TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Rainer Aas 205850IADB

Pildigalerii veebirakenduse arendus

Bakalaureusetöö

Juhendaja: Meelis Antoi
Magistrikraad

Tallinn 2023

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: Rainer Aas

17.04.2023

Annotatsioon

Antud bakalaureusetöö sihiks on luua veebirakendus piltide ülesse laadimiseks ning kuvamiseks kasutades ära parimaid teeke ja salvestamisvõimalust. Teegid peavad rakenduses võimaldama piltide optimeerimist, töötlust ja ka kuvamist. Kuna rakendus on ka väga heaks põhjaks mõnele teisele, siis on rakendus avatud lähtekoodiga.

Leidmaks parimaid teeke ning salvestamisvõimalusi leiti esmalt kasutatavad tehnoloogiad, sh nii klient- kui ka teenusrakenduse jaoks kasutatavad raamistikud. Seejärel otsiti uuritavatest kategooriatest populaarseimad teegid, mis toetavad eelnevalt valitud tehnoloogiaid. Veel vaadeldi erinevaid salvestamisvõimalusi ning neist parim rakendati ka loodud rakenduse peal. Leiti ka meetod kuidas erinevaid teeke ja salvestamisvõimalusi võrrelda.

Arenduse käigus loodi veebirakendus kahes tükis: esimene tükk, kus on käsitletud klientrakenduse arendus ja teine tükk, kus on kaetud teenusepoolne rakendus. Mõlemas tükis on selgitatud tehtud otsuseid ja protsesse.

Töö arenduse tulemusena sai valmis töötav veebirakendus, mis võimaldab kõiki eelnevalt kirjeldatud protsesse. Rakendusele on loodud ka kokkuvõttev kasutusjuhend, mis aitab veebirakenduse tööle panemisega, seadistamisega ning kasutamisega.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 57 leheküljel, 7 peatükki, 20 joonist, 12 tabelit.

Abstract

Image Gallery Web Application Development

The aim of this thesis is to create a web application for uploading and displaying images using the best libraries and storage options. The libraries must enable image optimization, processing and also displaying. Since the application is also a very good basis for others, it is made to be open-source.

In order to find the best libraries and storage options, the technologies used were first found, including the frameworks used for both the client and service applications. Then the most popular libraries were found in the researched categories that also support the previously chosen technologies. Various storage options were also considered, and the best one was also applied to the application. A method was also found to compare different libraries and storage options.

During the development, the web application was created in two parts: the first part, which covers the development of the client application, and the second part, which covers the service-side application. Both parts explain the decisions and processes that were made.

As a result, a working web application was completed, which enables all the previously described processes. A conclusive user manual has also been created for the application, which helps with the start-up, configuration and use of the web application.

The thesis is in Estonian and contains 57 pages of text, 7 chapters, 20 figures, 12 tables.

Lühendite ja mõistete sõnastik

BSON	*Binary JSON*, binaarne JSON
CLI	*Command-line interface*, käsurea liides
CSS	*Cascading Style Sheets*, keel HTML dokumendi stiili kujundamiseks
dependency injection	Sõltuvuste süstimine, koodi disainimuster, kus üks klass on sõltuv teisest klassist
DOM	*Document Object Model*, Dokumendi objektmudel
HTML	*HyperText Markup Language*, kood veebilehe struktureerimiseks ning sisu loomiseks
HTTP	*Hypertext Transfer Protocol*, hüperteksti edastusprotokoll
IDE	*Integrated Development Environment*, integreeritud arenduskeskkond
JSON	*JavaScript Object Notation* , JavaScriptil põhinev andmevahetusvorming
JSX	JavaScript XML, süntaksilaiend JavaScriptile
kB	Kilobait (1024 baiti)
lightbox	Vaade mis kuvatakse veebilehe sisu kohal, hämardades taust ning tuues esile sisu
ms	Millisekund
NoSQL	*Non-SQL*, mitterelatsiooniline alternatiiv klassikalistele relatsioonilistele andmebaasidele
npm	*Node Package Manager*, tarkvara register ning JavaScript pakettide haldur
ORM	*Object Document Mapping*, objektide dokumentide kaardistamine
REST	*Representational State Transfer*, tarkvaraarhitektuuri laad
SOAP	*Simple Object Access Protocol*, veebiteenusega suhtlemise liidese protokoll
SPA	*Single-page application*, ühelehe rakendus
SQL	*Structured Query Language*, struktuurpäringukeel
URL	*Uniform Resource Locator*, internetiaadress

XML	*Extensible Markup Language*, märgistuskeel info salvestamiseks ja jagamiseks

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Piltide üleslaadimine veebirakendustes on tänapäeval üks paljudest võimalikest kasutaja põhilistest interaktsioonidest – olgu see siis mõnes sotsiaalvõrgustikus teiste inimestega erinevate piltide jagamine või profiilipildi muutmine.

Tänapäeval kasutavad peaaegu kõik, umbes 98% veebirakendustest, kliendipoolse programmeerimiskeelena JavaScripti [1] ning teenusepoolses rakenduses on populaarseim JavaScriptil põhinev Node.js [2]. JavaScriptile on loodud ka hulga erinevaid teeke, mis on mõeldud arendaja töö lihtsustamiseks, sealhulgas ka piltide kuvamiseks ning töötluseks mõeldud teeke. Probleemiks on aga nende teekide suur arvukus. Lisaks on piltide salvestamiseks mitmeid erinevaid võimalusi – kas laadida andmebaasi või pilve ning kui pilve, siis millist pilveteenust kasutada? Kõik see teeb piltide töötluse, optimeerimise ning lõpuks salvestamise ja ka kasutajale sujuva kuvamise üsna komplekkseks probleemiks mille kõige optimaalsema lahenduse leidmine ning rakendamine võib võtta arendajatel väga kaua väärtuslikku aega arendusprotsessist.

Käesoleva lõputöö eesmärgiks on uurida piltide töötluseks ning kuvamiseks mõeldud JavaScripti teeke ning piltide salvestamiseks erinevaid võimalusi ja läbi analüüside ning rakenduse peal läbi viidud katsetuste teel selgitada välja parim lahendus. Lõpptulemuseks peab olema rakendus, kus on kõiki parimaid tehnoloogiaid korraga kasutatud, et luua responsiivne, kiire ja sujuv kogemus.

2 Metoodika

Käesoleval lõputöö käigus vaadeldakse antud lõputöös käsitletud probleemi lahendavaid eksisteerivaid teeke ja salvestamisvõimalusi ning tehakse tehnoloogia valik nii klient- kui ka teenusrakenduse arenduseks. Probleemi analüüsis pakutakse välja ka loodava lahenduse skoop.

Lahenduse analüüsi käigus määratakse nii funktsionaalsed kui ka mittefunktsionaalsed nõuded ning seejärel leitakse metoodika mille põhjal saab võrreldud erinevaid lahendusi ning mille põhjal saab valitud neist ka parimad.

Järgnevalt saab loodud rakendus kus kohas saab käsitletud lahendusi kasutusele võetud ja testitud. Lõpuks, leides parim lahendus eelnevalt valitud metoodika põhjal, tehakse sinna ka kasutajaliides ning lühike ja kokkuvõttev kasutaja juhend, et loodud lahendus oleks ka tulevikus kasutatav.

3 Probleemi analüüs

Käesolevas peatükis tuuakse välja tehnoloogiad ja teegid millega võiks antud lõputöös käsitletavat probleemi lahendada. Selleks, et teha teekide valik tuleks aga esmalt paika panna teatud tehnoloogiad millega rakendust hakatakse looma. Täpsemalt tuleks esmalt määrata nii klient kui ka teenuse poolse rakenduse raamistikud. Seda sellepärast, et pärast teekide valik oleks kitsendatum ja ei tekiks konflikte – üldjuhul teeke luuakse teatud raamistikele. Veel, et pärast ei tekiks kordust, valitakse üksiti ka teised kasutatavad tehnoloogiad.

3.1 Tehnoloogia valik

Lähtuvalt antud lõputöö ühest eesmärgist uurida ja leida parimad JavaScripti teegid piltide töötluseks, mis toimub teenusrakenduse poolel, ja kuvamiseks klientrakenduse poolel, saab kasutatud rakenduse arenduseks nii kliendi- kui ka teenusepoolse programmeerimiskeelena JavaScripti. Ühe programmeerimiskeele kasutamine mõlemal tasandil annab ka omad eelised, näiteks: kiirem ja järjepidevam koodi kirjutamine ning silumine. Sellega aga kogu asi ei piirdu, kuna JavaScriptile on loodud hulga raamistikke, mis teevad arenduse lihtsamaks.

Neli enimlevinumat raamistikku kliendipoolse rakenduse arenduseks [3] on:

- React – Populaarsuselt esimene suure kommuuniga ning väga tugeva põhjaga komponendi põhine, mis lubab luua taaskasutatavaid kasutajaliidese osi, ning deklaratiivne raamistik mille tõttu on see lihtsasti arusaadav ja efektiivne [4]. Raamistik kasutab elementide loomiseks JSX'i ehk JavaScripti süntaksilaiendit, mis võimaldab kirjutada JavaScripti koodi kasutades HTML'i sarnast süntaksit [5].

- Angular – Populaarsuselt teine raamistik kliendipoolse rakenduse arenduseks kasutades HTML'i ja TypeScripti, mis on tugevalt tüübitud JavaScripti koodi kirjutamiseks. Raamistik on komponendi põhine ning kasutab HTML malle, mis

ütleb kuidas tuleks vaateid renderdada. Veel kasutab raamistik direktiive ehk klasse mis lisavad rakenduste elementidele täiendavat käitumist ning raamistikku on sisse ehitatud ka sõltuvuste süstimine, ehk inglise keeles *dependency injection* [6].

- Vue – Populaarsuselt kolmas, progressiivne JavaScripti raamistik kasutajaliideste ehitamiseks. See kasutab standardset HTML'i, CSS'i ning JavaScripti ja pakub deklaratiivset ning komponendipõhist mudelit mis aitab efektiivselt nii kompleksseid kui ka lihtsamaid kasutajaliideseid ehitada [7].

- Svelte – Populaarsuselt neljas raamistik. Svelte pakub teistsugust lähenemist veebirakenduste ehitamiseks: kui tavaliselt tehakse suur osa tööst kasutaja brauseris rakenduse töötamise hetkel, siis Svelte teeb seda kompileerimisel, mis toimub ainult rakenduse ehitamise hetkel, mistõttu luuakse väga hästi optimeeritud JavaScripti kood. Raamistik kasutab väga klassikalist lähenemist HTML'i, CSS'i ja JavaScriptiga, lisades ainult mõned üksikud laiendused HTML'ile ning JavaScriptile mis teeb antud raamistiku õppimise lihtsamaks [8].

Eelnevalt väljatoodud neli raamistikku on kõik avatud lähtekoodiga ning on mõeldud SPA (*Single-page application*), mis tähendab, et laetakse ainult üks veebidokument mille sisu hakatakse uuendama, arenduseks.

Tabelis 1 on võrreldud kliendipoolse rakenduse arenduseks mõeldud JavaScripti raamistikke võttes arvesse autori kogemust, õppimiskeerukust, jõudlust võrreldes teiste väljatoodud raamistikega ning kogukonna suurust.

Tabel 1. Kliendipoolse raamistike võrdlus.

Raamistiku nimi	**Autori kogemus**	**Õppimiskeerukus**	**Jõudlus**	**Kogukond**
React	Suur	Väike [9]	Keskmine [9]	Suur [9]
Angular	Puudub	Suur [9]	Nõrk [9]	Keskmine [9]
Vue	Vähene	Keskmine [9]	Hea [9]	Keskmine [9]
Svelte	Puudub	Väike [9]	Parim [9]	Väike [9]

Võttes arvesse Tabelis 1 välja toodud võrdlust on React raamistik antud lõputöö raames kõige mõistlikum, kuna autoril on antud raamistikuga suur kogemus ning edasise õppimise keerukus madal ja mis teeb arendusprotsessi lihtsamaks. Lisaks on React raamistikul suurim kogukond, mis teeb materjalide leidmise kergemaks. Veel, on mõistlik kasutusele võtta ka Google Material Designil põhinevat Material UI'd, mis on avatud lähtekoodiga Reacti komponentide teek [10], tehes disaini protsess kiiremaks.

Järgnevalt on välja toodud teenusepoolse rakenduse arenduseks valitud raamistik ning sellega kooskõlastatavad tehnoloogiad ning lisaks ka andmebaasi valik.

Teenusepoolse rakenduse arenduseks on suuresti ainult üks variant: Node.js. Seda asjaoludel kuna Node.js on kirjutatud JavaScriptiga, võimaldab kirjutada teenusepoolses rakenduses ka JavaScripti koodi ning lubab kasutada JavaScripti teeke ja npm paketihaldurit kust leidub sadu tuhandeid teeke ning pakette, sealhulgas ka antud lõputööks uurivataid pildi töötluseks teeke. Node.js on avatud lähtekoodiga JavaScripti käituskeskkond mis on väga hästi skaleeritav ning suure jõudlusega [11].

Lisaks saab kasutusele võetud ka Node.js'i peale ehitatud kiire ja minimalistlik Express.js raamistik, mis toob sisse palju kasulikke ja aega kokkuhoidvaid featuure, aitab lihtsasti üles seadistada serverit ja haldab HTTP päringuid ja vastuseid [11].

Andmebaasiks saab kasutatud dokumendipõhist NoSQL (*non-SQL*) andmebaasi. Seda suuresti ühel määraval põhjusel: NoSQL andmebaasides ei ole fikseeritud andmemudeleid, mis on antud lõputöö raames uuritud tehnoloogiate koguse ning ümberkohandamise tõttu kasulik, kuna saab kiiresti ümber muuta ja kasutada uusi andmemudeleid. Lisaks see lubab lihtsamini rakendust skaleerida. Veel on NoSQL puhul pärimine kiirem, kuna erinevalt SQL (*Structured Query Language*) tüüpi andmebaasidest, on NoSQL andmebaasides andmed hoitud nii, et need oleksid päringutele optimeeritud ning ei ole vaja pärida üle mitme tabeli, mis võib minna paljude tabelite tõttu kulukaks protsessiks [12].

Kaks enim kasutatavat NoSQL andmebaasi on [13]:

- MongoDB – MongoDB on NoSQL andmebaas, mis on väga hästi skaleeritav ja paindlik. MongoDB't on võimalik seadistada nii pilves kui ka serveris. Pilves on

sisse ehitatud turvalisus, pidev varukoopiate loomine ja palju muid võimalusi. Lisaks on MongoDB täiesti tasuta ja avatud lähtekoodiga.

- DynamoDB – Amazoni loodud täielikult hallatav NoSQL andmebaas, mis on loodud suure jõudlusega rakenduste käitamiseks mis tahes ulatuses. Antud andmebaas töötab ainult Amazon Web Services pilves, mis pakub sisseehitatud turvalisust, pidevat automaatsed varukoopiate loomist, andmete impordi ja ekspordi tööriistu jpm [14].

Tulenevalt MongoDB suuremast paindlikkusest, suuremast hulgast andmemudelite toetusest, paremast päringu keelest [15] ja lisaks ka autori kogemusest MongoDB'ga saab valitud just see andmebaas antud lõputöö raames kasutamiseks.

Järgmisena on võrreldud kahte populaarseimat arenduskeskkonda ning põhjendatud antud lõputöö ellu viimiseks sobilikku ja väljavalitud keskkonda.

Arenduskeskkonnaks võib valida mitme erineva JavaScripti toetava integreeritud arenduskeskkonna (IDE) – keskkonnad aitamaks arenduse protsessi, lihtsustades koodi kirjutamist, silumist ja automatiseerimist, jpm [16] – ning tekstiredaktori vahel, millest kaks populaarseimat valikut on Microsofti Visual Studio ning JetBrainsi Webstorm.

Mõlemad arenduskeskkonnad on korralikud ning võimelised toetamaks professionaalset arendust [17]. Autori valikus mängib aga kriitilist rolli asjaolu, et Visual Studiol on tasuta versioon [18], kus aga Webstorm on tasuline [19]. Lisaks pakub Microsoft ka Visual Studio Code programmi, mis on nagu väikesemahuline ning mini versioon Visual Studiost kuhu on sisse ehitatud JavaScripti, TypeScripti ning Node.js'i tugi ning on oma lihtsuse ja sisseehitatud JavaScripti ning Node.js'i toe poolest antud lõputöö raameks kõige sobilikum variant.

Projektis tehtud muudatuste jälgimiseks tuleb kasutusele võtta ka mõni versioonihalduskeskkond. Järgmiseks on välja toodud neist populaarseimad ja on selgitatud millist antud lõputöö raames kasutama hakatakse.

Versioonihalduse keskkonna valikul, nagu oli ka arenduskeskkonna valikul, on tähtis, et pakutav teenus oleks tasuta võimalustega, aga ka väga laialt kasutatud ja rohkete materjalidega. Antud kriteeriumite põhjal kolm populaarseimat ning kõige kõrgemalt hinnatumat valikut arendajate seas on GitHub, GitLab ning Bitbucket [20].

- GitHub asutati 2007 ning on tänaseni enim kastutatud versioonihalduskeskkond: tänaseks kasutab antud keskkonda üle 100 miljoni kasutaja [21]. 2022 aastal panustati üle 413 miljoni korra avatud lähtekoodiga projektidesse ning alustati 52 miljonit uut avatud lähtekoodiga projekti [22]. Tasuta valikuga saab platvormil teha lõputult nii privaatseid kui ka avalikke repositooriumeid millel ei ole koostööliste piirangut ning pakub kokku 500 megabaiti salvestusmälu [23].

- Bitbucket on Atlassiani arendatud versioonihalduskeskkond ning selletõttu on Bitbucketit võimalik integreerida ka väga laialdaselt levinud Atlassiani Jira ning Trello keskkondadega [24]. Bitbucketi tasuta versiooniga saab luua lõputult repositooriumeid, nii privaatseid kui ka avalikke, mille kallal saavad korraga tööd teha kuni 5 inimest ning platvormil on võimalik hoida kuni 1 gigabait faile [25].

- GitLab paistab silma oma rohkete ning väljaarenenud featuuride osas ning on teistest väljatoodud keskkondadest noorim: projektiga alustati aastal 2011 [26]. Ka GitLab pakub tasuta teenuse valikut millega saab teha lõputult privaatseid ja avalikke repositooriumeid mida saab jagada kuni 5 inimesega ning salvestada kokku kuni 5 gigabaiti faile [27].

Tulenevalt GitHubi populaarsusest, suurest kommuunist ning tasuta kasutamise võimalustest võetakse see kasutusele ka antud lõputöö koodi haldamiseks. Lisaks saab kasutusele võetud GitHubi pakutud töölauarakendus, mis pakub mugavat kasutajaliidest veel tõhusamaks töövooks.

Kuna planeeritud veebirakendusel on klientrakendus teenusepoolsest rakendusest eraldatud modulaarsuse, taaskasutatavuse, lihtsama halduse jpm põhjuste tõttu, tuleb teha valik ka teatud API arhitektuuri disainmustrite vahel ning täpsemalt siis kahe populaarseima [28]: REST'i ning SOAP'i vahel.

REST ehk *Representational State Transfer* on tänapäeval kordades populaarsem API arhitektuuri disain kui SOAP ehk *Simple Object Access Protocol* [28]. REST'i on lihtsam kirjutada ning õppida ja on ka kiirem. Lisaks kasutab REST ära HTTP'le kohaseid võimalusi ja vastused REST päringutele on väiksemates tekstiformaatides nagu JSON (*JavaScript Object Notation*). SOAP päringutele tulevad vastused alati XML formaadis. SOAP'i eeliseks on sisseehitatud turvalisus ja transaktsioonide vastavus paljudele ettevõtete vajadustele, kuid see teeb ka SOAP'i aeglasemaks ja kobakamaks [29].

Oma lihtsuse, kiiruse ning paindlikkuse poolest sobib REST antud lõputöö rakenduse arenduseks paremini.

3.2 Teekide ülevaade

Teekide valikusse on valitud npm (*Node Package Manager)* paketihaldurist, mis on maailma suurim tarkvararegister mõeldud JavaScripti teekide halduseks [30], allalaadimiste järgi populaarseimad mitte aegunud teegid, tehes märksõnaga otsing ning sorteerides populaarsuse järgi kahanevalt.

Piltide töötluse teekide otsimiseks sai kasutatud otsingu märksõnu „*image*“ ja „*image processing*“ ning piltide kuvamise teekide otsinguks märksõnu „*lightbox*“, „*React lightbox*“ ja „*React gallery*“. Oluline on veel, et kõik teegid oleksid avatud lähtekoodiga, kõik võimalused peavad olema tasuta saadaval ning nendel peab olema ka dokumentatsioon.

3.2.1 Teegid pildi töötluseks

Pildi töötluseks teekide valikul on tähtis, et valitud teek võimaldaks teenusepoolses rakenduses vähemalt pildi suuruse muutmist ning mõnda lihtsamat pildi välimust muutvat funktsiooni, näiteks pildi pööramine või värvi manipuleerimine.

Eelnevalt kirjeldatud nõuete kohaselt välja valitud kolm populaarsemat teeki on järgmised:

- Sharp – Tänaseni kõige populaarseim pilditöötluseks mõeldud Node.js teek millel on npm’i järgi nädalased allalaadimised tõusnud üle 2 miljoni alla laadimise ning on pidevas kasvujoones. Kirjelduse kohaselt kasutatakse seda Node.js teeki kõige rohkem tavavormingus suurte piltide teisendamiseks väiksemateks veebisõbralikeks erineva mõõtmetega JPEG, PNG, WebP, GIF ja AVIF formaatides piltideks. Lisaks pakub teek toiminguid nagu pildi pööramine, pildilt osade välja ekstraheerimist, mitme pildi üheks tegemist, gammakorrektsiooni, jpm. Teegi autor väidab, et on kiirem eksisteeriv lahendus erinevate piltide suuruse muutmiseks. Veel on antud raamistiku eeliseks asjaolu, et jookseb nii Linux, Windows kui ka Mac platvormidel ja ei nõua ühtegi täiendavat alla

laadimist ega sõltuvusi [31], mis tähendab, et seda on teoorias väga lihtne kasutusele võtta.

- Jimp – Populaarsuselt teine piltide töötluseks loodud Node.js teek. Teegil on npm järgi ligi 1.7 miljonit allalaadimist nädalas ning trend liigub kasvavas joones. Antud teegi suurimaks eeliseks on asjaolu, et see on kirjutatud täielikult JavaScriptis, ehk sellel on null sõltuvust ning selle pärast on ka selle teegi paigaldamine lihtne ja universaalne. Teek võimaldab mitmeid erinevaid pildi manipulatsiooni võimalusi nagu näiteks: värvi manipuleerimist, kahe pildi ühinemist, pööramist, suuruse muutmist jm [32].

- Node-canvas – Populaarsuselt kolmas teek milles on lisaks animatsioonidele, mängugraafikatele, andmete visualiseerimisele ning reaalajas videotöötlusele ka pildi töötluseks ettenähtud funktsionaalsus [33]. Teek rakendab veebi Canvas API't ning selle tõttu on antud teegile väga lihtne leida materjale, kuna see on juba laialdaselt kasutusel. Antud teek saavutab ligi miljon allalaadimist nädalas ning on samuti kasvujoonel [34].

3.2.2 Teegid piltide kuvamiseks

Piltide kuvamiseks on valitud kahte tüüpi teeke: teegid üksikute piltide lähemalt vaatamiseks ehk niinimetatud *lightbox* ehk valguskasti stiilis ning teegid kõikide piltide ruudustikus korraga kuvamiseks ehk siis galerii stiilis.

Antud teegid peavad olema ühilduvad kliendi poolse rakenduse arenduseks välja valitud React raamistikuga ning valguskastis kuvamiseks teegid peavad olema tänapäeva rakendusele kohaselt sujuvad ning sisaldama animatsioone.

Kriteeriumitele vastavad väljavalitud teegid piltide valguskastis kuvamiseks ja sirvimiseks:

- Yet Another React Lightbox – Kõige populaarsem modernne nii piltide kui ka videote valguskastis kuvamiseks Reactis ettenähtud teek. Toetab kõikidel platvormidel kõikvõimalikke navigeerimisvõimalusi ja suumimist. Teek kohandub automaatselt piltide suurustega, on kõrge jõudlusega ning võimaldab kasutajaliidese täielikku ümberkohandamist. Lisaks ei ole teegiga kaasas kõiki

lisa funktsioone vaid neid on võimalik eraldi lisada, mille tõttu on teek võimalikult väikesemahuline ja modulaarne [35].

- react-spring-lightbox – Teek mõeldud ainult valguskastis kuvamiseks. Valguskastis navigeerimiseks ning pildi suumimiseks on võimalik kasutada nii arvuti hiire, klaviatuuri kui ka mobiilseadmetele ettenähtud juhtnuppe ja viipamisi. Teek toetab ainult modernseid brausereid ning kasutab kõrge jõudlusega react-springil põhinevaid animatsioone. Teegil ei ole välist CSS'i ja lubab rakendada täielikult kohandatud kasutajaliidest [36].

Kriteeriumitele vastavad väljavalitud teegid kõikide piltide kuvamiseks galeriis:

- react-photoswipe-gallery – Reacti jaoks kohandatud photoswipe teegi, mis on kõige populaarsem teek piltide kuvamiseks, rakendamine. Teegi suurimaks eeliseks on tema kohandatav ning funktsioonirikas olemus – teek sisaldab kahte põhi, ning antud lõputöö raames uuritud funktsionaalsust, ühes: nii piltide valguskastis kui ka galeriis kuvamist. Raamistik toetab ainult modernseid veebibrausereid ning pilte mõõtmetega kuni 3000 * 3000 pikslit [37].

- React Photo Gallery – Teine populaarseim teek React raamistikule piltide galerii koostamiseks. Teek hoiab piltide originaalset kuvasuhet ning pildid kohandavad ennast kas horisontaalselt või vertikaalselt ümber teiste piltide. Lisaks on võimalik antud teegis kasutada enda loodud komponente, et võimaldada näiteks piltide valimist või ümber tõstmist galeriis [38].

Lisaks on väga heaks variandiks piltide galeriis kuvamiseks ka Material UI sisse ehitatud Image List komponent, kuna lõputöö projekti raames võetakse antud komponendi raamistik niikuinii kasutusele. See tähendaks, et ei peaks projekti sisestama peale valguskastis kuvamiseks ettenähtud teegi rohkem sõltuvusi, hoides seeläbi ka projekti mahtu kokku. Komponendiga saab esitada pilte viiel erineval viisil ning on ka kohandatav.

3.3 Salvestamisvõimalused

Salvestamise võimalustest on vaadeldud kolme viisi kuidas pilte hoida ning kus kohast pärast ka lugeda saab: andmebaasi salvestamine, pilve salvestamine ning kohalikku failisüsteemi salvestamine.

3.3.1 Andmebaasi kogu faili salvestamine

Tehnoloogia valikus välja valitud andmebaasi, MongoDB'sse, pildifailide salvestamiseks on kaks varianti: kasutada GridFS spetsifikatsiooni või BSON struktuuri.

- BSON – BSON ehk Binaarne JSON kodeerib tüübi- ja pikkusteavet ning lisab piltide salvestamist võimaldava binaarse andmetüübi, mida JSON ei võimalda [39]. Antud variandiga saab hoida ühes dokumendis kuni 16 megabaiti suuruseid faile. Kui dokument ületab 16 megabaiti peab kasutusele võtma GridFS variandi [40].

- GridFS – Spetsifikatsioon mis on mõeldud failide hoidmiseks, mis ületavad BSON-dokumendi 16 megabaidi suurust piirangut. Antud variandiga ei hoita faili ühes dokumendis, vaid jagatakse fail osadeks ning hoitakse igat osa erinevas dokumendis. Iga osa, väljaarvatud viimane osa, on 255 kilobaidi suurune. Viimane osa on täpselt nii suur kui on vaja. Veel, kasutab GridFS kahte kollektsiooni: üks kus hoitakse faili osasid ning teine kus hoitakse faili metaandmeid. Antud spetsifikatsiooni võib kasutada ka väiksemate failide hoidmiseks ning sellest võib ka kasu olla, näiteks kui tahta laadida faili osaliselt mällu [40].

Selleks, et loodav rakendus oleks võimalikult pandilik ning arendusprotsess kiire ning, et ei peaks iga faili puhul otsustama millist varianti kasutada, mis võib rakenduse kiirust alla tuua, saab teiste salvestamisvõimalustega võrdlusesse toodud GridFS spetsifikatsiooni variant.

3.3.2 Pilvetehnoloogiate ülevaade

Pilvehoidla kasutamine tähendaks seda, et kogu faili informatsioon ja sisu salvestatakse valitud pilvehoidlasse ning andmebaasis oleks viide antud faili asukohale või siis URL'le antud pilve hoidlas.

Antud variandi puhul on tähtis, et tegemist oleks objekti põhise salvestamisega ning seda põhjusel, et see töötab just kõige paremini antud olukorras, kus on üks staatiline fail mida on vaja mitmeid kordi lugeda. Lisaks lubab objektipõhine salvestus hoida struktureerimata andmeid koos erinevate metaandmetega, on kõige paremini skaleeritav ning ka kõige odavam variant [41]. Veel on tähtis, et antud variantidel oleks Node.js'ga ühilduvad klient teegid, mis lubavad kasutada teenuseid programmiliselt.

Pilvetehnoloogiatest kolm suurimat [42] objektisalvestus tehnoloogiat:

- Google Cloud – Google'i loodud pilve teenus mis pakub üle 150. teenuse ning toote mitmest erinevast valdkonnast [43]. Lisaks pakutakse uutele registreerinutele 300$ tasuta krediiti nende teenuste kasutamiseks, mis kehtib 90 päeva. Veel on võimalik kasutada tasuta hoidlat kuni 5 gigabaidi täitumiseni, aga seda ainult kasutades hoidla piirkonnana Ameerika Ühendriike. Kui püsida tasuta võimaluste piires, ei kasutata ära ka krediiti. Tasuta kasutuslimiidil ei ole aegumise tähtaega [44].

- Amazon S3 – Amazoni poolt pakutud objektide salvestusteenus, mis pakub skaleeritavust, andmete kättesaadavust, turvalisust ja jõudlust üle 100 erineva teenuse [45]. Nende pakutud hoidlat on võimalik kasutada tasuta kuni 5 gigabaidi suuruses ja ainult 12 kuud alates selle kasutuselevõtust [46]. Antud teenuse eeliseks on see, et on võimalik kasutada ka teenust nimega Amazon CloudFront, mis lubab edastada sisu turvaliselt madala latentsusaja ja suure edastuskiirusega ning võimaldab tasuta iga kuu kuni 1 terabait andmeid üle kanda [47].

- Azure Blob Storage – Microsofti loodud objektide salvestamise teenus, mis on optimeeritud suure hulga struktureerimata andmete salvestamiseks [48]. Uutele kasutajatele pakutakse 200$ väärtuses krediiti, mida on võimalik ära kasutada 30 päeva jooksul, aga peale seda ei saa tasuta enam antud pilve hoidlat kasutada [49].

Kuna lõputöö autorile mängib kõige suuremat rolli tasuta võimaluste olemasolu ning kõikidel vaadeldud hoidlatel on suuresti sama funktsionaalsus mõne üksiku erinevusega, siis saab antud lõputöö raames katsetatud ja võrreldud teiste meetoditega neist seda, millel on kõige rohkem tasuta võimalusi: Google Cloudi.

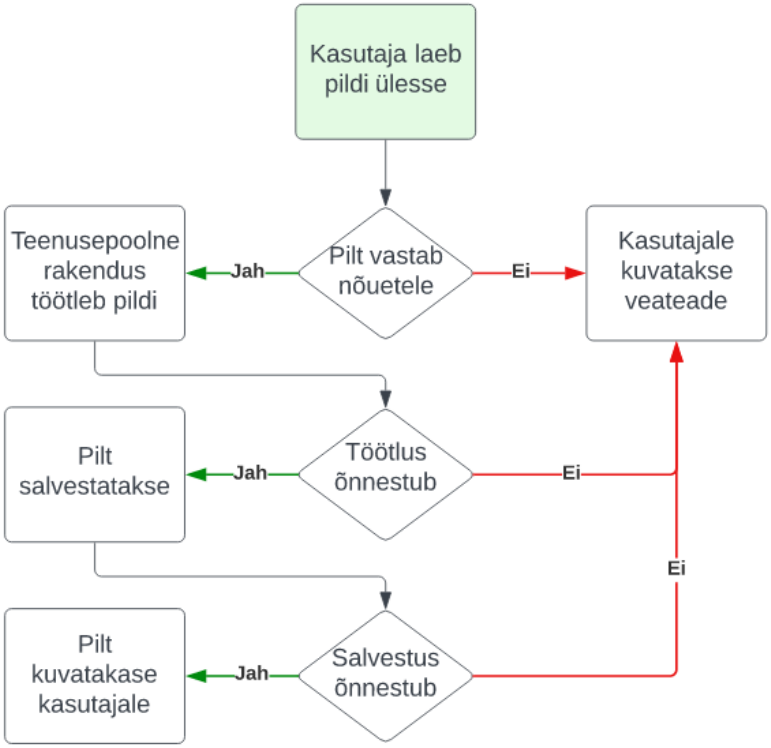
3.3.3 Kohalikku failisüsteemi salvestamine

Kolmas ja kõige otsekohesem salvestamise võimalus on hoida pilte kohalikus failisüsteemis. Selleks tuleb ainult määrata pildi salvestamiseks asukoht ning pärast sealt neid ka lugeda. Lisaks on antud variandi puhul täielik kontroll kõikide failide üle ja kasutada saab kuni oma failihoidla täitumiseni täiesti tasuta.

3.4 Loodav lahendus

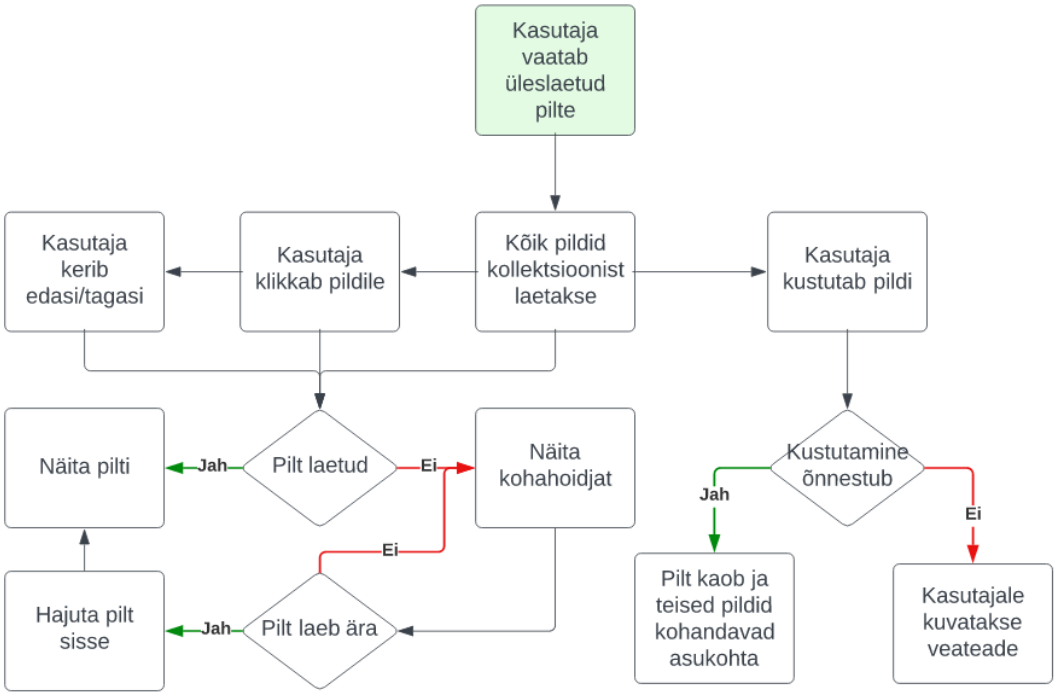
Antud lõputöö probleemi lahenduseks, mis peaks siis tõestama parimaid tehnoloogiaid ning nende koostööd, pakub autor välja uue veebirakenduse. Selleks, et tõestada valitud teeke ning salvestamise optsiooni peaks veebirakendus võimaldama piltidega tehtavaid põhilisi interaktsioone: kasutaja valitud pildi ülesse laadimist, ülesse laetud piltide sirvimist ja ka nende kustutamist.

Kui kasutaja laeb pildi ülesse, siis pilt peab esmalt läbima nõuete kontrolli ning seejärel teenusrakenduse poolel töötav pildi töötluse teek peab optimeerima pildi, et ülesse laadimine oleks kiirem ja võtaks hoidlas vähem ruumi. Loodavas lahenduses on määratud pildile nõuded, et ta oleks kõikide teekide poolt toetatud formaadis ning jääks alla määratud maksimum faili suuruse. Formaadid ning suurused saab täpsemalt paika pandud lõplikus teenusepoolse rakenduse arenduse peatükis. Samuti peab antud teek võimaldama pildi suuruse muutmist kasutaja nõutud mõõtmetele. Kui pilt on ülesse laetud peab saama ka pilti valitud salvestamise kohast kasutajale serveerida. Äsja üleslaetud pilti või pilte võiks kohe kasutajale kuvada, kinnitamaks et protsess õnnestus. Juhul kui üleslaadimine ebaõnnestub, tuleks selle põhjusest viivitamatult kasutajale teada anda. Joonisel 1 on antud protsess visualiseeritud voodiagrammi abil.



Joonis 1. Pildi ülesse laadimise voodiagramm.

Seejärel peab kuvamiseks mõeldud teek või teegid võimaldama kahtmoodi pildi kuvamist: ükshaaval sirvimist nii, et pilt venitatakse võimalikult suureks kasutaja vaates säilitades pildi kuvasuhet ja galeriis kuvamist, et pildid oleksid üksteise kõrval. Pildid peaksid tekkima nähtavale sujuvalt kas siis sisse hajudes või kasutades esialgu mõnda kohahoidjat, milleks võiks näiteks olla pildist hägune versioon. Piltide kustutamiseks on piltide nurgas ka mõni nupp. Kui kustutamine õnnestub, kaob valitud pilt ning ülejäänud pildid kohandavad asukohta. Juhul kui ei õnnestu, siis kuvatakse kasutajale veateade. Joonisel 2 on piltide kuvamise ning kustutamise protsess visualiseeritud voodiagrammi näol.



Joonis 2. Piltide kuvamise ning kustutamise voodiagramm.

Kuigi rakendus ei paku võib-olla nii suurt funktsionaalsust kui mõni rakendus vajab (pildi lõikamine ja värvifiltrite rakendamine veebirakenduses, jne), ning seda asjaolul, et fookus on pigem tehnoloogiate uurimisel, võrdlusel ja tõestamisel, siis võiks antud lahendus siiski olla juba väga heaks põhjaks edasiseks arenduseks, näiteks mõnele planeeritud veebirakendusele kus on antud funktsionaalsus nõutud, kuna põhilised protsessid on rakendatud.

Välja pakutud lahenduse eesmärk ei ole kasumit saada, vaid teha arendajate ning niisama huviliste elu lihtsamaks. Sellest lähtuvalt saab projekt olema avalik ning igaühel on õigus seda ka kasutada.

4 Lahenduse analüüs

Käesolevas peatükis tuuakse välja loodava süsteemi funktsionaalsed kui ka mittefunktsionaalsed nõuded ja selgitatakse välja millise metoodika põhjal erinevaid teeke ning salvestamisvõimalusi võrdlema hakatakse ning kuidas neist parimad valitakse.

4.1 Nõuete määramine

Nõuete määramisel võeti arvesse asjaolu, et tegemist on eelkõige prototüübiga mille põhieesmärgiks on katsetada ning tõestada parimaid tehnoloogiaid, kus on ainult üks kasutaja grupp kes saab kasutada veebirakenduses kõiki funktsioone. Funktsioonid katavad kõiki tavalisi operatsioone mida ootaks ühelt pildigaleriilt.

4.1.1 Funktsionaalsed nõuded

Kasutaja funktsionaalsed nõuded:

- Kasutajana soovin üles laadida oma pilti klikkides.
- Kasutajana soovin üles laadida oma pilti seda lohistades.
- Kasutajana soovin üles laadida enimlevinumaid pildi formaate.
- Kasutajana soovin üles laadida ühe pildi korraga.
- Kasutajana soovin üles laadida mitu pilti korraga.
- Kasutajana soovin näha veateateid.
- Kasutajana soovin näha äsja üleslaetud pilte.
- Kasutajana soovin näha kõiki üleslaetud pilte.
- Kasutajana soovin kustutada üleslaetud pilte.
- Kasutajana soovin sirvida üleslaetud pilte ükshaaval kuvades pilti võimalikult suurelt.

4.1.2 Mittefunktsionaalsed nõuded

Kasutaja mittefunktsionaalsed nõuded:

- Kasutajana soovin kasutada enimlevinumaid veebibrausereid: Chrome, Edge ning Firefox [50].
- Kasutajana soovin, et rakendus oleks lihtsasti kasutatav.
- Kasutajana soovin kasutada rakendust erinevate ekraanisuuruste juures, sealhulgas ka telefoni peal.
- Kasutajana soovin kasutada enda eelistatud veebibrauserit.
- Kasutajana soovin, et pildi üleslaadimine käiks kiiresti. Selleks võetakse üleslaadimise kiirust mõjutavate protsesside võrdlusel üheks võrdlusparameetriks ka kiirus.
- Kasutajana soovin, et pildid tuleksid nähtavale sujuvalt.

4.2 Teekide võrdlemise metoodika valik

Väljavalitud teeke saab võrreldud tabelites, kus iga teegi juures on mingi arv parameetreid. Iga teegi saavutatud tulemused saab tabelisse kirja pandud ning antud tulemuste põhjal võrdlemiseks saab eeskujuks võetud prantsuse matemaatiku Jeans-Charles de Borda 1770. aastal loodud meetodit, mida nimetatakse ka Borda count meetodiks [51]. See tähendab, et tulemusi hakatakse võrdlema kategooriate kaupa teiste samas kategoorias olevate teekide vastu ning iga teek saab mingi arv punkte olenevalt saavutatud parameetri tulemusest. Maksimum punktide arvuks on antud kategoorias olevate teekide arv ning miinimumiks üks. Igas hinnatavas parameetris saab iga teek erineva skoori. Lõpptulemusena parim on teek mis sai kõige rohkem punkte.

Näide: piltide töötluse jaoks mõeldud teekide kategoorias on teegil x parim kiirus, teegil y natuke halvem ja teegil z halvim – siin saab teek x kolm punkti, teek y kaks punkti ning teek z ühe punkti ning võitjaks oleks teek x, kuna saavutas suurima punktisumma.

Juhul kui lõpptulemuses tekib viigi seis, siis saab kasutusele võetud lisa parameeter, kus saab punkte antud veebidomeenil http://www.npmcompare.com asuva tööriista antud

tulemuste järgi. Antud tööriist võrdleb teeke mitme erineva parameetriga ning arvutab nende põhjal lõpptulemuse.

Kõikidel teekidel ühisteks võrdlusparameetriteks on:

- Teegi populaarsus – Teegi populaarsuse all on mõeldud iganädalaste allalaadimiste arvu, mida on näha npm paketihalduri lehel iga teegi juures. Mida rohkem allalaadimisi, seda parem.

- Teegi suurus – Siinkohal on mõeldud teegi lahti pakitud suurust kilobaitides, mida on näha samuti npm paketihalduris teekide juures. Väiksem arv on antud kategoorias parem, sest väiksema suurusega rakendus käivitub ning töötab kiiremini ja võtab vähem ressursse [52].

- Koodi kogus – Pildi töötluseks mõeldud teekide jaoks tähendab see vajatud operatsioonide (funktsioonide) arvu koodis, et muuta ühe pildi suurust. Piltide valguskastis kuvamiseks mõeldud teekide jaoks loetakse vajatud minimaalse koodi ridade kogust, et valguskasti funktsionaalsus tööle saaks minimaalsel tasemel: pildi avamine ning sirvimine. Ning piltide galeriis kuvamise jaoks minimaalne koodi ridade kogus, et kõik (vähemalt 2) pildid tekiksid nähtavale. Antud parameetri puhul on väiksem arv parem.

Pildi töötluseks mõeldud teekide võrdlusesse saab lisaks võetud parameetriks kiirus. Antud teekide kiirust saab hinnatud pildi suuruse muutmise aja põhjal. Aja võtmist saab alustatud vahetult enne teegi välja kutsumist kasutades ära JavaScripti console.time() meetodit, mis paneb taimeri käima ning lõpetatud peale teegi töö lõppemist meetodiga console.endTime(), mis paneb käima pandud taimeri kinni. Aeg mõõdetakse millisekundites ning madalam aeg on parem. Aega mõõdetakse kolmel korral ühe megabaidi suuruse JPG failiga ning saadud tulemustest võetakse keskmine tulemus.

Veel, nii pildi valguskastis kuvamiseks mõeldud teekide kui ka piltide galeriis kuvamiseks mõeldud teekide võrdluses saab arvesse võetud antud lõputöös loodavale projektile kasulikke teekide toetatud funktsionaalsused, näiteks: automaatne kasutaja seadmega kohandumine ning isikupärastamine.

4.3 Salvestamisvõimaluste võrdlemise metoodika valik

Erinevate salvestamisvõimaluste võrdlemiseks saab kasutatud samasugust metoodikat nagu teekide puhul eelnevalt kirjeldatud. Erinevuseks siin on aga parameetrid ning viigi puhul otsustav meetod.

Parameetriteks salvestamisvõimaluste võrdlemisel saab valitud:

- Salvestamise kiirus – Aega saab mõõdetud salvestamise päringu saatmise algusest kuni eduka vaste saamiseni. Edukas vaste on siinkohal HTTP koodiga vahemikus 200 kuni 299. Aega mõõdetakse millisekundites ja mida madalam aeg, seda parem. Aega jälgitakse Chrome'i brauserisse sisse ehitatud arendajate tööriistade abil, jälgides sellest võrgu sektsiooni. Aja mõõtmisel proovitakse nii üksiku pildi salvestamist kui ka mitme, näiteks kolme, pildi salvestamist korraga. Kõik korrad kasutatakse ühte ja sama ühe megabaidi suurust JPG faili. Aega mõõdetakse kolmel erineval korral nii üksiku kui ka kolme faili korraga ülesse laadimisel ning tulemusena võetakse nendest keskmine arv.

- Lugemise kiirus – Täpselt sama nagu eelnevalt kirjeldatud salvestamise kiirus ainult erinevusega, et ei saadeta salvestamise jaoks päring, vaid pärimise jaoks päring. Lugemiseks päringuid katsetatakse Postman rakenduse abil ning eduka vaste saamiseks kulunud aeg loetakse samuti sealt. Katsetatakse kõikide piltide lugemist korraga, kuna on antud loodava rakenduse puhul ainukeseks piltide lugemisviisiks. Meetodid kus on andmebaas ka seotud, tehakse enne lugemist andmebaas tühjaks ning laetakse neli pilti ülesse. Seejärel loetakse antud nelja pilti kolmel erineval korral ning tulemuseks võetakse kolme korra keskmine aeg. Lugemiseks ei ole vaja veel pilti näidata, vaid lugeda ainult pildi informatsioon.

- Koodi kogus – Kui palju operatsioone koodis on vaja teostada salvestamiseks ning lugemiseks. Need liidetakse kokku ühe parameetri alla. Väiksem arv on siinkohal parem.

Punktitabeli abil välja selgitatud kahe parima variandi puhul saab veel vaadeldud mõnda asjaolu loodavast punktitabelist eraldi. Esiteks, mis võimalused on varukoopiate hoidmiseks ja loomiseks. Kolmandaks turvalisus ning neljandaks ka skaleerimine.

5 Lahenduse läbiviimine

Lahenduse koostamiseks tehakse esmalt rakenduse põhi, mis koosneb kahest omavahel suhtlevast tükist: teenusepoolne ja kliendipoolne rakendus. Teenusepoolne rakendus vastutab piltide töötluse, salvestamise ja lugemise eest ja kliendipoolne rakendus saadab teenusepoolsesse rakendusse päringuid piltide ülesse laadimiseks ja pärib sealt ka pilte, et neid kuvada.

Iga tehnoloogia arendus ning testimine teostatakse eraldiseisvates harudes, kasutades ära versioonihalduse võimalusi. See võimaldab igat erinevat tükki eraldiseisvalt katsetada ning hiljem nende juurde vajadusel tagasi naasta.

5.1 Rakenduse põhja arendus

Käesolevas peatükis vaadeldakse rakenduse põhja arendust, seal tehtud valikuid, kasutatud tehnoloogiaid ning muid tehnoloogilisi aspekte.

5.1.1 Teenusepoolse rakenduse põhja arendus

Antud lõputöö raames on teenusepoolseks rakenduseks valitud Node.js, mis on JavaScripti *runtime*, ehk see võimaldab jooksutada teenuse pool serverit kasutades JavaScripti. Koos sellega on kasutatud eelnevas peatükis välja valitud MongoDB andmebaasi, mis omakorda kasutab veel Mongoose ODM (*Object Document Mapping*) raamistikku, mis teeb andmebaasi objektide halduse lihtsamaks.

Rakenduse loomise esimeseks sammuks on kasutada käsklust „npm init“, mis loob faili nimega package.json, mis on suuresti Node.js’i tuumaks. See on JSON tüüpi fail, mis elab projekti juurkaustas ja see hoiab endas tähtsat informatsiooni projekti kohta, näiteks: projekti sõltuvused, versioon, nimi, jpm. Seda informatsiooni kasutab npm ja selle CLI (*command line interface* ehk käsurea liides), et aru saada kuidas peaks projekt töötama ja lubab jooksutada antud projekti, skripte, paigaldada veel sõltuvusi jne. Veel, npm uuendab ja haldab seda faili automaatselt kui peaks lisanduma, eemalduma või uuenema mõni sõltuvus [53].

Edasi on mõistlik installeerida mõned abistavad teegid kasutades npm’i ning need seadistada:

- Express – Node.js'i peale ehitatud veebiraamistik, mis sisaldab palju kasulikke ja aega kokkuhoidvaid featuure, aitab lihtsasti üles seadistada serverit ja aitab hallata HTTP päringuid ja vastuseid [11].

- Nodemon – Utiil mis jälgib serveri töö ajal projektis toimuvaid muudatusi ning automaatselt taaskäivitab selle, et uued muudatused sisse viia. Loodud kiiremaks arendusprotsessiks ja on ka selle pärast siin kasutusele võetud [54]. Et projekt ka käivituks antud utiili abiga on loodud uus skript package.json faili, mis käivitab projekti kasutades just antud utiili. Joonisel 4 on näha package.json sisu koos loodud skriptidega.

- Eslint – Tööriist ECMAScripti (JavaScripti standard, mille eesmärk on tagada veebilehtede koostalitlus erinevates brauserites [55]) ning JavaScripti koodis leiduvate mustrite tuvastamiseks ja aruandluseks, eesmärgiga muuta kood järjepidavamaks ja vältimaks vigu. Eslint lubab lisada üksikuid reegleid ja ka kogukonna loodud pistikprogramme, konfiguratsioone ja muud [56]. Antud tööriist on installeeritud kasutades käsklust „npm init @eslint/config", ehk see aitab lihtsasti initsialiseerida antud tööriista, pakkudes välja mõningad populaarsemad konfiguratsioonid ja pistikprogrammid ning luues nendega .eslintrc.json fail, mis on konfiguratsiooni fail antud tööriistale. Antud projektis kasutusele võtmiseks on valitud väljapakutud standardsed valikud, milleks on: Airbnb loodud stiili konfiguratsioon ja üks pistikprogramm, mis pakub tuge kasutamaks ECMAScript versioon 6-ga kaasnenud „import" ja „export" märksõnu.

- Babel – Babel on JavaScripti kompilaator, mis lubab kasutada kõige uuemaid JavaScripti featuure, näiteks nagu märksõnu „import" ja „export" [57]. Selle lisamiseks on projekti lisatud kolm Babel'i osa: @babel/core, ehk Babel'i tuum [58], @babel/preset-env, eelseadistus mis teeb arenduse lihtsamaks ja koodi suuruse väiksemaks [59] ja viimaseks @babel/cli, millega on võimalik kompileerida faile otse käsurealt [60]. Seejärel on loodud fail nimega .babelrc, mis on Babel'i konfiguratsiooni fail ja sinna on lisatud standard põhi [61]. Lisaks on lisatud üks pistikprogramm, @babel/plugin-transform-runtime, mis võimaldab Babel'i sisestatud abikoodi taaskasutada, et säästa koodi suurust ja lisaks sellele vajalikku @babel/runtime pistikprogrammi [62]. Veel, kuna projektis on

kasutusel Eslint, siis selleks, et Eslint saaks Babel'i tekitatud koodist aru, on vaja projekti lisada @babel/Eslint-parser [63]. Viimaseks on lisatud veel @babel/node, mis on käsurea liides mis töötab täpselt samamoodi nagu Node.js'i käsurea liides, aga kompileerib kasutades Babel'it [64]. Et projekt kompileeriks kasutades Babel'it on muudetud ka projekti käivitamiseks mõeldud skriptid package.json failis. Muudetud skripte on näha joonisel 4, kus on näha ka ülejäänud package.json sisu.

- Multer – Vahevara mis haldab multipart/form-data kodeeringut, mida kasutatakse failide ülesse laadimiseks [65] ning on ka abiks antud lõputöö raames piltide ülesse laadimiseks.

Järgmisena on paika pandud projekti struktuur. Esimese sammuna on loodud kaust sisaldamaks põhilisi JavaScripti faile kasutades parimaid tavasid.

Kõik loodavad failid mis sisaldavad koodi mida on enne kasutamist vaja manipuleerida, sh ka Eslint 6 lisatud import ja export märksõnad, saavad olema „src" nimelises kataloogis [66]. Sinna kataloogi on loodud ka serveri käivitamiseks kõik vajalik kood ning jällegi jälgides parimaid tavasid, on eraldatud serverit käivitav kood kaheks eraldi JavaScript failiks: app.js, mis sisaldab API deklaratsiooni ja index.js, kuhu kuulub võrgu konfiguratsioon, näiteks kus pordi peal server tööle hakkab [67]. Veel on defineeritud fail mis käsitleb kõiki HTTP päringuid.

Selleks on loodud ruuteri fail, mis sisaldab endas Express'ga kaasnevat ruuterit, mis haldab milliste päringute peale mida vastata ja mis päringutega üldse tehakse. Ruuteri failis defineeritud HTTP päringud põhinevad eelnevalt nõuete põhjal paika pandud operatsioonidest ning seal defineeritud URL'id põhinevad parimatel REST API tavadel, mis on seotud API versioonihaldamisega ning URL'de nimetamisega. Tulenevalt versiooni halduse tavast saab iga URL'i ette lisatud ettekääne /api/v1, mis näitab, et tegemist on API esimese versiooniga, ning nimetamisega seotud heaks tavaks on kasutada ainult nimisõnu kas ainsuses või mitmuses. Lisaks kuna REST API puhul on tavaks saata ja lugeda andmeid JSON vormingus, siis on lisatud projekti ka body-parser vahevara, mis lubab lugeda saadetud informatsiooni JSON kujul otse päringu sisust. Rakendusele on veel konfigureeritud, et ainult JSON tüüpi päringu sisuga sisu loetakse [68]. Tabelis 2 on

kirjeldatud kõik võimalikud antud projektis loodavad päringud, mis katavad kõik kasutaja nõuded.

Tabel 2. Teenusrakenduse võimaldatud API päringud.

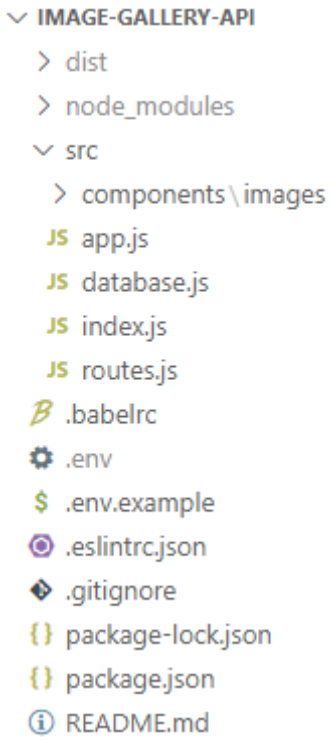
Kirjeldus	**Päringu tüüp**	**Ressurss**
Pildi postitamine	POST	/api/v1/images
Piltide pärimine	GET	/api/v1/images
Pildi kustutamine	DELETE	/api/v1/images/:imageId

Järgnevalt on otsustatud jagada edasised failid komponendi põhiselt ning mitte rolli põhiselt. See tähendab, et kood jagatakse komponentideks, millest igaüks saab oma kausta või spetsiaalse koodibaasi, kus on ainult selle komponendi jaoks vajalik kood mida teiste komponentidega ei jagata. See aitab hoida komponendid ja koodibaasi lihtsana, vältida sõltuvustest tekkivaid segadusi ja aitab ka paremini skaleerida. Sellist arhitektuuri nimetatakse ka mikroteenuste arhitektuuriks [69]. Joonisel 3 on välja toodud lõplik projekti struktuur, mis sisaldab ka juba järgnevaid tehtud muudatusi.

Seejärel on seadistatud antud projektis kasutusele võetav MongoDB andmebaas kasutades MongoDB Atlas't, mis on täielikult hallatav pilveandmebaas, mis haldab kogu kasutuselevõtu komplekssust, haldamist, turvalisust, jne ning pakub ka mugavat kasutajaliidest. MongoDB Atlast kasutades saab MongoDB andmebaasi paigaldada ning seadistada ainult mõne minutiga [70], mis on ka põhjus miks antud variant on valitud antud piiratud ajaga lõputöö arenduseks. Veel on MongoDB Atlasel ka tasuta kasutusvõimalus, mis on mõeldud õppimiseks ja MongoDB avastamiseks, pakkudes kuni 5 gigabaiti salvestusruumi [71].

Selleks, et hoida andmebaasi ühendamiseks mõeldud sõne turvalises kohas, kasutatakse ära keskkonnamuutujaid ehk inglise keeles *environment variables*. Ning selleks, et lisada ka andmebaasi ühendussõne sinna, on kasutusele võetud moodul nimega dotenv, mis lubab kasutusele võtta uue faili nimega .env, kust loetakse väärtused, mis lisatakse projekti käivitamisel keskkonnamuutujatesse [72]. See on hea viis hoidmaks tundlikku informatsiooni koodist väljaspool. Lisaks saab antud fail lisatud .gitignore faili, mis vastutab selle eest, et antud faili ei tehta versioonihaldustarkvarale nähtavaks ning iga kasutaja peaks looma endale isikliku .env faili. Selleks, et ka tulevased kasutajad saaksid

seda võimalust ära kasutada, on lisatud projekti ka näide .env failist, kus on näha mis muutujaid on vaja, et projekt edukalt käivitada. Sellega kaasneb aga üks probleem: kuna projektis on kasutatud ECMAScript 6-ga kaasnevaid „import“ ja „export“ sõnu, siis laaditakse kõigepealt moodulite imporditud moodulid ehk projekt läbitakse sügavuti (*depth-first traversal*). See tähendab, et kui laadida .env fail sisse kohe projekti esimeses laetud failis, siis selle imporditud failid ei näe uusi .env failis leiduvaid väärtuseid. Selle probleemi lahendamiseks on lisatud Babel'i konfiguratsiooni veel üks pistikprogramm nimega babel-plugin-inline-dotenv, mis vastutab, et .env fail laetakse kohe projekti käivitamisel ning on kõikidele failidele koheselt nähtav.



Joonis 3. Teenusepoolse rakenduse põhja struktuur.

```
{
  "name": "image-gallery-api",
  "version": "1.0.0",
  "description": "",
  "main": "src/index.js",
  "scripts": {
    "build": "babel src --out-dir dist --copy-files",
    "start": "npm run build && node dist/index.js",
    "devStart": "nodemon --exec babel-node src/index.js",
    "lint": "eslint src"
  },
  "author": "Rainer Aas",
  "license": "ISC",
  "dependencies": {
    "@babel/runtime": "^7.20.13",
    "babel-plugin-inline-dotenv": "^1.7.0",
    "body-parser": "^1.20.2",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "mongoose": "^7.0.0",
    "multer": "^1.4.5-lts.1"
  },
  "devDependencies": {
    "@babel/cli": "^7.20.7",
    "@babel/core": "^7.20.12",
    "@babel/eslint-parser": "^7.19.1",
    "@babel/node": "^7.20.7",
    "@babel/plugin-transform-runtime": "^7.19.6",
    "@babel/preset-env": "^7.20.2",
    "eslint": "^8.34.0",
    "eslint-config-airbnb-base": "^15.0.0",
    "eslint-plugin-import": "^2.27.5",
    "nodemon": "^2.0.20"
  }
}
```

Joonis 4. Package.json faili sisu.

Eelnevalt joonisel 4 on näha, et projekti sõltuvused on jagatud kahte kategooriasse: dependencies ja devDependencies. Dependencies on sõltuvused mida on vaja programmi sujuvaks toimimiseks ning testimiseks ja devDependencies on paketid, mida kasutatakse ainult arenduse eesmärkidel ning neid ei käivitata programmi jooksmisel ega testimisel [73].

Edasine teenusepoolse rakenduse arendus toimub pilditöötlusteekide ning salvestamisvõimaluste võrdlustes, kus juba katsetatakse ja võrreldakse erinevaid tehnoloogiaid.

5.1.2 Kliendipoolse rakenduse põhja arendus

Antud lõputöö raames eelmises peatükis väljavalitud kliendipoolse rakenduse arenduseks väljavalitud React raamistikuga alustamiseks on kasutatud käsureal käsklust npx create-react-app, mis loob projekti põhja koos näidisfailidega, millest enamus kustutatakse.

Peale üleliigsete näidisfailide kustutamise on loodud projekti struktuur, kus failid jagatakse nende tüüpide kaupa: komponendid on *components* nimelises kaustas, vaated on *views* nimelises kaustas.

Järgnevaks on installitud Eslint sarnastel põhjustel nagu eelnevalt teenusepoolses rakenduse põhja arenduses põhjendatud põhjusega – et hoida koodi järjepidevana ja hoiduda vigadest. Selleks on kastutatud käsurea käsklust „npm init @eslint/config“ ning valitud sealsed enimlevinumad sätted, mille peale luuakse ka .eslintrc.json fail, mis sisaldab Eslint'i konfiguratsiooni.

Järgmise sammuna luuakse kaks vaadet: üks piltide ülesse laadimiseks ja teine piltide kuvamiseks. Kuigi nõuetes on, et peab saama ülesse laadida pilte ka lohistades, siis see funktsionaalsus lisatakse kui hakatakse arendama lõpplahendust, kuna pole teiste tehnoloogiatega koostööks väljaselgitamiseks oluline funktsionaalsus. Seniks, aga kasutatakse ära HTML'i *input* elementi, mis võimaldab valida üks kuni mitu faili. Iga kord kui sisestatud fail või failid muutuvad, siis nendega tehakse POST päring teenusepoolse rakenduse poole. Kuvamiseks mõeldud vaadet hakatakse arendama alles kuvamisvõimaluste võrdlemise juures.

Järgmiseks sammuks on paigaldada ruuter, mis võimaldab rakenduses navigeerida erinevate URL'de põhjal. Selleks on alla laetud npm'i kaudu react-router-dom pakett. Et see ka tööle hakkaks, on loodud komponentide kausta ruuteri komponent, mis loob vajalikele URL'dele nõutud vaated. Nõutud vaated ning nende URL'id defineeritakse eraldi failis nimega router.js, kus iga vaate juurde kuulub tema URL ja temaga kaasneva vaate nimi, mis ühildub kausta nimega vaadete kaustas. Ruuteri komponent seejärel impordib seda faili ning laeb nõutud URL'de peale nõutud vaated. Vaated laetakse

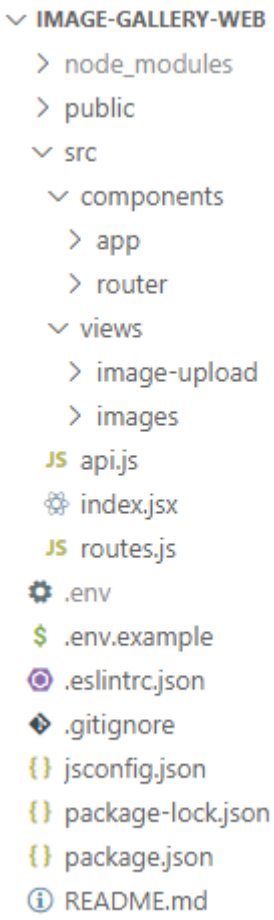
vaadete kaustast dünaamiliselt kasutades Reacti sisse ehitatud niinimetatud laiska laadimist, või siis inglise keeles *lazy loading*, et rakendus oleks võimalikult väikesemahuline ning kiire [74]. Kui proovitakse navigeerida väljapoole defineeritud URL'e, siis suunatakse kasutaja tagasi vaikimisi URL'i peale, milleks saab olema piltide vaade.

Teenusepoolse rakendusega suhtlemiseks võetakse kliendipoolses rakenduses kasutusele ka Axios, mis on populaarne lubadusepõhine (inglise keeles *promise-based*) HTTP-klient Node.js'i ja brauseri jaoks [75], mida on lihtne kasutusele võtta. Lisaks, et hoida teenusepoolse rakenduse URL'i, kasutatakse ära .env faili võimalusi, sarnaselt nagu teenusepoolses rakenduses. Erinevus siin on aga, et ei pea eraldi sõltuvusi kasutusele võtma, et rakendus neid lugeda oskaks. Ainukeseks tingimuseks on, et iga muutuja ees peab olema märksõna REACT_APP, muidu ei loeta väärtuseid välja.

Axios kasutusele võtmiseks on mõistlik luua ka instants et vältida koodi kordamist läbi konfigureerimise, näiteks teenuserakenduse URL'i määramine.

Veel luuakse projekti fail nimega jsconfig.json, kus saab määrata juurkausta asukoha, et muuta rakenduse kood loetavamaks absoluutsete importide abil ning vältimaks suhtelisi importe [76].

Jooniselt 5 on näha loodud kliendipoolse rakenduse põhja struktuuri.



Joonis 5. Kliendipoolse rakenduse põhja struktuur.

5.2 Pilditöötlusteekide võrdlemine

Antud peatükis on võrreldud eelnevalt väljavalitud piltide töötluseks teeke eelnevalt väljavalitud metoodika põhjal. Nende võrdlemiseks arendatakse edasi teenusepoolset rakendust ning pilte veel ei salvestata, kuna pole võrdluses vajalikuks sammuks. Küll aga töödeldud pildid vaadatakse üle, et veenduda tulemuse õigsuses. Selleks tagastatakse teenusepoolsest rakendusest töödeldud pildi puhver, mis on lihtsalt binaarne kuju pildi andmetest, mille abil kuvatakse klientrakenduses pilti.

Esimese pilditöötlusteegina on vaadeldud ning katsetatud Sharp teeki. Teegil on nädalas 2.6 miljonit alla laadimist ning on lahti pakituna 522 kilobaidi suurune [31]. Koodi ridade ning kiiruse mõõtmiseks on antud teek paigaldatud teenusepoolsesse rakendusse kasutades npm'i. Suuruse muutmiseks on antud teegiga mitmeid võimalusi: kas muuta suurust ainult andes ette laius või kõrgus või hoopiski mõlemad. Kui anda ette nii kõrgus kui ka laius, pakub teek erinevaid võimalusi kuidas pilt peaks lõpuks raami mahutuma [77]. Lõpuks töödeldud pilti on ka võimalik mitmeid erinevaid viise edasi anda: kas enda

poolt valitud ning teegi poolt toetatud faili formaadis, milleks võivad olla JPEG, PNG, WebP, AVIF, TIFF, GIF või DZI, või hoopis puhvrina [78].

Pildi suuruse muutmiseks on vaja teha ainult kolm operatsiooni: kõigepealt tuleb teegist instants luua ning talle kas pildi fail või puhver sisendiks anda, seejärel anda suuruse muutmiseks käsklus koos kirjeldatud suurusega ning viimase sammuna täpsustama kuidas peaks töödeldud pildi ammendama. Kuna tegemist on asünkroonse protsessiga, tuleb tulemus ka ära oodata, kasutades märksõna „await“. Joonisel 6 on näha programmikoodi, kus on minimaalne kogus koodi, et pildi suurust muuta Sharp teegiga.

```
const result = await sharp(buffer).resize(150, 150).toBuffer();
```

Joonis 6. Teegi Sharp minimaalne kood muutmaks pildi suurust.

Teekide katsetamiseks kasutatud ühe megabaidise pildi faili suuruse mõõtmetele 150 pikslit * 150 pikslit muutis teek esimene kord kiirusega 12.93 millisekundit, teine kord 11.85 millisekundit ning kolmas kord 11.7 millisekundit. See teeb antud teegi kiiruse keskmiseks tulemuseks umbes 12.2 millisekundit.

Järgmiseks on vaadeldud teeki nimega Jimp. Antud teegil on nädalas 1.7 miljonit allalaadimist ning tema lahti pakitud suurus on 4.79 megabaiti [32]. Et katsetada antud koodi kiirust ning leidmaks minimaalse koodi koguse arvu muutmaks pildi suurust on paigaldatud ka antud teek teenusepoolsesse rakendusse kasutades npm’i.

Ka antud teegi puhul on võimalik pildi suurust muuta mitmel erineval viisil: andes sisse ainult kas laius või kõrgus või mõlemad. Ühe parameetri andmisel saab teiseks suurusparameetriks määrata automaatse suuruse, kus siis teek otsustab ise mis oleks õigeks suuruseks, et hoida pildi kuvasuhet. Veel annab määrata pildi lõplik positsioon pärast suuruse muutmist. Peale pildi töötlemist on ka selle teegiga võimalik kirjutada tulemus uute faili teegi toetatud kujul või siis jällegi, hoopis puhvri kujul. Tulemuse kirjutamisel peab määrama ka faili tüübi. Teek toetab salvestamist PNG, JPEG või BMP formaati [32].

Selleks et antud teegiga pilti töödelda on vajalik teha vähemalt kolm operatsiooni. Esmalt tuleb luua jällegi teegist instants ning sellele sisendiks anda pildi fail või selle puhver ning kuna faili lugemine on asünkroonne, siis selleks, et edasi toimetada, peab antud lugemise

tulemuse esmalt ära ootama. Seejärel saab loetud pilti töödelda ning siinkohal on teostatud siis pildi suuruse muutmine. Viimase ning kolmanda sammuna on vaja tulemus kuidagi salvestada. Joonisel 7 on välja toodud kood mida on vaja, et muuta Jimp abiga pildi suurust ja saada tulemuseks puhver.

```
const result = await Jimp.read(buffer)
       .then((image) => image.resize(150, 150)
       .getBufferAsync(mimetype));
```

Joonis 7. Teegi Jimp minimaalne kood muutmaks pildi suurust.

Kiiruse katsetamiseks ühe megabaidise faili suuruse muutmiseks mõõtmetele 150 pikslit * 150 pikslit kulus teegil esimesel korral 419.2 millisekundit, teisel korral 402.18 millisekundit ning kolmandal ja viimasel korral 416.28 millisekundit. See teeb teegi keskmiseks kiiruseks 412.5 millisekundit.

Viimasena on vaadeldud antud lõputöö raames node-canvas nimelist teeki. Teegil on 1 miljon allalaadimist nädalas ning on lahti pakituna kõigest 367 kilobaiti suur [34]. Katsetamaks pildi suuruse muutmiseks vajaliku koodi kogust ning kiirust saab ka antud teek paigaldatud teenusepoolsesse rakendusse kasutades npm'i.

Pildi suuruse muutmiseks on antud teegiga võimalik suuresti kõike võimalik teha, kuna võrreldes eelnevalt kahe vaadeldud teegiga ei manipuleerita otseselt pilti vaid pilt joonistatakse enda valitud mõõtmetega lõuendile valides pildi alguskordinaadid ning pildi mõõtmed. Antud lõuendiga saab veel teha väga palju erinevaid graafikaga seotud operatsioone ja isegi nullist joonistada pilti. Valminud pilti saab salvestada enda valitud asukohale kettal või hoida puhvrina. Antud teek võimaldab veel valminud kujutiste salvestamist PDF faili [34].

Node-canvas teegiga pildi suuruse muutmiseks on vaja teha viis erinevat operatsiooni. Esmalt on vaja soovitud pilt sisse laadida ning kuna tegemist on asünkroonse meetodiga, peab antud meetodi vastuse ka ära ootama. Seejärel on vaja luua lõuend enda valitud mõõtmetega ning pildi suuruse muutmisel on siinkohal mõistlik valida mõõtmeteks nõutud pildi mõõtmed, et ei tekiks tühjasid ääriseid. Järgnevalt on vaja lõuendi joonistamise kontekst pärida 2D (kahemõõtmelises) formaadis, mis lubab lõuendiga edasised toiminguid sooritada, sh ka pildi joonistamine antud lõuendile. Seejärel ongi tarvis ära kasutada eelnevalt nõutud konteksti, et joonistada pilt uute mõõtmetega

eelnevalt määratud mõõtmetega lõuendile alustades enda määratud asukohast, milleks on siinkohal mõistlik valida vasak ülemine nurk, ehk koordinaadid 0 ja 0. Viimaseks sammuks on joonistatud lõuend salvestada. Joonisel 8 on näha programmikoodi muutmaks pildi suurust kasutades node-canvase teeki kõikide sammudega.

```
const image = await loadImage(buffer);
const canvas = createCanvas(150, 150);
const context = canvas.getContext('2d');

context.drawImage(image, 0, 0, 150, 150);

const result = canvas.toBuffer();
```

Joonis 8. Teegi node-canvas minimaalne kood muutmaks pildi suurust.

Antud teegiga ja eelnevalt kirjeldatud viiesammulise protsessiga kiiruse katsetamiseks kasutatud ühe megabaidise pildi faili suuruse muutmiseks mõõtudele 150 pikslit * 150 pikslit kulus antud teegil esimesel korral 57.5 millisekundit, teisel korral 44.05 millisekundit ning kolmandal korral 44.14 millisekundit. See teeb node-canvase keskmiseks kiiruseks umbes 48.6 millisekundit.

Tabelis 3 on näha väljavalitud pilditöötlusteekide saavutatud tulemused tabelisse kirja panduna eelnevalt väljaselgitatud võrdlemise metoodika põhjal.

Tabel 3. Pilditöötlusteekide võrdlustabel.

Teegi nimetus	**Allalaadimisi nädalas**	**Teegi suurus lahti pakituna**	**Operatsioonide arv muutmaks pildi suurust**	**Keskmine kiirus teegi töö algusest lõpuni**
Sharp	2.6 milj.	522 kB	3	12.2 ms
Jimp	1.7 milj.	4790 kB	3	412.5 ms
Node-canvas	1 milj.	367 kB	5	48.6 ms

Tabelis 4 on näha eelnevate tulemuste põhjal teekide punktitabelit. Kuna Jimp ja Sharp saavutasid sama tulemuse operatsioonide arvu kategoorias, siis sai selles kategoorias antud mõlemale võrdselt punkte.

Tabel 4. Pilditöötlusteekide punktitabel.

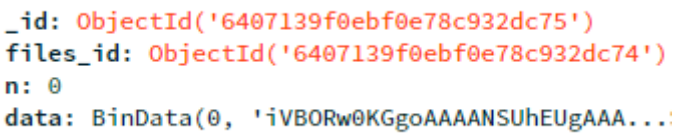
Teegi nimetus	**Allalaadimisi nädalas**	**Teegi suurus lahti pakituna**	**Operatsioonide arv muutmaks pildi suurust**	**Keskmine kiirus teegi töö algusest lõpuni**
Sharp	3	2	3	3
Jimp	2	1	3	1
Node-canvas	1	3	2	2

Eelnevas tabelis on näha, et Sharp sai kokku 11 punkti, Jimp 7 punkti ning node-canvas 8 punkti. See tähendab, et parim teek piltide töötluseks teenusepoolses rakenduses on Sharp ning seda saab kasutatud ka antud lõputöö raames loodava rakenduse lõpptulemuses.

5.3 Salvestamisvõimaluste võrdlemine

Eelnevas peatükis selgitatud metoodika põhjal saab esimese salvestusmeetodina vaadeldud andmebaasi kogu info salvestamist.

Andmebaasis kogu pildi informatsiooni hoidmiseks, nagu eelnevalt kirjeldatud, kasutatakse GridFS spetsifikatsiooni. Kuna teenusepoolses rakenduses on kasutusel Multer vahevara, saab ära kasutatud nende GridFS salvestamismootor, mis vastutab selle eest, et laetud pildid jõuaksid MongoDB andmebaasi vastavalt GridFS spetsifikatsioonile [79]. See tähendab, et pilt hoitakse lahus kahes kollektsioonis: *chunks*, kus hoitakse pildifaili põhiosasid ning *files*, kus on hoiustatud faili metaandmed. Joonisel 9 on näha kollektsiooni *chunks* sisu ning joonisel 10 kollektsiooni *files* sisu peale näidisfaili ülesse laadimist.



Joonis 9. Kollektsiooni *chunks* sisu peale näidisfaili üleslaadimist.

Joonis 10. Kollektsiooni *files* sisu peale näidisfaili üleslaadimist.

Selleks, et kasutada seda varianti, laetakse antud salvestamismootor npm paketihaldurist ka projekti ning seejärel see seadistatakse. Kuna seadistus nõuab andmebaasi ühendust, siis saab ära kasutada projekti põhjas loodud andmebaasi instantsi ning selle ühendust. Veel saab seadistatud salvestatud failide nimetus ning salvestus sihtkoha nimetust, kuigi need pole kohustuslikud sammud. Joonisel 9 on näha seadistus GridFS ning Multer'i abiga piltide ülesse laadimiseks.

```
const storage = new GridFsStorage({
  db: dbInstance.connection,
  file: (req, file) => ({
    filename: `${Date.now()}-${file.originalname}`,
    bucketName: 'files',
  }),
});

const upload = multer({ storage });
```

Joonis 11. Multer'i GridFS kasutuseks seadistus.

Salvestatud andmeid loetakse andmebaasist kasutades ära Multer'i loodud andmebaasi ühendust. Selleks, et ülesse laetud pilte ka pärast kuvada, oleks vaja need veel eraldi alla laadida. Seda aga siin veel ei tehta, kuna pole tulemuste võrdluseks vajalik. Juhul kui antud viis osutub parimaks, tehakse kõik ülejäänud vajalikud sammud, et pilte ka kuvada.

Ühe pildi salvestamiseks kulus esimesel korral 496 millisekundit, teisel korral 317 millisekundit ning kolmandal korral 271 millisekundit. See teeb keskmiseks üksiku pildi ülesse laadimise kiiruseks umbes 361 millisekundit.

Kolme pildi korraga salvestamiseks kulus esimene kord 565 millisekundit, teisel korral 276 millisekundit ning kolmanda korral 294 millisekundit. See teeb keskmiseks kolme pildi korraga ülesse laadimise kiiruseks umbes 378 millisekundit. Kahe salvestamisviisi tulemuste kokku liitmisel teeb see GridFS'i salvestamise kiiruse tulemuseks 739 millisekundit.

Nelja ülesse laetud faili korraga lugemise kiiruseks tuli esimesel korral 20 millisekundit, teisel korral 18 millisekundit ning kolmandal korral 17 millisekundit. See teeb antud meetodi nelja ülesse laetud faili lugemise keskmiseks kiiruseks umbes 18 millisekundit.

Salvestamiseks on antud salvestamismeetodiga vaja teha suuresti ainult 4 operatsiooni. Esiteks tuleb luua ühendus andmebaasiga ning luua GridFS salvestusobjekt andes parameetriks sisse eelnevalt loodud ühendus. Seejärel tuleb loodud salvestusobjekt anda Multer'le sisendiks ning määrata ülesse laadimise URL'i juurde käitajaks, et see ka ülesse laetud faile töötleks ja ikkagi ülesse laeks.

Lugemisega tuleb teha 5 operatsiooni: esmalt tuleb luua ühendus andmebaasiga või kasutada ära salvestusmootori loodud ühendust kutsudes see sisse ehitatud meetodiga esile, seejärel andmebaasist kollektsioon sisse lugeda ja pärida sealt kõik dokumendid ning siis need üle käia ja salvestada need väljasaadetavasse tulemusse ning lõpuks siis tulemus ka välja saata. Tulemuse välja saatmist siinkohal operatsioonide hulka ei loeta, kuna on kõikide meetodite kohustuslikuks tükiks. Kokku teeb see nii lugemiseks kui ka kirjutamiseks 9 operatsiooni.

Järgnevaks vaadeldakse kohalikku failisüsteemi salvestamise võimalust.

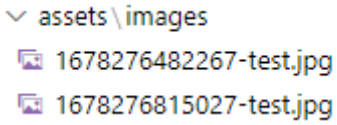
Ülesse laetud pilte hakatakse hoidma mõnes kaustas serveri poolses rakenduses, siinkohal on selleks valitud siis kaust nimega *assets* ja sealne alamkaust *images*. Sinna salvestamiseks kasutatakse ära jällegi projektis kasutusel olevat Multer'i vahevara, aga erinevalt eelnevas kirjeldatud salvestamismeetodis, pole vaja mingit lisa paketti alla laadida. Selleks jällegi konfigureeritakse salvestamiseks sihtkaust ning nagu eelnevalt GridFS faili salvestamiseks, valikuliselt ka salvestatavate failide nimetused. Joonisel 12

on näha Multer'i seadistus, et failid jõuaksid kohalikku failisüsteemi. Joonisel 13 on näha kohalikus failisüsteemis kaustade sisu pärast piltide salvestamist.

```
const storage = multer.diskStorage({
  destination: 'assets/images',
  filename: (req, file, callback) => {
    callback(null, `${Date.now()}-${file.originalname}`);
  },
});

const upload = multer({ storage });
```

Joonis 12. Multer'i seadistus kohalikku failisüsteemi salvestamiseks.



Joonis 13. Kohaliku failisüsteemi sisu pärast salvestamist.

Selleks, et pärast ka näha ülesse laetud pilte tuleks teha salvestuse sihtkaust ligipääsetavaks kasutades Express'i sisseehitatud *express.static* meetodit, mis on mõeldud staatiliste failide teenindamiseks [80]. See võimaldab pilte kliendipoolses rakenduses ükshaaval lugeda ja kuvada. Selleks, et näha mis pildid antud kaustas saadaval on, tuleb lugeda kausta sisu kasutades Node'i sisse ehitatud *file system* moodulit. Siinkohal saab katsetatud minimaalse koodiga, aga kui antud variant peaks osutuma parimaks ning lõpplahenduses kasutatavaks, oleks mõistlik hoida faili metaandmed eraldi andmebaasi dokumentides, et pärast oleks lihtsam neid pärida ning erinevaid tehinguid teostada – siis ei peaks ka *file system* moodulit kasutama.

Ühe pildi salvestamine antud meetodiga osutus väga kiireks: esimesel korral kulus kõigest 16 millisekundit, teisel korral 11 millisekundit ning kolmandal korral 10 millisekundit. See teeb keskmiseks üksiku pildi ülesse laadimise kiiruseks umbes 12.3 millisekundit.

Kolme pildi korraga salvestamiseks kulus esimene kord 25 millisekundit, teisel korral 13 millisekundit ning kolmanda korral 15 millisekundit. See teeb keskmiseks kolme pildi korraga ülesse laadimise kiiruseks umbes 17.6 millisekundit. Kahe salvestamisviisi

tulemuste kokku liitmisel teeb see kohalikku failisüsteemi salvestamise kiiruse tulemuseks 29.9 millisekundit.

Nelja ülesse laetud piltide lugemise kiiruseks antud salvestamise meetodiga tuli esimesel korral 7 millisekundit, teisel korral 4 millisekundit ning kolmandal korral 2 millisekundit. See teeb antud meetodi nelja ülesse laetud faili lugemise keskmiseks kiiruseks umbes 4.3 millisekundit.

Selleks, et kohalikku failisüsteemi salvestada on vaja sooritada 3 operatsiooni. Esiteks tuleb luua Multer'i kohaliku failisüsteemi salvestamisobjekt, määrates seal ära ka salvestuse jaoks kaust. Järgnevalt tuleb loodud salvestusobjekt anda Multer'le sisendiks ja määrata ülesse laadimiseks URL'i juurde see ka käitajaks.

Failide lugemiseks, aga mitte nägemiseks, saab hakkama kõigest kahe operatsiooniga: esiteks, et üldse mingit kasu oleks ülesse laetud failidest, tuleb teha kaust nähtavaks ning ligipääsetavaks Express'i abiga ning teiseks, et näha saadaval faile kasutada Node'i sisse ehitatud *file system* moodulit ning lugeda salvestuskausta sisu. Kokku, nii lugemiseks kui ka salvestamiseks, teeb see kohaliku failisüsteemi meetodi jaoks kõigest 5 operatsiooni.

Järgmisena on vaadeldud pilve salvestamise võimalust.

Pilve salvestamiseks kasutatakse ära Google Cloud Storage'i pakutud võimalusi. Selleks seadistatakse ennem konto valmis ning luuakse salvestamiseks koht ehk *bucket*. Veel on vaja luua objekti ligipääsemiseks vajalik kasutaja grupp, kuskohast on leitavad ka antud salvestamiskoha kasutamiseks vajalikud andmed JSON faili kujul. Antud andmed saavad üle viidud turvaliseks hoidmiseks projekti .env faili. Veel on vaja paigaldada rakendusse Google Storage'i klient teek, mille abil saame Node.js'st otse käsklusi anda Google Cloud Storage'le. Viimaseks põhiliseks seadistuse sammuks on nagu eelnevatel salvestamisvõimaluste vaatlusel vaja seadistada ka projektis kasutusel olev Multer vahevara, aga siin tuleb ainult määrata failide ajutiseks hoidmiseks salvestusobjektina vahemälu. Joonisel 14 on näha eelnevalt kirjeldatud Multer'i seadistus.

```
const storage = multer.memoryStorage();
const upload = multer({ storage });
```

Joonis 14. Multer'i seadistus pilve salvestamiseks.

Google Cloud Storage ühe faili pilve ülesse laadimisel kulus esimesel korral 240 millisekundit ning nii teisel kui ka kolmandal korral 197 millisekundit. See teeb ühe pildifaili pilve ülesse laadimise keskmiseks tulemuseks 211 millisekundit.

Pilve kolme pildi salvestamisel korraga läks esimese päringuga 397 millisekundit, teisel korral 200 millisekundit ning kolmandal korral 224 millisekundit. See teeb kolme faili pilve ülesse laadimise keskmiseks ajaks 273.6 millisekundit ning salvestamise kiiruse kogu tulemuseks 484.6 millisekundit.

Nelja pildi lugemisel pilvest kulus esimene kord 43 millisekundit, teine kord 38 millisekundit ning kolmas kord 37 millisekundit. See teeb keskmiseks lugemise tulemuseks 39.3 millisekundit.

Et Google Cloud Storage pilve andmeid kirjutada on vaja teha 6 operatsiooni. Esiteks tuleb jälle Multer ära seadistada, et faile töödelda, aga seekord kasutatakse ära Multer'i vahemälu salvestusobjekti, mis antakse siis Multer'le ka sisendiks ning jällegi määrata ülesse laadimiseks URL'i juurde see ka käitajaks. Seejärel tuleb luua klient teegi pakutud objekt, kus tuleb sisendiks anda eelnvalt loodud pilve objekti ligipääsemiseks vajalikud andmed. Järgnevalt tuleb luua salvestuskohast instants ning luua sinna uus fail kasutaja valitud faili andmetega.

Pilvest lugemiseks kasutatakse ära eelnevalt salvestamise jaoks loodud instantsi, aga seekord failide lugemiseks. See tähendab, et saab hakkama ühe operatsiooniga. Kokkuvõttes teeb see nii lugemiseks kui ka kirjutamiseks 7 operatsiooni. Nagu eelnevalt kohalikku failisüsteemi salvestamisega, oleks mõistlik ka antud salvestamisvõimalusega hoida pildi andmeid andmebaasis ning sealt pärida.

Tabelis 5 on näha kokkuvõtvalt kõikide vaadeldud salvestamisvõimaluste tulemusi.

Tabel 5. Salvestamisvõimaluste võrdlustabel.

Salvestamismeetod	**Salvestamise kiirus (1 faili + 3 faili tulemus)**	**Lugemise kiirus (4 faili)**	**Operatsioonide arv (salvestamine + lugemine)**
Andmebaasi (GridFS)	739 ms	18 ms	9

Salvestamismeetod	Salvestamise kiirus (1 faili + 3 faili tulemus)	Lugemise kiirus (4 faili)	Operatsioonide arv (salvestamine + lugemine)
Kohalikku failisüsteemi	29.9 ms	4.3 ms	5
Pilve	484.6 ms	39.3 ms	7

Tabelis 6 on näha jagatud punkte eelnevalt välja toodud tulemuste põhjal. Andmebaasi salvestamise meetod sai kokku 4 punkti, pilve salvestamine 5 punkti ning kohalikus failisüsteemis hoidmise meetod saavutas maksimaalse tulemuse 9 punktiga.

Tabel 6. Salvestamisvõimaluste punktitabel.

Salvestamismeetod	Salvestamise kiirus (1 faili + 3 faili tulemus)	Lugemise kiirus (4 faili)	Operatsioonide arv (salvestamine + lugemine)
Andmebaasi (GridFS)	1	2	1
Kohalikku failisüsteemi	3	3	3
Pilve	2	1	2

Nüüd saab vaadeldud kahe parima punktitabeli abil väljaselgitatud salvestamisvõimaluste, kohalikus failisüsteemis hoidmise ning pilves hoidmise, lisa asjaolusid, nagu kirjeldatud salvestamisvõimaluste võrdlemise metoodika juures.

- Varukoopiad – Kohalikus failisüsteemis olevate piltide varukoopiate haldamiseks oleks üheks variandiks hoida varukoopiaid teisel eraldiseisval kettal või siis hoopis pilves. Pilves hoides ongi suuresti juba varukoopiad hallatud – kui midagi kohaliku failisüsteemiga juhtub, ei kao sealt ka andmed. Seetõttu on varukoopiate halduseks mõistlik juba kohe kõike pilves hoida.

- Turvalisus – Turvalisuse tagamiseks kohalikus failisüsteemis oleks vaja teostada pidevat monitooringut ning nõuab teatud ekspertiisi turvalisuse osas kus aga pilves hoides tagatakse turvalisus pilveteenuse poolt pakkudes mitmeid erinevaid võimalusi [81].

- Skaleerimine – Laienemiseks kohalikus failisüsteemis oleks vaja lisada uusi ressursse läbi uue riistvara ja tarkvara ning need nõuaksid lisa tööd, raha, teadmisi jpm, aga pilves laienemiseks piisab üldjuhul mõnest klikist ning läheb kokkuvõttes vähem maksma kui kohaliku failisüsteemi laiendamine [81].

Eelnevalt kirjeldatud punktide juures selgub, et kõige paremaks variandiks, kui arvestada vaadeldud asjaolusid ka peale kiiruse, on siiski pilveteenuse kasutamine ning seda saab kasutatud ka antud lõputöö lõpplahenduse koostamisel.

5.4 Pildi kuvamisvõimaluste võrdlemine

Antud peatükis vaadeldakse ning võrreldakse eelnevalt väljaselgitatud piltide kuvamisvõimalusi, nii galeriis kuvamiseks kui ka lähemalt ehk valguskastis kuvamiseks, eelnevalt kirjeldatud metoodika põhjal. Kuvamiseks ning testimiseks vajalikke pilte veel ei pärita teenusepoolsest rakendusest, vaid kasutatakse ära internetis leiduvaid pilte.

Esimesena on vaadeldud galeriis kuvamiseks mõeldud teeke ning ka Material UI'sse, mis on lõpplahenduses niikuinii kasutusel, sisse ehitatud *Image List* komponenti. Esmalt vaadeldakse teeke ning seejärel seda ka eelnevalt kirjeldatud komponenti.

Kõigepealt vaadeldakse teeki nimega React-photoswipe-gallery. Selle kasutamiseks on vaja paigaldada ka PhotoSwipe, ehk teek millele on antud React'le kohandatud teek peale ehitatud. See tähendab, et kogu suurus lahti pakituna antud võimaluse kasutamiseks on 1.291 megabaiti ehk 1291 kilobaiti. Selle asjaolu tõttu saab ka arvestatud teegi allalaadimiste alla mõlema teegi allalaadimised kokku liidetuna. See tähendab, et antud teegil on kokku umbes 150 000 allalaadimist nädalas. Et teeki lähemalt vaadelda ja katsetada paigaldatakse see kliendipoolsesse rakendusse kasutades npm paketihaldurit.

Selleks, et antud teegiga kuvada kõiki pilte on vaja panna pildi element *Item* nimelise komponendi sisse mis omakorda peab kõik olema *Gallery*, mis loob siis photoswipe konteksti, nimelise komponendi sees. Lühidalt: igal pildil peab olema oma *Item* komponent ning need kõik peavad asetsema *Gallery* komponendi sees. Mõistlik oleks pilte hoida massiivis objektidena, kust saab komponentide loomise hetkel üle itereerida ning luua täpselt nii palju komponente kui vaja. Iga pilt nõuab nelja atribuuti: originaalsuurusele viidet, eelvaatele viidet, laiust ning ka kõrgust. Kokku, et kuvada kõiki

pilte, on vaja kirjutada 9 rida koodi. Loetavuse mõttes trepitud komponendi atribuute ei loeta eraldi ridadena. Joonisel 15 on näha koodi, et kõiki pilte käesoleva teegiga kuvada.

```
<Gallery>
  {images.map((image) => (
    <Item
      original={image.original}
      thumbnail={image.thumbnail}
      width={image.width}
      height={image.height}
    >
      {({ ref, open }) => (
        <img
          ref={ref}
          onClick={open}
          src={image.original}
          alt=""
        />
      )}
    </Item>
  ))}
</Gallery>
```

Joonis 15. Kood piltide kuvamiseks react-photoswipe-gallery teegiga.

Teek on oma loomuselt väga funktsioonide rohke ning sisaldab mitmeid antud lõputöö projekti koostamiseks kasulikke funktsioone. Esiteks ning suurimaks lisafunktsionaalsuseks mis teegil on, on sisseehitatud valguskastis piltide kuvamine. See tähendab, et kasutades antud teeki, poleks vaja lisa teeki valguskastis kuvamiseks. Veel saab lisada piltide slaididele valguskastis pealkirju, lisada kõiki PhotoSwipe toetatud pistikprogramme, kasutada kohandatud kasutajaliidese elemente, kohandada valguskasti sisu ja kohandada kõiki välimuse aspekte.

Järgmisena on vaadeldud React Photo Gallery nimelist teeki. Teek on võrreldes eelnevalt vaadeldud teegiga väga väikesemahuline: tema suurus lahti pakituna on kõigest 56.4 kilobaiti. Allalaadimis on nädalas antud teegil umbes 18 000. Teegi lähemalt katsetamiseks laetakse ta kliendipoolsesse rakendusse kasutades npm paketihaldurit.

Kõikide piltide kuvamiseks antud teegiga on vaja kirjutada ainult üks rida koodi: vaja luua *Gallery* komponent ning anda atribuudiks piltide objektides massiivi viide. Joonisel 16 on välja toodud koodi rida, et kuvada pilte.

```
<Gallery photos={photos} />
```

Joonis 16. Kood piltide kuvamiseks React Photo Gallery teegiga.

Oma väikesemahulisuse tõttu ei toeta antud teek nii palju funktsionaalsust kui eelnevalt vaadeldud teek. Antud lõputöö lahenduse loomiseks sisaldab antud teek kasulikke lisafunktsioone nagu: kohandatud piltide elementide kasutamine, rea või veeru suuna paigutuse valimine, pildid kohandavad ennast üksteise ümber, pildid hoiavad originaalset kuvasuhet.

Tabelis 7 on välja toodud võrdlustabel kahe eelnevalt vaadeldud teegi ning nende võrreldud parameetritega.

Tabel 7. Piltide galeeris kuvamiseks mõeldud teekide võrdlustabel.

Galeriis kuvamis-võimalus	**Allalaadimisi nädalas**	**Teegi suurus**	**Koodi ridu kuvamiseks**	**Kasulikke lisa-funktsioone**
React-photoswipe-gallery	150 000	1291 kB	9	6
React Photo Gallery	18 000	56.4 kB	1	4

Tabelis 8 on näha teekide saavutatud punkte eelnevalt selgitatud võrdluse põhjal. React-photoswipe-gallery saavutas tulemuse 6 punkti ning React Photo Gallery samuti 6 punkti. See tähendab, et teegid saavutasid viigiseisu.

Tabel 8. Piltide galeriis kuvamiseks mõeldud teekide punktitabel.

Galeriis kuvamis-võimalus	**Allalaadimisi nädalas**	**Teegi suurus**	**Koodi ridu kuvamiseks**	**Kasulikke lisa-funktsioone**
React-photoswipe-gallery	2	1	1	2
React Photo Gallery	1	2	2	1

Lõplik teekide valik sooritatakse peale valguskastis kuvamiseks mõeldud teekide vaatlust, võttes seejärel arvesse kogu suurust ning koodi ridu, et oleks olemas nii galeriis kui ka valguskastis kuvamise võimalused. Kuna react-photoswipe-gallery toetab nii valguskastis kui ka galeriis kuvamise võimalusi, siis võrreldakse React Photo Gallery't Material UI *image list* komponendiga, kuna mõlemad on mõeldud ainult galeriis piltide kuvamiseks.

Kuna Material UI on juba lõputöös niikuinii kasutusel, siis alla pole midagi vaja lisaks laadida. Selleks et pilte kuvada, tuleb kirjutada 7 rida koodi. Joonisel 17 on näha koodi, et Material UI *image list* komponendiga pilte kuvada.

```
<ImageList variant="masonry">
  {images.map((image) => (
    <ImageListItem key={image.src}>
      <img
        src={image.src}
        alt={image.title}
        loading="lazy"
       />
    </ImageListItem>
  ))}
</ImageList>
```

Joonis 17. Kood piltide kuvamiseks Material UI *image list* komponendiga.

Material UI *image list* komponent toetab viies erinevas stiilis kuvamist, piltidele tiitlite lisamist ning kõike olemasolevaid komponente saab muuta ning kohandada. Saab isegi täiesti kohandatud komponendi teha, mis on eksisteerivatest täielikult erinev. Veel hoiavad pildid oma kuvasuhet ning neid on võimalik ümber teiste piltide väga lihtsalt kohandada. Antud võimaluse eeliseks on veel dokumentatsiooni kogus – React Photo Gallery'l on see väga minimaalne.

Oma paindlikkuse, lisa allalaadimiste puudumise ning dokumentatsiooni rohkuse tõttu saab ainult piltide galeriis kuvamise võimalusena vaadeldud Material UI *image list* komponenti mõne valguskastis kuvamise teegiga ning seda kombinatsiooni react-photoswipe-gallery vastu võrreldud ja lõpplahendus valitud.

Järgmisena vaadeldakse valguskastis piltide kuvamiseks teeke ning neist esimesena teeki nimega Yet Another React Lightbox. Teegil on umbes 14 000 allalaadimist nädalas ning on lahti pakituna 193 kilobaidi suurune.

Et valguskast tööle hakkaks on vaja kirjutada 3 rida koodi: üks rida, kus hoitakse valguskasti olekut, et kas see on avatud või mitte, teine rida on valguskasti komponent ise ning kolmas rida on hetkel nupp, mille vajutuse peale muudetakse valguskasti olek avatuks. Hiljem tehakse see lihtsalt pildile vajutades lahti. Oleku hoidmiseks on väga

heaks variandiks ära kasutada React'i *useState* funktsiooni. Joonisel 18 on näha koodi antud valguskasti teegi kasutamiseks.

```
const [open, setOpen] = useState(false);

...

<button type="button" onClick={() => setOpen(true)}>Open</button>
<Lightbox open={open} close={() => setOpen(false)} slides={images} />
```

Joonis 18. Kood piltide valguskastis kuvamiseks Yet Another React Lightbox teegiga.

Antud teek sisaldab ka mitmeid kasulikke lisafunktsioone:

- piltide ette laadimine, et mitte kuvada osaliselt laetud pilte,
- automaatselt seadme ekraani suurusele kohanemine,
- piltide suumimine,
- kõikide kasutajaliidese osade isikupärastamise võimalus,
- mitmeid pistikprogramme lisafunktsionaalsuseks,
- klaviatuuri, hiire ning viibete toetus.

Järgmisena on vaadeldud react-spring-lightbox teeki. Teek saavutab umbes 3800 allalaadimist nädalas ja on kõigest 168 kilobaidi suurune.

Antud teegi tööle saamiseks on vaja teha natukene rohkem tööd. Algselt tuleb luua instants valguskasti komponendist, mis nõuab aga teatud seadistusi. Esiteks nõuab ta olekut, et kas valguskast on avatud või mitte. Teiseks nõuab ta funktsioone mis ütleks mida peaks tegema edasi või tagasi liikumisel. Kolmandaks nõuab ta olekut hetkel aktiivse pildi indeksi kohta. See tähendab, et kokku tuleb minimaalselt kirjutada 6 rida koodi. Kaks rida koodi olekute hoidmiseks, kaks rida koodi edasi ning tagasi liikumise funktsioonide jaoks ja üks rida nupule, et valguskasti avada. Siis veel üks rida valguskasti komponendi enda jaoks. Teeki ei ole sisse ehitatud mitte ühtegi kasutajaliidese komponenti – peab ise disainima kõik, aga pole siinkohal testimiseks vajalik, sest navigeerimine ning kuvamine toimivad ikka. Olekute hoidmiseks on jätkuvalt hea valik

kasutada React'i sisse ehitatud *useState* funktsiooni. Joonisel 19 on näha koodi antud teegi minimaalseks toimimiseks.

```
const [isOpen, setOpen] = useState(false);
const [currentImageIndex, setCurrentIndex] = useState(0);

const gotoPrevious = () => (
  currentImageIndex > 0 && setCurrentIndex(currentImageIndex - 1)
);

const gotoNext = () => (
    currentImageIndex + 1 < images.length &&
       setCurrentIndex(currentImageIndex + 1)
);

...

<button type="button" onClick={() => setOpen(true)}>Open</button>
<Lightbox
  isOpen={isOpen}
  onPrev={gotoPrevious}
  onNext={gotoNext}
  images={images}
  currentIndex={currentImageIndex}
/>
```

Joonis 19. Kood piltide valguskastis kuvamiseks react-spring-lightbox teegiga.

Teek sisaldab endas ka mõningaid kasulikke lisafunktsionaalsuseid. Esiteks, sarnaselt eelmisele, saab antud valguskastis navigeerimiseks kasutada klaviatuuri, hiirt ning kõiki viipeid. Veel saab antud teegiga suumida ning pildi peal ringi liikuda. Teek lubab ka kõiki kasutajaliidese elemente ise luua.

Tabelis 9 on välja toodud eelnevalt kirjeldatud kahe teegi võrdlused ning tulemused.

Tabel 9. Piltide valguskastis kuvamiseks mõeldud teekide võrdlustabel.

Valguskastis kuvamis-võimalus	**Allalaadimisi nädalas**	**Teegi suurus**	**Koodi ridu töötamaks**	**Kasulikke lisa-funktsioone**
Yet Another React Lightbox	14 000	193 kB	3	6
React-spring-lightbox	3800	168 kB	6	4

Tabelis 10 on välja toodud eelnevate tulemuste põhjal teekide punktitabel.

Tabel 10. Piltide valguskastis kuvamiseks mõeldud teekide punktitabel.

Valguskastis kuvamis-võimalus	**Allalaadimisi nädalas**	**Teegi suurus**	**Koodi ridu töötamaks**	**Kasulikke lisa-funktsioone**
Yet Another React Lightbox	2	1	2	2
React-spring-lightbox	1	2	1	1

Eelnevast tabelist on näha, et Yet Another React Lightbox saavutas tulemuseks 7 punkti ning react-spring-lightbox 5 punkti. See tähendab, et võrreldud saab Yet Another React Lightbox'i koos Material UI *image list* komponendiga react-photoswipe-gallery vastu.

Yet Another React Lightbox ning Material UI komponendi kasutamine lisaks projekti mahtu kokku ainult 193 kB, kus aga react-photoswipe-gallery lisaks ligi 6.5 korda rohkem mahtu ehk 1291 kB.

Eelnevalt selgitati välja, et react-photoswipe-gallery tööle saamiseks läks 9 rida koodi, Material UI *image list* kasutamiseks 7 rida koodi ja Yet Another React Lightbox kasutuselevõtuks läks 3 rida koodi, millest üks oli nupp mis avas valguskasti ning nagu seal ka kirjeldatud, siis see rida kaoks lõpplahenduses ära kuna valguskast avatakse pildile vajutades. See tähendab, et ka nende kahe kombinatsioonina läheks kokku 9 rida koodi, mis on täpselt sama palju nagu react-photoswipe-gallery puhul.

Suurem erinevus tekib aga vaadeldes eelnevalt väljaselgitatud lisafunktsioonide arvu. Material UI ning Yet Another React Lightbox'i peale kokku on kasulikke lisafunktsioone 11 – Material UI *image gallery* komponent sisaldab viite kasulikku lisafunktsiooni ning Yet Another React Lightbox kuute. Kasutades react-photoswipe-gallery't, oleks kasulikke lisafunktsioone kõigest 6.

Tabelis 11 on välja toodud eelnevalt kirjeldatud võrdlused.

Tabel 11. Lõplik kuvamisvõimaluste tehnoloogiate võrdlustabel.

Kuvamisvõimalus	**Suurus**	**Koodi ridu**	**Kasulikke lisafunktsioone**
React-photoswipe-gallery	1291 kB	9	6
Material UI *image list* + Yet Another React Lightbox	193 kB	9	11

Tabelis 12 on lõpliku kuvamisvõimaluste võrdlustabelist saadud tulemuste põhjal koostatud punktitabel. Kuna mõlemal kuvamisvõimalusel läks sama palju koodi ridu, siis sai mõlemale antud seal kategoorias sama palju punkte.

Tabel 12. Lõplik kuvamisvõimaluste tehnoloogiate punktitabel.

Kuvamisvõimalus	**Suurus**	**Koodi ridu**	**Kasulikke lisafunktsioone**
React-photoswipe-gallery	1	2	1
Material UI *image list* + Yet Another React Lightbox	2	2	2

Eelnevast tabelist (Tabel 12) on näha, et react-photoswipe-gallery saavutas tulemuseks 4 punkti ning Material UI *image list* ja Yet Another React Lightbox'i kombinatsioon saavutas tulemuseks 6 punkti. See tähendab, et lõpplahenduse arenduses saab kasutatud Material UI ning Yet Another React Lightbox'i koostööd.

5.5 Lõpplahenduse arendus

Käesolevas peatükis vaadeldakse lõpliku rakenduse arendust eelnevalt koostatud põhja peale, aga nüüd eelnevalt välja selgitatud parimate tehnoloogiatega ning vajalike arendustega.

Esmalt viiakse lõpuni teenusepoolne rakendus ning seejärel kliendipoolne rakendus, kus tehakse ka kasutajaliides.

5.5.1 Lõpliku teenusepoolse rakenduse arendus

Lõpliku teenusepoolse rakenduse arenduse jaoks on vaja esmalt projekti lisada eelnevalt väljaselgitatud parimate tehnoloogiate teegid: piltide töötluseks Sharp'i teek ning salvestamiseks Google Cloud Storage klient teek.

Järgnevalt on koostatud piltide ülesse laadimiseks, kustutamiseks ja pärimiseks äriloogika. Esimesena luuakse äriloogika piltide ülesse laadimiseks.

Kuna äriloogika on mõistlik hoida kontrollerist, mis vastutab kasutaja ja äriloogika vahelise suhtluse eest, eraldi, siis luuakse kaks faili: *image.controller.js*, mis vastutab kontrolleri rolli eest ja *image.core.js*, mis tegeleb äriloogikaga. Veel luuakse fail *image.js*, kus luuakse andmebaasi objekt piltide hoiustamiseks ning viimaseks fail nimega

index.js, mis tegeleb ainult kontrollerist meetodite eksportimisega, et nende importimiseks kirjeldatud teekond oleks puhtam. Pildi andmebaasi objektil on 6 välja, millest 5 on kohustuslikud: pildi allikas, nimetus, tüüp, laius ning kõrgus. Kuues väli on kohahoidja jaoks.

Piltide ülesse laadimiseks vajaliku äriloogika jaoks luuakse meetod, mis võtab sisendiks kasutaja ülesse laetud faili. Antud meetodisse on seejärel loodud sisendi valideerimine: esiteks vaadatakse, kas üldse on mingi sisend antud, teiseks vaadatakse ülesse laetud faili tüüpi ning kontrollitakse kas see kuulub ka toetatud failide hulka ning kolmandaks vaadatakse faili suurust, et see ei ületaks määratud limiiti. Juhul kui üks neist ei vasta tõele, püstitatakse veateade.

Eelnevalt mainitud toetatud failide tüüpe ning faili suurust hoitakse *config*, ehk konfiguratsiooni nimelises kaustas *image.js* nimelises failis, kuhu on välja toodud kaks muutujat ning mõlemad ka eksporditud. Esimene muutuja on massiiv, mis hoiab toetatud failitüüpide nimetusi. Sinna on lisatud kõik levinumad pildifaili tüübid, mida ka kõik projektis kasutatud teegid toetavad: PNG, JPG (JPEG) ja WebP. Teine muutuja antud konfiguratsioonifailis on maksimaalne lubatud faili suurus, mis on välja toodud baitides. Antud projekti raames on selleks kasutatud 10 megabaiti ning see arvutatud ümber baitideks.

Kui kasutaja sisend läbib valideerimise, siis päritakse pildist laius, kõrgus ja formaat kasutades ära Sharp'i metaandmete pärimise võimalusi ning luuakse, aga veel ei salvestata, nendega ka esialgne pildi andmebaasi objekt. Sinna objekti on veel ära lisatud juba ka pildi originaalne nimetus ja pildifaili tüüp. Järgmiseks on pilt optimeeritud. Selleks muudetakse pildi kvaliteeti, aga nii, et see poleks väga selgelt nähtav. Küll aga muudab antud kvaliteedi muutus mõne pildi puhul pildi suurust lausa poole võrra väiksemaks. Järgmise sammuna laetakse pilt Google Cloud Storage'sse ning selle õnnestumisel lisatakse pildi andmebaasi objekti ka pildile allikas, milleks on Google Cloud Storage'st kätte saadav äsja ülesse laetud pildifaili URL. Viimase sammuna luuakse pildile ka kohahoidja. Selleks otsustati ära kasutata BlurHash nimelist algoritmi, mis on Wolt'i loodud algoritm kompaktsete kohahoidjate kasutamiseks [82]. See võimaldab hoida kohahoidjat ühe lihtsa sõnena pildifaili andmebaasi objektis ning ei nõua Google Cloud Storage's lisa failide hoidmist. Kohahoidja on udune versioon originaalsest pildist. Selleks laetakse projekti ka BlurHash'i teek, mis võimaldab kodeerimist ja

dekodeerimist antud algoritmi jaoks [83]. Ennem algoritmi kasutamist ning kodeerimist on mõistlik pilt võimalikult väikeseks muuta, et protsess käiks kiiremini ning kuna kohahoidja on niikuinii udune, siis ei mõjuta pildi väikene suurus kohahoidja kvaliteeti. Selleks kasutatakse ära Sharp'i ning pilt muudetakse laiusele 32 pikslit. Suuruse otsustab Sharp teek ise, et hoida korrektsena pildi kuvasuhet. Lõpuks on kohahoidja lisatud ka pildi andmebaasi objektile ning nüüd see ka salvestatud andmebaasi ja seejärel meetodi lõpus tagastatakse loodud pildi andmebaasi objekt.

Eelnevalt kirjeldatud Google Cloud Storage klient teegi ning teenuste kasutamiseks on loodud eraldi hoidla nimega *services* ehk teenused ning sinna omakorda hoidla nimega *gcs*, mis on lühend Google Cloud Storage'st. Sinna kausta on seejärel loodud veel kolm faili. Esimeses failis asub instants Google Cloud Storage kliendist. Teises failis asub meetod mis võtab sisendina faili nimetuse ning puhvri ja salvestab selle Google Cloud Storage'sse ja teeb salvestatud objekti ka publikule nähtavaks, et hiljem oleks võimalik salvestatud pilti klientrakenduse poolelt pärida. Kolmandas failis asub meetod failide kustutamiseks Google Cloud Storage'st. See meetod võtab sisendina ainult faili nime ning annab Google Cloud Storage klient teegile käskluse antud nimega faili pilvest kustutada.

Järgmisena on loodud äriloogika piltide lugemiseks. Antud operatsiooniks õnneks ei ole väga palju vaja teha. Piltide lugemiseks loetakse andmebaasist piltide kollektsioonist kõik objektid ning need tagastatakse.

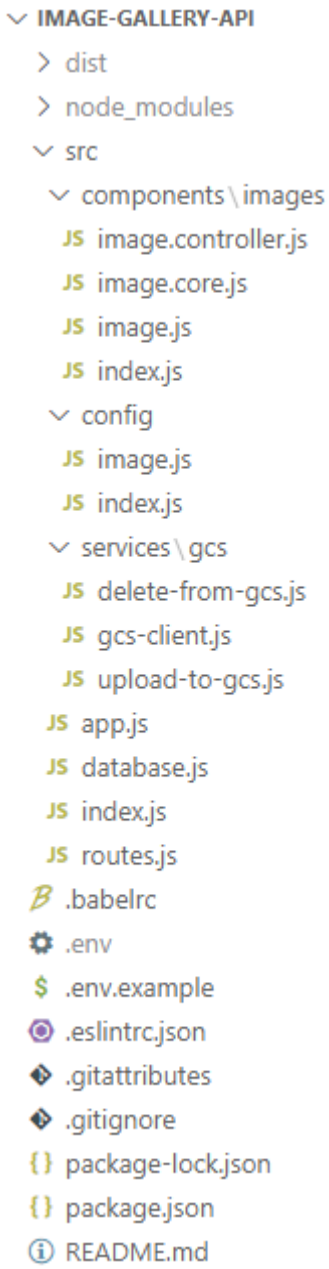
Viimase äriloogikana on loodud piltide kustutamise äriloogika. Selleks luuakse meetod, mis võtab sisendina kustutatava pildi ID ehk identifikatsiooni, mis peab klappima andmebaasis leiduva pildi objekti ID'ga. Esimese sammuna meetodis kontrollitakse kasutaja sisendit: vaadatakse kas üldse on antud mingi sisend või mitte. Juhul kui ei ole, püstitatakse veateade. Järgmiseks otsitakse sisendina antud ID'ga andmebaasist objekti. Seejärel tehakse järgmine kontroll, kas andmebaasist leiti otsitud objekt. Kui ei, siis jällegi püstitatakse veateade. Kui kõik läks hästi ja objekt leiti, siis esmalt kustutatakse see pilvest ning seejärel alles andmebaasist.

Peale nõutud funktsionaalsuste äriloogikate valmimist luuakse kontrollerid. Neid on vaja teha kolm: üks piltide ülesse laadimiseks, teine piltide lugemiseks ning kolmas piltide kustutamiseks. Kõik kontrollerid sisaldavad sarnast loogikat: kutsutakse eelnevalt loodud äriloogika meetodeid koos nõutud sisendiga ning kuna kõik loodud meetodid on

asünkroonsed, siis oodatakse tulemus ära ning seejärel see tagastatakse. Pildi ülesse laadimiseks kutsutakse ülesse laadimiseks loodud äriloogikat ning antakse sisendina kasutaja sisestatud pildi fail, oodatakse tulemus ning tagastatakse ülesse laetud pildi objekt. Piltide lugemiseks kasutatakse ära loodud lugemiseks mõeldud äriloogikat, aga ei anta mingit sisendit, kuna sellel meetodil polnud see nõutud. Kutsutud meetodi tulemus oodatakse ära ning tagastatakse massiiv kõikidest andmebaasis leiduvates pildiobjektidest. Piltide kustutamiseks kasutatakse ära kustutamiseks loodud äriloogika meetod kuhu on antud sisendiks kasutaja edastatud pildifaili ID ning oodatakse tulemus ära. Antud kontroller midagi peale eduka HTTP koodi, 204, mis näitab, et tegevus õnnestus, aga mingit sisu vastu ei tule, ei tagasta.

Veel, kuna äriloogika meetodid võivad visata veateateid ning rakenduse toimimise lõppemine nende tõttu ei ole oodatud, kuna veateated tahaks ka kasutajale edastada, on kõikidel kontrolleritel äriloogika kutsumine ning tulemuse tagastamine ümbrisetud *try catch* plokiga, kus *try* poolel on kutsutud äriloogika meetodeid ning tagastatud tulemus ja *catch* poolel püütakse kinni tekkinud veateated. Kinni püütud veateated edastatakse Express'i sisse antud veateadete halduriga, mis võtab ainult veateate vastu ning tagastab selle JSON kujul klientrakendusele, kutsudes meetodit *next* andes sisse tekkinud veateade.

Joonisel 20 on näha lõplikku teenusepoolse rakenduse struktuuri täies mahus.



Joonis 20. Lõpliku teenusepoolse rakenduse struktuur.

5.5.2 Lõpliku kliendipoolse rakenduse arendus

Kliendipoolse rakenduse arenduseks laetakse alla eelnevalt väljaselgitatud parimad tehnoloogiad kuvamiseks. Veel, kuna piltide kohahoidjateks on kasutatud BlurHash algoritmi, laetakse teek nende dekodeerimiseks ning pildi loomiseks. Esmalt arendatakse lõpuni ülesse laadimise funktsionaalsus ja luuakse ülesse laadimiseks lõplik kasutajaliides ning vaade.

Kuna pilte on vaja saada ka neid lohistades ülesse laadida, luuakse selleks komponent, mis võimaldab piltide lohistamist ning nende vastu võtmist. Loomulikult võimaldab

komponent ka peale klikkimist ning kohaliku failisüsteemi sirvimist. Komponendile on pandud ka piirang, et võtaks vastu ainult sama tüüpi faile, nagu teenusepoolne rakendus võimaldab – sellega saab ära hoida mõttetuid päringu tegemisi ning teenusepoolse rakenduse poole pöördumisi, kontrollides juba kliendipoolses rakenduses sisendi tüüpi. Ülesse saab laadida mitut pilti korraga ning juhul kui mõni neist on vigane ja tekib veateade, ei kao selle tõttu teised korrektsed pildid vaid need laetakse ikkagi ülesse.

Kuna kasutajale peab kuvama ka veateateid, on selleks loodud ka veateateid kuvav komponent: veateated tekivad ülesse laadimise kasti alla ning on selgelt nähtavad ning muust sisust eristatavad.

Kui kasutaja pildi või pildid ülesse laeb, peab kasutaja neid ka koheselt nägema. Selleks on lisatud eelvaadete komponent, mis asetseb kohe ülesse laadimise kasti all. Kasutaja edukalt ülesse laetud pildid ilmuvad sinna, kui teenusepoolne rakendus enda töö lõpetab ja piltide informatsiooni tagastab. Ülesse laadimise oleku nägemiseks on listaud ka edenemisriba. Kui pilt või pildid alles laevad ülesse, ei saa nii kaua uusi pilte lisada kuniks eelmine protsess on lõppenud.

Järgmiseks on arendatud piltide galerii vaade.

Esimese sammuna piltide galerii arenduse juures on teha pildi komponent, mis võetakse kasutusele ka ülesse laadimise juures kui pilti kasutajale esimest korda kuvatakse, et hoida ära koodi kordamist. Pildi komponenti on sisse ehitatud ka kohahoidja haldus. Esimesena kuvatakse pildi kohahoidja kasutades ära BlurHash teegi võimalusi, siis jälgitakse kui päris pilt ära laeb, kaob kohahoidja sujuvalt vaatest ning tekib päris pilt nähtavale. Mõne hetke pärast kaob DOM’st (Dokumendi objektmudel) ka kohahoidja element, kuna see pole enam niikuinii nähtaval.

Piltide kustutamiseks on lisatud pildi nurka üks nupp prügikasti ikooniga, mis ilmub nähtavale kui pildi kohal kursoriga hõljuda. Mobiilseadmete jaoks on antud nupp koguaeg nähtaval. Et vältida kogemata vajutusi on tehtud, et nupule vajutades muutub nupu ikoon linnukeseks ning sellele vajutades kustutatakse pilt. See tähendab, et kokku peab kaks korda vajutama ühte nuppu, et pilt kustuks. Kui pilti kustutamine õnnestub või kui midagi läheb valesti, kuvatakse kasutajale ka vastavad teated. Veel, pildi kustutamisel kohandavad ülejäänud pildid ennast ümber, et täita tühja kohta.

Valguskastis kuvamiseks kasutatakse ära eelnevalt väljaselgitatud Yet Another React Lightbox teegi võimalusi. Valguskast avatakse kui vajutada pildile – valguskast avaneb selle pildiga, millele parasjagu vajutati. Selleks antakse valguskastile ette pildi indeks. Veel lisatakse valguskastile kaks pistikprogrammi: üks mis võimaldab näha all piltide rada ning ka selle abil navigeerida ja teine mis lubab pilte suumida.

Juhul kui kasutaja pole veel ühtegi pilti ülesse laadinud, ehk teenusepoolne rakendus tagastab tühja massiivi, siis kuvatakse kohahoidja mis viitab pildi ülesse laadimise URL'le. Selleks, et vältida kohahoidja ilmumist ennem kui teenusepoolne rakendus jõuab midagi tagastada, kontrollitakse ennem kuvamist pildi massiivi eksisteerimist. See tähendab, et piltide massiivi muutuja ei tohi olla *undefined* ehk defineerimata.

Kõikidele elementidele on loodud ka stiilid, et veebirakendus näeks välja ning tunduks modernne, reageeriv ja sujuv. Stiilideks on rakenduses kasutusel nii Material UI võimalused kui ka rakendusse laetud styled-components'ga loodud stiilitud komponendid.

Lisas 2 on näha loodud kliendipoolse rakenduse vaateid nii mobiilis kui ka arvutis.

6 Lõpplahenduse testimine

Antud peatükis testitakse lõpplahendust ning vaadatakse, et see oleks ka eelnevalt määratud nõuetele vastav. Antud lõputöö raames teostatakse lõpplahenduse testimist manuaalselt.

Esimesena vaadatakse, et rakendus oleks soovitud brauserites toimiv. Nendeks brauseriteks on Chrome, Firefox ning Edge. Kõikides eelnevalt mainitud brauserites testitakse kõiki funktsionaalseid nõudeid, et veenduda nende toimimises. Testide tulemused on:

- Chrome – Pilte saab laadida klikkides ning sirvides kohalikku failisüsteemi ja ka neid lohistades. Saab laadida ühte kui ka mitut pilti korraga ning kõikides määratud pildiformaatides. Vigade puhul kuvatakse edukalt ka veateated ning eduka ülesse laadimise puhul on näha kohe ka ülesse laetud pilti või pilte. Kõikide piltide vaates on kõik pildid ilusti nähtaval ning hajuvad sujuvalt vaatesse sisse. Pilte on võimalik ka kustutada ning saab sirvida valguskastis.

- Firefox – Samasugused tulemused nagu Chrome'i puhul.

- Edge – Läbis samasuguste tulemustega nagu eelnevad kaks brauserit.

Järgmisena testitakse rakendust erinevate seadmetega ja ekraanisuurustega. Erinevate seadmete ja ekraanisuuruste emuleerimiseks kasutatakse ära Chrome'i sisse ehitatud arendajate tööriistade komplektis olevat *Device Mode*'i ehk seadme režiimi. Testitud seadmed on: iPhone SE, Samsung Galaxy A51/A71 ning iPad Air. Testitakse nii maastiku kui ka portree seadme orientatsioone. Veel leitakse ekraani suurus kus maalt pole veebirakendus enam kasutatav. Jällegi, nagu eelnevalt brauserite puhul, testitakse kõiki funktsionaalseid nõudeid.

Testimise tulemusena leiti, et veebirakendus on kõikidel testitud seadmetel mõlemas orientatsioonis täielikult kasutatav ja funktsionaalne. Minimaalne ekraani laius on 206 pikslit ning kõrgus 260 pikslit – nendest allapoole minnes ei ole veebileht enam väga hästi loetav ning arusaadav, aga need on tõesti väga väikesed mõõdud ning ükski tänapäeval kasutusel olev seade ei tohiks sinna lähedale isegi minna.

Järgmiseks on testitud vigade haldust. Selleks saab meelega tekitatud veaolukordi ning visatud teenusepoolsest rakendusest vigu funktsioonidest millega kasutaja saab suhelda ehk tekitatakse operatiivseid vigu. Programmeerija poolsete vigade puhul on oodatud, et rakendus lakkab töötamast kuna viitab rakenduse sisesele koodi veale. Operatiivsete vigade puhul on oodatavaks tulemuseks on see, et rakendus jätkab tööd, aga ka kasutajale kuvatakse veateated. Vigu loobitakse kõikidest äriloogika meetoditest ning ka Google Cloud Storage'i teenustest.

Eelnevalt loobitud vigadest tekkis kõikide puhul oodatud olukord ehk rakendus jätkas töötamast, aga kõikide puhul ei olnud kasutajale kuvatud veateadet. Kui piltide pärimisel visata viga, siis seda ei kuvatud kasutajale. Selle parandamiseks sai lisatud kliendipoolsesse rakendusse piltide pärimisse ka vigade kinni püüdmine ning vea puhul nüüd ka kuvatakse veateade.

Viimaseks on vaadatud kui kaua aega kulub üksiku ühe megabaidise faili ülesse laadimiseks ning kolme ühe megabaidise faili ülesse laadimiseks korraga ning nelja ülesse laetud pildi pärimiseks ning võrreldakse eelnevalt salvestamisvõimaluste võrdlemise juures saavutatud tulemusega. Veel vaadatakse kui kaua aega kulub nüüd nelja faili lugemisele keskmiselt ning jällegi võrreldakse eelnevalt leitud salvestamisvõimaluste võrdluses leitud tulemusega.

Kolmel korral üksiku ühe megabaidise faili ülesse laadimiseks läks keskmiselt 331 millisekundit ning kolme ühe megabaidise faili ülesse laadimiseks keskmiselt 415 millisekundit. Võrreldes eelnevalt salvestamisvõimaluste võrdluses tehtud katsetega, kus mõlema salvestamise tulemuseks kokku oli 484.6 millisekundit, on lõpplahenduse salvestamise tulemuseks kokku 746 millisekundit ehk kogu funktsionaalsuse ning lisafunktsioonide lisamisega rakendusse lisandus salvestamise tulemusse kuskil 263 millisekundit.

Nelja faili lugemisele kulus lõpplahendusel kolmel korral keskmiselt 32 millisekundit, mis on salvestamisvõimaluste võrdluses saavutatud 39.3 millisekundist 7.3 millisekundit kiirem. Erinevus tuleb tõenäoliselt asjaolust, et ei loeta enam otse Google Cloud Storage'st vaid andmebaasist, kus on viited Google Cloud Storage's olevatele piltidele.

7 Hinnang lõpptulemusele

Lõpplahendusele sai rakendatud kõik väljaselgitatud parimaid tehnoloogiad piltide ülesse laadimiseks, optimeerimiseks ning kuvamiseks mille abiga said valmis kõik analüüsi käigus määratud funktsionaalsed ja mittefunktsionaalsed nõuded. Veel, kasutatud teekidega ning loodud visuaalsete komponentidega ja stiilimisega saavutati lõpplahenduses modernne visuaalne tunnetus.

Rakendatud tehnoloogiad võimaldavad eraldiseisvalt kõik midagi kasulikku, aga lõpplahenduses pidi need kõik korraga kasutusele võtma ja kõik tehnoloogiad pidid üksteist täiendama. Näiteks: valguskastis piltide kuvamiseks mõeldud teek lisas valguskasti funktsionaalsuse, sh ka suumimise, aga selle avamiseks pidi Material UI poolt pakutud pildigaleriis piltidele lisama nii-öelda nupu funktsionaalsuse ehk pildile vajutades pidi avanema valguskast just selle pildiga kuhu kasutaja vajutab.

Programmikoodi kirjutamisel ning failide organiseerimisel kasutati parimaid tavasid mistõttu on lõpptulemuse kood hästi organiseeritud ning loogiliselt tükkideks jaotatud. Antud asjaolu on ilmselt ka lõpplahenduse suurimaks eeliseks.

Mõlemale loodud rakenduse poolele sai lisatud ka kasutusjuhendi failid, mis seletab kuidas rakendust tööle panna, mis selle jaoks vaja on ning kuidas erinevaid konfiguratsioone muuta saab ja kuidas üldse rakendust kasutada.

Kuna loodud rakendus võiks olla mõnele väga heaks põhjaks edasi arenduseks, õppimiseks vms, siis on lõpptulemuse kasutuselevõtmist katsetatud autoriga sarnaste tehnoloogiliste oskustega inimese peal. Nendele oli projekti tööle saamine ja selle kasutamine intuitiivne, aga loodud kasutusjuhendist oli ka palju kasu.

Lõpplahenduse kliendipoolne rakendus ja ka teenusepoolne rakendus on leitav GitHub keskkonnast:

- Kliendipoolne rakendus: https://github.com/RainerAas/image-gallery-web
- Teenusepoolne rakendus: https://github.com/RainerAas/image-gallery-api

7.1 Võimalused edasiseks arenduseks

Antud projekti on võimalus edaspidi edasi arendada. Esiteks, mistahes rakendust sellest edasi arendada, võiks mõelda ka erinevate keelte valikute peale, et kasutaja kogemus oleks veelgi mugavam ning vabaneda koodi sisestest sõnedest. Kõik sõned tuleks eraldada tõlkefailidesse, et kõik need oleksid ühe koha pealt leitavad ning nende muutmine oleks lihtsam. Selleks võiks kasutusele võtta raamistiku nimega react-i18next ning koostada tõlkefailid .JSON failide kujul *public* kausta.

Kui tahta viia galerii funktsionaalsus veebirakenduses veelgi paremale tasemele, siis võiks esimese asjana kliendipoolsel rakendusel pakkuda erinevaid pildi manipulatsiooni võimalusi, näiteks kärpimist ning filtrite lisamist, ennem kui pilt üldse saata teenusepoolsele rakendusele ülesse laadimiseks. Veel, kui pilt ei vasta nõuetele, võiks saada pildi teisendada nii, et see oleks nõuetele vastav ning saaks ikkagi ülesse laadida. Selleks tuleks kuvada kasutajale teavitus ning pakkuda valik kas siis pildi teisendamiseks või antud pildi vahele jätmiseks.

Kasutajale oleks veel mõistlik lisada ühe kuni mitme pildi selekteerimise valik, et kustutada näiteks mitut pilti korraga, vältides seeläbi korduvat klikkimist. Valitud piltidega võiks saada veel erinevaid operatsioone teostada, näiteks: lisada lemmikutesse või valitud pilte alla laadida. Veel võiks lubada kasutajal pilte galeriis ümber tõsta või isegi grupeerida.

Mõeldes kasutusmugavusele, oleks üheks kindlaks mugavuseks veel arendada teatud filtreerimine. Suure koguse piltide tõttu võib teatud piltide leidmine osutuda keeruliseks. Selleks saaks lisada näiteks filtri, mis võimaldaks piltide sorteerimist näiteks ülesse laadimise kuupäeva järgi, otsimist kuupäeva vahemiku või nime järgi. Veel, kui lisada eelnevalt kirjeldatud piltide lemmikuks lisamine saaks teha otsingu kus on ainult lemmikuks märgitud pildid nähtaval.

Jätkates suure pildi koguse teemal, siis kuigi loodud lõpplahenduses on kasutusel niinimetatud laisk laadimine mis tähendab, et piltide sisu laetakse päriselt alles siis, kui need kasutaja vaatesse tekivad, oleks kindlasti mõistlik lisada ka teatud paginatsioon ehk vaate jagamine lehekülgedeks, et vältida veebirakenduse aeglustamist läbi lõputu arvu piltide päringu teostamise. Selleks võiks piirata piltide päringu juures päritud piltide arvu mingi mõistliku arvuni. Seejärel saaks jagada kas pildid erinevatele lehekülgedele või

teha see lõputu kerimise stiilis nii, et kui kasutaja on teatud piltide arvuni kerinud, päritakse uus sama kogus pilte sealt maalt kus eelmine päring lõppes ning laetakse need sisse. See protsess korduks nii kaua, kuniks pole enam ühtegi pärimata pilti.

Kuna koostatud rakendus on väga heaks põhjaks mõnele huvilisele, siis on tegelikkuses edasi arenduseks lõputult võimalusi – sellele võib ükskõik missuguse rakenduse ehitada.

8 Kokkuvõte

Antud lõputöö eesmärgiks oli uurida erinevaid teeke pildifailide töötluseks ning kuvamiseks ja erinevaid salvestamise võimalusi ning leida neist parim kombinatsioon. Selleks pidi kõiki variante katsetama loodud piltide üleslaadimiseks mõeldud rakenduse peal, millele pidi lõpuks rakendama kõik väljaselgitatud parimad võimalused.

Lõputöö analüüsis toodi välja erinevaid tehnoloogiaid ja teeke probleemi lahendamiseks, määrati nõuded ja ka loodud rakenduse skoop. Analüüsi käigus tehti kindlaks kuidas toimus teekide ning salvestamisvõimaluste võrdlemine. Lõputöö lahenduse läbiviimises selgitati kuidas on nii klientrakendus kui ka teenusepoolne rakendus ülesse ehitatud, mis tehnoloogiaid kasutati ning kuidas erinevaid tehnoloogilisi otsuseid läbi viidi.

Loodud rakenduses sai kasutusele võetud kõik parimad tehnoloogiad piltide ülesse laadimise protsessi jaoks, kus toimub piltide optimeerimine ning pilve salvestamine. Ka kuvamise jaoks, kus toimub piltide sujuv sisse laadimine ning on võimaldatud nii galeriis kui ka valguskastis kuvamine. Analüüsi käigus määratud nõuded said kõik täidetud. Koostatud rakendus on väga heaks põhjaks mõnele arendajale, ehitamaks uut rakendust või lihtsalt mõnele õppurile, kes soovib sellega midagi õppida või selle peal oma katsetusi teha.

Kasutatud kirjandus

[1] W3Techs, „Usage statistics of JavaScript as client-side programming language on websites,“ veebruar 2023. [Võrgumaterjal]. Available: https://w3techs.com/technologies/details/cp-javascript. [Kasutatud 15 veebruar 2023].

[2] Stack Overflow, „2022 Stack Overflow Developer Survey - Web frameworks and technologies,“ 2022. [Võrgumaterjal]. Available: https://survey.stackoverflow.co/2022/#most-popular-technologies-webframe. [Kasutatud 16 veebruar 2023].

[3] State of JavaScript, „Front-end frameworks,“ 2022. [Võrgumaterjal]. Available: https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/. [Kasutatud 16 veebruar 2023].

[4] ReactJS, „React,“ 2023. [Võrgumaterjal]. Available: https://reactjs.org/. [Kasutatud 16 veebruar 2023].

[5] ReactJS, „Introducing JSX,“ 2023. [Võrgumaterjal]. Available: https://reactjs.org/docs/introducing-jsx.html. [Kasutatud 16 veebruar 2023].

[6] Angular, „Understanding Angular,“ 2023. [Võrgumaterjal]. Available: https://angular.io/guide/understanding-angular-overview. [Kasutatud 17 veebruar 2023].

[7] Vue.js, „Introduction,“ 2023. [Võrgumaterjal]. Available: https://vuejs.org/guide/introduction.html. [Kasutatud 17 veebruar 2023].

[8] MDN Web Docs, „Getting started with Svelte,“ 2022. [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Svelte_getting_started. [Kasutatud 17 veebruar 2023].

[9] H. Boisdequin, „React vs Vue vs Angular vs Svelte,“ oktoober 2020. [Võrgumaterjal]. Available: https://dev.to/hb/react-vs-vue-vs-angular-vs-svelte-1fdm. [Kasutatud 18 veebruar 2023].

[10] Material UI, „Material UI - Overview,“ [Võrgumaterjal]. Available: https://mui.com/material-ui/getting-started/overview/. [Kasutatud 19 veebruar 2023].

[11] MDN Web Docs, „Express/Node introduction,“ september 2022. [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction. [Kasutatud 20 veebruar 2023].

[12] MongoDB, „NoSQL vs. SQL Databases,“ [Võrgumaterjal]. Available: https://www.mongodb.com/nosql-explained/nosql-vs-sql. [Kasutatud 20 veebruar 2023].

[13] Stack Overflow, „2022 Stack Overflow Developer Survey - Databases,“ 2022. [Võrgumaterjal]. Available: https://survey.stackoverflow.co/2022/#most-popular-technologies-database. [Kasutatud 21 veebruar 2023].

[14] Amazon Web Services, „Amazon DynamoDB,“ [Võrgumaterjal]. Available: https://aws.amazon.com/dynamodb/. [Kasutatud 22 veebruar 2023].

[15] MongoDB, „Comparing DynamoDB and MongoDB,“ [Võrgumaterjal]. Available: https://www.mongodb.com/compare/mongodb-dynamodb. [Kasutatud 22 veebruar 2023].

[16] Amazon, „What is An IDE?,“ [Võrgumaterjal]. Available: https://aws.amazon.com/what-is/ide/. [Kasutatud 25 veebruar 2023].

[17] K. Yusov, „Top 15 JavaScript IDEs and JS Editors for Frontend Development 2023,“ 2023. [Võrgumaterjal]. Available: https://jelvix.com/blog/best-javascript-ides. [Kasutatud 25 veebruar 2023].

[18] Microsoft, „Compare Visual Studio 2022 Editions,“ 2022. [Võrgumaterjal]. Available: https://visualstudio.microsoft.com/vs/compare/. [Kasutatud 25 veebruar 2023].

[19] JetBrains, „Subscription options & Pricing,“ 2023. [Võrgumaterjal]. Available: https://www.jetbrains.com/webstorm/buy/#personal. [Kasutatud 25 veebruar 2023].

[20] TrustRadius, „Version Control Software,“ 2023. [Võrgumaterjal]. Available: https://www.trustradius.com/version-control. [Kasutatud 26 veebruar 2023].

[21] GitHub, „About,“ 2023. [Võrgumaterjal]. Available: https://github.com/about. [Kasutatud 27 veebruar 2023].

[22] M. Woodward, „Octoverse 2022: 10 years of tracking open source,“ GitHub, november 2022. [Võrgumaterjal]. Available: https://github.blog/2022-11-17-octoverse-2022-10-years-of-tracking-open-source/. [Kasutatud 27 veebruar 2023].

[23] GitHub, „Pricing,“ 2023. [Võrgumaterjal]. Available: https://github.com/pricing. [Kasutatud 27 veebruar 2023].

[24] Bitbucket, „A brief overview of Bitbucket,“ 2023. [Võrgumaterjal]. Available: https://bitbucket.org/product/guides/getting-started/overview. [Kasutatud 27 veebruar 2023].

[25] Bitbucket, „Plans and pricing,“ 2023. [Võrgumaterjal]. Available: https://www.atlassian.com/software/bitbucket/pricing. [Kasutatud 27 veebruar 2023].

[26] GitLab, „About GitLab,“ 2023. [Võrgumaterjal]. Available: https://about.gitlab.com/company/. [Kasutatud 27 veebruar 2023].

[27] GitLab, „Pricing,“ 2023. [Võrgumaterjal]. Available: https://about.gitlab.com/pricing/. [Kasutatud 27 veebruar 2023].

[28] Postman, „2022 State of the API Report,“ 2022. [Võrgumaterjal]. Available: https://www.postman.com/state-of-api/. [Kasutatud 1 märts 2023].

[29] Red Hat, „REST vs. SOAP,“ aprill 2019. [Võrgumaterjal]. Available: https://www.redhat.com/en/topics/integration/whats-the-difference-between-soap-rest. [Kasutatud 2 märts 2023].

[30] npm, „About npm,“ [Võrgumaterjal]. Available: https://docs.npmjs.com/about-npm. [Kasutatud 2 märts 2023].

[31] npm, „Sharp,“ 2023. [Võrgumaterjal]. Available: https://www.npmjs.com/package/sharp. [Kasutatud 4 märts 2023].

[32] npm, „Jimp,“ 2023. [Võrgumaterjal]. Available: https://www.npmjs.com/package/jimp. [Kasutatud 4 märts 2023].

[33] MDN Web Docs, „Canvas API,“ 2023. [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API. [Kasutatud 4 märts 2023].

[34] npm, „node-canvas,“ 2023. [Võrgumaterjal]. Available: https://www.npmjs.com/package/canvas. [Kasutatud 4 märts 2023].

[35] npm, „Yet Another React Lightbox,“ 2023. [Võrgumaterjal]. Available: https://www.npmjs.com/package/yet-another-react-lightbox. [Kasutatud 6 märts 2023].

[36] npm, „react-spring-lightbox,“ 2023. [Võrgumaterjal]. Available: https://www.npmjs.com/package/react-spring-lightbox. [Kasutatud 6 märts 2023].

[37] PhotoSwipe, „Getting Started,“ 2023. [Võrgumaterjal]. Available: https://photoswipe.com/getting-started/. [Kasutatud 6 märts 2023].

[38] npm, „React Photo Gallery,“ 2023. [Võrgumaterjal]. Available: https://www.npmjs.com/package/react-photo-gallery. [Kasutatud 6 märts 2023].

[39] MongoDB, „JSON and BSON,“ 2023. [Võrgumaterjal]. Available: https://www.mongodb.com/json-and-bson. [Kasutatud 7 märts 2023].

[40] MongoDB, „GridFS,“ 2023. [Võrgumaterjal]. Available: https://www.mongodb.com/docs/manual/core/gridfs/. [Kasutatud 7 märts 2023].

[41] Google Cloud, „What is Object storage?,“ 2023. [Võrgumaterjal]. Available: https://cloud.google.com/learn/what-is-object-storage. [Kasutatud 7 märts 2023].

[42] Stack Overflow, „2022 Stack Overflow Developer Survey - Cloud platforms,“ 2022. [Võrgumaterjal]. Available: https://survey.stackoverflow.co/2022/#technology-most-popular-technologies. [Kasutatud 8 märts 2023].

[43] Google, „Google Cloud products,“ 2023. [Võrgumaterjal]. Available: https://cloud.google.com/products. [Kasutatud 8 märts 2023].

[44] Google, „Free cloud features and trial offer,“ veebruar 2023. [Võrgumaterjal]. Available: https://cloud.google.com/free/docs/free-cloud-features#free-tier-usage-limits. [Kasutatud 8 märts 2023].

[45] Amazon Web Services, „Amazon S3,“ 2023. [Võrgumaterjal]. Available: https://aws.amazon.com/s3/. [Kasutatud 8 märts 2023].

[46] Amazon Web Services, „AWS Free Tier,“ 2023. [Võrgumaterjal]. Available: https://aws.amazon.com/free. [Kasutatud 8 märts 2023].

[47] Amazon Web Services, „Amazon CloudFront,“ 2023. [Võrgumaterjal]. Available: https://aws.amazon.com/cloudfront/. [Kasutatud 8 märts 2023].

[48] Microsoft, „Introduction to Azure Blob Storage,“ veebruar 2023. [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction. [Kasutatud 8 märts 2023].

[49] Microsoft Azure, „Create Your Azure Free Account Today,“ 2023. [Võrgumaterjal]. Available: https://azure.microsoft.com/en-us/free/. [Kasutatud 8 märts 2023].

[50] statcounter, „Desktop Browser Market Share Worldwide,“ 2022. [Võrgumaterjal]. Available: https://gs.statcounter.com/browser-market-share/desktop/worldwide/2022. [Kasutatud 9 märts 2023].

[51] Lumen Learning, „Borda Count,“ [Võrgumaterjal]. Available: https://courses.lumenlearning.com/waymakermath4libarts/chapter/borda-count/. [Kasutatud 11 märts 2023].

[52] A. Polak, „Reduce Node modules for better performance,“ 2019. [Võrgumaterjal]. Available: https://tsh.io/blog/reduce-node-modules-for-better-performance/. [Kasutatud 11 märts 2023].

[53] heynode, „What is package.json?,“ [Võrgumaterjal]. Available: https://heynode.com/tutorial/what-packagejson/. [Kasutatud 11 märts 2023].

[54] nodemon, „nodemon,“ [Võrgumaterjal]. Available: https://nodemon.io/. [Kasutatud 11 märts 2023].

[55] M. Aranda, „What's the difference between JavaScript and ECMAScript?,“ oktoober 2017. [Võrgumaterjal]. Available: https://www.freecodecamp.org/news/whats-the-difference-between-javascript-and-ecmascript-cba48c73a2b5/. [Kasutatud 11 märts 2023].

[56] ESLint, „Getting Started with ESLint,“ [Võrgumaterjal]. Available: https://eslint.org/docs/latest/use/getting-started. [Kasutatud 12 märts 2023].

[57] A. Okoro, „How to Setup Babel in Node.js,“ august 2021. [Võrgumaterjal]. Available: https://www.freecodecamp.org/news/setup-babel-in-nodejs/. [Kasutatud 12 märts 2023].

[58] Babel, „@babel/core,“ [Võrgumaterjal]. Available: https://babeljs.io/docs/babel-core. [Kasutatud 12 märts 2023].

[59] Babel, „@babel/preset-env,“ [Võrgumaterjal]. Available: https://babeljs.io/docs/babel-preset-env. [Kasutatud 14 märts 2023].

[60] Babel, „@babel/cli,“ [Võrgumaterjal]. Available: https://babeljs.io/docs/babel-cli. [Kasutatud 15 märts 2023].

[61] Babel, „Using Babel,“ [Võrgumaterjal]. Available: https://babeljs.io/setup#installation. [Kasutatud 16 märts 2023].

[62] Babel, „@babel/plugin-transform-runtime,“ [Võrgumaterjal]. Available: https://babeljs.io/docs/babel-plugin-transform-runtime. [Kasutatud 15 märts 2023].

[63] npm, „@babel/eslint-parser,“ [Võrgumaterjal]. Available: https://www.npmjs.com/package/@babel/eslint-parser. [Kasutatud 15 märts 2023].

[64] Babel, „@babel/node,“ [Võrgumaterjal]. Available: https://babeljs.io/docs/babel-node. [Kasutatud 16 märts 2023].

[65] npm, „Multer,“ 2023. [Võrgumaterjal]. Available: https://www.npmjs.com/package/multer. [Kasutatud 17 märts 2023].

[66] M. J. Ryan, „Directory structure for JavaScript/Node projects,“ 2018. [Võrgumaterjal]. Available: https://gist.github.com/tracker1/59f2c13044315f88bee9. [Kasutatud 20 märts 2023].

[67] Y. Goldberg, „Separate Express 'app' and 'server',“ 2020. [Võrgumaterjal]. Available:

https://github.com/goldbergyoni/nodebestpractices/blob/master/sections/projectstructre/separateexpress.md. [Kasutatud 20 märts 2023].

[68] R. D. John Au-Yeung, „Best practices for REST API design,“ märts 2020. [Võrgumaterjal]. Available: https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/#h-versioning-our-apis. [Kasutatud 21 märts 2023].

[69] Y. Goldberg, „Structure your solution by components,“ november 2022. [Võrgumaterjal]. Available: https://github.com/goldbergyoni/nodebestpractices/blob/master/sections/projectstructre/breakintcomponents.md. [Kasutatud 22 märts 2023].

[70] MongoDB, „What is MongoDB Atlas?,“ 2023. [Võrgumaterjal]. Available: https://www.mongodb.com/docs/atlas/. [Kasutatud 22 märts 2023].

[71] MongoDB, „MongoDB Pricing,“ 2023. [Võrgumaterjal]. Available: https://www.mongodb.com/pricing. [Kasutatud 22 märts 2023].

[72] npm, „dotenv,“ 2023. [Võrgumaterjal]. Available: https://www.npmjs.com/package/dotenv. [Kasutatud 24 märts 2023].

[73] KnowledgeHut, „Installing Dev Dependencies with npm: Beginners' Guide,“ veebruar 2023. [Võrgumaterjal]. Available: https://www.knowledgehut.com/blog/web-development/npm-install-dev-dependencies. [Kasutatud 24 märts 2023].

[74] A. Hovhannisyan, „Dynamically Importing Components with React.lazy,“ juuni 2022. [Võrgumaterjal]. Available: https://www.aleksandrhovhannisyan.com/blog/react-lazy-dynamic-imports/. [Kasutatud 25 märts 2023].

[75] Axios, „Getting Started,“ [Võrgumaterjal]. Available: https://axios-http.com/docs/intro. [Kasutatud 26 märts 2023].

[76] S. Shah, „React JS imports made easy with Absolute paths using jsconfig.json,“ detsember 2020. [Võrgumaterjal]. Available: https://saurabhshah23.medium.com/react-app-with-absolute-paths-using-jsconfig-json-2b07b1cb24d4. [Kasutatud 31 märts 2023].

[77] sharp, „Resizing images,“ [Võrgumaterjal]. Available: https://sharp.pixelplumbing.com/api-resize. [Kasutatud 1 aprill 2023].

[78] W3Schools, „Node.js Buffer Module,“ [Võrgumaterjal]. Available: https://www.w3schools.com/nodejs/ref_buffer.asp. [Kasutatud 2 aprill 2023].

[79] npm, „Multer's GridFS storage engine,“ 2023. [Võrgumaterjal]. Available: https://www.npmjs.com/package/multer-gridfs-storage. [Kasutatud 2 aprill 2023].

[80] Express, „Serving static files in Express,“ [Võrgumaterjal]. Available: https://expressjs.com/en/starter/static-files.html. [Kasutatud 2 aprill 2023].

[81] Paul Diamond, „Cloud storage vs. on-premises servers: 9 things to keep in mind,“ september 2020. [Võrgumaterjal]. Available: https://www.microsoft.com/en-ww/microsoft-365/business-insights-ideas/resources/cloud-storage-vs-on-premises-servers. [Kasutatud 3 aprill 2023].

[82] Wolt, „BlurHash,“ [Võrgumaterjal]. Available: https://blurha.sh/. [Kasutatud 4 aprill 2023].

[83] npm, „blurhash,“ 2023. [Võrgumaterjal]. Available: https://www.npmjs.com/package/blurhash. [Kasutatud 4 aprill 2023].

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

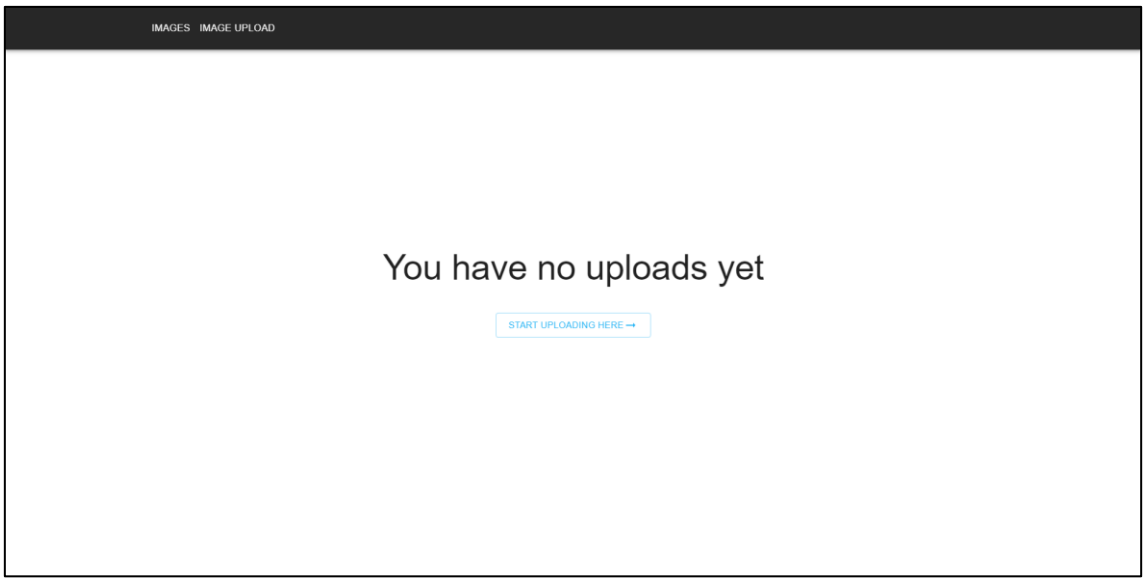
Mina, Rainer Aas

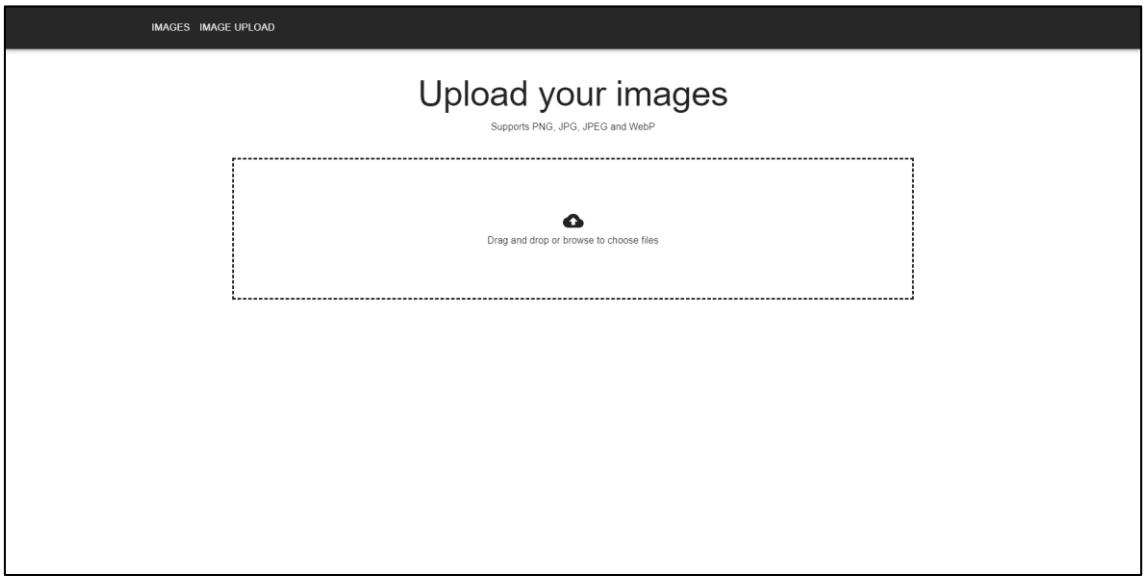
1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Pildigalerii veebirakenduse arendus“, mille juhendaja on Meelis Antoi
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil 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.

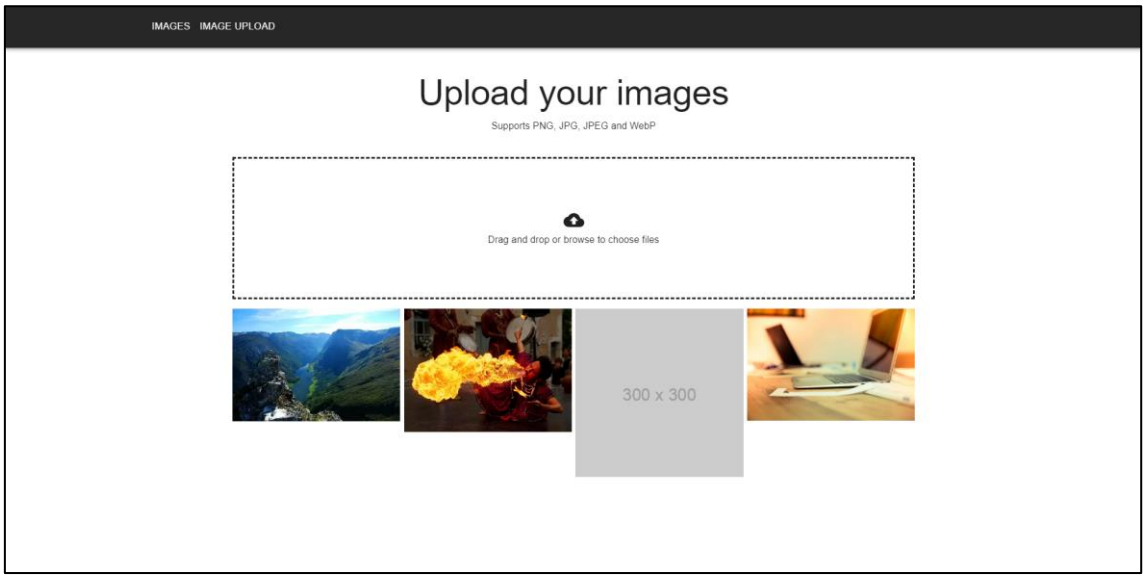
17.04.2023

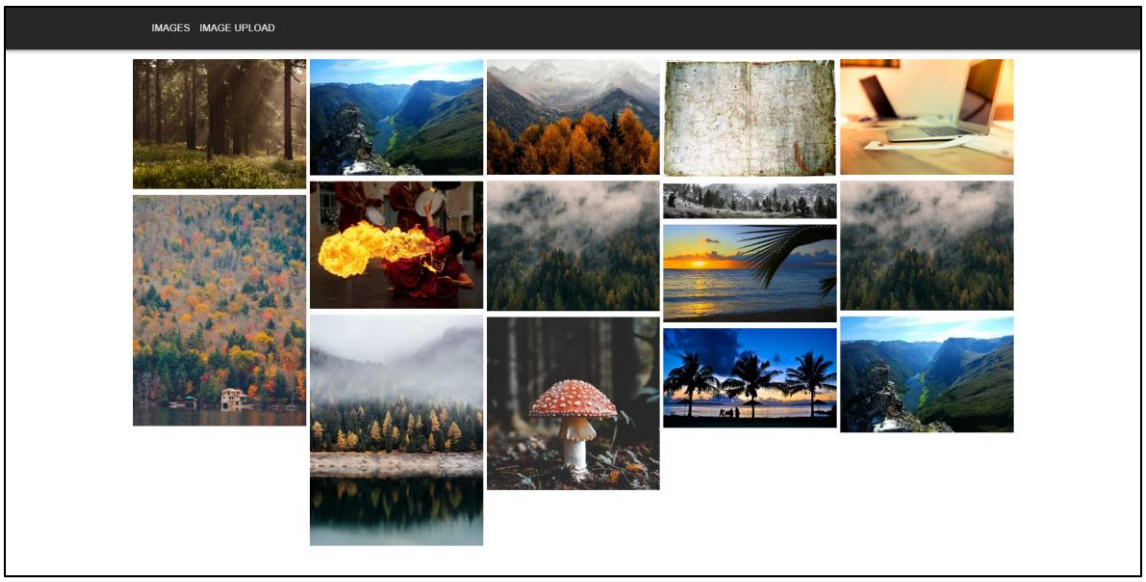
1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või 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.

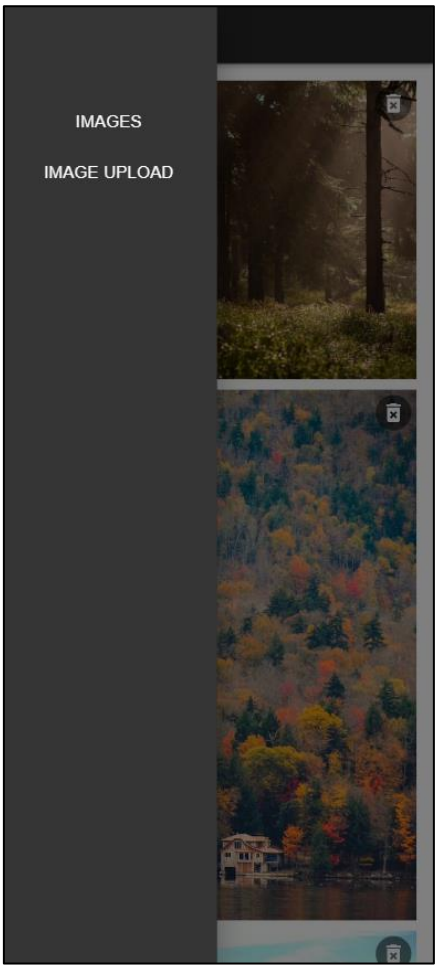
Lisa 2 – Kliendipoolse rakenduse vaated






IMAGES IMAGE UPLOAD
Upload your images
Supports PNG, JPG, JPEG and WebP
Drag and drop or browse to choose files
300 x 300


IMAGES IMAGE UPLOAD


IMAGES
IMAGE UPLOAD