TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Maksim Suhhomjatnikov 179458IABB

# Kasutajaliidese komponentide teegi Atomic-UI loomine

Bakalaureusetöö

Juhendaja: Karl-Erik Karu
MSc

Tallinn 2022

# Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Maksim Suhhomjatnikov
18.05.2022

# Annotatsioon

Selle bakalaureuse lõputöö käigus luuakse kasutajaliidese arendamise jaoks komponenditeek. Antud komponenditeek annab võimaluse kasutajale luua kasutajaliideseid kasutades eelnevalt valmistatud ehitusplokke Samas on kasutajal lubatud muuta ehitusplokki omal soovil. Tulemuseks on võimalus arendada lihtsasti üksteisest erinevat kasutajaliidest.

Antud lõputöö on võimalik jagada kolmeks osaks. Esimeses osas on väljatoodud kasutajaliidese ja komponenditeegi arendamise teooria. Teises osas on esitatud komponenditeegi arhitektuur ja projekti realiseerimine. Kolmandas osas on väljatoodud fokusgruppide analüüs ning edasiarendamise visioon.

Komponenditeegi loomisel oli kasutatud ReactJS ja StoryBook raamistikke. Lõputöö käigus loodud veebirakendus asub järgneval veebiaadressil: https://msuhhomjatnikov.github.io/atomic-ui/

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 42 leheküljel, 4 peatükki, 30 joonist.

# Abstract

## Creation of user interface component library Atomic-UI

In this graduation thesis a component library has been created for building user interfaces. This component library provides an opportunity for the user to create user interfaces with pre-created building blocks. User can use an option to modify components at own wish. The result is a new way to develop non-similar user interfaces.

This thesis can be divided into three parts. The first part presents the theory of user interface and component library development. The second part presents the architecture of the component library and the realization of the project. In the third part, the analysis of focus groups and the vision of further development are presented.

ReactJS and StoryBook frameworks were used to create the component library. The web application created during the thesis can be found at the following web address: https://msuhhomjatnikov.github.io/atomic-ui/

The thesis is in Estonian and contains 42 pages of text, 4 chapters, 30 figures.

# Lühendite ja mõistete sõnastik

| | |
|---|---|
| API | Application Programming Interface, rakendusliides |
| Babel | On JavaScripti kompilaator |
| CSS | Cascading Style Sheets, on küljendamisel kasutatav märgistuskeel. |
| DOM | Document Object Model, on platvormist ja keelest sõltumatu suhtlemise liides. |
| ECMAScript | On Ecma International poolt standardiseeritud programmeerimiskeel |
| ES6, ES7, ES8 | EMCAScripti erineva versiooniga standardid |
| Front-End | Veebilehele ilmuv kasutajaliides |
| HTML | HyperText Markup Language, on keel, milles märgendatakse veebilehti. |
| IE | Internet Explorer |
| JS | JavaScript, on OOP programmeerimiskeel |
| JSX | ReactJS-i osa, on JavaScripti keelelaiend |
| LESS | Leaner Style Sheets, on CSS-i tagasiühilduv keelelaiend |
| NodeJs | On mitmeplatvormne JavaScripti käitussüsteem |
| NPM | Node.js Package Manager, paketihaldur, on osana Node.jsst. |
| OOP | Object Oriented Programming, objektorienteeritud programmerimine |
| TDD | Test-driven development, on tarkvaraarenduse meetod |
| UI | User Interface, kasutajaliides |
| webpack | On JavaScripti mooduli koostaja |
| VR | Virtual Reality, virtuaalrealsus |
| Vue | JS raamistik kasutaja liideste loomiseks |

# Sisukord

# Jooniste loetelu

# 1. Sissejuhatus

## 1.1 Taust ja probleem

Kasutajaliides (edaspidi: UI) on interaktsiooniviis kasutaja ja programmi vahel ning nagu öeldakse: hea UI võtab arvesse inimlikke nõrkusi, edastab töö masinale, minimeerib vigu ja kasutaja häirimist. [1] See tähendab, et kui iga telefoni või arvutisse installitud rakendus või programm on abiline, siis liides on see, mis aitab sellega suhelda ehk interakteeruda, et sellest oleks abi igapäevastes tegevustes.

Aastatepikkuse töö käigus Front-end insenerina autor on kasutanud kasutajaliidese arendamiseks mitmeid komponenditeeke. Nende kasutamine oli mugav, kuid mitte piisavalt. Põhiline probleem, millega autor on kokku puutunud, on nende komponentide muutmine. Kulus palju tööd, et need vastaksid antud disainile.

## 1.2 Eesmärk

Töö eesmärgiks on luua komponentide teek, mis võimaldab kasutajal antuid komponente hõlpsalt muuta. Lisaks hakkab teek sisaldama juba olemasolevaid stiile, millega kasutaja saab töötada ja neid kasutada. Eesmärgi saavutamiseks on vaja:

- Arendada komponentide projekti disaini
- Valida komponentide projekti arhitektuuri
- Määratleda fokusgrupid selleks, et koguda ning analüüsida erinevat tagasisidet oodatava tulemuse suhtes
- Analüüsida olemasolevaid komponentide projekte

## 1.3 Töö struktuur

Töö esimeses peatükis vastatakse järgmistele küsimustele "Mis on kasutajaliidese komponent?", "Mis on komponenditeek?", "Millal peaksin komponenditeeki kasutama?" ja "Millised on komponenditeegi eelised?". Arutatakse digisüsteemide liidesed, atomaarse disaini metoodikat ja tööriistad projekti realiseerimiseks. Peatüki lõpus teostatakse teiste komponenditeekide analüüs.

Töö teises peatükkis vaadatakse projekti realiseerimise protsessi üle. Selles peatükis autor kirjeldab kõik vajalikud sammud selleks, et luua ja testida esimese komponendi kasutades ReactJS-i ja näidata seda kasutajatele GitHub Pages ja StoryBook abil.

Töö kolmandas peatükis analüüsitakse komponentide testimisel saadud andmeid kolmes fookusgrupis ja määratletakse edasiarendamise visioon.

# 2. Metoodika

## 2.1 Objekt

Järgnevalt antakse ülevaade liidese tüüpidest, komponenditeegist ja töö fookuses olevast objektist ehk komponendist.

### 2.1.1 Digisüsteemide liidesed

Digisüsteemide liidesed ise võivad olla erinevad: graafilised, hääle-, žesti- või lihtsalt käsureana. Nende kaudu saame juurdepääsu uutele võimalustele, mida rakendus või programm meile annab.

- Käsurealiides ehk tekstiliides on üks kasutaja ja arvuti vahelise interaktsiooni esimesi viise. See on nagu otsatu lõuend, millele kasutaja sisestab käskude teksti ja saab töö tulemuse teksti kujul.
- Hääleliides aitab meil süsteemi häälega juhtida. Näiteks häälassistendid Siri Apple'il või Alexa Amazonil.
- Žestiliidesed aitavad meil süsteemi juhtida, näiteks käe asendit muutes. Neid võib kohata Xboxi, PlayStationi või Nintendo konsoolide videomängudes.

Graafilisi võib jagada 3 tüüpi: mobiililiidesed, veebiliidesed või mängu- ja materjaliliidesed.

- Mängu- ja materjaliliides on vahetult seotud mängitava mängu mehaanikaga. Liides erineb mängust: nupud, žestid, hiireliigutused, mängupuldi klahvivajutused või VR-i (virtuaalreaalsus) 3D-liides.
- Veebiliides on tehnoloogia, mis võimaldab luua täisväärtuslikke veebirakendusi, mis ei jää töölauatarkvarale alla. Selliste rakenduste eeliseks on see, et neid ei pea kuhugi installima, kõik on brauseris saadaval.
- Mobiililiidesed on tavaliselt mobiilseadme puuteekraan, mis nõuab teistsugust lähenemisviisi kui veebiliides. Kasutaja käitumine nutitelefonidega suhtlemisel erineb arvutiga töötamisest ekraani suuruse ja eraldi hiire / puuteplaadiga klaviatuuri puudumise tõttu. Siinsed elemendid täidavad ekraani täielikult ning plokid ja süsteemid sõltuvad operatsioonisüsteemi nõuetest.

Levinud on ka muud liidesed, näiteks:

- Riistvaraliides – ühendab kaks objekti, näiteks telefoni ja arvuti.
- Programmiliides (API) – loob rakenduste vahel ühenduse, näiteks autoriseerimine sotsiaalvõrgustike kaudu veebilehtedel.
- Riistvara/-tarkvaraliides – tarkvara juhitavate tehniliste elementide kombinatsioon.

### 2.1.2 Komponent

Antud töö keskpunktiks on komponent ehk olem/objekt veebilehel. Mõistet "komponent" kasutatakse tavaliselt ka siis, kui käsitletakse süsteeme ja arendamist üldiselt. Antud töös aga keskendun elemendile, mida kasutaja saab mingil viisil näha või millega interakteeruda. Lühidalt öeldes on komponendid rakenduse mis tahes osa, mida saab loogiliselt rühmitada ja käsitleda ühe elemendina, mida saab ideaaltingimustes ülejäänud rakenduse osa standardplokina uuesti kasutada. [2]

Üksikasjalimalt käsitledes on komponent üks element või elementide rühm, mis koos loovad objekti. Komponendi ülesehitus võib olla HTML-i elementidest või see saab koosneda muudest komponentidest. Tavaliselt ei sõltu komponent mingist infrastruktuurist. Kuid tavaliselt on komponendis mingi loogika või omadused, mis võivad selle stiili ja käitumist muuta. Raamistikupõhiste komponentide (nt React või Vue) olemasolu teeb need interaktsioonid lihtsamaks. Reacti rakenduses töötades on Reacti-põhiste komponentide kasutamine mõttekas. Raamistiku sees loodava komponendi väljunditeks on alati HTML, CSS ja JavaScript.

Nagu nimetatud, võivad komponendid koosneda ka muudest komponentidest. Sarnased suuremad komponendid muudavad selle haldamise tülikamaks. Kuna üks komponent sõltub mitmest teisest, muutub see kiiresti keerukamaks. Levinud lähenemisviis komponentide struktureerimisele on Atomic Design. Mõiste Atomic Design võttis kasutusele Brad Frost [3]**Error! Reference source not found.**.

### 2.1.3 Komponenditeek

Komponenditeekide vaatlemisel võib mõelda suurele projektile, kus on palju arendajaid ja disainereid. Kus ülesandeks on projektile või ettevõttele standardite ja põhimõtete kehtestamine. Samas ei olene komponenditeegi kontseptsioon ja selle eelised rakenduse või projekti suurusest. Väidan, et väike projekt peaks komponenditeegist saama sama palju kasu kui suurem projekt. Komponenditeek on korduvkasutatavate komponentide kogum.

See võib olla projektisisene kaust, mis sisaldab rakenduses kasutatavaid üldkomponente. See võib olla eraldatud pakett npm-is. See võib olla ka suurema projekteerimissüsteemi osa. Komponenditeek võib olla väike või suur. Puuduvad ametlikud eeskirjad selle kohta, mida komponenditeek tähendab. See sõltub projektist ja vajadustest. Need võivad olla enda loodud komponendid või suure ettevõtte avalik teek.

Siiski jääb õhku küsimus, millal peab komponenditeeki looma ja kasutama? Kui kasutatakse analoogset, identset komponenti kahes või enamas erinevas kohas, on soovitatav luua üldkomponent ja paigutada see teegi kausta. Seda komponenti saab seejärel kasutada nendes ja edaspidistes kohtades. Kui komponendid peavad mingil moel eristuma, on alati võimalus seda omaduste või muu loogika abil kohandada, et määratleda selle lõplik olemus.
Kõige klassikalisem näide on tavaline nupp. Tõenäoliselt kasutatakse nuppe rakenduses paljudes kohtades ja see, et nupp asub komponenditeegis, annab võimaluse selle nupu jaoks kasutada sama komponenti. Vastutasuks saab rakendus ühtse ilme ja tunnetuse ning suurepärane on see, et muudatuste vajaduse korral tuleb need teha ühes kohas.

Komponenditeegi kasutamine pakub arendusprotsessis suuri eeliseid. Üks komponenditeegi kasutamise suuri eeliseid on see, et on üks usaldusväärsete komponentide allikas. Erinevates kohtades ei rakendata variatsioone. Kogu HTML, stiil ja loogika on leitavad ühest kohast. See muudab ühiskasutuse lihtsamaks. Uut projekti saab alustada väga kiiresti ja saada kiire juurdepääs komponentidele. Komponendi muudatused kajastuvad kõigis projektides.

Lisaks muudab komponentide ühe allika olemasolu palju lihtsamaks disaineritega suhtlemise ja ühise arusaamani jõudmise nende komponentide välimusest ja tunnetusest. Näiteks on võimalus

lisada Storybooki, mida käsitletakse antud töös veidi hiljem, et oleks ühine koht komponentide kuvamiseks. Hiljem saab kohtuda koos disaineritega ja huvitatud osapooltega, et arutada muudatusi ja uusi funktsioone. See muudab disainiga töötamise iteratiivsemalt lihtsamaks. Uusi versioone saab hõlmata kiiremini ja uusi asju on lihtsam proovida.

Kui kõik komponendid on ühes kohas, on nende hooldus lihtsam. Komponentide värskendamine või uute funktsioonide lisamine toimub üks kord ja see kajastub selle komponendi igas teostuses. Uute värskenduste lisamine on palju kiirem ja järjekindlam. Organisatsioonis töötamise puhul on uutel töötajatel lihtsam aimu saada arhitektuurist ja sellest, milliste "kividega" peate mängima.

Lihtsam on luua teegis läbimõeldud testide kogumi komponentidele, kui need on rühmitatud ja teostatud ühes kohas. Testid muutuvad asjakohasemaks ja olulisemaks. Uute funktsioonide lisamisel on lihtne kasutada ka TDD [4], luues enne funktsiooni juurutamist uued testid. See muudab projektid töökindlamaks.

## 2.2 Atomaarse disaini metoodika

Looduses aatomielemendid ühinevad, moodustades molekule. Need molekulid võivad täiendavalt ühineda, moodustades suhteliselt keerukaid organisme. Täiendav näitlikustamine:

- Aatomid on kogu mateeria peamised ehitusplokid. Igal keemilisel elemendil on erinevad omadused ja neid ei saa tähendust minetamata edasi lagundada.
- Molekulid on kahe või enama aatomi rühmad, mida hoiavad koos keemilised sidemed.
- Organismid on molekulide kogumid, mis toimivad koos üksusena.

Põhimõtteks jääb see, et aatomid ühinevad molekulideks, mis seejärel ühinevad organismide moodustamiseks. See aatomiteooria tähendab, et kogu teadaolevas universumis leiduva mateeria saab jaotada lõplikuks aatomielementide hulgaks.

Nagu käsitletakse antud töös varem, kuidas kogu universumi mateeriat saab jaotada lõplikuks aatomielementide hulgaks. Nii toimub ka liidestega, liideseid saab lagundada sarnaseks piiratud elementide kogumiks. Kuna on sarnane lõplik "ehitusplokkide" kogum, saame ~~oma~~

kasutajaliideste projekteerimiseks ja arendamiseks rakendada sama protsessi, mis toimub loodusmaailmas.

Atomaarne disain [3] on metoodika, mis koosneb viiest erinevast etapist, mis toimivad koos, et luua liideste disainisüsteemid läbimõeldumal ja hierarhilisemal viisil. Atomaarse projekteerimise viis etappi:

- Aatomid
- Molekulid
- Organismid
- Mallid
- Lehed

Atomaarne disain ei ole lineaarne protsess, vaid pigem vaimne mudel, mis aitab arnendajatel ja disaineritel mõelda oma kasutajaliidestest kui ühtsest tervikust ja detailide kogumist üheaegselt. Kõik viis etappi mängivad liideste projekteerimise süsteemide hierarhias võtmerolli. Järgnevalt antakse ülevaade mainitud etappidest.

### 2.2.1 Aatomid

Kui aatomid on mateeria peamised ehitusplokid, on liideste aatomid fundamentaalsed ehitusplokid, mis moodustavad kõik meie kasutajaliidesed. Need aatomid sisaldavad põhilisi HTML-i elemente, nagu tekstisisendid, nupud, sildid ja muud elemendid, mida ei saa edasi lagundada ilma tööd katkestamata.

Mallide teegi kontekstis näitavad aatomid esmapilgul kõiki teie põhistiile, mis võib olla kasulik viide, mille juurde projekteerimissüsteemi arendades ja hooldades naasta**Error! Reference source not found.**.

### 2.2.2 Molekulid

Liidestes on molekulid suhteliselt lihtsad UI elementide rühmad, mis toimivad koos üksusena. Näiteks saab otsinguvormi molekuli loomiseks kombineerida silti, teksti sisestusvälja ja nuppu.

Elementide ühendamine lihtsateks funktsioneerivateks rühmadeks on see, mida kasutajaliideste loomisel on alati tehtud.

### 2.2.3 Organismid

Organismid on suhteliselt keerukad kasutajaliidese komponendid, mis koosnevad molekulide ja/või aatomite ja/või muude organismide rühmadest. Need organismid moodustavad liidese erinevad osad.

### 2.2.4 Mallid

Mallid on lehetaseme objektid, mis paigutavad komponendid maketti ja formuleerivad sisu põhistruktuuri.

### 2.2.5 Lehed

Lehed kohaldavad mallidele tegelikku sisu ja formuleerivad variante, et esitleda lõplikku kasutajaliidest ja testida projekteerimissüsteemi töökindlust.

Antud viis etappi moodustavad atomaarnse disaini. Need viis erinevat etappi toimivad koos, et luua tõhusaid kasutajaliidese projekteerimissüsteeme.

## 2.3 Projekti teostamise kasutatud tööriistad

### 2.3.1 ReactJS

Komponenditeegi teostamiseks kasutati ReactJS-i. React [5], [6] on JavaScriptil [7] põhinev teek kasutajaliidese arendamiseks. Järgnevalt esitatakse peamised põhjused miks valiti React tööriistaks:

- Dünaamiliste rakenduste lihtne loomine: React muudab dünaamiliste veebirakenduste loomise lihtsaks, sest nõuab vähem kodeerimist ja pakub rohkem funktsioone, erinevalt JavaScriptist, kus kodeerimine muutub sageli väga kiiresti keeruliseks [5], [6].
- Parem jõudlus: React kasutab Virtual DOM-i, luues seega veebirakendusi kiiremini. Virtual DOM võrdleb komponentide eelmisi olekuid ja värskendab ainult Real DOM-i elemente, mis on muutunud, mitte ei värskenda kõiki komponente uuesti, nagu seda teevad tavalised veebirakendused [5], [6].
- Korduvkasutatavad komponendid: komponendid on iga Reacti rakenduse oluline osa ja üks rakendus koosneb tavaliselt mitmest komponendist. Nendel komponentidel on loogika

ja juhtelemendid ning neid saab kogu rakenduses uuesti kasutada, mis omakorda vähendab oluliselt rakenduse arendamise aega [5], [6].

- Ühesuunaline andmevoog: React järgib ühesuunalist andmevoogu. See tähendab, et Reacti rakenduse arendamisel pesastavad arendajad sageli alamkomponendid (child) ülemkomponentide (parent) sisse. Et andmed liiguvad ühes suunas, on lihtsam vigu siluda ja kiiresti tuvastada, kus rakenduses probleem parasjagu esineb [5], [6].

Mudelivaate kontrolleri (MVC) arhitektuuris on React "view", mis vastutab rakenduse välimuse ja tunnetuse eest.

### 2.3.2 StoryBook

Lisaks ReactJS-ile leiab teegi teostamisel kasutust Storybook. Storybook - on JavaScripti instrument, mis võimaldab arendajatel luua organiseeritud kasutajaliidese süsteeme, muutes loomisprotsessi ja dokumentatsiooni tõhusamaks ja kasutajasõbralikumaks. Kui komponent on loodud, võimaldab Storybook luua "materjali" faili, millesse saab komponendi importida ja luua isoleeritud iFramed-programmikeskkonnas selle komponendi abil erinevaid kasutusnäiteid [8]**Error! Reference source not found.**.

### 2.3.3 GitHub Pages

Komponenditeegi ja nende komponentide kasutamise dokumentatsiooni mugavaks eelvaateks leiab kasutust GitHub Pages. GitHub Pages on veebisaitide staatiline majutusteenus, mis aktsepteerib HTML-i, CSS-i ja JavaScripti faile otse GitHubi hoidlast, käitab faile ehitusprotsessis valikuliselt ja avaldab veebisaidi [9].

## 2.4 Teiste komponenditeekide analüüs ja võrdlus

Antud töös valmiva komponenditeegi loomiseks analüüsitakse neid mida autor on kõige rohkem ja kõige sagedamini kasutanud. Sellisteks teekideks on Material UI ja Bootstrap Components.

### 2.4.1 Material UI

Material UI [10] on avatud lähtekoodiga liidese struktuur Reacti komponentidele. See on loodud Lessi abil. Less (Leaner Style Sheets) on CSS-i tagasiühilduv keelelaiend. Material UI põhineb Google'i projektil "Material" ja pakub frontend-graafika arendamisel kvaliteetset digikogemust.

Material UI eelised:

- Hea dokumentatsioon, mis muudab Material UI-ga töötamise üldiselt hõlpsamaks.
- Komponenditeegi värskendused on regulaarsed.
- Komponendid ise on disainilt ja värvitoonilt ühtsed, mis on arendatud rakenduses või veebilehel visuaalselt väga atraktiivne.

Oma maitsest lähtuvalt tooksin puudustena välja järgmised punktid:

- Material Designi animeeritud üleminekute laialdane kasutamine aitab küll liidese ellu äratada ja annab sellele elava isikupära, kuid animatsioonid võivad tekitada vastupidise tulemuse, sest inimaju on visuaalse stimulatsiooni suhtes väga tundlik ja animatsioon põhjustab kognitiivseid katkestusi, mis võivad häirida kasutaja mõtlemist. Samuti võivad animatsioonid kulutada palju süsteemiressursse ja kokkuvõttes tuua rakenduse kasutamisel mitte alati meeldiva kogemuse.
- Komponentide stiilide kohandamine võib olla väga aeganõudev. Väga lihtsate muudatuste, komponendi laiuse või värvi muutmise puhul ei nõua see palju probleeme.

```
<Slider
  defaultValue={30}
  sx={{
    width: 300,
    color: 'green',
  }}
/>
```

Joonis 1. Material UI komponendi värvi jalaiuse muutmine

Suuremad muudatused eeldavad aga enamat. Näiteks komponendi oleku muutmiseks on koodi välimus väga kohmakas:

```
const SuccessSlider = styled(Slider)<SliderProps>(({ theme }) => ({
  width: 300,
  color: theme.palette.success.main,
  '& .MuiSlider-thumb': {
    '&:hover, &.Mui-focusVisible': {
      boxShadow: `0px 0px 0px 8px ${alpha(theme.palette.success.main, 0.16)}`,
    },
    '&.Mui-active': {
      boxShadow: `0px 0px 0px 14px ${alpha(theme.palette.success.main, 0.16)}`,
    },
  },
}));
```

Joonis 2. Material UI komponendi oleku muutmine

- Material UI piirab teiste kaubamärkide tõhusust disainisüsteemi kasutamisel. Jah, disainerid võivad kaubamärgi identiteedi toetamiseks lisada logosid, värvipalette ja muid eristavaid tegureid, kuid toode, mis järgib materjali projekteerimise spetsifikatsioone, on peaaegu alati seotud ka Google'iga.

### 2.4.2 Bootstrap Components

Nagu Material UI, on ka Bootstrap Components [11] ehitatud Lessi põhjal, mis kompileeritakse NodeJS-i abil. Bootstrap loodi mitte selleks, et olla suurepärase välimusega ja käituda hästi uusimates lauaarvutibrauserites (sh IE), vaid ka tahvelarvutite ja nutitelefonide brauserites Responsive CSS-i kaudu.

Positiivne on see, et Bootstrap aitab hoida kõvasti aega kokku, kuna sellel on ka väga hea dokumentatsioon. Samuti aitab see vältida vigu brauserites, sest kuna tegu on avatud lähtekoodiga projektiga, võib igaüks esitada brauseri vigu ja koodiparandusi. See on arendaja jaoks äärmiselt väärtuslik ressurss, sest võite olla kindlad, et kogukond on teie koodi tavaliste brauseri vigade kõrvaldamiseks täiustanud.

Puuduste poolelt võin välja tuua kolm punkti:

- Mõne projekti puhul tuleb palju klasse ja stiile ümber kirjutada. Mõned stiilid ei pruugi kompileeruda ja need tuleb uude faili ümber kirjutada.

- Ilma CSS-i muudatusteta näevad rakendused ja veebilehed samasugused välja, sest tegu on sama visuaalse stiiliga komponentide kogumiga.
- Selle omandamiseks kulub omajagu aega. Selle teegi tõhusaks kasutamiseks peab teadma palju komponendiklasse ja nende klasside kombinatsioone.

### 2.4.3 Järeldused

Antud töös valmiva teegi teostamiseks saab välja tuua järgmised punktid:

- Komponentidega töötamise hõlbustamiseks peab olema hästi kirjutatud dokumentatsioon. See peab sisaldama: teegi projekti paigaldamise protsessi, nende komponentide ja nende parameetrite kasutamise kirjeldust.
- Töötada välja süsteem komponentide kohandamiseks nii, et see oleks enamikule kasutajatele harjumuspärane.
- Pakkuda komponentide kohandamiseks värvipaletti.
- Muuta komponendid võimalikult tõhusaks, et need nõuaks vähe ressursse.

# 3. Peamised tulemused

## 3.1 Rakenduse initsialiseerimine

### 3.1.1 Yarn

React-rakendusega töötamise alustamiseks peab valima paketi installeri. Antud töös valiti Yarn kuna selle eeliseks on see, et ta võimaldab installida pakette paralleelselt, erinevalt npm-ist, mis installib need jadamisi. Tänu sellele saab see suurte failidega kiiremini hakkama. Samuti teostab Yarn pakettide allalaadimise ajal taustal turbekontrolli. See kasutab paketi litsentsiteavet tagamaks, et see ei laadiks pahatahtlikke skripte ega põhjustaks sõltuvuskonflikte, ning kontrollib kontrollsummade pakette turvalise andmeedastuse tagamiseks.

Siiski on esmaseks installimiseks soovitatav kasutada npm-i. Pärast npm-i installimist saab Yarni installimiseks ja värskendamiseks teha järgmised toimingud.

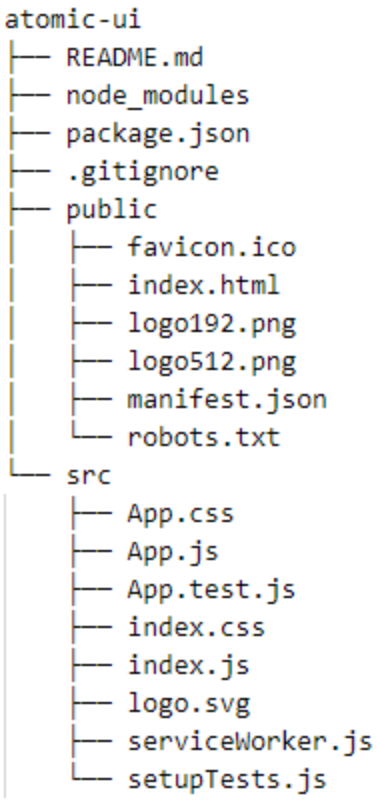
```
npm install --global yarn
```

### 3.1.2 React-rakendus

React-rakenduse loomiseks peab arvutis olema Node.js versioon 14 või uuem [5]. Rakenduse initsialiseerimiseks käsureal tuleb kirjutada järgmine käsk:

```
yarn create-react-app atomic-ui
```

Selle käsu täitmisel luuakse praeguses kaustas kataloog atomic-ui. Selles kaustas luuakse projekti esialgne struktuur ja installitakse üleminekusõltuvused:




Joonis 3. React rakenduse esialgne struktuur

Järgmisena kontrollime, kas rakendus loodi õigesti. Selleks sisestame terminali jadamisi järgmised käsud:

```
cd atomic-ui
yarn start
```

### 3.1.3 Package.json fail

Fail package.json [12] on projekti manifest. See suudab teha paljusid asju, mis pole üldse omavahel seotud. See on instrumentide keskne konfiguratsioonihoidla.

### 3.1.4 StoryBooki integreerimine

StoryBooki [8] integreerimise eesmärgil projekti. Kataloogis `./atomic-ui` olles tuleb kirjutada järgmine käsk:

```
yarn storybook
```

On oluline, et algne rakendus oleks juba initsialiseeritud. Eespool märgitud käsk teeb kohalikus keskkonnas järgmised muudatused:

- Installib vajalikud sõltuvused.
- Installib Storybooki käivitamiseks ja koostamiseks vajalikud skriptid.
- Lisab Storybooki vaikekonfiguratsiooni.
- Lisab töö alustamiseks mõne malliloo.

Järgmisena saab rakenduse seest eemaldada mõned failid ja kaustad ning struktuur tuleb selline:

```
atomic-ui
├── .storybook
│   ├── main.js
│   ├── preview.js
├── README.md
├── node_modules
├── package.json
├── .gitignore
└── src
    ├── stories
├── Button.js
├── Button.stories.js
…
    ├── App.js
    ├── index.js
```

Joonis 4. Struktuur pärast StoryBook-i initsialiseerimist

Faili `package.json` sees on jaotise `scripts` ilme nüüd järgmine:

```
"scripts": {
    ...,
    "storybook": "start-storybook -p 6006 -s public",
    "build-storybook": "build-storybook -s public"
},
```

Joonis 5. StoryBook skriptid käivitamiseks ja koondamiseks

Esimene skript võimaldab StoryBooki kohapeal käivitada, teine koondab kogu projekti selle edaspidiseks kasutamiseks mujal.

Ilmus ka kaust `.storybook`, mille sees on kaks faili: `preview.js` ja `main.js`. Esimene vastutab materjalide visualiseerimise juhtimise ning globaalsete dekoraatorite ja parameetrite lisamise eest, teine aga juhib Storybooki serveri käitumist.

### 3.1.5 GitHub

Järgmiseks tuleb luua hoidla, kuhu projekt ise salvestatakse. Selles abistavad Git [13] ja GitHub [14]. Git on hajus versioonihaldussüsteem ja GitHub on veebiteenus IT-projektide majutamiseks ja nendega ühisarenduseks. Projekti paigutamiseks platvormile loome GitHubi keskkonnas hoidla ja kasutame järgmist käskude kogumit, needki sisestatakse jadamisi:

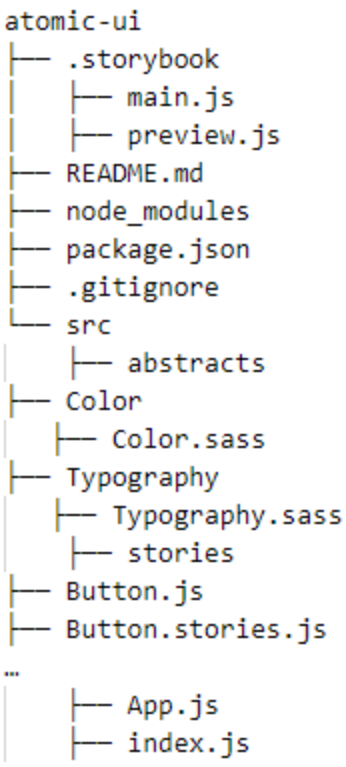
```
git remote add origin git@github.com:msuhhomjatnikov/atomic-ui.git
git branch -M main
git push -u origin main
```

### 3.1.6 SASS

Selles projektis kasutan SASS-i. SASS (Syntactically Awesome Styleseets) [15] on CSS-i eelprotsessor, mis võimaldab kasutada muutujaid, matemaatilisi tehteid, mixine, silmuseid, funktsioone, importi ja muid huvitavaid funktsioone, mis muudavad CSS-i kirjutamise palju võimsamaks. Täiendame SASS-i järgmise käsuga:

```
yarn add sass
```

Nüüd saab kiirendada kogu edasist tööd komponentide värvide ja fontidega [1], [2], samuti parandada stiilide kirjutamist. Muudame projekti struktuuri järgmiselt ja lisame uued failid värvi- ja fondistiilide jaoks:




Joonis 6. Projekti struktuur peale abstracts kausta lisamist

Tulemuseks on nüüd kaust abstracts, mis sisaldab muutujaid värvi ja fondi jaoks. Edaspidi saab seda kasutada nagu selles näites:

```
@import "../../abstracts/Typography/Typography.sass"

.status-badge
  display: inline-flex
  align-items: center
  height: 22px
  border-radius: 3px
  padding: 5px 6px
  white-space: nowrap
  color: var(pink-550)
  @include body-m-bold
```

Joonis 7. SASS klass koos värvi ja fondi muutujatega

### 3.1.7 Komponentide struktuur

Komponentide loomiseks tuleb esmalt kõik nõuetekohaselt struktureerida. Selleks tuleb luua juurkaustas src uue, nimega components. Siis nimetada järgmise kausta komponendi nimega, antud töös juhtumis siis Chip. Iga komponent sisaldab nelja faili: (Chip.js), stiilide fail (Chip.sass), testidega fail (Chip.test.js) ja fail eksportimiseks (index.js). Samuti võib olla kaust assets, mis salvestab erinevaid kujutisi. Struktuur on siis järgmine:

```
└── src
    ├── abstracts
├── Color
   ├── Color.sass
├── Typography
   ├── Typography.sass
    ├── components
├── Chip
   ├── assets
   ├── Chip.js
   ├── Chip.Sass
   ├── index.js
   ├── Chip.test.js
...
    ├── stories
    ├── App.js
    ├── index.js
```

Joonis 8. Komponentide struktureerimine

### 3.1.8 Esimese komponendi loomine

Reactis saab komponente väljendada kahel viisil: funktsioonina ja ES6-klassina [5], [6]. Reacti vaatenurgast on need kaks komponenti ekvivalentsed.

```
function helloWorldComponent() {
  return <h1>Hello World</h1>;
}
```

Joonis 9. Komponent funktsioonina

```
class helloWorldComponent extends React.Component {
  render() {
    return <h1>Hello World</h1>;
  }
}
```

Joonis 10. Komponent ES6-klassina

Komponent võib hõlmata parameetreid (nn props), mida saab sisemiselt kasutada. Label, isActive, onChecked – props, mis edastatakse komponendile.

```
const Chip = ({
  label,
  isActive,
  onChecked,
  classNames,
}) => {
  const handleClick = () => {
    onChecked((old) => !old);
  }

  if (!label) return null;

  return (
    <>
      <button className={`chip ${classNames.container ? classNames.container : ''}`} onClick={handleClick}>
        {isActive ? <CheckCircle /> : null}
        <span className={`chip__label ${classNames.item ? classNames.item : ''}`}>{label}</span>
      </button>
    </>
  );
};
```

Joonis 11. Komponent ilma memoriseerimiseta

Reacti üks olulisemaid osi on memoriseerimine . Eespoolses näites renderdatakse see komponent uuesti, kui midagi väljaspool seda komponenti muudetakse. Seetõttu saab teha mõned muudatused: esmalt funktsioonis handleClick ja seejärel komponendis endas. Funktsioon muutub, kui funktsiooni sisest parameetrit muudetakse, ja komponent ise muutub, kui muudetakse mistahes aspekti. Selle tulemusena suurendab see komponentide ja rakenduse enda jõudlust [5].

```
const Chip = memo(function Chip({
  label,
  isActive,
  onChecked,
  className,
}) {
  const handleClick = useCallback(() => {
    onChecked((old) => !old);
  }, [onChecked]);

  if (!label) return null;

  return (
    <>
      <button className={`chip ${classNames.container ? classNames.container : ''}`} onClick={handleClick}>
        {isActive ? <CheckCircle /> : null}
        <span className={`chip__label ${classNames.item ? classNames.item : ''}`}>{label}</span>
      </button>
    </>
  );
});
```

Joonis 12. Memoriseeritud komponent

Samuti tuleb määrata saadavate parameetrite tüübid. Suur abi on projektist prop-types [16], kui rakendus on kirjutatud JS-is.

```
yarn add prop-types
```

Seejärel näitame komponendi funktsiooni all nende tüübid:

```
Chip.propTypes = {
  label: string,
  isActive: bool,
  onChecked: func,
  classNames: object,
};
```

Joonis 13. Komponendi tüübid ja nende defineerimine

Lisaks kõigele sellele tuleb komponent eksportida, et seda saaks ka mujal kasutada.

```
export default Chip;
```

`export default` leiab kasutust ühe klassi, primitiivi või funktsiooni eksportimiseks moodulist. Seda eksportimist kasutatakse failis `index.js` komponentide importimise lihtsustamiseks kolmandate osapoolte failidesse.

### 3.1.9 Parameeter classNames

Komponendil on parameetrites esitatud üks väli – classNames. See väli võimaldab kasutajal komponendi iga osa oma äranägemisel muuta. See parameeter on objekt, mis hõlmab klasse. Varem esitatud komponendi puhul võib objektis classNames olla kaks välja:

```
const classNames = {
    container: 'some-class-name',
    item: 'some-class-name',
};
```

Joonis 14. Komponentide väli, mis võimaldab selle muuta

### 3.1.10 StoryBook Story

Komponendi Chip esimese Story loomiseks tuleb luua fail, mis sisaldab sõna stories: `Chip.stories.js`. Sest StoryBooki konfiguratsioon ütleb, et see näeb ainult faile, mis vastavad järgmistele parameetritele:

```
"stories": [
  "../src/**/*.stories.mdx",
  "../src/**/*.stories.@(js|jsx|ts|tsx)"
],
```

Joonis 15. Konfiguratsioon Story loomiseks

Faili sees ekspordime story konfiguratsiooni ja määrame selle stroy nime ja selle, millist komponenti see kasutab:

```
export default {
  title: 'Atoms/Chip',
  component: Chip,
};
```

Joonis 16. Story määramine

Nüüd tuleb StoryBookile [8] teatada, millised story selles failis tegelikult eksisteerivad. Selleks määrame funktsiooni, mis hõlmab kõik argumendid ja edastab komponendile:

```
export const Template = (args) => {
  const [isActive, setIsActive] = useState(false);

  const handleCheck = useCallback((bool) => {
    setIsActive(bool);
  }, []);

  return (
    <Chip
      isActive={isActive}
      onChecked={handleCheck}
      {...args}
    />
  );
};
```

Joonis 17. Funktsioon mis annab StoryBookile teada millised story eksisteerivad

Pärast seda on võimalik selle funktsiooni abil näidata komponente, mille sees on näiteks erinev tekst.

```
export const LabelHello = Template.bind({})
LabelHello.args = {
  label: "Hello",
}

export const LabelDiploma = Template.bind({})
LabelDiploma.args = {
  label: "Diploma",
}
```

Joonis 18. Komponentid erineva tekstiga StoryBook-is

### 3.1.11 Komponentide testimine

Komponentide testimisel leidsid kasutust Jest [17] ja Enzyme [18].

```
yarn add jest
yarn add enzyme enzyme-adapter-react-16
```

Nende pakettide kombinatsioon kiirendab ja lihtsustab komponentide testimist. Siis tuleb määrata, millist adapterit testimiseks kasutada. Selleks loome kataloogis ./src faili setupTests.js ja määrame, millist adapterit kasutada. Kasutasin viimast, 16. adapterit.

```
import { configure } from 'enzyme';
import Adapter from 'enzyme-adapter-react-16';

configure({ adapter: new Adapter() });
```

Joonis 19. Test adapteri määramine

Komponendiga kataloogis loodi varem fail nimega `Chip.test.js`, mille sees saab testida komponendi käitumist. Et kuidas reageerib propsile jne.

```
describe('onChecked', () => {
  it('on click calls func', () => {
    const onChangeSpy = jest.fn();
    const wrapper = shallow(<Chip label="test" isActive={true} onChecked={onChangeSpy} />);
    const event = { target: { checked: true } };
    wrapper.find('button').simulate('click', event);
    expect(onChangeSpy).toHaveBeenCalledWith(true);
  });
});
```

Joonis 20. Test mis kontrollib komponendi onChecked funktsiooni

### 3.1.12 Konfiguratsioon teegi laadimiseks npm-keskkonda

Mõned brauserid ei mõista uusimat ECMAScripti ja React JSX süntaksit, mistõttu tuleb ES6, ES7, ES8 või JSX kood tagasiühilduvas versioonis JavaScripti koodiks teisendada. See teeb pakett Babel [19].

```
yarn add @babel/cli @babel/core @babel/preset-env @babel/preset-react
```

Lisame faili .babelrc ja näitame, milliseid pakette tuleb kasutada.

```
{
  "presets": ["@babel/preset-react", "@babel/preset-env"],
}
```

Joonis 21. Babel parkettide määratlemine

`@babel/preset-env` teisendab kogu koodi, mis kasutab uusimaid JavaScripti funktsioone, brauseritega ühilduvaks koodiks, mis ei toeta uusimaid JavaScripti funktsioone. See plugin võib ka failipaketi väiksemaks muuta.

`@babel/preset-react` teisendab viimase JSX-koodi JavaScripti koodiks.

Webpack [20]**Error! Reference source not found.** on staatiline moodulipakett tänapäevaste JavaScripti rakenduste jaoks. Vaja on vaid luua veebipaketi konfiguratsioonifail.

```
yarn add webpack webpack-cli
```

Tuleb luua faili webpack.config.js koos konfiguratsiooniga. Konfiguratsiooni kõige olulisemad osised on:

- Objekt "entry" - koht, kuhu veebipakett vaatab paketi ehitamise alustamiseks.
- Objekt "output" teatab veebipaketile, kuhu pakette luua ja kuidas neid faile nimetada.
- Objekt "loader" võimaldab veebipaketil töödelda teist tüüpi faile ja teisendada need kehtivateks mooduliteks, mida rakendus saab kasutada ja sõltuvusgraafikule lisada.
- Objekt "module" määrab, kuidas projekti raames töödeldakse eri tüüpi mooduleid.
- Objekt "resolve" muudab moodulite lahendamise viisi.
- Väli "target" näitab, millises keskkonnas on vaja projekt kompileerida.

```
const path = require('path');
const { CleanWebpackPlugin } = require('clean-webpack-plugin');
const nodeExternals = require('webpack-node-externals');
const pkg = require('./package.json');

module.exports = {
    entry: './src/index.js',
    externals: [
        nodeExternals(),
        {
            react: "react",
            'react-dom': "react-dom"
        }
    ],
    output: {
        filename: 'Main.js',
        path: path.resolve(__dirname, 'dist'),
        library: pkg.name,
        libraryTarget: 'commonjs2',
    },
    target: 'web',
    plugins: [new CleanWebpackPlugin()],
    module: {
        rules: [
            {
                test: /\.(js|jsx|ts|tsx)$/,
                exclude: /node_modules/,
                use: ['babel-loader'],
            },
            {
                test: /\.sass$/,
                use: ['style-loader', 'css-loader', 'sass-loader'],
                include: path.resolve(__dirname, './src'),
            },
            {
                test: /\.svg$/,
                use: ['@svgr/webpack', 'url-loader']
            },
        ],
    },
    resolve: {
        extensions: ['.ts', '.tsx', '.js', '.jsx', '.json'],
        alias: {
            react: path.resolve('./node_modules/react')
        }
    },
};
```

Joonis 22. webpack konfiguratsioon

Tuleb lisada elementi package.json ka uue skripti, mis loob kaustas Dist projektikoostu:

```
"scripts": {
    …,
    "build": "webpack --mode production",
    …,
},
```

Joonis 23. Skript mis loob projektikausta

Näiteks selliseid pakette nagu React prop-types ei tohi hõlmata. Neid või muid pakette saab hõlpsasti välistada, kui lisada webpack-node-externals. Niiviisi saab vähendada kasutajate allalaaditava paketi suurust.

```
const nodeExternals = require('webpack-node-externals');
module.exports = {
  ...
  target: 'node',
  externals: [nodeExternals()]
}
```

Joonis 24. Pakette välistamine

Kuid need peavad olema projektis React ja need peavad olema esitatud elemendis package.json.

```
"peerDependencies": {
   "react": "^16.12.0",
   "prop-types": "^15.7.2"
}
```

Joonis 25. Välistatud pakette defineerimine package.json failis

Järgmisena tuleb näidata, et webpack kasutaks kohalikku Reacti, mitte teegis asuvat. Seetõttu kasutame webpacki konfiguratsioonis:

```
resolve: {
        extensions: ['.ts', '.tsx', '.js', '.jsx', '.json'],
        alias: {
            react: path.resolve('./node_modules/react')
        }
    },
```

Joonis 26. Kohalikku Reacti kasutamine

Jääb üle vaid projekt npm-is avaldada. Selleks piisab kahe käsu sisestamisest terminali:

```
npm login
npm publish
```

### 3.1.13 Github Pages

Et anda võimalus komponentidega eelnevalt tutvuda, saab rakenduse avaldamiskohaks GitHub Pages [9]. Esmalt tuleb installida pakett `gh-pages`.

```
yarn add gh-pages
```

Konfiguratsioonifailis package.json määrame lingi GitHubi kasutaja avalehele ja lisame uued skriptid:

```
"homepage": http://msuhhomjatnikov.github.io/atomic-ui

"scripts": {
    ...,
    "predeploy": "yarn run build-storybook",
    "deploy-storybook": "gh-pages -d storybook-static",
    "build-storybook": "build-storybook -s public",
    ...,
  },
```

Joonis 27. Skriptid rakenduse avaldamiseks

Rakenduse avaldamiseks piisab, kui sisestada:

```
yarn run deploy-storybook
```

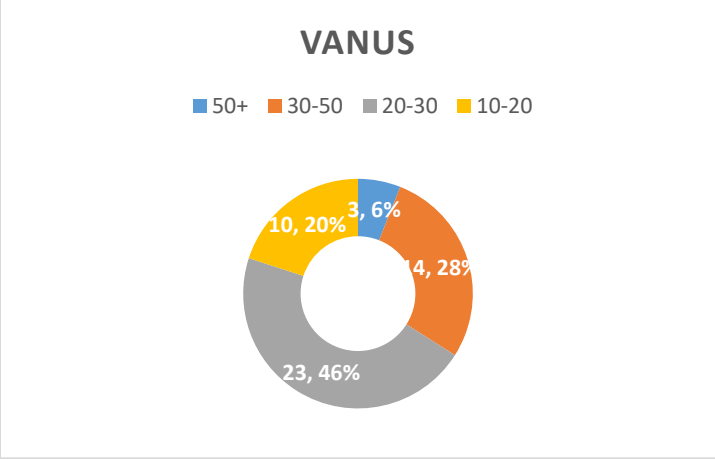
# 4. Analüüs ja järeldused

Tehtud töö analüüsimiseks valiti kolm fookusgruppi: arendajate grupp, disainerite grupp ja kasutajate grupp. Kogu vajaliku info kogumiseks kasutajate ja arendajate kohta kasutati Google Formsi ja disainerite grupi puhul toimus ka üks-ühele vestlus, sest disainerite grupp on teistest väiksem.

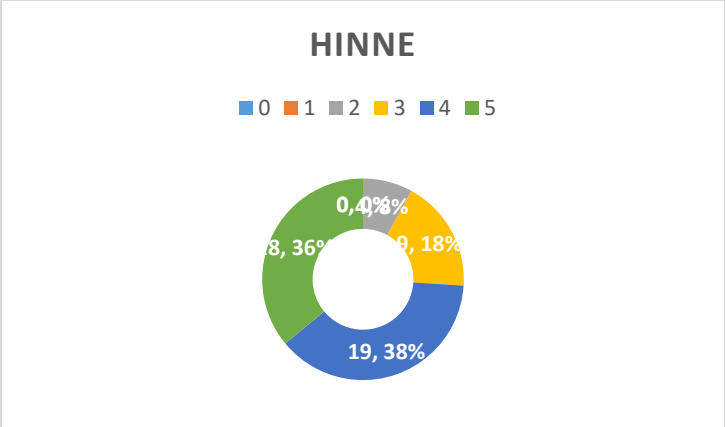
## 4.1 Kasutajate grupi küsitluse analüüs

Kasutajate grupi küsitlemiseks koostati vorm, mis sisaldab 2 kohustuslikku küsimust ja ühte täiendavat valikulist küsimust. Kasutajate grupp koosneb viiekümnest inimesest ja sinna kuuluvad: koolilapsed, üliõpilased, lapsevanemad, õpetajad jne. Mõnega neist toimus protsess üks-ühele vormis, kas kohapeal või videolingi kaudu.

Peamised küsimused olid kasutaja vanus ja kui intuitiivsed need komponendid olid.




Joonis 28. Kasutajate vanuse diagramm




Joonis 29. Kasutajate hinnete diagramm

Täiendava küsimusena andsin kasutajatele võimaluse anda tekstilist tagasisidet. Pärast saadud info analüüsimist olid peamisteks probleemideks komponentide algvärvid ja font.

## 4.2 Disainerite grupi küsitluse analüüs

Selles grupis oli kaks inimest. Üks neiu ja üks noormees. Vestlus nendega toimus üks-ühele vormis, et kirjeldada täpsemalt probleemi, mida anutud töös püütakse lahendada. Peamised kriteeriumid olid: hinnang teegi värvipaletile, hinnang fontidele ja tekstile, hinnang komponentide väliskujundusele ja töökogemusele nendega. Arutasime ka punkte, mis on seotud üldise kasutuskogemuse, esitluse ja disaini täiustamisega.

Kõiki punkte mida vestluse käigus mainiti püüti ka kompinenditeegi loomisel arvesse võtta. Peamised probleemid olid komponentide margins ja paddings, samuti font. Tänu nendele vestlustele sai leitud font, mis rahuldas disainereid ja sobis ka komponentide välimusega. Samuti anti ~~mulle~~ nõu, kuidas mõne kujutist kasutava komponendi kvaliteeti parandada. Üldiselt oli tagasiside sarnane, mis veidi hõlbustas tööd.

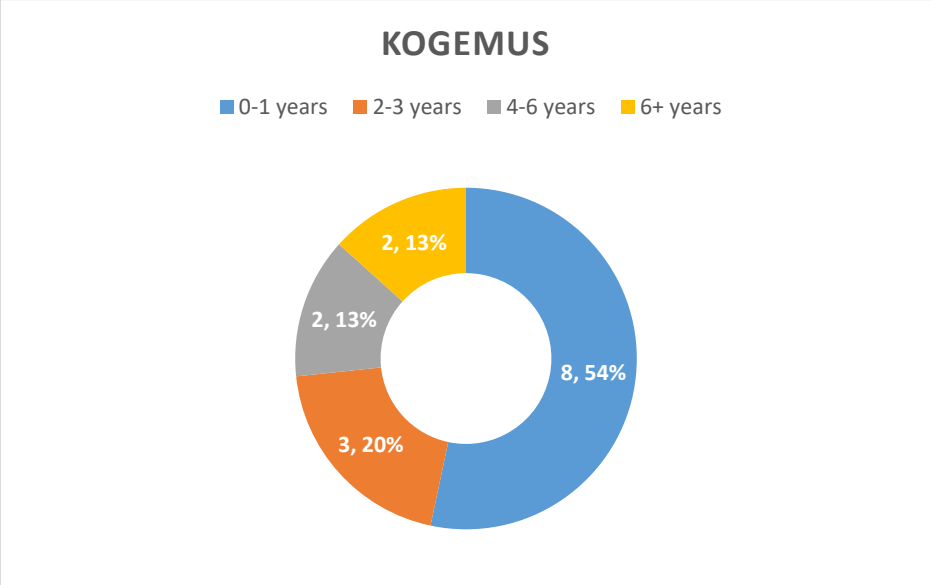
## 4.3 Arendajate grupi küsitluse analüüs

Arendajate grupp koosnes 15 inimesest, kellel oli erinev töökogemus Front-End arendajana. Nende 15 inimese seas oli inimesi, kellega töö autor on kunagi koos töötanud või praegu töötan, tudengeid ja entusiaste, kes õpivad kasutajaliidese programmeerimist.

Peamised küsimused, millele sooviti vastuseid saada, olid:

- Töökogemus Front-End arendajana?
- Kas õnnestus teeki oma projekti installimine?
- Kas õnnestus kasutada tehtud komponente?
- Kui mugav oli komponente oma vajadustest lähtuvalt muuta?
- Kui hea oli kogemus komponenditeegiga?

Lisaks koguti tagasisidet komponentide muudatuste ja komponenditeegi täiustamise kohta.




Joonis 30. Arendajate kogemuse diagramm

Kõik 15 inimest suutsid oma projekti teeki installida, mis näitab, et webpacki konfiguratsioon on õigesti seadistatud ja toimib nõuetekohaselt.

Üks inimene 15-st ei suutnud komponente kasutada. See võib tähendada, et komponenditeegi kasutamise dokumentatsiooni saab ja tuleb täiustada.

Üksteist inimest 15-st suhtusid komponentide kohandamise meetodisse positiivselt. Ülejäänud jätsid tagasiside. Selle tulemusena on vaja komponenditeegi kasutamise dokumentatsiooni lisada täiendavaid punkte.

Samuti saadi teavet selle kohta, kuidas komponenditeeki täiustada. Teised arendajad juhtisid tähelepanu vigadele komponentide töös, eelkõige RadioButtoni ja Tooltipi töös. Nad juhtisid tähelepanu sellele, et mõned komponendid ei toimi mõnes konteineris universaalselt, ning palusid ka muuta rakenduse vaikefonti ja lisada komponenditeeki fondistiile.

## 4.4 Tehtud küsitluste tulemus

Intervjuude abil jõuti mõne punktini, mis vajab lisategevust või täiustamist.

Esmalt tuleb muuta komponentide vaikefonti. Vestluse käigus disaineritega leiti font, mis on üldise hinnangu kohaselt komponentidega sarnase stiiliga ja mida saab kasutada komponentide kasutamise ajal kasutajakogemuse parandamiseks. Samuti muudeti komponentide vaikevärvi.

Pärast arendajate tagasiside uurimist parandati eelnevalt nimetatud komponentide vead. Samuti kirjutati kordumise vältimiseks testid, mis testivad konkreetseid nimetatud vigu.

Projekti dokumentatsiooni täiustamiseks koostati plaan, mis hõlmab mõningaid eespool nimetatud probleeme ja mitmesuguseid võimalikke juhtumeid.

## 4.5 Miks sellised tulemused?

Töö autor leiab, et üks põhjusi, miks seda teeki on lihtne kasutada, on atomaarse disaini metoodika. See lihtsustab oluliselt tööd komponentidega ja vabastab seega arendajatele aega.

Kuna töö tehti disainerite abiga, on kasutajakogemus kõvasti tõusnud, nagu näitab ka kasutajate kasutajakogemuse diagramm.

Kuna iga kasutajaliidese disainer teab, mis on klassid, peaks klasside objekti abil komponentide muutmise meetod kogu protsessi lihtsustama.

## 4.6 Äriline kasu ja kasum

Nagu varem mainitud, lihtsustab ja kiirendab komponenditeek rakenduse arendamist, sest ühe koodijupi muutmiseks tuleb seda muuta vaid ühes kohas. Näiteks idufirmade puhul on ajal tööprotsessis väga oluline roll. Alati tuleb võimalikult kiiresti uusi muudatusi lisada ja siinkohal ongi ühtsest komponenditeegist abi. Alati võib kokku tulla disainerite ja teiste huvitatud osapooltega, et arutada muudatusi ja uusi funktsioone. See muudab disainiga töötamise iteratiivsemalt lihtsamaks. Uusi funktsioone saab hõlmata kiiremini ja uusi asju on lihtsam proovida.
Paljude meeskondadega töötades toimuvad uued uuendused palju kiiremini ja uued töötajad hakkavad kiiremini mõistma oma töö olemust.

## 4.7 Töö edasiarenduse võimalused

Töö käigus selgitati välja mitmeid punkte, mida saab edaspidi lisada. Esimene punkt, mida esile tuua, on see, et saab lisada teise dokumentatsiooni, mis on üksikasjalikum, hõlmates teegi mitmesugused mittestandardsed aspektid. Et praegune dokumentatsioon genereeritakse automaatselt, on kasulikum lugeda kasutaja kirjutatud dokumentatsiooni.

Teiseks saab lisada uusi komponente. Projekti praeguses versioonis kasutatakse ainult neid komponente, mis asuvad atomaarse disaini esimeses etapis, see tähendab aatomeid. Edasise arengu seisukohalt mängib teise ja kolmanda etapi komponentide kätte trumbid ja annab suurema varieeruvuse. Lisaks tasub jälgida olemasolevaid komponente ja parandada vigu, samuti lisada teste, et need vead edaspidi ei korduks.

Kolmandaks saab lisada valmis komponendiplokke, millega saab vähendada arendajate koodi kirjutamist. Näiteks saab lisada komponendi, mis sisaldab kõiki Flexboxi atribuute, ja seeläbi asendada kõik flex-elemendid või konteinerid uue komponendiga, mis kiirendab veelgi CSS-stiilide kirjutamist.

Samuti on võimalik muuta selle projekti kontseptsiooni ja teha sellest avatud lähtekoodiga projekt. Et see areneks veelgi suuremate sammudena ja tagaks suure hulga kasutajate poolse testimise tõttu hea kvaliteedi.

# 5. Kokkuvõte

Antud lõputöö eesmärgiks oli komponentide teeki loomine, mis võimaldab kasutajal antuid komponente hõlpsalt muuta. Antud töö lugemise käigus saadi teada, mis on kasutajaliides, komponent ja komponenditeek. Tegeleti komponenditeegi sees komponentide muutmise probleemiga. Jõuti eesmärgini ja töötasime välja komponenditeegi, millel on vaikestiilid ja mis võimaldab arendajal neid muuta.

Käsitleti komponentide loomise protsessi atomaarse disaini alusel. Uuriti ka teisi komponenditeeke, et tuvastada, mida antud projekti käigus valmivas teha või täiustada.

Töö autor leiab, et tulemuseks on kasutajasõbralik komponenditeek. Komponentide muutmise võimalus klasside abil peab olema mugav, mida näitab ka küsitluse tulemus. Samuti on väga oluline asjaolu, et komponenditeeki saab StoryBooki abil vaadata enne selle oma projekti installimist, sest see aitab komponente ja nende toimimist eelnevalt hinnata.

Kuna töö autor on selles valdkonnas töötanud kolme suurema grupiga, õnnestus lõputöö raames luua hea kasutajakogemusega teek. Töö raames õnnestus saada tagasisidet erinevas vanuses kasutajatelt, mis aitas tuvastada ka tulevikus lahendatavaid probleeme. Arendajate ja disainerite tagasiside aitas tuvastada komponentide ja teegi kui terviku probleeme, parandada vigu ja komponentide visuaalset osa. Kindlasti leidub mõningaid punkte, mida saab teha, kuid ilmselt jäävad need edasiseks arendamiseks.

# Kasutatud kirjanduse loetelu


[1] SendPoints, GUI: Graphical User Interface Design, 2015.

[2] S. Kurg, Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability (3rd Edition), New Riders Publishing, 2013.

[3] B. Frost, Atomic Design, Brad Frost, 2016.

[4] K. Beck, Test Driven Development: By Example 1st Edition, Addison-Wesley Professional, 2002.

[5] „ReactJS Documentation," [Võrgumaterjal]. Available: https://reactjs.org/docs/getting-started.html.

[6] E. P. Alex Banks, Learning React: Functional Web Development with React and Redux, O'Reilly Media, 2017.

[7] D. Crockford, JavaScript: The Good Parts, O'Reilly Media, 2008.

[8] „StoryBook Documentation," [Võrgumaterjal]. Available: https://storybook.js.org/docs/react/get-started/introduction.

[9] „GitHub Pages Documentation," [Võrgumaterjal]. Available: https://docs.github.com/en/pages.

[10] „Material UI Documentation," [Võrgumaterjal]. Available: https://mui.com/material-ui/getting-started/installation/.

[11] „Bootstrap Documentation," [Võrgumaterjal]. Available: https://getbootstrap.com/docs/3.3/getting-started/.

[12] „NPM Documentation," [Võrgumaterjal]. Available: https://docs.npmjs.com/cli/v7/configuring-npm/package-json.

[13] „Git Documentation," [Võrgumaterjal]. Available: https://git-scm.com/docs/git.

[14] „GitHub Documentation," [Võrgumaterjal]. Available: https://docs.github.com/en.

[15] „SASS Documentation," [Võrgumaterjal]. Available: https://sass-lang.com/documentation.

[16] „PropTypes Documentation," [Võrgumaterjal]. Available: https://www.npmjs.com/package/prop-types.

[17] „Jest Documentation," [Võrgumaterjal]. Available: https://jestjs.io/docs/getting-started.

[18] „Enzyme Documentation," [Võrgumaterjal]. Available: https://enzymejs.github.io/enzyme/.

[19] „Babel Documentation," [Võrgumaterjal]. Available: https://babeljs.io/docs/en/.

[20] „WebPack Documentation," [Võrgumaterjal]. Available: https://webpack.js.org/concepts/.

# Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Maksim Suhhomjatnikov

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Kasutajaliidese komponentide teegi Atomic-UI loomine", mille juhendaja on Karl-Erik Karu.
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

18.05.2022

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.