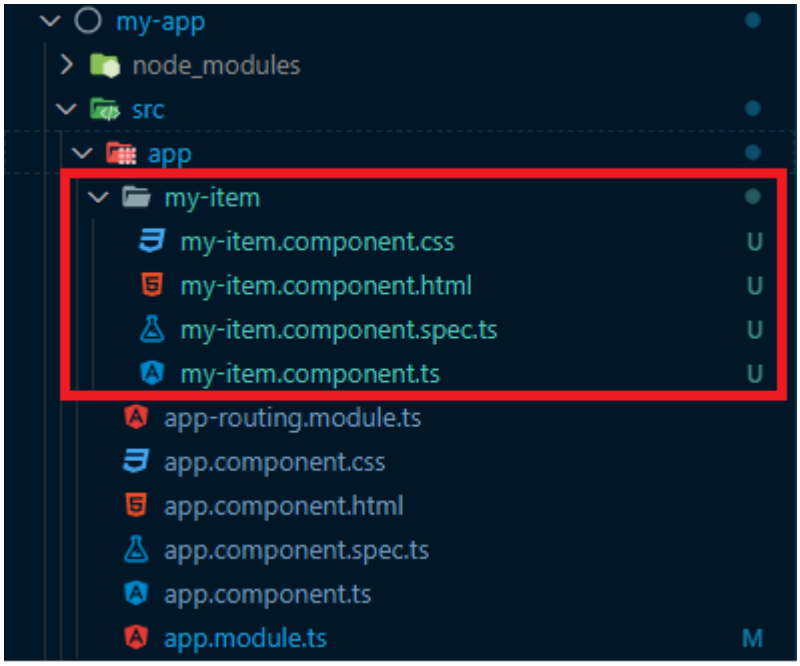



KUVA 10. Uuden Angular projektin rakenne.

### 3.1.2 Komponentin Luonti

Projektikansiossa annettu komento ”ng g component my-item” luo uuden komponentin projektille, joista g viittaa komentoon generate ja component, että sen tulee luoda komponentti ja my-item sille annettava nimi.

Uusi äsken luotu komponentti on korostettu kuvassa 11 punaisella laatikolla projektirakenteessa, jonka Angular CLI juuri loi.




KUVA 11. Angular CLI:n luoma uusi komponentti.

### 3.1.3 Projektin ajo lokaalisti

Projektikansiossa komennolla ”ng serve”, käynnistää projektin lokaalisti verkkoselaimessa nähtävänä sovelluksena. Optio –open aukaisee automaattisesti selaimen.

Sovelluksen portti on oletuksena 4200, eli osoitteesta http://localhost:4200/ pääse tarkastelemaan miltä sovellus näyttää. Komentorivillä ng serve ollessa päällä, päivittää työkalu automaattisesti sovellusta, jos muutoksia ohjelmakoodiin tehdään.

### 3.1.4 Syntaksi

Angular komponenttien logiikka, käyttöliittymä ja tyylit ovat eri tiedostoissa, app.component.ts/html/css.

Alla on komponentin app.component.ts tiedoston sisältö (kuva 12). Tiedostossa alustetaan komponentti ja luokan sisällä luodaan message-muuttuja, joka alustetaan constructorissa ja annetaan tyhjä tekstiarvo. Tiedoston lopussa on changeMessage-funktio, joka vaihtaa message-muuttujan arvon.

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public message: string;

  constructor() {
    this.message = "";
  }

  ngOnInit(): void {
  }

  public changeMessage() {
    this.message = "Hello World"
  }

}
```

KUVA 12. Angular-projektin tiedosto app.component.ts

Funktiota changeMessage kutsutaan app.component.html tiedostosta, kun painiketta on painettu (kuva 13). Komponentissa sisältää vain heading-komponentti, jossa näytetään message-muuttujan arvo, sekä painike, jolla kutsutaan changeMessage-funktiota. app.component.html. tiedostossa oleva router-outlet-komponenti on automaattisesti, jos luotaessa projektia lisäsi Angular Router moduulin projektiin.

```
<div>
  <h2>{{this.message}}</h2>
  <button (click)="changeMessage()">Change Message</button>
</div>

<router-outlet></router-outlet>
```

KUVA 13. Angular-projektin tiedosto app.component.html

## 3.2 Svelte

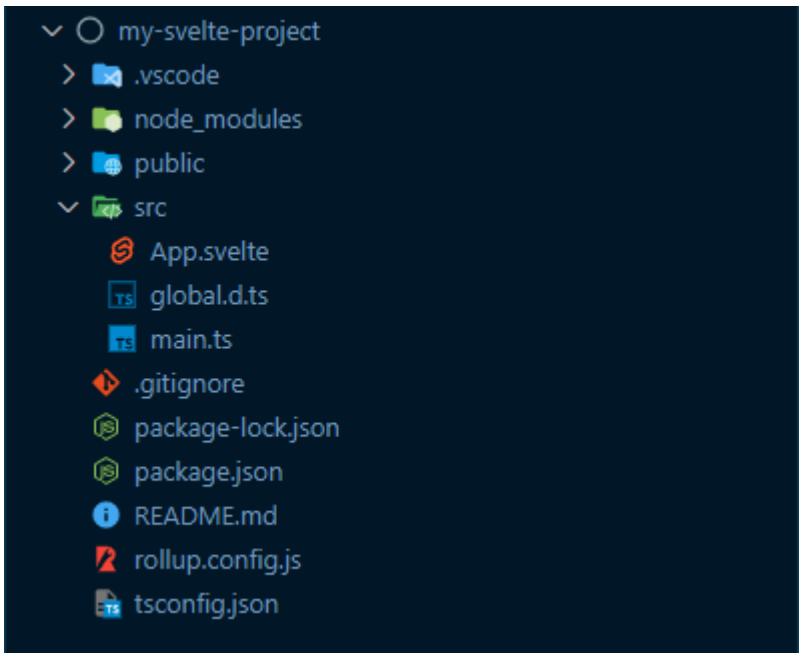
Svelte ei sisällä samankaltaista CLI työkalua kuten Angular, joten joitakin tehtäviä täytyy kehittäjän itse tehdä manuaalisesti projektikansiossa, kuten uuden komponentin luominen projektikansiossa IDE:n kautta.

### 3.2.1 Projektin luonti

Svelte projektin luominen tapahtuu helpoiten komennolla ”npx degit sveltejs/template my-svelte-project”, joka kloonaa sapluunan Svelte projektille. Tämä luo uuden projektin nimeltään my-svelte-project, jonka jälkeen täytyy komento ”npm install” projektikansiossa, joka asentaa node paketit kyseiselle projektille.

Svelte projektin pystyy muuntaa tukemaan TypeScript ohjelmointikieltä komennolla ”node scripts/setupTypeScript.js”.

Tämän jälkeen kuvan 14 mukainen Svelte projekti on kloonautunut hakemistoon, jossa se ajettiin.




KUVA 14. Uuden Svelte projektin rakenne.

### 3.2.2 Projektin ajo lokaalisti

Projektikansiossa komennolla ”npm run dev”, käynnistää projektin lokaalisti verkkoselaimessa nähtävänä sovelluksena.

Sovelluksen portti on oletuksena 5000, eli osoitteesta http://localhost:5000/ pääse tarkastelemaan miltä sovellus näyttää. Komentorivillä npm run dev ollessa päällä, päivittää työkalu automaattisesti sovellusta, jos muutoksia ohjelmakoodiin tehdään.

### 3.2.3 Syntaksi

Svelte komponenttien logiikka, käyttöliittymä ja tyylit ovat samassa tiedostossa. Tyylitiedoston voi halutessaan eriyttää erilliseen tiedostoon. Komponentti koostuu kolmesta lohkosta *script*, *main* ja *style*. Ensimmäisessä lohkossa tapahtuu komponentin logiikka, toisessa komponentin ulkoasu, ja kolmannessa komponentin tyyli.

Alla on komponentin App.svelte tiedoston sisältö (kuva 15). Tiedoston alussa script lohkossa määritetään message-muuttuja ja changeMessage-funktio, jolla voidaan asettaa kyseiselle muuttujalle arvo. Lohkossa main asetetaan ulkoasuksi heading-komponentti, jossa näytetään message-muuttujan arvo, sekä painike, jolla kutsutaan changeMessage-funktiota. Kolmannessa lohkossa, style, asetetaan tyyli main lohkolle ja sen sisällä olevalle heading-komponentille.

```
<script lang="ts">
    let message: string;

    function changeMessage() {
        this.message = "Hello World"
    }

</script>

<main>
    <h1>{message}!</h1>
    <button on:click={changeMessage}>Change Message</button>
</main>

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

    h1 {
        color: #ff3e00;
        text-transform: uppercase;
        font-size: 4em;
        font-weight: 100;
    }
</style>
```

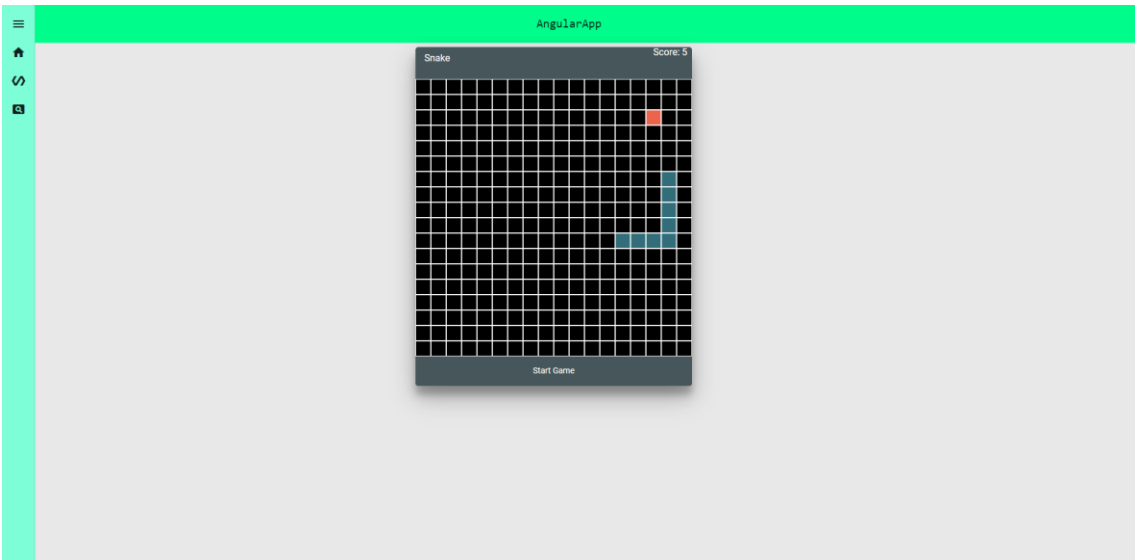
KUVA 15. Svelte-projektin tiedosto App.svelte

# 4 VERTAILU

Opinnäytetyön yhteydessä tehtiin matopeli molemmilla sovelluskehyksillä. Angular sovelluksessa käytettiin versiota 12 ja Svelte sovelluksessa versio 3. Molemmissa sovelluskehyksissä käytettiin Typescript ohjelmointikieltä. Projekteissa ideana oli tarkastella eroavaisuuksia sekä tutustua Svelten parissa työskentelyyn ilman aikaisempaa kokemusta ja miettiä oliko Svelten oppiminen kuinka hankalaa.

## 4.1 Angular Sovellus

Angular matopeli alkaa, kun käyttäjä painaa pelikentän alaosassa olevaa Start game painiketta. Tämän jälkeen mato alkaa liikkumaan pelilaudalla ja kuuntelee näppäimistön painallus tapahtumia, ainoastaan nuolinäppäimiä painamalla mato vaihtaa etenemissuuntaa. Näkymä on kuvan 16 mukainen, jossa mato on sinisen värinen ja hedelmä on punertava. Kun mato syö hedelmän uusi lisätään pelikentälle.




KUVA 16. Angular matopeli

Pelin ulkoasu on esitetty kuvassa 17 jonka pää alueet ovat pistealue, pelikenttä ja aloituspainike. Pelikentän alue kasvaa dynaamisesti sen mukaan mikä sille on asetettu kooksi ohjelman logiikkatiedostossa.

```
<div class="game-container">
  <div class="game-header">
    <h3 class="logo">Snake</h3>
    <div class="score-block">
      <h3 class="score" [ngClass]="{'new-best-score': newBestScore}">Score: {{score}}</h3>
    </div>
  </div>
  <div class="row" *ngFor="let column of board; let i = index;">
    <div class="column" [ngStyle]="{'background-color': setColors(i, j)}" *ngFor="let row of column; let j = index"></div>
  </div>
  <div class="start-button" [ngClass]="{'disable-clicks': gameStarted}" (click)="newGame()">Start Game</div>
</div>
```

KUVA 17. Angular matopeli ulkoasu

Pelikentän ruutujen värit palautetaan setColors funktiosta (kuva 18), jota kutsutaan jokaisen ruudun kohdalla pelin ulkoasussa. Funktio palauttaa ruudulle kuuluvan värin riippuen ruudun sijainnista, madon osien sekä hedelmän sijainnin perusteella.

```
setColors(col: number, row: number): string {
  if (this.isGameOver) {
    return COLORS.GAME_OVER;
  } else if (this.fruit.x === row && this.fruit.y === col) {
    return COLORS.FRUIT;
  } else if (this.snake.parts[0].x === row && this.snake.parts[0].y === col) {
    return COLORS.HEAD;
  } else if (this.board[col][row] === true) {
    return COLORS.BODY;
  }
  return COLORS.BOARD;
};
```

KUVA 18. Peliruudun värin asettaminen

Uuden pelin alustus tapahtuu kuvan 19 newGame funktiossa, jossa asetetaan peli alkaneeksi, pistemäärä nollaksi, madon liikkumissuunta ruudulla vasemmalle sekä pelin päivitys nopeudeksi 150 millisekuntia. Pelin alustuksessa annetaan myös madolle tieto liikkumissuunnasta ja generoidaan sen pituudeksi 3 ruutua. Pelikentälle myös asetetaan yksi hedelmä, joka tapahtuu kutsumalla resetFruit funktiota. Sen jälkeen kutsutaan updatePositions funktiota, jossa pelin päivittyminen tapahtuu.

```
newGame(): void {
  this.gameStarted = true;
  this.score = 0;
  this.tempDirection = CONTROLS.LEFT;
  this.isGameOver = false;
  this.interval = 150;
  this.snake = {
    direction: CONTROLS.LEFT,
    parts: []
  };

  for (let i = 0; i < 3; i++) {
    this.snake.parts.push({ x: 8 + i, y: 8 });
  }

  this.resetFruit();
  this.updatePositions();
}
```

KUVA 19. Uuden pelin alustus funktio.

Madon etenemissuunnan muuttaminen tapahtuu, kun käyttäjä painaa nuolinäppäintä. Nuolinäppäimien painalluksia kuunnellaan Kuvan 20 osoittamassa handleKeyboardEvents funktiossa, joka on sidottu kuuntelemaan webselaimen ikkunassa tapahtuvia näppäimistön painalluksia @HostListener Decorator:n avulla. Funktio tarkastaa onko näppäinpainalluksen tapahtuman avain nuolinäppäin ja sen mukaan määrittää uuden etenemissuunnan madolle.

```
@HostListener('window:keydown', ['$event'])
handleKeyboardEvents(e: KeyboardEvent) {
  switch (e.key) {
    case 'ArrowLeft':
      this.tempDirection = CONTROLS.LEFT
      break;
    case 'ArrowRight':
      this.tempDirection = CONTROLS.RIGHT;
      break;
    case 'ArrowUp':
      this.tempDirection = CONTROLS.UP;
      break;
    case 'ArrowDown':
      this.tempDirection = CONTROLS.DOWN;
      break;
  }
}
```

KUVA *20.* Käyttäjän syötteen kuunteleminen.

Madon liikkuminen tapahtuu kuvassa 21 näytetyssä updatePositions funktiossa. Funktion alussa haetaan madon pään uusi sijainti muuttujaan repositionHead funktiosta, jonka avulla tarkastetaan, onko mato koskettanut seinään boardcollision funktiolla, itseensä selfCollision funktiolla vai hedelmään fruitCollision funktiolla. Funktiot palauttavat boolean arvon sen mukaan onko mato johonkin näistä osunut. Madosta poistetaan viimeinen osa ja otetaan talteen muuttujaan, jonka muutetaan false arvoksi kertomaan pelilaudalla, ettei mato ole enään ruudulla. Matoon lisätään uusi ”pää” sen osien ensimmäiseksi elementiksi ja laitetaan pelitaululla kyseisen sijainti true arvoksi kertomaan madon uusi sijainti. Madon suunnaksi annetaan tempDirection muutujan arvo ja asetetaan aikakatkaistu kutsu samaiseen funktioon jola madon liikettä säädettiin.

```
updatePositions(): void {
  let newHead = this.repositionHead();
  let me = this;

  if (this.boardCollision(newHead)) {
    return this.gameOver();
  }

  if (this.selfCollision(newHead)) {
    return this.gameOver();
  } else if (this.fruitCollision(newHead)) {
    this.eatFruit();
  }

  let oldTail = this.snake.parts.pop();
  this.board[oldTail!.y][oldTail!.x] = false;

  this.snake.parts.unshift(newHead);
  this.board[newHead.y][newHead.x] = true;

  this.snake.direction = this.tempDirection;

  setTimeout(() => {
    me.updatePositions();
  }, this.interval);
}
```

KUVA 21. Madon liikkumisen updatePositions funktio.

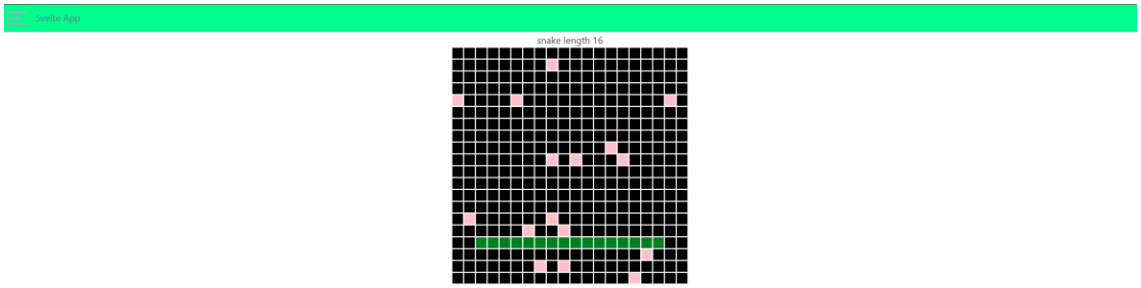
Pään uusi sijainti pelilaudalla tapahtuu repositionHead funktiossa (Kuva 22). Muuttujaan newHead kopioidaan madon vanhan pään sijainti, jonka jälkeen tarkastetaan, mikä on tempDirection muuttujaan asetettu suunta arvo, jonka mukaan newHead muuttujan x- tai y-akselin arvoa muutetaan.

```
repositionHead(): any {
  let newHead = Object.assign({}, this.snake.parts[0]);

  if (this.tempDirection === CONTROLS.LEFT) {
    newHead.x -= 1;
  } else if (this.tempDirection === CONTROLS.RIGHT) {
    newHead.x += 1;
  } else if (this.tempDirection === CONTROLS.UP) {
    newHead.y -= 1;
  } else if (this.tempDirection === CONTROLS.DOWN) {
    newHead.y += 1;
  }

  return newHead;
}
```

KUVA 22. Pään uuden sijainnin hakeminen repositionHead funktio.

## 4.2 Svelte sovellus

Svelte matopeli alkaa heti, kun sivu on latautunut. Mato alkaa liikkumaan pelikentällä oikealle. Kun mato alkaa liikkumaan pelilaudalla, sovellus kuuntelee näppäimistön painallus tapahtumia, ainoastaan nuolinäppäimiä painamalla mato vaihtaa etenemissuuntaa. Näkymä on kuvan 23 mukainen, jossa mato on vihreä värinen ja hedelmät ovat punertavia. Kun mato syö hedelmän, lisätään pelikentälle kaksi uutta hedelmää ja madon eteneminen nopeutuu.




Kuva 23. Svelte Matopeli

Pelin ulkoasu on esitetty kuvassa 24 jonka pää alueet ovat pistealue, pelikenttä. Pelikentän alue kasvaa dynaamisesti sen mukaan mikä sille on asetettu kooksi ohjelman logiikkatiedostossa. Pistealueen teksti vaihdetaan sen mukaan, onko peli vielä käynnissä vai onko mato osunut seinään.

```
<main>
    {#if lost}
    <h1 class="tcenter">you lost</h1>
    <h3 class="tcenter">Hit <code class="enter">ENTER</code> to restart</h3>
    {/if}
    <h3 class="tcenter">snake length {snakePosition.length}</h3>
    <div class="center">
    <div>
        {#each gridWithSnake as row, i}
        <div class="row">
            {#each row as cell, k}
            <div
                on:click={() => (grid[i][k] = 'food')}
                class={`square ${cell}`} />
            {/each}
        </div>
        {/each}
    </div>
    </div>

    {#if lost}
    <div class="center restart">
        <button on:click={restart}> Start again </button>
    </div>
    {/if}
</main>
```

KUVA 24. Svelte matopeli ulkoasu

Pelissä käytettävät päämuuttujat näkyvät kuvassa 25. Pelinpäivitysnopeudeksi on asetettu 200 ms, pelikentän kooksi 20 ruutua, käärmeen pituus on 0 ja totuusarvo tunnistamaan onko peli hävitty, joka on aloituksessa epätosi. Pelikentän ruudut perustuvat solu tyyppiseen arvoon, joka voi olla tyhjä, käärme tai ruoka. Pelikenttä alustetaan aluksi, että kaikki ruudut ovat tyhjiä. Käärmeen sijainnissa käytetään numeraalista rivijoukko arvoa, joka määrittää missä kohtaa pelikenttää käärme sijaitsee. Käärmeen etenemissuunta määritellään numeraalisella rivi arvolla, joka määrittää x ja y suunnan.

```
let TICK_DELAY = 200;
let GRID_SIZE = 20;
let SNAKE_HEAD = 0;
let lost = false;

type Cell = "empty" | "snake" | "food";

let grid: Cell[][] = [...Array(GRID_SIZE)].map(() =>
  [...Array(GRID_SIZE)].map(() => "empty")
);

let snakePosition: Array<[number, number]> = [[12, 13]];
let direction = [0, 1];
let gridWithSnake = grid;
```

KUVA 25. Pelin päämuuttujat

Käärmeen etenemissuunnan muuttaminen tapahtuu, kun käyttäjä painaa nuolinäppäintä. Nuolinäppäimien painalluksia kuunnellaan Kuvan 26 osoittamassa on:keydown funktiossa, joka on sidottu kuuntelemaan verkkoselaimen ikkunassa tapahtuvia näppäimistön painalluksia svelte:window tapahtumienkuuntelijan avulla. Funktio tarkastaa onko näppäinpainalluksen tapahtuman avain nuolinäppäin ja sen mukaan määrittää uuden etenemissuunnan ohjelman direction muuttujalle.

```
<svelte:window
on:keydown={(e) => {
  switch (e.key) {
    case 'ArrowLeft':
      direction = [0, -1];
      break;
    case 'ArrowRight':
      direction = [0, 1];
      break;
    case 'ArrowUp':
      direction = [-1, 0];
      break;
    case 'ArrowDown':
      direction = [1, 0];
      break;
    case 'Enter':
      restart();
      break;
  }
}} />
```

KUVA 26. Käyttäjän syötteen kuunteleminen.

onMount funktio (kuva 27) on Svelten komponentin elämänkaari funktio, joka käynnistyy, kun komponentti on renderöity asiakirjaobjektimalliin. Tästä funktiosta kutsutaan fn-muuttujassa olevaa Lambda ilmaisua, jolle annetaan ohjelmaan määritetty päivitysnopeus ja peli käynnistyy.

```
onMount(() => {
  fn(TICK_DELAY);
});
```

KUVA 27. Svelte onMount elämänkaari funktio.

Käärmeen liikkuminen tapahtuu kuvassa 28 näytetyssä fn-muuttujassa olevassa funktiossa joka sisältää setTimeout() funktion. Funktion alussa haetaan madon pään uusi sijainti newHead-muuttujaan, joka tapahtuu lisäämällä käärmeen sijaintiin etenemissuunta. Sisäisen isOutOfBounds funktion avulla tarkastetaan, onko mato pelikentällä, joka palauttaa onko sille annettu numeraalinen arvo pienempikuin 0 tai suurempi kuin pelilaudan arvosta vähennetty 1. Tämän funktion jälkeen tehdään tarkastus molemmilla newHead rivijoukko muuttujan arvoilla tarkastus onko käärme pelikentällä. Tämän jälkeen tehdään tarkastus, onko mato

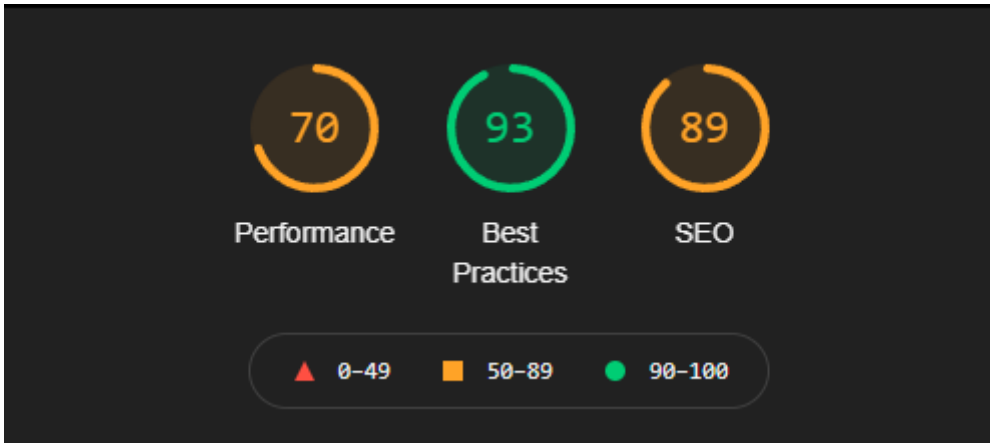
syönyt hedelmää, joka tapahtuu tarkistamalla, onko gridWithSnake muuttujan sijainnissa hedelmä, jossa käärmeen pää on, jolloin kutsutaan randomFood funktiota, joka sijoittaa pelikentälle 2 uutta hedelmää ja asetetaan ateFood totuusarvoon totta arvo, jota käytetään kasvattamaan käärmeen pituutta. Muuttujaan snakeBody kopoidaan käärmeen sijainti ja kasvatetaan sen pituutta, riippuen söikö käärme juuri hedelmän. Tämän jälkeen tarkastetaan, onko käärmeen pää törmännyt juuri asetettuun snakeBody muuttujaan. Näiden jälkeen asetetaan käärmeen sijainnille sen uuden pään sijainti sekä käärmeen keho ja kutsutaan tätä samaa lambda ilmaisua, mutta nopeutetaan pelin nopeutta pienentämällä päivitysnopeutta käärmeen kehon pituuden perusteella. Peli ei kuitenkaan nopeudu, jos käärmeen pituus on jo yli 15.

```
const fn = (n: number) => {
  setTimeout(() => {
    const [x, y] = snakePosition[SNAKE_HEAD];
    const [dx, dy] = direction;
    const newHead = [dx + x, y + dy] as [number, number];
    function isOutOfBounds(n: number) {
      return n < 0 || n > GRID_SIZE - 1;
    }
    if (isOutOfBounds(newHead[0]) || isOutOfBounds(newHead[1])) {
      lost = true;
      return;
    }
    let ateFood = false;
    if (gridWithSnake[newHead[0]][newHead[1]] === "food") {
      ateFood = true;
      randomFood();
    }
    const snakeBody = snakePosition.slice(
      0,
      snakePosition.length - (ateFood ? 0 : 1)
    );
    if (snakeBody.some((x) => x[0] === newHead[0] && x[1] === newHead[1])) {
      lost = true;
      return;
    }
    snakePosition = [newHead, ...snakeBody];
    fn(TICK_DELAY - Math.min(snakePosition.length, 15) * 10);
  }, n);
};
```

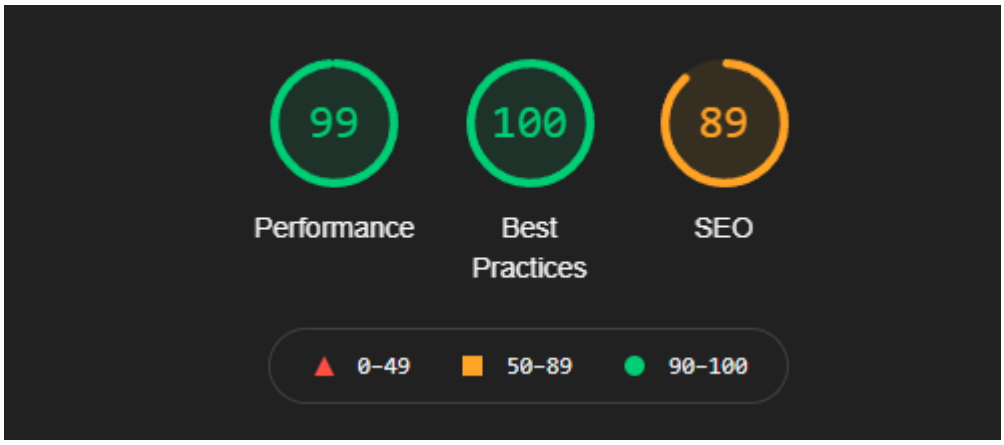
KUVA 28. Käärmeen tarkastukset.

### 4.3 Vertailu Angular vs Svelte

Sovelluksien suurempien erojen selvittämiseksi niitä testataan. Testiympäristönä toimi verkkoselain, joka oli tässä tapauksessa Google Chrome. Chrome sisältää automatisoidun työkalun nimeltä Lighthouse jolla voidaan testata verkkosivuja. Työkalu mittaa sivujen hakukoneoptimointia, suorituskykyä sekä käytettävyyttä, ja pisteyttää verkkosivun nollan ja sadan välillä näissä kategorioissa. Lighthousen tuottama pisteytys molemmista toteutuksista on esitetty kuvissa 29 ja 30.




KUVA 29. Angular lighthouse tulokset




KUVA 30. Svelte lighthouse tulokset

Tulokset olivat hakukoneoptimoinnin ja parhaiden käytänteiden osalta hyvinkin samankaltaiset, mutta suorituskyvyn ero oli yllättävänkin suuri näinkin pienessä toteutuksessa, jossa Svelte oli huomattavasti parempi.

Suorituskyvyn mittaus tarkastaa sovellusten latausaikoja, sekä myös aikaa, kunnes sovelluksesta on tullut täysin interaktiivinen. Parhaiden käytänteiden pisteytys kuvastaa onko sovelluksen toteutuksessa käytetty parhaita käytäntöjä. Tämä saattaa osittain vaikuttaa suorituskykyyn, mutta tässä tapauksessa sen ei pitäisi vaikuttaa merkittävästi koska molemmissa se oli sijoittunut yli 90 pisteen yli. Viimeinen pisteytys kertoo siitä, kuinka hyvin sivu on hakukoneoptimoitu. Molemmat saivat samat pisteet, joten ne ovat yhtä hakukoneoptimoituja.

Taulukkoon 3 on tiivistetty tietoa molempien sovelluksien lighthouse tuloksista, kehitysympäristössä käynnistysajasta ja niputettujen tiedostojen koot, sekä kehittäjäyhteisöön liittyviä tilastoja sovelluskehyksistä.

TAULUKKO 3. Sovelluskehysten vertailu

| | Angular | Svelte |
|---|---|---|
| Projektien käynnistys | 7.2 sek | 2.4 sek |
| Niputetun tiedostojen koko | 4.73 MB | 265 KB |
| Snake-pelin koodirivien Lkm. | 323 | 173 |
| Suorituskyky (Lighthouse) | 70/100 | 99/100 |
| Parhaat käytännöt (Lighthouse) | 93/100 | 100/100 |
| SEO (Lighthouse) | 89/100 | 89/100 |
| Edistäjät (GitHub) | 1489 | 456 |
| Käyttäjät (GitHub) | 2 miljoonaa | 66 tuhatta |
| Tähdet (GitHub) | 77.6 tuhatta | 52 tuhatta |

Taulukko osoittaa, että Svelte käynnistyy kolmanneksen nopeammin nopeammin kuin Angular ja tuottaa huomattavasti pienemmän niputetun ohjelmakoodipaketin selaimelle ajettavaksi. Svelte sai myös paremmat tulokset lighhousen tuottamissa tuloksissa.

Projektin niputetun ohjelmakoodipaketti, johon on sisällytetty koko sovellus, oli Sveltessä pienempi. Svelte ei ole ainoastaan sovelluskehys vaan se on myös kääntäjä, joka niputtaa ohjelmakoodin tehokkaammin kuin tässä tapauksessa Angular.

Koodirivien määrää on myös hyvä tuoda esille toteutuksista. Näin pienissä toteutuksissa koodirivien ero ei vielä ole huomattava, mutta projektien kasvaessa saattaisi koodirivien lukumäärien ero myös kasvaa. Koodirivien vähäisempi määrä Sveltessä johtuu siitä, että komponenttien rakentamiseen vaaditaan vähemmän koodirivejä, eikä toistettavaa koodia jouduta tekemään. Sveltessä on pyritty siihen, että komponentit pysyvät helposti ymmärrettävinä ja mahdollisimman suppeina, kun Angular pyrkii komponenttien uudelleenkäytettävyyteen ja modulaarisuuteen.

Taulukossa esille tuodut edistäjät (enlg. contributors) tarkoittaa kehittäjiä, jotka edistävät sovelluskehysten koodin kehitystä ja luovat niihin muutoksia. Angularilla on yli neljä kertaa enemmän edistäjiä kuin Sveltellä, mikä merkitsee, että Angularilla on suurempi kehittäjäyhteisö. Angularin elinvoimaisuuden takaa myös se, että sen taustalla iso yhtiö nimeltään Google.

Svelten kehittäjäyhteisö on merkittävästi pienempi kuin Angularin, mutta se on nuorempi teknologia. Vaikkakin nykyinen Angular on julkaistu samana vuonna, pohjautuu se AngularJS sovelluskehykseen, jonka ensijulkaisu oli vuonna 2010, eli kuusi vuotta aiemmin kuin Svelten. Svelten elinvoimaisuus ei ole aivan yhtä taattu, koska sillä ei ole teknologiajättiä sen taustalla. Sitä kuitenkin kehitetään aktiivisesti ja se on myös tullut vuosien saatossa useamman kehittäjän tietoisuuteen, sekä saanut arvostusta eri kehittäjäyhteisöissä.

Taulukon käyttäjämäärät viittaavat käyttäjiin, joiden projektit ovat riippuvaisia Svelten tai Angularin GitHub-repositorystä. Angularilla käyttäjiä on 2 000 000. Tämä kertoo, että Angular on laajemmin käytössä kuin Svelte, jolla on 66 000 käyttäjään. Suosiota voidaan tarkastella tähtien perusteella, joita voidaan antaa GitHubissa. Angularille on annettu 77 600 kappaletta tähtiä, kun Sveltellä niitä on jo 52 000. Angularille annettujen tähtien määrä on merkittävästi vähäisempi suhteutettuna käyttäjämääriin. Svelten käyttäjämäärät ja tähdet ovat melko lähellä toisiaan, joka kertoo siitä, että sitä kokeilleet kehittäjät ovat pitäneet Sveltestä.

Käyttäjämäärien ja edistäjien lukumäärä osoittaa, että Angular pysyy vankassa asemassa yhtenä käytetyimpänä sovelluskehyksenä vielä pitkään. Svelten tulevaisuus ei vielä ole yhtä varma näitä lukuja katsoessa.

# 5 POHDINTA

Työn tavoitteen oli tutusta Svelte sovelluskehykseen ja miten se erosi jo tunnettuun asemaan päässeen Angular-sovelluskehyksen kanssa. Tämä saavutettiin tarkastelemalla niitä teoriatasolla, sekä vertailemalla kummallakin sovelluskehyksellä rakennettuja verkkosivujen koodiesimerkkien tapoja toteuttaa asioita. Tavoitteena oli myös selvittää, vakiinnuttaako Svelte asemansa yhdeksi käytetyimmäksi sovelluskehykseksi tarkastelemalla kehittäjäyhteisöjen arvioita, sekä vertailemalla sovelluskehysten latausmääriä NPM-rekisteristä.

Tutkimuksen perusteella pystytään toteamaan, että suurimmat eroavaisuudet sovelluskehyksillä oli projektirakenteissa ja miten ne manipuloivat asiakirjaobjektimallia. Komponenttien kirjoittaminen Sveltellä on sujuvaa ja selkeämpää, koska se muistuttaa tyyliltään perinteiseen HTML-dokumenttiin tuotettua HTML- CSS- ja JS-koodia, kuitenkin pysyen komponenttilähtöisenä. Angular taasen paloittelee komponenttiin liittyvät koodit eri tiedostoihin, joka saa projektikansion paisumaan ja voi tuottaa omia haasteitansa. Tämän takia Svelte tuntui aloittelijaystävällisemmältä sillä komponentin koodit pysyvät samassa tiedostossa ja loogisesti eroteltu.

Testausten perusteella Sveltellä tuotettu lopputuote oli suorituskyvyltään parempi ja kooltaan pienempi kuin Angularilla tuotettu lopputuote. Sovellusten toteutuksesta ei voida tehdä täydellisiä johtopäätöksiä siitä, kumpi on parempi sillä koodin kirjoittaja vaikuttaa siihen, kuinka hyvin sovellukset on toteutettu. Asioita voidaan toteuttaa huonosti hyvilläkin työkaluilla.

Käyttäjämäärien perusteella Svelte ei ole vielä saavuttanut samaa asemaa kuin Angular, mutta se on kerännyt paljon positiivista palautetta kehittäjiltä, jotka ovat sitä kokeilleet. Uskon että Svelte pystyy vakiinnuttamaan asemansa yhtenä käytetyimpänä sovelluskehyksenä, kun kolmansien osapuolten tuottamat kirjastot ja tuki kasvaa niin myös tiettyjen toiminnallisuuksien saatavuus helpottuu.

Kokonaisuudessaan jokainen sovelluskehys sisältää omia hyviä ja huonoja puolia, joita täytyy tarkastella, kun projektille sopivaa sovelluskehystä arvioidaan.

Svelte on aloittelijaystävällisempi kuin Angular, mutta molemmat ovat hyviä valintoja oppimisen kannalta, jos haluaa kehittää itseään kehittäjänä ja uuden sovelluskehyksen oppiminen tuntuu huomattavasti helpommalta, jos omistaa kokemuksesta toisesta sovelluskehyksestä.

**LÄHTEET**

2021 Developer Survey. n.d. StackOverFlow. Luettu 15.10.2021. *https://insights.stackoverflow.com/survey/2021#section-most-loved-dreaded-and-wanted-web-frameworks*

Ali, J. 2013 Instant Node Package Module. Packt Publishing.

Angular docs. n.d. Introduction to modules. Luettu 22.09.2021 https://angular.io/guide/architecture-modules

Bampakos, A. & Deeleman, P. 2020. Learning Angular - Third Edition. 3rd edition. Packt Publishing.

Beniwal, R. 2021. 'NPMREC: NPM Packages and Similar Projects Recommendation System', in Data Analytics and Management. [Online]. Singapore: Springer Singapore.

Brown, T. 2016. Working with the DOM. 1st edition. O'Reilly Media, Inc.

Davis, B. 2021. Why is JavaScript loosely typed. Luettu 15.08.2021 https://www.mvorganizing.org/why-is-javascript-loosely-typed/

Doshi, L. 2021. Introduction to JavaScript. Verkkosivu. Luettu 03.10.2021. https://www.geeksforgeeks.org/introduction-to-javascript/

Fain, Y. & Moiseev, A. 2020. TypeScript Quickly. 1st edition. Manning Publications.

Ferguson, R. 2019. Beginning JavaScript: The Ultimate Guide to Modern JavaScript Development. Berkeley, CA: Apress L. P.

Freeman, A. 2020. Pro Angular 9 Build Powerful and Dynamic Web Apps. 4th ed. 2020. [Online]. Berkeley, CA: Apress.

Harris, R. 2018. Virtual DOM is pure overhead. Luettu 15.07.2021 https://svelte.dev/blog/virtual-dom-is-pure-overhead

Harris. R. 2019. Svelte 3: Rethinking reactivity. Luettu 15.06.2021 https://svelte.dev/blog/svelte-3-rethinking-reactivity

Kernes, S. 2021. Svelte Is 2021's Most Loved Web Framework for Developers. ITProToday 26.8.2021. Luettu 17.10.2021. https://www.itprotoday.com/web-application-management/svelte-2021s-most-loved-web-framework-developers

MDN Web Docs. n.d. What is Javascript. Luettu 13.08.2021 https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

Mead, A. 2018. Learning Node.js Development Learn the fundamentals of Node.js and deploy and test Node.js applications on the web. 1st ed. PACKT Publishing.

Robie, J. n.d. What is the Document Object Model? Luettu 21.08.2021 https://www.w3.org/TR/WD-DOM/introduction.html

Segala, A. 2020. Svelte 3 Up and Running. 1st edition. Packt Publishing.

Sheth, K. 2021. What is Virtual DOM? How Virtual DOM works? What is Reconciliation? What is diffing algorithm? What makes React so fast? Luettu 20.08.2021 https://dev.to/koolkishan/what-is-virtual-dom-how-virtual-dom-works-what-is-reconciliation-what-is-diffing-algorithm-what-makes-react-so-fast-327a

Terradillos, P. 2015. Incremental DOM 101: What is it and why I should care? Luettu 17.07.2021 https://auth0.com/blog/incremental-dom/

Web.dev. n.d. 2021. Lighthouse performance scoring. Luettu 5.11.2021 https://web.dev/performance-scoring/6