TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Rasmus Paks 155152IAPB

JAVASCRIPTI NODE.JS NING JAVA SPRING PLATVORMIL REALISEERITUD VEEBIRAKENDUSTE ANALÜÜS NING VÕRDLUS

bakalaureusetöö

Juhendaja: Jekaterina Tšukrejeva
MSc

Tallinn 2018

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: Rasmus Paks

21.05.2018

Annotatsioon

Käesoleva lõputöö raames arendati kaks rakendust, mille funktsionaalsus on identne. Arenduste eesmärk oli võrrelda erinevaid tehnoloogiaid ja lähenemisi veebirakenduste loomisel ning tekitada võrdlusmoment sama funktsionaalsuse arendamises kahel erineval arendajate seas populaarsel platvormil ning lähenemisel. Lõputöö raames võrreldi platvorme ja lähenemisi sellistes aspektides, mis on veebirakenduse arendamisel üldjuhul triviaalsed ülesanded. Rakenduse esimese variandi arendas autor oma tööülesannete käigus ettevõttes töötades ning teise versiooni hiljem, et saaks tekitada kahe populaarse platvormi eelised ja puudused.

Töö tulemusena tekkis ülevaade ja võrdlus erinevatest raamistikest ning lähenemistest ja mida arvesse võtta nende juurutamisel arenduses. Töö kõige olulisemaks tulemuseks on asjaolu, et Java Spring platvorm on täna arendajatele asendamatu ja võimekas raamistik, mida kasutades on otstarbekas ning efektiivne arendada veebirakendusi, mis kasutavad oma andmebaasisüsteemis objekt relatsioonilist andmemudelit ning mis peavad võimaldama kasutajate lisatud failidega töötamist.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 29 leheküljel, 5 peatükki, 21 joonist, 1 tabelit.

Abstract

The analysis and comparison of web applications developed on JavaScript Node.js and Java Spring platform

In this study, two web applications were developed and both of them have the same functionality. The aim of the development is to compare different frameworks and approaches in web application development process and to create comparison between developing the same functionality and different approaches in two of the most popular back end frameworks today - Java Spring and JavaScript Node.js framework. Aspects of development that are compared in this study are chosen because these functionalities are quite basic tasks in web application development. The first application was developed on Java Spring framework during author' s working time in a company and the second one later in order to compare these two majorly popular frameworks.

As a result of this study, an overview of two different frameworks and approaches were made and also of what to take into consideration when adapting those frameworks and approaches into development process. The most important outcome of the study is that Java Spring framework is irreplaceable and powerful framework which benefits mostly web application development when using object-relational database management systems as a data source and require handling user uploaded file management procedures. Secondly, it is useful to adapt some kind of templating engine for HTML in case developed application requires HTML based user interface in order to be able to create less user interface descriptions and more structured component description system and use parts of the user interface repeatedly.

The thesis is in Estonian and contains 29 pages of text, 5 chapters, 21 figures, 1 table.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, reeglistik, mille alusel rakendusprogramm saab teiselt programmilt teenuseid [1].
Back end	Rakenduse kasutajale nähtamatu osa programmis.
Front end	Kasutajaga või väliskeskkonnaga liidestatav osa programmis [2].
HTTP	*Hypertext Transfer Protocol*, klient-server-protokoll HTML-dokumentide edastuseks veebis [3].
URL	*Uniform Resource Locator*, Interneti-aadress, mis on mõeldud Interneti objekti määratlemiseks [4].
DOM	*Document Object Model*, HTML'i kirjutamiseks kasutatav API [5].
HTML	*HyperText Markup Language*, hüperteksti märgistuskeel, milles märgistatakse veebilehti [6].
CSS	*Cascading Style Sheets*, HTML-dokumendi ilmet määravate laadilehtede hierarhiline süsteem [7].

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Paljudes tarkvaraarendusprojektides kasutatakse arenduskiiruse ning lihtsuse huvides ära erinevaid teiste arendajate poolt loodud arendusraamistikke. Raamistikke on väga erinevaid, need võivad olla kliendipoolse funktsionaalsuse loomiseks, nagu jQuery või serveripoolsed, nagu näiteks antud lõputöös kasutatavad Spring ja Node.js raamistikud. Raamistiku kasutamise eesmärk on eelkõige mitte arendada ja testida enda loodud funktsionaalsust vaid kasutada kindla funktsionaalsuse tegemiseks teiste poolt juba loodud ja testitud tarkvara. Raamistiku valik määrab suurel hulgal ära edasise arenduse käigu - lihtsuse, kiirus ja efektiivsuse. Samade eesmärkidega raamistikke on palju ning sageli on keeruline esmapilgul valida raamistike seast just enda arendusse selline, mis võimaldaks luua soovitud funktsionaalsust efektiivselt, millel oleks korralik dokumentatsioon ning mille õppimiskurv ei oleks liialt suur.

Lõputöö eesmärk on realiseerida kaks sama funktsionaalsusega rakendust kasutades arendustegevuses erinevaid raamistikke ning lähenemisi ja luua võrdlusmoment veebirakenduste arendamisel kasutatud raamistikest ja lähenemistest. Sellise võrdluse alusel on arendajatel võimalik hinnata raamistiku ja lähenemise sobivust enda arendusse ning võimalikke kasutamisel tekkivaid probleeme ja kitsendusi.

Lõputöö arenduste lähtetingimused andis autorile ettevõtte, kellel oli vaja veebirakendusse luua töös kirjeldatav moodul ning ülesandeks on veebirakenduste arendamine ning arendustegevuse käigus märkmete tegemine ja võrdlusmomendi leidmine ning selle lugejale arusaadavalt ja lühidalt lõputöösse kirjutamine.

Käesoleva töö põhisisu on jaotatud kolmeks peatükiks, mille sisu on järgnev:
Peatükis 2 kirjeldatakse ning tuuakse praktilisi koodinäiteid arendustegevuses kasutuses olevate raamistike ning arhitektuuri osas.
Peatükis 3 kirjeldatakse realiseeritud rakenduste võimalusi ning kasutusjuhte.
Peatükis 4 võrreldakse ning selgitatakse funktsionaalsuste ja lähenemiste realiseerimist kahes erinevas rakenduses.

1.1 Eesmärk, probleem ja metoodika

Lõputöö peamiseks eesmärgiks on arendada varasemalt arendatud rakendusele alternatiiv, kasutades arendustegevuseks teisi tehnoloogiaid ning raamistikke ning jõuda selguseni, kumma rakenduse arendustegevus oli efektiivsem, kiirem ning nõudis arendajalt vähem erinevate kitsendustega ja mitte ettenähtavate probleemidega tegelemist. Lisaks järelduste alusel koostada arendamistegevuse lõpus võrdlus kahes rakenduses kasutatud tehnoloogiatel samas funktsionaalsuse arendamise eelised ja puudused, samuti analüüsida erinevaid arenduspõhimõtteid.

Peamiseks lõputöös käsitletavaks probleemiks on see, et on olemas väga mitmeid erinevaid back end poole arendamise raamistikke, kuid keeruline on valida sobiv just enda arendusse.

Püstitatud eesmärkide saavutamiseks arendas autor kaks identse funktsionaalsusega rakendust ning tegi arendustegevuse käigus märkmeid ning hilisemalt tegi märkmete alusel rakenduste realiseerimise võrdluse ning analüüsi.

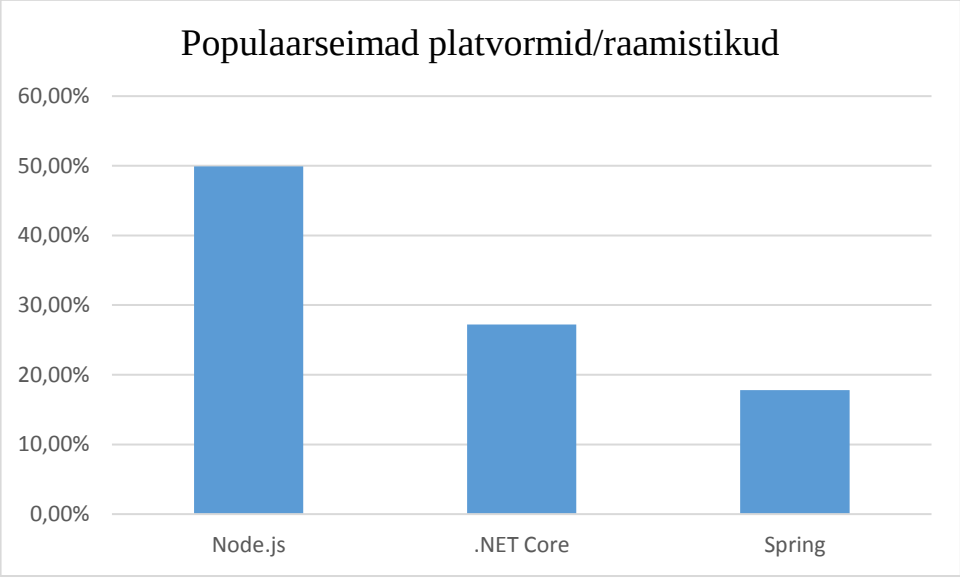
2 Kasutatavad raamistikud ja arhitektuur

Selles peatükis kirjeldatakse peamisi arenduses kasutatud raamistikke, tehnoloogiaid ja rakenduste arendamisel loodud rakenduskihte.

2.1 Raamistike valimine

Arendustegevuse lihtsustamiseks ning efektiivsuse tõusuks kasutatakse ära erinevaid raamistikke, kuhu on ära lahendatud teatud korduvfunktsionaalsus arendustegevusel ning mida arendajatel on võimalik enda tarkvaras ära kasutada. Põhjuseks, miks võeti just lõputöö raames arendatud rakendustes kasutusele Spring ning Node.js raamistikud, oli eelkõige nende populaarsus ning fakt, et tänased tööpakkujad nende raamistike/tehnoloogiate tundmist oma töötajatelt eeldavad.

Järgnevalt tuuakse välja statistika *Stack Overflow* veebiportaali küsitlustulemustest. *Stack Overflow* on veebipõhine kommuun arendajatele, mida külastab kuu jooksul üle 50 miljoni arendaja [8]. Küsitluses osales kokku peaaegu sada tuhat arendajat. Joonis 1 esitab kokkuvõtte elukutseliste arendajate vastuste kohta antud küsitluse punktis, kus taheti teada, milliseid platvorme arendamisel nemad peamiselt kasutasid. Joonisel on esimesed kolm kõige populaarsemat raamistikku, mida arendajad kasutasid rakenduste back end poole arenduseks [9].

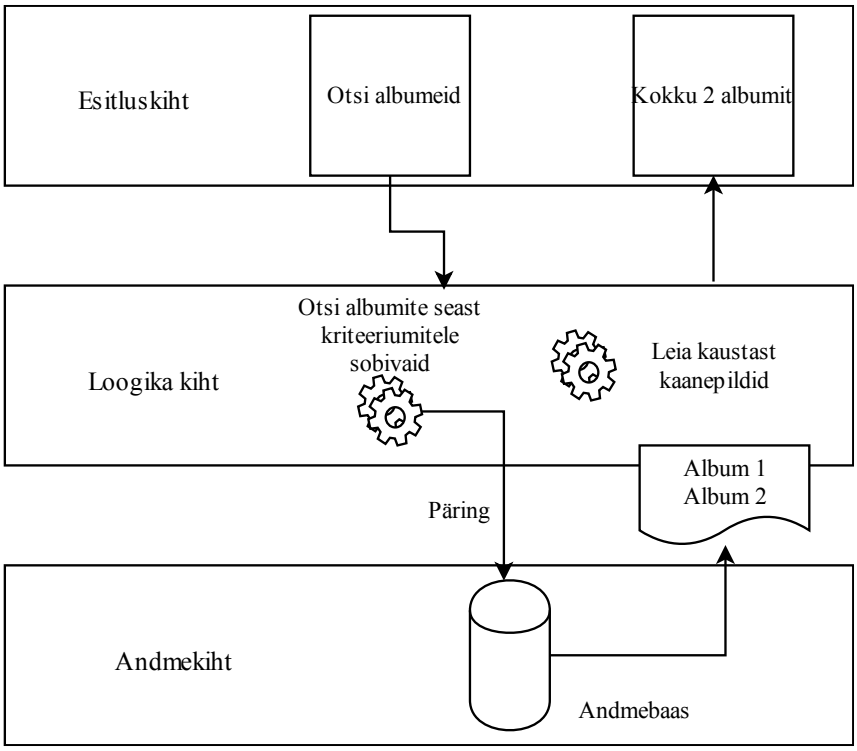


Joonis 1. *Stack Overflow* küsitluse tulemus.

Kuna autor omab minimaalset kokkupuudet kokkupuudet C# keelega, mis on põhiline programmeerimiskeel .NET Core platvormil arendamiseks, siis otsustati realiseerida rakenduse üks versioon Node.js platvormil kasutades JavaScript programmeerimiskeelt ning teine rakendus realiseeriti Spring platvormil kasutades Java programmeerimiskeelt. Mõlemad raamistikud on arendajate seas väga populaarsed ning see tagab eelkõige vastava platvormi pideva uuendamise uute funktsionaalsustega ja parandustega.

2.2 Mitmekihiline arhitektuur ehk *n-tier architecture*

Loodud rakenduste puhul hõlmas arendustegevus kolme kihi arendamist ja suhtlema panemist. Rakenduse esimest kihti nimetatakse esitluse kihiks ning ka kasutajaliideseks (i.k *presentation tier*) ja selle kihti peamiseks ülesandeks on andmete esitamine kasutajale arusaadaval kujul. Rakenduse teine kiht on loogika kiht (i.k *logic tier*), mis töötleb kasutaja päringuid, teostab kontrolli kasutaja päringute üle ning on vahekihiks esimese ja kolmanda kihi vahel. Rakenduse kolmas kiht on andmekiht (i.k *data tier*), mille eesmärgiks on andmete ja failide salvestamine ning nende pärimine, lisamine, kustutamine ja muutmine [10]. Joonis 2 esitab näite rakenduse kihtide toimimisest albumite otsingu näitel.



Joonis 2. Mitmekihilise arhitektuuriga veebirakenduse toimimine albumi otsimisel näitel.

2.3 JSON

JSON (*Javascript Object Notation)* on liiasuseta andmevahetusformaat, mis baseerub JavaScripti programmeerimiskeele alamosadel [11]. Kuna rakenduse Node.js platvormil realiseeritud variandi implementatsioonis kasutatakse JavaScripti nii front endis kui ka back-endis, siis on andmevahetus Node.js platvormil lihtne ja kiire. Samuti on JSON andmevahetusformaat väga kergesti kasutatav Java programmeerimiskeelega, kus on mitmeid teeke JSON'i teisendamiseks Java objektideks ning vastupidi. Nendel põhjustel korraldati andmevahetus JSON struktuuril.

Spring kasutab vaikimisi JSON'i teisendamiseks Jackson teeki, mille abil on võimalik klassikaline Java objekti teha JSON formaati ning ka vastupidi. Näiteks on kasutajaliidesesse vaja saata ressurss, mis Java objektina näeks välja joonisel 3 esitatud kujul.

```
class User {
       private String firstName = "Eesnimi";
       private String lastName = "Perenimi";
       private Integer age = 25;
       // getters, setters
}
```

Joonis 3. Java klass.

Selline Java objekt teisendatakse automaatselt JSON struktuuriks, mida on võimalik kergesti brauseripoolses JavaScriptis itereerida ning andmetega manipuleerida. Joonis 4 esitab JSON struktuuris andmete näite.

```
{
    "firstName": "Eesnimi",
    "lastName": "Perenimi",
    "age": 25
}
```

Joonis 4. JSON struktuuris andmed.

2.4 REST API

REST (*Representation State Transfer*) on veebiteenuste arhitektuuristiil, mille abil on võimalik arendada teenuseid erinevate rakenduste toimimiseks vajalike ressursside

pärimiseks ja lisamiseks. REST API populaarsus seisneb selles, et REST teenused ehitatakse üles olemasolevatele süsteemidele ning see kasutab ära standardseid HTTP (*HyperText Transfer Protocol*) meetodeid, millest populaarsemad on: GET, POST, PUT, DELELTE, andmevahetuseks ning ei nõua uute standardite või tehnoloogiate loomist. REST API abil võib andmevahetus toimida erinevates andmeformaatides: XML, JSON või CSV [12]. Implementeeritud rakendustes kasutatakse eranditult JSON formaati.

Valdav enamus veebiteenustest realiseeritakse just REST teenuste näol ning oluliselt vähem teise sarnase, SOAP teenuste näol. REST eelis on eelkõige veebirakenduste seisukohalt see, et suudab andmevahetusformaadina kasutada nii XML, JSON kui ka CSV andmeformaati, kuid SOAP võimaldab realiseerida ainult XML formaadis andmevahetust [12].

REST API puhul kasutatakse URL'e, millele igale määratakse ära kindel HTTP meetod ning ressurss, mida otspunktist päritakse või lisatakse. Mitmele erinevalt HTTP meetodile võib vastata täpselt sama URL [12].

2.5 jQuery ja AJAX

JQuery on populaarne kliendipoolne JavaScripti teek, mis lihtsustab arendajatel HTML's loodud kasutajaliidese DOM (*Document Object Model*) elementide programmeerimist, lisamist ja manipuleerimist. JQuery on tasuta ning avatud lähtekoodiga tarkvara, mida haldab suur kogukond arendajaid [13]. Joonis 5 esitab näite, kuidas lihtsustab HTML elemendile ligipääsemist ja väärtuse kättesaamist JQuery kasutamine võrreldes tavalise JavaScriptiga.

```
//JavaScript
document.getElementById("element").value;
//jQuery
$("#element").val();
```

Joonis 5. Javascripti ja jQuery võrdlus elemendile ligipääsemiseks.

Joonis 5 järelduseks on see, et tuleb kirjutada vähem koodi, et saada ligi HTML elemendile ja küsida tema väärtus ning ka kõikides teistest aspektides on jQuery eelis

eelkõige väiksem koodihulk ning läbi selle parem ülevaade tehtule ning on lihtsam avastada arenduses tekkinud vigasid.

JQuery üks suur eelis on teha AJAX (*Asynchronous Javascript and XML*) päringuid väiksema koodihulgaga ning efektiivsemalt kontrollida päringu erinevaid etappe. AJAX on tehnoloogia, mille abil on võimalik veebilehe sisu uuendada lehte taas laadimata, pärida ning vastu võtta informatsiooni serverist pärast seda, kui veebileht on kasutajale laetud ja kuvatud ning samuti võimaldab asünkroonselt saata andmeid taustal serverile töötlemiseks [14]. Arendatud rakendused kasutavad kõikide andmete pärimiseks ja saatmiseks asünkroonseid AJAX päringuid.

Tänaseks on asünkroonsetest päringutest saanud veebiarenduses standard, sest muudab kasutajale kasutuskogemuse paremaks ning võimaldab veebilehe kasutamist jube enne seda, kui veebileht on vajalike ressursside kättesaamise lõpetanud. Varasemad veebirakendused pärisid enne lehe kuvamist vajalikud ressursid ära ning lisasid ressursid serveripoolses programmis HTML'le ning siis saatsid juba täidetud HTML lehe veebilehitsejale kuvamiseks. Selline lähenemine tähendab aga seda, et pidevalt tuleb veebilehte täismahus uuendada ning ka neid osasid, mis veebilehel tegelikult kasutaja uue päringu peale ei muutunud [14].

2.6 Spring raamistik

Spring on tasuta ning avatud lähtekoodiga raamistik, mille abil on võimalik efektiivsemalt ja lihtsamalt kirjutada Java programmeerimiskeeles rakendusi. Springi tuumikfunktsionaalsuseid saab kasutada, et arendada ükskõik mis eesmärgiga Java rakendusi, kuid samuti sisaldab raamistik endas laiendusi, mis on ehitatud Java EE (*Enterprise Edition*) platvormi peale ning lihtsustavad just veebirakenduste ehitamisprotsessi [15].

Veebirakenduste ehitamise seisukohalt on Springi suurim eelis erinevate komponentide arv ning nende kasutamise lihtsus. Springi kasutades on võimalik rakenduse ressursse arendades kasutada tavalisi Java objekte (i.k *POJO*), koos väljade *getterite* ja *setteritega* ning selline lahendus on oma olemuselt lihtsakoeline, kuid samas võimekas. Spring võimaldab Java objektideks automaatselt teha päringuna tulevat JSON struktuuri või

andmebaasist päritavat ressurssi, kus Spring on automaatselt võimeline teisendama andmebaasi veergude väärtused Java objektide väljade väärtusteks.

Springil on oma lahendus kasutaja autoriseerimiseks ning kasutajate rollipõhisteks gruppideks jagamiseks, mis aitab lihtsustada ärireeglite jõustamist, mille kohaselt on mõnele ressursile juurdepääs lubatud teatud rolliga kasutajal või üldsegi keelata lehtedele ligisaamist [15]. Joonis 6 kirjeldab REST API meetodi loomist Java programmeerimiskeeles.

```
@Unauthenticated
@RequestMapping(value = "/getEventTypes", method = GET)
public ResponseEntity getEventTypes() {
      // respond with a resource
}
```

Joonis 6. Spring raamistikul programmeeritud REST API otspunkt.

Esimene REST API kontrollermeetodile lisatav annotatsioon näites määrab ära selle, et päringuid saab teha ka kasutaja, kes ole ennast autentinud. Kui eemaldada esimene annotatsioon, siis päringule tuleb vastuseks ressurss ainult siis, kui kasutaja on rakenduses ennast sisse loginud.

2.7 Node.js platvorm

Node.js on avatud lähtekoodiga JavaScripti platvorm, mida kasutatakse kiirete ning skaleeruvate serveri- ning veebirakenduste arendamiseks. Node.js rakendused baseeruvad Google'i JavaScripti jooksutamise mootori peal. Node.js'i, erandina teistele populaarsetele teekidele nagu näiteks AngularJS, kasutatakse serveripoolse JavaScripti loomiseks, mitte kliendipoolse [16], [17].

Node.js platvorm toimib sarnaselt varem kirjeldatud AJAX põhimõttele - ehitatud üles asünkroonsete sündmuste haldamise põhimõttele. Üldjoontes tähendab see seda, et Node.js rakenduse käivitades määratakse ära domeen/URL, millele päringuid tehes rakendus peab vastama, kuid juhul, kui päringuid rakendusele ei ole, siis lülitab rakendus ennast unerežiimile. Erandina veebirakendustele, mis on ehitatud üles samaaegseid

päringud haldama eraldi operatsioonisüsteemi lõimedes, kus on pidevalt lõimed koormatud, isegi siis, kui parasjagu päringuid, mida töödelda, pole [16], [17].

2.8 Kokkuvõte

2. peatüki eesmärk oli lugejale kirjeldada arendustes kasutatavaid suuremaid raamistikke ja tehnoloogiaid, nende valikupõhjuseid ning tuua praktilisi näiteid jooniste näol. Antud ülevaade kirjeldab ära olulisemad teadmised raamistike kohta ning võimaldab hilisemas selgituse ja võrdluse etapis lihtsamalt mõista koodinäiteid ja raamistike sisulisi omadusi. Edasises tegevuses kirjeldatakse ära rakenduse eesmärgid, kasutusvõimalused ja lühiülevaade, et viia lugeja kurssi arenduse skoobis oleva funktsionaalsusega.

3 Rakenduse ülevaade

Antud peatükis antakse lühiülevaade rakenduste kasutusjuhtudest, REST API otspunktidest ning rakendustest kasutatud andmebaasitabelitest.

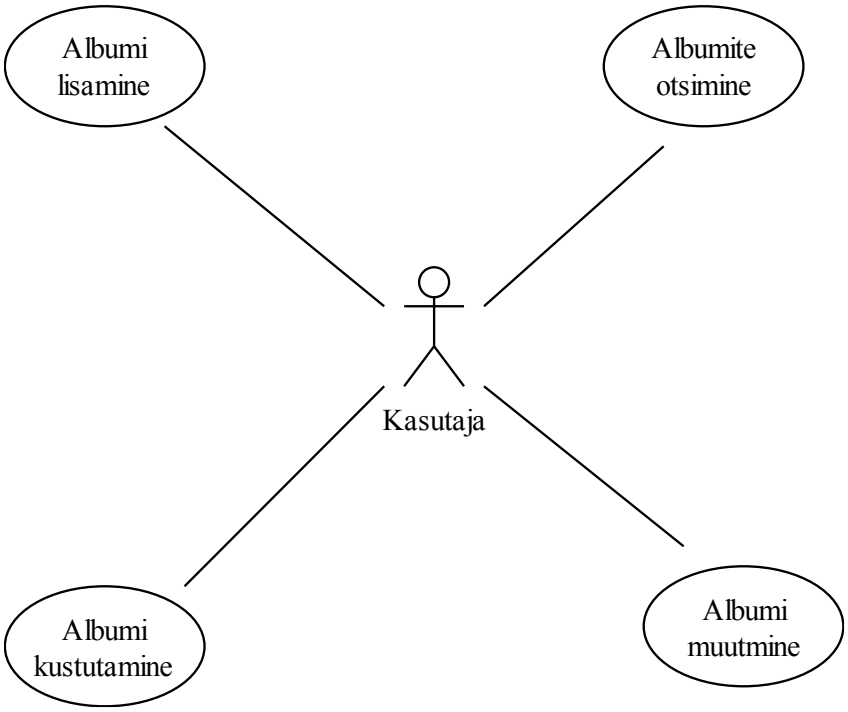
3.1 Rakenduse kirjeldus

Lõputöö raames realiseeritud rakendused võimaldavad lisada, muuta, kustutada ning otsida pildialbumeid. Rakenduses hoitakse iga pildialbumi kohta järgnevat informatsiooni: albumi nimi, albumi tüüp, albumi kirjeldus, toimumise kuupäevad, URL asukoht ning kaanepilt. Rakenduse eesmärk oli luua tsentraalne albumite hoidmise funktsionaalsus suuremasse veebirakendusse, kus hoitakse ettevõtte erinevatel üritustel tehtud albumite lühiinformatsiooni. Kõik albumi pildifailid hoiustatakse *Google Photos* pilveteenuses ning realiseeritud rakendus käitub nagu vahelüli kasutajaliidese ning *Google Photos* vahel. See tähendab seda, et konkreetselt ühtegi pildifaili, peale albumi kaanepildi, rakenduses või serveris ei hoita vaid albumi lisamisel määratakse ära URL, kus asub konkreetne pildialbum ning iga albumi peale vajutades suunatakse kasutaja albumi asukohale pilveteenuses.

Arenduse tulemusena valmis ettevõtte siseveebi rakendusse uus moodul, mis võimaldab nüüdsest kõikidel ettevõtte töötajatel lihtsamalt leida üles ettevõtte üritustel tehtud albumeid ning albumeid kas lisada, muuta või kustutada. Kõiki lisatud albumeid on võimalik otsida albumi tüübi ja toimumisvahemiku järgi. Kirjeldatud moodul lihtsustab nüüdsest infovahetust, sest kõik albumid on ligipääsetavad ning nähtavad ühest kohast.

3.2 Rakenduse kasutusjuhud

Joonis 7 esitab rakenduse kasutaja võimalikud kasutusjuhud realiseeritud rakendustes.



Joonis 7. Rakendsute kasutusjuhtude diagramm.

Kasutusjuhtude kirjeldused

Kasutusjuht: Albumi lisamine.

Tegutseja: Kasutaja.

Eeltingimused: Kasutaja on avanud rakenduse veebilehitsejas.

Järeltingimused: Uus album on lisatud albumite nimekirja.

Põhistsenaarium: Kasutaja läheb albumi lisamise lehele, kus täidab ära järgnevad kohustuslikud väljad - albumi tüüp, albumi nimi, albumi algus- ja lõppkuupäev, albumi asukoha URL ning albumi kaanepilt.

Kasutusjuht: Albumi otsimine.

Tegutseja: Kasutaja.

Eeltingimused: Kasutaja on avanud rakenduse veebilehitsejas.

Järeltingimused: Kasutaja näeb vastavalt otsinguparameetritele albumite nimekirja.

Põhistsenaarium: Kasutaja saab sisestada otsinguväljadesse albumi tüübi ning ürituse toimimiskuupäevade vahemiku ning vastavalt nendele sisenditele toimub otsing. Samuti võib kasutaja jätta kõik otsingulahtrid tühjaks ja otsida kõiki albumeid.

Kasutusjuht: Albumi kustutamine.

Tegutseja: Kasutaja.

Eeltingimused: Kasutaja on avanud rakenduse veebilehitsejas.

Järeltingimused: Soovitud album on albumite nimekirjast kustutatud.

Põhistsenaarium: Kasutaja valib albumi, mida soovib kustutada ning ta suunatakse albumi muutmise lehele, kus on tal võimalik albumit kustutada. Albumi kustutamisel küsitakse kasutajalt kinnitust, et olla kindel, et kasutaja ei vajutanud nuppu kogemata ning album kustutatakse kinnitamisel.

Kasutusjuht: Albumi muutmine.

Tegutseja: Kasutaja.

Eeltingimused: Kasutaja on avanud rakenduse veebilehitsejas.

Järeltingimused: Soovitud albumi andmed on muudetud.

Põhistsenaarium: Kasutaja valib albumi, mida soovib muuta ning ta suunatakse albumi muutmise lehele, kus on tal võimalik album muuta. Albumit muutes tuleb järgida reeglit, kus kõik järgnevad väljad oleksid täidetud - albumi tüüp, albumi nimi, albumi algus- ja lõppkuupäev, albumi asukoha URL ning albumi kaanepilt.

3.3 REST API otspunktid

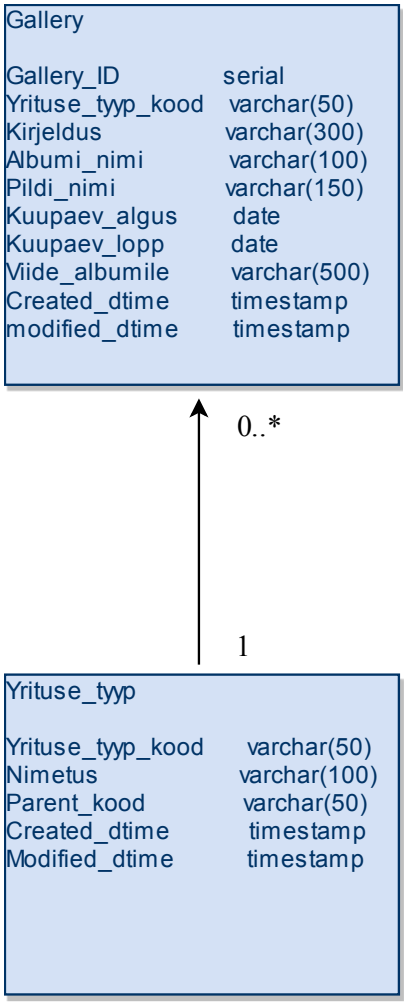
Järgnevalt kirjeldatakse ära kõik rakenduse REST API otspunktid. Andmed tagastatakse REST API kliendile üldjuhul JSON formaadis andmetena ning REST API on universaalne, mis tähendab, et neid otspunkte kasutavad vaadeldavad rakendused kui ka on võimalik teistel rakendustel kasutada neid REST API otspunkte, kui peaks vaja minema. Näiteks, kui tehakse mobiilirakendus, kuhu vastavaid andmeid on tarvis, siis on võimalik REST API käest soovituid andmeid ka saada ja ressursse lisada. Tabel 1 esitab lühikokkuvõte programmeeritud REST API otspunktidest.

Tabel 1. REST API otspunktid.

URL **HTTP MEETOD**	**Kirjeldus**	**Parameetrid**
rest/events/getEventsType GET	Tagastab kõik ürituse ülemtüübid.	-
rest/events/getSubEventTypes/:eventTypeCode GET	Tagastab kõik kindla ürituse ülemtüübi alamtüübid.	:eventTypeCode[sõne] - ülemtüübi identifikaator.
rest/events/addAlbum POST	Lisab uue albumi.	*form-data* kodeeringus uue albumi andmed.
rest/events/downloadImage/:id GET	Tagastab albumi kaanepildi.	:id[number] - albumi identifikaator.
rest/events/getRecentlyAddedAlbums GET	Tagastab viimati lisatud albumid.	-
rest/events/getAlbum/:id GET	Tagastab kindla albumi andmed.	:id[number]- albumi identifikaator.
rest/events/editAlbum POST	Muudab olemasolevat albumit.	*form-data* kodeeringus albumi uued andmed.
rest/events/delete/:id/:xmin DELETE	Kustutab albumi.	*:id[number]* - albumi identifikaator :xmin[number]- albumi versiooni identifikaator andmebaasisüsteemis.
rest/events/searchAlbums POST	Tagastab otsingule vastavad albumid.	JSON formaadis otsinguandmed.

3.4 Andmebaas

Varasemalt ettevõtte juures arendatud rakenduse puhul kasutas rakendus nelja baastabelit, milleks olid allpool joonisel nähtavad tabelid ning nendega oli seotud valdkondade tabel ning isikute tabel. Kuna esialgses ega teises rakenduses ei olnud rakenduse skoobis autentimise funktsionaalsuse arendamine, siis eeldatakse uues rakenduses, et kasutaja on kogu aeg autenditud ja isikute tabelit luua polnud vaja ning valdkondade info on yrituse_tyyp tabeliga liidetud. Joonis 8 esitab Node.js platvormil arendatud rakenduses kasutatavaid andmebaasitabelid ja nende omavahelisi seoseid. Lisaks loodi andmebaasisüsteemis 2 vaadet ja 4 funktsiooni ning üks andmebaasikasutaja.



Joonis 8. Andmebaasi tabelite diagramm.

4 Rakenduste võrdlus

Siin peatükis võrreldakse ning selgitatakse erinevatel arendusplatvormidel sama funktsionaalsuse arendamise eeliseid ja puuduseid ning erinevaid arenduspõhimõtteid. Võrdlusesse on võetud kõige olulisemad funktsionaalsused, mis on veebiarenduses üldjuhul triviaalsed ülesanded.

4.1 REST API loomine

Spring raamistikul REST API loomisel tuleb luua klass, kuhu tuleb lisada vajalikud annotatsioonid, et raamistik teaks, et tegemist on REST kontrolleri klassiga, mis on võimeline ressursse tagastama ja vastu võtma. Joonis 9 esitab Java klassi annotatsioone, mida on vaja REST API toimimiseks lisada.

```
RestController
@RequestMapping("/rest/events")
```

Joonis 9. Spring raamistikul REST API kontrollerklassi loomine.

Ülemine annotatsioon määrab ära, et loodud klass on REST kontroller ning alumine annotatsioon määrab ära suhtelise URL tee, mida kontroller haldab.
Selleks, et luua Java programmikoodis meetodit, mille eesmärk oleks REST API realiseerimine, tuleb ära määrata kindlasti järgmine annotatsioon meetodi kohale. Joonis 10 esitab näite Spring raamistikul REST API meetodi defineerimiseks.

```
@RequestMapping(value = "/newAlbum", method = POST)
    public ResponseEntity addGallery(GalleryDataJson galleryDataJson)
{
       // add resource
    }
```

Joonis 10. Spring raamistikul REST kontrollermeetodi loomine.

Annotatsioon meetodi kohal määrab ära suhtelise URL tee Java klassi sees ning HTTP meetodi, mida antud meetod kontrollib. Vastava annotatsiooniga klassi ees ja meetodi ees kujuneb välja terviklik URL sellele otspunktile, mis joonise 9 ja 10 põhjal oleks järgmine:

/rest/events/newAlbum, kus URL kaks esimest poolt määrab ära klassi üldine URL ja kolmanda poole meetodi annotatsiooni kirjutatud *value* väärtus.

Ressurss, mida ülaltoodud näites lisatakse ning mis tuleb POST päringuga on kasutajaliidesest saates *form-data* kodeeringus, kuid Spring suudab automaatselt antud kodeeringu teisendada Java objektiks ning ka päringu sisuks oleva pildifaili teisendada automaatselt *MultipartFile* objektiks, mis salvestab faili automaatselt ajutiselt vahemällu või kõvakettale ning mida on arendajal lihtne püsivalt salvestada vajadusel failisüsteemi või andmebaasi. Selline failide ja ka muude andmeformaatide automaatne teisendamine Java objekti väljade väärusteks on Springi raamistikul suureks eeliseks, sest võimaldab efektiivselt päringu sisuks tulnud väärtustega manipuleerida ning edasi saata andmesalvestuskihti.

Node.js platvormil REST API arendamiseks kasutati Express JS vahetarkvara, mis aitab arendajatel luua lühemalt ja lihtsamalt REST API URL skeemi ning töötleb päringuna REST API'sse tulevaid JSON andmeid, et arendaja saaks päringu kohale jõudes koheselt JSON objektiga soovitud manipulatsioone teha. Küll aga ei suuda Express JS *form-data* kodeeringus tulevaid andmeid vastu võtta ning seega võeti arenduses kasutusse vahetarkvara nimega multer, mis on spetsiaalselt loodud antud kodeeringus tulevate päringute sisu töötlemiseks ja konverteerimiseks JSON formaati.

Sarnaselt Spring raamistikuga, tuleb ka Express JS vahetarkvaras määrata rakenduse käivitamisfailis suhteline URL tee, kus asub REST kontroller ning vastava failis sees veel alamtee kindla ressurssiga manipuleerimise funktsiooni seadistamiseks. Joonis 11 kirjeldab Node.js platvormil vajalikke ridu, et luua REST API teenus.

```
var restController = require('./routes/rest');

app.use('/rest/events/', restController);

router.post('/addAlbum', uploadModule.upload.single('image'),
function (req, res) {
   // add a resource
});
```

Joonis 11. Node.js's REST API konfigureerimine.

Joonise 11 põhjal näeb välja REST API otspunktide seadistamine Node.js platvormil, kus esimene rida tuleb lisada rakenduse käivitamisfaili ning see impordib vastavat faili, kus otspunktid on, käivitusfaili. Teine rida määrab ära REST API kontrolleri alguse URL'i ning kolmas rida on juba konkreetne funktsioon, mis asub REST teenuse failis ning kuulab URL'i, mille väärtuseks on /rest/events/addAlbum. Parameetrid, mida Express JS kontrolleri funktsioon omab on: **req** ja **res**, kus **req** tähistab päringut ning sinna salvestatakse päringu andmed JSON kujul ning **res** on objekt, mis on funktsiooni vastuseks päringule. Selleks võib olla mingi kindel ressurss, kollektsioon ressurssidest või hoopiski kindlad veateated.

REST API tööle saamiseks on lihtsam seadistada Node.js platvormil Express JS vahetarkvara, sest tuleb kirjutada vähem koodi ning samuti on koodistruktuur autori jaoks arusaadavam. Juhul, kui REST API ei eelda keerukamaid kodeeringutüüpe, näiteks *form-data*, siis autor leiab, et Node.js kasutamine REST API funktsionaalsuse lahendamiseks on piisavalt lihtne ning arusaadav, et eelistada antud tehnoloogiat funktsionaalsuse implementeerimiseks. Kui REST API funktsionaalsus eeldab aga ka failide vastuvõtmist või üleüldiselt keerulisemaid andmestruktuure, kui seda on JSON, siis selle koha pealt on Springil hea lahendus ning lihtsustab märkimisväärselt näiteks failide üleslaadimise haldamist.

4.2 Kasutajaliides

Spring raamistikule ehitatud rakenduses kasutati kasutajaliidese ehitamiseks klassikalist HTML5'te, mille süntaksi järgi on korrektne luua HTML div element klassiga "proov" ja *id*'ga "test" nagu seda on tehtud joonisel 12.

```
<div id=“test” class=“proov”></div>
```

Joonis 12. HTML keeles *div* elemendi loomine.

Rakenduse hilisemas versioonis soovis autor klassikalise HTML'i asemel kasutada mõnda HTML mallimootorit. Mallimootor peaks arendajal aitama koostada puhtamat ja selgemat HTMLi veebilehe kirjeldust, mille hilisem hooldamine peaks olema lihtsam ning aitama luua komponente ja võimaluse neid taaskasutada.

Rakenduse arendamisel kasutati Node.js platvormi HTML mallide loomise teeki Jade. Jade'i tööpõhimõte on lihtne - arendaja kirjeldab vastavalt Jade'i süntaksile ära veebilehel vajaminevad elemendid ning Jade teisendab malli klassikalisse HTML'i, mida kõik veebibrauserid on võimelised tõlgendama ning kasutajale kuvama. Jade'i süntaksi järgi oleks korrektne luua HTML div element klassiga "proov" ja *id*'ga "test" nagu seda on tehtud joonisel 13.

```
#test.proov
```

Joonis 13. Jade raamistikul *div* elemendi loomine.

Sellise süntaksi järgi genereeritakse veebibrauserile identne HTML, mis on nähtav joonisel 12.

Samuti oli võimalik Jade'i kasutades koostada realiseeritud rakenduses albumi lisamise ja muutmise vormi komponent, Komponent sisaldab endas kõiki väljasid, mis tuleb täita albumit lisades või kustutades. Komponendi sisu tuleb mõlemasse faili importida ning nii on võimalik teha muudatuste korral korrektuurid ainult komponenti endasse ning albumi muutmise ja lisamise lehele tuleb automaatselt uuendatud vorm. Klassikalise HTML'i korral on aga samad vormiväljad eraldi HTML dokumentides ning muudatuste tegemiseks tuleb muuta mõlemat faili.

Rakenduses Jade'i kasutamine tegi oluliselt lihtsamaks HTML'i tegemise ning võimaldas paremini avastada vigu, mis oli veebilehe kirjeldamisel tehtud, sest Jade'i süntaksi järgi ei ole HTML elementidel sulguvaid *tag*'e ning seega on lihtsam hallata elemente, millel on palju alamelemente. Samas aga on Jade's väga olulised taanded, just nendega hallatakse elementide kuuluvust teatud ülemelementi ning iga taandeviga tähendab, et element ei ole enam osa soovitud ülemelemendist. Joonis 14 esitab näite Jade koodi treppimisest.

```
.row
    .col
        h3
```

Joonis 14. Elementide treppimine Jade süntaksi järgi.

Joonisel 14 on näide, kuidas käib elementide lisamine teiste elementide sisse Jade's. Näites on kolm elementi, millest *col* ja *h3* on *row* alamelemendid ning *h3* omakorda *col* alaelement. Sellise elementide hierarhia määrabki ära elementide ees olev taane.

Antud rakenduses Jade'i kasutamine oli otstarbekas, sest vähendas oluliselt koodi mahtu, mida oli vaja, et kirjeldada ära rakenduses kasutatavad veebilehe elemendid ning võimaldas luua ühe vormi komponendi, mida oli võimalik ära kasutada kahel erineval lehel. Mallide loomise keeled on üldjuhul otstarbekad juurutamiseks kasutajaliidestes, kus soovitatakse mitmel lehel kasutada sarnaseid elementide kombinatsioone või oleks näiteks vaja luua dünaamilist HTML'i, mida peamiselt kasutavad populaarse MVC (*Model-View-Controller*) arhitektuuriga rakendused.

4.3 Andmebaasisüsteemiga töötamine

Spring raamistikul arendatud rakendus kasutab välist teeki nimega MyBatis 3, et ühilduda ja teha päringuid Oracle andmebaasisüsteemi. MyBatis kasutab andmebaasiga ühildumiseks ära Java arendajate poolt kirjutatud JDBC (*Java Database Connectivity*) API't.

MyBatise suureks eeliseks on andmebaasi ridade teisendamine andmete pärimisel Java objektideks ning samuti Java objektide lisamine andmebaasitabelitesse. Samuti võimaldab MyBatis lisaks klassikalistele sõne, numbri või kuupäevaformaatidele automaatselt toimida ka mitmesuguste teiste andmetüüpidega, nagu näiteks BLOB (*binary large object*). Antud projekti puhul oli see suur eelis ning lihtsustas autoril kaanepiltide hoidmise funktsionaalsuse arendamist, sest ei nõudnud erifunktsionaalsuse arendamist.

Samuti võimaldab MyBatis koostada dünaamilisi SQL lauseid, mis aitas lihtsustada SQL-päringute WHERE klausleid, kuhu tulid kitsendused otsinguväljadelt ning võisid olla samuti ka määramata. Joonis 15 esitab näite MyBatis dünaamilisest SQL lausest.

```
<select id="getGalleryEntries" resultType="Gallery">
     SELECT * FROM GALLERY
     <where>
       <if test="galleryData.parentCategory != null">
       YRITUSE_TYYP.PARENT_KOOD = #{galleryData.parentCategory}
       </if>
       <if test="galleryData.endDate != null">
          AND GALLERY.KUUPAEV_LOPP = #{galleryData.endDate}
        </if>
        <if test="galleryData.startDate != null">
          AND GALLERY.KUUPAEV_ALGUS >= #{galleryData.startDate}
        </if>
     </where>
</select>
```

Joonis 15. MyBatise dünaamiline SQL lause.

Selline dünaamiline SQL-päring suudab automaatselt WHERE klauslisse lisada kitsendusi juhul kui kasutaja albumeid otsides need määras. Juhul, kui ükski WHERE klauslis kasutuses olev objektiväli ei ole määratud, siis antud päringu lõppu WHERE klauslit ei lisata üldsegi ning kui peaks ainult olema määratud galerii lõppkuupäev (*galleryData.endDate*), siis MyBatis suudab automaatselt kaotada WHERE klausli algusest AND sõna, sest selline SQL lause muuljuhul ei oleks korrektne.

Node.js platvormil arendatud rakendus kasutab samuti välist teeki, mille nimeks on node-postgres. Antud teegi peamisteks võimalusteks on PostgreSQL andmebaasiga ühenduse loomine ning päringute tegemine.

Kirjeldatud teegil aga puuduvad sellised võimalused nagu on MyBatisel. SQL päringuid ei saa koostada dünaamiliselt vaid saab koostada kas tavalise JavaScripti sõnena või parameetritega päringuga. Parameetritega päringu peamine eesmärk on päringu klauslitesse lisada mingi kitsendus, mille täpne väärtus sõltub kasutaja valikust kasutajaliideses. Joonis 16 esitab node-postgres teegil parameetritega päringu süntaksit.

```
var query = 'SELECT YRITUSE_TYYP_KOOD, NIMETUS FROM
YRITUSE_TYYBID WHERE PARENT_KOOD = $1 ORDER BY NIMETUS;'
```

Joonis 16. Node-postgre' parameetritega SQL päring.

Nii on võimalik päringus ära määrata muutuja, mille sisu on võimalik enne andmebaasipäringut muuta. Joonis 17 esitab JavaScriptis kollektsioonile väärtuste lisamist.

```
var values = [];
values[0] = 'VARYR';
```

Joonis 17. JavaScriptis kollektsioonile väärtuste omistamine.

Järgnevalt tuleb SQL päringu toimimiseks väärtustada ära muutuja $1 ning selle jaoks tuleb salvestada soovitud muutuja/muutujad kollektsiooni, kus iga soovitud muutuja väärtus on kollektsioonis vastaval kohal ehk oletame, et päringus on kaks muutujat ($1 ja $2). Sellisel juhul peab olema nende väärtuste asukoht kollektsioonis vastav, vastasel juhul ei anna päring soovitud vastust. Joonis 18 esitab node-postgres teegiga parameetritega päringu tegemise.

```
database.client.query(query, values);
```

Joonis 18. JavaScriptis andmebaasipäringu tegemine.

Andmebaasi poole pöördudes tuleb kutsuda välja teegi funktsioon *query* ning sisenditeks anda SQL lause ning kollektsioon väärtustest, juhul kui tegemiste on parameetritega päringuga.

Spring raamistikul arendatud rakenduse eelis Node.js omast andmebaasiga töötamise ees on autori arvates märkimisväärne, sest MyBatis võimaldab esiteks konstrueerida keerulisetele päringutele lihtsama dünaamilise alternatiivi ning võimaldab arendajatel lisatööd tegemata automaatselt andmebaasiread teisendada Java objektideks ning vastupidi. Puuduseks on aga fakt, et MyBatise seadistamine on oluliselt keerulisem, sest tegemist on keerukama ülesehitusega tehnoloogiaga, kus SQL päringud asuvad XML failis ning Java liidesest neid peab välja kutsuma. Node-postgre eelis on tema lihtsus ning fakt, et ei ole vaja palju erinevaid meetodite väljakutseid või faile vaid andmebaasi saab päringu teha lihtsalt - kasutades *query* funktiooni node-postgre mooduli pealt.

Oluliselt lihtsam on arendata rakendus Java Spring raamistikul, kui rakenduse andmebaasisüsteem kasutab relatsioonilist või objekt relatsioonilist andmemudelit, sest

teostab automaatselt andmete muutmise Java objektiks ja tänu sellele võimaldab lihtsamalt manipuleerida andmetega. Node.js platvormi kasutamine on eelistatud, kui andmebaasisüsteem kasutab dokumentidel või võti-väärtus paaridel põhinevat andmemudelit. Seda selle pärast, et loetletud andmemudelid kasutavad andmete hoidmiseks kas puhast JSON formaati või mõne erisusega seda formaati ning JSON on aga JavaScripti programmeerimiskeele osa ja nii oleks andmevahetus lihtsam ja väliseid sõltuvusi vaja vähem.

4.4 Kaanepiltide hoiustamine

Mõlemas rakenduses lahendati pildifailide salvestamine erinevalt. Rakenduses, mis kasutab Spring raamistikku salvestati pildifailid andmebaasis. Node.js platvormil implementeeritud rakenduses salvestati pildifailid failisüsteemi.

Springi rakendus suudab hästi vastu võtta *form-data* kodeeringus tulevaid päringuid ning samuti suudab hästi hallata päringu kehas olevaid faile. Andmebaasitabelis on veerg tüübiga BLOB (*binary large object*), mis on peaasjalikult mõeldud just failide hoidmiseks. Faile hoitakse seal binaarsõne kujul ning selle maksimaalsuuruseks on 2GB [18]. Nii teisendatakse päringu sisuks olnud failiobjekt baidijärjestikuks ning salvestatakse see järjestik andmebaasi. Samuti tuleb sellise lahenduse puhul salvestada andmebaasi kindlasti ka faililaiend, et hiljem oleks võimalik määrata faili andmetüüp ning see fail kliendile saata.

Node.js platvormil lahendati failide hoidmine ära failisüsteemi kasutades ning tehti failikaust, kuhu päringute sisuks olevad pildifailid üles laeti. Sellise lahenduse puhul tuli luua tekstifail, kus hoitakse ühte numbrit ning see number omistatakse uuele pildile. See välistab olukorra, kus kausta võiks tekkida mitu sama nimega pildifaili. Andmebaasis hoitakse nüüd aga pildi asemel pildi numbrit ja laiendit ning vajaduse korral on pildi kasutajaliidesesse saatmine lihtne, sest kasutakse ära failisüsteemi funktsiooni *sendFile(),* mis võimaldab faili otse kaustast asünkroonselt kliendini saata. Erinevuseks andmebaasis hoidmisele on eelkõige see, et andmebaasist pildifailide saatmiseks tuleb eelnevalt pärida andmebaasist salvestatud pildifaili baidijärjend ning see saata päringule vastuseks koos HTTP päisega, mis kirjeldaks ära, mis tüüpi on antud vastus.

Antud rakenduse näitel leiab autor, et andmebaasis failide hoidmine osutus siiski paremaks variandiks, sest:

- Andmete muutmisel võib tekkida olukord, kus muutmine tuleb pooleli jätta, sellise lahenduse puhul ei ole vaja erifunktsionaalsust, et ka juba lisatud pilti manipuleerida - kõik toimib andmebaasi poole pealt automaatselt. Selline automaatne muudatuse tagasivõtmine ei toimi aga andmebaasiväliste ressurssidega ja nõuab erifunktsionaalsust sellises olukorras käitumiseks.
- Andmebaasi tabelist rea kustutamisel kustub sealt ka selle reaga seotud pildifail. Failisüsteemis hoidmise puhul tuleb kustutada ära andmebaasis vastav rida ning siis kustutada veel eraldi ka kaustast pildifail.
- Piltide nimetamiseks failisüsteemis hoides tuli luua erifunktsionaalsus, mis suudaks tagada olukorra, et piltide nimed kaustas mitte kunagi ei kattu. Vastasel juhul kirjutatakse vana fail uuega üle ning rakenduses vajalik fail on kadunud. Andmebaasis pilte hoides sellist funktsionaalsust tegema ei pidanud, sest isegi kui pildinimed klapiksid, siis on ikkagi iga tabelirida seotud oma baidijärjestikuga, kuhu sisse on salvestatud pildi andmed ning nimede klappimine kuidagi failide ülekirjutamist ei saa tekitada.
- Andmebaasis olevatest andmetest varukoopiate tegemisel salvestatakse ka reaga seonduv pildifail, kuid failisüsteemis faile hoides tuleks luua erifunktsionaalsus failidest varukoopiate tegemiseks.

4.5 Andmebaasi arenduspraktikad

Node.js rakenduse variandis implementeeriti andmebaasis kaks otstarbekat andmebaaside arenduspraktikat - andmebaasi kapseldamine ning optimistlik lähenemine ridade uuendamisele.

Andmebaasi kapseldamine tähendab objekti andmestruktuuride ja nendega manipuleerimise protseduuride varjamist teiste objektide eest. Andmebaasis on võimalik sellist põhimõtet rakendada kasutades vaateid, hetktõmmiseid ja rutiine/funktsioone. Juurdepääs andmetele ainult rutiinide, hetktõmmiste ja vaadete kaudu võimaldab realiseerida andmebaasi turvasüsteemis ühe kihi [19].

Sellise kapseldamise eelduseks on andmebaasis kasutajate loomine, läbi mille rakendus töötaks andmebaasiga ning andmebaasi kasutajale tuleb anda õigusi ainult sellisel määral, mis on minimaalselt võimalik, et rakendus saaks toimida [19].

Antud rakenduses PostgreSQL andmebaasisüsteemi kasutades loodi selliseks kapseldamiseks soodumus ehk loodi kasutaja ning jagati kasutajale õigusi. Samuti loodi baastabelite põhjal vaated ning andmete muutmiseks, lisamiseks ja kustutamiseks loodi PostgreSQL funktsioonid. Tehtud kasutajale anti õigused pärida andmeid vaadetest ning käivitada funktsioone ning võeti kasutajalt ära mittevajaolevad õigused baastabelitega otse töötamiseks.

Kapseldamise realiseerimise põhjuseks arendatud rakenduses oli tutvuda kapseldamise põhimõttega ning realiseerida selline disainimuster andmebaasis, mis aitaks luua lisakihi andmete sügavuti ja mitmekesiseks kaitsmiseks.

Samuti arendati PostgreSQL andmebaasisüsteemis optimistlik lähenemine ridade samaaegsele muutmisele. Eelkõige on sellist funktsionaalsust vaja siis, kui võib tekkida olukord, et mitu kasutajat loevad samal ajal muutmiseks sama rida. Sellises olukorras peab muutja muudetud rea tagasikirjutamisel kontrollima, et ega rida ei ole vahepeal muutnud. Kui rida on vahepeal muudetud, siis andmemuudatust toimuda ei tohi [20].

Optimistlikku lähenemist PostgreSQL's aitab luua andmebaasisüsteemis eksisteeriv süsteemne ja nähtamatu veerg *xmin*. Iga tabeli rida on seotud *xmin* veeruga ning seal hoiab andmebaasisüsteem rea versiooni. Juhul, kui rida muudetakse, siis uus *xmin* väärtus selles reas on eelnevast väärtusest suurem [20]. Kasutades *xmin* veergu on võimalus realiseerida funktsionaalsus, mis takistaks kasutajal andmemuudatuse õnnestumist olukorras, kus muudatuste tegemise ajal on keegi juba selle rea andmeid muutnud.

Sellise lähenemise realiseerimiseks tuli kliendile, kes soovib kasutajaliidese kaudu andmeid muuta, saata koos andmetega kaasa ka nende andmetega andmebaasi tabeli reas seotud *xmin* veeru väärtus. Kui klient oli lõpetanud andmete muutmise ja soovis muudatust läbi viia, siis tegi rakendus AJAX päringu serveriprogrammi, kuhu saatis uued andmed ning talle varasemalt saadetud *xmin* väärtuse. Nende andmetega kutsuti välja andmebaasiskeemis olev funktsioon, mille abil realiseeriti optimistlik lähenemine

andmebaasi poolelt. Joonis 19 esitab PostgreSQL funktsiooni, millega realiseeriti kirjeldatud lähenemine.

```
CREATE FUNCTION update_entry (
       //Parameters
) RETURNS BOOLEAN LANGUAGE plpgsql SECURITY DEFINER AS $$
BEGIN
    UPDATE gallery
       SET ...
     WHERE gallery.gallery_id = update_entry.GALLERY_ID
       AND gallery.xmin  = update_entry.XMIN;
    RETURN FOUND;
```

Joonis 19. PostgreSQL funktsioon optimistliku lähenemise realiseerimiseks.

Andmemuudatuste läbiviimiseks on oluline, et *xmin* veeru väärtus muudetavas reas oleks sama, mis saatis klient andmemuudatuse tegemise päringus.

Optimistliku lähenemise realiseerimise põhjus rakenduses oli tutvuda selle lähenemisega ning luua rakendusse lisafunktsionaalsus, mis suudaks vältida ridade samaaegse muutmise ning võimaldaks andmemuudatuse õnnestumise või ebaõnnestumise olukorda fikseerida. Samuti sooviti antud lähenemine realiseerida just põhjusel, et tegemist on peamiselt veebirakendustes levinud arenduspraktikaga, kuidas hallata ridade samaaegset muutmist, kui konfliktid on harvad või pole nende tekkimine suur probleem [20].

4.6 Sõltuvuste haldamine

Rakenduste sõltuvuste haldamine Java rakenduses toimib läbi Gradle'i tarkvara, mis on peamiselt just arendatud Java ja muude programmeerimiskeelte, mis kasutavad komplieerimiseks Java Virtual Machine'i, rakenduste ehitamise tarkvaraks ja pakettide haldamiseks [21]. Antud projektis võrreldakse Gradle tarkvara pakettide haldamise seisukohalt.

Gradle kasutab pakettide rakendusse lisamiseks mõnda repositoorumit, milleks võib olla avalik repositoorium, nagu näiteks Maven Central, JCenter või Android rakenduste puhul Google Android repositoorium. Samas võib repositoorium olla ka ettevõttesisene, kuhu

on lisatud teatud funktsionaalsusega paketid, mida võiks ära kasutada mitmes erinevas projektis [21]. Joonis 20 esitab Gradle's sõltuvuste lisamist.

```
dependencies {
    testCompile 'junit:junit:4.12'
}
```

Joonis 20. Gradle's sõltuvuse lisamine.

Gradle'i põhiliseks failiks on *build.gradle* fail, kuhu sisse on arendajal võimalik loetleda kõik rakenduse toimimiseks vajalikud välised paketid ning Gradle suudab automaatselt leida repositoorumist ülesse vastava paketi ning see ka projekti importida. Pärast faili lisamist peab arendaja andma Gradle'le käsu sünkroniseerida projekt ning selle tulemusena on uus pakett projekti imporditud ja valmis kasutusele võtmiseks.

Node.js platvormil arendatud rakendus kasutab populaarset npm pakettide haldamise tööriista. Npm tööriist keskendub JavaScripti teekidele ning npm repositoorium võib sarnaselt Gradle'le olla kas avalik või privaatne, näiteks ettevõttesisene [22].

Npm'i põhiliseks failiks projektis on *package.json*, kuhu arendajal peab kõik projektis vajalikud teegid ning nende versioonid JSON kujul kirjutama. Joonis 21 esitab npm's sõltuvuste haldamist.

```
"dependencies": {
      "cookie-parser": "~1.4.3",
      "debug": "~2.6.9",
      "express": "~4.16.0"
}
```

Joonis 21. Npm's sõltuvuse lisamine.

Nii määratakse ära *package.json* failis vajalikud moodulid ning mooduli nimekirja lisades tuleb arendajal käivitada käsk **npm install,** mille järgi laetakse repositoorumist alla soovitud paketid ning lisatakse lähtekood projekti struktuuri ning arendajal on võimalik seda funktsionaalsust oma rakenduses ära kasutada.

Mõlemad kirjeldatud tehnoloogiad on arendatud rakendustes asendamatud tööriistad, et hallata projekti sõltuvusi. Mõlema tööriista eesmärk on automatiseerida sõltuvuste lisamist projekti, et kaotada ära manuaalne sõltuvuste lisamine, kus arendaja pidi Internetiavarustest leidma sobiva paketi ning selle siis alla tõmbama ning lähtekoodi manuaalselt projekti sobivasse kausta lisama. Projekti sõltuvuste koha pealt on mõlemad kirjeldatud tööriistad üsnagi lihtsad, et neid kasutada ning toimivad kogemuste põhjal probleemideta. Siiski leiab autor, et keerulisem oli kasutada Gradle't ning seda põhjusel, et Gradle'i funktsionaalsus on tegelikult palju suurem, kui puhtalt projekti sõltuvuste haldamine, ning see tekitab projekti mitmeid erinevaid näiliselt sama eemärgiga faile, kuid siiski on tegemist rakenduse erinevate tasemete sõltuvuste haldamisfailidega. Npm'i puhul oli asi aga lihtsam, sest terve projekti peale on üks fail ning kõik soovitud sõltuvused sisestatakse sinna. Tähele tuleb panna seda, et mõlemad tööriistad võimaldavad projekti konfigureerimise koha pealt veel mitmeid lisafunktsionaalsusi peale sõltuvuste haldamise, kuid antud juhul on käsitletud just seda aspekti.

4.7 Järeldused

Rakenduste arendamise käigus selgus fakt, et klassikalise mitmekihilise veebirakenduse arendamiseks sobivad ja on otstarbekad mõlemad kasutatud peamised platvormid - Spring ja Node.js. Siiski oleks olnud just antud rakenduse funktsionaalsust silmas pidades kõige paremaks lahenduseks kombinatsioon lähenemistest ja raamistikest, mida kasutati mõlemas rakenduses.

REST teenuse loomisel, andmebaasiga töötamisel ning kaanepiltide hoidmisel oli parem lahendus just Java programmeerimiskeeles Spring raamistikul arendatud rakendusel. Spring on võimekas raamistik andmebaasiga töötamiseks, juhul, kui tegemist on objekt relatsioonilise andmebaasisüsteemiga, sest suudab mugavalt rakendada ORM (*Object Relation Mapping*) põhimõtet andmebaasiga töötamiseks, mille käigus teisendab andmebaasitabelite veergude väärtused Java objekti väljade väärtusteks ja vastupidi. Samuti suudab Spring REST API loomisel märkimisväärselt lihtsustada failide üleslaadimist, sest ei nõua lisasõltuvusi või lisafunktsionaalsuse arendamist. Samuti võimaldas MyBatis raamistik koostada Java rakenduses mugavaid dünaamilisi SQL lauseid, mis aitas vähendada päringu keerukust võrreldes tavalise SQL lausega. Lisaks on veebirakenduses pildifailide hoidmine Spring raamistikku kasutades efektiivne ning

toimib probleemideta koostöös andmebaasisüsteemiga ning rakenduses pildifailide andmebaasi salvestamine osutus lõppkokkuvõttes paremaks lahenduseks.

Kasutajaliidese arendamise koha pealt on selge eelis Node.js platvormil arendatud rakendusel, sest kasutab mallide loomise keelt Jade, mis võimaldas arendatud rakenduses realiseerida komponentidel põhineva kasutajaliidese ning Jade' kasutamine vähendas koodihulka ja tegi kasutajaliidese arendamise efektiivsemaks ning kergemini muudetavaks ja hooldatavaks. Samuti osutus otstarbekaks ning põhjendatuks uudsete andmebaasipõhimõtete rakendamine võrreldes esialgse rakendusega, mille käigus esiteks kapseldati andmebaas, et luua lisakiht andmete kaitseks ning teiseks optimistlik lähenemine ridade uuendamiseks, et tagada andmemuudatuste korrektsus veebirakenduse töös.

Järgmisel korral veebirakendust arendades kasutab autor Java Spring raamistiku, realiseerib optimistliku lähenemise ja andmebaasi kapseldamise ning kasutab mallimootorit HTML kirjelduste loomiseks. Samuti eelistab failide hoiustamiseks andmebaasisüsteemi.

5 Kokkuvõte

Antud töö põhieesmärgiks oli realiseerida kahel erineval populaarsel arendusplatvormil ja mitmel eri lähenemisel kaks sama funktsionaalsusega rakendust ning tekitada arendustegevuse käigus võrdlusmoment raamistike ja erinevate lähenemiste vahel veebirakendustes üldjuhul triviaalsete funktsionaalsete nõuete arendamisel.

Töö tulemustest selgus, et mõlemad rakendused olid arenduse skoobis olevate kasutusjuhtude realiseerimiseks sobivad ning otstarbekad. Siiski saab öelda võrdlemiste tulemuste ning autori tähelepanekute alusel arendustegevusel, et Java Spring raamistikku tuleks juurutada arenduses, mis eeldab failide üleslaadimist ning kasutab objekt relatsioonilist andmebaasisüsteem. Node.js platvorm sobib rakenduse arendamiseks, mis kasutavad võti-väärtus paaridel põhinevat andmemudelit andmebaasisüsteemis. Samuti saab järelduseks välja tuua selle, et andmebaasi kasutatavates rakendustes tuleks implementeerida andmebaasis kapseldamise põhimõte, et tekitada andmete kaitsmiseks lisakiht. Samuti tuleks juurutada veebirakendustes optimistlik lähenemine ridade muutmisele, kus andmete samaaegne muutmine on vähetõenäoline või pole sellise olukorra tekkimine suureks probleemiks. See lähenemine on üks võimalus, kuidas vältida andmemuudatuste käigus soovimatu olukorra tekkimine. Oluliseks järelduseks on ka see, et juhul, kui kasutajaliidese arendamine eeldab samu või sarnaseid komponente mitmel lehel või hoopiski dünaamilist HTML loomist, siis tuleks kasutada HTML mallide loomise tehnoloogiaid, et vähendada kasutajaliidese arendamise ja täiendamise keerukust.

Töö käigus tehtud võrdlust on veel võimalik täiendada - spetsiifilisemalt võrrelda töös käsitletud aspekte või lisada juurde mõne triviaalse funktsionaalsuse loomise võrdlus, nagu näiteks kasutajate autentimine ja nende rollipõhisteks gruppideks jagamine.

Kasutatud kirjandus

[1] V.Hanson ja A.Tavast, "Arvutikasutaja sõnastik". [Võrgumaterjal]. Asukoht: http://www.keeleveeb.ee/dict/speciality/aks/dict.cgi?word=API&lang=en. Kasutatud: 7.5.2018.

[2] V.Hanson ja A.Tavast, "Arvutikasutaja sõnastik". [Võrgumaterjal]. Asukoht: http://www.keeleveeb.ee/dict/speciality/aks/dict.cgi?word=front+end&lang=en. Kasutatud: 7.5.2018.

[3] V.Hanson ja A.Tavast, "Arvutikasutaja sõnastik". [Võrgumaterjal]. Asukoht: http://www.keeleveeb.ee/dict/speciality/aks/dict.cgi?word=HTTP&lang=en. Kasutatud: 7.5.2018.

[4] V.Hanson ja A.Tavast, "Arvutikasutaja sõnastik". [Võrgumaterjal]. Asukoht: http://www.keeleveeb.ee/dict/speciality/aks/dict.cgi?word=URL&lang=en. Kasutatud: 7.5.2018.

[5] J. Robie, "What is the Document Object Model?". [Võrgumaterjal]. Asukoht: https://www.w3.org/TR/WD-DOM/introduction.html. Kasutatud: 7.5.2018.

[6] V.Hanson and A.Tavast, "Arvutikasutaja sõnastik". [Võrgumaterjal]. Asukoht: http://www.keeleveeb.ee/dict/speciality/aks/dict.cgi?word=HTML&lang=en. Kasutatud: 7.5.2018.

[7] V.Hanson and A.Tavast, "Arvutikasutaja sõnastik". [Võrgumaterjal]. Asukoht: http://www.keeleveeb.ee/dict/speciality/aks/dict.cgi?word=CSS&lang=en. Kasutatud: 7.5.2018.

[8] *Stack Overflow*. [Võrgumaterjal]. Asukoht: https://stackoverflow.com/. Kasutatud: 7.5.2018.

[9] "Developer survey results 2018," *Stack Overflow*. [Võrgumaterjal]. Asukoht: https://insights.stackoverflow.com/survey/2018/#overview. Kasutatud: 14.5.2018.

[10] "Three-Tier Architecture," *Microsoft Developer Network*. [Võrgumaterjal]. Asukoht: https://msdn.microsoft.com/en-us/library/ff648105.aspx. Kasutatud: 14.5.2018.

[11] *JSON*. [Võrgumaterjal]. Asukoht: http://json.org. Kasutatud: 7.5.2018.

[12] R.T.Fielding, "Representational State Transfer (REST)," "Architectural Styles and the Design of Network-based Software Architectures.", doktoritöö, Univ. of California, Irvine, 2000. [Võrgumaterjal]. Asukoht: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. Kasutatud: 7.5.2018.

[13] D.K.Taft, "jQuery Eases Javascript, AJAX Development". [Võrgumaterjal]. Asukoht: http://www.eweek.com/development/jquery-eases-javascript-ajax-development. Kasutatud: 7.5.2018.

[14] J.J.Garrett, "AJAX: A New Approach to Web Applications". [Võrgumaterjal]. Asukoht: http://www.adaptivepath.org/ideas/ajax-new-approach-web-applications/. Kasutatud: 7.5.2018.

[15] R.Khanna, P.Chutani, and P.Bhalla, "Study of Spring framework," *International Journal of Research*, vol.1, issue 10, pp. 419-423, November 2014. [Võrgumaterjal]. Asukoht: https://edupediapublications.org/journals/IJR/article/viewFile/858/807. Kasutatud: 7.5.2018.

[16] R.Posa, "Introduction to NODE JS - Node.js Basics". [Võrgumaterjal]. Asukoht: https://www.journaldev.com/7397/introduction-to-node-js-basics. Kasutatud: 7. mai, 2018.

[17] "About Node.js," *Node js*. [Võrgumaterjal]. Asukoht: https://nodejs.org/en/about/. Kasutatud: 7.5.2018.

[18] "BLOB data type," *Oracle Docs*. [Võrgumaterjal]. Asukoht: https://docs.oracle.com/javadb/10.6.2.1/ref/rrefblob.html. Kasutatud: 7.5.2018.

[19] E.Eessaar, "Andmebaaside turvalisuse tagamine," Tallinna Tehnikaülikooli kursusel "Andmebaasid II", peatükk 5, 2017 sügis. [Võrgumaterjal]. Asukoht: http://maurus.ttu.ee/346. Kasutatud: 8.5.2018.

[20] E.Eessaar, "Transaktsioonide haldus," Tallinna Tehnikaülikooli kursusel "Andmebaasid II", peatükk 6, 2017 sügis. [Võrgumaterjal]. Asukoht: http://maurus.ttu.ee/346. Kasutatud: 8.5.2018.

[21] *Gradle User Manual*. [Võrgumaterjal]. Asukoht: https://docs.gradle.org/current/userguide/userguide.html. Kasutatud: 8.5.2018.

[22] *NPM*. [Võrgumaterjal]. Asukoht: https://www.npmjs.com. Kasutatud: 8.5.2018.