TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kristjan Karama 179448IADB

Rakendusliidese arendamine Eesti Jalgpalli Liidule

Bakalaureusetöö

Juhendaja: Nadežda Furs-Nižnikova
Magistrikraad

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: Kristjan Karama

07.01.2021

Annotatsioon

Antud bakalaureusetöö eesmärgiks on luua rakendusliides Eesti Jalgpalli Liidu jaoks. Rakendusliides peab edastama informatsiooni kõigi Eesti jalgpalli süsteemi kuuluvate liigade kohta.

Lõpuks valmiv rakendusliides peab tagama, et Eesti Jalgpalli Liidu andmebaasis olevad andmed oleksid masintöödeldaval kujul kättesaadavad huvilistele. Täpsemalt peaks liidest võimalik kasutusele võtta EJL jaoks loodud mobiilirakendusel ja Eesti Jalgpalli süsteemi kuuluvatel klubidel.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 41 leheküljel, 5 peatükki, 9 joonist, 2 tabelit.

Abstract

API Development for the Estonian Football Association

The aim of this bachelor's thesis is to create an application interface for the Estonian Football Association. The API must provide information about all the leagues that belong in the Estonian football system.

Finally, the completed application interface must ensure that the data in the database of the Estonian Football Association would be available for interested parties in a machine-readable form. The API should be available to use for the Estonian Football Association mobile application and clubs that play in the the Estonian Football system.

The thesis is in Estonian and contains 41 pages of text, 5 chapters, 9 figures, 2 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides
BLL	*Business Logic Layer*, äriloogika kiht
DAL	*Data Access Layer*, andmekiht
DTO	*Data transfer object*, andmeedastusobjekt
EJL	Eesti Jalgpalli Liit
FIFA	*Fédération Internationale de Football Association*, Rahvusvaheline Jalgpalliliit
HTTP	*HyperText Transfer Protocol*, hüperteksti edastusprotokoll
HTTPS	*Hypertext Transfer Protocol Secure*, turvaline hüperteksti edastusprotokoll
JSON	*JavaScript Object Notation*, andmevorming/süntaks andmete salvestamiseks ning vahetamiseks
LINQ	*Language-Integrated Query*, objektorienteeritud päringukeel
ORM	*ObjectRelational Mapping*, objekt-relatsiooniline kaardistamine
SQL	*Structured Query Language*, struktureeritud päringukeel
UEFA	*Union of European Football Associations*, Euroopa Jalgpalliliit
UOW	*Unit Of Wo*rk, tööühik

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Märkimisväärne osa inimeste igapäevasest digitaalsest kogemusest põhineb rakendusliidestel ehk API-del. Tegemist on niinimetatud liidestega, mis võimaldavad kahel erineval süsteemil andmeid omavahel jagada. Näiteks sularaha pangaautomaadist võttes suhtleb automaat läbi API pangaga, kontrollimaks kas kontol on piisavalt vahendeid tegevuse sooritamiseks.

Seoses Eesti Jalgpalli Liidule mobiilirakenduse arendusega tekkis samuti vajadus tagada viis, kuidas kaks osapoolt üksteisega suhelda saaksid. Antud lõputöö keskendubki sellele, kuidas rakendusliidese loomine tagaks ligipääsu Eesti Jalgpalli Liidu andmebaasis olevatele andmetele. Lisaks sellele, et mobiilirakendus saaks endale vajamineva informatsiooni liidese abil kätte, peaks seda võimalik teha olema ka teistel osapooltel. Teiste huviliste all on eelkõige mõeldud Eesti jalgpalli süsteemi kuuluvaid klubisid.

Töö esimeses põhiosa peatükis antakse ülevaade ettevõttest, kellele rakendusliides valmistatakse ja kirjeldatakse ära põhifunktsionaalsus, mida API peab täitma. Järgnevalt analüüsitakse eksisteerivaid jalgpalliga tegelevaid API-sid ja antakse ülevaade tehnoloogidest, mida arenduses kasutatakse ja põhjendatakse valikuid. Neljandas peatükis kirjeldatakse lahenduse realiseerimist, kus on välja toodud olemi-suhte diagramm, kirjeldatakse ära rakenduse arhitektuur ja viis kuidas rakendust optimeeriti ja testiti. Viimane põhiosa peatükk keskendub rakenduse edasiarendusele.

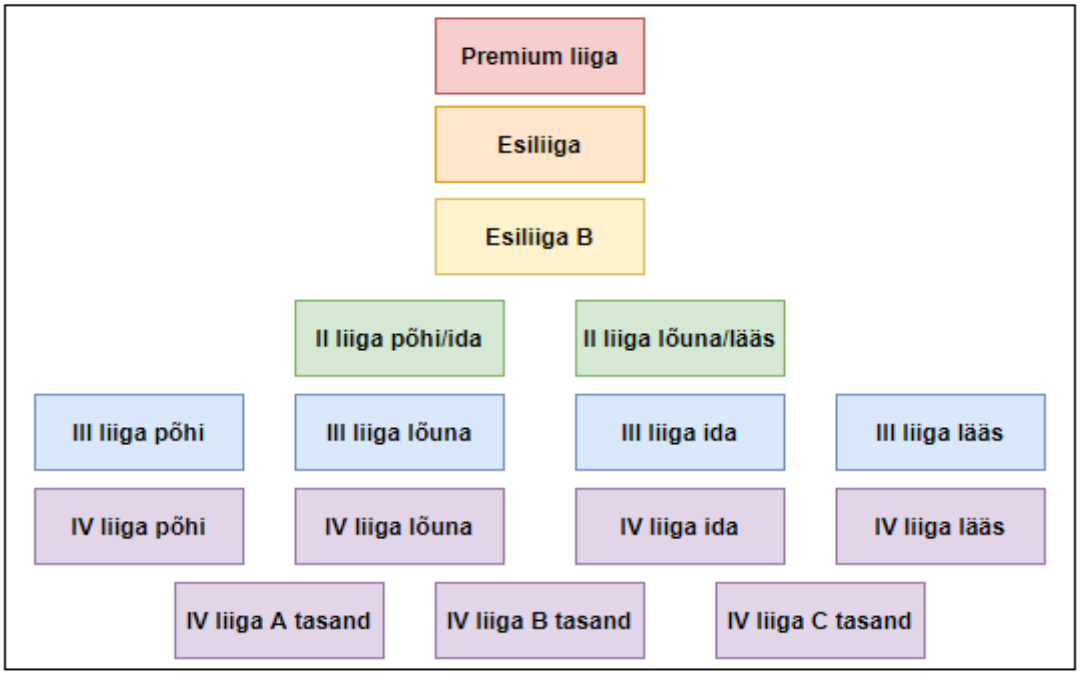
2 Probleemi püstitus

Käesolevas peatükis tutvustab autor organisatsiooni, kellele rakendusliidest arendatakse ja millega ühing tegeleb. Antakse ülevaate projekti eesmärgist, selle funktsionaalsusest ning autori rollist.

2.1 Valdkonna tutvustus

Rakendusliides arendatakse Eesti suurimale spordiorganisatsioonile. Eesti Jalgpalli Liit on 14. detsembril 1921 asutatud mittetulundusühing, mis korraldab rahvuskoondise tööd ja nende kodumänge, viib läbi Eesti täiskasvanute ja noorte meistrivõistlusi ning erinevaid karikavõistlusi, koolitab kohtunikke ja treenereid, litsentseerib klubisid ja mängijaid, töötab välja regulatsioone ning tegeleb infrastruktuuri arendamisega [1].

Täpsemalt tegeldakse jalgpallivõistluste korraldamisega järgmistel tasemetel: mehed, naised, noored, karikavõistlused, rahvaliigad, saali- ning rannajalgpall. Omakorda on jaotatud iga tase tugevuse, vanuse või geograafilise piirkonna põhjal liigadeks.



Joonis 1. Meeste jalgpalli liigade püramiid

Joonisel on kujutatud missugune näeb välja liiga süsteemi meeste jalgpalli puhul. Lisaks osalevad liigades mängivad meeskonnad kolmel karikavõistlustel, milleks on Tipneri karikavõistlus, Superkarikas ja Väike karikas. Samuti leiab iga liiga raames aset ülemineku mängud, võitjad liiguvad tase kõrgemale ja kaotajad langevad tase madalemale. Taolise loogika järgi on korraldatud ka teiste tasemete hooajalised võistlussarjad.

2.2 Eesmärgi püstitus

Hetkel puudub universaalne rakendusliides, mille abil oleks võimalik saada informatsiooni kogu Eesti jalgpallis toimuva kohta. Eksisteerivad liidesed on välja arendatud välismaa ettevõtete poolt ja nende skoopi ei kuulu kõik Eesti jalgpalli liigad ning rahvusvahelised võistlused, kus Eesti rahvuskoondised osalevad.

Antud töö eesmärgiks ongi luua API, mis edastaks masintöödeldaval kujul huvilistele Eesti Jalgpalli Liidu andmebaasis olevad andmed.

Töö skoopi kuulub ülevaade valitud tehnoloogiatest, lahenduse realiseerimine, rakenduse esmane optimeerimine, turvalisus ja testimine.

2.3 Autori roll

Autori ülesandeks on välja arendada mainitud rakendusliides. Arendusega toimetulekule aitab kaasa töö koostaja varasem API arenduse kogemus. Samuti tuleb kasuks autori huvi jalgpalli vastu ja eelnevad teadmised antud spordiala kohta. Just eksisteerivate eelteadmiste omamine aitab autoril oluliselt aega kokku hoida ja võimaldab keskenduda olulisemaile. Ei ole vaja hakata jalgpalli reeglistikuga tutvuma.

Lisaks töö koostajale on projekti kaasatud teised Eesti Jalgpalli Liidu IT osakonna ja teiste osakondade töötajad.

2.4 Rakenduse funktsionaalsus

Rakenduse funktsionaalsed nõuded tulenevad klientrakendusest, mille jaoks rakendusliides välja arendatakse. Seega andmed, mis kasutajale kättesaadavad peaksid olema saab struktureerida mobiilirakenduses planeeritavate vaadete põhjal. Järgnevalt on välja toodud peamised vaated.

Liiga vaade:

- avalehel peab kuvama tulevasi/reaalaja mänge ja uudiseid;
- liiga tabel;
- tulevased ja toimunud mängud;
- liiga statistika.

Võistkonna vaade:

- tulevased ja toimunud mängud;
- võistkonna statistika;
- võistkonna mängijad;
- andmed klubi kohta.

Mängija vaade:

- üldinfo mängija kohta;
- statistika mängija kohta;
- mängija mängitud mängud.

Koondise vaade:

- avalehel peab kuvama tulevasi/reaalaja mänge, uudiseid ja koondise statistika;
- turniiri tabel;
- tulevased ja toimunud mängud;
- koosseis ja personal.

Loetletud vaadete juures on välja toodud peamine funktsionaalsus, mida antud vaates teha või nägema peaks. Mainitud osade abil saab kursis olla tähtsamaga: võimalik teada kuidas meeskondadel liigas läheb, kuidas kulgeb mängijate hooajad ja saab aimu rahvuskoondiste olukorrast.

3 Valdkonna ja tehnoloogiate analüüs

Enne kui ise rakendusliidest arendama hakata, oleks mõistlik tutvuda missugused teenused on avalikult kättesaadavad. Lähemalt tuleks uurida kui laialdane on nendes Eesti Jalgpalli süsteemi kuuluvate liigade/turniiride kajastus, mitut rahvusvahelist võistlussarja on API kaudu võimalik jälgida ning missugune on nende poolt pakutav üldine funktsionaalsus.

3.1 Eksisteerivad lahendused

3.1.1 API-football

Tegemist on ühe populaarseima jalgpalli kajastava rakendusliidesega. Liidese kaudu on võimalik informatsiooni saada üle 690 jalgpalli liiga ja turniiri kohta. Samas Eesti jalgpallisüsteemi kajastamiseks on info kättesaadav, vaid Superkarika, Premium liiga ja Esiliiga jaoks [2].

Rakendusliidesest saab kätte liiga tabeli, mängud, sündmused, koosseisud, statistika ja olemas on info reaalajas toimuvate mängude skooride ja sündmuste kohta. Samuti on saadaval andmed, mis on ühe või teise meeskonna võiduvõimalused ja kuidas on kulgenud nende kunagised omavahelised mängud [3].

3.1.2 iSports API

iSports on ettevõte, mis on väljastanud juba viimased 12 aastat andmeid mitmete spordialade jälgimiseks. Nende jalgpalli andmevoog hõlmab rohkem kui 1600 rahvusvahelist liigat. Edastatakse infot liigas toimuvate mängude, reaalaja skooride ja liiga tabeliseisude kohta. Selle API kaudu on võimalik informatsiooni saada Eesti 11 liiga kohta. Lisaks saab jälgida kuidas kulgevad olulisemad rahvusvahelised jalgpalli turniirid[4].

3.1.3 API football

API football reklaamib end kui soodne ja lihtsasti kasutatav rakendusliides. Võimalik on saada informatsiooni reaalajas toimuvate mängude kohta, näha meeskondade paiknemist liigatabelis ja jälgida kui hästi või halvasti mängijatel antud hooajal läheb. Mainitud andmed on saadaval nelja olulisema Eesti liiga kohta ning erinevate vanusegruppide rahvusvaheliste võistluste kohta [5].

3.1.4 Eksisteerivate rakenduste analüüsi tulemus

Analüüsist selgus, et eksisteerivatel rakendusliidestel on olulisi puudujääke. Esiteks on Eesti jalgpalli liigade kajastus väga kesine. Andmed edastatakse peamiselt ainult paari tähtsama liiga/turniiri kohta ehk API kaudu ei ole võimalik saada informatsiooni Eesti madalamates liigades toimuva kohta. Rääkimata siis rahvaliigast, rannajalgpallist ja saalijalgpallist. Samuti puuduvad andmed erinevate vanuserühmade rahvusvaheliste turniiride kohta.

Vaadeldud rakendusliidesed on kirjutatud pigem liigade vaatenurgast, puudub põhjalik ülevaade liigas mängivatest meeskondadest. Näiteks puudub info, mis mängijad täpsemalt sellel hooajal meeskonna koosseisu kuuluvad ja missugune näeb välja käimasoleval hooajal võistkonna statistika. Samuti edastatav info mõnevõrra puudulik või ei vasta see sageli tegelikkusele, nt ei ole võimalik näha mängus osalevate meeskondade formatsiooni või on selleks määratud kõige levinud mängijate asetus 4-4-2. Lisaks sellele ei ole ükski vaadeldud API arvestanud EJL reeglitest ja jalgpalli liigade korraldustest tingitud iseärasustega. Näiteks on kombeks mõned liigad hooaja keskel paremas järjestikku järgi niiöelda lahku lüüa kas siis kaheks või rohkemaks eraldiseisvaks divisioniks.

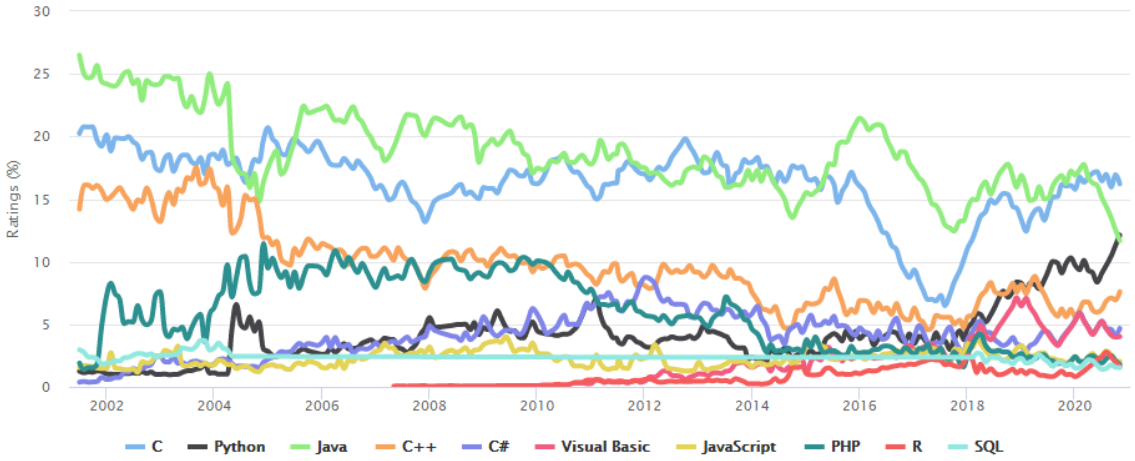
Lisaks eeldaks iga rakendusliidese kasutuselevõtmine igakuist väljaminekut. Hind kujuneks sõltuvalt mitut liigat soovitakse jälgida ja kui palju päringuid ühe kuu jooksul API vastu teostatakse.

Tuginedes kõigele eelnevalt mainutule osutus mõistlikuks ja otstarbekaks ise välja arendada rakendusliides lähtuvalt Eesti Jalgpalli Liidu andmebaasis olevatest andmetest. Isearendatud lahendus tagaks, et võimalik on jälgida kõiki Eesti Jalgpalli süsteemi kuuluvaid liigasid, tagaks parema andmete kvaliteedi ja võimaldab arvestada liigade korraldusest tingitud erandjuhtumitega.

3.2 Tehnoloogiate valikud

3.2.1 Programmeerimiskeele ja raamistiku valik

Programmeerimiskeeli, milles antud rakendus kirjutada on palju, täpsemalt eksisteerib maailmas ligikaudu üle 700 erineva keele, millest omakorda 50 on laialdasemalt kasutusel [6]. Kuna rakendusele on eelnevalt määratud tellija poolt tähtaeg, siis võetakse lähemalt vaatluse alla programmeerimiskeeled, millel on suur kasutajaskond ja millega on autor juba varasemalt kokku puutunud.



Joonis 2. Populaarsemad programmeerimiskeeled TIOBE indeksi kohaselt [7]

Potentsiaalsete programmeerimiskeelte kohta, mille abil rakendust võiks arendada koostas autor tabeli, kus hindas enda kogemust keeles ja ajakulu, mis kuluks antud keeles enda täiendamiseks. Töötundide hindamisel on lähtutud eelnevast kogemusest, mis on kulunud end programmeerimiskeelega kurssi viimaseks. Tavaliselt on see periood jäänud kahe nädala juurde. Kogemus hea tähendab, et autor on vähemalt aasta selles programmeerimiskeeles arendanud, keskmise kogemuse all mõeldakse kuue kuulist perioodi ja nõrk tähistab kolme kuud.

Tabel 1. Programmeerimiskeelte võrdlus

Keel	**Kogemus**	**Töötunnid keele õppimiseks**
Python	Nõrk	80
PHP	Keskmine	40
Java	Hea	0
C#	Hea	0
Javascript	Nõrk	80

Lisaks sellele, et on võrreldud omavahel erinevaid programmeerimiskeeli peab autor mõistikuks koostada samasugune tabel nende keeltega seotud populaarsemate raamistike kohta, milles võiks rakendust arendada. Projektis raamistiku kasutamine võimaldab mingeid tegevusi lihtsustada ning pakub arhitektuurilist tuge. Võrreldes keelte õppimisega kulub ühe raamistiku tundma õppimiseks rohkem aega, lähtudes enda kogemusest on selleks perioodiks vähemalt kolm nädalat, mis on üpris optimistilik hinnang.

Tabel 2. Raamistike võrdlus

Keel	**Raamistik**	**Kogemus**	**Töötunnid raamistiku õppimiseks**
Python	Django	Puudub	120
PHP	Laravel	Puudub	120
Java	Spring Boot	Keskmine	40
C#	ASP.NET Core	Hea	0
Javascript	Express.js	Puudub	120

Antud tulemuste põhjal selgub, et mõistlik oleks arendada rakendus kasutades selleks C# või Java programmeerimiskeelt, kuna autoril ei kuluks töötunde enda täiendamisele antud keeltes. Samuti on autor varasemalt juba kokku puutunud mõlema keele raamistikega, mida saab kasutada hea arhitektuuri tavadega rakendusliidese arendamiseks. Lõplik valik osutub ASP.NET Core raamistiku kasuks, kuna autor hindab arendusprotsessi ja enda taset selles raamistikus paremaks.

ASP.NET Core puhul on tegemist avatud lähtekoodiga veebiraamistikuga, mis võimaldab luua multiplatvormseid veebirakendusi, kasutades C# programmeerimiskeelt. Microsofti poolt arendatud raamistik on disainitud spetsiaalselt keskmiste ja suurte rakenduse

arendamiseks. Põhiline arenduskeskkond on Visual Studio Windows platvormil - kuid toetatud on ka kõik teised platvormid (Visual Studio Code, Visual Studio for Mac). Projekti raames on plaanis kasutusele võtta viimane stabiilne ASP.Net Core 3.1 versioon [8].

Raamistikku puhul hindab töö koostaja võimalust päringute teostamiseks kasutada objektorienteeritud päringukeelt LINQ-u. Antud keel võimaldab objektide kaudu SQL päringuid teostada lihtsamalt ja kiiremalt, pakkudes päringute kirjutamisel vihjeid ning kompileerimisel kontrollib päringu süntaksi [9]. Samuti saab mugavalt projekti vajaminevaid pakette kaasata kasutades selleks paketihaldurit nimega NuGet. Hetkel on laiemalt avalikkusega jagatud üle 200 000 tuhande unikaalse paketi. Lisaks on võimalik enda kood pakendada NuGet paketiks ja seda projektis implementeerida [10].

3.2.2 Andmebaasi valik

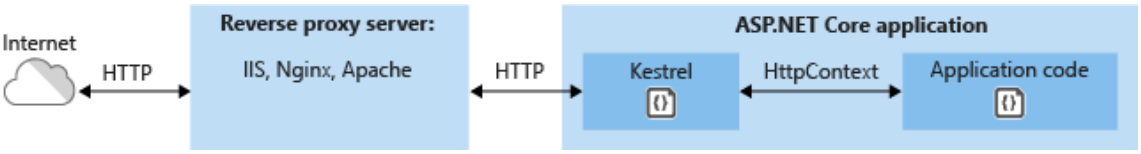
Kuna arendatav rakendus kirjutatakse juba olemasoleva MySQli andmebaasi vastu, siis tuleneb andmebaasi valik projekti nõudest. Täpsemalt on hetkel kasutusel andmebaasi versioon 10.2.32.

3.2.3 Rakenduse majutus

Lõpuks valmiv ASP.NET Core rakendus on plaanis paigaldada Zone Media OÜ poolt pakutavasse privaatserverisse. Mainitud serveriplatvorm osutus valituks, sest Eesti Jalgpalli Liit kasutab juba pikemalt aega Zone domeeni- ja serveriteenuseid. Seega oleks tagatud, et kõikide rakenduste haldus käiks läbi ühe platvormi.

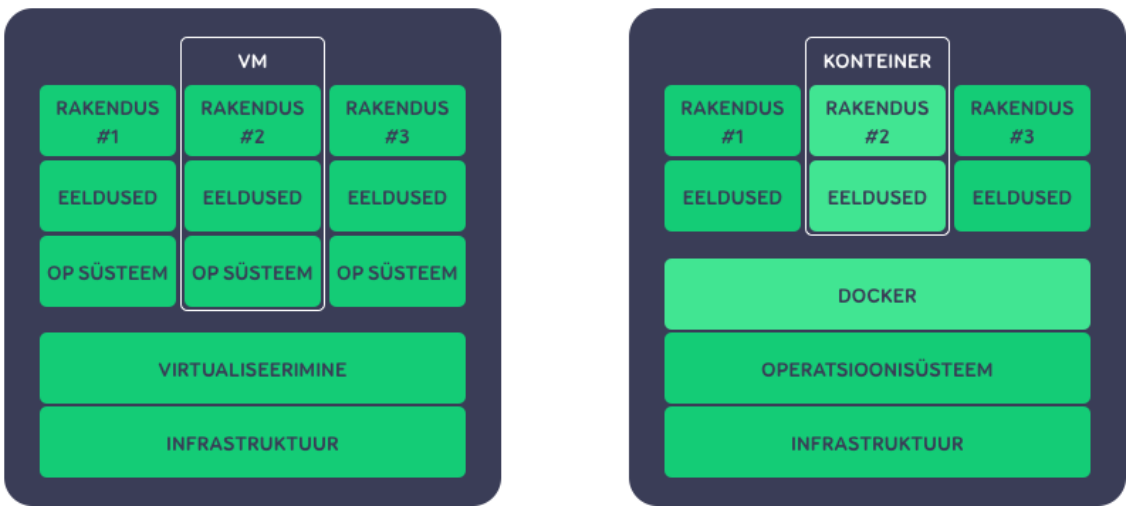
Serverisse seadistatakse Apache veebiserver. Tegemist on ühe maailma kõige populaarseima veebiserveriga, mis on saadaval pea igale riistvarale ja operatsioonisüsteemile. Nende dokumentatsioonist leiab arvukalt näited ja tekkinud vigade korral on võimalik internetist lihtsasti abi saada. Apache just võrdluses maailmas järjest suuremat populaarsust koguva Nginx serveriga osutus valituks seoses selle laialdase leviku ja suurema kasutajaskonna tõttu. Nginx veebiserverit peetakse enamus juhtudel mõnevõrra kiiremaks, kuid antud rakenduse kontekstis ei mängi minimaalsed jõudluse parandused suurt rolli [11].

Apache veebiserver täidab antud projektis pöördproksi rolli. Pöördproksi võtab päringuid klientidelt vastu ja suunab need edasi ASP.NET Core rakendusele. Samuti edastab pöördproksi serverilt saadud vastused klientidele tagasi. Proksi tagab, et rakendus ei suhtle otse kliendiga [12]



Joonis 3. Rakenduse poole pöördumine [13]

Rakenduse paigaldamise lihtustamiseks kasutatakse Dockerit, mis võimaldab rakenduse kokku pakkida konteineriks. Docker on tööriist, mis on loodud konteinerite abil rakenduste loomise, juurutamise ja käivitamise hõlbustamiseks. Konteinerid võimaldavad arendajal pakendada rakenduse koos kõigi vajalike osadega, näiteks teegid ja muud sõltuvused, ning juurutada selle ühe paketina. Sellisel juhul saab arendaja tänu konteinerile olla kindel, et rakendus töötab mis tahes muus masinas [14].



Joonis 4. Tehniline erinevus virtualiseerimise ja konteinerite vahel [9]

Virtualiseerimine on eraldatud kiht riistvara pealt, mis muudab ühe serveri mitmeks serveriks. Virtualiseerimine võimaldab jooksutada ühel masinal mitut virtuaalmasinat.

Iga virtuaalmasin on omaette operatsioonisüsteemi koopia, mis võib kokku võtta 10+ GB ruumi ning ka käivitumine on üldjuhul päris aeglane [14].

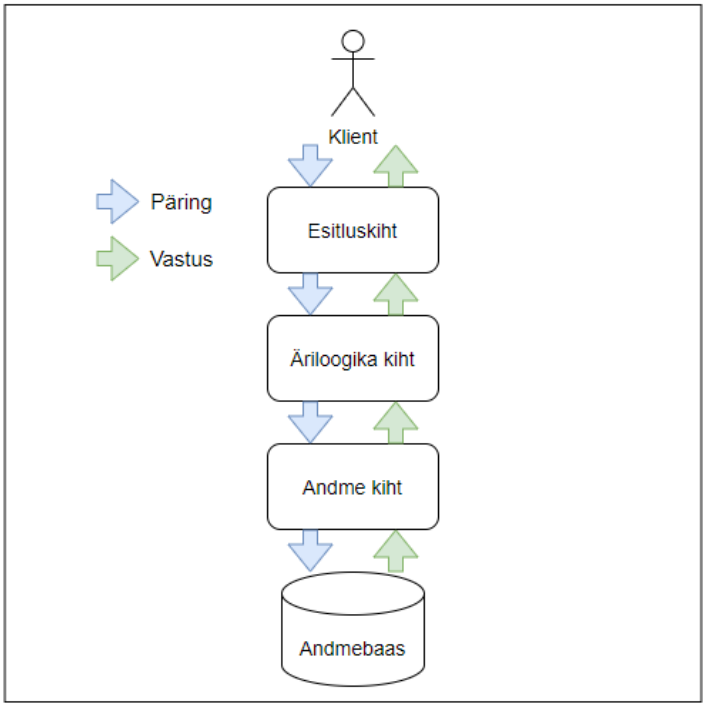
Konteiner on teisalt poolt aga eraldatud kiht, mis koosneb ainult rakenduse koodist ning selle eeldustest. Ühel masinal võib joosta mitmeid konteinereid ning need jagavad sama masina operatsioonisüsteemi komponente, jooksutades igat konteinerit oma isoleeritud keskkonnas. Konteinerid üldjuhul võtavad väga vähe ruumi, jäädes 100MB ringi ning käivituvad pea koheselt [14].

3.3 Rakenduse arhitektuur

Loodavas ASP.NET Core rakenduses on plaanis rakendada mitmekihilist arhitektuuri, kus esitluse, töötlemise ja andmete kättesaamise protsessid on üksteisest loogiliselt eraldatud. Mitmekihiline arhitektuurimudel aitab luua paindlikku ja korduvalt kasutavat tarkvara. Muudatuste puhul on vajalik need teha vaid üksikutes kihtides, mitte kogu rakenduses korraga. See lubab hakkama saada vähema töö, lühema aja ja väiksema potentsiaalsete vigade hulgaga [15].

Mitmekihiline arhitektuur on üldistus. Siin võivad vastavalt vajadusele lisanduda erinevad kihid või jaotuda kirjeldatud kihid omakorda osadeks. Loodava rakenduse puhul oleks kasutusel kolmekihiline arhitektuur, mis on üks tüüpilisem ja enim kasutatavam viis kuidas rakendust struktureerida. Rakendus koosneb sellisel juhul andmekihist, äriloogika kihist ja esitluskihist [15].

- andmekiht – tegeleb andmebaasi suhtlusega. Teostab päringuid andmebaasi vastu ja tagastab saadud tulemused teistele kihtidele;
- äriloogika kiht – kihis on ära kirjeldatud äriloogikast tulenevad reeglid ja funktsionaalsuse piirangud;
- esitluskiht – kliendile kõige lähemal olev kiht [16].



Joonis 5. Kolmekihiline arhitektuuri mudel [17]

Antud joonis illustreerib missugune näeb välja suhtlus erinevate kihtide vahel. Täpsemalt kuidas liiguvad andmed, kui klient teostab päringu rakenduse vastu.

4 Lahenduse realiseerimine

Neljandas peatükis kirjeldab autor mõnede töö osade realiseerimist. Esiteks antakse ülevaade andmebaasi ehitusest ja kirjeldatakse kuidas koodi mõistes implementeeritakse mitmekihiline arhitektuur. Järgnevalt keskendutakse rakenduse optimeerimisele, turvalisusele ja testimisele.

4.1 Andmebaasi mudel

Andmemudelite kirjelduse esitamiseks on koosatud olemi-suhte diagramm. Skeem on omakorda jaotatud kaheks osaks - funktsionaalseteks vaadeteks. Üks osa skeemist kirjeldab ära olemid, mis on vajalikud koondise funktsionaalsuse välja töötlemiseks (ja teine keskendub konkreetselt iga distsipliiniga (muru, saali- ja rannajalgpalliga) seotud liigadele. Mõlema skeemiga on võimalik tutvuda Lisa 1 ja Lisa 2.

4.2 Domeen

Domeeni projektis on ära defineeritud kõik olemid, mis on rakendusliidese loomiseks vajalikud. ASP.NET Core puhul hoolitseb ORM teostamise eest Entity Framework, mis seob C# kirjutatud klassid andmebaasi tabelite ja veergudega. Kuna autor ei soovi kasutada andmebaasis defineeritud tabelite ja veergude nimesid, siis on oluline defineerida annotatsiooniga täpsemalt, mis tabel tuleb antud klassiga ära siduda [18].

```
[Table("ejl_mobile_video")]
public class Video : DomainEntity
{
        [Column("league_id")]
        public int LeagueId { get; set; }

        [Column("video_id")]
        public string VideoId { get; set; }

        [Column("channel_title")]
        public string ChannelTitle { get; set; }

        [Column("video_title")]
        public string VideoTitle { get; set; }

        [Column("video_thumbnail")]
        public string VideoThumbnail { get; set; }
}
```

Joonis 6. ORM teostamine

Joonisel on kujutatud missugune ORM teostamine koodi mõistes välja näeb. Lisaks on pildilt näha, et domeeni objekt täidab DomainEntity liidest, milles on defineeritud idenitifikaatori väli. Kuna iga andmebaasis oleval tabelil on määratud unikaalne id veerg, siis täidavad kõik domeeni objektid seda liidest.

4.3 Andmekiht

Rakenduses loodud DAL (*Data Access Layer*) projekti eesmärgiks teostada suhtlus andmebaasiga. Andmekihis on kasutusele võetud repositooriumite muster (*Repository pattern*). Anmebaasis olevatele tabelitele on loodud vastavad repositoorimud, kuhu on koondatud kõik andmebaasiga toimingud. Klassis sisaldavad endas andmebaasi päringuid, mis täidavad rakenduse funktsionaalseid nõudeid. Päringute teostamiseks on kasutusel objektorienteeritud päringukeel LINQ. Iga repositoorium põhineb baasrepositooriumil ja täidab liideses defineeritud meetodeid [19].

Andmebaasist välja küsitud objektid mappitakse ümber andmeedastus objektideks ehk DTO-deks, mis seejärel saadetakse edasi järgmisse kihti.Iga kiht tegeleb konkreetselt selles kihis defineeritud objektidega. Töö koostamisel on eelistatud objektide mappimine teostada manuaalselt selle asemel, et Automapper teek kasutusele võtta. Automapper nagu nimigi annab teada automatiseerib ühest objketist teise konverteerimise, et arendaja ei peaks selle peale aega kulutama [20]. Samas on autor seisukohal, et implementeerides

kõik mappimised ise on parem ülevaade sellest, mis atribuudid omavahel ära seotakse ja vigade korral on need kergemini tuvastatavad. Taoline manuaalne lähenemine on küll ajakulukam, kuid pikas perspektiivis võib aega hoopis kokku hoida.

```
public class VideoRepository : BaseRepository<DAL.App.DTO.Video, Domain.Video, AppDbContext>, IVideoRepository
{
        public VideoRepository(AppDbContext repositoryDbContext) : base(repositoryDbContext, new VideoMapper())
        {
        }

        public async Task<List<Video>> GetLandingPageVideos()
        {
                var query =  await RepositoryDbSet
                        .AsNoTracking()
                        .Select(a => VideoMapper.MapFromDomain(a)).ToListAsync();

                return query.GroupBy(a => a.LeagueId).SelectMany(e => e.Take(2)).ToList();
        }

        public async Task<List<Video>> GetVideosByLeagueId(int leagueId)
        {
                return await RepositoryDbSet
                        .AsNoTracking()
                        .Where(s => s.LeagueId == leagueId)
                        .Select(a => VideoMapper.MapFromDomain(a)).ToListAsync();
        }
 }
```

Joonis 7. Repositooriumi ülesehitus

Andmebaasis tehtud muudatuste konfliktide ära hoidmise eest vastutab UOW klass (Unit of Work). Tööüksus klass peab meeles muudatused, mis ühe transaktsiooni ajal on aset leidnud ja teostab need juhul kui ühtegi probleemi ei esine. Konfliktide korral jäetakse kogu protsess pooleli ja andmebaasis ühtegi muudatust ei teostata [19].

4.4 Äriloogika kiht

BLL (Business Logic Layer) nimeline projekt vastutab rakenduse äriloogika eest. Eesmärgiks on teenus ehk *service* klassides hoolitseda funktsionaalsuse juhtimise eest, töödeldes selleks alumisest kihist saadud andmeid vastavalt kasutajalt ülemisest kihist tulnud päringutele. Juhul kui projektis pole koheselt tarvis vaja rakendada äriloogikat, tegeleb kiht lihtsalt objektide mappimisega ühest andmeedastus objektist teise.

4.5 Esitluskiht

Koodi mõistes koosneb esitluskiht kontroller klassidest. Kontrollerid tegelevad sissetulevate HTTP päringutega ja vastuse tagasi saatmisega kliendile, kes päringu teostas.

ASP.NET Core API Controllerite puhul on oluline:

- implementeeriks ControllerBase klassi, kus on defineeritud atribuudid ja meetodid mis on vajalikud HTTP päringute töötlemiseks. Oluline on tähelepanu pöörata, et kontroller ei kasutaks liidest nimetusega Controller, sest see sisalduv funktsionaalsus on mõeldud veebilehtedega tegelemiseks;
- annotatsioon ApiController, mis annab teada et kontroller tegeleb HTTP päringutega
- annotatsioon Route, mis kirjeldab ära kontrolleri routimise;
- juhul kui soovitakse kasutada API versiooniseerimist, siis on oluline annotatsiooniga ära tähistada, mitmenda versiooni funktsionaalsuse täitmisega kontroller tegeleb [21].

```
[ApiVersion("1.0")]
[Route("api/v{version:apiVersion}/[controller]")]
[ApiController]
public class VideosController : ControllerBase
{
        private readonly IAppBLL _bll;
        private readonly IMemoryCache _memoryCache;

        public VideosController(IAppBLL bll, IMemoryCache memoryCache)
        {
                _bll = bll;
                _memoryCache = memoryCache;
        }

        /// <summary>
        /// Get all landing page videos
        /// </summary>
        /// <returns>Array of landing page videos</returns>
        /// <response code="200">The array of videos was successfully retrieved.</response>
        [HttpGet]
        public async Task<ActionResult<List<Video>>> GetLandingPageVideos()
        {

                var videos
                        = await CacheHelper.GetOrSetCacheObject(
                        _memoryCache,
                        Request,
                        async () =>
                        (await _bll.Videos.GetLandingPageVideos())
                        .Select(PublicApi.v1.Mappers.VideoMapper.MapFromBLL).ToList(),
                        TimeSpan.FromSeconds(120)
                );

                if (videos == null)
                {
                        return NotFound();
                }

                return videos;
        }

        ...
}
```

Joonis 8. Kontrolleri ülesehitus

Lisaks kuulub iga kontrolleris defineeritud meetodi juurde annotatsioon, mis täpsustab mis tüüpi päringut meetod ootab ja täita oskab ning mis parameetrite saatmisega on võimalik selle poole pöörduda [21].

4.6 Rakenduse optimeerimine

4.6.1 Vahemälu

Vahemälu kasutamine rakenduses võib oluliselt parandada rakenduse jõudlust. Salvestamine vahemällu on eelkõige mõeldud harva muutuvate andmetega tegelemiseks.

Vahemälus hoiustatakse koopia andmetest, mida saab seejärel tagastada kiiremini kui algallikast andmeid küsides [22].

ASP.NET Core toetab mitut erinevat viisi kuidas vahemällu andmeid talletada:

- IMemoryCache - informatsioon salvestatakse veebiserveri vahemällu;
- ResponseCache - kliendile antakse juhised kuidas ja kui kaua andmeid meeles peaks hoidma;
- Distributed caching - andmete talletamise eest vastutab eraldiseisev rakendus või kasutatakse selleks mõnda teenust [22].

Kuna nõue on vähendada otsest koormust andmebaasile, vähendada tehtavate päringute hulka, siis otsustatakse kasutada IMemoryCachel põhinevat vahemälu. Vahemällu salvestamise puhul on oluline meetodis ära defineerida ajavahemik, kui kaua soovitakse päringust saadud vastus mälus talletada. Periood, kui kaua midagi meeles hoida sõltub kui oluline on andmete uudsus. Näiteks konkreetse rakenduse puhul meetodites, mis tegelevad andmete edastamisega reaalajas jalgpalli mängude kohta on defineeritud 30 sekundiline periood. Andmetele, mis on rohkem staatilise iseloomuga on määratud pikem ajahulk.

4.6.2 Cron tööd

Andmebaasi ja rakendusliidese koormuse vähendamiseks luuakse skriptid päringute jaoks, mis eeldavad suuremamahulisi arvutisi ja mida ei ole mõistlik iga päringu jaoks teostada. Selle asemel, et kogu arvutus iga päringu peale uuesti teha salvestatakse eelnevalt saadud tulemused andmebaasi tabelitesse ja vajadusel saab andmed välja küsida.

Täpsemalt on oluline vältida korduvat arvutamise järgmiste tegevuste kohta:

- mängijate statistika arvutused;
- liigas mängivate meeskondade külastatavuse arvutamine;
- liiga tabelite arvutused iga distsipliini jaoks.

Loodud koodijuppe võib vaadelda kui algoritme. Näiteks koondiste liiga tabelite koostamiseks vaadatakse järgmiseid tunnuseid kui kaks või enam sama grupi võistkonda on võrdsetel punktidel, et välja selgitada kumb võistkond peaks tabelis eespool paiknema:

1. omavaheliste mängude punkte;
2. omavaheliste mängude väravate vahet;
3. omavaheliste mängude suuremat löödud väravate arvu;
4. omavaheliste mängude suuremat võõrsil löödud väravate arvu.

Kui kõnealused meeskonnad on endiselt võrdses seisus teostatakse järgmine võrdlus kogu grupis peetud mängude põhjal ja arvestatakse:

1. üldist väravate vahet;
2. suuremat löödud väravate arvu;
3. suuremat võõrsil löödud väravate arvu;
4. suuremat löödud väravate arvu;
5. paremat distsiplinaarrekordit (kumb meeskond on saanud vähem kollaseid ja punaseid kaarte) [23].

Loodud skriptid seatakse üles Jalgpalli Liidule kuuluvasse serverisse, kus paiknevad kõik teised cron tööd. Niisuguste tööde all mõistetakse koodijuppe, mis teatud aja möödudes serveris jooksutakse [24]. Täpsema ajavahemiku määramine tuleneb sellest, kui uudseid andmeid vajatakse. Juhul kui on möödunud kindel periood, käivitatakse koodijupp ning juhul kui andmebaasi on sisestatud andmeid, mis võiksid mõjutada arvutuse lõpp resultaate teostatakse arvutus uuesti. Programmid luuakse kasutades selleks PHP programmeerimiskeelt ja SQL päringukeelt ehk kasutatakse samu tehnoloogiad, millel põhinevad eelnevalt üles seadistatud cron tööd.

4.7 Rakenduse turvalisus ja ligipääsetavus

ASP.NET Core raamistik pakub sisseehitatud funktsioone, mille abil on võimalik rakendust kaitsta ja tagada selle turvalisus.

SQL süstimise tuntuid kui ka SQL injectioni eest on võimalik rakendust kaitsta koostades kõik andmebaasi päringud kasutades selleks LINQ tehnoloogiat. LINQ hoolitseb selle eest, et päringud ei koosne sõnede manipuleerimise või liitmise abil, vaid andmed edastatakse andmebaasi parameetrite kaudu. Samuti on võimalik koostada tavalisi ehk niinimetatud *raw* SQL päriguid, kuid siis peab arendaja ise hoolitsema, et päringutes olevad muutujad oleksid parameetritega. [25].

Rakenduses *Startup* klassis on võimalik määrata vaikimisi HTTPS protokolli kasutamine tänu vahevarale, mille ülesandeks on HTTP päringud ümber suunata HTTPS-ile. HTTPS on turvaline protokoll autenditud ja krüpteeritud informatsiooni edastamiseks arvutivõrkudes. HTTPS-i puhul toimub andmevahetus kasutaja ja veebiserver vahel kõrvaliste jaoks loetamatul kujul ning ühendus on kaitstud vahendajarünnete eest [25].

Pöördproksi kasutusele võtmine aitab samuti kaasa rakenduse turvalisuse tagamiseks. Proksi abil ei pea teenus kunagi oma õiget aadressi avaldama. See omakorda muudab rakenduse keerulisemaks sihtmärgiks ründajatele. Näiteks DdoS rünnaku puhul on võimalik sihtmärgiks vaid seada proksi, mis sel juhul tegeleb aktiivselt pahaloomulise liikluse blokeerimisega [12].

Rakenduse kliendi autoriseerimiseks on loodud kaks erinevat võimalust. Kui päringu teostab mobiilirakenduse kasutaja lisatakse HTTP päisesse seadme token, mis on iga kasutaja puhul unikaalne. Token abil kontrollitakse kas kasutajal on luba juurde pääseda soovitud ressurssidele.

Teiseks on rakenduses üles seatud niinimetatud turvaliste IP aadresside ehk *safelisti* kasutamise võimalus. Sel juhul teostatakse konkreetses kontrolleris või meetodis IP aadressi kontrollimine. Kontrollitakse *kas appsettings.json* failis *AdminSafeListis* asuvas massiivis on olemas päringut teostanud IP aadress. Juurdepääs on lubatud, kui massiiv sisaldab IP aadressi, muul juhul tagastatakse olekukood HTTP 403 keelatud [26].

4.8 Testimine

Valminud rakendusliidesele on teostatud manuaalne testimine. Testimise käigus oli oluline kinnitada, et iga päring tagastab reaalselt seda, mida see peaks ning kontrollida, et päringust kättesaadavad andmed vastaksid tegelikkusele. Andmete õiguse kontrollimisel tuginetakse jalgpall.ee, FIFA ja UEFA lehekülgedel leiduvatele

resultaatidele. Enne igat testimist teostatakse suitsutestimine ehk pealiskaudne testimine eesmärgiga teha kindlaks, et poleks põhjalikumalt testimist takistavaid probleeme (näiteks programm ei käivitu või kõiki põhifunktsioone pole võimalik testida) [27].

Testimisel osalevad antud projektiga seotud arendajad ja EJL töötajad. Loodud rakendusliidest on võimalik testida manuaalselt päringuid teostades veebibrauseri kaudu või kasutades päringute tegemiseks Swaggeri poolt genereeritud graafilist rakendusliidest. Kuna loodud rakendusliidest hakkab kasutama mobiilirakendus, siis on võimalik hiljem ka selle vahendusel ning just klientrakenduse beeta testimise faasis veenduda liidese töökindluses.

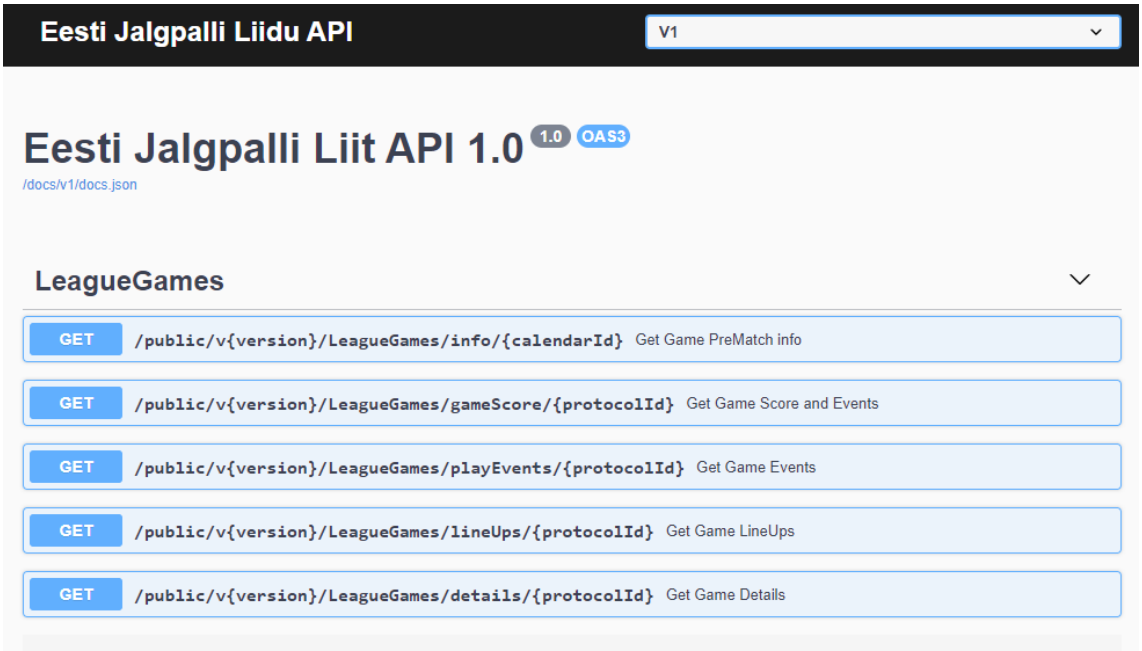
Peamised andmed mille õiguses tuli veenduda:

- mängija statistika arvutused sõltuvalt distsipliinist (mängude arv, minutid, kollased ja punased kaardid);
- liiga statistika arvutused (külastatavus kodus ja võõrsil)
- konfidentsiaalsete andmete mittekuvamine (noorte liigas mängu tulemused ja liigatabeli paremusjärjestus);
- korrektsete minutite märkimine väljavahetatud ja sisse toodud mängijatele;
- klubis treenitud mängijate staatuste õigesti kuvamine.

Kuna üldpildis oli rakendusliides juba arendajate poolt erinevate sisenditega juba põhjalikult testitud, siis EJL töötajate peamiseks ülesandeks oli ära kaardistada/meelde tuletada erandjuhtumid, millele tasuks tähelepanu pöörata.

4.9 Dokumentatsioon

Loodud rakendusliidese dokumenteerimiseks on kasutatud Swagger raamistikku. Swagger võimaldab automatiseerida dokumentatsiooni loomist luues kontrolleris defineeritud meetodite põhjal graafilise liidese. Loodud interaktiivse API konsooli abil saab ülevaate rakendusliidese funktsionaalsusest ja võimaldab mugavalt päringuid *endpointide* vastu teostada. Seega lisaks dokumentatsioonile saab Swaggerit kasutada kui tööriista testimaks loodud funktsionaalsust [28].



Joonis 9. Swagger dokumentatsioon

Nagu jooniselt näha võimaldab Swaggeri poolt loodud graafiline liides saada hea ülevaate API-st ja päringutest, mis selle abil on võimalik teostada. Swagger genereerib vastava dokumentatsiooni igale rakendusliidese versioonile.

5 Rakenduse edasiarendused

Rakenduse tulevased arendused on seotud funktsionaalsuse täiendamisega, automaattestide välja töötlemisega ning projekti tarkvara uuendamisega. Mainitud edasiarendused on asjad, mis lähiajal tööse tuleks võtta.

Esimese niinimetatud lisafunktsionaalsuse all peab autor silmas võimalust rakendusliidese abil rohkem informatsiooni kätte saada. Näiteks oleks hea saada ülevaade, kuidas eelnevad kohtumised kahe meeskonna vahel on möödunud, kumb meeskond on rohkem olnud edukam, resultatiivsem. Samuti oleks kasulik näha missugune on võistkonna hetke vorm, kas mängule tullakse neile positiivselt lõppenud mängust.

Järgmiseks oluliseks arenduseks võib pidada Eesti jalgpalli süsteemis mängivate klubide *endpointide* täiustamist. Sõltuvalt siis klubide soovist ja vajadusest võib ette tulla, et mõningaid andmeid oleks mõistlik teistmoodi/teisel kujul esitleda.

Lisaks tuleb rakendusliidese luua uued *endpointid*, mis teeksid andmed kättesaadavaks tarkvaraettevõtte Dotcomsport jaoks. Mainitud organisatsioon arendab spordialade jaoks välja tooteid ja teenuseid. Jalgpalli jaoks on olemas mängija arendussüsteem, mille EJL plaanib lähiajal kasutusele võtta. API abil saab mugavalt edastada Dotcomspordi jaoks vajaliku informatsiooni koondise meeskondade, personali ja mängijate kohta. Tehes andmed kättesaadavaks teise süsteemi jaoks on seal võimalik saada põhjalikum ülevaade koondise koosseisust, kanditaadidest ja jälgida nende arengut [29].

Automaattestide lisamine rakendusse põhineks eelkõige üksustestide ehk JUnit testide välja töötamisel. Testimine võimaldaks avastada programmis tekkinud vigu nii loomise kui ka muutmise ajal. Seega oleks kontroll, et uued muudatused rakendusliideses ei lõhuks ära olemasolevat funktsionaalsust.

Kuna projekti alguses oli viimaseks stabiilseks raamistiku versiooniks ASP.NET Core 3.1, siis tuleks tulevikus rakendus üle viia hiljuti Microsofti poolt väljastatud uusima versiooni peale. Lisaks peab olema kursis teiste rakenduses kasutatud tehnoloogiate uuenduskuuridega.

Kokkuvõte

Töö eesmärgiks oli luua rakendusliides, mis tagaks ligipääsu Eesti Jalgpalli Liidu andmebaasis olevatele andmetele. Loodud API täidab EJL poolt kirjeldatud nõudeid ja kokkulepitud funktsionaalsust. Rakendusliidese abil on võimalik saada informatsiooni kõigi Eesti jalgpalli süsteemi kuuluvate liigade kohta. Ühe hooaja jooksul on andmed kättesaadavad üle 150 liiga kohta. Lisaks on kajastatud rahvusvahelised võistlused, kus Eesti rahvuskoondised osalevad.

Loodud ASP.NET Core rakenduses on kasutatud mitmekihilist arhitektuuri, kus esitluse, töötlemise ja andmete kättesaamise protsessid on üksteisest loogiliselt eraldatud. Rakendusliides on dokumenteeritud ja kasutusele võetud Eesti jalgpalli jälgimiseks mõeldud mobiilirakenduse poolt. Samuti on võimalik tagada ligipääs teistele osapooltele, kes soovivad antud teenust tarbida.

Kasutatud kirjandus

[1] „Jalgpalliliidu asutamisest möödus 96 aastat,“ [Võrgumaterjal]. Available: http://jalgpall.ee/ejl/uudised/jalgpalliliidu-asutamisest-moodus-96-aastat-n13077. []

[2] API-Football Coverage, [Võrgumaterjal]. Available: https://www.api-football.com/coverage

[3] API-Football Documentation, [Võrgumaterjal]. Available: https://www.api-football.com/documentation

[4] iSports API, [Võrgumaterjal]. Available: https://www.isportsapi.com/home/about.html

[5] API football, [Võrgumaterjal]. Available: https://apifootball.com/

[6] „How Many Computer Programming Languages Are There?,“ [Võrgumaterjal]. Available: https://careerkarma.com/blog/how-many-coding-languages-are-there/

[7] „TIOBE Index for December 2020,“ [Võrgumaterjal]. Available: https://www.tiobe.com/tiobe-index/

[8] „Veebirakenduste arendamine ASP.NET Core platvormil,“ [Võrgumaterjal]. Available: https://old.taltech.ee/taiendusoppijale/koolituskalender/algavad-koolitused/algavad-koolitused-2/?koolitus=8324

[9] „Microsoft Visual Studio ja C#,“ [Võrgumaterjal]. Available: https://digiarhiiv.ut.ee/Ained/Doc/VFailid/CSharp_ja_VS.pdf

[10] „An introduction to NuGet,“ [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/nuget/what-is-nuget

[11] „Veebiserverite tarkvara vordlus,“ [Võrgumaterjal]. Available: https://wiki.itcollege.ee/index.php/Veebiserverite_tarkvara_vordlus

[12] „What Is A Reverse Proxy?,“ [Võrgumaterjal]. Available: https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy

[13] „Kestrel web server implementation in ASP.NET Core,“ [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?view=aspnetcore-5.0

[14] „Docker – mis asi see on ja miks kõik seda kasutavad?,“ [Võrgumaterjal]. Available: https://blog.twn.ee/et/docker-mis-asi-see-on-ja-miks-koik-seda-kasutatavad

[15] „Mitmekihiline arhitektuur,“ [Võrgumaterjal]. Available: https://eopearhiiv.edu.ee/e-kursused/eucip/arendus/1631_mitmekihiline_arhitektuur.html

[16] „Common web application architectures,“ [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures

[17] „N-Tier Architecture in ASP.NET Core,“ [Võrgumaterjal]. Available: https://medium.com/@chathuranga94/n-tier-architecture-in-asp-net-core-d1f1b14f2010

[18] „Working with Data in ASP.NET Core Apps,“ [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/work-with-data-in-asp-net-core-apps

[19] „Design the infrastructure persistence layer,“ [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design

[20] „AutoMapper,“ [Võrgumaterjal]. Available: https://automapper

[21] „How to Create Web APIs in ASP.NET Core [RESTful pattern],“ [Võrgumaterjal]. Available: https://www.yogihosting.com/aspnet-core-api-controllers/

[22] „Cache in-memory in ASP.NET Core,“ [Võrgumaterjal]. https://docs.microsoft.com/en-us/aspnet/core/performance/caching/memory?view=aspnetcore-2.2

[23] „Regulations of the UEFA Nations League 2020/21,“ [Võrgumaterjal]. Available: https://documents.uefa.com/viewer/document/jJTWTpzi2KN9D8VRYz~Bpg

[24] „WordPressi cron-tööde seadistamine,“ [Võrgumaterjal]. https://docs.microsoft.com/en-us/aspnet/core/performance/caching/memory?view=aspnetcore-2.2

[25] „Overview of ASP.NET Core Security [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/core/security/?view=aspnetcore-5.0

[26] „Client IP safelist for ASP.NET Core,“ [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/core/security/ip-safelist?view=aspnetcore-5.0

[27] „Tarkvara testimist käsitlev juhendmaterjal“ [Võrgumaterjal]. Available: https://www.mkm.ee/sites/default/files/tarkvara_testimise_juhis_-_koopia.doc

[28] „Documenting Your Existing APIs: API Documentation Made Easy with OpenAPI & Swagger,“ [Võrgumaterjal]. Available: https://swagger.io/resources/articles/documenting-apis-with-swagger/

[29] „Dotcomsport,“ [Võrgumaterjal]. Available: https://dotcomsport.nl/us/dotcomclub-us/

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

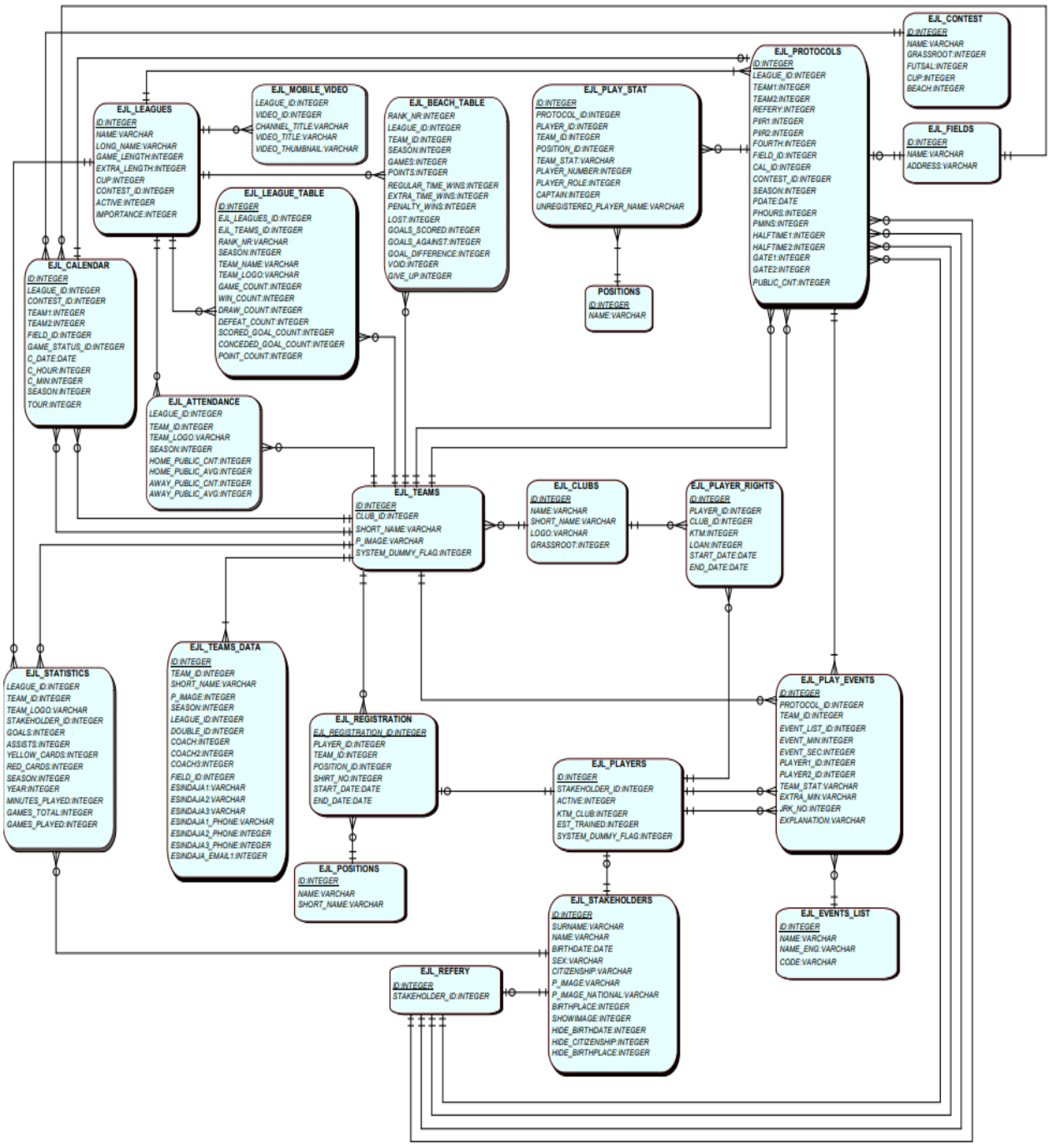
Mina, Kristjan Karama

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Eesti Jalgpalli Liidule rakendusliidese arendus“, mille juhendaja on Nadežda Furs-Nižnikova
   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.

07.01.2021

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.

Lisa 2 - Liigade olemi-suhte diagramm



Lisa 3 - Koondiste olemi-suhte diagramm