TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Raigo Vilur 142758IAPB

ROBOTITE LIIKUMISE HINDAMIN, SALVESTAMINE JA TAASESITUS VEEBIKESKKONNAS

bakalaureusetöö

Juhendaja: Gert Kanter

Tehnikateaduste magister

Tallinn 2017

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: Raigo Vilur

16.05.2017

Annotatsioon

Käesoleva bakalaureusetöö ülesandeks on luua automaatne hindamissüsteem koos tulemuste vaatamise veebiliidesega Tallinna Tehnikaülikooli õppeaine Robotite programmeerimine jaoks. Rakenduse abil on õppeaine käigus tudengeid lihtsam hinnata objektiivselt ning järjepidavalt ühtsete hindamiskriteeriumite alusel. Töös on kirjeldatud süsteemi nõudeid, struktuuri, disaini, tähtsamate protsesside kirjeldusi ning loomisel tehtud otsuseid koos põhjendustega. Samuti on antud tulemusele hinnang ning pakutakse välja võimalikke süsteemi laiendamise võimalusi.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 32 leheküljel, 4 peatkki, 12 joonist ja 3 tabelit.

Abstract

Assessment, recording and replay of robots' movement in a web environment

This bachelor’s thesis aims to provide an overview of a system that automatically assesses students’ work during the course of “Robot Programming” in Tallinn University of Technology. In the course students must program robots to solve a number of tasks in set test areas. The system described in this thesis will help the lecturers to objectively grade students work by the same set of rules.

The thesis provides an overview of the system by describing the requirements, design, structure and decisions made during the development process with explanations. It also provides assessment of the development process and gives ideas for future expansion of the system. Descriptions of the more complex processes are also provided.

The system is composed of four parts: assessment module, database, server and the user interface. The interface is built as a single page application using React as the main framework. Server is built as an interface to access the database by following REST architectural style and is built in Node.js with Express. User interface communicates with the server via HTTP protocol. The database runs on PostgreSQL with image data saved on server hard drive. The assessment module is written in C++ and captures robots with an Xbox One Kinect camera. Special tags (AprilTags) are used to identify robots and other information via the camera. The module then processes the camera image stream and gives an assessment for the student programmed robot’s performance. The assessment and all the gathered metadata with recorded images are then saved in the database for later review via the user interface.

The thesis is written in Estonian and contains 32 pages of text, 4 chapters, 12 figures and 3 tables.

Lühendite ja mõistete sõnastik

DELETE	HTTP meetod andmete kustutamiseks serveris
GET	HTTP meetod andmete pärimiseks serverilt
HTTP	*Hypertext Transfer Protocol* on protokoll andmete edastamiseks arvutivõrkudes
Katse	Üks üliõpilase ülesande täitmise sooritus robotiga katseväljakul aines Robotite Programmeerimine
ORM	*Object Relational Mapping* on võte relatsioonilste andmete teisendamiseks andmebaasi ja programmi objektide vahel objekt orienteeritud keeltes
POST	HTTP meetod andmete lisamiseks või uuendamiseks
REST	*Representational state transfer* on veebiressurside ligipääsu kirjeldamise viis, kus kasutatakse (HTTP protokolli puhul) HTTP meetodeid (*state transitions*) andmetega opereerimiseks
URI	*Unified resource identifier* on sõne, mille alusel on võimalik ühte ressurssi määrata veebiserveris.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Kuivõrd robootika hõlmab endas nii riistvara kui tarkvara, siis robootika valdkonnas alustamiseks on vaja peale tarkvaaraarendusoskuste ka üldiseid teadmisi elektroonikast ja riistvarast. Robootika vajab lisaks arvutile veel erinevaid elektroonilisi seadmeid (mikrokontrollereid, mootoreid jne) ning nendega valesti ümber käies on oht nende seadmete lõhkumiseks suur ning seetõttu jätavad paljud inimesed kartuse tõttu tegemata esimese sammu robootikas. Just seetõttu on otsustatud ülikoolis informaatika esimese kursuse tudengitele andma ainet Robotite programmeerimine. Nii saavad tudengid teha esimesi katsetusi juhendaja käe all ning nii mõnelgi võib sellest esimesest sammust piisata, et edaspidigi oma elu robotitega siduda.

Kuna aine on hindeline siis tekkis aine planeerimisel probleem: tudengite objektiivne hindamine on täpne töö ning seda tuleb teha kõigile võrdsete reeglite järgi. Inimene kipub aga kergelt vigu tegema ja seetõttu on tudengite kõikide katsete ühesuguse mõõdupuu järgi hindamine võrdlemisi keeruline. Sellest tekkis ka idee, et hindamisel võiks kasutada automaatset süsteemi, nagu seda teevad juba paljudes programmeerimisainetes automaattestid.

1.1 Eesmärgid

Töö eesmärgiks on luua platvorm tudengite hindamiseks aines Robotite programmeerimine. Aine läbimiseks tuleb tudengitel Lego Mindstorms robotite abil meeskondades ülesanded lahendama. Süsteem peab jälgima tudengite poolt programmeeritud robotite katseid kahel testväljakul, andma sooritusele hinnangu ja salvestama katse soorituse kohta metaandmeid. Samuti peab katset olema hiljem võimalik taasesitada veebikeskkonnas. Lisaks katsete taasesitusele peab veebikeskkond laskma praktikumi juhendajal sisestada moodustatud tiime ning tiimi liikmeid ka hiljem vaadata. Töö eesmärgiks ei ole erinevate ülesannete täpsete hindamisjuhiste implementeerimine vaid hindamiseks platvormi loomine. Seega peab katsete juurde

lisamine süsteemi olema võimalikult lihtne, kuid ei pea olema saavutatav ilma koodis muudatusi tegemata. Uute ülesannete lisamine peab toimuma kindlaks määratud liidese kaudu.

1.2 Metoodika

Projekti algstaadiumis läbitakse analüüsi etapp, kus kaardistatakse süsteemile rakendatavad nõuded ja planeeritakse süsteemi struktuur. Seejärel alustatakse süsteemi loomist agiilsel meetodil: kohtutakse iganädalaselt korra süsteemi tellijaga (aine õppejõuga), kellega arutatakse läbi eelmisel nädalal implementeeritud funktsionaalsus ning arutatakse järgmise nädala eesmärke ja nende eesmärkide täitmise võimalusi. Süsteem ehitatakse mitmekihilise klient-server rakendusena keeltes C++ ja Javascript.

1.3 Ülevaade tööst

Töö on jaotatud neljaks suuremaks peatükiks:

1. Analüüs - selles peatükis analüüsitakse loodavat süsteemi, pannakse paika nõuded ja olemid ning nendevahelised suhted.
2. Arendus - analüüsitud süsteemi realiseerimine koos tehnoloogiliste otsuste ja loodud lahenduste põhjustega
3. Tulevik - kirjeldatakse võimalusi, kuidas saaks süsteemi edasi arendada ja paremaks teha
4. Hinnang - antakse loodud süsteemile hinnang.

2 Analüüs

Selles peatükis kirjeldatakse süsteemi funktsionaalsed ja funktsioonivälised nõuded.

2.1 Funktsionaalsed nõuded

1. Kasutaja saab teha õppeaine katse testväljakul ning süsteem salvestab selle.
2. Süsteem annab katsele automaatse hinnangu, mis on kõikide sama ülesande katsete jaoks ühestel alustel.
3. Kasutaja ei pea süsteemi teavitama katse algusest, süsteem peab alguse ise registreerima.
4. Süsteem peab ise registreerima katse lõpu.
5. Süsteem peab kuvama veebiliideses kõikide katsete nimekirja.
6. Süsteem peab lubama veebiliideses kõiki katseid taasesitada.
7. Süsteem peab veebiliideses kuvama katse taasesituse juures katsel kogutud metaandmeid.
8. Süsteem peab suutma veebiliideses katseid filtreerida tiimide alusel.
9. Katseid peab saama vaadata ilma autentimata
10. Administraator peab saama süsteemi sisse logida.
11. Administraator peab veebiliidese kaudu saama sisestada tiime.
12. Administraator peab veebiliidese kaudu saama sisestada tiimide liikmeid.
13. Administraator peab veebiliidese kaudu nägema tiime ning tiimide liikmeid.
14. Administraator peab veebiliidese kaudu saama tiime kustutada.
15. Administraator peab veebiliidese kaudu saama tiimide liikmeid muuta.
16. Administraator peab veebiliidese kaudu saama muuta katse sooritanud tiimi.

2.2 Funktsioonivälised nõuded

1. Süstemi kasutajaliides on ingliskeelne.
2. Süsteemi kood on ingliskeelne.
3. Veebiliides peab olema *responsive*, ehk veebiliides peab olema lihtsasti kasutatav ka mobiilsetel seadetel.
4. Süsteem peab töötama Ubuntu operatsioonisüsteemil.

2.3 Rakenduse arhitektuur

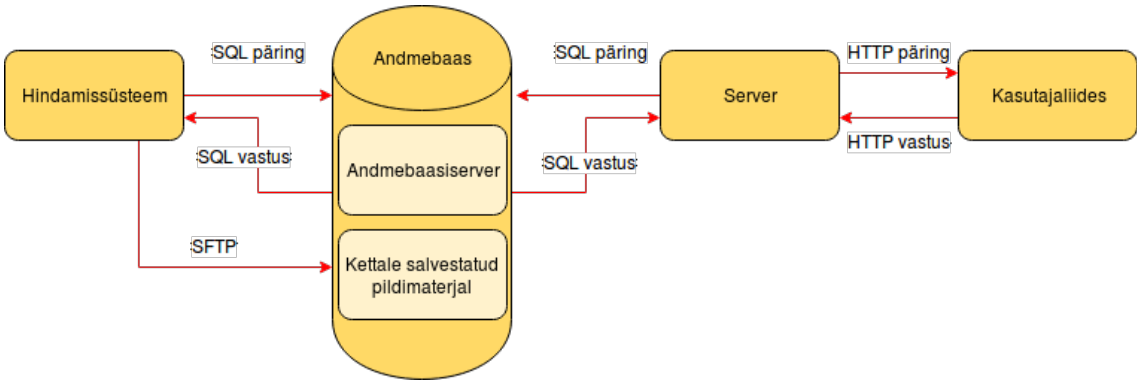
Analüüsi käigus selgus, et rakendus on kõige mõistlikum jaotada neljaks eri funktsionaalsusega osaks:

1. Hindamissüsteem - rakendus, mille ülesandeks on katse salvestamine videona (piltide voona), katse töötlemine ja töödeldud info salvestamine andmebaasi.

2. Andmebaas - andmebaas koosneb andmebaasiserverist ja serveri kettal olevatest failidest. Andmebaasiserveris talletatakse katsete metaandmed ning serveri kettale salvestatakse katsete jooksul salvestatud pildid.

3. Kasutajaliidese server - server on ühenduseks kasutajaliidese ja andmebaasi vahel. Server on üles ehitatud kasutades REST põhimõtet.

4. Kasutajaliides - kasutajaliidese ülesanne on lõppkasutajatele (tudengite, praktikumide õppejõududele) katsete kohta kogutud andmete esitamine ja administraatoritele (praktikumide õppejõudele) mugavalt süsteemi seadistamise võimaldamine.

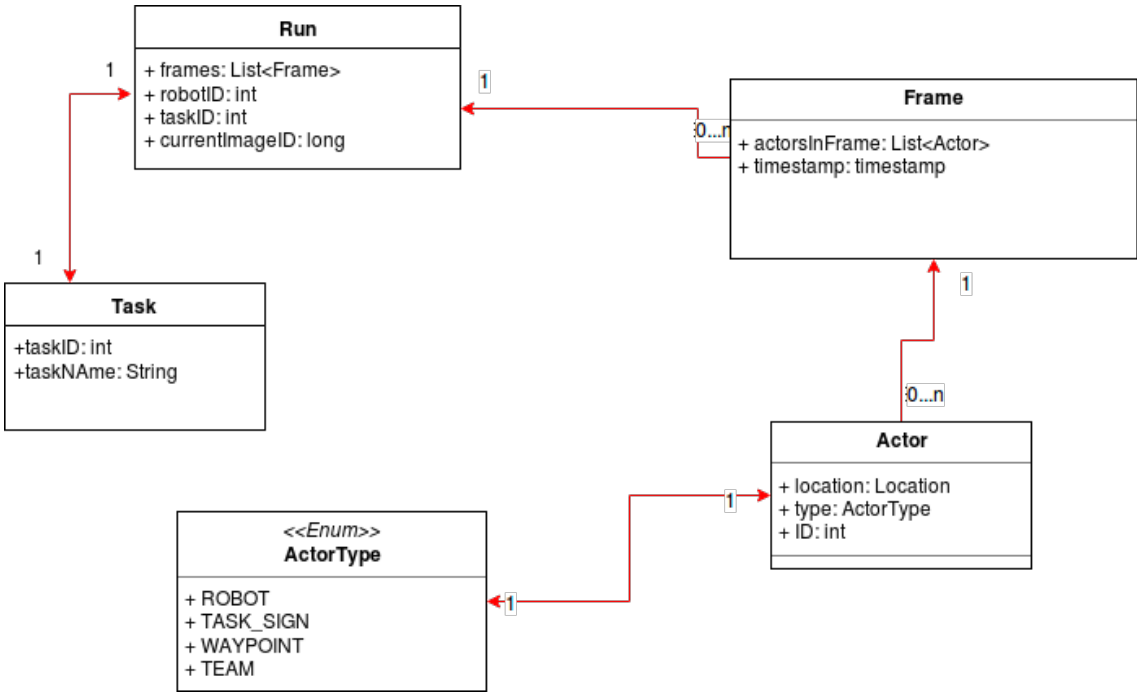


Joonis 1: Süsteemi struktuur

2.4 Olemi-suhte diagramm

Antud peatükis kirjeldatakse süsteemi peamise osa - hindamissüsteemi - olemi-suhte diagramme.

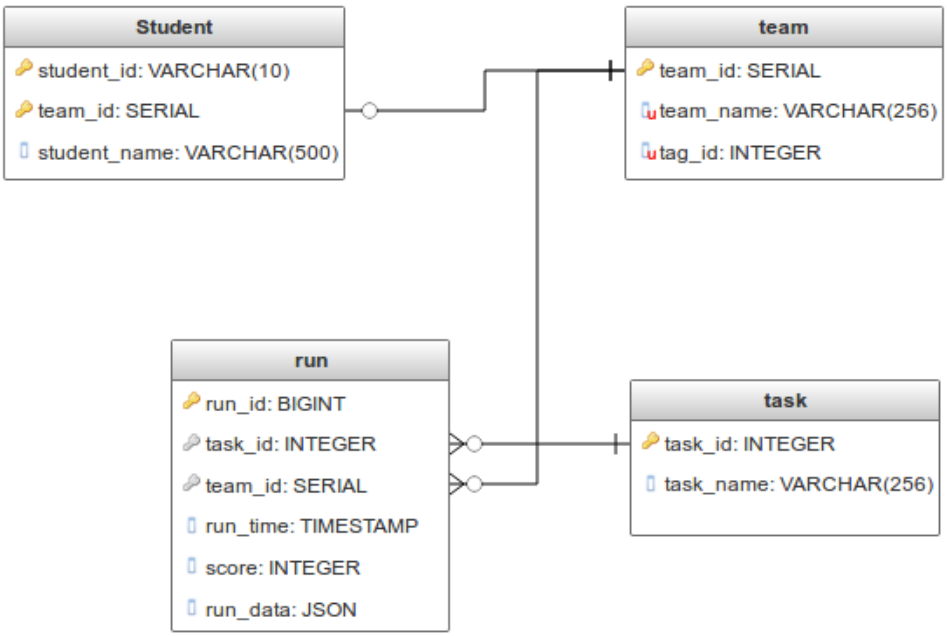
Hindamissüsteemi põhiobjektid on katse (*run*), kaader (*frame*) ja tegutseja (*actor*). Nende omavahelisi sõltuvusi kirjeldatakse joonisel nr 2.



Joonis 2: Olemi-suhte diagramm

2.5 Andmebaasi disain

Andmebaas peab olema suuteline salvestama vähemalt katseid, nendega seotud tiime ja tiimide liikmeid. Joonisel nr 3 on kujutatud nende tabelite välju ja tabelitevahelisi seoseid.



Joonis 3: Katsetega seotud andmetabelite disain

Lisaks nendele tabelitele on andmebaasis veel tabel administraatorikasutajate salvestamiseks. Administraatorite andmete tabelit on kujutatud joonisel nr 4.

Joonis 4: Administraatori kasutajate andmete tabel

2.6 Kasutajaliidese vaated

Kuigu enamik projektile kulutatud ajast läks hindamissüsteemi loomisele tuli luua ka veebiliides tulemuste kuvamiseks. Analüüsi käigus pandi paika, et kasutajaliideses peavad olema minimaalselt järgmised vaated:

1. Vabalt ligipääsetavad vaated
    1. Kõikide katsete nimekirja vaade.
    2. Ühe katse vaade, kus on võimalus katset taasesitada videopildina ning vaadata katse tulemusi.
    3. Tiimide nimekirja vaade.
    4. Ühe tiimi vaade, kus kuvatakse antud tiimi katsete nimekirja.
2. Administraatori vaated
    1. Sisselogimisvaade
    2. Tiimide lisamise vaade
    3. Tiimi ja tiimi liikmete (tudengite) vaatamise vaade
    4. Tudengite lisamise vaade
    5. Tudengite kustutamise vaade

3 Arendus

Antud peatükis kirjeldatakse süsteemi arendusel valitud tehnoloogiaid ja loodud lahendusi.

3.1 Kasutatud tehnoloogiad

Järgnevates peatükkides kirjeldatakse ja põhjendatakse süsteemi tehnoloogilisi valikuid. Ülevaade on jaotatud vastavalt süsteemi loogilisele jaotusele kolmeks mooduliks: hindamissüsteem, veebiliides koos teda toetava serveriga ning andmebaas.

3.1.1 Hindamissüsteem

Hindamissüsteem on kirjutatud kastuades C++ versiooni 11. C++ sai valituks, kuna süsteem peab hindamiseks töötlema suurtes kogustes pildiandmeid. C++ on kõrgtaseme keeltest kõige paremini riistvaral töötamiseks optimeeritud keel. C++-il on ka lai tugi erinevatele pilditöötluseks vajalikele teekidele ning sellele on loodud AprilTags-i teek, mida robotite tuvastamiseks kasutatakse.

AprilTags on teek, mis suudab pildiandmetest välja lugeda spetsiaalseid väljaprinditavaid märke. Märgid on spetsiaalselt loodud robotite tuvastamiseks piltidelt ja on erinevalt alternatiividest (nagu näiteks QR koodid) robustsem, töötades hästi kaamera suhtes suurte nurkade all ja kehvades valgustingimustes. Teek saavutab selle eelise QR koodide ees salvestades märkidele tunduvalt vähem andmeid (4-12 bitti) ning märgid on suuremad [1].

Robot Operating System (lühidalt *ROS*) on vahevara mitmetest erinevatest komponentidest koosnevate automaatsete süsteemide loomiseks. ROS teeb komponentide vahelise suhtluse lihtsamaks, kasutades selleks standardiseeritud sõnumeid. Lisaks aitab ROS veel hallata sõltuvusi. Käesolevas töös kasutatakse ROS-i Kinect One kaamera ja hindamissüsteemi vaheliseks suhtluseks [2].

Kinect for Xbox One kaamera (lühidalt *Kinect One*) on Microsofti loodud kaamera, mis lisaks videopildile suudab salvestada ka sügavusandmeid. Kuigi antud projektis sügavusandmeid ei kasutata sai Kinect One valitud pidades silmas süsteemi võimalikku laiendamist tulevikus [3].

IAI Kinect2 (teise nimega *Kinect2Bridge*) on teek, mis suudab ühendada Kinect One kaamera ROS-i ökosüsteemi, teisendades kaamera salvestatud pildiandmed ROS-i sõnumiteks [4].

CMake on laialdaselt kasutatav vabavaraline C++ kompileerimisprotsessi juhtimise tööriist. CMake lihtsustab tunduvalt C++ rakenduste sõltuvuste haldamise, automatiseerides nii *header* failide kui ka teekide otsimise süsteemist, et neid rakenduse kompileerimisel kasutada [5]. ROS kasutab CMake'i sisemiselt [6].

OpenCV on pilditöötluseks kasutatav teek. OpenCV on vajalik, et AprilTags-i teek suudaks piltidelt üles leida tähised. OpenCV võimaldab, pidades süsteemi laiendamisvõimalusi silmas, leida vähe kergema vaevaga pildilt erinevaid objekte, näiteks ülesande täitmiseks vajalikke objekte [7].

SFTP on protokoll failide edastamiseks kasutades SSH protokolli. Antud rakenduses kasutatakse SFTP-d piltide saatmiseks hindamissüsteemist andmebaasi [8].

SSH on protokoll turvalise ühenduse loomiseks kahe arvuti vahel [9].

3.1.2 Veebiliides ja server

Veebiliides on loodud ühelehelise rakendusena (*single page application*) kasutades Reacti.

React on kasutajaliideste loomise raamistik. Reactiga ainuüksi ei saa luua tervet esitluskihti, kuna teek tegeleb ainult vaadete kokku panemisega komponentidest. Reacti põhiline konkurent on Angular [10]. Kasutajaliidese tehnoloogiat valides toimuski otsustamine nende raamistike vahel, kuna mõlemad olid töö kirjutamise hetkel laialdaselt kasutusel olevad ning seetõttu oli raamistike kohta saada olevat informatsiooni laialdaselt. Angulari eeliseks Reacti ees oli, et ta pakub ka võimalusi hallata andmete liiklust esitluskihis [11], kuid valituks osutus siiski React, kuna töö

autorile meeldis Reactile omane põhimõte taaskasutada komponente võimalikult palju ja võime lihtsalt integreerida JavaScript ja HTML ühtseks klassiks vähendades sellega loodavate lähtekoodi failide hulka [12].

React Router on Reacti raamistikuga laialdaselt kasutatav vaadete vahel navigeerimist võimaldav teek. React Router loob vaadete jaoks eraldi URI-d ning suudab tänu sellele brauseril pöörduda lingi kaudu kindla vaate poole ka üheleheliseses rakenduses, kus kogu rakendus asub ühel serveri URI-l [13].

Axios on teek, mida kasutatakse kasutajaliidesest andmepäringute tegemiseks. Antud projektis kasutatakse Axiost REST liidesest andmete küsimiseks [14].

Serverina kasutatakse **Node.js**-i [15] ning **Expressi** raamistikku [16]. Algselt oli üks võimalikest serveri raamistikest valikus veel JAVA ja Spring [17]. Mõlemad raamistikud on laialdaselt kasutusel ning seetõttu ei ole kummalgi puudust dokumentatsioonist ega toetavatest teekidest. Springi peamine eelis Expressi ees oli autori tunduvalt suurem kokkupuude antud raamistikuga, kuid loodava süsteemi analüüsi käigus selgus, et serveri ülesandeks jääb ainult andmebaasi ning kasutajaliidese vahelise suhtluse korraldamine REST liidese kaudu, siis kaldus valik tunduvalt väiksema ja lihtsama Node.js-i ning Expressi poole. Expressi kasutades sai serveri lähtekoodi tunduvalt vähendada.

Sequelize on teek, mis võimaldab teha Node.js rakendusest päringuid PostgreSQL andmebaasi. Teek võimaldab ka kasutada ORM-i, kuid antud projektis seda omadust ei rakendata kuna serveri loomise alguseks oli andmebaasi tabelid juba paigas ning seetõttu polnud mõtet neid hakata uuesti serveris mudelitena defineerima [18].

JWT (*JSON Web Token*) on tehnoloogia REST API autentimiseks (õiguste tuvastamiseks kahe poole vahel). JWT võimaldab luua autentimiseks ühe sõne (*tokeni*), mida hilisematele päringutele kaasa panna. JWT võimaldab tokeni sisse kodeerida ka infot, mille abil server ja klient üksteist ära tunnevad (näiteks kasutaja kasutajanime või mõne sessiooni identifikaatori) [19]. Rakenduses kasutatakse JWT implementatsiooni Node.js-is - node-jwt [20]

Ant design (*Antd*) on üldkasutatavate Reacti kujundatud komponentide teek. Antd kasutamine vähendab lehekülje disainile kulutatavat aega [21].

3.1.3 Andmebaas

Andmebaasimootorina kasutatakse antud projektis PostgreSQL-i andmebaasi [22]. Kuna rakenduse tekitatavad andmed (tudeng, tiim, robot, katse) on tugevalt relatsioonilised oli selge, et luua tuleb SQL tüüpi andmebaas. Kuna autoril on pikaajaline kogemus PostgreSQL-iga töötamisel, sai paljude SQL andmebaasimootorite seast valitud välja just PostgreSQL. PostgreSQL-i jaoks on olemas teegid nii C++-i kui ka Node.js rakendustega ühenduste loomiseks ja päringute tegemiseks ning see on vabavaraline tarkvara.

Pildimaterjali ei salvestata otse andmebaasi, selle jaoks on loodud failisüsteemi spetsiaalne kataloog koos kaustastruktuuriga. Põhjused, miks ei ole pildimaterjal salvestatud andmebaasi:

1. Andmebaasi päringud on alati aeglasemad, kui otse kettalt lugemine. Põhjuseks on päringule lisanduvad andmebaasimootori operatsioonid.
2. Kui pildimaterjal on salvestatud andmebaasi, peab kasutajaliides kasutama pilti, tehes päringu vastavale REST liidesele. Seetõttu peab server pildi laadima andmebaasist omale mällu enne saatmist. Otse failisüsteemist pilti laadides võib server pildi otse kasutajaliidesele saata kettalt loetud andmevoona (*stream*-ina).
3. Kuigi on olemas andmebaasimootor, Microsofti SQL-server [23], mis toetab faili voogedastust andmebaasist, on tegu tasulise tootega. Microsofr SQL-serverist on olemas ka tasuta versioon (*Express edition*), kuid seda on tugevalt piiratud [24]. Voogedastus andmebaasist vajab ka serveripoolset lisaloogikat võrreldes failisüsteemist edastusega. Samuti ei lahenda see esimeses punktis välja toodud puudust.

Samas on andmebaasi salvestamisel ka mõned eelised:

1. Andmete varundamine käib ühest kohast, ei ole vaja luua eraldi ketta varundamise protsessi.
2. Piltide endi haldamine on lihtsam. Näiteks kui on vaja pilte mingist ajavahemikust, siis saab need andmebaasist kätte ühe päringuga.

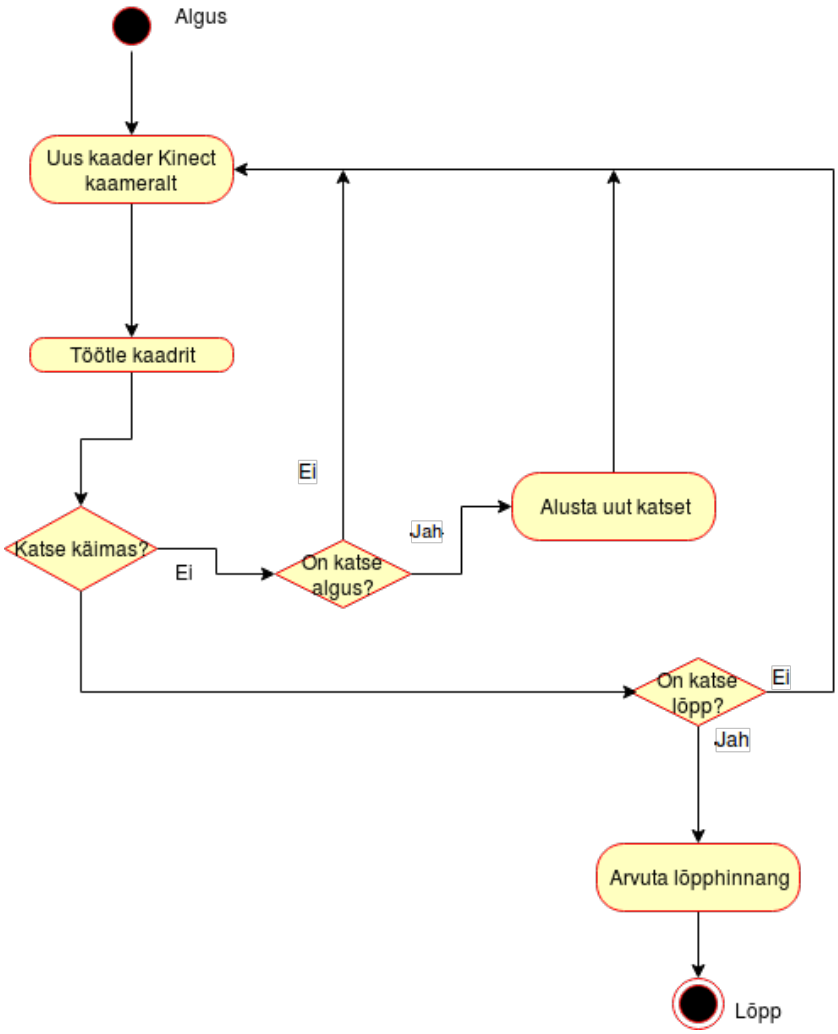
Antud projektis otsustati salvestada pilte kettale, sest pildid moodustavad rakenduse andmevoost enamiku ning seetõttu on nende kiire kättesaadavus prioriteetne. Samas pole serveri rikke tõttu piltide kaotsi minek katastroofilise tähtsusega, sest see takistab ainult katse järelvaatamist - hinnang katse sooritusele on juba antud. Takistuseks ei jää ka piltide haldamisel tekkivad keerukused, kuna pildid on alati seotud mingi katsega ning eraldiseisvalt tähtsust ei oma (üks kaader videopildist eraldi ei anna palju informatsiooni katse kohta), käib piltide pärimine alati kindla katse kaudu. Katsetele saab aga rakendada kõiksugu andmebaasi päringuid.

3.2 Hindamissüsteemi tähtsamad protsessid

Selles peatükis kirjeldatakse hindamissüsteemi tähtsamaid protsesse.

3.2.1 Ühe katse töötlemise loogika

Joonisel 5 on kirjeldatud ühe katse protsessimise loogikat.



Joonis 5: Ühe katse töötlemise loogika

Protsess algab, kui hindamissüsteem saab uue kaadri kaameralt. Uus kaader töödeldakse ning kaadrilt leitakse kõik AprilTagid, mille alusel luuakse tegutsejate (*Actor*) objektid. Tegutsejatest luuakse objekt kaader. Edasi on kolm võimalikku tegevuste käiku:

1. Hetkel juba käib katse. Sel juhul antakse kaadri objekt üle vastava katse hindajale ning hindaja ütleb vastuseks, kas on aeg katse lõpetada. Kui jah, siis katse lõpetatakse, vastasel juhul algab protsess uuesti uue kaadri saamisega kaameralt. Olemasolev kaadri objekt lisatakse katse objekti külge ning kaader pildina salvestatakse andmebaasi.
2. Hetkel pole ühtegi katset pooleli. Katse alustamise protsessist täpsemalt punktis 3.2.2. Kui alustamistingimused on täidetud, küsitakse andmebaasist katsele uus ID ning luuakse uus katse objekt. Kaadri objekt lisatakse katsele ning kaader pildina salvestatakse.

3. Kolmas variant on, et katse kaadrist ei leita roboti *tag*-i, mistõttu käimasolev katse lõpetatakse. Ühegi katse jooksul ei tohi robot lahkuda, et oleks võimalik katseid pooleli jätta süsteemile manuaalseid sisendeid andmata.

Protsessi implementatsiooni on kirjeldatud joonisel nr 6.

```
void MainProcessor::processRun(std::shared_ptr<Frame> frame, cv::Mat image) {
    if (thisRun == nullptr) {
        //No run currectly started, check if start conditions are met:
        if (!processRunStart(frame)) {
            return;
        }
    }
    int robotIdFromFrame = currentTask->getRobotIDFromFrame(frame);

    if (robotIdFromFrame == ROBOT_NOT_FOUND
        && emptyFrameCounter != RuntimeVariables::maxEmptyFrames) {
        emptyFrameCounter++;
        return;
    } else {
        emptyFrameCounter = 0;
        thisRun->addFrameToRun(frame);
    }

    //Run ends, if the end conditions for the task have been met or there is no robot visible
    if (currentTask->isRunEndCondition(frame, image, RuntimeVariables::coordinateReferenceSize)
            || emptyFrameCounter == RuntimeVariables::maxEmptyFrames) {

        ROS_INFO("Run ended");
        //Let task assessor calculate final score and data
        Utils::TaskAssessment assessment =
                currentTask->process(thisRun, RuntimeVariables::coordinateReferenceSize);

        databaseService->saveRun(thisRun, assessment);

        //Reset run data;
        thisRun = nullptr;
        emptyFrameCounter = 0;
        return;
    }

    databaseService->saveImage(image, thisRun->getCurrentImageID(), thisRun->getRunID());
    thisRun->incrementImageID();
}
```

Joonis 6: Ühe katse protsessimise loogika koodinäide

3.2.2 Katse alustamine

Et katset oleks võimalik alustada süsteemile vastavat signaali andmata (näiteks mõne nupuvajutusega), on testväljakule kleebitud üks kindel AprilTag (stardimärk). Süsteem registreerib uue katse alguse, kui stardimärk kaetakse (robot asetatakse stardimärgile) ning ilmub uuesti nähtavale (robot lahkub stardimärgilt). Robot peab seejuures jääma kaamera vaatevälja. Protsessi implementatsiooni on kujutatud joonisel nr 7.

```
bool MainProcessor::processRunStart(std::shared_ptr<Frame> frame) {
    //Run starts if the start tag (id = 0) has been hidden by a robot and then becomes visible again.
    //Run start is only checked if no run is currently active.
    for (std::shared_ptr<Actor> actor : frame->getActorsInFrame()) {
        if (actor->getType() == ActorType::WAYPOINT && actor->getID() == START_WAYPOINT_ID) {
            if (this->startWayPointHasBeenCovered) {
                ROS_INFO("New run started");
                thisRun = std::make_shared<Run>(Run());
                thisRun->setTaskID(currentTaskId);
                thisRun->setRobotID(currentTask->getRobotIDFromFrame(frame));
                thisRun->setRunID(databaseService->requestRunId());
                this->startWayPointHasBeenCovered = false;
                return true;
            }
            return false;
        }
    }

    if (!this->startWayPointHasBeenCovered) {
        this->startWayPointHasBeenCovered = true;
    }

    return false;
}
```

Joonis 7: Katse alustamise koodinäide

Teine võimalus katse alustamise automaatseks registreerimiseks oli, et süsteem ootab kuni stardimärk kaetakse ning hakkab seejärel jälgima kuni robot stardimärgilt ära liigub. Selle lahenduse eelis esimese ees on, et katse algus registreeritakse varem (liikumise alguses mitte siis kui stardimärgilt on ära liigutud). Samas on teise lahenduse implementeerimine aga tunduvalt keerukam: nimelt peab kuidagi eraldama üksteisest kaks erinevat sündmust - roboti asetamine algusmärgile ja roboti tähiselt ära liikumine. Kummalgi juhul pole kaamera vaateväljas märki ning robot liigub. Teise lahenduse keerukuse tõttu saigi otsustatud esimese (märk kaetakse robotiga ja robot lahkub märgi kohalt) kasuks.

Kuna katseväljakuid võib olla rohkem kui üks tekkis katse alustamisel probleem: nimelt katse alguses on vaja teada katse ID-d, et hakata pilte koheselt salvestama (vastasel juhul peab terve katse aja pilte mälus hoidma). Aga kuna väljakuid on rohkem, ei saa ID-d hindamissüsteem ise välja mõelda, sest ta võib minna teiste väljakutega konflikti. Selle probleemi lahenduseks peab andmebaas meeles katse ID-de jada (*sequence*), mille käest iga hindamissüsteem katse alguses uue ID küsib.

3.2.3 Katse lõpetamine

Katse lõpetamiseks on kaks erinevat moodust:

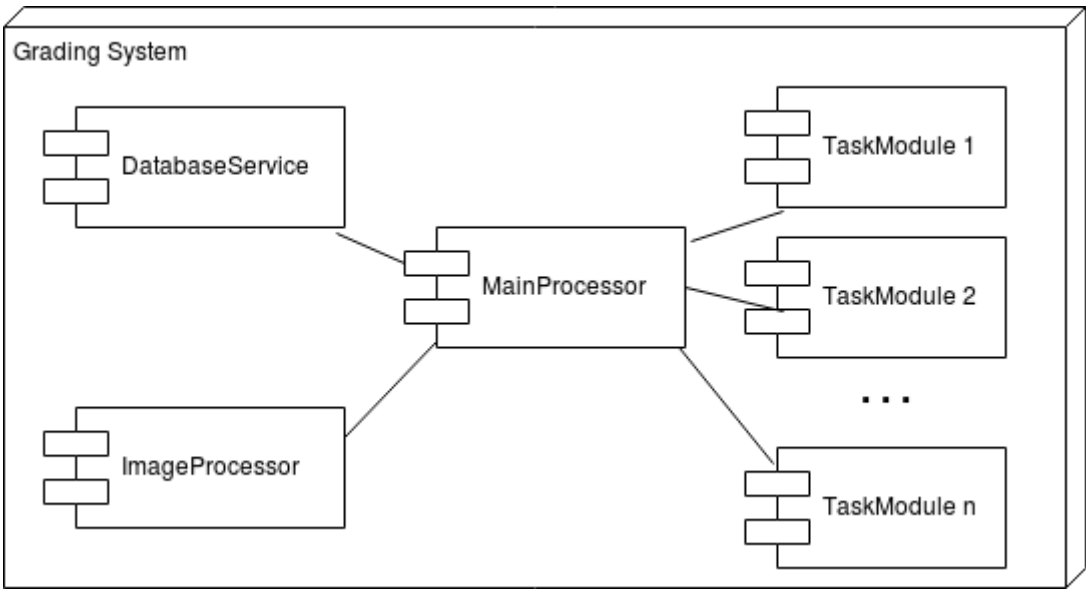
1. Robot lahkub kaamera vaateväljast (eemaldatakse katseväljakult).

2. Praegu hinnatava ülesande hindamismoodul annab teada, et katse on lõpetatud.

Kui on tekkinud üks eelnevalt mainitud olukordadest annab hindamissüsteem hindamismoodulile kõik siiani kogutud katse tegutsejate kohta kogutud teabe ning laseb hindamissüsteemil arvutada hinnangu koos teiste metaandmetega (tulemuse). Seejärel salvestab hindamissüsteem tulemuse koos katse andmetega andmebaasi.

3.3 Hindamissüsteemi komponentide struktuur

Hindamissüsteem on jaotatud kolmeks põhikomponendiks vastavalt komponentide ülesannetele. Põhikomponentidele lisanduvad veel hindamismoodulid. Põhikomponendid on *ImageProcessor*, mis tegeleb pildilt tegutsejate lugemisega, *DatabaseService*, mis tegeleb andmete salvestamisega ning *MainProcessor*, mis tegeleb katsete elutsükliga ja hindamismoodulitega suhtlemisega.



Joonis 8: Hindamissüsteemi komponentide struktuur

3.4 Hindamissüsteemi hindajate liides

Rakenduse loomisel arvestati asjaoluga, et tudengitele antavad ülesanded võivad muutuda ning seetõttu oli vaja koostada ühtne liides, mille kaudu oleks tulevikus võimalik lisada uute ülesannete hindamismooduleid. Igat ülesannet iseloomustab arvuline ID, mille alusel hindamissüsteem valib välja õige hindaja hindajate nimekirjast

ning alustab hindamismooduliga suhtlust läbi kindlaks määratud liidese. Liides peab andma hindamissüsteemile informatsiooni kahel juhul:

1. Ainult hindamismoodul oskab hinnata, kas katse lõpetamise tingimused on saavutatud.
2. Ainult hindamismoodul suudab anda hinnangu katse sooritusele.

Sellest tulenevalt valmis liides, mis koosneb kahest meetodist. Liides on kujutatud joonisel nr 9.

```
class TaskInterface {
public:
    /**
     * Interface for task definition
     * @param coordinates - vector of coordinates captured by the system
     * @param referenceSize - calculated distance between two coordinate points
     */
    virtual Utils::TaskAssessment process(std::shared_ptr<Run> run, double referenceSize) = 0;

    /**
     * Function that gets called by the main processor to let task decide if the run should be finished.
     * @param frame - current frame decoded into actors
     * @param image - raw frame
     * @param referenceSize - coordinate distance modifier
     * @return true, if the run should end
     */
    virtual bool isRunEndCondition(std::shared_ptr<Frame> frame, cv::Mat image, double referenceSize) = 0;

    /**
     * Gets the first robot ID from frame.
     * @param frame frame decoded into actors
     * @return id of the robot
     */
    int getRobotIDFromFrame(std::shared_ptr<Frame> frame);
};
```

Joonis 9: Ülesannetega suhtlemise liides

3.5 AprilTagide jaotus

Rakendus kasutab AprilTagsi märkide versiooni 36h11. Neid märke on 586 erineva ID-ga [1] . See arv on piisavalt suur, et katta kõik rakenduse vajadused, seega teist märkide versiooni lisaks kasutada pole vaja. Iga lisaversiooni tuvastamine pildilt tähendab uut pilditöötlust ning seetõttu pole mõistlik rohkem kui üht versiooni kasutada. Märkide ID-d on jaotatud vahemikeks:

1. 0 - 9: robotid. Kuigi praegu on süsteemi loomisel arvestatud, et roboteid on rajal ainult üks, võib tulevikus lisada ülesandeid, millel peavad mitu robotit korraga üht ülesannet lahendama. Erinevatel robotitel võivad ID-d kattuda kui nad ei ole sama katse ajal rajal. Seega on ID-sid, et korraga rajal jälgida 10-t robotit.

2. 10 - 59: ülesanded. Kuna ülesande tuvastab kaamera ära rajale paigutatud AprilTagi järgi, siis on ülesannetele eraldatud 50 ID-d.
3. 60 - 149: rajatähised (*waypoint*). Ülesande jaoks vabalt kasutatavad ID-d. Rajatähiste põhieesmärk on kontrollida, kas robot läbib etteantud punktid, kuid nende kasutamine on ülesande jaoks vaba. Ainult rajatähis ID-ga 60 on kindlaks määratud kui alguspunkt, mille järgi hindamissüsteem katset alustab.
4. 150 - 586: tiimid. Need on mõeldud kaamera vaatevälja paigutamiseks ning nende järgi saab hindamissüsteem aru, mis tiim praegu katset sooritab.

3.6 REST liidesed

Andmebaasi ja kasutajaliidese vaheliseks suhtluseks sai loodud hulk REST põhimõtteid järgivaid liideseid:

Tabel 1: REST liidesed

Ressurss	**Meetodid**	**Vajab autentimist?**	**Kommentaar**
/authenticate	POST	ei	Administraatori autentimise liides
/team	GET	ei	Tagastab tiim objektide nimekirja
/team	POST	jah	Võimaldab tiimi lisamist
/team/:teamID/run	GET	ei	Tagastab ühe tiimi katsed
/run/:runID	GET	ei	Tagastab ühe katse andmed
/run	GET	ei	Tagastab katsete nimekirja
/student	GET, POST	jah	Võimaldab tudengite lisamist ja nimekirja pärimist
/team/:team_id/students	GET	jah	Tagastab nimekirja ühe tiimi liikmetest
/team/:team_id/student/:student_id	DELETE	jah	Eemaldab tudengi tiimi liikmete seast
/images/:run_id/:image_id	GET	ei	Tagastab kaadri katse taasesituse jaoks

API liidestel on eesliide /api/v1 ning lisaks on kõigil autentimist vajavatel URI-del eesliide */admin*. Võttes näiteks ressursi */team*, siis selle ressursi täispikk POST URI aadress on */api/v1/admin/team*. Kõikidele ülejäänud URI-dele GET päringu tegemine tagastab kasutajaliidese. See on vajalik kuna kasutajaliides on loodud ühelehelise rakendusena (*single page application*). Nimelt kui kasutaja on läinud mingile

kasutajaliidese vaatele ning seal lehel olles otsustab lehe uuesti laadida, teeb *browser* uue päringu serverisse parasjagu aadressiribal olevale aadressile. Ühelehelise rakenduse puhul neid aadresse serveris reaalselt ei eksisteeri ning seetõttu ei oska server neile vastata, küll aga teab neid aadresse kasutajaliides, kes õige vaate ette laeb. Seetõttu on vaja seadistada serverit kõigile GET päringutele, mis tehakse tundmatule aadressile, tagastama kasutajaliidest. Teisi HTTP meetodeid nii suunama ühe ressursi juurde ei pea, sest *browser* laeb kasutajale nähtavaid lehti alati GET päringuga.

3.7 REST liidese turvalisus ja kasutajate autentimine

REST liidesed on jaotatud kaheks: autentimist vajavad ja vabad liidesed. Vabad liidesed on kättesaadavad ainult rakenduse kasutajaliidese kaudu. Selle kindlustab serveripoolne CORS filter. Autentimist vajavad liidesed on saadaval ainult kasutajaliidesest spetsiaalse API tokeni olemasolul. Kasutajaliides saab tokeni serverilt, kui ta saadab autentimisliidesesse õige kasutajanime ja parooli. Kasutajaliides lisab saadud tokeni igale autentimist vajavale päringule. Serveri poolepeal on kõikidel autentimist vajavatel liidestel ees filter (*middleware*), mis kontrollib tokeni olemasolu. Juhul, kui tokenit ei ole või see on vigane, siis filter saadab kasutajaliidesele veateate. Korrektse tokeni puhul lastakse päring edasi konkreetse REST liidese juurde. Selline lahendus eemaldab kõigilt REST liideselt kohustuse kontollida kasutajat. Liideseni jõudes on juba teada, et kasutajal on luba liidesele ligipääsuks. Filtri implementatsiooni on näha koodinäitest nr 10.

```
var adminRoutes = express.Router();
adminRoutes.use(function(req, res, next) {
  //Allows API user to choose what way to send the token:
  var token = req.body.token || req.query.token || req.headers['x-access-token'];
  if (token) {
    jwt.verify(token,  tokenSecret, function(err, decoded) {
      if (err) {
        //verification returned an error:
        return res.json({
            success: false,
            message: 'Failed to authenticate token.'
          });
      } else {
        //verification Successful
        next(); //redirects to requested endpoint
      }
    });

  } else {
    //No token provided
    return res.status(403).send({
        success: false,
        message: 'No token provided.'
    });
  }
});
```

Joonis 10: Autentimise koodinäide

Token genereeritakse kasutades Node.js teeki node-JSONWebToken [20].

3.8 Administraatori sisselogimisandmete salvestamine andmebaasi

Administraatori tuvastamiseks salvestatakse andmebaasi andministraatori kasutajanimi ja paroolist ja soolast arvutatud räsi. Iga kord kui kasutaja üritab administraatorina sisse logida võetakse sisestatud kasutajanimega seotud räsi väli andmebaasist, eraldatakse sellest sool ja räsi ise. Seejärel kasutaja sisestatud paroolist arvutatakse salvestatud soola kasutades uus räsi ning võrreldakse salvestatuga. Nende üks ühele kokku langemisel antakse kasutajale ligipääs. Ligipääsu andmise protsessi kirjeldatakse eelmises peatükis.

Räsi arvutamiseks kasutatakse scrypt algoritmi ning selle implementatsiooni Node.js-is: node-scrypt [17]. Scrypti eelis teiste räsifunktsioonide ees on põhiliselt see, et algoritm on loodud kasutama palju mälu (umbes 16 Mb ühe arvutuse jaoks on soovituslik seadistus [18]). Piisavalt, et räsi arvutamisel ei piisaks protsessori juures olevast vahemälust ning protsessor peaks kasutama üldist mälu. See on aga aeglasem. Samuti

töötab see suurte farmide, mis on loodud räside murdmiseks, vastu, kuna need põhinevad paljudel paralleelselt töötavatel protsessoritel. Kui juba 100 tuuma paralleelselt üritaks räsi arvutada, oleks nõutav mälumaht (kasutades eespool mainitud 16 Mb arvutuse kohta) 1600Mb ehk 1,6Gb. Suuremad räside arvutamise serverid võivad kasutada kordades rohkem tuumasid, kasutades ära graafikakaartide võimet teha palju arvutusi paralleelselt.

3.9 Kasutajaliides

Kasutajaliides on jaotatud kaheks suuremaks komponentide hulgaks: vaadete komponendid, mis realiseerivad mingit kindlat vaadet, ja struktuurseteks komponentideks, mida vaated kasutavad. Vaated ja neid realiseerivad komponendid on kirjeldatud tabelis 2.

Tabel 2: Kasutajaliidese komponendid ja vaated, mida nad realiseerivad

Komponent	**Realiseeritud vaated**
RunListView	Katsete nimekiri
TeamView	Meeskondade nimekiri
TeamRunView	Meeskonna katsete nimekiri
RunView	Ühe katse vaade
AdminLogin	Administraatori sisse logimise vaade
AdminTeam	Administraatori tiimide lisamise/vaatamise vaade
AdminStudent	Administraatori tudengite sisestamise/kustutamise vaade
About	Lisainformatsioon rakenduse kohta
AdminHome	Administraatori avaleht

Loodud struktuursed komponendid on kirjeldatud tabelis nr 3.

Tabel 3: Struktuursed kasutajaliidese komponendid

Komponent	**Roll**	**Komponenti kasutavad vaated**
RunLink	Link ühe katse lehele	RunListView, TeamRunView
Nav	Navigeerimismenüü	Kõik vaated
AdminNav	Administraatori navigeerimismenüü	Kõik administraatori vaated
TeamLink	Link ühe meeskonna katsete vaatesse	TeamView
Player	Katse vaates katse taasesitamise komponent	RunView
AdminFeatures	Administraatori tööriistad ühe katse vaates	RunView
StudentInsertForm	Tudengite lisamise vorm	AdminStudent
TeamInsertForm	Meeskondade lisamise vorm	AdminTeam

3.10 Administraatori staatuse haldamine kasutajaliideses

Selleks, et lihtsamini hoida teavet administraatori staatuse kohta on selleks loodud spetsiaalsed komponendid: AuthService, LoginActions, LoginStore ja AuthenticatedComponent. Selles peatükis kirjutatakse nendest komponentidest.

AuthService on komponent, mis haldab sisselogimise protsessi. AuthService alustab tööd kui kasutaja vajutab nupule “Log In”. Selle peale teeb komponent päringu serveri autentimisliidesele ning positiivse tulemuse korral annab töö koos serverilt saadud tokeniga töö edasi komponendile LoginAction. Nupule “Log Out” vajutamisel käib protsess samuti läbi AuthService’i, kuid sel puhul pole vaja serverile päringu teha ning käsklus edastatakse otse edasi komponendile LoginAction.

LoginAction tegeleb tokeni haldamisega brauseri LocalStorage'is. Saades tokeni AuthService moodulilt LoginAction salvestab saadud tokeni brauseris. LocalStorage'i eelis küpsiste ees on tema suurem turvalisus ning suurem salvestusruum [25]. Välja logimisel LoginAction aga kustutab tokeni.

LoginStore on mõeldud autentimist vajavate komponentide poolseks kasutamiseks. See komponent tagastab brauserisse salvestatud tokeni, et komponendid saaksid seda kasutada päringute tegemiseks. Lisaks tokenile see komponent tagastab ka tokenisse lisatud kasutajanime.

AuthenticatedComponent on *wrapper* komponent, mida kasutatakse kõikide autentimist vajavate vaadete ümber. Ta küsib LoginStore'ist tokeni ning kui ta seda ei saa, suunab kasutaja edasi sisselogimislehele. Kui aga LoginStore tagastab tokeni, siis lisab AuthenticatedComponent selle oma alamkomponendile. AuthenticatedComponenti implementatsiooni on näha joonisel 11.

```
export default (ComposedComponent) => {
  return class AuthenticatedComponent extends Component {
    componentWillMount() {
      if (!LoginStore.isLoggedIn()) {
        browserHistory.push("/admin/login");
      }
      this.state = this._getLoginState();
    }
    componentWillUpdate() {
      if (!LoginStore.isLoggedIn()) {
        browserHistory.push("/admin/login");
      }
    }
    _getLoginState() {
      return {
        userLoggedIn: LoginStore.isLoggedIn(),
        user: LoginStore.user,
        jwt: LoginStore.jwt
      };
    }
    componentDidMount() {
      LoginStore.addChangeListener(this._onChange.bind(this));
    }
    _onChange() {
      this.setState(this._getLoginState());
    }
    componentWillUnmount() {
        LoginStore.removeChangeListener(this._onChange.bind(this));
    }
    render() { return (
      <div>
        <ComposedComponent
          {...this.props}
          user={this.state.user}
          jwt={this.state.jwt}
          userLoggedIn={this.state.userLoggedIn} />
      </div>
      );}
  }
};
```

Joonis 11: Autoriseerimise *wrapper* komponent

Selline lahendus võimaldab luua lihtsalt autentimist vajavat komponenti. Sellise komponendi implementatsiooni on näha joonisel nr 12.

```
export default AuthenticatedComponent(class AdminHome extends Component {

  render() {
    return (
      <div className="admin-content center-div">
        <h1 className="page-title">Admin control panel</h1>
        <p>Welcome home, {this.props.user.username}</p>
        <div className="admin-content-div"></div>
      </div>
    );
  }
});
```

Joonis 12: *Wrapper* componendi kasutamise näide

4 Hinnang töö tulemusele

Kuigi süsteem sai projekti lõpuks realiseeritud jäid üks algselt planeeritud funktsionaalne nõue implementeerimata:

1. Administraator peab veebiliidese kaudu saama tiime kustutada.

Analüüsi käigus jäi meeskonna kustutamise stsenaarium piisavalt põhjalikult läbi analüüsimata ning seetõttu tulid kustutamisega seotud probleemid välja alles arenduse jooksul. Nimelt meeskonna kustutamisel tekivad probleemid juba sooritatud katsetega: neid ei saa ära kustutada, sest sel juhul kaoks tudengite hindamise alus. Samuti ei saa neid jätta andmebaasi ilma viiteta tiimile, sest hiljem ei ole võimalik kokku viia katset ja katse sooritanud tudengeid. Seega otsustati arenduse käigus tiimide kustutamisest loobuda.

Väga palju aega kulus töö autoril autori jaoks tundmatute tehnoloogiate tundma õppimiseks: autori senine kokkupuude kasutajaliideste loomise raamistikega oli siiani peaaegu olematu. Teine suurem probleem oli, et ei saanud süsteemi tervikuna katsetada, sest testväljakut töö tegemise ajal ei olnud veel valminud. Samuti tuli oodata ka Kinect One kaamera saabumist. Seetõttu tuli testida paljuski puuduolevaid osasid emuleerides. Näiteks tuli luua kaamera puudumise tõttu programm, mis saadaks ROS-i sõnumitena eelnevalt tavalise kaameraga salvestatud pilte. Sellegipoolest said need takistused ületatud ning põhiline funktsionaalsus sai loodud.

Töö alguses paika pandud tööprotsess töötas suurepäraselt: klient oli kogu aja teadlik süsteemi arengust ning tehniliste keerukuste lahendamiseks ei kulunud seetõttu palju aega. Iganädalastest kohtumistest jäid ära üksikud, kuid protsess selle tõttu ei kannatanud.

5 Tulevik

Süsteemi analüüsil tekkis palju ideid, mis antud töö raames osutusid liiga mahukaks või nende lisamiseni lihtsalt ei jõutud. Käesolevas peatükis loetletakse neid.

1. Võimalus kuvada veebiliideses katse taasesituses lisaks pildi peal (*overlay*-na) erinevaid andmekihte kogutud andmete visualiseerimiseks. Näiteks roboti läbitud teekond või ülesandespetsiifilisi andmeid nagu mõne rajatähise läbimine.
2. Kõikide katsete vaadet võiks saada filtreerida ajavahemiku alusel.
3. Kindlasti tuleb süsteemi veel täisulatuses katsetada, kui vastavad väljakud saavad valmis.

6 Kokkuvõte

Antud töö eesmärgiks oli luua automaatne hindamissüsteem TTÜ ainele Robotite programmeerimine, mis suudaks lisaks hindamisele ka võimaldada katsete taasesitust.

Eesmärgi saavutamiseks analüüsiti süsteemi vajadusi ning pandi paika süsteemi nõuded. Seejärel algas arendus agiilsel meetodil kaasates klienti arenduse protsessi võimalikult palju.

Töö tulemusena valmis neljakihiline süsteem, mis suudab lisaks hinnangu andmisele katseid salvestada videopildina ning katse soorituse kohta koguda metaandmeid. Süsteem on võimeline videosalvestust taasesitama ning kuvama kogutud andmeid veebikeskkonnas. Valmimata jäid ainult mõned algselt planeeritud vaated.

Töö võimalikuks edasiarenduseks on lisada süsteemile võimalus salvestada metaandmeid pildiliselt ning seejärel neid kuvada taasesituskeskkonnas kihtidena katse salvestuse peal.

Töö käigus saavutati eesmärk luua automaatne hindamissüsteem, mis võimaldaks paremini läbi viia ainet Robotite programmeerimine.

Kasutatud kirjandus

[1] The APRIL Robotics Laboratory at the University of Michigan, AprilTag Visual Fiducial System [WWW] https://april.eecs.umich.edu/software/apriltag.html (16.05.2017)

[2] ROS vahevara dokumentatsioon [WWW] http://www.ros.org/about-ros/ (16.05.2017)

[3] Kinect for Xbox One dokumentatsioon [WWW] http://www.xbox.com/en-US/xbox-one/accessories/kinect (16.05.2017)

[4] Kinect for Xbox One ROS-i võrku ühendamise teegi dokumentatsioon [WWW] https://github.com/code-iai/iai_kinect2 (16.05.2017)

[5] Cmake'i koduleht [WWW] https://cmake.org/ (16.05.2017)

[6] Catkini dokumentatsioon [WWW] http://docs.ros.org/api/catkin/html/ (16.05.2017)

[7] OpenCV koduleht [WWW] http://opencv.org/ (16.05.2017)

[8] SFTP kirjeldus SSH protokolli kodulehel [WWW] https://www.ssh.com/ssh/sftp/ (16.05.2017)

[9] SSH protokolli koduleht [WWW] https://www.ssh.com/ (16.05.2017)

[10] Sacha Greif, Front-end Frameworks [WWW] http://stateofjs.com/2016/frontend/ (16.05.2017)

[11] Angulari koduleht [WWW] https://angular.io/ (16.05.2017)

[12] Reacti koduleht [WWW] https://github.com/facebook/react (16.05.2017)

[13] React-Routeri dokumentatsioon [WWW] https://github.com/reacttraining/react-router (16.05.2017)

[14] Axios teegi dokumentatsioon [WWW] https://github.com/mzabriskie/axios (16.05.2017)

[15] Node.js koduleht [WWW] https://nodejs.org/en/ (16.05.2017)

[16] Express.js koduleht [WWW] https://expressjs.com/ (16.05.2017)

[17] Springi koduleht [WWW] https://spring.io/ (16.05.2017)

[18] Sequelize teegi koduleht [WWW] http://docs.sequelizejs.com/en/v3/ (16.05.2017)

[19] JSON web tokeni spetsifikatsioon [WWW] https://jwt.io/ (16.05.2017)

[20] JSON web tokeni implementatsioon Node.js-ile [WWW] https://github.com/auth0/node-jsonwebtoken (16.05.2017)

[21] And design React komponentide koduleht [WWW] https://ant.design/ (16.05.2017)

[22] PostgreSQL koduleht [WWW] https://www.postgresql.org/ (16.05.2017)

[23] Microsoft SQL serveri dokumentatsioon FileStream andmetüübi kohta [WWW] https://msdn.microsoft.com/en-us/library/bb933993(SQL.100).aspx (16.05.2017)

[24] Microsoft SQL serveri *Express edition*i omadused [WWW] https://www.microsoft.com/en-us/sql-server/sql-server-editions-express (16.05.2017)

[25] Scrypti implementatsioon Node.js-ile [WWW] https://github.com/barrysteyn/node-scrypt (16.05.2017)

[26] Scrypti selgitus leheküljel Gibson Research Corporation [WWW] https://www.grc.com/sqrl/scrypt.htm (16.05.2017