Tallinna Ülikool

Digitehnoloogiate instituut

Informaatika õppekava

Vue.js rakenduse arendus isikliku portfoolio lehe näitel

Bakalaureusetöö

Autor: Fred Korts

Juhendaja: Romil Rõbtšenkov

Autor: ……....”…….……” 2018

Juhendaja: ….……....”…….……” 2018

Instituudi direktor:.……....”…….……” 2018

Tallinn 2018

Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

…… ……

(kuupäev) (autor)

Sisukord

Sissejuhatus

JavaScript raamistike hüppeline levik viimase viie aasta jooksul on teinud arendaja töö keerulisemaks. Iga raamistik tutvustab oma viisi rakenduste arendamiseks ning arendaja peab otsustama, milline on tema vajadustele vastavalt kõige sobilikum. Tihti tehakse otsus raamistiku hetkelise populaarsuse põhjal. Selline põhjendus võib pikemas perspektiivis kujuneda arendaja aja raiskamiseks, kuna uute raamistike populaarsus tavaliselt vaibub poole aasta jooksul. Autor ei garanteeri, et antud töös käsitletud raamistik on endiselt aktuaalne viie aasta pärast. Samas on tegemist ühe tugevaima kandidaadiga veebiarendajate valikute hulgas (Swift, 2018).

Käesoleva bakalaureusetöö eesmärgiks on luua Vue.js rakendus ja uurida selle arenduskäiku isikliku portfoolio lehe näitel. Autori varasem kokkupuude Vue.js raamistikuga seisnes seminaritöö teemal „Vue.js raamistiku õppematerjal“. Selle seminaritöö raames loodud õppematerjal[1] tutvustas lugejale raamistiku põhilisi mõisteid ja funktsionaalsusi. Autor jagas oma õppematerjali võimalikult laialdaste kogemustega arendajatele. Lugejate peamine kriitika oli, et autor ei käsitlenud kindla rakenduse loomist. Autor loodab selle bakalaureusetööga selle probleemi lahendada.

Autor keskendub peamiselt arendusetappidel ja valminud rakenduse osade üldisele ülevaatele. See tähendab, et kirjeldatakse olulisemad rakenduse loomisel tekkinud probleemid ja lahenduse otsused. Selle töö eesmärk ei ole pakkuda lugejale juhendit sarnase rakenduse loomiseks. Samas teeb autor oma rakenduse lähtekoodi vabalt kättesaadavaks. Autor eeldab, et lugeja on tuttav enim levinumate IT-alaste mõistetega.

Bakalaureusetöö on jaotatud viieks peatükiks. Esimeses peatükis tutvustab autor oma rakenduse plaani ja seletab, kust selle idee tuli, milline on selle väärtus ja millist funktsionaalsust hakkab see rakendus pakkuma. Teine peatükk kirjeldab lähemalt Vue.js raamistikust ja kasutatud tehnoloogiast. Kolmas ja neljas peatükk on kõige sisukamad ja tutvustavad arendustprotsessi, mis algab disainist ja lõppeb koodi kompileerimisega.

[1] https://github.com/fredkorts/seminaritoo/wiki

1. Rakenduse kirjeldus

Arendatava projekti eesmärk on luua üheleherakendus, kus külastaja saab navigeerida lehe sisu vahelt, lisada/kustutada informatsiooni ja logida sisse/välja ilma lehte taaslaadimata. Rakendus luuakse Vue.js raamistiku baasil. Tegemist on progressiivse JavaScript raamistikuga, mida kasutatakse eesrakenduste arendamisel. Selle peamine otstarve on kasutajaliidese arendamine, kuid õigete tööriistade rakendamisel on võimalik luua ka üheleherakendusi.

Rakenduse idee tuli autorile kui potentsiaalne lahendus reaalsele probleemile. Nimelt tööotsinguil küsitakse tihti kandidaadilt CVd ja kas koodinäidet või tehtud tööde nimekirja. Autoril tekkis võimalus lahendada kaks probleemi ühes – luua portfoolioleht hetkel populaarsust koguva raamistiku baasil ja dokumenteerida oma protsess akadeemilise tööna.

Kuna tegemist on portfooliolehega, kus andmeid saab lisada või kustutada, tuleb luua kaks vaadet: tavakasutaja ja autenditud kasutaja vaade. Tavakasutajal on piiratud võimalused, st talle on ainult lubatud navigeerida sisu vahel. Autenditud kasutaja on portfoolio omanik ning tema vaates tekivad lisavõimalused, nagu sisu lisamise nupp ja kustutamise link iga eraldiseisva infotüki kohta. Hetkel ei ole vajadust lisakasutajate jaoks, seega kasutaja registreerimise aken jääb ära.

Interaktsioonidisaini üks kümnest kasutusheuristikast väidab, et kasutajat tuleb hoida informeerituna, kui süsteemis toimub muutus, andes talle situatsioonist lähtuvalt teavituse (nt küsime kasutajalt enne millegi kustutamist kinnitust) (Nielsen 2015). Antud töös on kolm protsessi mille teostamisel või muutmisel tuleb teda teavitada: sisu kustutamisel, sisu lisamisel ja kasutaja autentimise seisundi muutusel. Autor kirjeldab seda pikemalt disainipeatükis.

Kasutajaliides tuleb kindlasti enne rakenduse arendamist prototüüpida ning autor pakub esialgseid disainijooniseid ja mõtteid kasutajaliidese kohta. Disain muutub arenduse käigus ning algseid kavandeid tuleb käsitleda üldiste ideedena.

2. Kasutatav tehnoloogia

Tänapäeva veebiarendajal on suur valik lahendusi igale võimalikule probleemile ning uue projektiga alustamisel võib tekkida segadus. Kindlasti on soovitatav teha endale või tiimile selgeks rakenduse nõuded enne kui hakatakse tehnoloogiat valima. Tihti valitakse arendajale tuttav lahendus selle asemel, et uurida alternatiive, mis võivad säästa aega ja raha.

Üldiselt jaotub rakenduse arendus kaheks osaks, eesrakendus (ingl *front end*) ja tagarakendus (ingl *back end*). Eesrakenduse alla käib kõik, mis on külastajale nähtaval, ja tagarakenduse osa sisaldab rakenduse loogikat, nagu andmete käitlemine ja autentimine.

Selles peatükis toob autor välja tähtsaimad kasutatavad tehnoloogiad ning põhjendab nende valikut. Tuleb mainida, et rakendusesiseselt kasutab autor HTML/CSS koodi, kuid ei näe põhjust seda tehnoloogiat pikemalt kirjeldada, kuna autor eeldab, et potentsiaalne lugeja on sellega juba tuttav.

Node.js[2] on avatud lähtekoodiga mitmeplatvormiline arenduskeskkond, mida kasutatakse serveri-poolsete rakenduste arendamisel. Node.js rakendusi kirjutatakse JavaScript keeles ja see võimaldab lisada projekti lisamooduleid, mis rakenduse võimalusi laiendavad (Tutorialspoint, kuupäev puudub). Mooduleid lisatakse paketihalduri NPM (ingl *node packet manager*) abil. Selle projekti raames toimub Vue.js rakenduse arendus, sellele lisamoodulite paigaldamine ja tulemuse jooksutamine NPMi abil.

Vue.js[3] on eesrakenduse raamistik, mille peamine funktsionaalsus seisneb kasutajaliidese ehituses. Tegemist on progressiivse raamistikuga, mis tähendab, et sellega saab kasutajaliidest mistahes projekti arenduse faasis täiendada. Samas võimaldab Vue.js luua üheleherakendusi (ingl *single page application*) nagu selles töös käsitletav rakendus.

Plaanitud rakenduses on eesmärgiks lubada kasutajal navigeerida vaadete vahel ilma lehte taaslaadimata. Vue.js rakendab virtuaalset DOM (ingl *document object model*) funktsionaalsust, mis tähendab, et ilma tervet lehte taaslaadimata saab muuta seda lehe

[2] https://nodejs.org/en/
[3] https://vuejs.org/

osa, kus muudatus toimub. On ka teisi raamistike (React, Angular, Zorium jm.), mis sama võimalust pakuvad, kuid Vue.js eelis on selle suhteline raskusaste ja uudsus võrreldes teiste raamistikega.

Vuetify[4] on komponendipõhine raamistik. Selle eesmärk on pakkuda puhast, semantilist ja taaskasutatavat lahendust, et rakenduse visuaalse struktuuri arendamine oleks võimalikult sujuv (Leider, kuupäev puudub). Vuetify võimaldab semantilist koodi, sest kõik selle kasutajaliidese osad on eelnevalt kompileeritud Vue.js komponendid, kus igal osal on oma ülesehitus ja võimalikud sisendid juba eelnevalt defineeritud. Näiteks saab interaktiivse kuupäeva valimise komponendi välja kutsuda süntaksiga: `<v-date-picker></v-date-picker>`

Projekti nõuetefaasis kaalus autor kolme variandi vahel: Bootstrap 4, Semantic UI ja Vuetify. Autor on eelnevalt kokku puutunud ainult esimese kahega, kuid Vuetify tundus kõige loogilisem, kuna see pakkus nii struktuuri kui ka kasutajaliidese osi komponentide kujul. See tähendab, et plaanitav rakendus jälgiks ühtset koodistruktuuri.

Firebase[5] on Google poolt välja arendatud platvorm, mis pakub mitmeid teenuseid ja mida on kerge oma projekti integreerida. Arendajal on võimalik aega säästa ning keskenduda oma rakenduse koodile, selle asemel et üles seada andmebaasi või leiutada uut kasutaja autentimise süsteemi.

Vuex[6] on teek, mis aitab rakendada Flux arhitektuuri Vue.js rakenduses. Flux on Facebooki arendajate välja töötatud rakenduse arhitektuur, mis käsitleb rakendusesisesest informatsiooni (Gore 2016).

Vue.js lubab rakendusesisestel komponentidel suhelda mitmel moel. Peamine meetod on saata infot vanemalt lapsele ning see on eelistatud väiksemate rakenduste puhul. Vastassuunas tuleb luua vanemale eraldi funktsioonid, mis kuulavad spetsiifilist käitumist lapskomponendil. Suurema rakenduse puhul võib ette kujutada, et selline ülesehitus võib

[4] https://vuetifyjs.com/en/
[5] https://firebase.google.com/
[6] https://vuex.vuejs.org/en/

väga kiirelt muutuda korralageduseks, kui arendaja peab pidevalt meeles pidama, kes mida kuulas või kes mida saadab ja kuhu.

Vuexist võib mõelda kui Vue.js rakenduse andmebaasist. Selle ainuke miinus seisneb selles, et lehe taaslaadimine kaotab informatsiooni. Vuex ei saa kasutada päris andmebaasi asendusena. Selle eesmärk autori rakenduses on hoida väärtusi, mis kasutaja nähtut juhivad.

3. Rakenduse disain

Disaini hea arendusstrateegia on uue projekti alguses hädavajalik. See säästab aega ja raha. Vastasel juhul tuleb projekti lõppfaasides asju ümber mõelda ning sellele võib kuluda rohkem aega, kui arendaja on algse plaaniga juba loonud komponente või lahendusi, mida ei lähe vaja (Baytech, 2013a). Efektiivne disaini arendusstrateegia koosneb seitsmest etapist (Meazey, 2017):

1. eesmärgi tuvastamine;
2. ulatuse defineerimine;
3. rakenduse skeemi prototüüpimine;
4. sisu loomine;
5. visuaalsete elementide disainimine;
6. testimine;
7. lehe ülespanek.

Veebilehe arenduse algul tuleb teha kindlaks, mis on selle lehe eesmärk: kas pakkuda informatsiooni või midagi müüa. Ei ole soovitatav minna mõlemas suunas, kuna külastajal võib tekkida arvamus, et teda mõjutakse ostma selle firma tooteid või teenuseid (Baytech, 2013b). Autori rakenduse eesmärk on pakkuda tööandjale CV ja koodinäide ühes. Sellest tulenevalt võiks rakenduse disain põhineda CV dokumendi struktuurist.

Eesmärgi kindlaks tegemisel tuleb läbi mõelda lehe ulatus. Selles etapis tuleb kliendiga (või endale) selgeks teha kõik plaanitavad komponendid. Selle alla kuuluvad navigatsiooni menüüd, teavitamise süsteem, kasutaja autentimise meetodid, kasutajagrupid, sisutüübid jms. Tihti alustatakse projekti ja arenduse jooksul nõuab klient kas lisafunktsionaalsust või tahab midagi muuta, mis ajab ajakava segi. Sellel põhjusel on tugevalt soovitatav allkirjastada kliendiga leping, mis defineerib töö ulatuse ja tingimused lisaarenduse puhul.

Kolmandal etapil algab eelnevate ideede praktiline rakendamine. Selle etapi all on soovitatav visandada või kokku panna üldine lehe struktuur. Traditsiooniliselt tehakse seda kas traatmudel (ingl w*ireframe model*) või paberprototüübi kujul. Tihti tehakse selles

etapis ka konkurentide uuring. Vaadatakse, kuidas on teised sarnaseid projekte teostanud, ja võetakse neist inspiratsiooni.

Sisu loomine ei võta kaua aega, kuna eelnevalt on läbi mõeldud kindlad nõuded ja kuidas neid esitada. Antud projektis on andmekogum minimalistlik. Peamised sisukategooriad on: eelnevad töökogemused; tehtud tööd; hariduskäik; kaaskiri.

Visuaalne disain on kommunikatsioonimeetod autori ja külastaja vahel. Selle rakenduse kontekstis on selleks autor ja potentsiaalne tööandja. Hea disain pole mitte ainult maitse asi, vaid ka külastaja tähelepanu haaramise meetod. Veebilehe edu oleneb heast disainist ning suurem osa sellest peatükist on pühendatud selle välja töötamisele.

Selles peatükis kirjeldab autor bakalaureusetöös käsitletud rakenduse disaini protsessi, skeeme ja disaini otsuseid. Eelnevalt kirjeldatud seitsmeetapilisest arendusstrateegiast käsitletakse ainult esimest viit punkti. Lehe ülespanek jääb ära, kuna autor ei soovi hetkel seda üldsusele avalikuks teha ja jagab lähtekoodi ainult selle töö lugejatele ja potentsiaalsele tööandjale. Lisaks otsustas autor disaini testimise vahele jätta, kuna selle visuaalne struktuur põhineb populaarsel raamistikul, mida on eelnevalt testitud.

3.1. Rakenduse nõuded

Autori rakenduse eesmärk on eelnevalt mitu korda välja toodud, seega järgmise sammuna tuleks uurida plaanitava rakenduse ulatust ehk mis elemendid on vajalikud. Rakenduse disaini inspiratsiooniks on autor võtnud CV dokumendi. Traditsiooniline CV koosneb isiklikust informatsioonist, hariduskäigust, täiendkoolitusest, töökogemusest, keelteoskusest, arvutioskusest ja täiendavast infost (Artmedia, kuupäev puudub).

CV kirjutamisel ei ole rangeid reegleid. Peamine nõue on, et see oleks loetav, kindlale lugejale suunatud ja huvitav (Monster Worldwide, kuupäev puudub). Siit saab eeldada, et kuna sihtrühmaks on veebiarendusvaldkonna inimesed, saab mõned traditsioonilised osad välja jätta, näiteks arvutioskus, täiendav info ja keelteoskus. Arvutioskus on eeldatud, täiendavat infot saab jagada esimesel kohtumisel ja keelteoskus ei ole sellise töö juures kriitiline.

Isiklik informatsioon, hariduskäik, töökogemus ja tehtud tööd peaksid ilmtingimata rakenduse disainis olema. Isiklik informatsioon peaks sisaldama kontakt infot, autorist pilt, ning viis peamist oskust vähenevas järjekorras. Isiklik informatsioon peaks olema pidevalt nähtav, mistahes lehel külastaja hetkel viibib.

Hariduskäik, töökogemus ja tehtud tööd peaks olema eraldi lehtedel, erinevalt traditsioonilisest CV-dokumendist, kus sellised osad on üksteise all. Selleks tuleb rakendada navigatsioonimenüü. Iga lehe all peaks olema sellega seotud nimistu. Kui välja arvata tehtud tööde osa, on tegemist tekstilise sisuga ning oleks vaja kindlaks teha primaarse ja sekundaarse tähtsusega info (nt hariduse juures otsustada mis on tähtsam info, kas kooli nimi või eriala nimetus).

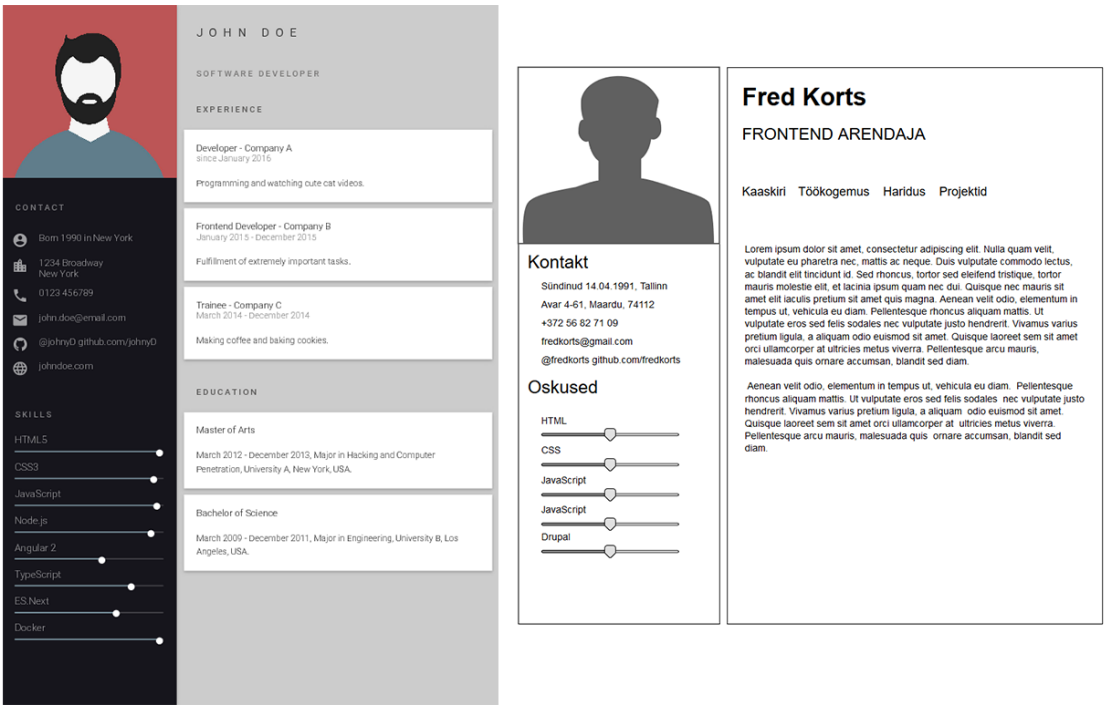
Rakendusel on vaja infosisestusvõimalust. Selle jaoks tuleb välja töötada vormid ja teavitused, mis annavad omanikule teada, kas info sisestamine õnnestus. Vormid tuleks teha nähtavaks ainult juhul, kui kasutaja on autenditud. Tavakülastajale ei tohi lubada infot sisestada. Siit saab eeldada, et on vaja sisselogimise vormi. Kuna tuleb ainult üks administratiivõigustega kasutaja, saab registreerimisakna ära jätta.

3.2. Disainijoonised

Enne projekti kallal tööle asumist tuleb uurida, kas keegi on midagi sarnast teinud. Esialgne otsing pakkus tehnoloogia valiku poolest paar sarnast lahendust, kuid disaini poolest võeti kasutusele ainult mõned omadused, mis on iseloomulikud CV-dokumendile (vertikaalne nimistu tehtud töödest, hariduskäigust ja töökogemus).

Üks kõige huvipakkuvam Vue.js projekt oli PDF-dokumendi genereerimise rakendus, mille autoriks on Sara Steiert. Rakendusega saab tutvuda autori GitHub lehel[7]. Selle väärtus käesoleva töö autorile seisnes rakenduse väljastatud disainides. Autor otsustas ühe variandi põhjal luua oma disaini (vt joonis 1).

[7] https://github.com/salomonelli/best-resume-ever/

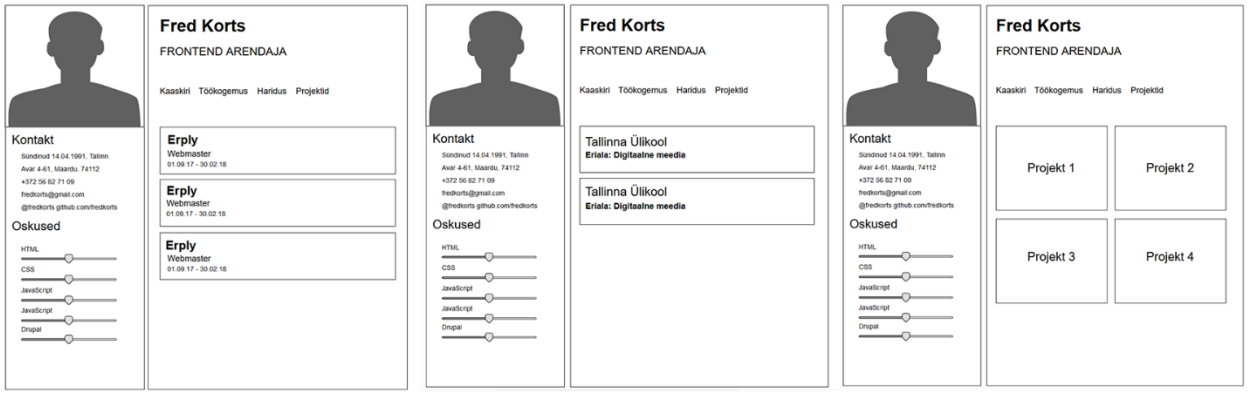


Joonis 1. Vasakul Sara Steiert disain (Steiert, 2017) ja paremal autori esialgne skeem.

Joonis 1 annab üldise ülevaate rakenduse struktuurist. Rakendus on jagatud kaheks tähtsaks osaks. Umbes 30% ruumist on pühendatud kontaktinfo ja viie parima oskuse jaoks. Parem pool ehk sisuosa näitab kaaskirja, kogemust, haridust või tehtud töid. Menüü navigeerimisel muutub teksti sisuala vastavalt valitud vaatele.

3.3. Sisu ülesehitus

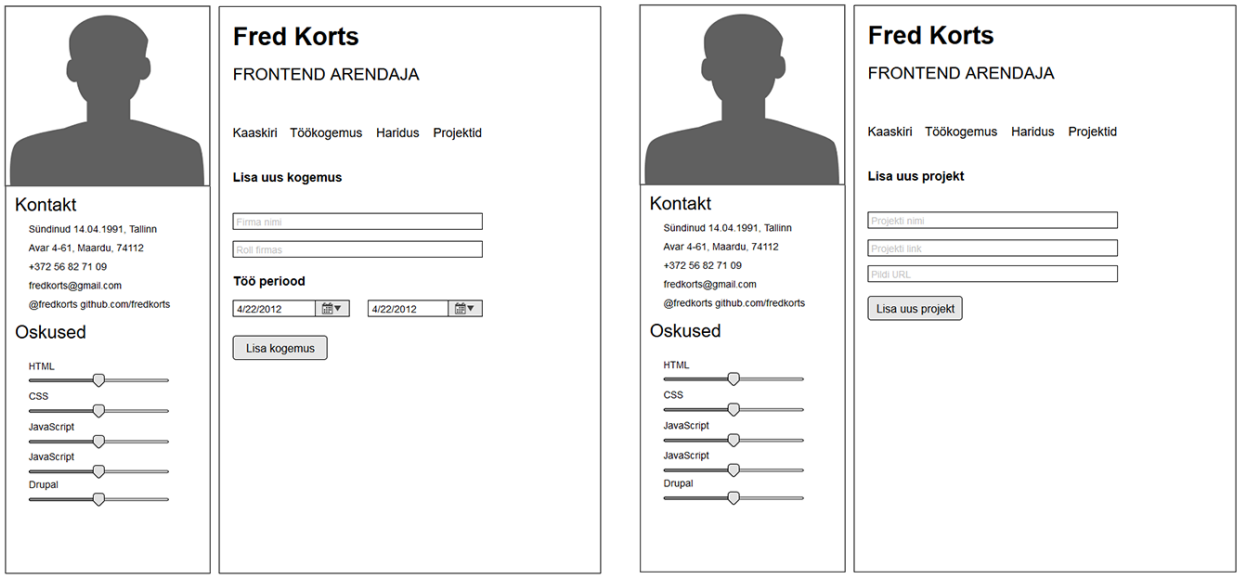
Joonisel 2 on kirjeldatud ülejäänud kolm vaadet. Töökogemuse ja hariduse vaade on struktuurilt sarnased, sest neil on võrdne kogus infot iga eraldiseisva andmeühiku kohta. Tehtud tööde osa järgib hetkest veebiarenduse trendi, et lisatakse link vajutatavale pildile. Pilt tavaliselt sisaldab endas kas firma logo või ekraanitõmmist.



Joonis 2. Vasakul töökogemuse vaade, keskel hariduse vaade, paremal portfoolio vaade.

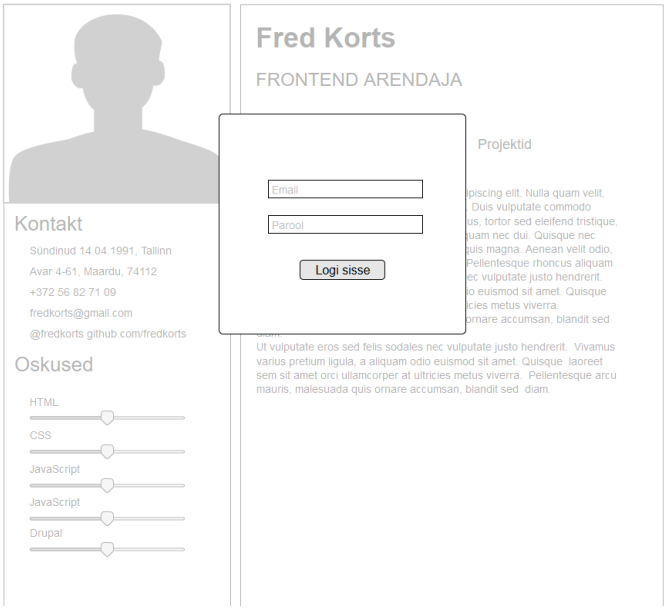
Üks nõuetest sellel projektil on, et saaks lisada ja kustutada informatsiooni. Esimese sammuna tuleb läbi mõelda info lisamise vorm. Kuna töökogemuse ja hariduse all on sarnane info, võime koondada selle vormi ühe disaini alla, seega jääb luua veel projektide vaade.

Töökogemuse ja hariduse vaade vajab kahte välja, kus saame paluda kasutajalt nii firma/kooli nime ja rolli firmas / eriala nimetust. Lisaks on vaja kaks kuupäevavaliku elementi, mis määraksid perioodi alguse ja lõpu. Projektide vaade vajab kolm välja: projekti nimi; projekti link; projekti URL.



Joonis 3. Vasakul töökogemus ja hariduse vormi disain, paremal projektid.

Viimane element, mida selles peatükis tuleks käsitleda, on kasutaja sisselogimisaken, kuna andmete sisestamise vormid tekivad ainult autenditud kasutajale. Vajalik on nii link kui ka aken, mida see avab. Autori rakendus on üheleherakendus, seega aken ei tohi viia eraldi lehele. Kõige sobilikum kuju sellisele elemendile on dialoog aken (ingl *pop-up window*) (vt joonis 4).



Joonis 4. Sisselogimisaken *pop-up* kujul

Autori rakenduses on kuus tüüpi sisu. Esimesed kaks on kontaktinfo ja viie peamise oskuse astmeline pädevus. Kontaktinformatsioon sisaldab endas üldist teavet kandidaadi kohta ning viimasena on soovitatud linkida kas isiklikule lehele või allikale kus võib sinu kohta lähemalt lugeda (Artmedia, kuupäev puudub). Kuna tegemist on arendaja CVga, on mõttekam lisada viited GitHub või LinkedIn kontodele.

Autor otsustas varem tehtud näidete põhjal lisada oma viis peamist pädevust skaala kujul. Info on võetud LinkedIn profiilist ehk need on autori suhtlusvõrgustiku poolt kinnitatud andmed. Andmed on tõmmatud läbi LinkedIn API.

Rakenduse kolm dünaamilist infotüüpi on: töökogemus, tehtud tööd ja hariduskäik ehk neid saab lisada ja kustutada. Töökogemus ja hariduskäik on sarnase ülesehitusega, ning tehtud tööd kujutavad endast galeriid, kus iga pilt on lingitud tehtud töö aadressile. Töökogemus ja hariduskäigu sisudisain põhineb CV ülesehitusel.

4. Rakenduse arendamine

Selles peatükis kirjeldab autor rakenduse ülesseadmist, failistruktuuri, rakenduse tähtsamaid komponente ja nende arendusprotsessi. Koodinäiteid on kasutatud minimaalselt, kuna tegemist ei ole juhendi, vaid Vue.js rakenduse arendusprotsessi kirjeldusega. Autor kirjeldab probleemi või ülesannet iga alapeatüki juures, pakub välja võimalikud lahendused ja põhjendab oma valikut. Lähtekood on saadaval aadressil https://github.com/fredkorts/bakatoo.

4.1. Projekti ülesseadmine

Enne projekti arendamist on vaja otsustada, kuidas seda paigaldada ja hallata. Vähese kogemusega arendajad on harjunud, et luuakse tavaline *html*-fail ja lisatakse aadressid kõigile kasutatavatele tehnoloogiatele (nt JavaScript, jQuery, Vue.js) peamise faili päisesse. Õppimise eesmärgil pole sellel meetodil midagi viga, kuid suuremate projektide raames tuleb kasutada paketihaldurit (ingl *packet manager*).

Pakettide alla kuulub kogu projekti kood, mida autor ise ei kirjutanud ning millest sõltub rakenduse funktsionaalsus. Paketihaldurid lihtsustavad vanema protsessi, mille alla kuulub koodi otsimine, allalaadimine, lahtipakkimine ja projekti kopeerimine ühelauselisse käsku. Igal suuremal programmeerimiskeelel on oma paketihaldur.

JavaScript rakenduste arendamisel on lai valik paketihaldureid, kuid autor võttis kaalutlusele kolm, millega ta on isiklikult tuttav: NPM, Bower ja Yarn. NPM ja Bower olid veel hiljuti keskendunud erinevatele rakenduse pooltele. Bower leidis kasutust eesrakenduste (ingl *front-end*) pakettide haldamisel ja NPM oli tagarakendustele keskendatud (ingl *back-end*). Nende peamine erinevus seisnes selles, kuidas nad pakette paigaldasid ja haldasid. NPM tõmbas kõik paketid, mis olid vähegi üksteisega seotud, ning vahest tekkisid koopiad samast failist. Bower andis kasutajale valida ja otsustada, mis versiooni tuleb rakenduses kasutada (Hefetz, 2017). Eesrakenduse arendusel võib sellest tekkida liigne failisuurus, mis võib rikkuda kasutajakogemust suurema laadimisajaga. See probleem on lahendatud alates NPM kolmandast versioonist.

Lisaks eelnevalt mainitule on Boweri paigaldamiseks vaja NPMi (Hefetz, 2017). See tundub autorile liigse sõltuvusena, kuna NPM on võimeline täitma sama eesmärki. Autor eelistas NPMi Yarnile peamiselt varasema kogemuse olemasolu tõttu ja kuna see oli juba autori arenduskeskkonda paigaldatud.

Järgmiselt tuli autoril otsustada, milline tuleb failistruktuur. Väiksemate rakenduste puhul ei oma see tähtsust, kuid mahukama projekti puhul tuleb jagada failid nende kasutuseesmärkide järgi kaustadesse oragniseeritavuse nimel. Vue.js raamistiku autor on välja arendanud käsurea tööriista `vue-cli`. Selle eesmärk on panna püsti Vue.js projekti põhi ning kõik sellega seotud kaustad ja seadistada kriitilisemad süsteemid, mida kasutatakse eesrakenduste arendamisel.

```
npm install -g vue-cli
```

Koodinäide 1. NPM käsk, et paigaldada vue-cli

Autori esimene ülesanne oli paigaldada `vue-cli` NPM käsu abil (vt koodinäide 1). `vue-cli` pakub mitmeid erinevaid malle, mille vahel saab valida olenevalt arendatava projekti eesmärgist. Webpack on suur osa veebiarendus maastikust peamiselt tänu oma töötlemise funktsionaalsusele. Webpack võtab sisendiks projekti koodi ja genereerib ühe kokku pakitud faili. Selle eelis on, et külastaja veebilehitseja peab pärima ainult ühte rakenduse faili.

```
vue init webpack bakatoo
```

Koodinäide 2. Vue.js projekti ülesseadmise käsk

Malli valikul tuleb anda vue-cli käsk (vt koodinäide 2). Käsu sisendiks on käsu tüüp (init), mall, mida tuleb alla tõmmata, ja kausta nimi, kuhu tuleb projekt. Enne projekti seadistamist küsitakse arendajalt üle mõned projektiga seotud küsimused (vt joonis 14). Autor otsustas kasutada peamiselt vaikimisi väärtuseid, välja arvatud testimise osas, mille autor otsustas sellest projektist välja jätta. Webpack ja sellega seotud testimine on omaette teema, ning selle töö puhul ei ole kriitilise tähtsusega.

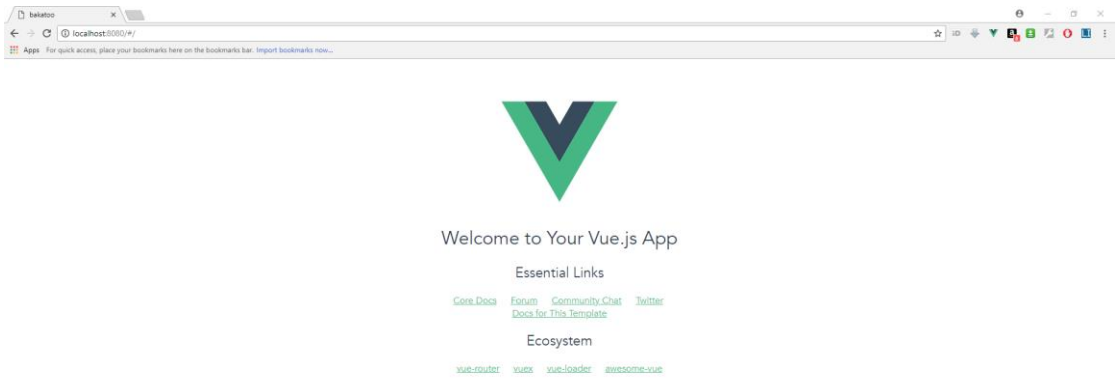
```
? Project name bakatoo
? Project description Vue.js rakenduse rakendusuuring
? Author Fred Korts <fredkorts@gmail.com>
? Vue build standalone
? Install vue-router? Yes
? Use ESLint to lint your code? No
? Set up unit tests No
? Setup e2e tests with Nightwatch? No
? Should we run `npm install` for you after the project has been created? (recommended) npm

   vue-cli · Generated "bakatoo".
```

Joonis 14. Webpack seadistamine

Peale kasutaja kinnitust seadistuse otsustes tõmbab vue-cli webpack malli ja kõik sellega seotud failid. Projekt on valmis arenduseks, ning viimasena tuleb jooksutada webpack veebiserveri käivituskäsk (vt koodinäide 3). Tulemuseks on lokaalserveris jooksev rakendus (vt joonis 15). Eelnevalt kirjeldatud protsess ja sellega seotud tehnoloogiad on hetkel kõige levinum viis kuidas kiirelt seadistada oma projekt ja panna see jooksma lokaalserveris.

```
npm run dev
```

Koodinäide 3. Veebiserveri käivitamine.



Joonis 15. Lokaalserveris jooksev rakendus.

Suurematel projektidel on kaks olekut: arendus- (ingl *development*) ja tootmisfaas (ingl *production*). Autori rakendus oli senimaani arendusfaasis, mis tähendab, et projekt oli jaotatud komponentideks, vigu näidati jooksvalt konsoolis ja kiirus ei mänginud tähtsat rolli. Tootmisfaasis rakendusega on just vastupidi. Projekti osad tuleb kompileerida

üheks failiks, vigu ja konsooliteateid ei tohi olla ning rakendus laadimise kiirus on kriitiline hea kasutajakogemuse nimel.

Autori projekt sai ülesseatud webpack mallil, mille üheks tugevuseks on, et arendaja saab kontrollida kuidas projekt kompileeritakse. Peamine konfiguratsiooni fail `webpack.prod.conf.js` asub `/build` kaustas, ning seal määratakse kuidas kasutada erinevaid kompileerimise protsessiga seotud vahendeid. Näiteks saab määrata, kuidas tõlgitakse SCSS failid tavalise CSS kujule või optimeerida stiili failid vanemate lehitsejate kasuks. Autoril ei olnud lisasoove projekti kompileerimise kohta ning jättis selle faili muutmata.

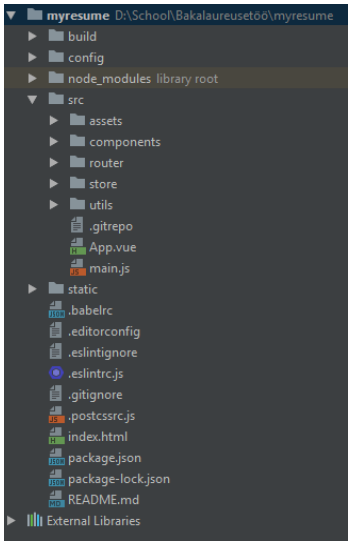
```
npm run build
```

Koodinäide 7. NPM kompileerimiskäsk.

Kompileerimiskäsuga (vt koodinäide 7) algab protsess mis eraldab JavaScript ja CSS eraldi failidesse. Tulemuseks on kompaktsed failid, mille eeliseks on lühikesed laadimisajad. Uued failid lükatakse `/dist` kausta, mis tekib NPM käsu täitmisel. Projekt on selle etapi lõpuks tehtud, ning piisab `index.html` ja `/dist` kaustast, et jooksutada fail serverist.

4.2. Projekti ülesehitus

Webpack genereeritud failistruktuur sisaldab viit kausta ja mõnd konfiguratsioonifaili, mis on seotud projekti haldussüsteemiga. Allpool välja toodud illustratsioon (vt joonis 16) kuulub autori rakenduse valmiskujule.



Joonis 16. Projekti failistruktuur

Autor tegi genereeritud struktuurile jooksva arenduse käigus mõned muudatused, nagu `store` ja `utils` kaustad, mis sisaldavad Vue.js raamistikuga seotud teenuseid ja abifunktsioone, milles tuleb järgnevates peatükkides juttu. Ülejäänud faile saab kirjeldada lühidalt järgmiselt:

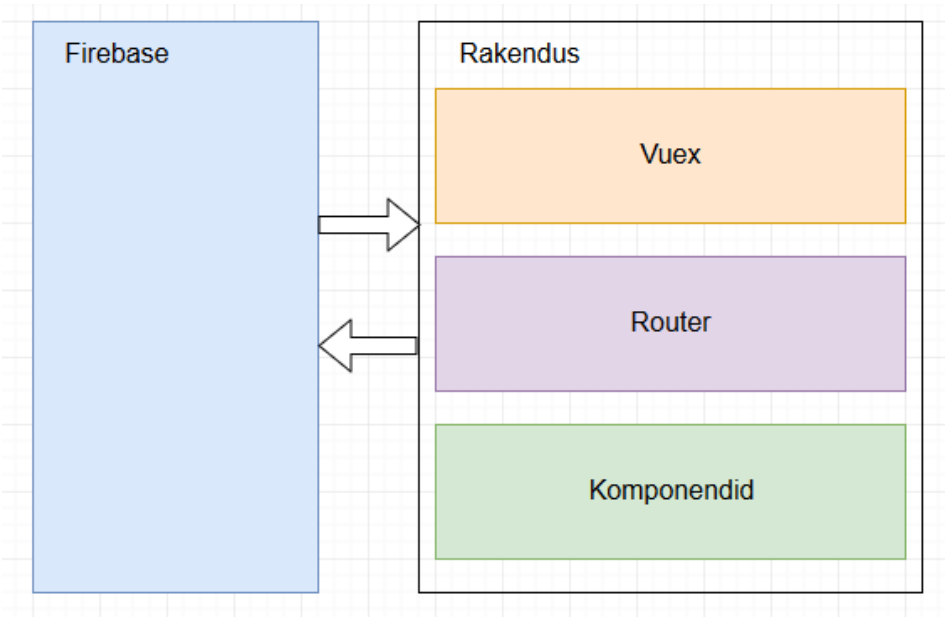
- `build` kaust sisaldab kõiki faile, mis on seotud *webpack*-serveri konfiguratsiooniga. Tavaliselt neid faile ei muudeta, kui just ei ole vaja lisa funktsionaalsusi paigaldada või eemaldada.
- `config` hoiab endas peamist faili mille abil saab määrata käesoleva projekti ehitamise etappi.
- `node_modules` sisaldab projekti pakette, mis said paigaldatud NPM käsuga.
- `src` on arendatava rakenduse peamine kaust, ning selle sisu oleneb autorist.
    - `assets` kaust sisaldab kõiki faile, mis on seotud projekti vältel paigaltatud lisamoodulitega.
    - `components` sisaldab faile, millest koosneb autori rakendus. See kaust on veel omajagu jaotatud kaheks kaustaks, mis sisaldavad üldist ja autenditud vaadet.
    - `router` hoiab faile, mis on seotud rakenduse marsruutimise funktsionaalsusega.

    - `store` sisaldab Vuex oleku kontrolli funktsionaalsusega seotud faile.
    - `utils` on mõeldud lisafunktsionaalsuste jaoks ning hoiab autori poolt kirjutatud konfiguratsiooni faile.
    - `App.vue` on kõige esimene komponent, mida laetakse rakenduse laadimisel.
    - `main.js` on Vue.js rakenduse päis, kus defineeritakse kasutatavad teenused ja süsteemid mida rakenduse siseselt hakatakse kasutama.
- `static` on webpack poolt loodud kaust, kuhu pannakse failid, mida on vaja rakenduse ehitamise lõppfaasis ignoreerida.
- `index.html` on rakenduse hierarhia kõige ülemine fail ning seda kasutatakse, et tõmmata Vue.js rakendus käima.

Projektihaldusega on seotud omaette failid, mida tuleks mainida. `package.json` on selle projektis kasutuses pakettide nimekiri, mis said paigaldatud NPM käsuga. `.gitignore` on versioonihaldusega Git seotud fail, mis hoiab nimekirja failidest, mille muutusi ei jälgita. `.eslintrc.js` on seotud ESLint[8] laiendusega, mis otsib autori JavaScript koodis struktuuri või treppimis vigu.

Autori rakendus koosneb kahest osast, millest üks on rakendus ja teine on sellega ühilduv andmebaasi süsteem. Rakendus jaguneb omakorda kolmeks alamosaks, mis moodustab selle põhifunktsionaalsuse. Autor on loonud joonise, et anda parema ülevaate plaanitavast rakendusest (vt joonis 17). Kahe osa vahelised jooned näitavad, et tegemist on andmete vahetusprotsessiga.

[8] https://eslint.org/



Joonis 17. Rakenduse skeem

Vuex haldab rakenduse tähtsamaid oleku väärtusi millest sõltub mida kasutaja näeb. Router vahetab erinevate linkide all oleva sisu. Komponendid on osad mis moodustavad rakenduse struktuuri ning see jaotub omakorda alamosadeks. Autor kirjeldab igat osa pikemalt sellele pühendatud peatükis ja kuidas nad moodustavad täieliku Vue.js ühelehe rakenduse.

4.3. Rakenduse põhiosa

Autori rakenduse üles ehitus põhineb `vue-cli` poolt paigaldatud webpack mallil. Rakendusel on kaks põhilist faili, mis lehitsejas vaate moodustavad. Selleks on `index.html` ja `main.js`. `Index.html` sisaldab tavalist HTML-koodi, mille külge kinnitub Vue.js objekt, mis konfigureeritakse `main.js` failis. See objekt võtab sisendiks väärtuste massiivi, mis määrab rakenduse käitumise ja andmed millega seda teostatakse.

`main.js` on Vue.js rakenduse hierarhias kõige tähtsam fail, sest siin imporditakse ja registreeritakse rakenduses kasutatavad tehnoloogiad. Nende tehnoloogiate alla langevad sellised Vue.js ökosüsteemi osad nagu Vuex ja Vuetify. Peale paigaldamist on registreeritud moodulid kättesaadavad terve rakenduse ulatuses.

Autor registreeris oma rakenduse `main.js` failis järgnevad moodulid: Vue-router, Vuefire, Vuetify, *store* (Vuex eksporditud objekt) ja autori loodud DateFilter funktsioon. Vuefire ja Vuetify on ainukesed moodulid autori rakenduses, mis loevad pistikprogrammidena (ingl *plugin*). Sellised moodulid registreeritakse `Vue.use()` süntaksiga. Vue-router ja store objektid on eraldi failides konfigureeritud ja eksporditud, et neid saaks importida `main.js` faili ja registreerida Vue.js objekti omadusena.

DateFilter on JavaSkripti funktsioon, mille autor ise kirjutas, et kuupäevi puhtamale kujule formaatida. Vue.js lubab registreerida kasutaja poolt kirjutatud filterfunktsioone ja kasutada neid süntaksi osana. Autori defineeritud filtri registreerimise jaoks kasutatakse `Vue.filter()` süntaksit, mille sisendiks on tekstiline väärtus, mis määrab filtri nimetuse, ja konfigureeritud objekt, mille nimi autori rakenduses on DateFilter.

Tavaline Vue.js objekt sisaldab elemendi nime, millele rakendus kinnitub, andmeid, meetodeid ja mitmeid elutsükli konkse (ingl *lifecycle hooks*, autori tõlge). Autori sätestatud Vue.js objekt on kinnitatud „#app“ *html*-elemendi külge, mis asub eelnevalt mainitud `index.html` failis. Kuna autori rakendus on eraldatud komponentideks, tuleb siin kohal määrata rakenduse pealmine komponent. Autori rakenduses on selleks `App.vue`.

Autori rakenduse esimene versioon koosnes ühes failist, `App.vue`. Sellise struktuuri miinus seisnes halvas loetavuses ning mitmes kohas tekkis korduv kood. Autori lahendus oli kasutada Vue.js raamistiku komponentide funktsionaalsust. Komponendid aitavad jaotada rakenduse struktuuri ja loogika eraldi seisvateks alamosadeks. Igal komponendil on oma *html*-struktuur, käitumist defineeriv skript ja selle komponendiga seotud stiilid. Komponentfailid määratakse `.vue` faili laiendiga.

Autor jaotas oma komponendid kolme kausta vahel vastavalt selle otstarvele: `components`, `components/add` ja `components/menu`. `components` kaust sisaldab üldotstarbelisi komponente, `components/add` kaust sisaldab autenditud vaatega seotud lisamise vorme ja `components/menu` kaust hoiab marsruutimisega seotud sisu komponente. Rakenduses on kasutatud järgnevalt kirjeldatud komponendid:

- `App.vue` rakenduse esimene komponent, mis määrab selle üldise struktuuri ja kutsub välja teisi komponente.
- `Sidebar.vue` rakenduse vasakpoolne kõrvalmenüütaoline struktuur, mis hoiab autori üldiseid ja kontaktandmeid.
- `SignIn.vue` rakenduse sisselogimise kasutajaliides.
- `Home.vue` rakenduse „esileht“ ehk esimene komponent, mille sisu esitakse rakenduse käivitamisel.
- `Projects.vue` näitab andmeid seotud autori tehtud töödega.
- `Experience.vue` näitab andmeid seotud autori eelmise töökogemustega.
- `Education.vue` näitab andmeid seotud autori hariduskäiguga.
- `AddToProfile.vue` üldine vaade, mis kontrollib millist andmete lisamise vormi näidata autenditud kasutajale.
- `addProject.vue` uue tehtud töö lisamise vorm.
- `addExperience.vue` uue kogemuse lisamise vorm.
- `addEducation.vue` uue hariduse lisamise vorm.

Autori rakenduse `Home.vue`, `Projects.vue`, `Experience.vue`, `Education.vue` komponendid pärivad andmebaasilt andmeid ja esitlevad neid kasutajale. Igal mainitud komponendil on ühendus andmebaasiga, massiiv andmete hoidmiseks ja kustutamise nupp, mis saadab andmebaasile käsu kustutada vastava IDga andmed. Andmete lisamise funktsionaalsus on eraldatud `AddToProfile.vue` komponenti, et vältida koodi korduvust.

`AddToProfile.vue` on vahelüli `App.vue` ja `addProject.vue`, `addExperience.vue`, `addEducation.vue` komponentide vahel. Autor impordib viimased kolm komponenti `AddToProfile.vue` siseselt ja kontrollib, millist näidata vastavalt millisele menüü lingile on vajutatud. Selline struktuuri lahendus väldib järjekordset koodi korduvust.

4.4. Vuex

Autoril tuli otsustada, kuidas käsitleda komponentide vahelist suhtlust. Väiksema rakenduse puhul võib olla tegemist ainult mõne komponendiga, kus info liigub vanemalt lapsele ja vastupidi. Autori rakenduses on üle kümne komponendi ja andmed, mis ei liigu hierarhiliselt vanemalt lapsele. Autor võttis kasutusele Vuex teegi, sest selle andmetele saab ligi mistahes komponendist terve rakenduse ulatuses.

Vuex on teek, mis aitab rakendada Flux arhitektuuri Vue.js rakenduses. Flux on Facebooki arendajate välja töötatud rakendusarhitektuur, mis järgib kindlaid printsiipe, et käsitleda rakendusesisest informatsiooni (Gore 2016). Autor otsustas koodi loetavuse nimel luua eraldi faili (store.js) Vuex objekti konfigureerimiseks. See fail asub `/store` kaustas. Autor registreerib eksporditud Vuex objekti `main.js` failis globaalse muutujana mis lubab kasutada Vuex objekti mistahes komponendi sees. Vuex objekt mida autor konfigureerib koosneb neljast osast:

- `state` on autori rakenduse tsentraalne andmete hoius. Autor kasutab seda, et hoida boolean väärtusi, millest sõltub rakenduse vaade. Näiteks, `addToProfile: false` väärtus määrab, kas näidata andmete nimekirja või andmete sisestus vaadet. Selle väärtuse muutuse puhul muutub ka sellega seotud vaade.
- `getters` defineerib meetodid, mille abil loetakse `state` massiivist väärtused. Iga `state` väärtuse puhul on soovitatud kasutada eraldi `getter` meetodit.
- `mutations` sisaldab meetodeid, mis muudavad `state` väärtust. Igal autori `state` väärtusele on pühendatud meetod, mis muudab ainult selle väärtust.
- `actions` ei muuda otseselt väärtusi, vaid kutsub välja `mutations` meetodi mis teostab muudatuse.

Autori rakenduses on Vuex objekt registreeritud Vue.js objekti omadusena, mis lubab ligipääsu `$store` andmetele läbi selle funktsioonide. Dollari märk muutuja ees määrab, et tegemist on globaalse muutujaga, mida saab kasutada terve rakenduse ulatuses. Autori komponendid saavad käsitleda `$store` andmeid kahel moel, andmete pärimine läbi

`$store.getters.meetod` ja andmete muutmine `$store.dispatch(meetod)` meetodiga.

4.5. Router

Üheleherakendus, millel on rohkem kui üks vaade, vajab navigeerimis lahendust, mis lubab muuta vaadet ilma lehe taaslaadimiseta. Vue.js ametlik lahendus on Vue-*router* moodul. Vue-router lisab projektile võimaluse kasutada marsruutimise (ingl *routing*) funktsionaalsust.

Vue-*router* on sätestatud sarnaselt Vuex objektile. Selle konfiguratsiooni fail `index.js` asub eraldi `/router` kaustas. `index.js` ekspordib objekti, mille autor registreerib Vue.js objekti globaalse muutujana. `index.js` failis imporditakse iga vaate kohta vastav komponent ja registreeritakse sellele osutav link. Marsruutimise objekt koosneb `routes` massiivist, mis sisaldab igale vaatele vastavat alamosa. Iga alamosa sisaldab kolm kohustusliku väärtust:

- `path` on vaatele osutav link;
- `name` on vaate nimi;
- `component` määrab vaate komponenti.

Autori rakenduses on neli vaadet: `Home`, `Education`, `Experience` ja `Projects`. Autor otsustas, et vaikimisi näidetakse `Home` vaadet. Vaikimisi vaadet ehk peamist lehte määratakse `path` all kaldkriipsuga, mis viitab esilehe lingile.

Autor kasutab Vue-*router*'ile omast komponenti `routerView` sisukonteinerina. Defineeritud vaated ilmuvad sellel alal olenevalt milline neist on hetkel aktiivne. Aktiivset vaadet määratakse Vue-*router* lingiga (vt koodinäide 4). Kuna tavaline *html*-link ei toimi, on sellise lahenduse puhul vaja erilist süntaksit, mis sama eesmärki täidab. Autor kasutas linkide asemel nuppe, millest igal ühel oli Vue-*router* link vastavale vaatele.

```
<v-btn router to=”education”>Haridus<v-btn>
```

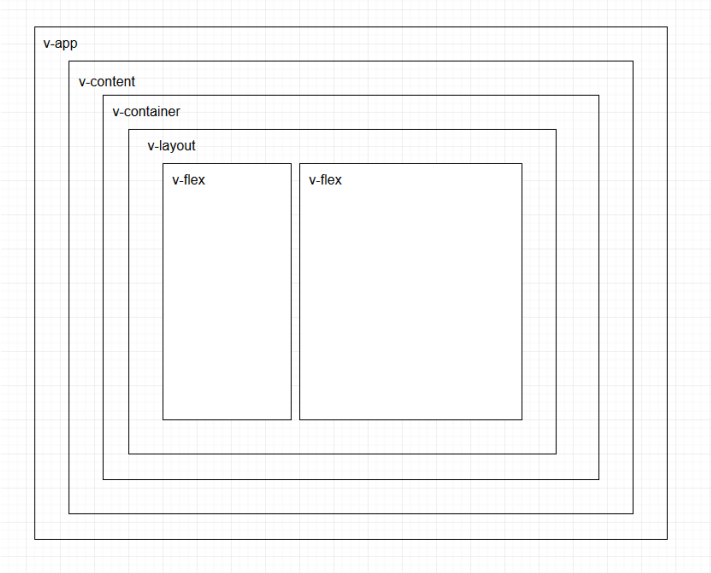
Koodinäide 4. Vue-*router* link.

Autoril oli ainult üks probleem eelnevalt kirjeldatud lahendusega. Sisu muutmine oli liiga jäik. Enne kui autor hakkas oma lahendust välja arendama, tuli uurida teiste arendajate kogemust sellel teemal. Autor avastas, et kuna `routerView` on dünaamiline komponent, on võimalik kontrollida selle ülemineku (ingl *transition*) animatsiooni. Piisas vaid `routerView` komponendi ümbritsemisest `transition` komponendiga.

4.6. Vuetify

Rakenduse visuaalne pool on struktureeritud Vuetify raamistikuga. Selle eesmärk on pakkuda taaskasutatavaid komponente, mille põhjal ehitada oma projekti struktuuri ja kasutajaliidest. Autoril oli valida Bootstrap, Semantic UI, Foundation ja Vuetify raamistiku vahel, kuid lõplik otsus põhines autori soovil tutvuda Vue.js ökosüsteemiga ning kuna rakendus oli niigi komponendipõhine, oli mõttekam jätkata samas vaimus.

Autor kasutas nii üldiseid struktuurikomponente kui ka spetsiifilisi kasutajaliidese komponente, et ehitada oma rakendus. Üldine struktuur sai paika `App.vue` failis. Struktuur koosneb viiest peamisest elemendist, millel on kindel hierarhiline struktuur (vt joonis 18). Iga element on eraldi komponent, millel on selle nimeline *html*-märgend. Igal komponendil on eelnevalt defineeritud CSS reeglid. Vuetify on üles ehitatud *mobile-first* printsiibil, mis tähendab, et struktuuri ehitusel alustatakse kõige väiksemast ekraani suurusest.



Joonis 18. Vuetify struktuuri komponentide hierarhia

Vuetify dokumentatsioon kirjeldab iga komponendi lehe all kindlad väärtused, mida tohib sellele sööta. Neid väärtusi nimetatakse *prop*'ideks (edasi teksti sees: ’propp: propi, proppi’). Komponendi välimust ja käitumist saab kontrollida läbi propiväärtuste. Näiteks, kõige pealmine struktuuri komponent võtab vastu proppe, mis määravad rakenduse värvi skeemi. Autori rakenduse loomise hetkel oli toetatud kaks peamist stiili, `light` ja `dark` mida sai määrata `v-app` komponendil selle nimelise propiga.

Sama moodi lubab Vuetify muuta struktuurireegleid, andes komponendile vastava propi. Näiteks, autoril tuli struktuuri loomisel määrata, kuidas sisukomponendid jaotuvad lehitseja vaates. Selle jaoks tuli anda kahele `v-flex` komponendile propid, mis määravad nende jaotuse. Jaotust kontrollitakse sarnaselt Bootstrapiga tulpade abil. Maksimaalne tulpade arv ühel real on kaksteist ja igal ekraani suurusel (ekraani suurused väiksemast suuremani: xs, sm, md, lg, xl) on sellele vastav propp. Autori rakenduses on mõlemal `v-flex` komponendil kaks proppi, `xs` ja `md`. Mõlemale lisandub tulpade arv mille järgi komponente jaotatakse vastaval suurusel.

Lisaks struktuurile leidis autor mitmeid komponente, mida sai rakendada kasutajaliidese loomisel. Selle asemel, et kirjeldada igat kasutatud komponenti, tunneb autor, et mõttekam oleks keskenduda ühele kasutajaliidese osale. Rakendusel on kolm erinevat andmete sisestamise vormi, mis erinevad üksteisest ainult andmekategooria järgi. Autor

sooviks lähemalt vaadata ühte neist, nimelt hariduse sisestamise vorm `addEducation` komponendi all.

Hariduse sisestamise vorm koosneb viiest komponendist: kahest `v-text-field` komponendist, kahest `v-date-picker` komponendist ja ühest kinnitus nupust `v-btn`. Tekstivälja komponent võtab mitmeid erinevaid proppe, aga kõige tähtsam on `required`, mis määrab, et see väli ei tohi tühjaks jääda. Autoril ei olnud põhjust määrata lisaproppe, kuna rakenduse käsitletavad andmed ei vaja lisavalideerimist.

Autor võttis kasutusele `v-date-picker`, sest alternatiiv oleks olnud kasutada tavalist tekstivälja. Projekti algul pakuti ainult ühes stiilis kalendrit. Tulemus töötas, kuid võttis palju ruumi. Antud bakalaureusetöö kirjutamise ajal lasti välja uuendus, mis pakkus kompaktsema välimusega kalendreid, mis ilmuvad ainult väljale vajutades. Komponendiuuendus soovitas ümbritseda `v-date-picker` menüü komponendi `v-menu` vahele. Autor eelistas seda ruumi säästmise kaalutlusel.

Vuetify `v-btn` komponent pakub iga juhuse jaoks erinevaid nuppe. Nupu stiil oleneb talle antud propist ja klassist. Igale nupule on võimalik lisada ikoone `v-icon` komponendi kujul. Vuetify ei paku suurt valikut ikoone (ainult hädavajalikemaid) ning dokumentatsioonis on soovitatud paigaldada eraldi ikoonide moodul. Autor otsustas, et talle piisab olemasolevast valikust. `v-btn` on tavaline HTML `button` element, millel puudub seos rakenduse funktsioonidega. Meetod tuleb kinnitada nupule Vue.js `@click` süntaksiga (vt koodinäide 5).

```
<v-btn @click=”meetod”><v-icon>add</v-icon></v-btn>
```

Koodinäide 5. Näide nupust, mille vajutusel käivitub funktsioon.

4.7. Andmete käsitlemine

Autor otsustas kasutada Firebase andmebaasi lahendust, kuna planeeritav andmete maht oli väike ja eesmärgiks oli luua rakendus, mida saab jooksutada kus iganes ilma suure vaevata. Firebase’i oli niigi kerge lisada projekti, kuid Vue.js on teinud selle veelgi mugavamaks. Vue.js arendus tiim on välja lasknud VueFire teegi mille paigaldamisel

tekib Vue.js objektile laiendus nimega *firebase*. Firebase laiendus lubab siduda andmebaasi andmed otse Vue.js objektiga, ning andmete muutuse puhul peegeldatakse see kasutaja vaates.

VueFire eeldab Firebase olemasolu. Kuna tegemist on välise mooduliga tuli selle paigaldada NPM käsuga. Autor otsustas konfigureerida Firebase eraldi `firebaseConfig.js` failis `/utils` kaustas kuna muud moodulid olid juba sama moodi eraldatud koodi puhtuse nimel. Vastasel juhul oleks `main.js` fail loetamatu. Parem on selliseid probleeme vältida.

Autor ei hakka kirjeldama, kuidas luua Firebase andmebaasi. Selle jaoks on eraldi juhendid, kuigi autor arvab, et see protsess on tehtud võimalikult lihtsaks ning iga üks vähesegi IT kogemusega inimene saaks sellega hakkama. Kui andmebaas on valmis antakse kasutajale sellega seotud skripti mille lisamisel rakendusse tekib ühendus andmebaasiga.

`firebaseConfig.js` sisaldab Firebase teenuse importi ja autori andmebaasi ühendusega seotud andmed. Autor ei kirjelda neid selles töös, kuid iga üks saab seda järgi vaadata lähtekoodist. Autor palub, et lugeja ei taaskasutaks selle tööga seotud ühenduse andmeid. Viimase sammuna loob autor `fireb` muutuja mis saab sisendiks Firebase objekti ja `config` muutuja. `config` muutuja hoiab andmebaasi ühendusega seotud infot. Autor ekspordib loodud objekti, ning impordib selle komponentides mis vajavad andmebaasi ühendust.

Autori rakendus teostab kolme tüüpi operatsiooni andmebaasiga: andmete lugemine, sisestamine ja kustutamine. Andmete lugemisega ja kustutamisega on seotud kolm komponenti (`Education`, `Experience`, `Projects`), mis pärivad andmeid kasutajale esitamiseks. Kuna nende kolme komponendi ülesehitus ja protsess on identne on autor otsustanud anda ülevaate operatsioonidest, selle asemel et korrata sama juttu iga komponendi juures.

Lugemisega seotud komponendid impordivad esimese sammuna Firebase `fireb` objekti mille autor eksportis `firebaseConfig.js` failis. Sellele objektile määratakse viide andmebaasi tabelile kus autor hoiab selle vaatega seotud andmeid ja tulemus salvestatakse

muutujasse. Samale muutujale viidatakse nüüd komponendi *firebase* laienduse all, mis lubab neid andmeid kasutada komponendi struktuuris.

Kuna andmed on massiivi kujul, kasutab autor `v-for` süntaksit, et esitada andmed komponendi struktuuris. `V-for` on Vue.js raamistikule omane tsükli süntaks (vt koodinäide 6). Iga andme ühik on massiiv, mis sisaldab selle kategooriaga seotud infot, ning Vue.js lubab sellele ligipääsu loogeliste sulgude vahel. Antud struktuur on kasutatav ainult `v-for` tsükli siseselt.

```
<v-flex v-for=”e in example”>
     <h3>{{ e.title }}</h3>
     <p>{{ e.subtitle }}</p>
     <p>{{ e.date }}</p>
</v-flex>
```

Koodinäide 6. Näide v-for tsüklist ja andmete esitusest struktuuris.

Kustutamine toimub lugemisega samadel komponentidel. Juhul kui on tegemist autenditud kasutajaga, tekib eelnevalt loetud andmetele igale individuaalsele andme ühikule punane kustutamise nupp. Nupu vajutusel käivitub meetod, mille sisendiks on vajutatud andme ühiku ID. Kuna autor on mitu korda kogemata andmeid kustutanud, otsustas ta enne funktsiooni jooksutamist kinnitust küsida. Kasutaja kinnitusel jooksutakse `remove()` käsk, mis kuulub Firebase teenuse alla.

Andmete lisamine toimub `addEducation`, `addToProfile` ja `addProject` komponentides. Struktuuri poolest on igal komponendil andmesisestus vorm, Firebase objekti import, tühjad väärtused skriptis, millega seotakse teksti sisestus elemendid (tekstiväljad ja kuupäevavalija) ja meetod mis lükkab uued andmed tabelisse. Peale andmete lisamist näidetakse kasutajale teadet, kas operatsioon õnnestus.

Andmete lisamise vormi teksti väljad on seotud `v-model` atribuudiga. Tegemist on Vue.js omase süntaksiga, nagu eelnevalt kirjeldatud `v-for`. `V-model` seob sisestus elemendi sellega vastava väärtusega komponendi skriptis. Tulemuseks on reaktiivsed andmed, mis muutuvad vastavalt kasutaja sisestusele. Autor struktureeris iga komponendi andmed objektina kuna andmebaas võtab sisendiks massiivi.

Andmete sisestus funktsioontäidab kolme ülesannet. Esiteks lükatakse kasutaja sisend andmebaasi, teiseks tehakse andmeväljad puhtaks andes igale väärtusele tühja stringi ja kolmandaks jooksutakse Vuex meetod `showAddToProfile`, mis omakorda muudab `addToProfile` väärtuse selle vastandiks. Andmete sisestus vaade oleneb sellest muutjast, ning selle muutuse puhul vahetatakse vaade andmete esitus vaate vastu.

4.8. Autentimine

Autoril oli vaja eraldada autenditud ja tavavaated, et vältida situatsioone, kus mittevolitatud isikud saavad muuta rakenduse andmeid. Kuigi autor ei plaani seda rakendust avalikkusele avada, on selle GitHub kood avalikult kättesaadav. Autor otsustas luua sisse- ja väljalogimise funktsionaalsuse ja võttis kasutusele Firebase autentimise teenuse, kuna Firebase oli juba projektis saadaval.

Autor eraldas autentimise `SignIn` komponenti. Selle struktuur koosneb `v-dialog` Vuetify komponendist mis on hüpikaken (ingl *pop-up window*) ja kahest funktsioonist: `toggleModal()` ja `signIn()`. `toggleModal()` ülesanne on sulgeda aken õnnestunud sisse logimise puhul.

`signIn()` funktsioon käivitab Firebase autentimisega seotud meetodi `signInWithEmailAndPassword()` mille sisendiks on kasutaja nimi ja parool. Õnnestunud sisse logimisel luuakse kasutaja sessioon Firebase autentimis teenuse poolel ja määratakse `loggedIn` muutuja väärtuseks `true`.

Väljalogimise funktsioon toimub läbi Firebase `signOut()` meetodi mis leidub `App.vue` failis ehk kõige pealmises komponendis. Kuna tegemist on ühe nupuga, ei olnud mõtet luua selle jaoks eraldi komponenti. `signOut()` ei vaja sisendit, ning selle käivitamisel saadab Firebase autentimise teenusele käsu hävitada kasutaja sessioon ja määrab `loggedIn` muutuja väärtuseks `false`.

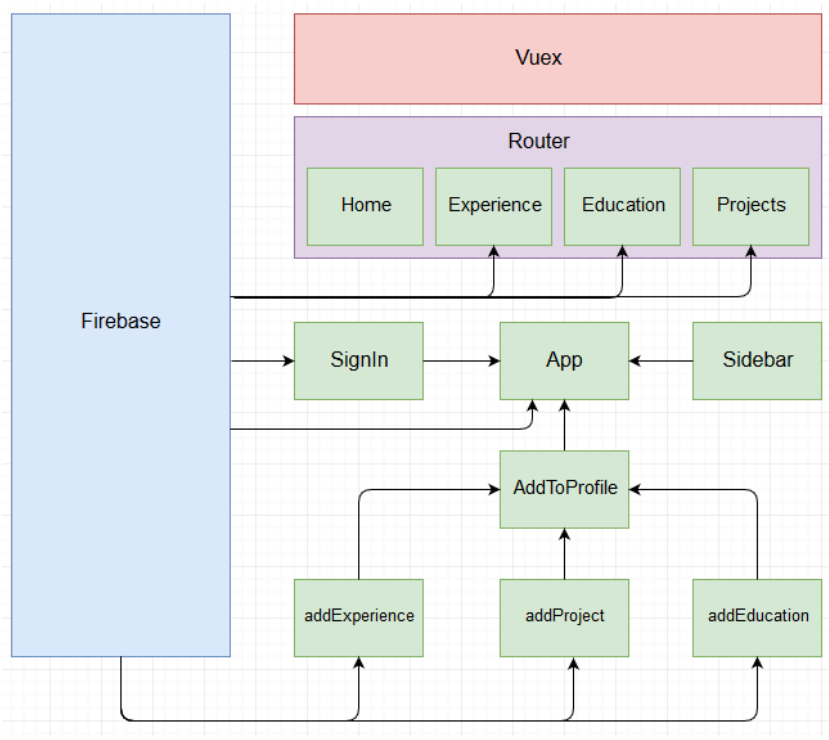
Autor kasutas `v-if` süntaksit, et eraldada autenditud ja külastaja vaated. `V-if` on Vue.js tingimuslause süntaks mis võtab sisendiks boolean muutuja. Autori rakendusel on mitu sellist väärtust, kuid kõige tähtsam on `loggedIn`. Rakendus saab kontrollida, millist

vaadet näidetakse vastavalt `loggedIn` muutujale mille väärtus oleneb kasutaja olekust. Probleem seisneb selles, et lehe taaslaadimisel lähevad rakenduse andmed algolekusse ja tuleb korrata sisse logimise protsessi. Autori üheleherakenduse eesmärk on vältida taaslaadimist, kuid juhul kui midagi sellist juhtub on vaja lahendust mis säilitab kasutaja oleku.

Firebase ametlik dokumentatsioon soovitab luua funktsiooni, mis käivitub lehe laadimisel ja pärib kasutaja staatust (Firebase, kuupäev puudub). Autori lahendus sellele soovitusele oli luua Vuex `actions` meetod, mis rakendab Firebase autentimise teenuse `Auth` objekti. `Auth` objekt pakub `onAuthStateChanged()` funktsiooni mis pärib kasutaja olekut. Juhul kui kasutaja staatus on endiselt sisse logitud, rakenduse taaslaadimisel, määratakse `loggedIn` muutujale `true` väärtus. Autor pani selle funktsiooni `store.js` faili projekti arengu algul, et seda oleks mugavam välja kutsuda, juhul kui seda läheb vaja mitmes komponendis. Projekti lõpus tuli välja, et seda oli vaja ainult ühes kohas, `App.vue` failis.

5. Valminud rakendus

Autori töö tulemusena valminud rakendus vastab suuremas osas autori esialgsetele soovidele ja eesmärkidele. Järgnevalt soovib autor anda lühikese ülevaate valminud rakenduse ülesehitusest (vt joonis 19). Selle töö kolmandas peatükis saadi välja toodud arendatava rakenduse üldine skeem. Antud skeem põhines autori üldisel ideel ning ei olnud veel korralikult läbi mõeldud. Puudus nimekiri kõikidest komponentidest ja nende vahelistest seostest.



Joonis 19. Valminud rakenduse skeem.

Autori valminud rakendus koosneb üheteistkümnest komponendist ja kolmest lisa moodulist. Autor ei lisanud Vuetify raamistiku skeemile, kuna iga komponent kasutab seda oma struktuuri siseselt. Skeemi joonisel kujutatud nooled kirjeldavad suhet lapse- ja vanemakomponendi vahel, ning iga joon kujutab importkäsku. Keskne komponent `App` koosneb üldisest struktuurist, mis impordib rakenduse alam komponendid.

Vuex ja Router on globaalsed teenused, mida saab kasutada terve rakenduse ulatuses, mille tõttu puudub vajadus konkreetse ühenduse järgi. Antud skeemil on Router kujutatud nelja komponendi ümber. Router sisesed komponendid ei ole ühenduses `App` komponendiga, kuna Router võimaldab kasutada `router-view` komponenti mille eesmärk on näidata ja vahetada sisu. `router-view` on kasutuses `App` komponendi all.

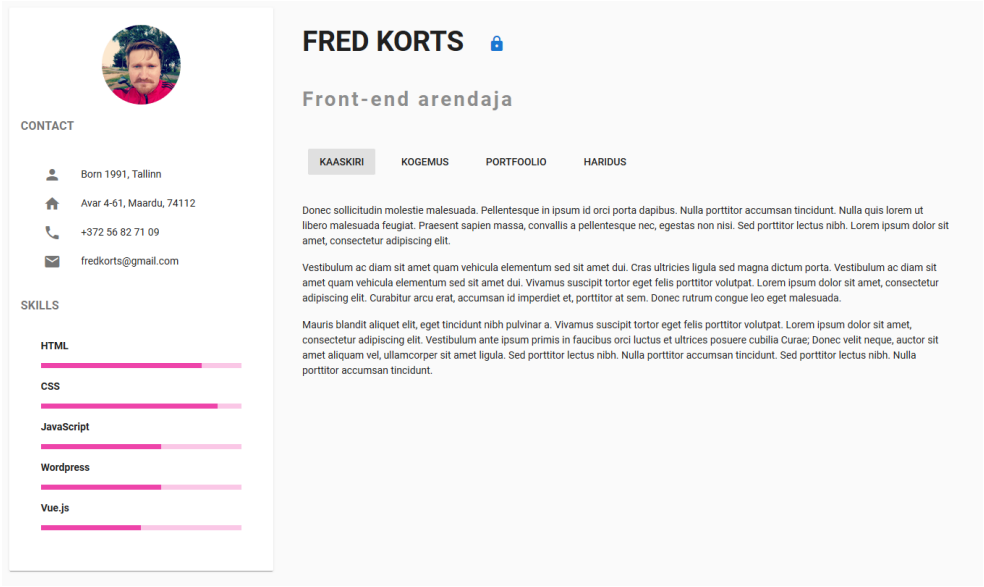
Skeemil kujutatud Firebase koosneb kahest osast: andmebaasi ja autentimise teenus. Andmebaasi osa võimaldab lugeda ja kustutada andmeid järgnevate komponentide all: `Experience`, `Education`, `Projects`. Lisamine toimub `addExperience`, `addProject` ja `addEducation` komponentide all. Autentimise teenus on eraldi osa Firebase all, kuid autor otsustas üldistada seda skeemil ühe osana. Ainuke komponent, mis seda kasutab, on `SignIn`. Väljalogimine teostatakse `App` komponendi all.

Autor otsustas lehe struktuuri ja disaini elemendid üles ehitada Vuetify raamistikul. Vuetify pakub Vue.js komponendi loogikal loodud disaini elemente. Selle disainid põhinevad Google enda Material Design spetsifikatsioonidel. Google eesmärk oli luua visuaalne raamistik mis lubab sarnast kogemust mistahes operatsioonisüsteemil või veebilehitsejal (Spradlin, 2014). Material Design leidis esmakordset kasutust Google Now debüüdil, ning sellest ajast alates on see kasvanud Google tsentraalseks disaini printsiibiks.

Võib vaielda, et värviskeem on teine kõige tähtsam osa rakendusel peale funktsionaalsust. Kasutajaliidese värvivalik on kriitilise tähtsusega inimese ja arvuti vahelises interaktsioonis. See aitab kasutajal paremini eristada tähtsat informatsiooni või funktsionaalsust (Babich, 2017).

Limiteeritud värvivalikuga skeem on soovitatav, et vältida segadust oma külastajate seas. Enamik inimesi eelistab kahe-kolme värvilist skeemi, ning liigne värvikastus võib tekitada külastajate seas segadust (O'Donovan, Agarwala & Hertzmann, 2011). Kõige kergem viis luua värvi skeem on valida neutraalsed värvid ja võtta vähem kasutatavaks värviks midagi erksat. Kuna tegemist on CV dokumendil põhineval disainil otsustas autor võtta primaarseks ja sekundaarseks värvid valged toonid, ning kolmandaks lilla tooni (vt joonis 20).

Joonis 20. Rakenduse värviskeem.



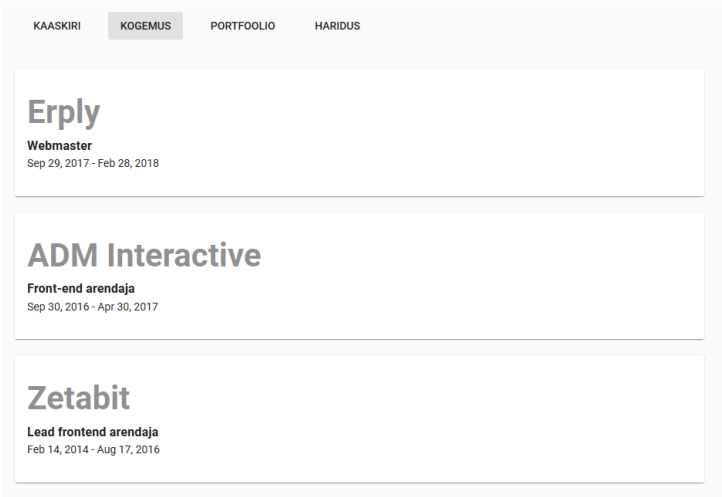
Joonis 21. Valminud prototüübi ekraanitõmmis.

Võttes aluseks varasemalt tehtud joonised tehti prototüüp (vt joonis 21), mille struktuur jäi samaks, kuid suurem osa kasutajaliidese disainist muutus. Autor analüüsib selliseid osi, nagu kontaktinformatsiooni komponent, erinevate sisu tüüpide struktuuri ja nende andmete sisestamise vormid. Viimasena kirjeldab kasutaja sisselogimist ja sellega seotud kasutajaliidest.

Esialgne plaan nägi ette, et kasutaja profiilipilt esitatakse ruudu kujulises raamis ning see võtab oma konteineri täislaiuse. Esimesed katsed sellise disainiga tõid esile ebakõla disainis. Pilt oli liiga suur ja selle kirevad värvid võtsid suurema osa tähelepanust endale. Autor otsustas teha pildi väiksemaks ja suurendas selle `border-radius` CSS reeglit, kuni tekkis ringikujuline raam. Sellise lahenduse eelis on autori arvates vähem tähelepanu hõivav ja parem kooskõlastus ülejäänud disainiga.

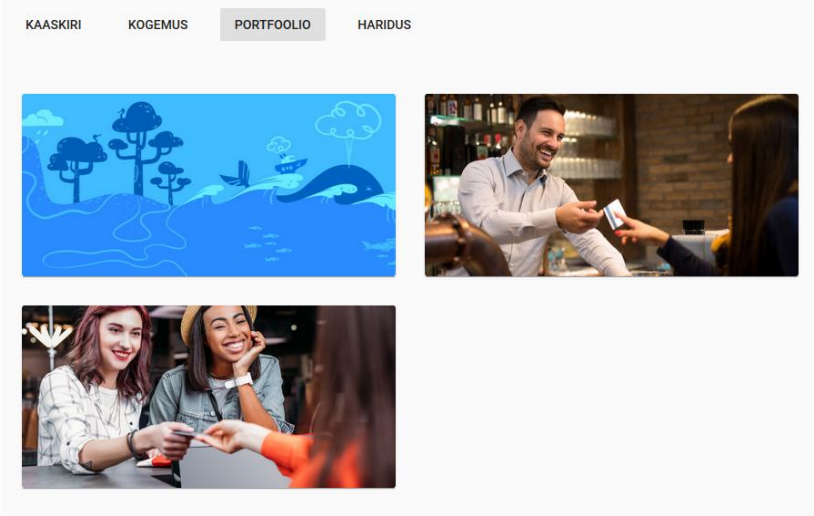
Autor otsustas kasutada ikoone kontaktinfo all, et tõmmata neile tähelepanu ja tõsta informatsiooni arusaadavust (Mertz, 2012). Teine punkt, mille poolest erineb kontakt info on GitHub või LinkedIn lehe puudus. Autor otsustas, et kuna hetkel ei plaani oma lehte üles panna ning käesolev rakendust on plaanis jagada GitHub lingina, on see liigne informatsioon. LinkedIn jäi ära, sest muidu peegeldaks rakendus sotsiaalvõrgustiku profiiliandmeid.

Top viie oskuse skaala oli algsel disainil kujutatud `slider` elemendina, mis tähendab, et külastaja saab seda liigutada mis tahes suunas. Autor ei soovinud, et see osa oleks manipuleeritav, seega tuli uurida alternatiive. Vuetify pakub mitu komponenti, mis sobiksid `slider` elemendi asemele, kuid valituks sai `v-progress-linear` komponent. Ametlikult kasutatakse selliseid elemente, et näidata protsessi kulgu. Värviskeemi poolest on tegemist ühest kolmest elemendist, mis kasutab eelnevalt valitud lilla tooni.



Joonis 22. Töö kogemuse vaade

Töökogemuse ja hariduse vaade jäi samaks mis esialgses plaanis. Autor kasutas Vuetify `v-card` komponenti, et hoida andmete sisu (vt joonis 22). Sama komponent leiab kasutust küljeriba struktuuris ja tehtud tööde vaates. Sisu poolest otsustas autor panna rõhu firma/institutsiooni nimele. CV kirjutamisel puuduvad ranged reeglid, nagu eelnevalt mainitud, aga siiski tuleks hoida ühtset stiili (Green, 2017).

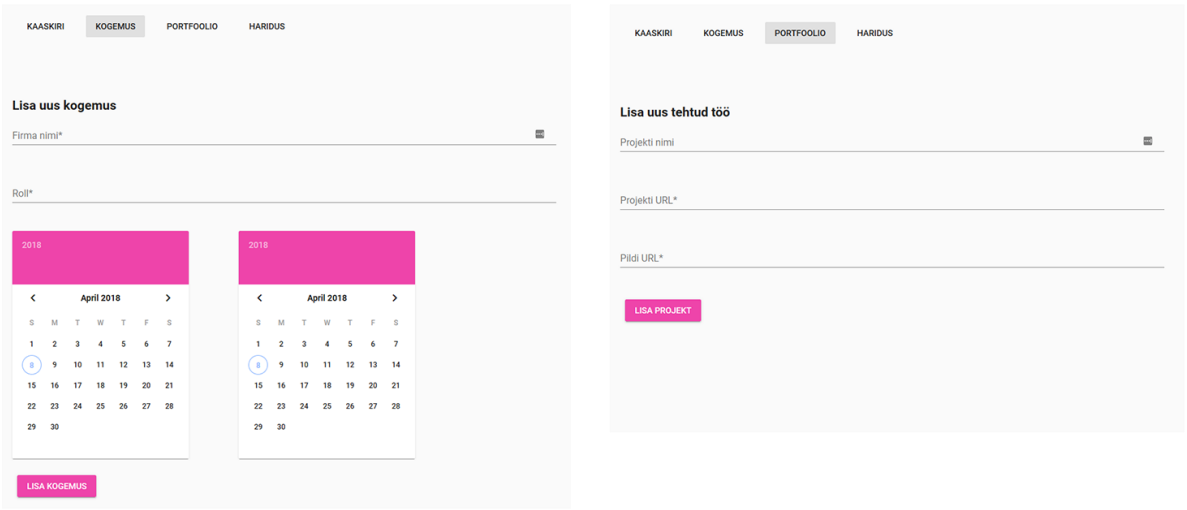


Joonis 23. Portfoolio lehe vaade

Autori eesmärk oli luua minimalistlik portfooliovaade, kus tähelepanu keskel on tehtud töö logo või ekraanitõmmis. Pildile klikates visatakse külastaja tehtud töö aadressile. Jällegi leidis kasutust Vuetify `v-card` komponent. Pildil hiirega hõljudes tekib elemendi alla suurem vari, kuid väljaspool seda ei esine rohkem infot (vt joonis 23).

Andmete sisestamine toimub eraldi vormi juures ning igal vaatel välja arvatud kaaskiri on oma vorm. Esialgse plaaniga võrreldes on vormid jäänud suhteliselt sarnaseks. Suurim erinevus on ajaperioodi valiku komponent. Autor oli plaaninud kasutada jQuery *date picker* elementi, kuid leidis alternatiivi Vuetify oma `v-date-picker` komponendi näol.

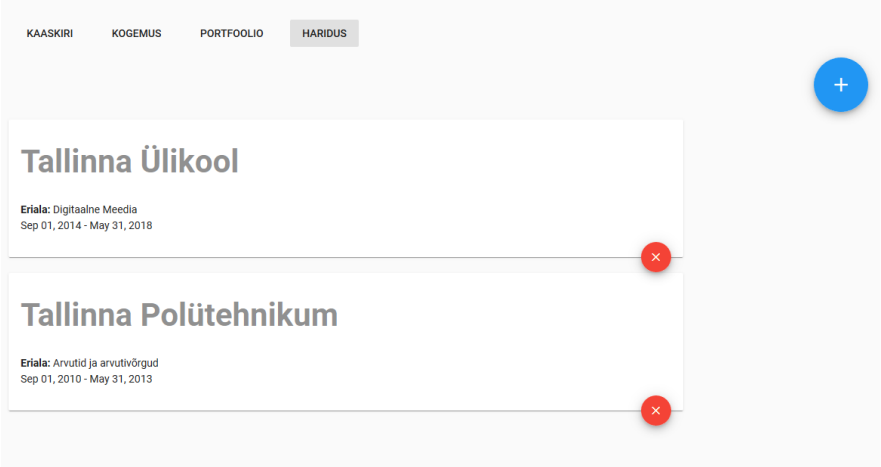
Vorme on kokku kolm, kuid hariduse ja töö kogemuse omad sisaldavad täpselt sama struktuuri. Praktiliselt on rakenduses kasutuses kaks erinevat struktuuri. Vormid sisaldavad peamiselt andmete sisestus välju, sisestus nuppu ja kuupäeva valimis komponenti, et määrata töö/hariduse perioodi. Mõlema puhul on kasutatud lillat tooni (vt joonis 24).



Joonis 24. Andmete lisamise vormid.

Andmete kustutamise funktsioon on nähtav ainult autenditud kasutajale. Kustutamisnupp tekib andmete juurde all paremasse nurka (vt joonis 11). Nupp on Vuetify komponent, mis on absoluutselt positsioneeritud, et see hõljuks pooleldi oma konteinerist väljas. Selles kontekstis on info konteineriks `v-card` komponent.

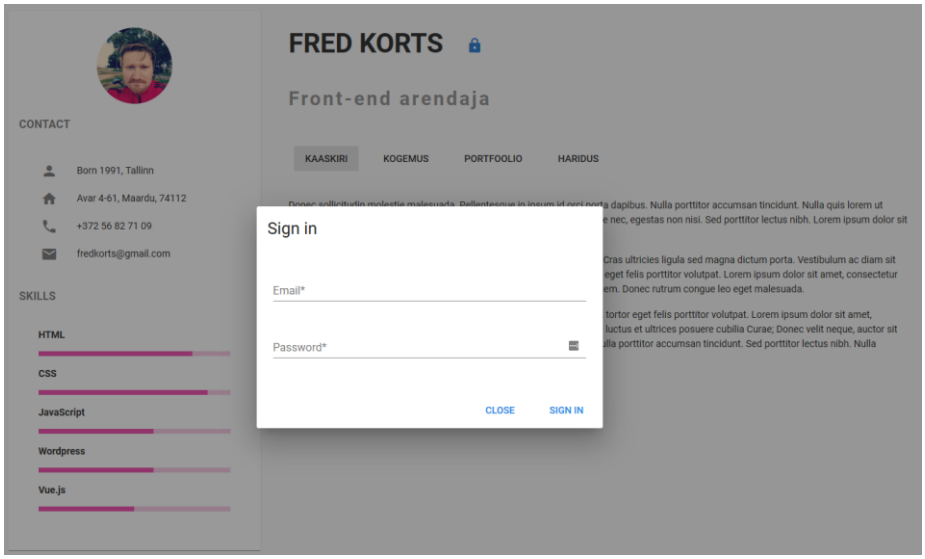
Hariduse, töö kogemuse ja tehtud tööde vaatel on kaks osa. Selle andmed mida esitab külastajale ja andmete lisamise vorm. Vorm näidatakse ainult autenditud kasutajale, kuid selle aktiveerimise nupp on omaette element, mida tuleks kirjeldada (vt joonis 25). Tegemist on eelnevas paragrahvis mainitud nupuga sarnase elemendiga. Autor otsustas ühe nupu poolt, et säästa ruumi ja teha kasutajakogemus sujuvamaks. Nupu värviks sai sinine toon, et teavitada külastajat positiivsest funktsionaalsusest. Erinevalt punasest, mida inimesed seostavad tugeva emotsiooniga, on sinine palju sõbralikum ning tihti kasutatakse seda elementidel mis kinnitavad või lisavad. Nupp muutub kustutamise nupu sarnaseks vormi vaates, et viidata vormi sulgemise funktsionaalsusele.



Joonis 25. Vormiga seotud nupud.

Autor otsustas sisselogimisel kasutada hüpikakent (ingl *pop-up window*) (vt joonis 27). Vuetify pakub sellise funktsionaalsuse jaoks `dialog` komponenti. Akna disain pole muutunud arenduse jooksul märkimisväärselt. Ainuke lisa on *close* nupp, et väljuda aknast. Sisselogimise link on sinise luku ikooni kujul. Lukk muutub vastavalt autenditud olekule, ning animeerib ühes olekust teise kui kasutaja logib sisse või välja (vt joonis 26).

Joonis 26. Kasutaja sisse/välja logimise nupud



Joonis 27. Sisse logimis aken.

Rakenduse arendus ei pakkunud suurt tehnilist väljakutset, kuid projekti vältel tekkis kaks probleemi, millest ainult üks leidis korraliku lahenduse. Autor oli projekti algul plaaninud kasutada LinkedIn API'd, et tõmmata oma profiili alt sotsiaalvõrgustiku poolt heaks kiidetud oskused. Iga oskuse all on arv, mis näitab, mitu inimest on seda kinnitanud. Autor plaanis näidata neid oskusi `Sidebar` komponendi all ning demonstreerida, kuidas see arv muutub reaalajas. Kahjuks on LinkedIn otsustanud sulgeda oma API ning autoril ei õnnestunud neid andmeid tõmmata (Wright, 2015).

Teine suurim mure oli seotud kasutaja autentimisega. Autor uuris kõiki võimalike lahendusi, nagu cookie ja localStorage, kuid otsustas kasutada Firebase autentimise teenust. Autori põhjenduseks oli, et Firebase pakub valmis lahendust ja vastasel juhul peaks autor arendama oma loogika nullist. Aja säästülikuse nimel oli parem kasutada olemasolevat teenust.

Autori töö tulemus on saadaval GitHub repositooriumis aadressil https://github.com/fredkorts/bakatoo. Autor lisas projekti kirjelduse alla lühikese juhendi, kuidas paigaldada valminud rakenduse oma arenduskeskkonnas.

Kokkuvõte

Tänapäeval on portfoolio võimalus arendajal näidata, kes ta on, millist tööd on teinud ja mis suunas soovib areneda. Autor otsustas arendada oma rakenduse Vue.js põhjal, et näidata potentsiaalsele tööandjale oma oskust aktuaalse tehnoloogiaga. Käesoleva bakalaureusetöö eesmärgiks oli kirjeldada selle rakenduse arenduse etappe. Kogu protsess on kirjeldatud viies peatükis, millest rakenduse disain, rakenduse arendamine ja valminud rakendus moodustavad kõige mahukama osa.

Rakenduse disaini peatükis määrati arendatava projekti eesmärk. Eesmärgist tulenevalt analüüsis autor, milliseid komponente tuleks sellise rakenduse puhul kasutada. Enne disainijooniste loomist tuli autoril uurida sarnaseid lahendusi. Sellest peatükist tulenevad skeemid said aluseks arendatava rakenduse disainile ning suuremas osas jäi struktuur samaks.

Rakenduse arenduse peatükk on selle töö kõige suurem osa ning kirjeldab autori otsuseid projekti arenduskeskkonna ülesseadmisega ja tähtsamate komponentide ülesehitusega. Autor andis peatüki algul üldise vaate arendatava rakenduse tähtsamatest osadest. Peatüki vältel tutvustas autor Vue.js peamist objekti, komponente, komponentide vahelist suhtlust, marsruutimist ja välise Firebase teenuse kasutust andmete käsitlemisel. Koodinäited selles peatükis olid peamiselt illustratiivsed, et paremini seletada Vue.js süntaksist.

Viimases peatükis andis autor ülevaate valminud rakenduse komponentide ja moodulite vahelistest suhetest skeemi näol. Autor kirjeldas, kuidas toimuvad peamised rakenduse funktsionaalsused nagu andmete lugemine, lisamine ja kustutamine. Samas peatükis autor toob välja valminud rakenduse ekraanitõmmiseid ja võrdleb tulemust esialgsete kavanditega. Peatüki lõpus jagab autor kogemust probleemidega, mis rakenduse arenduse vältel aset leidsid.

Valminud rakendus on autori esimene katse sellises mahus Vue.js projektiga. Üldine mulje on, et tegemist on intuitiivse raamistikuga, mida võib kindlasti soovitada

arendajatele, kes otsivad madala õppekõveraga eesrakenduse tehnoloogiat oma järgmise projekti jaoks.

Autoril ei ole konkreetseid plaane valminud rakendusega, välja arvatud mõned jooksvad muudatused, millega autor tegeleb hobi korras. Valminud rakendus on saadaval autori GitHub repositooriumil aadressil https://github.com/fredkorts/bakatoo.

Kasutatud kirjandus

Andersen, B. (2017). *Vue Instance Lifecycle & Hooks*. Loetud aadressil https://codingexplained.com/coding/front-end/vue-js/vue-instance-lifecycle-hooks

Artmedia (Kuupäev puudub). *Eduka CV koostamine*. Loetud aadresil http://www.artmedia.ee/cv-koostamine/cv-koostamine/

Babich, N. (2017). *The Underestimated Power Of Color In Mobile App Design*. Loetud aadresil https://www.smashingmagazine.com/2017/01/underestimated-power-color-mobile-app-design/

Baytech. (2013a). *Creating an Effective Website Design Strategy.* Loetud aadresil https://www.baytechwebdesign.com/creating-an-effective-website-design-strategy/

Baytech. (2013b). *Choosing the Purpose of Your Website*. Loetud aadresil https://www.baytechwebdesign.com/choosing-the-purpose-of-your-website/

Firebase (Kuupäev puudub). *Authenticate with Firebase using Password-Based Accounts using Javascript*. Loetud aadresil https://firebase.google.com/docs/auth/web/password-auth

Gore, A. (2016) *WTF is Vuex? A Beginner's Guide To Vue's Application Data Store*. Loetud aadresil https://medium.com/js-dojo/vuex-for-the-clueless-the-missing-primer-on-vues-application-data-store-33fa51ffc3af

Green, A. (2017). *On your resume, which comes first – job title or company name?* Loetud aadresil http://www.askamanager.org/2017/06/on-your-resume-which-comes-first-job-title-or-company-name.html

Hefetz, A. (2017). *Bower is dead, long live npm. And Yarn. And webpack.* Loetud aadresil https://snyk.io/blog/bower-is-dead/

Leider, J. (Kuupäev puudub) *Introduction.* Loetud aadressil https://github.com/vuetifyjs/vuetify

Mertz, N. (2012). *How and Why Icons Improve Your Web Design*. Loetud aadresil http://blog.usabilla.com/how-and-why-icons-improve-you-web-design/

Monster Worldwide (Kuupäev puudub). *What are the basic elements of a CV?* Loetud aadresil https://www.monster.co.uk/career-advice/article/what-are-the-basic-elements-of-a-cv

Nielsen, J. (1995) *10 Usability Heuristics for User Interface Design.* Loetud aadressil https://www.nngroup.com/articles/ten-usability-heuristics/

O'Donovan, P. Agarwala, A. Hertzmann, A. (2011). *Color Compatability From Large Datasets*. Loetud aadresil http://www.dgp.toronto.edu/~donovan/color/colorcomp.pdf

Spradlin, L. (2014). *Exclusive: Quantum Paper And Google's Upcoming Effort To Make Consistent UI Simple*. Loetud aadresil

https://www.androidpolice.com/2014/06/11/exclusive-quantum-paper-and-googles-upcoming-effort-to-make-consistent-ui-simple/

Steiert, S. (2017). *Build fast and easy multiple beautiful resumes and create your best CV ever! Made with Vue and Less*. Loetud aadresil https://github.com/salomonelli/best-resume-ever/

Swift, J. (2018). *Stack Overflow Analysis on JavaScript Framework Trends*. Loetud aadresil http://www.i-programmer.info/news/167-javascript/11460-stack-overflow-analysis-of-javascript-framework-trends.html

Tutorialspoint. (Kuupäev puudub). *Node.js – Introduction.* Loetud aadresil https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm

Vue.js (Kuupäev puudub). *The Vue Instance*. Loetud aadresil https://vuejs.org/v2/guide/instance.html

Wright, M. (2015). *LinkedIn takes aim at developers with plans to lock down most of its APIs*. Loetud aadresil https://thenextweb.com/dd/2015/02/12/linkedin-takes-aim-developers-plans-lock-apis/

Summary

Vue.js Application Development by the Example of Personal Portfolio Site

Nowadays a portfolio is a way for the developer to introduce themselves by exhibiting their work and displaying their goals. The author chose to build a Vue.js application in order to show potential employers his skills with the latest technology in web development. The purpose of this bachelor's thesis was to describe the steps toward developing such an application. The author describes the entire process within five chapters of which the design and development chapters make up the bulk of the work.

The goal of the project was established at the start of the design chapter. The author analyzed the scope of the project based on this goal. Before any actual work could begin the author had to find and analyze works of similar nature. The wireframe models that resulted from this chapter were the basis for the structural design of the project.

The development chapter is by far the largest section of this bachelor's thesis and it describes the authors decisions in building his application. The author gave a brief introduction of the most vital parts of his application at the start of this chapter. During this chapter the author introduces the main Vue.js object, components, communication between components, routing and using the Firebase service to manage data. The author does offer some code examples throughout the chapter but they are mostly illustrative in order to better explain certain parts of the Vue.js syntax.

In the last chapter the author describes the finished application and the many relations between its component parts in the form of a schematic. The author describes the main functionalities of the application like reading, adding and deleting data. In the same chapter the author compares the initial wireframe models with the finished design. At the end of the chapter the author shares his experience with the main problems that occurred during development.

The finished application is the authors first attempt at a Vue.js project of this scope. The general impression left on the author is that the Vue.js framework is very intuitive and that it's perfect for people who are looking for a framework with a low difficulty curve.

The author doesn't have any concrete plans with the finished application in the near future. The finished application is available on the authors GitHub repository located at https://github.com/fredkorts/bakatoo.