TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

**Rebekka Breedis**

# JavaScripti kasutajaliidese raamistike võrdlus

**Bakalaureusetöö (9 EAP)**

Juhendaja: Lidia Feklistova, MSc

Tartu 2021

## JavaScripti kasutajaliidese raamistike võrdlus

**Lühikokkuvõte:**

Veebiarenduses kasutatakse kasutajaliideste loomiseks JavaScripti kasutajaliidese raamistikke. Kasutajaliidese raamistikke on aastate jooksul välja tulnud mitmeid ning algaja arendaja jaoks võib valiku langetamine osutuda keeruliseks. Antud bakalaureusetöö eesmärk on luua ülevaatlik raamistike võrdlev materjal, mille abil on veebirakenduse loojal võimalik langetada informeeritud valik. Töös selgitatakse välja hetkel populaarseimad raamistikud, tuuakse esile tähtsad aspektid, mida tasub raamistiku valimisel tähele panna ning luuakse ülevaade raamistikega seotud teemadest. Raamistikke analüüsitakse vastavalt tähtsatele aspektidele ning iga raamistikuga luuakse testrakendus.

**Võtmesõnad:**

JavaScript, raamistik, kasutajaliides, veebirakendus, Vue.js, Angular, React

**CERCS:** P175 Informaatika, süsteemiteooria

## Comparison of JavaScript user interface frameworks

**Abstract:**

JavaScript UI frameworks are often used in web development to create user interfaces. Over the years many UI frameworks have been created and it can be difficult for a novice developer to make a choice. The aim of this bachelor's thesis is to create a material that gives the future UI developer an overview of the frameworks and assists them in making an informed decision. This bachelor's thesis identifies the current most popular frameworks, highlights important aspects that need to be taken into account when choosing a framework and gives an overview of topics related to these frameworks. Frameworks are analyzed according to important aspects, and a test application is created with each framework.

**Keywords:**

JavaScript, framework, UI, web application, Vue.js, Angular, React

**CERCS:** P175 Informatics, systems theory

# Sisukord

# 1 Sissejuhatus

Juba veebi algusajast on tahtnud arendajad luua veebirakendusi, mis käituvad nagu töölaua rakendused. Töölaua rakendused on kasutajatele mugavad ning kiired, sest info kuvamiseks ei ole tarvis rakenduse vaadet pidevalt värskendada nagu veebilehitsejas [1]. Kliendipoolsed JavaScripti raamistikud ja teegid aitavad muuta veebirakendusi täpselt sama mugavaks. Raamistik on tarkvara, mis võimaldab luua interaktiivsemaid ning keerukama struktuuriga veebirakendusi. Nimelt aitavad need hallata projekti struktuuri ja võimaldavad arendajal keskenduda vajalike funktsionaalsuste loomisele [1]. Raamistike valik on aga lai ning tähtsaid aspekte, mida raamistiku valimisel meeles pidada, on mitmeid. Näiteks tuleb meeles pidada nii kogukonna aktiivsust kui ka õppimiskurvi taset. Algaja arendajale võib valik osutuda keeruliseks. Lisaks aitab läbimõeldud valik kokku hoida nii aega kui ka ressursse, sest raamistikud on erinevad ja nende tundmaõppimine ei pruugi olla triviaalne.

Bakalaureusetöö eesmärk on tuua välja algaja arendajatele tähtsamad aspektid raamistiku valimisel ning koostada ülevaade populaarseimatest kliendipoolsetest JavaScripti raamistikest. Töös selgitatakse välja, millised on kevad 2021 seisuga kõige populaarsemad raamistikud ning võrreldakse neid. Tulemuseks tekib analüüs, mis annab lugejale ülevaate iga raamistiku, taustast, eripärast, tugevustest ja nõrkustest ning raamistike vahelistest erinevustest ja sarnasustest. Tekkinud materjali abiga on lugejal võimalik langetada informeeritud valik.

Selleks, et analüüsi teostada tutvutakse kõigepealt iga raamistiku tausta ning eripäradega. Iga valitud raamistikuga luuakse praktiliseks võrdluseks samasugune kindlat ülesannet täitev testrakendus. See järel pannakse paika ühised kriteeriumid, mida võiks raamistiku valimisel silmas pidada. Näiteks hinnatakse raamistikuga kaasaskäivat dokumentatsiooni, abistavate raamistike olemasolu ja raamistiku toetust arendajate poolt.

Tausta peatükis tutvustatakse JavaScripti ennast ning raamistikega seotud teemasid. Selgitatakse välja kevad 2021 seisuga populaarseimad raamistikud ning tutvustatakse ühiseid kriteeriumeid, mida iga rakenduse puhul hinnatakse. Kirjeldatakse loodava testrakenduse struktuuri, funktsionaalseid nõudmisi ning ühiseid eesmärke. Raamistike kirjelduse ja testrakenduse teostuse peatükis tutvustatakse lähemalt iga raamistiku tausta, põhimõisteid ning töötatakse läbi varasemalt kirja pandud aspektid. Seejärel kirjeldatakse testrakenduse teostust konkreetses raamistikus alates projekti ülesseadmisest kuni valitud komponendi koodi lahkamiseni, et anda lugejale visuaalne ettekujutus raamistikuga töötamisest. Testrakenduste lähtekoodid on saadaval lisas 1. Tulemuste peatükis pannakse kirja iga raamistiku tugevused ning nõrkused ning tehakse järeldused. Töö lõpus on kokkuvõte, kasutatud kirjandus ning lisad.

# 2 Taust

## 2.1 JavaScript

Aastal 1991 lõi Tim Berners-Lee alguse veebile [2]. Peale esimese veebilehe loomise on Berners-Lee autoriks kolmele tehnoloogiale, mis on ka tänapäeval veebiarenduses kasutusel. Kuna tehnoloogia oli uus, oli veebileht küllaltki primitiivne ning veebilehe funktsionaalsus limiteeritud.

Aastal 1994 firma nimega Netscape, mis oli loodud uurima ning ekspluateerima veebi, mõistis, et veeb tuleb muuta dünaamilisemaks [3]. Probleemi lahendamiseks lõi Netscape'i insener Brendan Eich aluse uuele skriptikeelele, mida tuntakse tänapäeval JavaScripti nime all. Nimi JavaScript oli turundustiimi idee kasutada ära tollase koostööpartneri Sun (hiljem tuntud kui Oracle) programmeerimiskeele Java populaarsust. Vastupidiselt levinud arvamusele, on Javal ja JavaScriptil väga vähe ühist. Java ning JavaScript jagavad vaid sarnast süntaksit. Aastal 1996 pöördus Netscape ECMA Internationali poole, et muuta JavaScript standardiks [4]. Standardiseeritud skriptikeele nimeks sai ECMAScript. Vaatamata sellele eelistab kasutajaskond kasutada nime JavaScript ning seda nimetust kasutatakse ka bakalaureusetöös edaspidi.

JavaScript on loodud jooksma kasutaja keskkondades, milleks on peamiselt kasutaja veebilehitseja [5]. JavaScript on dünaamiline skriptikeel sisseehitatud objektide ning meetoditega. JavaScript võimaldab manipuleerida veebilehe sisu kasutades DOM (ingl. k. *Domain Object Model*) märgendeid, luua kasutajaliidesele kanvaa (ingl. k. *canvas*) abil graafikat, suhelda teiste rakendusliidestega (ingl. k. *Application Program Interface)* [6]. Eelmainitud funktsionaalsused andsid võimaluse pakkuda veebilehe külastajatele interaktiivsemat kogemust.

## 2.2 Üheleherakendus

Kaasaarvatud JavaScripti populaarsusega kasvas ka projektide struktuuriline ning funktsionaalne keerukus. Nagu sissejuhatuses sai mainitud soovivad arendajad luua veebirakendusi, mis käituvad nagu töölaua rakendused. Töölaua rakenduste puhul ei ole tarvis uue info kuvamiseks vaadet uuesti värskendada nagu esimeste hüperlinkidega veebilehtede puhul. Selle olukorra parandamiseks võeti kasutusele üheleherakenduse kontseptsioon.

Üheleherakenduse kontseptsioon sai alguse aastal 2005 kui tekkis AJAX [7]. AJAX (ingl. k. *Asynchronous JavaScript and XML*) on kogum, mis kombineerib omavahel mitmed tolleaegsed tehnoloogiad nagu JavaScript, HTML ja XMLHttpRequest objektid ja mitmed muud. AJAX-i abil on võimalik kuvada kasutajale uut informatsiooni serverist ilma veebirakendust värskendamata.

Traditsioonilised veebilehed kasutavad serveripoolset arhitektuuri [1]. Traditsioonilise veebilehe puhul suhtleb kasutaja veebirakenduse vaatega ehk kasutajaliidesega (ingl. k. *front-end*). Veebilehitseja kannab kasutaja tegevuse edasi päringutena serverisse. Server küsib andmebaasist vajaliku info, töötleb seda vastavalt ja sobitab info sobivale vaatele. Valmis vaade saadetakse tagasi veebilehitsejasse HTML failina. Selleks, et kasutaja uut infot näeks tuleb veebileht uuesti laadida ehk värskendada.

Üheleherakendus eemaldab serverilt esitlemise ülesannete kohustuse ja annab selle üle veebilehitsejale [1]. Kui varem vastutas sobiliku vaate valimise ja uue info sobitamise eest server siis nüüd vastutab selle eest veebilehitseja. Serveriga suheldakse põhiliselt vaid andmete saamise eesmärgil, mis muudab päringute tegemise kiiremaks.

Järgnevalt on toodud mõisted, mille tähenduse teadmine aitab paremini mõista, kuidas üheleherakendus töötab:

- põhi on rakenduse kasutajaliidesele põhjapanev HTML fail. Sisaldab ühte kas tühja  märgendit või regioonideks jaotatud  märgendit, kuhu sisestatakse vastavalt vaated;
- mall on DOM märgend, näiteks tekst või <*div*> märgend. Mallis on ka tavaliselt paika pandud kohatäited, kuhu pannakse tegelikud andmed kui õige aeg on käes;
- vaade on andmeid sisaldav mall, millega veebirakenduse kasutaja saab suhelda.

Kui kasutaja avab esimest korda üheleherakenduse salvestab veebilehitseja põhja ning kõik vajalikud mallid [1]. Põhi on esimene asi, mida kasutaja näeb. Tavaliselt on kirjutatud reeglid, mis määravad, millised vaated kasutajale kuvatakse. Edaspidi, vahetatakse vaateid vastavalt kasutaja tegevusele veebirakenduses. Vahetused toimuvad kiirelt ja ilma rakendust värskendamata, sest vajalikud vaated on juba varasemast veebilehitsejas olemas. Tavaliselt vastutab vaadete vahetamise ja andmetega sidumise eest MV* raamistik.

## 2.3 MV* disainimustrid

Kuna üheleherakenduste idee seisneb kohustuste lahususes ja kasutajaliides koosneb mitmetest erinevatest tükkidest siis on eriti kriitiline hoida projekti struktuur puhtana. Seetõttu kasutatakse kasutajaliidese haldamiseks MV* disaini mustritega raamistikke.

MV* struktuur viitab arhitektuuri mustritele, mis läbi rollide eraldamise võimaldavad tarkvara arhitektuuri paremini struktureerida [8]. Termin MV* esindab erinevaid kasutajaliidese arhitektuuri mustreid, kus tavaliselt täht "M" esindab mudelit, täht "V" esindab vaadet ning tärn on komponent, mis igal mustril on erinev.

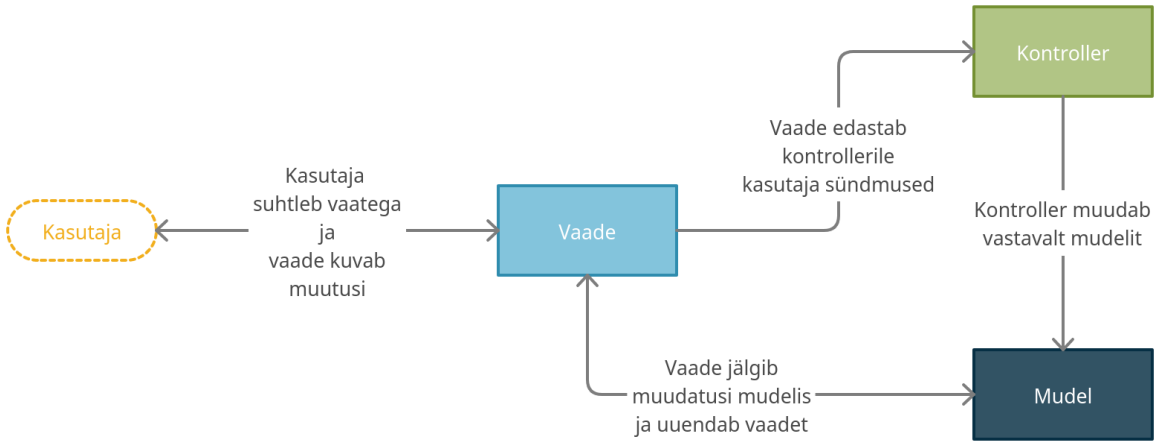
Järgnevalt on defineeritud mudeli ja vaate tähendus antud töö kontekstis:

- mudel hõlmab endas rakendusele omaseid andmeid, mida rakendus kasutab. Mudelid on taaskasutatavad. Ühel mudelil võib olla mitu eksemplari (ingl. k. *instance*) ja iga eksemplar sisaldab ühe unikaalse objekti andmeid. Mudeliks võib olla näiteks kasutaja andmed. Igal kasutajal on olemas salasõna ja kasutajanimi, kuid igal kasutajal on need reeglina erinevad.
- vaate puhul on tegemist kasutajaliidesega, mis kuvab mudelite andmeid eelnevas peatükis selgitatud mallis ja mille kaudu kasutaja rakendusega suhtleb.

MV* arhitektuuri mustrid võimaldavad tagaliidest (ingl. k. *back-end*) ja kasutajaliidest paralleelselt arendada. Lisaks lihtsustab mustri kasutamine koodi testimist ning mõistmist. Kõige levinumad mustrid on MVC, MVP ja MVVM [1].

### 2.3.1 MVC

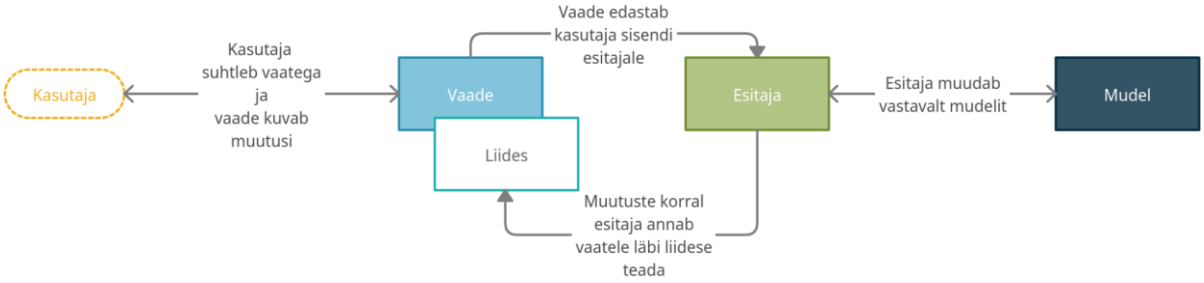
MVC disainimuster koosneb kolmest põhikomponendist: mudel, vaade ja kontroller (ingl. k. *controller*). Joonis 1 näitab, kuidas kasutaja sisend liigub läbi komponentide, et kuvada tehtud muudatused kasutajale. Kontroller teostab suhtlust kasutaja ning mudeli vahel. Läbi vaate võtab kontroller vastu kasutajapoolse signaali ja saadab vastava info muudatustest edasi mudelile. Vaade jälgib pidevalt, kas mudelis on toimunud muudatusi ja uuendab ennast vastavalt. Vaade kuvab muudatused kasutajale [1, 8]. Praktikas on mudeliks andmebaas või fail, mis hoiab andmeid, vaateks on HTML ja CSS kujundusega fail ning kontrolleriks JavaScripti ja HTML-iga kirjutatud fail [9].




*Joonis 1. MVC muster*

### 2.3.2 MVP

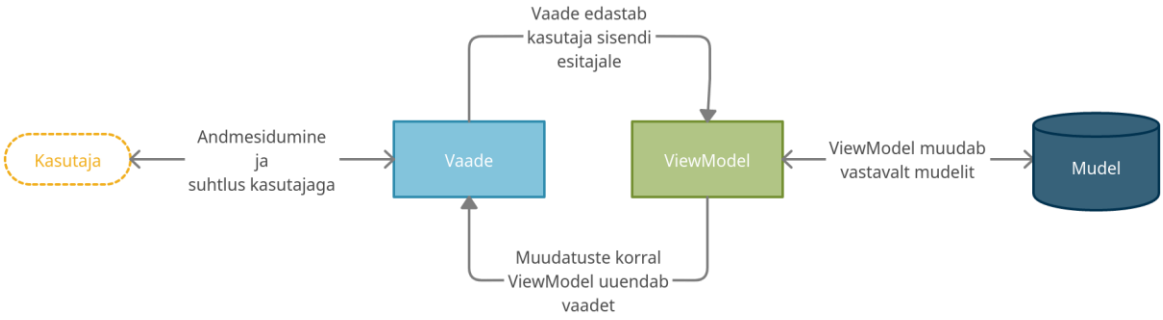
MVP puhul on tegemist MVC variatsiooniga. MVP disainimuster koosneb kolmest põhikomponendist: mudel, vaade ja esitaja (ingl. k. *presenter*). MVP lööb vaate ja mudeli täielikult lahku. Erinevalt MVC mustrist ei saa vaade enam mudeli muutusi järgida, seega on vaade selle mustri puhul passiivne. Esitaja on vaate ja mudeli vahendaja. Esitaja teab kõike vaate loogikast ning saab muuta mudeli sisu. Esitaja suhtleb vaatega läbi liidese. Vaade reageerib kasutaja suhtlusele ning annab info edasi esitajale. Esitaja muudab mudelit vastavalt vajadusele ja uuendab seejärel ühist kasutajaliidest [1].




*Joonis 2. MVP muster*

### 2.3.3 MVVM

MVVM katsub veelgi selgemini eraldada kasutajaliidest, äriloogikat ja rakenduse käitumist. MVVM disainimuster koosneb kolmest põhikomponendist: mudel, vaade ja esitamise mudel (ingl. k. *ViewModel*). Esitamise mudel sarnaneb esitajale, sest ta on samuti vahendaja vaate ning mudeli vahel. Esitamise mudel sisaldab nii vaate esitusloogikat kui ka mudeli andmete atribuute. Esitamise mudel ei suhtle vaatega läbi liidese vaid otse. Erinevalt esitaja ühendab ja sünkroniseerib esitamise mudel mudelit ja vaadet andme sidumise abil [1].




*Joonis 3. MVVM muster*

## 2.4 Raamistiku olemus

Raamistik on keerulisem struktuur, mis tavaliselt koosneb mitmest teegist ja annab arendajale kätte projekti struktuuri, kuhu arendaja sisestab enda koodi. Teegid ning raamistikud sisaldavad taaskasutatavaid valmis kirjutatud klasse, mis täidavad kindlat ülesannet [10]. Paika pandud struktuur muudab arendusprotsessi lihtsamaks, sest arendaja ei pea enam rakenduse arhitektuuri peale mõtlema ning saab keskenduda rakenduse funktsionaalsuse loomisele. Raamistik otsustab ise, mida on vaja teha. Näiteks JavaScripti kasutajaliidese raamistike puhul sisaldab raamistik keerulist JavaScripti koodi, mis ise vastutab komponentide vahetamise ja muutmise eest. Teegi puhul otsustab kasutaja, millal teeki kasutatakse ning kus.

## 2.5 Raamistikude valimine

Selles peatükis tutvutakse platvormidega, mille abil vaadeldakse hetkel levinumaid JavaScripti kasutajaliidese raamistikke. Kogutud informatsiooni põhjal valiti võrdlemiseks ja analüüsimiseks välja kolm populaarseimat kasutajaliidese raamistikku.

### 2.5.1 Nõuded analüüsitavale raamistikule

Antud bakalaureusetöös valitakse analüüsimiseks järgnevate tunnustega raamistikud:

- kliendipoolne rakendus,
- üheleherakenduste loomise võimalus,
- avatud lähtekood,
- suur kasutajaskond.

Open Source Initiative sihtasutuse sõnul on avatud lähtekoodi puhul tarkvara kood avalik kasutamiseks kõigile [11]. Nii saavad mitmed huvilised panustada selle tarkvara arendamisse ning parandamisse. Ühine tarkvaraarendus võimaldab luua turvalisemat ja kvaliteetsemat tarkvara [11]. Raamistiku lisa teegid peavad olema samuti avatud lähtekoodiga.

### 2.5.2 Raamistike valimine võrdluseks

Selles alapeatükis on "Best of JavaScript" projekti abil välja toodud 11. aprill 2021 a. seisuga levinumad JavaScripti kasutajaliidese raamistikud [12]. Iga raamistiku populaarsuse määramiseks tuuakse välja andmed GitHub'i platvormilt, Stack Overflow keskkonnast ning npm registrist.

GitHub on platvorm, kus on võimalik majutada projektide lähtekoodi [13]. GitHub kasutab versioonihaldust Git ning on loonud mitmeid teenuseid, et pakkuda kasutajatele paremaid koostöö võimalusi [13]. Platvormilt võib leida väga palju avaliku lähtekoodiga tarkvara. GitHub'is on võimalik tõmmata enda süsteemi koopia avatud lähtekoodiga projektist [14]. Seda tehakse peamiselt katsetamise ning edasi arendamise eesmärgiga. Selle tegevuse arvu kajastatakse termini "Fork" all. Lisaks võivad kasutajad anda projektile tähe. Tüüpiliselt annavad kasutajad projektile tähe tunnustuse näitamise eesmärgil. Tähega märgitud projektid pannakse kasutaja jaoks eraldi nimekirja, kus kohast saab kiiresti tunnustatud projekti üles leida [15]. Tähtede arvu kajastatakse platvormil termini "Star" all. Seetõttu saame GitHub'i andmeid kasutada populaarsuse määramiseks.

Node Package Manager (lühend npm) on JavaScripti keelele mõeldud paketihaldur [16]. Pakett on kaust või fail, mis sisaldab tarkvara ning lisa faili, mis kirjeldab ära tarkvara metaandmed. Npm hoiustab andmebaasis nii avalikke kui ka privaatseid pakette [16]. Selle abil on lihtne pakette tõmmata, hallata, uuendada ja eemaldada.

Selleks, et levinumate raamistike seast välja valida kõige populaarsemad võrreldakse omavahel järgnevaid tunnuseid:

- GitHub platvormil oleva projekti tähtede arv,
- GitHub platvormil oleva projekti "Fork"-de arv,
- GitHub platvormil oleva projekti viimase uuenduse kuupäev,
- raamistiku loomise aasta platvormil GitHub,
- Npm registrist raamistiku alla laadimiste arv,
- LinkedIn tööpakkumiste arv Eestis, mis sisaldab raamistiku nime märksõnana.

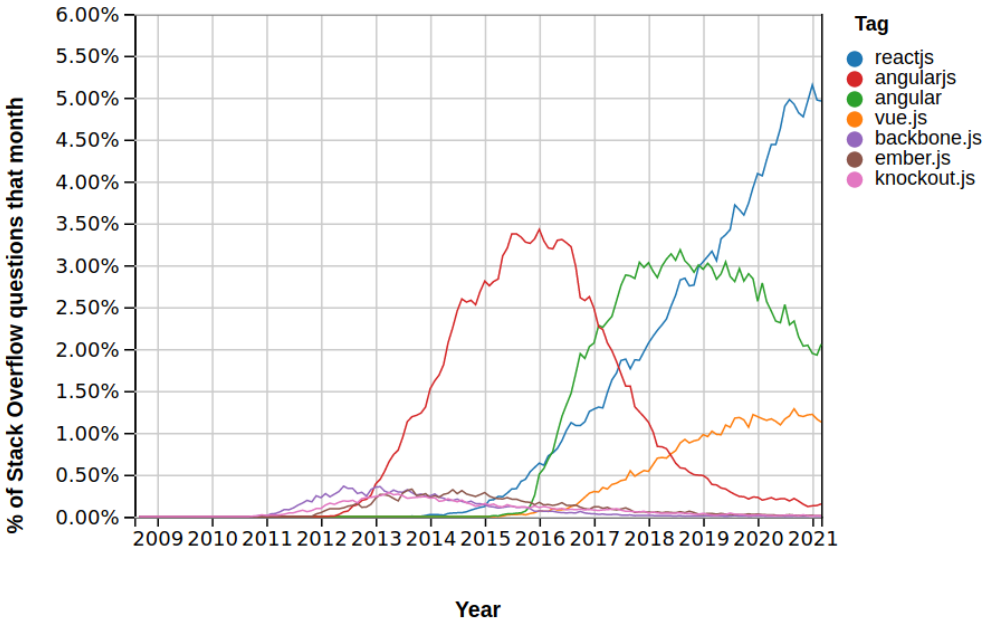
Tabelis 1 on esitatud 11 levinumat avatud lähtekoodiga kliendipoolset raamistiku, millega on võimalik luua üheleherakendusi. Tabelis on kajastatud ka eelnevalt mainitud tunnuste tulemused iga raamistiku kohta. Raamistike vue-next ning Vue.js pakkumised arvestatud kokku, sest vaadeldud tööpakkumised ei erista versiooni 2.x ja 3.x vahel. Keeruline olukord on ka raamistikega AngularJS ning Angular 2+. Tööpakkumistes kasutatakse tihti notatsiooni

“Angular”, mis ei pruugi viidata just Angular 2+ versioonile. Järgnevas tabelis on Angular 2+ tulemused saadud otsinguga “Angular 2+” ning AngularJS tulemused otsinguga “AngularJS”. Esmasel vaatel juba 5 tööpakkumist kattusid üksteisega.

Tabel 1. Levinumad JavaScripti kasutajaliidese raamistikud seisuga 11. aprill 2021 a.

| | GitHub’i tähti | GitHub’i forke | Viimane uuendus kuupäev | Loomis-kuupäev | Npm tõmbamisi 2020 a. jooksul [17] | LinkedIn tööpakkumiste arv (26.04.2021) |
|---|---|---|---|---|---|---|
| Vue.js (2.x) | 182 000 | 28 700 | 07.04.2021 | 2013 | 87 480 536 | 33 |
| React | 167 000 | 33 500 | 11.04.2021 | 2013 | 408 019 843 | 127 |
| Angular 2+ | 72 400 | 18 900 | 09.04.2021 | 2014 | 88 962 046 | 36 |
| AngularJS | 59 600 | 28 500 | 31.03.2021 | 2010 | 26 640 353 | 98 |
| Svelte | 46 000 | 2100 | 09.04.2021 | 2016 | 3 794 234 | - |
| Preact | 28 800 | 1600 | 06.04.2021 | 2015 | 16 548 969 | - |
| Backbone | 27 800 | 5600 | 04.05.2020 | 2010 | 26 645 296 | 15 |
| Ember | 21 800 | 4200 | 30.03.2021 | 2011 | 6 490 047 | 2 |
| vue-next (3.x) | 21 700 | 3600 | 07.04.2021 | 2018 | 87 480 536 | - |
| Alpine.js | 15 400 | 594 | 23.03.2021 | 2019 | 510 867 | - |
| Knockout | 9900 | 1600 | 28.02.2021 | 2010 | 2 078 157 | - |

Stack Overflow on veebiplatvorm, kus inimesed saavad küsida küsimusi ning leida vastuseid [18]. Arendajate jaoks on Stack Overflow väga oluline tööriist. JavaScripti teemalisi küsimusi on üle 2 miljoni [19]. Iga küsimus on märgistatud teemakohaste märksõnadega ja nende abil on võimalik näha statistikat kui aktiivselt raamistiku kasutajad küsimusi küsivad. Märksõnad, millel on alla 2000 küsimuse, ei ole joonisel 4 esindatud.




*Joonis 4. Küsimuste arv Stack Overflow platvormil raamistiku kaupa [20]*

GitHub'i tähtede, npm allalaadimiste ning Stack Overflow küsimuste arvu trendide järgi tulevad selgelt esile kolm raamistikku: Vue.js, React ja Angular. Nende vastu on kogukonnal kasvav huvi, Antud raamistikke uuendatakse tihti ning nende oskamist otsitakse ka Eesti tööturul. Tugevalt on langenud huvi vanemate raamistike vastu nagu AngularJS, Backbone, Ember ja Knockout. Backbone raamistikku pole uuendatud juba aasta aega kuigi veel leidub töökohti, mis nõuavad selle kasutusoskust. Vaikselt on tõusmas huvi uute raamistike vastu nagu Svelte, React'i põhjal ehitatud Preact ning Alpine.js, kuid siiski on nende allalaadimiste numbrid üsna väikesed võrreldes esikolmikuga. Lisaks ei ole nende raamistike küsimuste arv Stack Overflow platvormil ületanud 2000 lävendit.

Eespool kirjeldatut arvesse võttes analüüsitakse antud bakalaureusetöös järgnevaid JavaScripti kasutajaliidese raamistikke:

- Angular (Angular 2+),
- React,
- Vue.js.

Selleks, et välja toodud raamistikke mugavamalt õppida on vaja algteadmisi järgnevates tehnoloogiates: JavaScript, HTML ja CSS. Raamistikke lokaalselt kasutada on vaja installeerida Node.js ning npm.

### 2.5.3 Tähtsad aspektid raamistiku valimisel

Järgnevalt on toodud Emmit'i üheleherakenduste raamatu abiga välja tähtsad aspektid, millega tuleks raamistiku valimisel arvestada [1]:

1. **Populaarsus ja kogukond.** Kui raamistik on populaarne siis on sellel ilmselt suurem kogukond kui mõnel vähem populaarsel. Suure kogukonna puhul on väiksem võimalus, et kasutaja ei leia enda küsimusele vastust. Dokumentatsioonis ei ole alati kõik olukorrad kaetud ja kasutajal tuleb vastust otsida näiteks internetifoorumitest ja blogipostitustest. Suurem kasutajate arv tähendab suuremat võimalust, et ka teised on seda küsimust kuskil platvormil arutanud. Suurem kasutajaskond tähendab suuremat nõudlust raamistiku toetamise järgi.
2. **Raamistiku tugi.** Tuleb jälgida kes raamistiku eest hoolt kannab ning kuidas nad seda teevad. Kui tegemist on raamistikuga, millel on avatud lähtekood, siis on võimalik GitHub'i platvormilt sellist aktiivsust jälgida. Tasub ka tähele panna, kas raamistiku arendajad on paika pannud tulevikuplaanid.
3. **Litsents.** Tuleb arvestada, millise projekti jaoks raamistiku otsitakse. Tavakasutaja saab ilmselt hakkama tasuta tarkvaraga, kuid mõni kriitilisema teemaga projekt vajaks tasulist tarkvara. Algajale on oluline, et raamistik oleks tasuta kättesaadav ning vabalt kasutatav.
4. **Abistavad lisad/raamistikud.** Ühelt poolt on hea kui on olemas raamistiku sisesed abistavad teegid. Sellisel juhul on õppimiskurv väiksem, sest ilmselt on süntaks ja ülesehitus sarnane raamistikuga ning paigaldamine on kergem. Teiselt poolt on parem toetuda kolmanda osapoole lisadele, sest kui kindel teek peaks kaotama toetuse, siis on seda lihtsam välja vahetada uue vastu. Lisaks annab kolmanda osapoole moodulite kasutamine juurde paindlikkust, kasutaja saab ise valida, milline talle sobib. Seevastu tähendab, et kasutajal peab olema motivatsioon ja tahe õppida neid erinevaid mooduleid. Algaja jaoks on lihtsam esimene variant.
5. **Õppimiskurv ja dokumentatsioon.** Dokumentatsioon aitab kiiremini õppida raamistiku käsitlemist. Hea dokumentatsioon aitab lugejat kaasa koodinäidetega või interaktiivse demoga. Kuna tarkvara arendades tekib ikka vigu ning standardid muutuvad, siis toimub tarkvara pidev edasiarendamine. Seega tuleb hoida ka dokumentatsioon vastav, muidu on kasutajal keeruline raamistikku kasutada.

6. **Raamistike arhitektuur.** Raamistikud on ehitatud erinevalt, mõni jälgib MVC põhimõtteid. Teine on kokku seganud MVVM ja MVC ja kolmas kasutab hoopis muid põhimõtteid. Siin tekivad erinevused raamistike rakenduste struktuurides ja lahendustes. Arendajad on erinevad ja tasub katsetada, milline isiklikult kõige paremini sobib.

### 2.5.4 Testrakenduse loomine

Testrakenduse loomine on hea viis tarkvaraga tutvuda. Projektipõhine õppimine viib õppija sügavamale teema sisse, aitab praktilise kogemuse läbi kinnistada teooriat [21]. Testrakenduse loomine annab aimu sellest, kas raamistikuga töötamine on üldse sobiv. Lisaks võib see säästa palju aega ja ressursse, eriti kui on tegemist suurema ja keerukama projektiga. Isegi hobiprojekti puhul ei ole kerge arenduse jooksul tarkvara vahetada kui selgub, et midagi tarkvara puhul ei sobi või on puudu. Hobikorras arendajale võib sellise testrakenduse loomine olla kasuks ka tööle kandideerimisel. Isegi kui piirdutakse testrakenduse loomisega saab inimene näidata, et tal on varasem kogemus juba olemas.

### 2.5.5 Testrakenduse kirjeldus ning funktsionaalsused

Käesolevas bakalaureusetöös testrakenduse rollis on retseptikogumik. Rakendus on üheleherakenduse arhitektuuriga, mis tähendab, et see koosneb ühest kesksest konteinerist, mille sisu vahetub vastavalt kasutaja tegevusele. Rakendus kasutab tagaliidesega (ingl. k. *back-end*) suhtlemiseks REST (ingl. k. *Representation State Transfer*) standardit ning omab CRUD (ingl. k. *Create, Read, Update, Delete*)-funktsionaalsust.

Selleks, et demonstreerida üheleherakendusele omast dünaamilist sisu vahetamist ja kuvamist peab rakendus täitma järgnevaid funktsionaalsusi:

- kasutajale kuvatakse loodud retseptid nimekirjana,
- kasutaja peab saama listist valida retsepti, et näha rohkem retsepti detaile (pealkiri, valmistusaeg, kategooria, koostisosad, valmistusjuhis),
- kasutaja saab uusi retsepte juurde lisada,
- kasutaja saab retsepte kustutada,
- kasutaja saab retsepti muuta.

Praktilise osa eesmärk on luua kolm sama funktsionaalsuse ning sama välimusega retseptikogumikku kasutades kolme erinevat raamistikku. Retseptikogumiku disain on nähtav lisas 1. Ainuke erinevus saab olema kasutajaliidese raamistiku valik, ning selle

implementatsioon teiste teekidega. Eesmärgi täitmiseks kasutatakse kindlateks ülesanneteks järgnevaid teeke:

- tagaliidest imiteeritakse Mirage.js teegi abil,
- tagaliidesega suhtlemiseks kasutatakse raamistiku välist teeki Axios,
- marsruutimiseks ehk vaadete mugavaks vahetuseks kasutatakse raamistiku sisest teeki,
- CSS haldamiseks kasutatakse skriptimiskeelt SCSS.

Ka rakenduse struktuur peaks olemas võimalikult sarnane. Seega koosneb rakendus kindlatest komponentidest, milleks on:

1. tiitel ja retsepti lisamise vormile viiv nupp,
2. retseptide nimekiri,
3. nimekirja element, mis koosneb retsepti pealkirjast, kustutamise funktsiooniga nupust ning nupust, mis viib retsepti muutmise vormile,
4. retsepti lisamise vorm,
5. retsepti muutmise vorm,
6. retsepti detaile näitav ekraan.

## 2.6 Varasemalt tehtud tööd

Aastal 2016 kirjutas Erik Räni bakalaureusetöö teemal “JavaScripti raamistike võrdlus” [22]. Tema võrdles AngularJS, Backbone.js, React, Ember.js, KnockoutJS, Dojo ja Knockback raamistikke. Räni kasutas testrakenduse analüüsiks TodoMVC testrakendusi ning määras igale raamistikule koha pingereas. Pingerea loomiseks kasutas Räni erinevaid kriteeriumeid, milles omakorda olid tasemed. Iga tase andis mingi kindla punkti summa vahemikus null kuni kolm. Hinnang kolm oli kõige parem ning hinnang null kõige halvem. Kriteeriumiteks olid:

- startimishinnang,
- kogukond ja dokumentatsioon,
- arendustööriistade tugi,
- testimistugi,
- litsents äri vaatepunktist,
- keele ja vormingu tugi,
- TodoMVC testrakendusel põhinev keerukus,
- TodoMVC testrakendusel põhinev õppimine ja õppimiskiirus,
- TodoMVC testrakendusel põhinev jõudlus.

Suur rõhk oli töö autori enda kogemustel ning õppimiskiirusel. Mida suurema summa raamistik lõpuks kogus seda parem raamistik oli. Kõige suurema summa kogus AngularJS, mille ainukeseks negatiivseks küljeks oli aeglane õppimiskiirus.

Aastal 2015 kirjutas Jevgeni Rumjantsev magistritöö teemal "JavaScript raamistikude analüüs ja võrdlus struktureeritud SPA rakendustes" [23]. Kõigepealt selgitati välja kõige levinumad raamistikud. Analüüsiks valiti AngularJS, EmberJS, ReactJS ja BackboneJS. Aspektid, mida iga raamistiku puhul vaadeldi olid:

- komponentide kasutamine,
- arhitektuur ja selle loetavus ja haldamise paindlikkus,
- rakenduse kiirus,
- lihtsus arendamisel.

Lisaks loodi iga raamistikuga rakendus, mis koosnesid kogukonna poolt valmistatud levinumatest abikomponentidest. Näiteks AngularJS puhul kasutati komponente *ui-calendar*, *angular-google-maps* ja *angular-ui-bootstrap*. Lõpuks toodi välja raamistike plussid ning miinused ja lisati parimad praktikad rakenduse arendamiseks. Töö autor koostas tabeli, kus ta järjestas raamistikud enda eelistuse järgi ning esimeseks eelistuseks valis ta AngularJS raamistiku.

Mõlemast tööst on möödunud mitu aastat ja esile on kerkinud uued raamistikud. On kindlad fundamentaalsed aspektid, mida vaadeldakse ka antud bakalaureusetöös:

- dokumentatsioon ja õppimiskurv,
- populaarsus ja kogukond,
- raamistiku toetus,
- litsents,
- tööturg,
- rakenduse struktuur, koodi süntaks ja kasutuskogemus,
- ökosüsteem.

Erinevalt varasemalt tehtud töödest on võrdlus suunatud pigem algajatele. Bakalaureusetöö autor loob iga raamistikuga lihtsa, kuid päriselus kasutatava rakenduse, mis kasutab ainult vajalikke abiteeke. See eest bakalaureusetöö autor enda õppimiskiirust mõõdikuna ei kasuta, sest see ei ole usaldusväärne mõõdik. Selle asemel keskendutakse autori kasutuskogemusele.

# 3 Raamistike kirjeldus ja testrakenduse teostus

Selles peatükis kirjutatakse lähemalt Tausta peatükis välja valitud JavaScripti kasutajaliidese raamistikest. Kõigepealt tutvutakse iga raamistiku tausta ning eripäradega. Seejärel analüüsitakse tähtsaid aspekte ning kirjeldatakse testrakenduse teostust ning tuuakse raamistiku süntaksis näide.

Selleks, et järgnevaid raamistikke mugavamalt õppida on vaja algteadmisi järgnevates tehnoloogiates: JavaScript, HTML ja CSS. Selleks, et järgnevaid raamistikke lokaalselt kasutada on vaja installeerida JavaScripti käitussüsteem Node.js ning paketihaldur npm.

## 3.1 Angular

Angular on raamistik, mis võimaldab ehitada võimsaid üheleherakendusi [24]. Raamistiku arhitektuuris on toimunud palju suuri muutusi, mis on tekitanud kogukonnas palju segadust. Tihti aetakse segamini vanem versioon, mida kutsutakse nimega AngularJS ning uuem versioon, mida kutsutakse lihtsalt nimega Angular. Angular sai alguse aastal 2009 kui kahe Google töötaja hobiprojektina [25]. Esimene stabiilne versioon tuli välja aastal 2010. AngularJS oli üks esimesi raamistikke, mille struktuur põhines MVC arhitektuuril. Aastal 2016 tuli samade arendajate poolt välja raamistik Angular 2 [26]. Tegemist AngularJS ümber kirjutatud versiooniga, mis tegelikkuses oli üsna erinev vanast versioonist. Võrreldes vana raamistiku versiooniga kasutas uus versioon uut programmeerimis keelt, uut struktuuri ning uusi meetodeid. AngularJS ja Angular 2 olid nii erinevad, et mõlemat pidi tagasiühildus (ingl. k. *backwards compatibility*) oli pea võimatu.

Angular 2 (edaspidi Angular) ning sellega kaasaskäivad teegid on kirjutatud TypeScript'is. TypeScript on keel, mille aluseks on JavaScript [27]. TypeScript jagab JavaScripti süntaksit ning käitusaegset käitumist, kuid erinevalt JavaScriptist kontrollib TypeScript enne koodi jooksutamist tüüpe. Angular on üles ehitatud komponentide põhisele arhitektuurile, kus on oluline komponentide hierarhia [24]. Testprojekti struktuuri vaadates tuleb selgelt esile kohustuste lahususe printsiip.

### 3.1.1 Raamistiku põhimõisted

Järgnev peatükk on kirjutatud Angular'i dokumentatsiooni abil [24]. Angular'i rakendused koosnevad **komponentidest**. Iga Angular'i rakendus peab omama vähemalt juurkomponenti, mille sisse paigutatakse teised komponendid. Juurkomponent sisaldab teiste komponentide elementidest puud, mis sisestatakse veebilehte DOM märgenditena. Komponendis on määratud kasutajale näidatav HTML mall, selle stiil ning loogika osa, mis hoiab endas komponendile

omaseid andmeid ning kutsub välja teenuseid. Komponendid paigutatakse **moodulitesse** (ingl. k. *NgModules*). Nagu ka komponentide puhul peab rakendus omama vähemalt juurmoodulit. Moodul kirjeldab, kuidas antud komponenti õigesti kompileerida.

Iga komponent sisaldab **malli**, mis koosneb ekraanile kuvatavatest elementidest. Tegemist on HTML faililiga, mis kasutab spetsiifilist Angular'iga sobivat märgistuskeelt. Märgistuskeelega tulevad kaasa **direktiivid**, mis lisavad mallile juurde funktsionaalsust. Kõige levinumad on *ngIf* ja *ngFor*, mille abiga on võimalik mõjutada lõpliku DOM tulemust. Üks huvitav aspekt, mida Angular veel pakub on andmetoru. Andmetoru abil on võimalik kasutajale kuvatavad andmed muuta sobivale kujule. Näiteks on võimalik muuta kasutaja poolt sisestatud andmed väiketähestatuks enne edasist kasutamist.

Angular toetab kahesuunalist **andmetesidumist**, mis tähendab, et kasutaja muudatused veebilehe DOM märgendites kajastuvad automaatselt rakenduse sisestes andmeväljades. Peale selle on kolm ühesuunalist andmesidumismeetodit. Emakomponendilt (ingl. k. *parent-component*) tütarkomponendile (ingl. k. *child-component*) andmete saatmine, sündmuse saatmine tütarkomponendilt emakomponendile ja mallis mingi teenuses oleva andmevälja sisu kuvamine.

Igal komponendil on oma **elutsükkel**, mille igas etapis käivitatakse kindel meetod [28]. Nii on võimalik arendajal seda ära kasutada ja panna raamistik midagi komponendi elutsükli jooksul tegema. Näiteks komponendi laadimise (*ngOnInit*) ajal saab juba teha päringu andmebaasile, et saada kuvatavat infot.

### 3.1.2 Populaarsus ja kogukond

Võrreldes teiste töös analüüsitavate raamistikega on Angular'i puhul näha selget populaarsuse langustrendi. Langenud on küsimuste arv Stack Overflow platvormil ning madalad on ka GitHub'i tähtede ning forkide arv. Angular'i kodulehel on nimekiri erinevatest kohtadest, kus Angular'i huvilised saavad suhelda ja üksteist abistada. Angular'il oma foorum (ingl. k. *subreddit*) platvormil Reddit, kus oli töö kirjutamise hetkel 53 600 liiget [29]. Lisaks on ka kanal suhtlusplatvormil Discord, kus oli töö kirjutamise hetkel 8934 liiget [30]. Dokumentatsioonis on toodud välja mitu jutusaadet, kus räägitakse Angular'ist ning seotud teemadest ning märgitud on ka lähiajal toimuvad konverentsid ja töötoad [31]. Angular'il on blogi, kus jagatakse värskeid uudiseid ning nippe [32].

### 3.1.3 Raamistiku tugi

Hetkel arendab Angular'i suurkorporatsioon Google. Kuna tegemist on projektiga, mis on avatud lähtekoodiga ning üleval GitHub'i platvormil siis on see hea koht, kuskohast vaadelda raamistiku arengut. 26. aprilli 2021 a. seisuga on Angular'i projektil avatud probleeme (ingl. k. *open issues*) 2400 ning 204 lahendamata taotlust (ingl. k. *unresolved pull requests*). Kokku on juba suletud 20 126 probleeme ning taotlusi on vastu võetud või suletud 18 472. Antud projektiga on GitHub'is kaasa aidanud 1395 inimest [26].

### 3.1.4 Litsents

Angular on väljastatud MIT litsentsi all, mis lubab tarkvara kõigil kasutada, kopeerida, muuta, levitada [33].

### 3.1.5 Abistavad teegid

Angular'il on nende enda poolt välja arendatud korralik ökosüsteem, mis tuleb kaasa uue projekti loomisel. Näiteks Angular CLI, HttpClient, Router ja paljud muud [31]. Lisaks on olemas kogukonna poolt arendatud raamistikud, kuid Angular paneb rohkem rõhku nende endi poolt arendatud lahendustele. Kodulehel puudub viide suuremale teekide nimekirjale nagu on seda teistel raamistikel.

### 3.1.6 Dokumentatsioon ja õppimiskurv

Dokumentatsioon on üsna põhjalik [24]. Leidub nii pealiskaudsemaid kirjeldusi kui ka tehnilisemaid kirjeldusi tähtsamatest kontseptsioonidest. Angular'i meeskond on loonud juhendi, mis õpetab põhimõisteid ning selles luuakse ka lihtne rakendus [34]. Lisaks on olemas ka parimate harjumuste lehekülg, palju viiteid teistele lisaallikatele [31].

Õppimiskurv on Angular'i raamistikuga kõrgem. Kõigepealt tuleb selgeks teha TypeScript, et mugavalt arendada lokaalselt CLI tööriistaga loodud keskkonnas ning mõista dokumentatsioonis väljatoodud koodinäiteid. Projektiga tulevad kaasa mitmed erinevad moodulid ning teenused. Projekti struktuur on suur ning esialgu hirmutav. See eest toob selline kindel struktuur kaasa selle, et viise, kuidas ülesannet lahendada, on pigem vähe. Raamistik on mõeldud suuremate ja keerukamate projektide arendamiseks, mis muudab algaja jaoks õppimise keerukamaks. Õppimist raskendab fakt, et inimesed internetis ajavad Angular'i versioone omavahel sassi ning info otsimine internetist võtab kohati rohkem pingutust.

### 3.1.7 Testrakenduse teostus

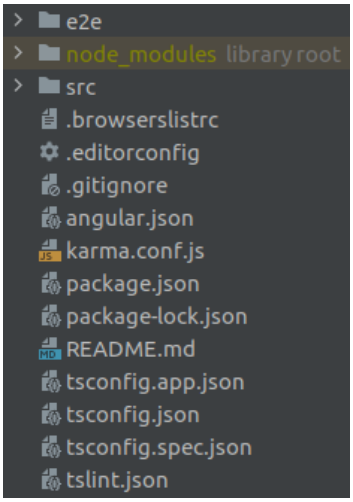
Järgnevates peatükkides kirjeldatakse täpsemalt Angular'i raamistikuga loodud testrakendust. Testrakenduse täielik lähtekood on leitav autori GitHub'i repositooriumist aadressil https://github.com/rebekkabb/angular-testrakenduslisas. Testrakenduse loomisel olid kasutusel 14.16.1 versiooni Node.js, 6.14.12 versiooni npm, 11.2.7 versiooni Angular CLI, 0.21.1 versiooniga Axios, 11.2.8 versiooniga Angular ning 0.1.41 versiooniga Mirage.js. Peale käsu "ng test" jooksutamist on projekti suuruseks 3.91MB. Kogu projekt koosneb 54 failist.

### 3.1.8 Projekti ülesseadmine

Esimese võimalusena saab Angular'i katsetada keskkonnas StackBlitz. Teine võimalus on luua Angular CLI tööriistaga lokaalne projekt ning arenduskeskkond. Angular CLI tuleb tõmmata läbi npm paketihalduri. Uus projekt luuakse käsuga "ng new projekti-nimi". Loomise käigus küsitakse, milliseid teeke ja skriptimiskeeli projektile kohe juurde lisada. Hiljem saab teeke juurde lisada kasutades käsku "npm install teegi-nimi" [35].

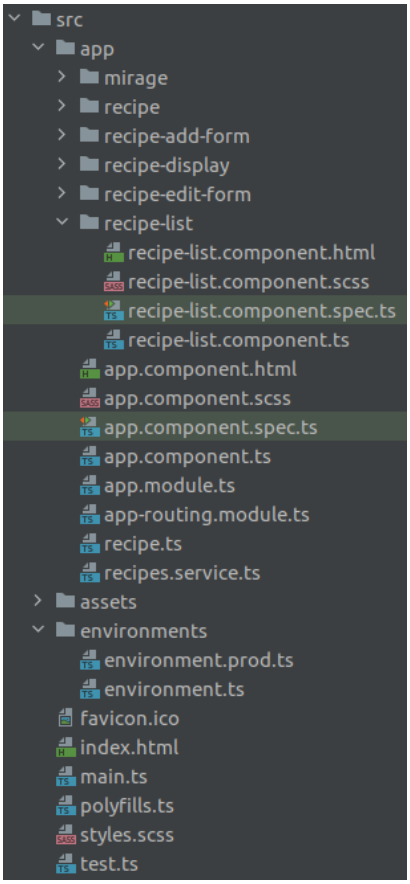
### 3.1.9 Projekti struktuur

Angular CLI loodud projekt on tihedalt sisustatud. Joonisel 5 ja 6 on näha suurt hulka faile. Failid *.gitignore*, *README.md* on seotud versioonihaldusega. Failid *package.json*, *package-lock.json*, *angular.json* sisaldavad konfiguratsiooni ning kasutatavate teekide informatsiooni. Ülejäänud failid joonisel 5 on seotud TypeScript'i, testimise ning kompileerimisega.




*Joonis 5. Testrakenduse struktuur ilma src kausta sisuta*

Joonisel 6 on nähtav kausta *src* sisu. Sellesse kausta luuakse iga komponendi jaoks oma kaust. See alamkaust sisaldab komponendi HTML malli, CSS kujundus fail, testimiseks mõeldud faili ning loogika faili. Lisaks on *src* kaustas rakenduse moodulid, ressursside kaust ning Mirage.js teegiga seotud failid.




*Joonis 6. Testrakenduse src kausta sisu*

### 3.1.10 Süntaks komponendi näitel

Selles peatükis vaadatakse komponenti *RecipeList*, et tutvuda lähemalt Angular'i süntaksiga ning tuua näiteid eelnevalt välja toodud teooriale. Nagu eelnevalt mainitud koosneb komponent mallist, kujundusest ning loogika osast. Joonisel 7 on näha, et komponendi mall sisaldab põhiliselt klassikalisi DOM märgendeid peale <*app-recipe*> elemendi, mis on lapskomponent, mis esindab retsepti nimekirja kirjet. Märgendis <*div*>, mille klassiks on *recipes*, on kasutatakse retsepti nimekirja kuvamiseks direktiivi **ngFor*. Selle direktiivi abil luuakse igale retsepti kirjele nimekirjas üks <*app-recipe*> isend, mis sisaldab vastavalt ainult selle retsepti andmeid. Lisaks on näha marsruutimisel kasutatav direktiiv *routerLink*, mis täpsustab, et antud <*a*> märgend juhatab kasutaja otspunkti /*add*.

```
<div class="recipe-list-container">
  <div class="header">
    <h1> Retseptid </h1>
    <a routerLink="/add" class="submit-button button-block">Lisa retsept</a>
    <hr/>
  </div>
  <div class="recipes">
    <div *ngFor="let recipe of recipes">
      <app-recipe [recipe]="recipe"></app-recipe>
    </div>
  </div>
</div>
```

*Joonis 7. Komponendi RecipeList HTML mall*

Komponendi *RecipeList* loogika failis kõigepealt imporditakse vajalikud teegid ning teenused. Joonisel 8 on näha, et antud komponendil on vaja paari Angular'i põhifunktsiooni, funktsioone *RecipeService* teenusest ning retsepti mudelit. Seejärel täpsustakse komponendi osad *@Component* all. Kujundus määratakse eraldi CSS failis ning on täiesti tavapärane seega sellest eraldi joonist vaja ei ole. Joonis 8 pealt on näha, kuidas kujundus määratakse loogika osas väärtuse *styleUrls* all. Nii teab komponent, milline kujundus mallile lisada. Mall on täpsustatud väärtuse *templateUrl* all. Väärtuses *selector* on kirjeldatud, kuidas komponenti teistes mallides välja kutsuda. Komponendi klassi sees on defineeritud retseptide nimekiri, mis saadakse *RecipeService* teenuse abiga. Antud komponent demostreerib elutsükli meetodi *ngOnInit* kasutust, et *RecipeService* teenuselt paluda andmebaasist võtta retseptide nimekiri.

```
import {Component, OnInit} from '@angular/core';
import {RecipesService} from '../recipes.service';
import {Recipe} from '../recipe';

@Component({
  selector: 'app-recipe-list',
  templateUrl: './recipe-list.component.html',
  styleUrls: ['./recipe-list.component.scss']
})
export class RecipeListComponent implements OnInit {
  recipes: Recipe[] = [];

  constructor(private recipesService: RecipesService) {
  }

  ngOnInit(): void {
    // this.listRecipes();

    this.fillRecipes();
  }

  // listRecipes(): void {
  //   this.recipesService.listRecipes((recipes: Recipe[]) => {
  //     this.recipes = recipes;
  //   });
  // }

  async fillRecipes(): Promise<void> {
    this.recipes = await this.recipesService.listRecipesAsync();
  }
}
```

*Joonis 8. Komponendi RecipeList teenuse fail*

Jooniselt 8 võib tähele panna, et siin on mitu rida välja kommenteeritud koodi. Välja kommenteeritud koodi näol on tegemist teise viisiga, kuidas teenuses väljakutsutavast meetodist saada andmed kätte. Teenusest andmete kättesaamine osutus töö autorile rakenduse loomisel kõige raskemaks ülesandeks. Ülesande täitmist raskendas põhiliselt TypeScript'i vähene oskus. Komponendiga tuleb kaasa testimiseks mõeldud fail, kuid seda antud töös sügavamalt ei käsitleta. Malli fail koosneb 12 loogikat sisaldavad koodireast. CSS fail koosneb 46 loogikat sisaldavad koodireast. Loogika fail sisaldab 20 rida koodi, millest 14 on loogikat sisaldavad koodiread ning ülejäänud on seotud koodi vormistusega (sulgude lõpp jms). Kokkuvõttes koosneb *RecipeList* 78 koodireast, millest 72 sisaldavad loogikat.

## 3.2 React

React on JavaScripti kasutajaliidese teek, millele lõi alguse Jordan Walke. Laiemale avalikkusele avaldati teek 2013 aasta juulis [36]. React'i abil on võimalik luua taaskasutatavaid kasutajaliidese komponente, mis on võimelised esitama muutuvaid andmeid. Kuigi React on tehniliselt teek, siis kogukond loeb seda siiski teiste populaarsete JavaScripti raamistikkude hulka, sest seda on võimalik kasutada täpselt samal moel nagu neid teisi raamistikke [37]. React on komponendi põhine ning paindlik [38]. Teeki on võimalik kasutada nii vähe kui vaja või nii palju kui vaja. See tähendab, et teegiga on võimalik luua terve kasutajaliidese projekti või saab piirduda pisikese kasutajaliidese osa loomisega ja lisada see enda olemasolevasse kasutajaliidesesse.

### 3.2.1 Raamistiku põhimõisted

Järgnev peatükk on kirjutatud React'i dokumentatsiooni „Main concepts" alapeatüki põhjal [38]. React kasutab erilist JavaScript laiendus süntaksit, mida kutsutakse nimega **JSX**. See ei ole kohustuslik, aga lihtsustab koodi lugemist ning kirjutamist. Iga React'i elemendi loomisel kasutatakse React'i meetodit *createElement*. **Elemendiks** loeme kas DOM märgendit või mõnda teist React komponendi isend. Lihtsa <*div*> märgendi loomise süntaks ilma JSX süntaksita on järgnev: "React.createElement('div')". Kuna on teada, et iga element luuakse mainitud meetodiga, siis JSX lihtsustab koodi järgnevalt: " ". Tulemus sarnaneb HTML süntaksile, aga kuna tegemist on JavaScriptiga, siis saab kasutada JavaScriptiga kaasatulevaid avaldisi. React'i elemente ei saa muuta, seega vaate uuendamisel luuakse uus element ning antakse edasi React DOM'ile.

React kasutab vaate uuendamisel teeki ReactDOM. Teek vastutab veebilehe DOM eeskirja uuendamise eest elementidele põhinedes. Põhimõtteliselt loob ReactDOM **virtuaalse DOM** seisu, et selle põhjal muudatusi võrrelda. React võrdleb seise kõrvutamise algoritmi abil [39]. Nii uuendab React vaid neid DOM märgendeid, mis on muutunud. Teised märgendid jäävad nii nagu nad on ning neid ei värskendata, mis kiirendab kogu protsessi.

**Komponent** on iseseisva seisuga taaskasutatav tükike kasutajaliidest. Komponent pannakse kokku eelmainitud React elementidest. Põhimõtteliselt on komponent funktsioon, mis võtab sisse *props* väärtused ja tagastab React elementide puu, mis kirjeldab suuremas pildis, mida tahetakse ekraanile kuvada. Väärtusteks võivad olla andmed või sündmuste kuulajad.

Komponente on kahte tüüpi: funktsiooni põhine (ingl. k. *function based*) ja klassipõhine (ingl. k. *class based*). Mõlemad täidavad täpselt sama eesmärki. Nagu näha joonistel 9 ja 10, ainuke

erinevus on nende süntaks, kus üks on defineeritud funktsioonina ja teine klassina, mis sisaldab meetodit *render*. Enne 16.8 uuendust ei saanud funktsioonipõhine komponent kasutada olekumuutujaid nagu klassipõhine komponent. Uuendusega tulid kaasa haagid (ingl. k. *hooks*), mis lubavad kasutada olekumuutujaid ka funktsioonipõhises komponendis.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

*Joonis 9. Komponent JavaScripti funktsioonina* [38]

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

*Joonis 10. Komponent JavaScripti klassina* [38]

Emakomponendi ja tütarkomponendi vaheline suhtlus on **ühesuunaline** ning toimub läbi *props* väljade. Komponendi sisene andmete suhtlus toimub samuti ühesuunaliselt. React'is on reegel, et **olekumuutujaid** ei tohi otse muuta, vaid tuleb kasutada *setState* meetodit. Meetodi *setState* kutsumine annab React'ile teada, et antud olekumuutuja on muutunud ning kutsub välja *render* meetodi, mis elementide puu tagastamisel sisaldab ka uuendatud muutujat. Tulemusena jõuab uuendus kasutaja ekraanile. Tütarkomponent saab emakomponendiga suhelda läbi sündmuste kuulaja. Kui tütarkomponendis toimub midagi, mida emakomponent ootab, saadetakse signaal üles. Signaaliga saab kaasa anda väärtuseid, näiteks *id* täpsustamiseks.

Igal komponendil on määratud **elutsükkel**, mis sisaldab kindlaid etappe. Igas etapis käivitatakse kindel meetod. Arendajal on võimalus sisestada enda koodi nendesse etappidesse. Enamasti kasutatakse meetodeid *constructor()*, *render()*, *componentDidMount()*, *ComponentDidUpdate()* ja *componentWillUnmount()*. Elutsükli diagramm on nähtav lisas 3.

### 3.2.2 Populaarsus ja kogukond

React on väga populaarne. Peatükis "Raamistike valimine" on näha, et aastal 2020 oli React'il enim allalaadimisi paketihalduris npm ning ka tööpakkumisi Eesti tööturul. Küsimuste arv on Stack Overflow platvormil on iga aastaga kasvanud ning on hetkel teistest kõige kõrgem. React'il on oma kanal suhtlusplatvormil Discord, kus oli töö kirjutamise hetkel 149 288 liiget [40]. Samuti on React'il oma foorum (ingl. k. *subreddit*) platvormil Reddit, kus oli 253 000 liiget [41]. Need on kohad, kus arendajad saavad omavahel suhelda ning üksteiselt abi küsida.

### 3.2.3 Teegi tugi

React'i arendust juhib väikene 8 liikmeline meeskond Facebookis. Kuna tegemist on projektiga, mis on avalik kõigile siis abistavad arendusega ka mitmed entusiastid üle maailma. Antud projektiga on GitHub'is kaasa aidanud 1542 inimest [36]. 25. märtsi 2021 a. seisuga on React'i projektil GitHub'i platvormil 530 avatud probleemi (ingl. k. *issues*) ja 180 taotlust lisada probleemi lahendust koodibaasi (ingl. k. *pull request*). Kõige viimane taotlus aktsepteeriti 24 märtsil 2021 a. Kokku on suletud või kinnitatud 10 644 taotlust ja 9735 probleemi. React'i ametlikult lehel on ka blogi, kuhu postitatakse informatsiooni nii ametliku meeskonna poolt hallatavatest lisateekidest kui ka React'i enda uutest versioonidest ja plaanidest.

### 3.2.4 Litsents

React on väljastatud MIT litsentsi all, mis lubab tarkvara kõigil kasutada, kopeerida, muuta, levitada [33].

### 3.2.5 Abistavad teegid

React'i ametlik arendustiimi on loonud paar tähtsamat teeki nagu *create-react-app*, *react-dev-tools*, *redux* ning *react-router*. Kodulehel ei ole täpselt välja toodud teisi nende poolt loodud teeke. GitHub'is on aga olemas nimekiri erinevate kogukonna poolt loodud teekidega [42]. Neid teeke leidub sadu kui mitte tuhandeid.

### 3.2.6 Dokumentatsioon ja õppimiskurv

React'i dokumentatsioon on väga põhjalik [38]. React'i dokumentatsioonis olemas kaks erineva tasemega juhendit. Ühe juhendiga tutvustatakse React'i põhiomadustest veidi pealiskaudsemalt ning luuakse nullist testrakendus. Selle juhise jooksul õpetatakse algajatele ka häid tavasid. Teine juhend räägib põhiomadustest sügavamalt koos koodinäidetega. Dokumentatsioon sisaldab ka eraldi sõnastikku, kus võib leida selgitusi igasugustele meetoditele või mõistetele.

Teegi õppimine on väga lihtne eriti peale praktilise juhise läbimist. Positiivne on see, et teeki saab katsetada hästi mitmel viisil, ei pea kohe täis projekti looma. Lisa teadmisi ei pea omama, et seda raamistikku kasutada. React'iga loodava projekti struktuur ei ole liiga suur. Veidi peab ette mõtlema ja mõistma, kuidas andmed komponentide vahel liikuma hakkavad, et meetodid õigesse komponenti panna.

### 3.2.7 Testrakenduse teostus

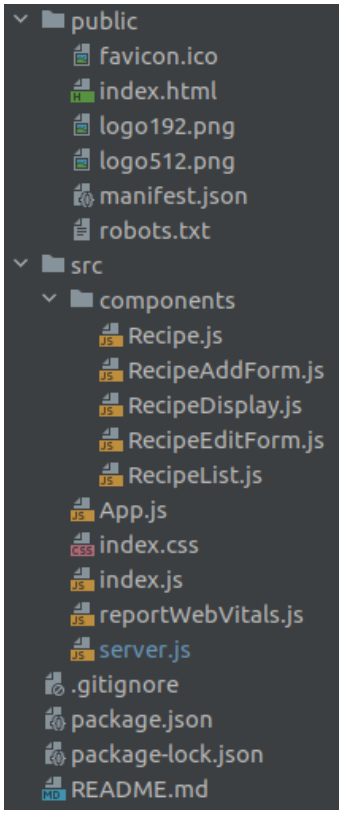
Järgnevates peatükkides kirjeldatakse täpsemalt React'i raamistikuga loodud testrakendust. Testrakenduse täielik lähtekood on leitav töö autori GitHub'i repositooriumist aadressil https://github.com/rebekkabb/react-testrakendus. Testrakenduse loomisel olid kasutusel 14.16.1 versiooni Node.js, 6.14.12 versiooni npm, 4.0.3 versiooni Create React App, 0.21.1 versiooniga Axios, 17.0.02 versiooniga React ning 0.1.41 versiooniga Mirage.js. Peale käsu "npm run build" jooksutamist luuakse rakendusest optimiseeritud versioon ning selles suuruseks on 2.2MB. Kogu projekt koosneb 20 failist.

### 3.2.8 Projekti ülesseadmine

React'i on võimalik katsetada erinevate veebi koodimängumaadega nagu *CodePen*, *CodeSandbox* või *Stackblitz*. Võimalus on kirjutada React'i komponente lokaalselt HTML lehe <script> tagi sisse ja kasutada neid komponente enda koodis DOM komponentidena. Nii saab lisada juba olemasolevale rakendusele interaktiivsust. Järgmine võimalus on luua erinevate tööriista ahelate abil lokaalne rakendus, mis juba algusest peale toetub React'ile. Erinevateks otstarbeteks on olemas erinevad tööriista ahelad. React'i dokumentatsioon soovitab üheleherakenduste loomise jaoks kasutada "Create React App" keskkonda. "Create React App" loob kasutajaliidese projekti malli, mida on võimalik kasutada kasutajale sobiva tagaliidesega. Peale malli loomise seab "Create React App" üles ka arendus keskkonnal, mis sisaldab kõike vajalikku, et arendusprotsess oleks lokaalselt mugav ja kiire. Teeke saab projektile juurde lisada kasutades käsku "npm install teegi-nimi" [38].

### 3.2.9 Projekti struktuur

Testrakenduse projekti struktuur on nähtav joonisel 11. Kaustas *public* asub fail *index.html*, mis on üheleherakenduse põhi ning mitmed pildifailid. Fail *index.html* kuvatakse kasutajale esimesena ning selle faili HTML malli sisse sisestatakse vastavalt vaateid ehk andmetega komponente. Joonis 5 allosas on näha versioonihalduse jaoks vajalikud failid ja projekti seadistusfailid. Failis *package.json* on kirjeldatud kõik teegid ja paketid, mida projekt kasutab. Projekti tuum asub kaustas *src*. Kaustas *components* on kõik komponendid, millest kasutajaliides kokku pannakse. Fail *server.js* sisaldab Mirage.js teegi jaoks vajalikku koodi. Kõige olulisemad failid on *App.js* ja *index.js*. Failis *App.js* määratakse ära juur komponent, antud komponent sisaldab ka marsruutimise koodi, mis määrab ära, milline komponent kindla otspunkti korral kuvatakse. Failis *index.js* määratakse, et *App.js* failis sisalduv komponent saab olema juur komponent. Viimaks faili *index.css* näol on tegemist CSS failiga, mis sisaldab kogu rakenduse kujundust.




*Joonis 11. React testrakenduse struktuur*

### 3.2.10 Süntaks komponendi näitel

Selles peatükis vaadatakse komponenti *RecipeList*, et tutvuda lähemalt React'i süntaksiga ning tuua näiteid eelnevalt välja toodud teooriale. Nagu eelnevalt mainitud koosneb komponent mallist, kujundusest ning loogika osast. Reacti's on kujunduse osa paigutatud eraldi CSS faili. Selles CSS failis on defineeritud kogu rakenduse kujundus. Joonisel 12 on näha vaadeldava komponendi faili, mis on JavaScripti fail ning sisaldab malli ja loogikat. Tegemist on funktsioonipõhise komponendiga. Faili alguses defineeritakse impordid. Antud komponendil on vaja tütarkomponenti *Recipe* ning ligipääsu marsruutimise teenuse sisule. Mall koosneb DOM märgenditest ning React elemendist <*Recipe*>. Lisaks on mallis kasutusel React element <*Link*>, millele klikkides suunab React kasutaja otspunktile */add*.

Retseptide nimekirja kuvamiseks on kasutatud nimekirja peal meetodit *map()*, mis loob iga kirjega uue *Recipe* isendi. Nii *RecipeList* komponent kui ka *Recipe* saavad endaga kaas *props* väärtuseid ning sündmusekuulajad. *RecipeList* komponent saab väärtuse *recipes*, mis on nimekiri retseptidest, ning *Recipe* saab väärtuse *currentRecipe*, mis määrab ära konkreetse isendi poolt kuvatavad andmed. Mõlemal komponendil on sündmusekuulajad *onDelete* ja *switchComponent*. Kui tütarkomponendis (*Recipe)* kutsutakse välja meetod nimega *onDelete*,

siis tuleb see signaal *RecipeList* komponendile koos kaasa antud parameetritega ja annab selle edasi emakomponendile (*App).*

```
import Recipe from "./Recipe";
import {Link} from "react-router-dom";

const RecipeList = ({recipes, onDelete, switchComponent}) => {
    return (
        <div className='recipe-list-container'>
            <div className="header">
                <h1>Retseptid</h1>
                <Link to='/add'>
                    <span className='submit-button button-block'>Lisa retsept</span>
                </Link>
                <hr/>
            </div>

            <div className='recipes'>
                {recipes.map((recipe) => (
                    <Recipe
                        key={recipe.id} currentRecipe={recipe} onDelete={onDelete} switchComponent={switchComponent}/>
                ))}
            </div>
        </div>
    )
}

export default RecipeList
```

*Joonis 12. Komponendi RecipeList loogika ning mall*

Selleks, et näha ka loogika osa paremini tuleb vaadelda juurkomponenti *App*, sest kogu loogika on sinna tõstetud. Joonisel 13 on näha elutsükli meetodi kasutust. Kuna ka *App* on funktsioonipõhine komponent siis kasutatakse *componentDidMount()* ja *componentDidUpdate()* asemel *useEffect()* meetodit. Nii saadetakse komponendi esmasel laadimisel kohe päring andmebaasi retseptide listi järgi. Meetodis *loadRecipes()* kasutatakse *set()* meetodit, et päringu vastus salvestada olekumuutujasse *recipes*.

```
function App() {

    const [recipes, setRecipes] = useState([])

    useEffect(() => {
        loadRecipes();
    }, []);

    const loadRecipes = () => {
        axios.get('/api/recipes/').then(res => {
            setRecipes(res.data.recipes)
        })
    }
```

*Joonis 13. Komponendi App loogika*

Kuna rakenduse kujundus on kõik ühes failis siis valime sealt välja ainult need väärtused, mida kasutatakse komponendis. Kujunduse fail koosneb sel juhul 57 koodireast, millest 44 on loogikat sisaldavad koodiread ja ülejäänud on seotud süntaksi vormistusega. Loogika ning malli fail sisaldab 23 rida koodi, millest 19 on loogikat sisaldavad koodiread, ülejäänud on seotud süntaksi vormistusega. Kokkuvõttes koosneb *RecipeList* 80 koodireast, millest 63 sisaldavad loogikat.

## 3.3 Vue.js

Aastal 2016 intervjuus Vivian Cromwelliga tõi Vue.js looja Evan You välja, kuidas ta Google 'is töötades puutus kokku raamistikuga Angular [43]. You leidis, et raamistikuga Angular kaasnesid mõningad piiritlused. You alustas Vue.js kirjutamist, eesmärgiga võtta Angular'i parimad osad ja eemaldada piiritlused, mis teda raamistiku puhul häirisid. Peale selle rõhutas You intervjuus, et Vue.js pöörab eraldi tähelepanu ka ligipääsetavusele, et kasutajad, kellel on juba veebiarenduse põhitehnoloogiate baas olemas, õpivad raamistiku kasutuse kiirelt ära. Aastal 2013 avaldas You enda projekti laiemale kasutajaskonnale platvormil GitHub [44].

Vue.js (edaspidi Vue) dokumentatsioon nimetab raamistikku progressiivseks JavaScripti raamistikuks. Esiteks on raamistikku lihtne sisestada juba olemasolevasse lahendusse. Raamistik on ehitatud viisil, mis võimaldab raamistikku rakendada järk-järgult. Teiseks on võimalik luua raamistikuga projekt, kus saab algusest peale tuua varasemalt serveri poolel olnud loogikat kasutajaliidesesse. Vue disain on osaliselt saanud inspiratsiooni MVVM arhitektuuri mudelist. Seetõttu võib Vue isenditega tegeledes kohata koodis muutujat "vm" (*view-model)* [45].

Hetkel on võimalik valida Vue 2 ja Vue 3 vahel. Antud bakalaureusetöös on analüüsimiseks valitud Vue 2, sest suurem osa Vue 3 ökosüsteemist (pistikmoodulid ja muud teegid) ei ole veel piisavalt välja arenenud või vastavalt uuendatud.

### 3.3.1 Raamistiku omadused

Järgnev peatükk on kirjutatud Vue dokumentatsiooni põhjal [46]. Vue on ehitatud **komponendi** põhisele arhitektuurile. See tähendab, et rakendused koosnevad väikestest taaskasutatavatest tükkidest, mida nimetatakse komponentideks. Joonisel 14 on näha klassikalist Vue komponenti. Komponent koosneb kolmest sektsioonist: HTML osa, skripti osa ja CSS osa. HTML osas kirjeldatakse komponendi HTML elemendid. Skripti osas määratakse erinevad atribuudid. Näiteks komponendi sisesed andmed (ingl. k. *data*), teisest komponendist

tulevad andmed (ingl. k. *props*), meetodid (ingl. k. *methods*), elutsükli meetodid (ingl. k. *lifecycle hooks*) ja palju muud. CSS osas määratakse komponendi kujundus.


*Joonis 14. Vue komponendi näidis [47]*

Vue raamistikus on kõik **reaktiivne**, seega DOM märgendid ning rakenduse andmed on omavahel sünkroonis. Andmete muutumisel uuendab Vue automaatselt DOM märgendeid, et see kajastaks tehtud muudatust. Raamistikuga tulevad kaasa ka **direktiivid**, mis lisavad DOM märgendile reaktiivset käitumist. Vue direktiive märgitakse eesliitega "v-".

Kasutajale kuvatava veebilehe DOM uuendamiseks kasutab Vue **virtuaalset DOM**'i. Virtuaalne DOM on sisu poolest väiksem kui veebilehe tegelik DOM, sest kogu komponentide puu salvestamise asemel hoiab see endas vaid muudetud komponente. Vue kasutab virtuaalset DOM'i, et olla teadlik muutustest, mida ta peab DOM'is tegema.

Komponentide vaheliselt rakendatakse **ühesuunalist andmesidumist** (ingl. k. *one-way binding*). Andmete vahetamiseks kasutatakse atribuuti *prop*. Selle abil on võimalik saata andmeid emakomponendist tütarkomponendile. Teistpidi saab andmevahetust teostada *$emit()* abil. Komponendi siseselt võimalik rakendada **kahesuunalist andmesidumist** (ingl. k. *two-way binding*). Näiteks on ankeedi väljale võimalik lisada direktiiv "v-model". Antud direktiiv uuendab vastavalt kasutaja sisendile skripti osas olevat andmevälja ja see sama sisend kuvatakse kasutajale andmeväljas.

Igal Vue rakendusel on juur **Vue isend**. Samuti on iga komponent Vue isend. Kõik koos moodustavad nad hierarhia, juur Vue isendi sees on teised Vue isendid. Vue isendi loomisel jooksutatakse isendi elutsükliga seotud funktsioone (ingl. k. *lifecycle hooks*). **Elutsükli** diagramm on nähtav lisas 4. Vue isendi loomisest alates käivitatakse järjest meetodid *beforeCreate(), created(), beforeMount(), mounted(), beforeUpdate(), updated(), beforeDestroy(), destroyed()*. Igas etapis on arendajal võimalik kutsuda välja enda meetodeid või tegevusi. Näiteks komponendi loomisel (ingl. k. *create*) saadetakse tagaliidesele päring, et saada kasutajate nimekiri ja siis need andmed *mounted* seisus kasutajale kuvada.

### 3.3.2 Populaarsus ja kogukond

Vue populaarsus on tõusutrendil ning inimesed suhtuvad sellesse positiivselt, ka kogukond on aktiivne. Peatükis "Raamistike valimine" on just Vue'l kõige rohkem GitHub'i tähti. Lisaks on Vue'l rohkem küsimusi platvormil StackOverFlow kui Angular'il ning pea sama palju allalaadimisi. Tuleb meeles hoida, et Angular on siiski suur korporatsiooni poolt toetatud aga Vue ei ole. Vue'l on oma foorum ning kanal suhtlusplatvormil Discord, kus oli töö kirjutamise hetkel 95 924 liiget [48]. Peaaegu iga kuu avaldatakse ka ametlikud uudised seoses raamistiku ja selle ökosüsteemiga ning jagatakse infot tähtsate sündmuste kohta (konverentsid, töötoad, lisaallikate allahindlused) [49].

### 3.3.3 Raamistiku tugi

Erinevalt eelnevast kahest raamistikust ei ole Vue raamistikul suur korporatsioonist omanikku. Raamistikul on olemas aktiivne arendustiim, mida juhib Evan You. Raamistikul on üle 40 sponsori. 21. märtsi 2021 a. seisuga on Vue projektil GitHub'i platvormil 331 avatud probleemi (ingl. k. *issues*) ja 195 taotlust lisada probleemi lahendust koodibaasi (ingl. k. *pull request*). Kõige viimane taotlus aktsepteeriti 19 märtsil 2021 a. Kokku on suletud või kinnitatud 1800 taotlust ja 9074 probleemi. Antud projektiga on GitHub'is kaasa aidanud 399 inimest [44].

Platvormil on ka olemas tegevuskava sellest, mis tulevasesse versiooni juurde lisatakse ning kirjas on ka plaanid pikemas perspektiivis [44].

### 3.3.4 Litsents

Vue on väljastatud MIT litsentsi all, mis lubab tarkvara kõigil kasutada, kopeerida, muuta, levitada [33].

### 3.3.5 Abistavad teegid/raamistikud

Vue ametlik arendustiimi on loonud tähtsamad teegid [44]:

1. *vue-router* on teek, mis aitab korraldada marsruutimist üheleheküljelistele rakendustele,
2. *vuex* on teek, mis aitab hallata olekuid,
3. *vue-cli* on teek, mille abiga on võimalik luua Vue projekti mall,
4. *vue-loader* on teek *Webpack* jaoks,
5. *vue-server-renderer* on teek serveripoolne *rendering support,*
6. *vue-class-component* on teek TypeScript'i jaoks,
7. *vue-rx* on teek *RxJS* integreerimiseks,
8. *vue-devtools* on veebilehitseja laiendus,

Peale ametliku arendustiimi teekide leidub sadu kogukonna poolt arendatud teeke [50].

### 3.3.6 Dokumentatsioon ja õppimiskurv

Dokumentatsioon on sisukas koos koodinäidetega, kuigi puudub juhend koos projekti loomisega nagu teistel [46]. Õppimiskurv on lihtne, projekti struktuur on väga selge ja kompaktne, komponentide struktuur on samuti kompaktne ning kood on arusaadav, meetodid on selgete nimetustega. Ka Vue puhul on positiivne, et raamistikku saab proovida ilma täis projekti tegemata.

### 3.3.7 Testrakenduse teostus

Järgnevates peatükkides kirjeldatakse täpsemalt Vue raamistikuga loodud testrakendust. Testrakenduse täielik lähtekood on leitav töö autori GitHub'i repositooriumist aadressil https://github.com/rebekkabb/vue-testrakendus. Testrakenduse loomisel olid kasutusel 14.16.1 versiooni Node.js, 6.14.12 versiooni npm, 4.0.3 versiooni Vue CLI, 0.21.1 versiooniga Axios, 2.6.11 versiooniga Vue ning 0.1.41 versiooniga Mirage.js. Peale käsu "npm run build" jooksutamist luuakse rakendusest pakitud versioon ning selles suuruseks on 2.4MB. Kogu projekt koosneb 16 failist.

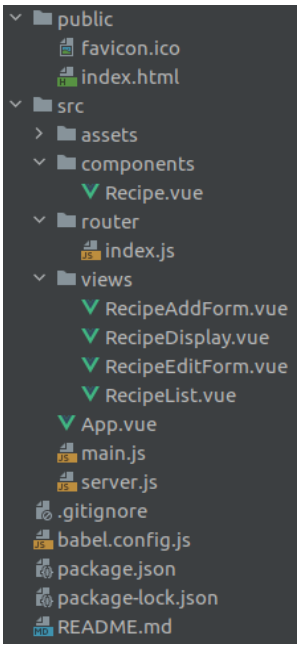
### 3.3.8 Projekti ülesseadmine

Vue installeerimiseks on mitu võimalust. Järgnevad kolme meetodit on kõik saadud Vue enda dokumentatsioonist [46]. Algajatele soovitatakse lisada Vue enda rakendusse HTML märgendi <*script*> abil. Suuremate projektide jaoks soovitatakse tõmmata raamistik npm abil. Viimaks on olemas ka tööriist Vue CLI ehk võimalus käsurea abil seadistada esmane projekt üheleheküljelise rakenduse jaoks. Seda meetodit ei soovitata algajatele, kellele võivad

projektiga kaasatulevad eripärad tundmatud olla. Vue CLI tuleb installeerida läbi npm-i ja seejärel “vue create projekti-nimi” käsuga luuakse projekti struktuur. Loodud projekt sisaldab endas erinevaid teeke, et oleks võimalik kiirelt ja mugavalt lokaalselt rakendust arendada. Näiteks seadistatakse automaatselt *hot reload*, mis tähendab, et muudatused koodis kajastuvad automaatselt veebilehitsejas. Eelmainitud boonuste tõttu kasutati ka Vue testrakenduse loomisel Vue CLI võimalust. Teekide lisamine sõltub projekti loomise viisist. Kui Vue lisati projekti HTML märgendi <*script*> abil, siis saab teeke lisada kasutades käsku “npm install teegi-nimi”. Vue CLI-ga loodud projektidele lisatakse teeke käsuga “vue add teegi-nimi”.

### 3.3.9 Projekti struktuur

Testrakenduse projekti struktuur on nähtav joonisel 15. Kaustas *public* asub fail *index.html*, mis on üheleherakenduse põhi. Fail *index.html* kuvatakse kasutajale esimesena ning selle faili HTML malli sisse sisestatakse vastavalt vaateid ehk andmetega komponente. Joonis 15 allosas on näha versioonihalduse jaoks vajalikud failid ja projekti seadistusfailid. Failis *package.json* on kirjeldatud kõik paketid, mida projekt kasutab. Projekti tuum asub kaustas *src*. Kaustas *assets* hoitakse näiteks erinevaid pildifaile või svg elemente. Kaustades *components* ja *views* on kõik komponendid, millest kasutajaliides koosneb. Tegelikult võib kõiki komponente hoida ka samas kaustas. See on täielikult kasutaja enda otsustada. Kaustas *router* on fail, mis sisaldab marsruutimiseks vajalikku koodi. Seal on kirjas, milline komponent kindla otspunkti korral kasutajale kuvatakse. Fail *server.js* sisaldab Mirage.js teegi jaoks vajalikku koodi. Fail *main.js* sisaldab koodi, millega tehakse juur Vue isend.




*Joonis 15. Vue testrakenduse struktuur*

### 3.3.10 Süntaks komponendi näitel

Selles peatükis vaadatakse komponenti *RecipeList*, et tutvuda lähemalt Vue süntaksiga ning tuua näiteid eelnevalt välja toodud teooriale. Joonisel 16 on nähtav komponendi *RecipeList* HTML koodi osa. Kogu HTML kood on <*template*> märgendi sees ning omakorda jaotatud erinevateks klassideks. Koodis on esimesena näha märgendi <*router-link*>. See märgend tuleb Vue marsruutimise teegist. Märgendile on määratud väärtus *to*, mis määrab otspunkti, kuhu link viib. Antud juhul on tegemist "/add" otspunktiga. Nagu eelmises peatükis sai mainitud, siis kaustas *router* on fail, mis määrab, millist komponenti kindla otspunkti puhul kasutajale kuvatakse.

```
<template>
  <div class="recipe-list-container">
    <div class="header">
      <h1> Retseptid </h1>
      <router-link to="/add" class="submit-button button-block"> Lisa retsept</router-link>
      <hr/>
    </div>
    <div class="recipes">
      <div v-for="recipe in recipes" :key="recipe.id">
        <Recipe :recipe="recipe"
                v-on:delete="deleteRecipe(recipe)"></Recipe>
      </div>
    </div>
  </div>
</template>
```

*Joonis 16. Komponendi RecipeList HTML osa*

Märgendi <*div*> sees on klass *recipes*, *kus* on näha Vue direktiivi *v-for* kasutamist. Direktiivi on kasutatud selleks, et sisestada iga retsepti andmed tütarkomponenti *Recipe*. Komponendil *Recipe* on kasutatud *v-on* direktiivi, mis reageerib DOM sündmustele. Antud juhul direktiiv käivitab meetodi *deleteRecipe()* kui ta saab tütarkomponendilt signaali märksõnaga *delete* ja retsepti *id* väärtuse.

Joonisel 17 on nähtav komponendi skripti osa. Kood asub märgendi <*script*> vahel. Alguses on kõik vajalikud impordid ja seejärel erinevad väljad, kus saab hoiustada andmeid, seise ja meetodeid. Antud näites välja *data* all hoitakse retseptide nimekirja. Välja *methods* all defineeritakse meetodid nagu eelnevalt mainitud *deleteRecipe()* meetod, mis vastutab retsepti kustutamise eest. Nagu jooniselt 17 võib näha on siin ka meetod *listRecipes()*, mida kutsutakse välja nii peale *deleteRecipe()* käivitamist ning ka elutsükli meetodi *created()* ajal. Nii tehakse kindlaks, et kasutaja näeb kõige värskemat infot nii rakenduse esmasel laadimisel kui ka peale retsepti kustutamist.

```
<script>
import Recipe from "../components/Recipe";
import axios from 'axios';

export default {
  name: 'RecipeList',
  components: {
    Recipe,
  },
  data() {
    return {
      recipes: []
    }
  },
  methods: {
    listRecipes() {
      axios.get('/api/recipes')
          .then(function (response) {
            this.recipes = response.data.recipes
          }.bind(this));
    },
    deleteRecipe: function (recipe) {
      axios.delete('/api/recipes/' + recipe.id).then(() => {
        this.listRecipes()
        alert('Retsept on kustutatud!')
      })
    },
  },
  created() {
    this.listRecipes()
  },
}
</script>
```

*Joonis 17. Komponendi RecipeList skripti osa*

Vue’s on kõik komponendi osad ühes failis. See fail koosneb 96 koodireast, millest 77 on loogikat sisaldavad koodiread ja ülejäänus on seotud süntaksi vormistusega. Malli osa koosneb 15 koodireast, millest 15 on loogikat sisaldavad. Skripti osa koosneb 32 koodireast, millest 23 sisaldavad loogikat. Kujunduse osa koosneb fail koosneb 49 koodireast, millest 39 on loogikat sisaldavad koodiread.

# 4 Tulemused

Selles peatükis analüüsitakse eelnevalt kirjutatud teoreetilisi aspekte ning praktilisi lähenemisi, tuuakse välja iga raamistiku positiivsed ja negatiivsed küljed ning tehakse järeldused.

## 4.1 Angular

Angular on komponendipõhine raamistik nagu ka teised analüüsis osalevad raamistikud. Angular'i komponentide sisu jaguneb tükkideks, seega on rakenduse struktuuriga vähe mängimisruumi. Seda on näha ka projekti ülesseadmisel. Angular'i saab kasutada vaid kindlates ettemääratud keskkondades. CLI tööriistaga projekti loomine oli mugav, sest erinevalt teistest pakkus see võimaluse kõik vajalikud teegid kohe üles seadistada. Samas tuleb Angular'i projektiga kaasa ka palju muud, mida algaja kohe ei tunne või ei soovi kasutada enda rakenduses. Testrakendus koosnes 54 failist ning pakituna oli see 3.9MB. Testrakendus oli teistest lahendustest ligi kaks korda suurem. See tekitas Mirage.js teegi lisamisel probleeme, sest teegi lisamiseks oli vaja keerukas struktuur selgeks teha. Kahjuks puudus teegil Angular'is installeerimise jaoks juhend. Lisaks on Angular'i kasutamiseks vaja selgeks õppida ka TypeScript. Kokkuvõttes kujuneb raamistiku õppimiskurv algaja jaoks üsna kõrgeks. Angular ei ole mõeldud väikesteks projektideks, vaid pigem suureks ning keerukateks, mida Angular aitab hästi oma tükeldatusega hallata. Angular omaette pakub rohkem funktsionaalsust kui teised töös vaadeldud raamistikud ning selle ümber on arendatud ökosüsteem põhiteekidest. Autori arvamusel on Angular'i projekti tegemisel omad tugevused ning nõrkused (vt. tabel 2).

Tabel 2. Angular'i tugevused ning nõrkused

| **Tugevused** | **Nõrkused** |
|---|---|
| Olulisemad teegid on juba sisse ehitatud ning tulevad projektiga kaasa. | Analüüsitavate seast kõige vähem populaarne ja väikese kogukonnaga. |
| Põhjalik CLI, teeb projekti ülesseadmise veelgi lihtsamaks. | Kõrge õppimiskurv. (nõuab TypeScript'i tundmist) |
| Suurkorporatsiooni poolt toetatud (Google). | Projekt on suur, tuleb kaasa lisadega, mida ei pruugi vaja olla. |
| Raamistikku veebirakenduse koodi saab kasutada mobiilirakenduste loomisel. | Vana ning uue versiooni nimetuste segamini ajamine võib mõjutada arendusprotsessi. |
| Tööturul on raamistiku järgi nõudlust. | Pole eriti paindlik. |
| Juhendiga dokumentatsioon | |

Angular ei ole enam nii populaarne nagu ta oli paar aastat tagasi, kogukond on samuti vähenenud. Samas on mitmeid firmasid, kes kasutavad enda toodetes Angular'i. Kindlust lisab see, et raamistiku arendab suurkorporatsioon Google. Seega raamistiku toetamist kindlasti jätkatakse. Angular'i arendajatel on tulevikuplaanid nii raamistikule kui ka ümbritsevale ökosüsteemile. Kogukond hoitakse blogide, foorumite ning teiste suhtluskeskkondade abil edasiarendustega kursis. Segadus versioonide nimetusega võib mõjutada arendusprotsessi ja tööotsinguid, sest ei ole selge kas viidatakse raamistikule AngularJS või Angular.

## 4.2 React

React'i kasuks räägib selle populaarsus ja aktiivne kogukond. Samuti on ka raamistiku arendajad aktiivsed ning hoiavad kogukonda kursis läbi suhtluskanalite, blogi ning foorumite. React'i saab lisada olemasolevasse projekti, kasutada koos teise kasutajaliidese raamistikuga või luua kogu kasutajaliides React'i põhjal. Kuna React on nii paindlik, siis on sellel ilma abiteekideta vähem funktsionaalsust kui Vue'l või Angular'il (*two-way databinding* jms*)*. Seetõttu toetub React tugevalt oma massiivsele ökosüsteemile, mis koosneb ametlikest teekidest ning tuhandetest kolmanda osapoole teekidest. Ühelt poolt on see hea – kui mõni teek peaks kaotama toetuse, saab leida sellele kiirelt asenduse. Samas nõuab selline paindlikkus kasutajalt suurt motivatsiooni neid teeke omavahel kombineerida ning õppida. Autori arvamusel on React'i projekti tegemisel omad tugevused ning nõrkused (vt. tabel 3).

Tabel 3. React'i tugevused ja nõrkused

| **Tugevused** | **Nõrkused** |
|---|---|
| Suur ökosüsteem. | Limiteeritud funktsionaalsus ilma kolmanda osapoole teekideta. |
| Aktiivne kogukond, hetkel kõige populaarseim raamistik. | Suure ökosüsteemi kasutamine nõuab motivatsiooni ja tahet õppida uut tarkvara. |
| Suurkorporatsiooni poolt toetatud (Facebook). | |
| Raamistiku veebirakenduse koodi saab kasutada mobiilirakenduste loomisel. | |
| Hea dokumentatsioon koos juhenditega (erinevad tasemed). | |
| Tööturul kõige enim vaja. | |
| Väga paindlik raamistiku struktuuri poolest kui ka ökosüsteemi poolest. | |

Loodud testrakendus oli väike ja koosnes vaid 20 failist ning pakituna oli see 2.2MB ning Mirage.js teegi lisamiseks oli ka olemas juhised.

## 4.3 Vue.js

Vue on populaarsuselt kohe peale React'i. Tegemist on raamistikuga, millel on väga aktiivne kogukond ning kirega arendav arendusmeeskond. Vue'l on võrreldes teistega GitHub'is kõige rohkem tähti ning arendusmeeskond hoiab kogukonda uudistega kursis läbi mitmete suhtluskanalite, blogi, foorumite ja ürituste. See on muljetavaldav, sest raamistikul ei ole suurkorporatsioonist toetajat, vaid sõltub mitmetest sponsoritest. Hiljuti tuli välja raamistiku järgmine versioon nimega Vue 3. Seega on põhjust arvata, et raamistiku toetus ei lõppe nii pea.

Vue raamistik on üsna sarnane React'ile. Mõlemad on paindlikud, intuitiivsed, tuginevad sarnastele põhimõistetele (*virtual DOM)* ning on lihtsa õppimiskurviga. Vue'd on samuti võimalik kasutada koos teiste kasutajaliidese raamistikega või saab luua kogu kasutajaliidese Vue peale. Erinevalt React'ist sisaldab Vue ka omadusi Angular'ist (*two-way data binding)*. Seega on Vue'l ilma abiteekideta rohkem funktsionaalsust kui React'il ja vähem kui Angular'il. Autori arvamusel on Vue projekti tegemisel omad tugevused ning nõrkused (vt. tabel 4).

Tabel 4. Vue tugevused ja nõrkused

| **Tugevused** | **Nõrkused** |
|---|---|
| Olulisemad teegid on ametliku tiimi poolt välja arendatud. | Ei ole toetatud suurkorporatsiooni poolt, tugineb sponsoritele. |
| Pakub omaette rohkem funktsionaalsust kui React, kuid on lihtsam kui Angular. | Dokumentatsioonis puudub projektiga juhend. |
| Väga paindlik nii raamistiku struktuuri poolest kui ka ökosüsteemi poolest. | Ei ole tööturul veel nii nõutud kui teised. |
| Kogukonna kasvav huvi ning väga aktiivne kasutajaskond. | |

Loodud testrakendus oli samuti väike ja koosnes vaid 16 failist ning pakituna oli see 2.4MB. Vue jaoks olid samuti olemas Mirage.js teegi poolt loodud juhised. Projekti struktuur oli arusaadav ning puhas. Komponendid on kompaktsed, sest nii HTML, CSS kui ka loogika on ühes failis, mis on algajale mugav. Dokumentatsiooni koodinäited põhjalikud ning relevantsed. Negatiive külg on see, et dokumentatsioonis puudub projektiga juhend ja Vue ei ole veel tööturul nii nõutud, kuigi seda küsitakse järjest enam.

## 4.4 Järeldused

Analüüsis kerkisid eelkõige esile React ja Vue. Mõlemat on intuitiivne kasutada. Projekti loomine oli mugav ning vajalike teekide lisamine oli samuti lihtne. Mõlemad raamistikud on populaarsed ning nende kogukonnad on aktiivsed. Mõlemad tunduvad sobivat algajale hästi. Raamistikud annavad ruumi katsetusteks ja arenguks, kuid ei koorma arendajat üle liigse informatsiooniga. Angular ei tundu parima esimese valikuna, sest see on mõeldud keerukamateks projektideks ja võrreldes teistega on selle õppimiskurv kõrgem.

Bakalaureusetöö autori arvates on nendest kolmest parimaks valikuks Vue, sest raamistikku on lihtne õppida, kogukond on aktiivne ja raamistik toetatud. Lisaks kasutab Vue omadusi, mis on nii React'is kui ka Angular'is. Seega on peale Vue õppimist lihtsam teha endale selgeks ka teised kaks raamistikku. See eest kui algajale on oluline olla konkurentsivõimeline tööturul, siis tasuks mõelda React'i õppimisele. Suurt rolli mängivad ka algaja eelistused koodi süntaksi kohta ning dokumentatsiooni keelekasutuse kohta.

# 5 Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli võrrelda ja analüüsida populaarseimaid JavaScripti kasutajaliidese raamistikke. Tulemusena tekkis ülevaatlik materjal, mis aitab algaja arendajal valiku tegemisel langetada informeeritud otsus.

Bakalaureusetöös toodi välja tähtsamad aspektid raamistiku valimisel, milleks on populaarsus ja kogukond, raamistiku tugi, litsents, abistavad teegid, dokumentatsioon ja õppimiskurv ning testrakenduse loomine. Koostati ülevaade hetkel levinumatest JavaScripti kasutajaliidese raamistikest ja valiti analüüsimiseks raamistikud Angular 2+, React ja Vue.js. Tutvuti valitud raamistike taustaga, analüüsiti neid vastavalt eelnevalt kirjeldatud aspektidele ja loodi iga raamistikuga lihtne päriselus kasutatav testrakendus. Töös esitati raamistike positiivsed ning negatiivsed küljed ning toodi välja raamistike erinevused ning sarnasused. Lisaks tutvustati töö jooksul mitmeid JavaScripti kasutajaliidestega seotud teemasid.

Läbiviidud analüüsi põhjal töö autor peab parimaks valikuks algaja jaoks Vue raamistik, sest see sisaldab omadusi nii Angular'ist kui ka React'ist. Lisaks on raamistiku populaarsus kasvamas, kogukond on aktiivne ning arendustiim toetab seda aktiivselt. Peale Vue õppimist on algajal olemas hea baas, et järgmisena selgeks õppida React ja Angular.

Tulevikus võiks sarnase võrdluse läbi viia vähem tuntud raamistikega nagu Svelte ja Alpine.js. Selgitada välja nende eripärad ja kasutusvõimalused. Edasiarendusena tasuks lasta mitmel algajal luua Angular 2+, React ja Vue raamistikkudega samamoodi testrakendused ja teostada selle informatsiooni põhjal analüüs.

## Kasutatud kirjandus


1. Emmit A. Scott Jr. SPA Design and Architecture: Understanding single-page web applications. New York: Manning Publications; 2016.
2. History of the Web – World Wide Web Foundation. https://webfoundation.org/about/vision/history-of-the-web/ (07.12.2020)
3. Rauschmayer A. Chapter 4. How JavaScript Was Created. 2015. http://speakingjs.com/es5/ch04.html (07.12.2020)
4. Rauschmayer A. Chapter 5. Standardization: ECMAScript. 2015. http://speakingjs.com/es5/ch05.html (07.12.2020)
5. Mozilla ja individuaalsed kaastöötajad. A re-introduction to JavaScript (JS tutorial) - JavaScript | MDN. 2005. https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript (07.12.2020)
6. Mozilla ja individuaalsed kaastöötajad. JavaScript - MDN Web Docs Glossary: Definitions of Web-related terms | MDN. 2005. https://developer.mozilla.org/en-US/docs/Glossary/JavaScript (07.12.2020)
7. Ajax - Developer guides | MDN. https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX (18.04.2021)
8. Osmani. A. Learning JavaScript Design Patterns. 2017. https://addyosmani.com/resources/essentialjsdesignpatterns/book/#detailmvcmvp (18.04.2021)
9. MVC - MDN Web Docs Glossary: Definitions of Web-related terms | MDN. https://developer.mozilla.org/en-US/docs/Glossary/MVC (18.04.2021)
10. Kumar S. Difference Between Library and Framework. 2020. https://www.c-sharpcorner.com/uploadfile/a85b23/framework-vs-library/ (8.12.2020)
11. About the Open Source Initiative | Open Source Initiative. https://opensource.org/about (16.04.2021)
12. Best of JavaScript. https://bestofjs.org/projects?tags=framework&sort=total (16.04.2021)
13. GitHub. Hello World · GitHub Guides. https://guides.github.com/activities/hello-world/ (16.04.2021)
14. GitHub. Fork a repo - GitHub Docs. https://docs.github.com/en/github/getting-started-with-github/fork-a-repo (16.04.2021)
15. GitHub. Saving repositories with stars - GitHub Docs. https://docs.github.com/en/github/getting-started-with-github/saving-repositories-with-stars (16.04.2021)
16. About npm | npm Docs. https://docs.npmjs.com/about-npm (16.04.2021)
17. npm-stat: download statistics for NPM packages. https://npm-stat.com/ (16.04.2021)
18. Stack Overflow - Where Developers Learn, Share, & Build Careers. Stack Overflow. https://stackoverflow.com/ (16.04.2021)
19. Newest “javascript” Questions. Stack Overflow. https://stackoverflow.com/questions/tagged/javascript (16.04.2021)

20. Stack Overflow Trends. https://insights.stackoverflow.com/trends?tags=reactjs%2Cvue.js%2Cangular (16.04.2021)
21. What Is Project-Based Learning Method for Self-Taught Developers | Hacker Noon. https://hackernoon.com/what-is-project-based-learning-method-for-self-taught-developers-97gi3vey (20.04.2021)
22. Räni E. JavaScripti raamistike võrdlus. TÜ arvutiteaduse instituudi bakalaureusetöö; 2016.
23. Rumjantsev J. JavaScript raamistikude analüüs ja võrdlus struktureeritud SPA rakendustes. TTÜ informaatika instituudi magistritöö; 2015.
24. Google. Angular - Introduction to Angular concepts. 2010. https://angular.io/guide/architecture (09.12.2021)
25. What about Angular? Angular JS history through the years (2009-2019). https://www.ryadel.com/en/angular-angularjs-history-through-years-2009-2019/ (18.04.2021)
26. GitHub. Angular GitHub repositoorium. https://github.com/angular/angular (08.12.2020)
27. Microsoft. Documentation - TypeScript for the New Programmer. 2012. https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html (09.12.2020)
28. Google. Angular - Lifecycle hooks. https://angular.io/guide/lifecycle-hooks (06.05.2021)
29. Reddit. Angular (2+). https://www.reddit.com/r/Angular2/ (06.05.2021)
30. Discord. Join the Angular Community Discord Server! https://discord.com/invite/angular (06.05.2021)
31. Google. Angular - EXPLORE ANGULAR RESOURCES. https://angular.io/resources (06.05.2021)
32. Google. Angular Blog. https://blog.angular.io/ (06.05.2021)
33. The MIT License | Open Source Initiative. https://opensource.org/licenses/MIT (21.04.2021)
34. Google. Angular - Tour of Heroes app and tutorial. https://angular.io/tutorial (06.05.2021)
35. Google. Angular - Getting started with Angular. https://angular.io/start (06.05.2021)
36. GitHub. React GitHub repositoorium. https://github.com/facebook/react/releases/tag/v0.3.0 (26.04.2021)
37. What is React? Is it a framework? https://start-up.house/en/blog/articles/what-is-react (06.05.2021)
38. Facebook Inc. Getting Started – React. https://reactjs.org/docs/getting-started.html (06.05.2021)
39. Facebook Inc. Virtual DOM and Internals – React. https://reactjs.org/docs/faq-internals.html (06.05.2021)
40. Discord. Join the Reactiflux Discord Server! https://discord.com/invite/reactiflux (06.05.2021)

41. Reddit. /r/ReactJS - The Front Page of React. https://www.reddit.com/r/reactjs/ (06.05.2021)
42. GitHub. Brillout R. brillout/awesome-react-components. 2021. https://github.com/brillout/awesome-react-components (06.05.2021)
43. Cromwell, V. Between the Wires: An interview with Vue.js creator Evan You. freeCodeCamp.org. 2017. https://www.freecodecamp.org/news/between-the-wires-an-interview-with-vue-js-creator-evan-you-e383cbf57cc4/ (09.12.2020)
44. GitHub. Vue.js GitHub repositoorium. https://github.com/vuejs/vue/releases (10.12.2020)
45. You. E. The Vue Instance — Vue.js. https://vuejs.org/v2/guide/instance.html (21.04.2021)
46. You. E. Introduction — Vue.js. https://vuejs.org/v2/guide/ (18.04.2021)
47. You. E. Single File Components | Vue.js. https://v3.vuejs.org/guide/single-file-component.html#introduction (21.04.2021)
48. Discord. Join the Vue Land Discord Server! https://discord.com/invite/HBherRA (06.05.2021)
49. You. E. Official Vue.js News. https://news.vuejs.org (21.04.2021)
50. GitHub. vuejs/awesome-vue. 2021. https://github.com/vuejs/awesome-vue (06.05.2021)

# Lisad

## Lisa 1. Testrakenduste lähtekoodid

Valminud testrakenduste lähtekoodid on saadaval autori GitHub platvormil aadressidel:

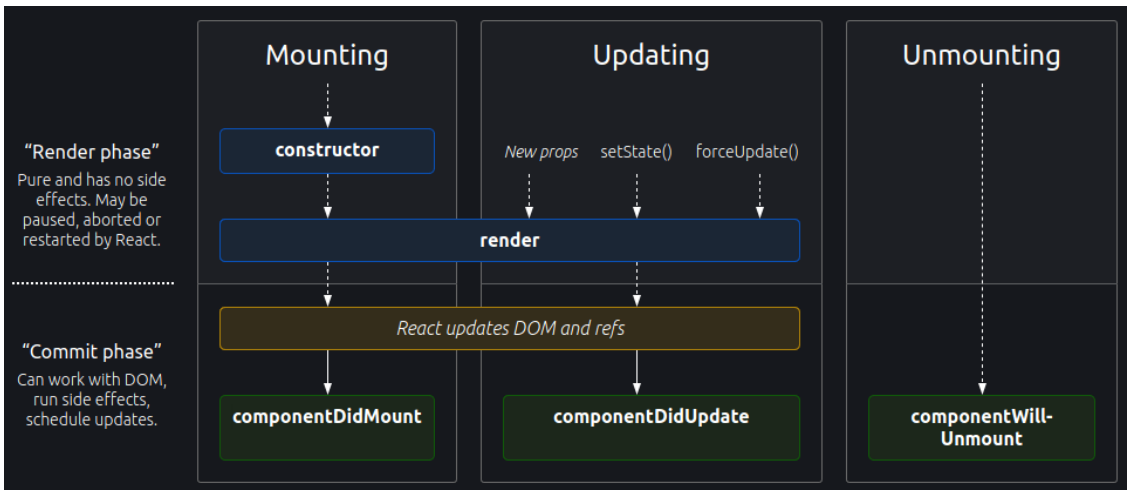
- Raamistiku Angular testrakendus: https://github.com/rebekkabb/angular-testrakendus.
- Raamistiku React testrakendus: https://github.com/rebekkabb/react-testrakendus.
- Raamistiku Vue testrakendus: https://github.com/rebekkabb/vue-testrakendus.

## Lisa 2. Testrakenduse disain

Kuvatõmmised testrakenduse komponentide disainist, mida kasutati raamistike loomisel.

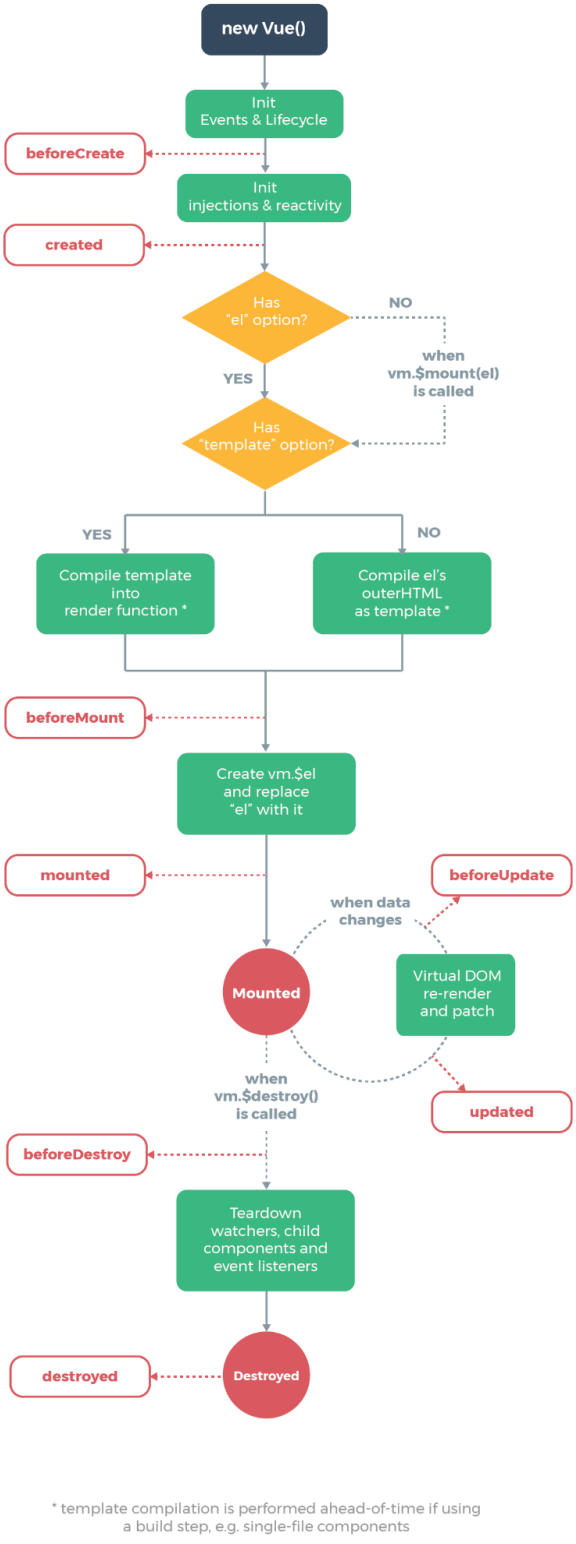

## Lisa 3. React komponendi elutsükkel

Diagramm React komponendi elutsüklist [38].



## Lisa 4. Vue komponendi elutsükkel

Diagramm Vue komponendi elutsüklist [46].



## Litsents

Mina, **Rebekka Breedis**,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

   **JavaScripti kasutajaliidese raamistike võrdlus**,

   mille juhendaja on Lidia Feklistova,

   reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

*Rebekka Breedis*
***07.05.2021***