TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Erki Miilberg 773794IDDR

Projektide ressursside haldamise rakenduse arendus

Diplomitöö

Juhendaja: Toomas Lepikult
PhD

Tallinn 2021

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: Erki Miilberg

17.05.2021

Annotatsioon

Käesoleva lõputöö “Projektide ressursside haldamise rakenduse arendus” eesmärk on luua tarkvaraline tööriist tarkvaraarendusteenust pakkuvatele ettevõtetele, millega töötajatel avaneb võimalus end iseseisvalt projektide tasemel hallata, minimeerides juhtkonna sekkumist projektidesse määramisel. Lisaks anda hea ülevaade ettevõttega seotud jooksvatest ja tulevastest projektidest ning töötajate kompetentside sobivusest projektidesse määramisel.

Töö teostamiseks on leitud optimaalne lahendus erinevate analüüside toel. Tulemusena on loodud kaasaegseid tehnoloogiaid rakendav ning parimatest praktikatest lähtuv kasutusvalmis veebirakendus, mis loob väärtust selle kasutajatele.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 29 leheküljel, 5 peatükki, 7 joonist, 4 tabelit.

Abstract

Development of Project Resource Management Application

The aim of the thesis is to develop a tool for software development service companies which would like to keep management costs low by letting employees manage themselves within projects. The end result, an application, would also provide a great overview of current and upcoming projects of the company.

In the analysis chapter of the thesis software functional and non-functional requirements are set, possible customers and users are discovered, already existing solutions are compared, and platform choice, system architecture, Technologies, frameworks for every part of the system are analysed.

Implementation chapter proovides detailed overview of system realization, containing description of database model, client application structures and layers, server application structure, authorization and interface, automated testing and code integration.

As a result of the implementation a web application is developed which has user interface which is provided by client application that runs on React framework. Client application depends on server application which is Node.js application that runs NestJS framework that is dependant to relational database PostgreSQL. To ensure proper quality applications are covered with unit and integration tests. Functional requirements are tested with end to end tests which run on whole system except third party integrations. Results chapter also describes further developments that are already being planned.

Requirements that were set for the software were completely fulfilled and system will be applied to one company at first.

The thesis is in Estonian and contains 29 pages of text, 5 chapters, 7 figures, 4 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface* ehk rakendusliides
Bearer	HTTP autentimise skeem
CRA	*Create React App*, React rakenduse loomise tööriist
DB	*Database* ehk andmebaas
DI	*Dependency Injection* ehk tarkvara disaini muster
Github	Pilvepõhine koodihoidla
GUI	*Graphical User Interface* ehk graafiline kasutajaliides
JSON	*Javascript Object Notation*, andmevahetuse vorming
ORM	*Object-Relational Mapping* ehk objekt-relatsiooniline kaardistamine
REST	*Representational State Transfer*, tarkvara arhitektuuri laad hajussüsteemidele
SDK	*Software Development Kit*, teek kolmanda osapoole teenusega suhtlemiseks
SPA	*Single-Page Application*, kasutajaliidese loomine kliendi poolel
SSR	*Server-Side Rendering*, kasutajaliidese loomine serveri poolel
URL	*Uniform Resource Locator* ehk internetiaadress

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tarkvaraarendusteenust pakkuvatel ettevõtetel ei ole tänasel päeval head lahendust projektide ning nende koosseisude planeerimiseks ja haldamiseks nii, et töötajatel oleks võimalik endil kaasa rääkida. Turul on mitmeid erinevaid tooteid, mis võivad küll anda hea ülevaate projektidest ja nendega seotud isikutest või aidata automaatselt projektide meeskondi kokku panna, kuid ükski neist ei ole selline töövahend, mis oleks väärtuslik nii ressursside planeerijatele kui ka meeskonna liikmetele.

Selleks, et täita tulevane projekt nõutud mahus ja kompetentsuses, tuleb tänasel päeval ettevõtte juhtkonnal või projektijuhil esmalt töötajate nimekirjast leida sobivad isikud, kellel on nõuetele vastav kompetents ning tööaja pikkus. Kuna pädevus kasvab ajas, siis tuleb paluda töötajatel mingi intervalliga oma kompetentsi uuendada. Seejärel tuleb nende seast välja filtreerida need, kellega seotud projektide lepingud hakkavad lõppema või on neid võimalik aktiivsetest projektidest asendada. Peale seda on vaja veel loota, et projekt on sobivatele isikutele piisavalt väljakutsuv

Probleemi saab lahendada tarkvaraga, mis suudaks indiviidide kompetentsi hinnata projektides ja koolitustel osalemise põhjal ning mis viiks nad kokku sobivate projektidega, et tekiks töötajatel võimalus märku anda, millistes projektides nad jätkata sooviks.

Käesoleva lõputöö eesmärk on vähendada administreerimise mahtu tarkvaraarendusteenust pakkuvates ettevõtetes ning seejuures anda rohkem otsustusõigust meeskonna liikmetele. Loodava tööriista kasutusala ei pea piirduma ainult tarkvaraarendusega, kuid tarkvara esimeses etapis lähtutakse eelkõige just selliste ettevõtete huvidest.

Eesmärgi saavutamiseks määratakse tarkvarale funktsionaalsed- ja mittefunktsionaalsed nõuded, võrreldakse analoogseid lahendusi ning selgitatakse analüüsi käigus välja sobivaim platvorm, süsteemi arhitektuur, tehnoloogiad ja raamistikud. Analüüsi

tulemuste põhjal luuakse nõuetele vastav funktsionaalne tarkvara, mida esitletakse potentsiaalsele kliendile.

2 Analüüs

Analüüsis on rakendatud vaatluse, võrdluse ja eksperimenteerimise metoodikaid. Selle käigus seatakse tarkvarale funktsionaalsed- ja mittefunktsionaalsed nõuded, võrreldakse analoogseid lahendusi ning selgitatakse välja potentsiaalsed kliendid, sobivaim platvorm, süsteemi arhitektuur ning tehnoloogiad ja raamistikud.

2.1 Tarkvara funktsionaalsed nõuded

Funktsionaalsed nõuded on kriteeriumid, mis defineerivad kindlad käitumised või funktsioonid, mida tarkvara peaks tegema. Funktsionaalsed nõuded jaotatud kasutajarollide ning nende tegevuse iseloomu järgi. [1]

- **Autentimata kasutaja** – süsteemi kasutaja, kellel puudub kasutajakonto või ei ole oma kontoga sisse loginud
    - Autentimata kasutaja saab sisse logida e-maili ning parooliga, Google või Github kontoga
    - Autentimata kasutaja saab luua kasutajakonto.
    - Autentimata kasutaja saab saab lähtestada parooli, kui see on ununenud.
- **Autentitud kasutaja** – süsteemi kasutaja, kes on enda isiku tuvastanud
    - Autentitud kasutaja saab välja logida.
    - Autentitud kasutaja saab luua uue organisatsiooni ning selle käigus loodud organisatsiooni haldajaks.
- **Organisatsiooni kasutaja** – autentitud kasutaja, kes on seotud organisatsiooniga.
    - Organisatsiooni kasutaja saab vaadata organisatsiooni informatsiooni.
    - Organisatsiooni kasutaja saab vaadata kõiki organisatsiooni projekte ning koolitusi.
    - Organisatsiooni kasutaja saab vaadata oma projektide taotlusi.
    - Organisatsiooni kasutaja saab muuta enda profiili.
    - Organisatsiooni kasutaja saab taotleda projekti vahetust.
    - Organisatsiooni kasutaja saab vaadata oma teavitusi.

- **Mitme organisatsiooniga seotud kasutaja** – organisatsiooni kasutaja, kes on seotud rohkem kui ühe organisatsiooniga.
    - Mitme organisatsiooniga seotud kasutaja saab vahetada organisatsiooni, keda esindab.
- **Organisatsiooni haldaja** – organisatsiooni kasutaja, kes on organisatsioonis haldaja rollis. Kehtivad kõik funktsionaalsed nõuded, mis kehtivad organisatsiooni kasutajale.
    - Organisatsiooni haldaja saab muuta organisatsiooni informatsiooni.
    - Organisatsiooni haldaja saab luua ja muuta kõiki organisatsiooni projekte ning koolitusi.
    - Organisatsiooni haldaja saab kinnitada projektiga seotud taotlusi.
    - Organisatsiooni haldaja saab tagasi lükata lükata projektiga seotud taotlusi.
    - Organisatsiooni haldaja saab lisada organisatsiooniga seotud isikuid.
    - Organisatsiooni haldaja saab muuta organisatsiooniga seotud isikuid.
    - Organisatsiooni haldaja saab kustutada organisatsiooniga seotud isikuid.
    - Organisatsiooni haldaja saab luua unikaalse viite, millega kasutaja saab end organisatsiooni isikuga siduda.
    - Organisatsiooni haldaja saab vaadata kõiki projektide taotlusi.
    - Organisatsiooni haldaja saab kutsuda organisatsiooni kasutajaid projektidesse.
    - Organisatsiooni haldaja saab luua organisatsiooni kompetentse.
    - Organisatsiooni haldaja saab muuta organisatsiooni kompetentse.
    - Organisatsiooni haldaja saab kustutada organisatsiooni kompetentse.

Joonisel 1 on kujutatud funktsionaalsed nõuded kasutuslugude skeemil, kus on kirjeldatud kasutajarollid ning nende tegevused lihtsustatud kujul.

Joonis 1. Kasutuslugude skeem

2.2 Tarkvara mittefunktsionaalsed nõuded

Mittefunktsionaalsed nõuded on tarkvara kriteeriumid, mis täpsustavad süsteemi töökäiku, mitte kindlaid omadusi. Järgnevalt on loetletud lõputöös loodava süsteemi mittefunktsionaalsed nõuded [1]:

- Kasutajaliides on inglise keeles.
- Kasutajaliides on kasutatav mobiilses seadmes.
- Süsteem on kaetud automaattestidega.

- Süsteem on turvaline ja usaldusväärne selle kasutajatele.
- Lähtekood on kirjutatud inglise keeles.

2.3 Potentsiaalsed kliendid

Süsteemi esialgne implementatsioon lähtub tarkvaraarendusteenust pakkuvate ettevõtete vajadusest, kus on korraga töös mitmeid erinevaid projekte. Loodavas süsteemis saab olema piisav abstraktsus, et oleks võimalik kohandada ka tegevusalade ettevõtetele või osakondadele. Süsteemi kasutajateks on nii ettevõtete juhid kui ka alluvad, kes on seotud projektidega.

2.4 Analoogsed lahendused

Sarnast funktsionaalsust pakkuvaid tooteid on turul mitmeid, kuid mitte ükski neist ei oma funktsionaalsust, kus projektide meeskondade liikmed saaksid end ise projektidesse määrata, mis on käesoleva lõputöö peamine eesmärk.

2.4.1 Intelligentne töö planeerimise platvorm Planless

Planless on veebirakendus, mille põhiline fookus on projektimeeskondade automaatsel planeerimisel. Selles tuleb esmalt defineerida projektid ja nende nõuded kompetentsidele ning seejärel isikud ja nende oskused. Projektide meeskonnad genereeritakse vastavalt kompetentsidele automaatselt ning neid on võimalik manageerijal muuta. Planeerimise kvaliteedi tõstmiseks kasutatakse tehisintellekti. [2]

Toode on väärtuslik ettevõtetele, mis soovivad projektide ressursi jagamist automatiseerida. Peamine erinevus on tavakasutaja rolli puudumine ehk ettevõtte töötajatel pole võimalik tutvuda projektidega.

2.4.2 Teamdeck - ressursside haldamise tarkvara meeskondadele

Teamdeck on kaugtöömeeskondadele mõeldud tööriist, mille põhifunktsionaalsuseks on ajaskaaladel projektiressursi planeerimine, töögraafikute loomine ning aja mõõtmine.

Lisaks on seal võimalik määrata tähtpäevi ja puhkuseid. Töötajad saavad teha taotlusi puhkusteks ja vabadeks päevadeks. [3]

Veebirakendus on lihtsa kasutajaliidesega ja kasuliku funktsionaalsusega. Palju on sellist funktsionaalsust, mille olemasolu looks lisaväärtust ka lõputöö lahendusele. Peamine erinevus on see, et taotlusi ei saa teha projektide vahel liigutamiseks, mis on käesoleva töö põhiline funktsionaalsus.

2.4.3 Meeskonnatöö tööriist EGroupware

EGroupware on avatud lähtekoodiga meeskonnatööle orienteeritud süsteem, mille peamine funktsionaalsus on kalender kohtumiste haldamiseks, kliendisuhete haldus, tööülesannete haldus, emaili klient ning projektide haldus. [4]

Kasutajaliides on aegunud. Väga palju on funktsionaalsust, mida ei pruugi enamikel ettevõtetel minna, kuid teeb kasutamise keeruliseks. Projektide haldus on vaid väike osa süsteemist ning on mõeldud eelkõige juhtkonnale.

2.5 Platvormi valik

Selleks, et mõista millisele platvormile rakendus esmalt ehitada, tuleb mõelda, kui laia kasutajaskonda platvorm hõlmab, kas on vaja kasutada mingi platvormi *native* funktsionaalsust, kui tähtis on kiirus, kui keeruline on toode, kas seda peab saama kasutada võrguühenduseta ja kui tihti seda uuendatakse. [5]

Peamisteks valikuteks on töölaua-, mobiili- või veebirakendus. Töölaua- ning mobiilirakendused on operatsioonisüsteemi spetsiifilised ehk iga operatsiooni jaoks tuleb luua eraldi rakendus. Ükski platvorm ei ole teist välistav ehk alati on võimalik toetada rohkemaid platvorme.

Kõige laiemat kasutajaskonda hõlmavad veebirakendused. Mobiilide operatsioonisüsteemide tõuketeavituste funktsionaalsus annaks rakendusele lisaväärtust. Kasutusmugavuse seisukohalt, kui juba paigaldatud, on kahtlemata kõige mõistlikumad valikud töölaua- ning mobiilirakendused. Neid on lihtne avada ning kiire kasutada, kuna, erinevalt veebirakendusest, ei eelda kasutamine, et iga kord on vaja kasutajaliides alla

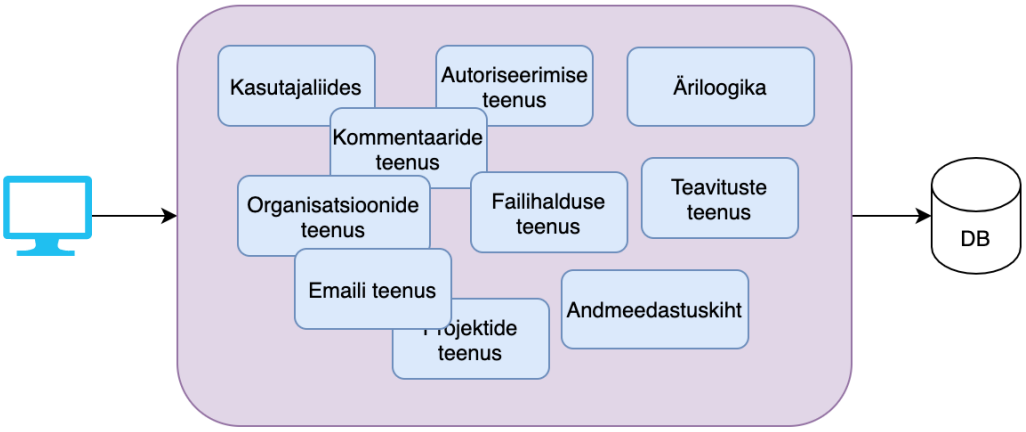
laadida. Antud töö lahendamiseks on mõistlik luua veebirakendus, sest hõlmab kõige laiemat kasutajaskonda.

2.6 Süsteemi arhitektuur

Süsteemi arhitektuuri valikul on kaks peamist suunda: monoliit või mikroteenused. Monoliitarhitekuuri peetakse traditsiooniliseks viisiks rakenduste ehitamisel, samas mikroteenuste populaarsus on viimastel aastatel tunduvalt kasvanud. [6]

Monoliitrakendustel on üldiselt üks suur koodihoidla, neid on lihtne ja kiire arendada, hea testida, siluda ning juurutada, kuid keerukuse ning osaliste arvu kasvades muutub aina raskemini arendatavaks. Kuna kõik süsteemi osad on omavahel seotud, siis on suure projekti puhul muudatuste tegemine keeruline ja veaohtlik ning skaleerida saab ainult tervet süsteemi. Lisaks on keeruline uue tehnoloogia kasutuselevõtt või vana uuendamine, sest väga suur osa süsteemist on kindlasti tehnoloogiast sõltuvuses. [7]

Joonisel 2 on monoliitne arhitektuur planeeritava rakenduse näitel, kus kogu süsteem on tihedalt omavahel seotud ning erinevad teenused üksteisest sõltuvuses.

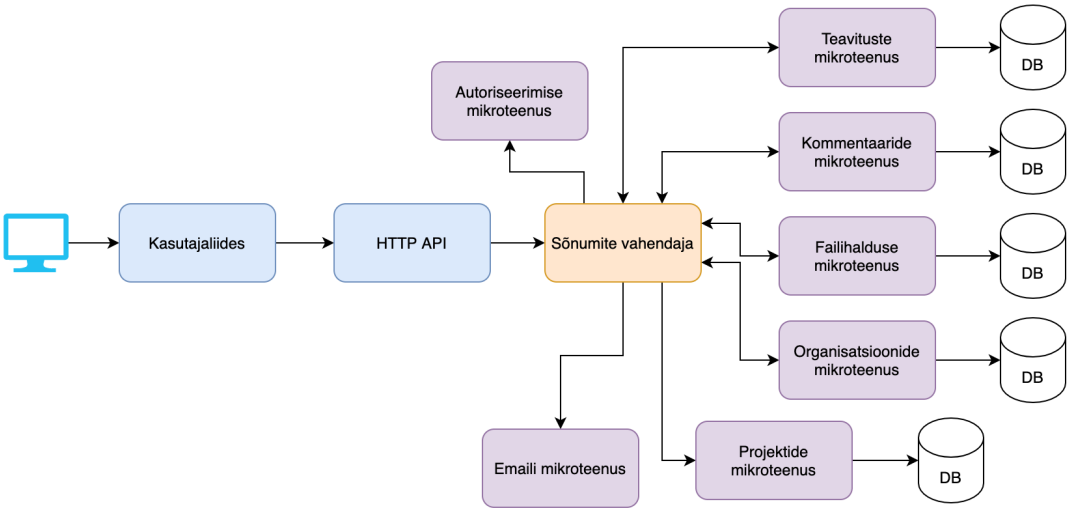


Joonis 2. Monoliitarhitektuur planeeritava rakenduse näitel

Mikroteenused, erinevalt monoliidist, on kogum täiesti iseseisvatest moodulitest, millel on teiste teenustega suhtlemiseks defineeritud *API*-d. Iga teenust on võimalik iseseisvalt juurutada ning skaleerida. Mikroteenuste arendus ja testimine on keeruline, kuid sobib

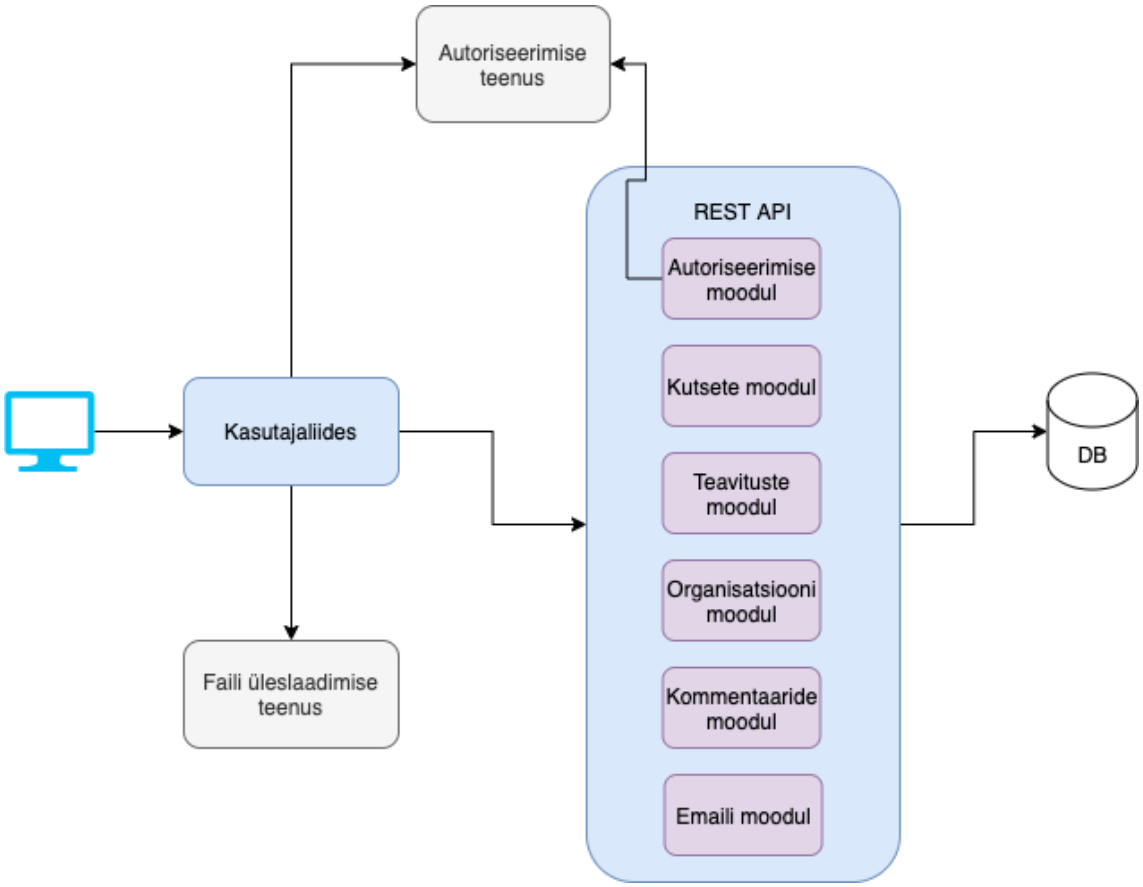
kompleksetele süsteemidele, millega töötab korraga palju arendajaid. Esialgse lahenduse loomine võtab palju rohkem aega kui monoliitarhitektuuri korral. [8]

Mikroteenuse arhitektuurile planeeritava rakenduse ehitamine on näha Joonisel 3, kus kõik teenused suhtlevad omavahel läbi vahendaja ning igal teenusel, mis seda vajab, on oma andmebaas.



Joonis 3. Mikroteenuste arhitektuur planeeritava rakenduse näitel

Mõlemal suunal on nii positiivseid kui negatiivseid omadusi. Mikroteenuste arhitektuuri suurem väärtus kujuneb alles arendusmeeskonna kasvades, aga monoliiti on keerukuse kasvades raske hallata. Seetõttu on valitud lahenduseks hübriid kahest rakenduse arhitektuuri suunast, kus klient-rakendus ja server-rakendus on eraldatud ja suhtlevad läbi liideste. Autoriseerimiseks ja failihalduseks on kasutusel kolmanda osapoole teenused. Joonisel 4 on kujutatud valitud lahendust.



Joonis 4. Valitud arhitektuuriline lahendus – hajutatud monoliit

2.7 Tehnoloogiad ja raamistikud

Tehnoloogiate ja raamistike valikul on peamine osa kaasaegsusel ning suurel rahulolevate kasutajate arvul. Vaadeldud on üksnes tehnoloogiaid ja raamistikke, mille arendus ei ole lõppenud, et oleks võimalik saada uuendusi ning parandusi.

2.7.1 Klient-rakendus

Klient-rakenduste loomisel oli pikalt ainus võimalus luua SSR (*Server-Side Rendering*) tüüpi veebirakendusi, kuid viimasel ajal on rohkem levinud SPA (*Single-Page Application*) tüüpi veebirakendused. Peamine erinevus on selles, et esimesel juhul on vajalik kasutada serveri ressurssi kasutajaliidese loomiseks, aga SPA puhul piisab vaid failihoidlast. SPA tüüpi rakenduste eeliseks on kiire vaadete laadimine, kuid esmane

laadimine, kus alla tuleb laadida kogu raamistik, võib võtta veidi kauem aega. Kuna SPA tüüpi veebirakendused on tänapäeval eelistatum variant, siis osutub see valituks ka käesoleva töö klient-rakenduse lahendusel. [9]

Raamistike valik on suur ning kasutamise osakaal on pidevas muutuses seoses tehnoloogia arenguga. 2020. aasta lõpus läbi viidud küsitluste tulemusel on kõige kasutatavamad raamistikud praegusel hetkel React, Angular, Vue.js, Svelte ning Preact. Kõik käsitletavad raamistikud on loodud Javascripti baasil. [10]

Tabelis 1 on nimetatud raamistike võrdlus, Rahulolu tähistab seda, kui suur osa küsitluses osalenutest märkisid, et nad on rahul antud raamistikuga. Huvi näitab protsenti osalenutest, kes sooviksid antud raamistikuga lähemalt tutvuda ja selle kasutusele võtta ning kasutamine tähistab seda, kui suur osa küsitlusel osalenutest raamistikku kasutanud on.

Tabel 1. 2020. aasta detsembris läbi viidud küsitluse tulemustel põhineva SPA raamistike võrdlus

Raamistik	**Rahulolu**	**Huvi**	**Kasutamine**
Svelte	89%	66%	15%
React	88%	58%	80%
Vue.js	85%	63%	49%
Preact	78%	40%	13%
Angular	42%	21%	56%

Valikul on kõige olulisem see, et raamistik oleks laialdaselt kasutuses ning kasutajate rahulolu. Reastades raamistikud rahulolu ning kasutamise protsentide summa järgi, siis võib järeldada, et kõige mõistlikumad valikud on React ning Vue.js. Sveltega, mis on kõige värskem, ollakse küll enim rahul, kuid kasutajaskond on väike ning seega vajab detailsemat analüüsi sobivuse osas. Analüüsi tulemusena on kõige mõistlikum raamistik töö teostamiseks React.

2.7.2 Autentimine

Turvalise ja usaldusväärse autentimise lahenduse loomine nõuab üsna palju ressurssi. Parima lahenduse saab seda mitte ehitades - kasutades kolmanda osapoole teenust, mis on ennast juba tõestanud. [10]

Selleks, et mitte serveri ressurssi kasutada, on eelistatud pilveteenused. Lisaks peaks olema tasuta kasutamise periood, et mitte maksta teenuse eest kuni rakendusel ei ole kasutajaid. Tabelis 2 on võrdluses neli populaarset teenust ja nende tasuta versioonide eelised.

Tabel 2. Autentimisteenuse pakkujate võrdlus

	Tasuta versioon	**Eelised**
OKTA	Tasuta kuni 7000 aktiivse kasutajani [12]	Palju teeke erinevates programmeerimiskeeltes, palju integratsioone, kasutajaliides [12]
AWS Cognito	Tasuta kuni 50000 aktiivse kasutajani [13]	Rohkelt integratsioone, kasutajaliides [13]
Firebase	Piiramatult tasuta [14]	Palju autentimisteenuste integratsioone, lihtne implementatsioon [14]
Auth0	Tasuta kuni 7000 aktiivse kasutaja täitumiseni, kuni kaks autentimisteenuse integratsiooni ning emaili ja parooliga sisselogimine [14]	Kasutajaliides, hea dokumentatsioon, erinevate e-mailide tugi 14]

Valituks osutus Firebase, mis omab kogu vajalikku funktsionaalsust, on turvaline, lihtsa implementatsiooniga. Kõige suurem eelis konkurentide ees on see, et tasuta versioonis ei ole kasutajate arv piiratud.

2.7.3 Server-rakendus

Valitud arhitektuuri tulemusena toimub klient-rakenduse ning server-rakenduse vaheline suhtlus läbi masinloetava liidese, kasutades HTTP-d (*Hyper Text Transfer Protocol*). Javascripti raamistikel on kõige lihtsam vahetada andmeid JSON (*Javascript Object Notation*) formaadis. See on universaalne, kergekaaluline ning kergesti loetav ka inimestele. [16]

Selleks, et andmevahetuse stiil oleks ühtlane, on loodud standard nimega REST (*Representational State Transfer*), mis kujutab endast printsiipide kogumit server-rakenduse loomisel. REST on olekuta ehk iga päringuga tuleb saata info kasutaja kohta. See annab võimaluse rakendust paremini skaleerida, mis tähendab, et koormuse jagamisel teenindavate rakenduste vahel on konkreetse tegevuse tulemus alati sama. [17]

Järgnevalt on nimekiri programmeerimiskeeltest on tänapäeval kasutatavad ning millele on Firebase autentimisteenuse SDK (*Software Development Kit*) saadaval:

- Node.js – Avatud lähtekoodiga, asünkroonne ehk sündmusjuhitava arhitektuuriga Javascripti käitlussüsteem, mis on disainitud loomaks skaleeruvaid võrgurakendusi. Põhilised eelised on arendaja produktiivsus, koodi taaskasutatavus, kiirus ja jõudlus ning rohke tööriistade ja teekide olemasolu. [18], [19]
- Java – Samaaegne, staatiliste andmetüüpidega ning objektorienteeritud programmeerimiskeel, mis on disainitud nii, et oleks võimalikult vähe sõltuvusi välistest teekidest. Java eelisteks on õppimise lihtsus, turvalisus ning stabiilsus. [20]
- Python – Algselt loodud skriptimiskeelena, millel on dünaamilised andmetüübid. Peamised eelised on lihtne süntaks, palju kasutajaid ja kõrge arendajate produktiivsus. [21], [22]
- Go – Üsna värske staatiliste tüüpidega ning kompileeritav programmeerimiskeel. Põhilised eelised on kiirus, õppimise lihtsus, skaleeritavus ning terviklikud arendustööriistad. [23], [24]
- C# – Kaasaegne üldotstarbeline objektorienteeritud programmeerimiskeel, mis on tugevalt tüübitud. See on kiire, turvaline ning modernne. [25]

Antud projekti teostamiseks on valitud Node.js, mis annab võimaluse jagada lähtekoodi klient-rakendusega, sest on samas programmeerimiskeeles. Lisaks on Node.js vähese mälukasutusega ning kerge kaaluga, aga dünaamiliste tüüpidega, mis tähendab, et tüüpimise vead tulevad välja alles koodi jooksutamisel. Sellel eesmärgil on loodud Typescript, mis võimaldab lisada staatilist tüübikirjeldust. Tüüpide kontroll toimub kompileerimise ajal, kui Typescript kood kompileeritakse Javascriptiks. [26], [27], [28]

Raamistiku valikul on oluline põhjalik dokumentatsioon, kasutamise lihtsus ning kasutajate arv. Rahulolevate kasutajate arvu peegeldavad Github keskkonnas raamistiku koodihoidlale jäetud reaktsioonid (*Star)*. Järgnevalt on loetletud viis kõige enam reaktsioone saanud Node.js raamistikku:

- Express – Minimalistlik ja paindlik raamistik, mis on orienteeritud kiirusele. Eranditult kõige populaarsem raamistik. [29]
- Meteor – Raamistik, mis ühendab kasutajaliidese ning server-rakenduse. Kasutatakse SSR tüüpi veebirakendustes. [30]
- NestJS – Skaleeruvate server-rakenduste loomiseks loodud raamistik, mis on oma olemuselt abstraktsioonikiht Express raamistikule. [31]
- Strapi – Raamistik, millega kaasneb sisuhaldustarkara. Võimalik määrata andmestruktuur läbi selleks loodud kasutajaliidese.[32]
- Koa – Kiirusele orienteeritud väikese kaaluga raamistik, mis on mõeldud väikesemahulistele server-rakendustele.[33]

Valituks osutus NestJS, millel on põhjalik dokumentatsioon ning antud valikust kõige suurema funktsionaalsusega. Sellega on hea luua testitavaid, skaleeruvaid, modulaarseid ning hallatavaid rakendusi. [31]

2.7.4 Andmebaas

Veebiarenduses on kasutusel peamiselt kahte tüüpi andmebaase: relatsioonilised (SQL) ning mitte-relatsioonilised (NoSQL) andmebaasid. Relatsioonilised andmebaasid on kindlalt struktureeritud - selleks, et talletada andmeid, tuleb luua esmalt andmebaasi skeem. Mitte-relatsioonilistel andmebaasidel seevastu ei ole kindlat struktuuri - andmed talletatakse dokumentidena ning nende struktuuri eelnevalt defineerima ei pea. [34]

Teostatavas rakenduses on andmed kindla struktuuriga ning seega on andmebaasi valik relatsioonilise andmebaasi kasuks. Levinumad relatsioonilised andmebaasisüsteemid on MySQL, Oracle, MSSQL ning PostgreSQL. [35]

Sellest valikust olid eelistatud avatud lähtekoodiga andmebaasid, milleks on MySQL ning PostgreSQL. Need on võimekuse ning funktsionaalsuse poolest kõik sobivad ning töö autor on neid kõiki kasutanud. Valituks osutus PostgreSQL, sest on paremini kinni pidanud SQL standarditest ning saab paremini hakkama samaaegsete tegevustega. [36]

2.7.5 Automaattestimine

Tarkvaraarenduses on peamiselt kolm testimise taset - ühiktestid, integratsiooni testid ning süsteemitestid. Iga tase on ühtviisi tähtis ja omab erinevat eesmärki. Ühiktestimise eesmärk on valideerida isoleeritult individuaalse mooduli korrektsust. Integratsiooni testid valideerivad, kas moodulid omavahel liidestuvad. Süsteemitestid kontrollivad terve süsteemi toimimist, hõlmates ka kolmanda osapoole teenuseid. Ühiktestide läbimine on kõige kiirem ning nende osakaal peaks olema kõige suurem. [37]

Kuna rakendusel on nii serveri- kui ka kliendipoolel kasutusel Javascript, siis on ühik- ning integratsiooni testideks mõistlik valida sama raamistik. Süsteemitestid ei sõltu kasutatavatest tehnoloogiatest ning seega on tarvis sõltumatut raamistikku.

Ühik- ja integratsioonitestide raamistike seas on kaks populaarset valikut: Mocha ning Jest. Kuna valitud server-rakenduse raamistikul NestJS on vaikimisi tugi Jestile, siis tasub see võtta kasutusele ka klient-rakendusel.

Süsteemitestide jaoks on peamiselt kasutusel Selenium (või selle baasil loodud raamistik) ja Cypress. Cypressi peamine eelis Seleniumi ees on kasutamise lihtsus ning kiirus. See on tänasel päeval eelistatuim vahend süsteemitestide loomiseks ning on ka antud lahenduse testimisel eelistuseks. [38]

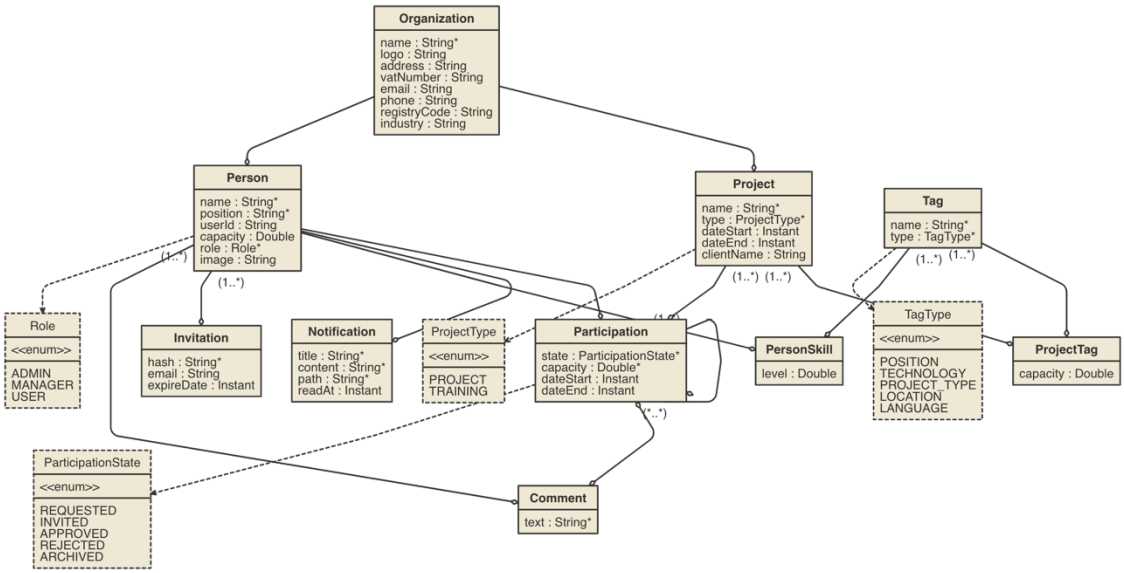
3 Teostus

Süsteemi realisatsioon on tehtud neljas etapis. Esmalt on kaardistatud esmane andmebaasi mudel. Teises etapis on loodud klient-rakendus, mis kasutab esialgu brauseri lokaalset mälu andmebaasina. Seejärel andmebaas koos server-rakendusega ning kõige viimasena süsteemitestide loomine funktsionaalsete nõuete põhjal.

3.1 Andmebaasi mudel

Andmebaasi mudeli koostamine algas olemite kaardistamisega. Mudel on loodud piisavalt abstraktsena tuleviku arenduste jaoks, kuid siiski piisavalt konkreetne, et abstraktsust äriloogikaga korrigeerima ei peaks.

Mudel lähtub sellest, et ühel organisatsioonil saab olla mitu projekti ning mitu seotud isikut. Isikuga võib olla seotud mingi kasutaja, aga alati mitte. Projektile on võimalik lisada mitu silti, igal projektiga seotud sildil võib olla erinev maht. Silte saab olla erinevat tüüpi ning kuvamis- ning käitumisloogika võib olla vastavalt tüübile erinev. Isik on seotud projektiga läbi osaluse olemi. Osaluse olemil võib olla viide teisele osaluse olemile, mis tähendab, et isik soovib muuta oma osalust projektis. Osalusel võib olla mitu kommentaari, mis on seotud alati seotud isikuga. Isikuga võivad seotud olla ka mitmed teavitused.



Joonis 5. Olemi-suhte diagramm

3.2 Klient-rakendus

Klient-rakendusest alustamine aitab kaardistada vajadusi ja teha vajadusel muudatusi andmemudelis, sest kõiki väikseid detaile ei ole võimalik ette näha. Selleks, et luua funktsionaalne prototüüp, võib andmeid talletada esialgu veebilehitseja mälus, tehes piisavalt abstraktse teenuskihi, mille hiljem saab lihtsasti asendada HTTP päringutega.

Esmaseks skeleti loomiseks on kasutatud tööriista CRA (*Create React App*), mis loob toimiva eelnevalt seadistatud React rakenduse. Selleks, et lähtekood oleks paremini loetav ning kasvades paremini hallatav, on Javascripti asemel kasutusel Typescript, mis lisab Javascriptile staatilised tüübid ning kompileerub tagasi Javascriptiks, mis on veebilehitsejatele tõlgendatav.

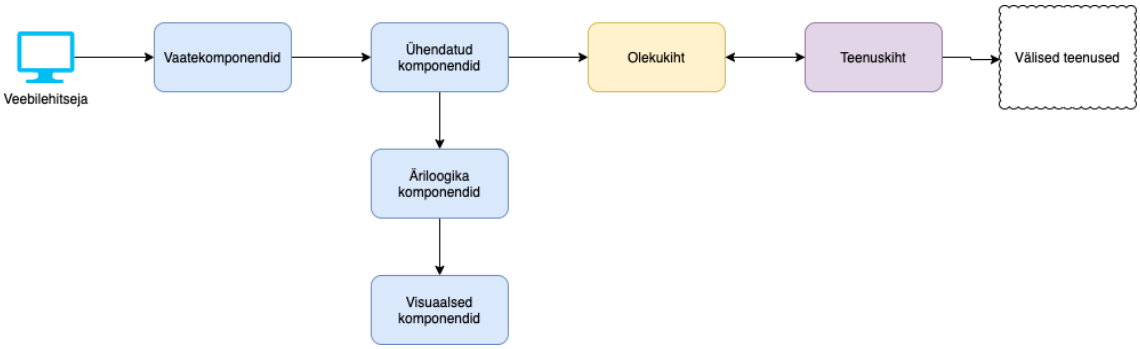
3.2.1 Struktuur

Rakenduse struktuur on jagatud kihtideks, kus igal kihil on oma roll:

- Teenuskiht – vahetab andmeid väliste teenustega ning teisendab vastused olemiteks. Teenuskihiga suhtleb üksnes olekukiht.
- Olekukiht – antud kihi eesmärk on vahendada rakenduses andmeid teenuste ja komponentide vahel ehk võtta vastu tellimusi ning anda märku, kui tellitu on kohal.

- Vaatekomponentide kiht – selle kihi eesmärk on otsustada vastavalt aadressiribale, milliseid komponente kuvama peaks.
- Ühendatud komponendite kiht – tellimusi ja nendele vastuseid ootavad komponendid, mis edastavad oleku äriloogika komponentidele.
- Äriloogika komponentide kiht – komponendid, mis otsustavad etteantud sisendi põhjal, milliseid visuaalseid komponente kuvatakse.
- Presentatsiooni komponentide kiht – visuaalsed komponendid, mille ainus eesmärk on hoolitseda enda väljanägemise eest. Antud kihil puudub seos olemitega.

Joonisel 6 on kujutatud klient-rakenduse struktuur ja erinevate kihtide omavaheline suhtlus.



Joonis 6. Klient-rakenduse kihid

3.2.2 Komponendid

Esimeses etapis said loodud presentatsiooni komponendid, mis peegeldavad otseselt disaineri loomingut. Need on loodud täiesti isoleeritult ja võimalikult abstraktselt ehk äriloogikat arvestamata. Kasutajaliidese disain oli ette antud vaadetena. Selleks, et luua nendest visuaalseid komponente, tuli neid omakorda tükeldada väiksemateks osadeks, leides igas vaates ühisosa ning erisusi.

Selleks, et aega kokku hoida, on presentatsiooni komponentide kiht loodud kolmanda osapoole teegi Chakra UI põhjal, mis on oma olemuselt samasugune visuaalsete komponentide kogumik nagu presentatsiooni komponentide kiht.

Presentatsiooni komponendid on arendatud ülejäänud rakendusest isoleeritult, mis annab võimaluse eraldada presentatsiooni komponendid hiljem ülejäänud rakendusest, et neid saaks kasutada ka teised kasutajaliidesed, kui selleks peaks vajadus tekkima.

Selleks, et vastavalt URL-ile erinevaid vaateid kuvada, on kasutusele võetud teek React Router, mis on võimeline jälgima muudatusi aadressiribal ning seda ise muutma, kasutades Javascript History API-d. Selle teegi abiga on kaardistatud erinevad aadressiriba kombinatsioonid vaatekomponentideks. Need komplekteerivad kokku vaate kontekstis vajalikud äriloogika- ja olekukihiga ühendatud komponendid.

Tabel 3. URL-i teede vastavus vaatekomponentidele

URL-i tee	**Nimetus**	**Skoop**
/	Avaleht	Avalik
/login	Kasutaja autentimine	Avalik
/create-account	Konto loomine	Avalik
/forgot-password	Parooli lähtestamine	Avalik
/logout	Kasutaja sessiooni lõpetamine	Autentitud kasutajad
/create-organization	Organisatsiooni loomine	Autentitud kasutajad
/personal/projects	Isikuga seotud projektid	Organisatsiooni kasutajad
/personal/trainings	Isikuga seotud koolitused	Organisatsiooni kasutajad
/personal/requests	Isiku projektide vahetuste soovid	Organisatsiooni kasutajad
/personal/profile	Isiku profiil	Organisatsiooni kasutajad
/marketplace/projects	Ettevõtte pakutavad projektid	Organisatsiooni kasutajad
/marketplace/trainings	Ettevõtte pakutavad koolitused	Organisatsiooni kasutajad

URL-i tee	Nimetus	Skoop
/marketplace/requests	Kõik ettevõtte projektide vahetde vahetuste soovid	Organisatsiooni haldajad
/organization/profile	Ettevõtte profiil	Organisatsiooni kasutajad

3.2.3 Teenuskiht

Teenuskihis on rakendatud tarkvaradisaini mustrit DI (*Dependency Injection*), mis teeb teenused paremini testitavaks ning parandab loetavust. Selleks on kasutusel tsyringe nimeline teek, mis on arendatud Microsofti poolt. Igal olemil on enda nimeline teenus. Tabel 4 on loetelu kõikidest teenuskihis olevatest teenustest ning nende sõltuvustest. HttpService ning FirebaseProvider vajavad toimimiseks kolmanda osapoole teeke.

Tabel 4. Teenuste nimekiri ja sõltuvused

Teenus	Sõltuvus	Kirjeldus
AuthenticationService	FirebaseProvider	Autentimisega seotud toimingute teenus
ApiService	HttpService, AuthenticationService	REST API-ga suhtlemiseks loodud teenus, mis paneb igale päringule kaasa *Authorization* ning *Accept* päised
AttachmentUploadService	FirebaseProvider	Failide üleslaadimise teenus
FirebaseProvider	Firebase teek	Teenus Firebase instantsi loomiseks
HttpService	Axios teek	HTTP päringute teenus
InvitationService	ApiService	Kutse teenus
NotificationService	ApiService	Teavituse teenus
OrganizationService	ApiService	Organisatsiooni teenus
ParticipationService	ApiService	Osalemise teenus
PersonService	ApiService	Isiku teenus

Teenus	Sõltuvus	Kirjeldus
ProjectService	ApiService	Projekti teenus
ProjectTagService	ApiService	Projekti siltide teenus
TagService	ApiService	Siltide teenus

3.2.4 Olekukiht

Olekukiht vahendab andmeid teenuste ja komponentide vahel ning annab komponentidele teada, kui andmed on muutunud. Olekukiht on lahendatud MobX teegi abil, millel on spetsiaalne integratsioon Reacti jaoks. Igal olemil on oma ladu, kust saab andmeid vastavalt omadustele tellida.

Olekukihi andmelaod on sessiooni skoobiga ehk iga lehe värskendusega lähtestatakse andmed.

3.3 Server-rakendus

Server-rakenduse loomisel on kasutusel Node.js raamistik NestJS. Rakendus on sõltuv PostgreSQL andmebaasist, kuid loodud selliselt, et andmebaasi saab konfiguratsiooni muudatusega välja vahetada. Andmebaasiga suhtlemiseks on kasutusel ORM nimega TypeORM.

3.3.1 Struktuur

Kuigi esialgne süsteemi arhitektuuriline lahendus on monoliit, siis tulevikus võib tekkida vajadus mikroteenuste arhitektuurile üle minna. Sellel eesmärgil on olemid jaotatud eraldi moodulitesse. Iga moodul on ühesuguse struktuuriga ning jaotatud järgnevatesse kihtidesse:

- Kontroller, mis võtab sisendina sisse HTTP päringu info ja annab selle edasi teenusele.
- Andmeedastusobjektid, mis defineerivad lubatud kasutaja sisendi.
- Olem, mis on vastavuses andmebaasi tabeliga.

- Teenused ehk äriloogika kiht, mille sisend/väljund on andmeedastusobjektid ning mis suhtlevad andmetele juurdepääsu kihiga.
- Kaardistajad, mis muudavad andmeedastusobjektid olemiks või vastupidi.
- Andmetele juurdepääsu kiht, mis on abstraktsioon andmebaasi ja äriloogika vahel.
- Mooduli spetsifikatsioon - sisendid ja väljundid.

Eri moodulite teenused suhtlevad omavahel asünkroonselt läbi sündmuste.

3.3.2 Autoriseerimine

Autoriseerimine on olekuta ehk klient peab iga päringuga saatma kaasa teabe enda kohta *Authorization* päises. Rakendus eeldab *Bearer* tüüpi struktuuriga päist, mis sisaldab JSON Web Tokenit, ning valideerib saadud tokeni valiidsust. Kuna autentimise lahenduseks on Firebase, siis tokeni verifitseerimine toimub läbi Firebase SDK.

Kontrolleritele antakse iga päringuga kaasa info kasutaja kohta ning need otsustavad, kas kasutajal on õigus antud ressursile ligi pääseda. Juhul, kui kasutaja ei ole end autoriseerinud, on tagastatav staatuskood 401. Juhul, kui kasutaja on end autoriseerinud, kuid tal ei ole õigust antud toimingut teha, siis tagastatav staatuskood on 403.

3.3.3 Liides

Server-rakenduse liides suhtleb läbi HTTP protokolli. Liides on REST tarkvaraarhitektuuri laadi, mis tagab liidese ühtsuse. Toetatud on GET, POST, PUT, PATCH ja DELETE toimingud ning vastuse vorming on JSON. Tabelis 5 on kirjeldatud server-rakenduse liidese teenused, kus teenuse veerg tähistab HTTP meetodit ja URL-i teed, kirjelduse veerg annab lühiülevaate teenusest, skoop näitab, mis kasutajarollid sinna ligi saavad ning nõutud sisend tähistab seda, mis parameetrid päringuga kaasa tuleb anda.

Tabel 5. Server-rakenduse liidese teenused

Teenus	**Kirjeldus**	**Skoop**	**Nõutud sisend**
POST /organizations	Uue organisatsiooni loomine	Autentitud kasutajad	Nimetus, loova isiku nimi

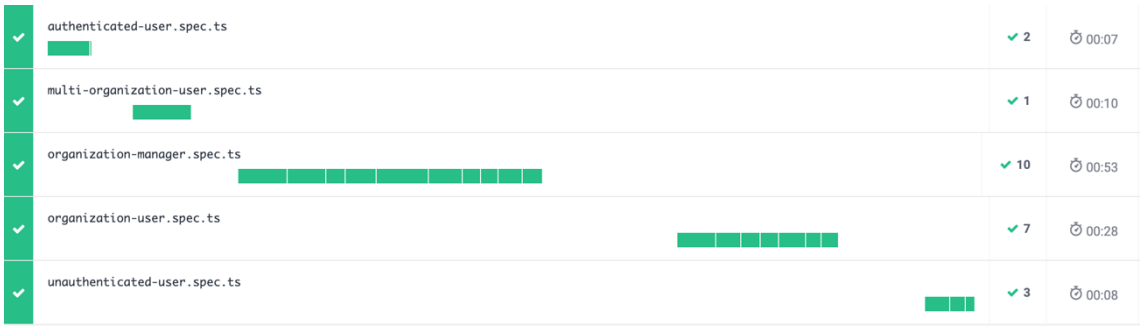
Teenus	Kirjeldus	Skoop	Nõutud sisend
PATCH /organizations/{id}	Organisatsiooni muutmine	Organisatsiooni haldajad	Organisatsiooni ID
DELETE /organizations/{id}	Organisatsiooni kustutamine	Organisatsiooni haldajad	Organisatsiooni ID
GET /organizations/{id}	Organisatsiooni andmed	Organisatsiooniga seotud kasutajad	Organisatsiooni ID
POST /persons	Uue isiku loomine	Organisatsiooni haldajad	Nimetus, organisatsiooni ID
PATCH /persons/{id}	Isiku muutmine	Organisatsiooni haldajad, isikuga seotud kasutaja	Isiku ID
DELETE /persons/{id}	Isiku kustutamine	Organisatsiooni haldajad	Isiku ID
GET /persons/{id}	Isiku andmed	Organisatsiooniga seotud kasutajad	Isiku ID
GET /persons	Kõik isikud	Organisatsiooniga seotud kasutajad	Organisatsiooni ID
POST /projects	Uue projekti loomine	Organisatsiooni haldajad	Nimetus, organisatsiooni ID
PATCH /projects/{id}	Projekti muutmine	Organisatsiooni haldajad	Projekti ID
DELETE /projects/{id}	Projekti kustutamine	Organisatsiooni haldajad	Projekti ID
GET /projects/{id}	Projekti andmed	Organisatsiooniga seotud kasutajad	Projekti ID
GET /projects	Kõik projektid	Organisatsiooniga seotud kasutajad	Organisatsiooni ID
POST /tags	Uue sildi loomine	Organisatsiooni haldajad	Nimetus, organisatsiooni ID
PATCH /tags/{id}	Sildi muutmine	Organisatsiooni haldajad	Sildi ID
DELETE /tags/{id}	Sildi kustutamine	Organisatsiooni haldajad	Sildi ID

Teenus	Kirjeldus	Skoop	Nõutud sisend
GET /tags/{id}	Sildi andmed	Organisatsiooniga seotud kasutajad	Sildi ID
GET /tags	Kõik sildid	Organisatsiooniga seotud kasutajad	Organisatsiooni ID
POST /participations	Uue osaluse loomine	Organisatsiooni seotud kasutajad	Projekti ID, Organisatsiooni ID
PATCH /participations/{id}	Osaluse muutmine	Organisatsiooni seotud kasutajad	Osaluse ID
GET /participations/{id}	Osaluse andmed	Organisatsiooniga seotud kasutajad	Osaluse ID
GET /participations	Kõik osalused	Organisatsiooniga seotud kasutajad	Organisatsiooni ID
PATCH /notifications/{id}	Teavituse loetuks märkimine	Teavitusega seotud kasutaja	Teavituse ID
GET /notifications	Kõik teavitused	Isikuga seotud kasutaja	Isiku ID
POST /invitations	Uue kutse loomine	Organisatsiooniga seotud kasutajad	Organisatsiooni ID
PUT /invitations/{hash}	Kutse aktsepteermine	Autentitud kasutaja	Kutse räsi
GET /invitations/{hash}	Kutse detailid	Autentitud kasutaja	Kutse räsi

3.4 Testimine

Selleks, et veenduda kogu süsteemi korrektses toimimises, on loodud süsteemitestid, mille käivitamiseks luuakse eraldi eksemplarid süsteemi kõikidest osadest - nii klient-rakendus, server-rakendus kui ka andmebaas. Süsteemitestid on kirjutatud funktsionaalsete nõuete põhjal. Need ei kata erandjuhtumeid, vaid veenduvad selles, et põhilised töövood vastavad nõuetele. Erandjuhtumeid katavad integratsiooni- ja ühiktestid. Kolmanda osapoole teenused on asendatud makettidega, mis imiteerivad kolmanda osapoole teenuse käitumist.

Süsteemiteste käivitab Cypress, mis lisaks tulemuste raporteerimisele teeb igast testijuhust video, et oleks lihtsam mõista, kus viga tekkis. Tulemused laetakse üles pilveteenusesse Cypress Dashboard. Joonis 2 on näha testide tulemuste ülevaade eelnimetatud teenuse kasutajaliideses. Testid on grupeeritud vastavalt kasutaja rollile. Kuvatud on testide arv, sooritamise aeg ning tulemused.

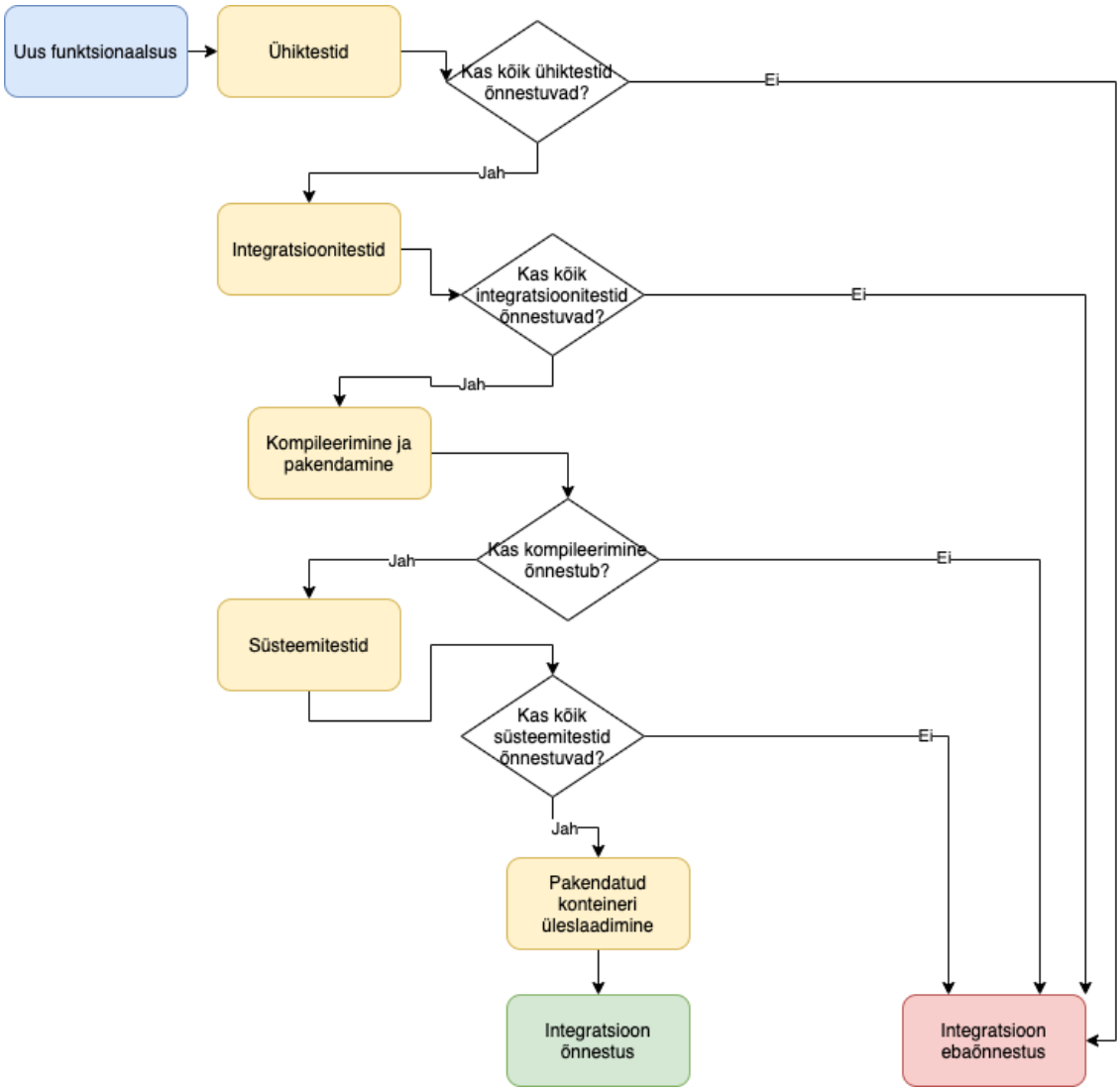


Joonis 7. Cypress Dashboardi testide tulemuste ülevaade

3.5 Integreerimine

Koodi integratsiooni protsess on automatiseeritud. Protsessi käivitab uue arenduse tõuge koodihoidlasse. Isoleeritud keskkonna loomiseks on kasutusel Docker, mis eraldab infrastruktuuri rakendusest. Docker loob konkreetse rakenduse käivitamiseks vajaliku infrastruktuuri. [39]

Esmalt paigaldatakse kõik vajalikud teegid. Seejärel käivitatakse ühik- ja integratsioonitestid. Eduka tulemuse korral kompileeritakse lähtekood ning pakitakse kokku konteineriks. Loodud konteineri käivitatakse koos ülejäänud süsteemi osadega ning käivitatakse süsteemitestid. Õnnestumise korral laetakse üles kokku pakitud konteiner DockerHub registrisse. Süsteemi saab juurutada igasse keskkonda, millel on Docker konteinerite tugi.



Joonis 8. Koodi integratsiooni vooskeem

4 Tulemused

Süsteemi realisatsiooni protsessi tulemusel on valminud süsteem, mis koosneb klient-rakendusest, server-rakendusest ning andmebaasist. Loodud klient-rakendus, mis ehitatud kaasaegsele React raamistikule, ühendub server-rakendusega läbi HTTP protokolli. Server-rakendus, mis on REST laadi liidesega NestJS raamistikul põhinev Node.js rakendus, on ehitatud modulaarsust silmas pidades, et oleks tulevikus võimalik funktsionaalsust eraldada, kui süsteemi keerukus aina kasvab. Äriloogika on kirjeldatud moodulite teenuskihtides. Seega andmebaas, milleks sai valitud PostgreSQL, on kasutusel üksnes andmete talletamiseks ning pärimiseks.

Kolmanda osapoole teenused on kasutusel kasutajate autentimiseks ning failide haldamiseks. Autentimiseks on kasutusel Firebase Authentication teenus, failide üles laadimiseks ning pärimiseks on kasutusel Firebase Storage teenus. Süsteemi osade struktuurid tagavad selle, et kolmanda osapoole teenuseid oleks tulevikus lihtne välja vahetada.

Funktsionaalsed nõuded on kaetud süsteemitestidega. Nendega koos käivitatakse süsteemi kõik osad – välja arvatud kolmanda osapoole teenused, mis asendatakse testimiseks makettidega.

Valminud veebirakenduse funktsionaalsus vastab seatud nõuetele. Kasutaja saab luua kasutajakonto, logida sisse, luua organisatsiooni, kutsuda sellesse teisi kasutajaid, luua projekte ning koolitusi, hallata organisatsiooni profiili ning organisatsiooniga seotud taotlusi. Organisatsiooni kasutajad saavad teavitusi uutest projektidest, nendesse kandideerida, hallata oma profiili.

4.1 Edasiarendused

Käesoleva lõputöö valmimisega ei lõppenud loodud rakenduse arendus. Järgnevalt on loetletud mõned funktsionaalsed, mis on plaanis järgmiste arendusiteratsioonidega implementeerida:

- Otsing – olemite ülene otsing on äärmiselt vajalik funktsionaalsus suurematele ettevõtetele, kus on väga palju projekte ja inimesi. Kiire ja tõhusa otsingu loomine on mahukas töö.
- Aja planeerimine ning arvestus – süsteemis on olemas kogu info selleks, et automaatselt pidada ajaarvestust projektide vahel. Kindlasti peaks olema ka redigeerimise võimalus, kui automaatselt genereeritu ei vasta tõele. Sellega kaasneks ka võimalus raporteid eksportida.
- API võtmete tugi – kolmandate osapoolte kaasamine täiendava funktsionaalsuse ja integratsioonide loomiseks annab rakendusele lisaväärtust. Selleks tuleks lisada server-rakendusele võtmete tugi ning kasutajaliides võtmete genereerimiseks ja haldamiseks.
- Logide monitoorimine – Kolmanda osapoole teenuse integreerimine rakenduste logide haldamiseks ja analüüsimiseks, et leida vead enne kui need suure hulga kasutajateni jõuavad.

5 Kokkuvõte

Käesoleva lõputöö eesmärk oli luua tarkvaraline tööriist, millega ettevõtted saaksid edukamalt hallata inimressursse projektides. Funktsionaalsed- ja mittefunktsionaalsed nõuded said määratud arvestades ühe konkreetse tarkvaraarendusettevõtte huve, kuid mõeldes laiemalt, et loodav süsteem oleks võimeline teenindama erinevaid sektoreid.

Analüüsi käigus selgitati välja sobivaim platvorm, süsteemi arhitektuur, tehnoloogiad ja raamistikud. Töö käigus loodi React raamistikul põhinev klient-rakendus, mis suhtles NestJS raamistikul server-rakendusega. Andmebaas, kuhu server-rakendus ühendus, oli PostgreSQL, mis on relatsioonilist tüüpi. Tarkvaralise lahenduse realiseerimisel oli põhirõhk lihtsusel, murede lahususel erinevates kihtides ning kvaliteedil, mis tagati automaattestidega.

Valminud süsteemi, mis vastab seatud funktsionaalsetele ja mittefunktsionaalsetele nõuetele, juurutab esialgu üks tarkvaraarendusettevõtte. Loodud tarkvara on plaanis tulevikus pakkuda rohkematele ettevõtetele.

Kasutatud kirjandus

[1] L. Chen, M. Ali Babar, ja B. Nuseibeh, *Characterizing architecturally significant requirements*, *IEEE Softw.*, kd 30, nr 2, lk 38–45, märts 2013, doi: 10.1109/MS.2012.174.

[2] *Planless - A New Way Of Planning & Organizing Work*. [Online]. Loetud aadressil: https://planless.io/. Kasutatud: 07.04.2021.

[3] *Teamdeck - leave management solution for your team.* [Online]. Loetud aadressil: https://teamdeck.io/features/leave-management/. Kasutatud: 12.04.2021.

[4] *EGroupware | Online Collaboration tools | Try it for free!* [Online]. Loetud aadressil: https://www.egroupware.org/en. Kasutatud: 12.05.2021.

[5] *App vs Website – Which to Develop First? | Brainhub*. [Online]. Loetud aadressil: https://brainhub.eu/library/app-vs-website-which-to-develop-first/. Kasutatud: 12.04.2021.

[6] *Microservices vs Monolith: which architecture is the best choice?* [Online]. Loetud aadressil: https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/. Kasutatud: 12.04.2021.

[7] *Monolithic Architecture pattern*. [Online]. Loetud aadressil: https://microservices.io/patterns/monolithic.html. Kasutatud: 12.04.2021.

[8] *Microservice Architecture pattern*. [Online]. Loetud aadressil: https://microservices.io/patterns/microservices.html. Kasutatud: 12.04.2021.

[9] *Who is better?: SSR application VS SPA application | INMEDIATUM*. [Online]. Loetud aadressil: https://inmediatum.com/en/blog/innovacion1/who-is-better-ssr-aplication-vs-spa-aplication/. Kasutatud: 12.04.2021.

[10] *State of JS 2020: Front-end Frameworks*. [Online]. Loetud aadressil: https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/. Kasutatud: 07.04.2021.

[11] *5 reasons to use third-party authentication instead of your own | Synopsys*. [Online]. Loetud aadressil: https://www.synopsys.com/blogs/software-security/5-reasons-third-party-authentication/. Kasutatud: 12.04.2021.

[12] *Pricing | Okta*. [Online]. Loetud aadressil: https://www.okta.com/pricing/. Kasutatud: 13.04.2021.

[13] *Pricing | Amazon Cognito | Amazon Web Services (AWS)*. [Online]. Loetud aadressil: https://aws.amazon.com/cognito/pricing/. Kasutatud: 13.04.2021.

[14] *Firebase Pricing*. [Online]. Loetud aadressil: https://firebase.google.com/pricing. Kasutatud: 13.04.2021.

[15] *Pricing - Auth0*. [Online]. Loetud aadressil: https://auth0.com/pricing/. Kasutatud: 13.04.2021.

[16] *JSON*. [Online]. Loetud aadressil: https://www.json.org/json-en.html. Kasutatud: 12.04.2021.

[17] *Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST)*. [Online]. Loetud aadressil: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. Kasutatud: 12.04.2021.

[18] *About Node.js*. [Online]. Loetud aadressil: https://nodejs.org/en/about/. Kasutatud: 23.04.2021.

[19] *Pros and Cons of Node.js Web App Development | AltexSoft*. [Online]. Loetud aadressil: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-node-js-web-app-development/. Kasutatud: 23.04.2021.

[20] *What is Java? Definition, Meaning & Features of Java Platforms*. [Online]. Loetud aadressil: https://www.guru99.com/java-platform.html. Kasutatud: 23.04.2021.

[21] *What is Python? Executive Summary | Python.org*. [Online]. Loetud aadressil: https://www.python.org/doc/essays/blurb/. Kasutatud: 23.04.2021.

[22] *Python Advantages and Disadvantages - Step in the right direction - TechVidvan*. [Online]. Loetud aadressil: https://techvidvan.com/tutorials/python-advantages-and-disadvantages/. Kasutatud: 23.04.2021.

[23] *Documentation - The Go Programming Language*. [Online]. Loetud aadressil: https://golang.org/doc/. Kasutatud: 23.04.2021.

[24] *Go Language - Benefits & Limitations*. [Online]. Loetud aadressil: https://datafloq.com/read/go-language-benefits-limitations/7296. Kasutatud: 23.04.2021.

[25] *What Is C# Language, Advantages & Features Of C# Language*. [Online]. Loetud aadressil: https://www.codexoxo.com/advantages-c-sharp-language/. Kasutatud: 23.04.2021.

[26] *Add the Firebase Admin SDK to your server*. [Online]. Loetud aadressil: https://firebase.google.com/docs/admin/setup. Kasutatud: 13.04.2021.

[27] *JavaScript data types and data structures - JavaScript | MDN*. [Online]. Loetud aadressil: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures. Kasutatud: 13.04.2021.

[28] *TypeScript: Typed JavaScript at Any Scale.* [Online]. Loetud aadressil: https://www.typescriptlang.org/. Kasutatud: 13.04.2021.

[29] *Express - Node.js web application framework*. [Online]. Loetud aadressil: https://expressjs.com/. Kasutatud: 25.04.2021.

[30] *Meteor API Docs | Meteor API Docs*. [Online]. Loetud aadressil: http://docs.meteor.com. Kasutatud: 25.04.2021.

[31] *Documentation | NestJS - A progressive Node.js framework*. [Online]. Loetud aadressil: https://docs.nestjs.com/. Kasutatud: 13.04.2021.

[32] *Strapi - Open source Node.js Headless CMS*. [Online]. Loetud aadressil: https://strapi.io/. Kasutatud: 25.04.2021.

[33] *Koa - next generation web framework for node.js*. [Online]. Loetud aadressil: https://koajs.com/. Kasutatud: 25.04.2021.

[34] *SQL vs. NoSQL Databases: What's the Difference? | Upwork*. [Online]. Loetud aadressil: https://www.upwork.com/resources/sql-vs-nosql-databases-whats-the-difference. Kasutatud: 13.04.2021.

[35] *Most popular relational database management systems 2020 | Statista*. [Online]. Loetud aadressil: https://www.statista.com/statistics/1131568/worldwide-popularity-ranking-relational-database-management-systems/. Kasutatud: 20.04.2021.

[36] *MySQL vs PostgreSQL -- Choose the Right Database for Your Project*. [Online]. Loetud aadressil: https://developer.okta.com/blog/2019/07/19/mysql-vs-postgres. Kasutatud: 25.04.2021.

[37] *The Difference Between Unit, Integration and Functional Testing*. [Online]. Loetud aadressil: https://www.softwaretestinghelp.com/the-difference-between-unit-integration-and-functional-testing/. Kasutatud: 13.04.2021.

[38] *JavaScript End to End Testing Framework | cypress.io*. [Online]. Loetud aadressil: https://www.cypress.io/. Kasutatud: 13.04.2021.

[39] *Docker overview*. [Online]. Loetud aadressil: https://docs.docker.com/get-started/overview/. Kasutatud: 25.04.2021.

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

Mina, Erki Miilberg

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Projektide ressursside haldamise rakenduse arendus“, mille juhendaja on Toomas Lepikult
   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.

17.05.2021

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.