TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Martin Paroll 111053IABB
Katrin Avloi 111897IABB
Kristi Seemen 111872IABB

RAHVAVIISIDE INFOSÜSTEEMI TARKVARA ARENDAMINE EESTI KIRJANDUSMUUSEUMILE

bakalaureusetöö

Juhendaja: Erki Eessaar
PhD

Tallinn 2020

Autorideklaratsioon

Kinnitame, et oleme 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.

Autorid: Martin Paroll, Katrin Avloi, Kristi Seemen

24.05.2020

Annotatsioon

Käesoleva töö eesmärgiks on arendada Eesti Kirjandusmuuseumile rahvaviiside infosüsteemi tarkvara, mis võimaldab hoida ja hallata rahvaviisidega seotud andmeid. Ülesanne lahendatakse esitatud tellimuse alusel meeskondliku projekti vormis.

Töö tegemisel järgivad töö autorid enda kokku pandud arendusmetoodikat, mis kasutab iteratiivset arendusmudelit ja laenab elemente erinevatest paindmetoodikatest. Tarkvara nõuete analüüsimisel ja loomisel kasutatakse andmekeskset lähenemist.

Projekti käigus analüüsitakse detailselt nõudeid, valitakse tehniline arhitektuur, luuakse andmebaas ja veebirakendus ning testitakse realiseeritud süsteemi.

Projekti tulemina antakse Eesti Kirjandusmuuseumile üle kasutusele võtmiseks valmis tarkvara koos lähtekoodi ja projekti dokumentatsiooniga. Töö autorid soovitavad valminud tarkvara edasi arendada, et saavutada kõik rahvaviiside infosüsteemi eesmärgid.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 80 leheküljel, 5 peatükki, 29 joonist, 4 tabelit.

Abstract

Development of the Folk Tunes Information System Software for the Estonian Literary Museum

The purpose of this bachelor thesis is to develop the folk tunes information system software for the Estonian Literary Museum. The software allows maintaining and managing data related to folk tunes. The authors of the thesis have resolved the requested requirements in the shape of a teamwork project.

The authors have utilized a self-assembled development method that uses an iterative approach. Elements from different agile methodologies (Scrum, Kanban) are used. A data-centric approach is used for software analysis and software creation. The system is decomposed into subsystems based on the main data objects (entity types). It facilitates development of the system step-by-step, presenting and organizing the system's models, as well as organizing navigation in the software. The system is modeled in UML (entity-relationship diagram and database design diagrams as class diagrams, state machine models of the main entity types, use case diagrams, package diagram to present dependencies between the subsystems, deployment diagram to describe the system's technical architecture). The resulting software can be characterized as a CRUD database application, which allows its users to create, read, update, and delete data in the database.

The project starts with collecting requirements (functional and non-functional requirements as well as requirements to data) and choosing the appropriate technical architecture. This is followed by the creation of the database and the web application. The final step involves testing of the created information system software. The database is implemented in PostgreSQL and the web application is created in PHP. CakePHP rapid development framework is used to generate the application based on the database base tables and the generated code is modified to satisfy the needs of the client. Docker containers are used to simplify development. The authors log their work in the project

and give an overview of effort that each team member put into the work. The authors analyze their work process and results.

The source code and documentation of the resulting software from this project will be handed over to the Estonian Literary Museum. The authors of the thesis recommend continuing the development of the software in order to meet all objectives of the folk tunes information system.

The thesis is in Estonian and contains 80 pages of text, 5 chapters, 29 figures, 4 tables.

Lühendite ja mõistete sõnastik

CakePHP	PHP tarkvara raamistik
CASE	*Computer Aided Software (System) Engineering*, tarkvara tarkvara- ja infosüsteemide modelleerimiseks
EA	*Enterprise Architect*, CASE tarkvara
EKM	Eesti Kirjandusmuuseum
ERA	Eesti Rahvaluule Arhiiv
HTML	*Hypertext Markup Language*, hüpertekst-märgistuskeel
HTTPS	*Hypertext Transfer Protocol Secure*, turvaline hüperteksti edastamise protokoll
Kivike	Eesti Kirjandusmuuseumi failirepositoorium
MVC	*Model-View-Controller*, mudel-vaade-kontroller tarkvara disainimuster
ORM	*Object-Relational Mapping*, objekt-relatsioonvastendus – objektorienteeritud mudelil põhineva tarkvara ja relatsioonilisel mudelil põhineva andmebaasi sidumiseks mõeldud programmeerimistehnoloogia
PHP	*Hypertext Preprocessor*, skriptimiskeel
Rahvaviis	Rahvalaulu meloodia. Töös kasutatakse sellele mõistele viitamiseks enamasti terminit „viis“
Regilaul	Eesti vanema rahvalaulu liik
Tellija	Eesti Kirjandusmuuseumi Eesti Rahvaluule Arhiiv
Teostaja	Töö autorid Martin Paroll, Katrin Avloi ja Kristi Seemen
UML	*Unified Modeling Language*, ühtne (unifitseeritud) modelleerimiskeel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Sissejuhatavas osas antakse lühiülevaade projekti taustast, eesmärgist ja töö tulemusel realiseeritud funktsionaalsustest ning kirjeldatakse töö edasine struktuur.

1.1 Üldine taust ja projekti lühikirjeldus

Eesti Kirjandusmuuseumil (EKM) puudub rahvaviiside (edaspidi viisid) andmete haldamise tarkvara. Selle loomise tellimuse esitas EKM-i Eesti Rahvaluule Arhiiv (ERA) Tallinna Tehnikaülikoolile. Selle ülesande lahendamiseks loodi töörühm, kuhu kuulusid teostaja poolelt käesoleva töö autorid ning tellija poolelt ERA vanemteadur Taive Särg ja digitaalarhivaar Olga Ivaškevitš.

EKM-is ei ole välja kujunenud arendusprotsesside standardit. Asutuse direktori poolt väljaantud käskkirja kohaselt planeerivad, viivad läbi ning menetlevad hankeid ja arendusi andmebaaside ning arhiivimaterjalide ja teadusandmete omanikud (osakonnad, peakasutajad). IT valdkonna spetsialistid nõustavad selles protsessis osalejaid ainult tehnilistes küsimustes. Sellest tulenevalt pakkusid töö autorid välja arendusprotsessi ning koostöövormid ja töövahendid nii tellijaga suhtlemiseks kui ka tarkvara realiseerimiseks.

Eeltöö käigus analüüsiti EKM-i poolt tarkvarale esitatud nõudeid ja olemasolevat olukorda, valiti realiseeritav lahendus ning koostati nimekiri realiseeritavatest funktsionaalsustest. Ühise arusaamise tagamiseks kirjeldati ja kinnitati koos tellijaga projekti skoop ja visioon. Antud dokumendis jagati vajatav funktsionaalsus kaheks, millest esimene osa realiseeriti käesoleva projektiga ning teine osa jääb tulevikus tegemiseks.

Projekti käigus koguti nõudeid süsteemile, loodi rahvaviiside infosüsteemi andmebaas ja veebirakendus, testiti realiseeritud süsteemi vastavust nõuetele ning koostati dokumentatsioon.

1.2 Probleem ja projekti eesmärk

EKM haldas viiside andmeid MS Access andmebaasisüsteemis loodud andmebaasis. Selles andmebaasis jaotati viiside andmed kahte suurde plokki: metaandmed ja kodeeringud. Hiljem kopeeriti osade viiside metaandmed Google Sheetsi eesmärgiga neid kontrollida ja korrastada.

Viisidega on veel seotud kaks olemasolevat EKM-i andmekogu: failid ja tekstid. Viisid ise nii piltidena kui ka helifailidena asuvad failirepositooriumis Kivike, kus hoitakse digiteeritud kujul arhiivimaterjale. Viiside tekste hoitakse Eesti regilaulude andmebaasis. Lisaks on EKM-il realiseeritud Kivikese juurde isikute moodul, kuhu on koondatud kõik isikutega seotud andmed ja mis võimaldab kõigil arhiivi andmebaasidel kasutada ühte ja sama isikutega seotud informatsiooni.

Eeltöö käigus tuvastati järgmised olulisemad viiside infosüsteemi eesmärgid.

- Võimaldada hoida ja hallata viiside metaandmeid ja kodeeringuid.
- Võimaldada ligipääsu seotud andmekogude andmetele ja isikute moodulile.
- Võimaldada analüüsida viiside andmeid teaduslikel eesmärkidel.
- Võimaldada ligipääsu avalikule kasutajale.

Käesoleva projekti põhieesmärgiks oli realiseerida esimene eesmärk – hoida ja hallata viiside metaandmeid ja kodeeringuid. See on eelduseks järgmiste eesmärkide realiseerimisele.

1.3 Lühiülevaade realiseeritud funktsionaalsusest

Projekti käigus loodi tarkvara, mis koosneb veebirakendusest ja andmebaasist. Loodud viiside andmebaasis hoitakse andmeid viiside kohta. Vastavalt EKM-i nõudele kasutati andmebaasisüsteemina PostgreSQLi.

Käesoleva projekti käigus realiseeritud funktsionaalsus lepiti kokku koostöös tellijaga.

- Viisi sisestamine, muutmine, vaatamine, otsimine ja kustutamine.
- Isikute lisamine, muutmine, vaatamine ja kustutamine.
- Kasutajate lisamine, muutmine, aktiveerimine ja deaktiveerimine.
- Rollid ja õigused, autentimine ja autoriseerimine.

- Klassifikaatorite väärtuste lisamine, muutmine, aktiveerimine, deaktiveerimine ja kustutamine.
- Sündmuste (kasutajate tegevused) lisamine ja vaatamine.

1.4 Töö edasine struktuur

Töö on jaotatud kolmeks põhiosaks. Metoodika all (peatükk 2) esitatakse töö objekti detailsem kirjeldus ning antakse ülevaade rakendatud tööprotsessidest ja kasutatud tööriistadest. Töö tulemuste osas (peatükk 3) esitatakse funktsionaalsed nõuded allsüsteemide kaupa, mittefunktsionaalsed nõuded, tarkvara tehniline arhitektuur, andmebaasi ja kasutajaliidese disaini põhimõtted ning läbi viidud testid. Viimases töö põhiosa peatükis (peatükk 4) analüüsitakse ja põhjendatakse töö käigus tehtud valikuid ning tehakse järeldusi nende sobivuse osas. Samuti on seal tehtud tööde detailsem logi ning nii töö autorite kui ka EKM-i hinnang läbi viidud projektile.

2 Metoodika

Käesolevas peatükis esitatakse objekti detailsem kirjeldus ning antakse ülevaade rakendatud tööprotsessidest ja kasutatud tööriistadest.

2.1 Objekti detailne kirjeldus

EKM-i arhiivis asuva viiside kogu võib andmekandjate alusel jagada kaheks: helikogud ja käsikirjalised kogud. Viiside kogu täpne suurus ei ole teada, kuna seda ei ole kunagi kokku loetud ning samas see pidevalt ka kasvab. Hinnanguliselt on viiside kogus umbes 100 000 viisi.

Suurem osa helisalvestustest on digiteeritud ja need asuvad EKM-i sisevõrgus. Salvestused on järjestatud kogude kaupa, kuid nende juures ei ole metaandmeid.

Käsikirjalistest kogudest on umbes 10 000 viisi kopeeritud viisikartoteeki. Kogud on kopeeritud kolmes eksemplaris ja järjestatud üks kogude, üks liikide-teemade ja üks kihelkondade põhjal. Viisikaartidele on kirjutatud ka metaandmed.

Enamik materjalist on publitseerimata ja viisi leidmiseks peab minema EKM-i kohale, kus paberkartoteekide ja registriraamatute põhjal on võimalik viise otsida. Helisalvestatud viiside osas ei ole olemas ei liigilist, piirkondlikku ega muud kartoteeki.

1980. aastatel loodi Ingrid Rüütli eestvedamisel rahvaviiside andmebaas, kuhu on viisikartoteegi alusel aastakümnete jooksul viise sisestatud. Andmebaasi viimane versioon on realiseeritud MS Access andmebaasisüsteemis. Selles andmebaasis on üle 6 000 viisi koos metaandmete ja kodeeringutega. Viisid on jagatud vormide järgi üherealisteks ja kaherealisteks viisideks. Kaherealiste viiside (ligi 5 000) metaandmete korrastamiseks kopeeriti need Google Sheetsi, kus neid on aja jooksul kontrollitud ja vajadusel ka muudetud. MS Accessi andmebaasi ei ole neid muudatusi viidud. Sellega peab arvestama olemasolevate andmete migreerimisel uude viiside infosüsteemi.

Andmebaasi edasiarendamiseks pöördus EKM Tallinna Tehnikaülikooli poole ning käesoleva töö autorid otsustasid teha antud projekti oma lõputöö raames.

Projekti tegemise protsessi võis jagada kaheks: eeltöö ja tarkvaraarendus.

Eeltöö eesmärgiks oli saada ülevaade:

- EKM-i arendusprotsessidest;
- EKM-is kasutusel olevatest infosüsteemidest;
- viiside seostest olemasolevate andmekogudega;
- viiside infosüsteemi eesmärkidest.

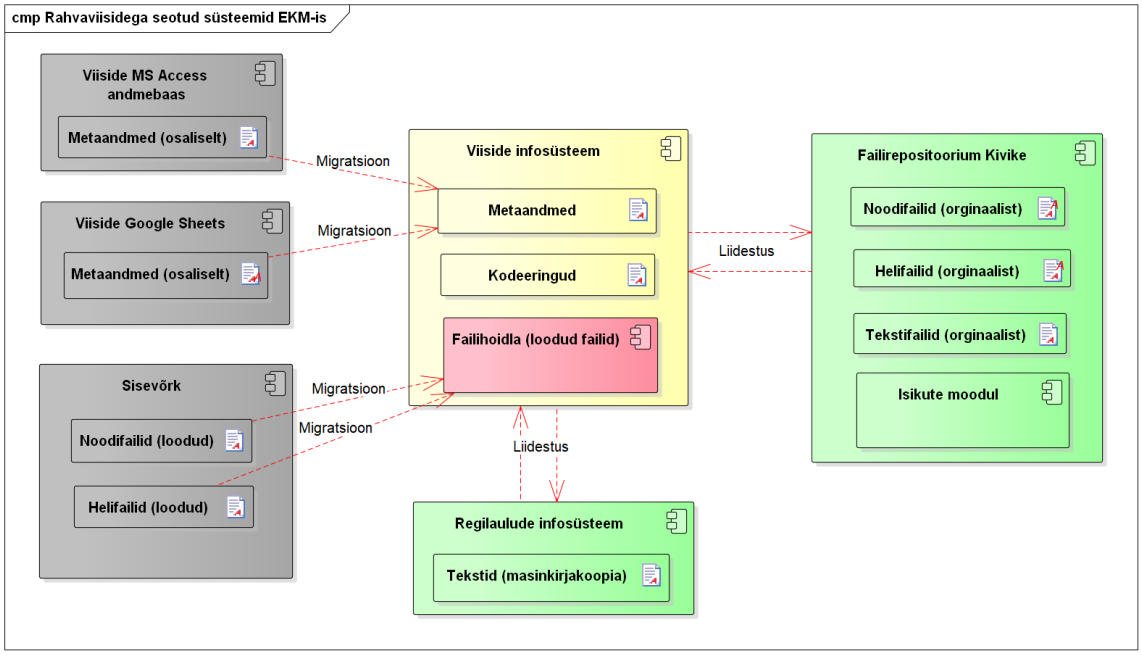
Eeltöö käigus selgus, et EKM-is on registreeritud üle 50 andmekogu, millest viisidega on seotud failirepositoorium Kivike ja Eesti regilaulude andmebaas. Kuna asutuses on ainult viis IT-töötajat, siis ei ole EKM-is arendusprotsesside standardit ning andmekogude arendusi planeerivad ja teevad andmekogude omanikud.

Andmekogude suur hulk on tingitud sellest, et andmekogudes on vaja hoida väga spetsiifilisi andmeid. Näiteks viiside iseloomustamiseks kasutatavad andmed ei kattu suurel määral vanasõnade iseloomustamiseks kasutatavate andmetega.

Eeltöö tulemusena otsustasid töö autorid arendada viiside jaoks eraldi tarkvara, mis võimaldab hoida ja hallata viiside metaandmeid ja kodeeringuid ning mis on eelduseks viiside infosüsteemi ülejäänud eesmärkide realiseerimisele.

Joonis 1 on toodud viisidega seotud süsteemide ülevaade EKM-is, mis kirjeldab muutusi infosüsteemide kasutamises nii peale käesoleva projekti valmimist kui ka peale olulisemate tuleviku arenduste realiseerimist. Värvidel on järgmine tähendus.

- Kollasega on tähistatud käesoleva projekti raames loodav komponent.
- Rohelisega on tähistatud olemasolevad komponendid, mis jäävad pikemas perspektiivis kasutusele.
- Punasega on tähistatud tulevikus loodavad komponendid ja liidestused ning vajalikud andmete ja failide migratsioonid.
- Halliga on tähistatud komponendid, mis peale migratsiooni asendatakse teiste süsteemidega.



Joonis 1. Viisidega seotud infosüsteemid EKM-is.

2.2 Tööprotsessi kirjeldus

Projekti aluseks oli EKM-i tellimus Tallinna Tehnikaülikoolile. Tellimuse täitmiseks loodi projekti meeskond, kuhu kuulusid teostaja poolelt käesoleva töö autorid, ning valiti lõputöö juhendaja. Projekti tegemiseks määras tellija omapoolsed meeskonnaliikmed. Eeltöö raames lepiti tellijaga kokku projekti skoop ja visioon.

Projekti alustamisel valiti projektijuhiks Martin Paroll. Muus osas rollide jaotust meeskonnas kokku ei lepitud, vaid tööülesannete jagunemine otsustati jooksvalt projekti käigus. Kokkuvõttes kujunesid igal meeskonnaliikmel olulisemad vastutusalad, kuna nii oli efektiivsem tööd teha. Lisaks projektijuhtimisele oli Martin Paroll põhiline rakenduse arendaja. Kristi Seemeni vastutada oli andmebaasi loomine ning samuti panustas ta märkimisväärselt nii analüüsi kui ka rakendusega seotud töödesse. Katrin Avloi põhirollideks olid analüüs ja testimine. Vajadusel tegid kõik liikmed projektis erinevaid töid. Olulisemad otsused võeti vastu ühiselt.

Töö tegemisel järgisid töö autorid enda kokku pandud arendusmetoodikat [10], mis kasutas iteratiivset arendusmudelit ja laenas nii Kanbani [2] kui ka Scrumi [42] elemente. Metoodikate arendus ja projektile sobivate hübriidide kokkupanek on tarkvaraarenduses levinud praktika [1].

Kanbani elementidest kasutati pidevat protsessi parendamist, poolelioleva töö piiramist (üks tegevus korraga) ning võimalikult kiiresti iga kasutusjuhu kliendini viimist. Scrumi elementidest kasutati järgmise arendustegevuse iteratsiooni planeerimist (kuid iteratsiooni pikkus ei olnud fikseeritud), ootel olevate tööde ettevalmistust ning kliendi tagasiside arvestamist järgmiste arenduste planeerimisel.

Enamlevinud agiilsetest meetoditest [1] rakendati enda arendusmetoodikas veel lühikesi iteratsioone, tihedat tarkvara tarnet ning sama meeskonda arendamiseks ja testimiseks. Kokku teostati 14 iteratsiooni.

Töö efektiivsemaks organiseerimiseks oli töö autoritel kokku lepitud regulaarne projekti koosolek (*Standup*), mille eesmärk oli teha ülevaade hetkeseisust, tõstatada tekkinud probleeme, võtta vastu otsuseid ja leppida kokku järgmised ülesanded. Kohanduti vastavalt olukorrale, keskendudes kas lähtuvalt Kanbanist ülesannetele või lähtuvalt Scrumist inimestele. Tellija osales regulaarsetel koosolekutel vastavalt vajadusele.

Töö autorid kaalusid lisaks ekstreemprogrammeerimise (*Extreme Programming*) meetodite [26] kasutusele võtmist, näiteks paarisprogrammeerimine, testipõhine arendus (*Test-driven Development*) ning refaktoreerimine. Viimaseid otsustati mitte kasutada.

Lõputöö meetodiks oli disainitegevusuuring [43], kus tehist arendatakse iteratiivselt ning sellega paralleelselt üritatakse seda kasutusele võtta. Meetodi keskseks objektiks on tehis ise. Iga iteratsiooni järgselt analüüsiti tehist uuesti ning hinnangu alusel planeeriti edasine arendus. Töö tulemusena tekkis uut teadmist arendatava süsteemi ja arenduseks kasutatud protsesside kohta.

Projekti realiseerimist alustati nõuete kogumisega, milleks korraldati tellijaga füüsilisi kokkusaamisi, suheldi e-kirjade ja videokoosolekute vahendusel. Samu töövorme kasutati kogu projekti jooksul vastavalt vajadusele.

Tellija oli kogu projekti vältel tihedalt analüüsi kaasatud ning kasutati kaasdisainimise põhimõtet [9]. Korraldati mitmeid kohtumisi, kus tellijal oli võimalik jagada oma soove tarkvara osas, sh mida ja kuidas seal teha peab saama. Arutati ühiselt, et milline võiks olla loodav lahendus kasutaja vaatest. Arvestades tellija esindajate tausta (oma valdkonna spetsialistid, aga puudulikud tehnoloogia ja tarkvaraarenduse teadmised) koostati kogutavatele andmetele esitatud nõuete analüüsimiseks eraldi kõigile

osapooltele arusaadavas keeles töödokument, mille valmimises tellija aktiivselt osales. Tellija ei vaadanud üle ega kinnitanud koostatud mudeleid (nt kontseptuaalne andmemudel, andmebaasi füüsilise disaini mudel). Iga iteratsiooni järel tutvustati valminud funktsionaalsust koheselt tellijale eesmärgiga saada kiiret tagasisidet sobivuse osas.

Süsteemi analüüsimisel kasutasid töö autorid andmekeskset lähenemist [21], mille eesmärgiks oli saada ülevaade süsteemi põhiobjekti (põhiolemitüübi) *Viis* andmetest ning nende muutumisest põhiobjekti elutsükli jooksul. Kogutud andmeid kasutati süsteemi modelleerimisel. Samast lähenemisest juhinduti ka edaspidisel tarkvara loomisel.

Tulenevalt iteratiivset mudelit järgiva arendusmetoodika valikust, realiseeriti projekti skoobis olev funktsionaalsus järk-järgult ning rakenduse valideerimisel kasutati agiilset lähenemist. See tähendas tööprotsesside vaates järgmist.

- Kui tellija vajadused seda nõudsid, siis tehti testimise käigus esile kerkinud teemades jooksvalt muudatusi ja otsuseid, mitte ei peetud kinni algsest plaanist.
- Oluline oli töötav ja ootustele vastav tarkvarasüsteem, mitte iga detaili dokumenteerimine.
- Testiti järk-järgult osalise funktsionaalsusega vastavalt arenduse valmimisele, sh tellijale anti tarkvara testimiseks osade kaupa.

Töö kirjutamisel ja esitatavate mudelite valikul kasutasid töö autorid allikatena Tallinna Tehnikaülikooli õppeaine „Andmebaasid II“ näiteprojekti „Ülikooli infosüsteemi vastuvõtuaegade allsüsteem“ [22] ja riigihanke „Sotsiaalkaitse infosüsteemi arendustööd“ alusdokumentatsiooni [46].

2.3 Tööriistade kirjeldus

Tööriistade valikul lähtusid autorid põhimõttest, et valitud tarkvara oleks kasutatav erinevatel platvormidel. Kaks töö autorit kasutasid operatsioonisüsteemi MacOS, üks autor operatsioonisüsteemi Windows.

Dokumentide hoidmiseks valisid töö autorid Google Drive keskkonna. Google Docs ja Google Sheets olid kasutusel nii erinevate töödokumentide koos toimetamiseks, tellijatega materjalide vahetamiseks kui ka lõputöö kirjutamiseks.

Süsteemi modelleeriti standardses visuaalses modelleerimiskeeles UML (*Unified Modeling Language*) [54] luues järgnevat tüüpi diagramme.

- Süsteemi allsüsteemide omavahelised sõltuvused paketidiagrammidena (paketiskeemidena).
- Olemi-suhte diagrammid ja andmebaasi disaini diagrammid klassidiagrammidena (klassiskeemidena).
- Põhiobjektide (põhiolemitüüpide) võimalikud elutsüklid olekudiagrammidena (seisundidiagrammidena, olekuskeemidena).
- Süsteemi kasutamise juhtumid kasutusjuhtude diagrammidena (kasutusmalliskeemidena).
- Süsteemi tehniline arhitektuur evitusskeemina.

Diagrammidele lisati vastavalt vajadusele sõnalisi kirjeldusi. CASE vahendina oli kasutusel Enterprise Architect (EA) [27]. Lisaks erinevat tüüpi mudelite loomisele võimaldas EA genereerida kontseptuaalsest andmemudelist andmebaasi füüsilise disaini mudeli esialgse versiooni, millest peale täiendamist sai omakorda genereerida baastabelite loomiseks mõeldud SQL laused. EA oli ainuke kasutusel olnud tarkvara, mis ei ühildunud operatsioonisüsteemiga MacOS. Kaks MacOS platvormi kasutanud töö autorit kasutasid Oracle virtualiseerimise tarkvara VirtualBox [37], käivitades selle abil Windows operatsioonisüsteemi ning sellel omakord EA tarkvara.

Tehnilise arhitektuuri diagrammid (vt Joonis 7 ja Joonis 8) on tehtud Diagrams.net [19] tarkvara abil.

Projektijuhtimiseks ning tarkvaraarendusest ülevaate saamiseks kasutasid töö autorid Jira keskkonda [30], kus hallati nii arenduse kui ka lõputöö kirjutamisega seotud ülesandeid.

Omavaheliseks suhtlemiseks kasutasid töö autorid Slack [57] keskkonda, kuhu loodi vestluskanalid erinevate teemade jaoks. Peamised kanalid suhtlemiseks olid järgmised:

- #andmebaas – andmebaasiga seotud küsimused;

- #koosolekud – kohtumiste korraldamine;
- #rakendus – rakenduse programmeerimine, testimine, juurutamine;
- #general – kõik eelnevatesse teemadesse mittesobiv.

Lisaks seadistati kanal #arenduskeskkond selliselt, et kõik Jira keskkonna tähtsamad toimingud jooksvate ülesannetega ning koodirepositooriumisse jõudnud *commit* ning *merge* teated peegeldatakse sinna. See andis töö autoritele hea ülevaate kaasautorite jooksvatest muudatustest.

Videokoosolekuteks nii töö autoritega kui ka tellijaga kasutati Google Hangouts teenust. Juhendajaga peeti koosolekuid Microsoft Teams [14] vahendusel.

Arenduskeskkond seati üles Docker platvormi tööriista Docker Compose [38] abil. Keskkond koosnes Nginx veebiserverist, PHP teenusserverist ning PostgreSQL andmebaasiserverist. PHP serveris oli kasutusel vaba lähtekoodiga veebirakenduste arendamise raamistik CakePHP [11]. Suur osa rakenduse koodi genereeriti andmebaasi põhjal CakePHP tööriistade abil.

Koodirepositooriumina oli kasutusel Bitbucket [7]. Koodibaasi haldamiseks kohaliku arvuti ning repositooriumi vahel kasutasid töö autorid tarkvara Sourcetree [47].

Koodi kirjutamiseks, tekstiredaktorina kasutati programme Sublime Text [48] ja Notepad++ [35]. Andmebaasiga töötamiseks olid kasutusel Valentina Studio [55] ja pgAdmin [39].

Kasutajaliidesesse on lisatud Bootstrap 4 stiilid [8] ja elementide joondamiseks erinevatele ekraanisuurustele skaleeruv Bootstrap 4 ruudustik [28]. Kasutusmugavuse suurendamiseks on lisatud kuupäeva valimise teek [31] ning teek salvestamata andmete kaotamise hoiatuse andmiseks [16].

3 Töö tulemused

Selles peatükis kirjeldatakse detailselt nõuded loodavale süsteemile ning selle tehniline arhitektuur, andmebaasi ja kasutajaliidese disaini põhimõtted, koodi ülesehitus ning testimise protsess. Lisaks loetletakse projekti tulemina EKM-ile üleantavad tehised ning esitatakse viiside infosüsteemi edasine arendusvaade.

3.1 Nõuded

Töö paremaks organiseerimiseks otsustati viiside infosüsteem jagada allsüsteemideks. Infosüsteem koosneb viiest funktsionaalsest allsüsteemist ja nende teenindatavatest (loetavatest ja muudetavatest) registritest, mis on esitatud Tabel 1. Allsüsteemid leiti põhiobjektide keskselt – igale põhiobjektile vastab funktsionaalne allsüsteem ja selle poolt teenindatav register. Iga funktsionaalne allsüsteem võimaldab põhiobjektile vastavate andmete haldust läbi terve põhiobjekti elutsükli.

Tabel 1. Funktsionaalsed allsüsteemid ja teenindatavad registrid.

Funktsionaalne allsüsteem	**Teenindatav register**	**Allsüsteemi tüüp**
Viiside funktsionaalne allsüsteem	Viiside register	Põhitegevusega seotud
Isikute funktsionaalne allsüsteem	Isikute register	Administratiivne
Kasutajate funktsionaalne allsüsteem	Kasutajate register	Administratiivne
Klassifikaatorite funktsionaalne allsüsteem	Klassifikaatorite register	Administratiivne
Sündmuste funktsionaalne allsüsteem	Sündmuste register	Administratiivne

Alapeatükkides 3.1.1 – 3.1.5 on kirjeldatud allsüsteemide kaupa iga allsüsteemi eesmärgid, kasutusjuhud ja äriarhitektuuri joonised (Joonis 2 – Joonis 6), kus on toodud allsüsteemi kasutavad rollid (pädevusalad) ja tööks vajatavad registrid. Kasutusjuhud kirjeldavad süsteemi kasutamise juhtusid, kus keegi või miski kasutab süsteemi tarkvara. Kasutusjuhud on esitatud tekstikirjeldusena kõrgtaseme formaadis, mille puhul esitatakse kasutusjuhu nimi, primaarsed tegutsejad (roll või rollid, mille esindajad

süsteemi tarkvara kasutades kasutusjuhu algatavad) ning kasutusjuhu vabatekstiline sõnaline selgitus.

Mittefunktsionaalsed nõuded on kirjeldatud alapeatükis 3.1.6.

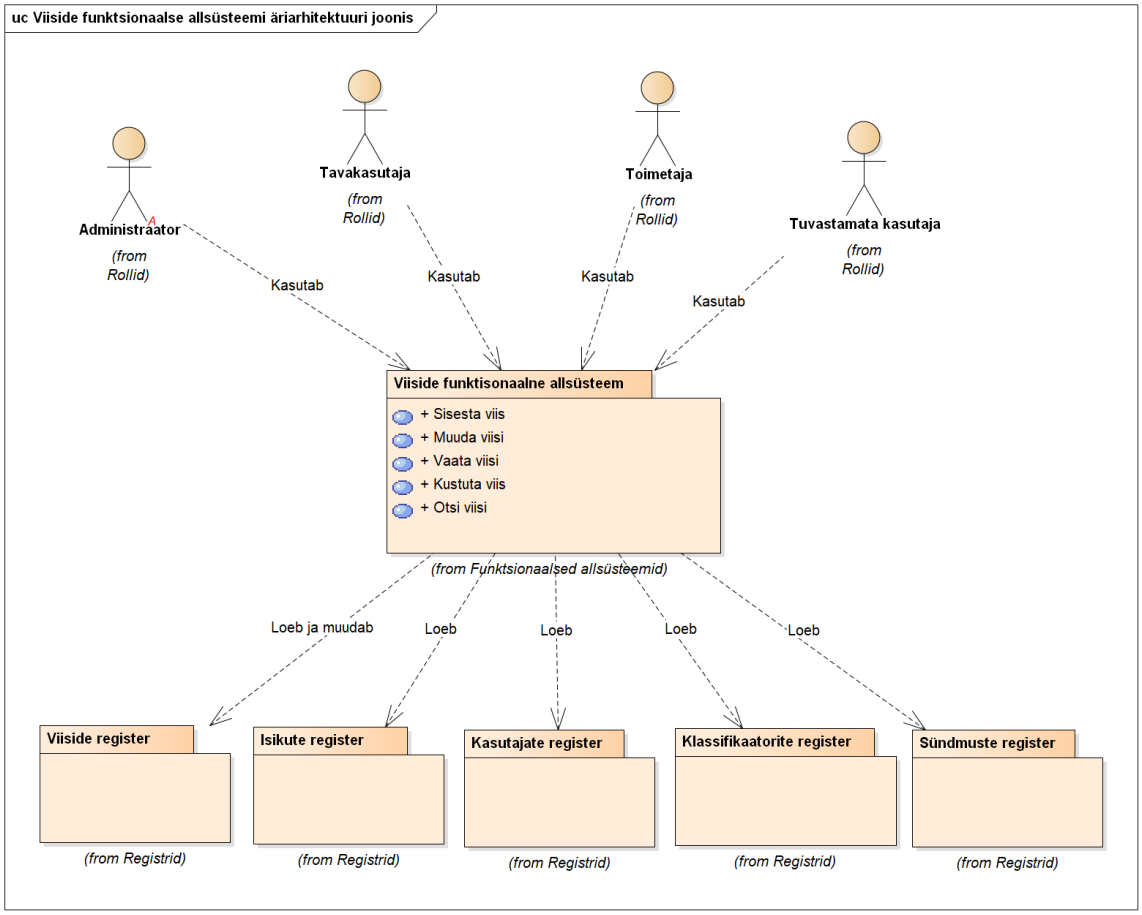
Kasutusjuhtude diagrammid on toodud Lisa 4. Diagrammid on esitatud rollide kaupa, andes ülevaate iga rolli õigustest allsüsteemide üleselt.

Registrite kirjeldus on toodud Lisa 5 – Lisa 9. Iga registri kohta on esitatud olemi-suhte diagrammid, olemi-suhte diagrammidel esitatud olemitüüpide ja atribuutide sõnalised kirjeldused ning süsteemi jaoks huvipakkuvate seisundite olemasolul ka registri põhiobjekti seisundidiagramm. Olemitüüpide ja atribuutide nimed esitatakse tekstikirjelduses nii eesti kui ka inglise keeles, et lihtsustada lugejal seose loomist andmebaasi disaini mudeliga, kus tabelite ja veergude nimed on inglise keeles.

3.1.1 Viiside allsüsteem

Viiside allsüsteemi eesmärgid on järgmised.

- Võimaldada hoida ja hallata (lisada, muuta, kustutada) kõiki viise ja viisidega seotud andmeid.
- Võimaldada saada ülevaadet olemasolevatest viisidest ja nende andmetest, sh kasutades soovitud kriteeriumitele vastavat otsingut.



Joonis 2. Viiside funktsionaalse allsüsteemi äriarhitektuuri joonis.

Järgnevalt on toodud viiside allsüsteemi kasutusjuhtude kirjeldused.

- **Kasutusjuht**: Sisesta viis

  Tegutsejad: Toimetaja või administraator

  Kirjeldus: Tegutseja alustab uue rahvaviisi sisestamist. Tegutseja sisestab viisi põhiandmed ning salvestab need. Süsteem kontrollib sisestatud andmete vastavust nõuetele ning salvestab viisi kirje andmebaasi. Eduka salvestamise järgselt sisestab tegutseja viisi alamobjektide andmed: viisi tegija, koht, esitus, laul, muusikalised tunnused, noodistus ja kodeering. Tegutseja salvestab iga alamobjekti andmed eraldi analoogselt viisi põhiandmete salvestamisega.

- **Kasutusjuht**: Muuda viisi

  Tegutsejad: Toimetaja või administraator

  Kirjeldus: Tegutseja alustab viisi muutmist. Tegutseja muudab soovitud andmed ja salvestab need. Süsteem kontrollib andmete vastavust nõuetele ning salvestab muudetud andmed andmebaasi.

- **Kasutusjuht**: Vaata viisi

  Tegutsejad: Tuvastamata kasutaja, tavakasutaja, toimetaja või administraator

  Kirjeldus: Tegutseja saab vaadata viise nimekirjana. Nimekirjast saab avada viisi detailandmete vaate. Viisi detailandmete vaatest saab avada iga alamobjekti (viisi tegija, koht, esitus, laul, muusikalised tunnused, noodistus, kodeering) detailandmete vaate. Toimetaja ja administraator saavad vaadata ka viisi sündmusi.

- **Kasutusjuht**: Kustuta viis

  Tegutsejad: Administraator

  Kirjeldus: Tegutseja alustab viisi kustutamist. Süsteem küsib tegutsejalt täiendavat kinnitust kustutamise soovi kohta. Tegutseja kinnitab kustutamise soovi. Süsteem kustutab viisi ja viisi alamobjektide andmed andmebaasist.

  Märkus: Administraator kustutab viisi ainult juhul, kui sama viis (erinevate viidete kombinatsiooniga) on andmebaasis juba olemas.

- **Kasutusjuht**: Otsi viisi

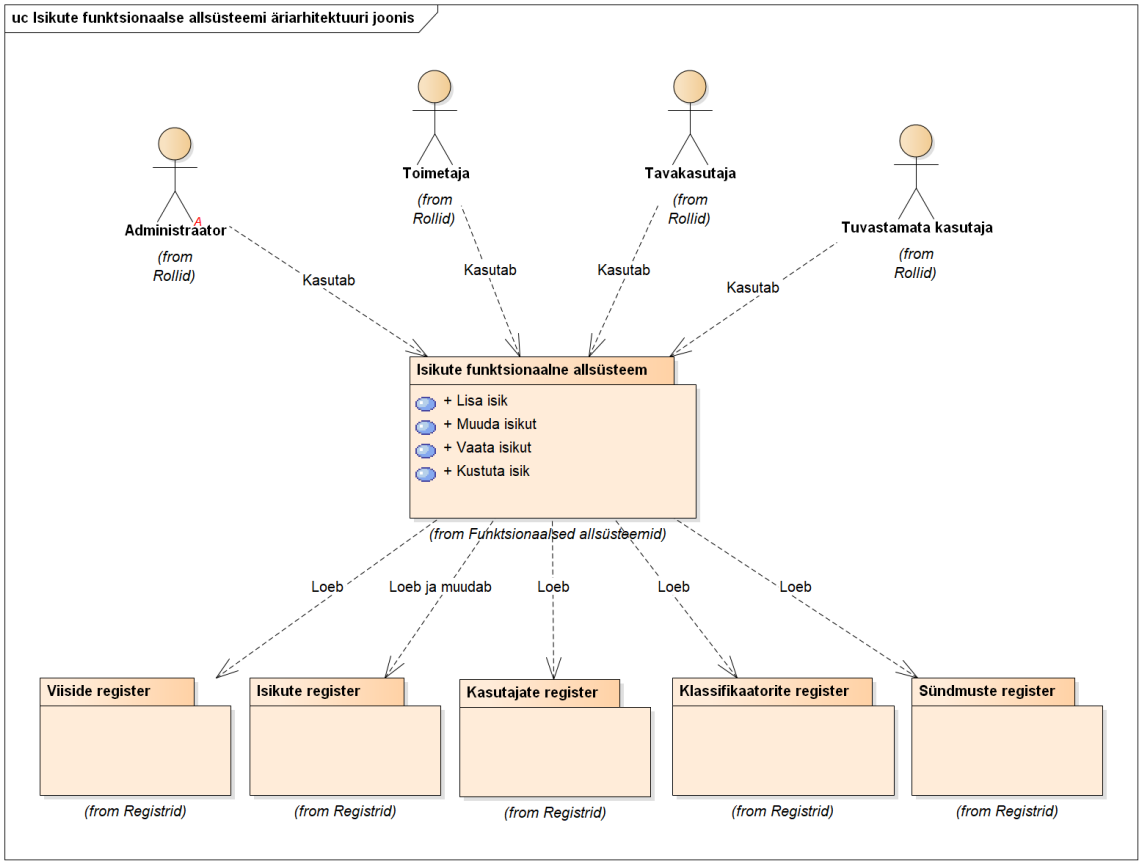
  Tegutsejad: Tuvastamata kasutaja, tavakasutaja, toimetaja või administraator

  Kirjeldus: Tegutseja alustab viisi otsimist sisestades otsinguväljale otsingu kriteeriumi. Süsteem otsib andmebaasist sellele kriteeriumile vastavad viisid ja kuvab need tegutsejale nimekirjana.

  Märkus: Viisi saab otsida viidete ja kartoteegikaardi numbri alusel.

3.1.2 Isikute allsüsteem

Isikute allsüsteemi eesmärgiks on võimaldada hallata (lisada, muuta, kustutada) viisiga seotud isikuid (nt viisikoguja, tekstikoguja, esitaja).



Joonis 3. Isikute funktsionaalse allsüsteemi äriarhitektuuri joonis.

Järgnevalt on toodud isikute allsüsteemi kasutusjuhtude kirjeldused.

- **Kasutusjuht**: Lisa isik

  Tegutsejad: Toimetaja või administraator

  Kirjeldus: Tegutseja alustab uue isiku sisestamist. Tegutseja sisestab isiku andmed ning salvestab need. Süsteem kontrollib sisestatud andmete vastavust nõuetele ning salvestab isiku kirje andmebaasi.

- **Kasutusjuht**: Muuda isik

  Tegutsejad: Toimetaja või administraator

  Kirjeldus: Tegutseja alustab isiku muutmist. Tegutseja muudab soovitud andmed ja salvestab need. Süsteem kontrollib andmete vastavust nõuetele ning salvestab muudetud andmed andmebaasi.

- **Kasutusjuht**: Vaata isikut

  Tegutsejad: Tuvastamata kasutaja, tavakasutaja, toimetaja või administraator

  Kirjeldus: Tegutseja saab vaadata isikuid nimekirjana. Nimekirjast saab avada isiku detailandmete vaate. Toimetaja ja administraator saavad vaadata ka isiku sündmusi.

- **Kasutusjuht**: Kustuta isik

  Tegutsejad: Administraator

  Kirjeldus: Tegutseja alustab isiku kustutamist. Süsteem küsib tegutsejalt täiendavat kinnitust kustutamise soovi kohta. Tegutseja kinnitab kustutamise soovi. Süsteem kustutab isiku andmebaasist.

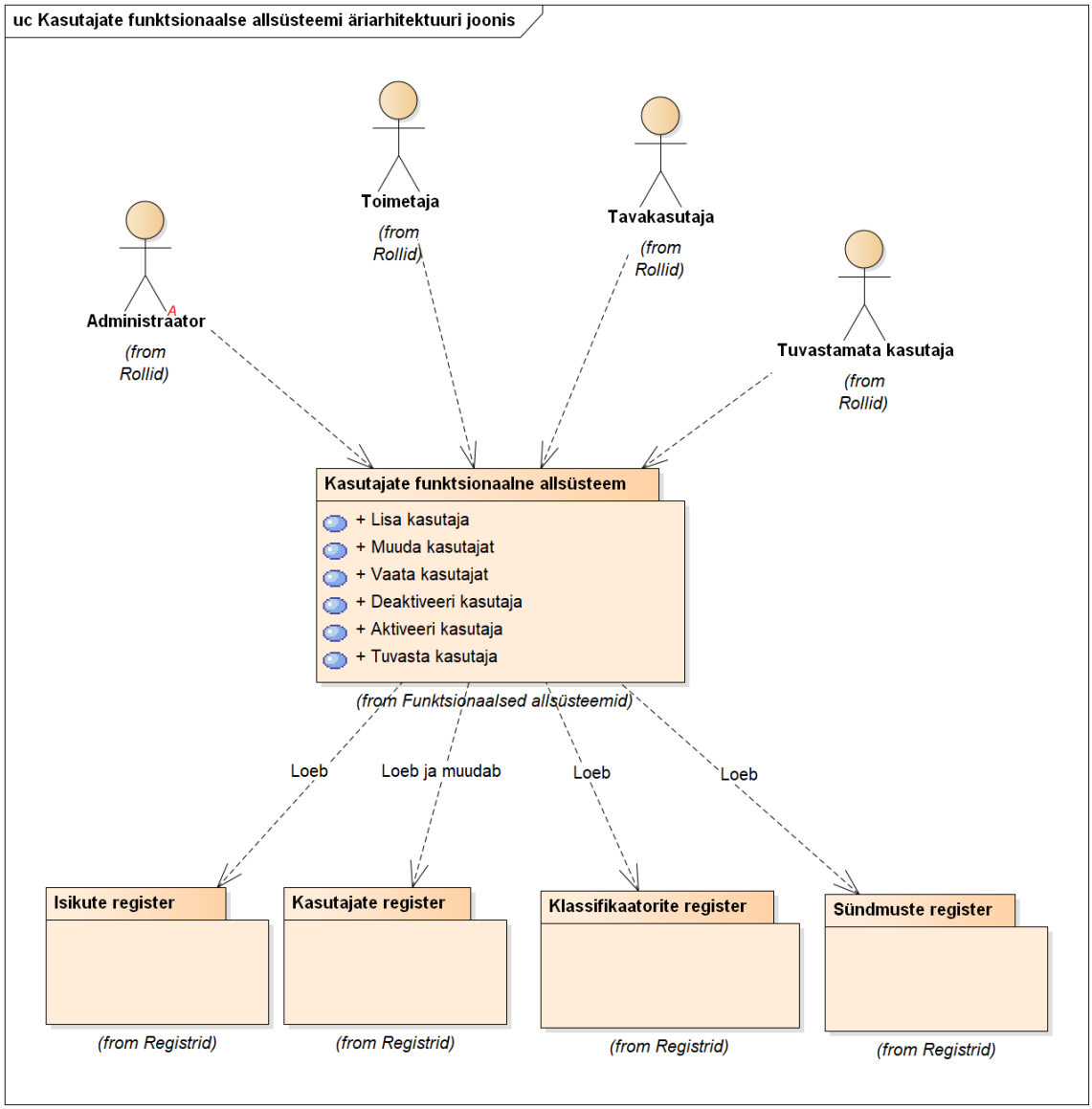
  Märkus: Kustutada saab ainult neid isikuid, kelle andmed ei ole kasutusel, st seotud teiste andmetega andmebaasis. Administraator kustutab isiku ainult juhul, kui sama isik (erineva nimede kombinatsiooniga) on andmebaasis juba olemas.

3.1.3 Kasutajate allsüsteem

Kasutajate allsüsteemi eesmärgid on järgmised.

- Võimaldada hallata süsteemi kasutajaid (lisada, muuta andmeid – sh parooli, muuta seisundit mitteaktiivseks ja tagasi aktiivseks).
- Võimaldada kasutajate tuvastamist süsteemi kasutamisel.

Süsteemi saab lisada järgmisi rolle: administraator, toimetaja ja tavakasutaja. Igal kasutajal saab korraga olla ainult üks roll. Lisaks on rollina käsitletav tuvastamata kasutaja, kelleks saab olla iga isik, kellel on juurdepääs süsteemi kasutajaliidesele. Eelkõige on see vajalik tulevikuvaates, kui viiside infosüsteem saab olema avalikult kättesaadav.



Joonis 4. Kasutajate funktsionaalse allsüsteemi äriarhitektuuri joonis.

Järgnevalt on toodud kasutajate allsüsteemi kasutusjuhtude kirjeldused.

- **Kasutusjuht**: Lisa kasutaja

  Tegutsejad: Administraator

  Kirjeldus: Tegutseja alustab uue kasutaja sisestamist. Tegutseja sisestab kasutaja andmed ning salvestab need. Süsteem kontrollib sisestatud andmete vastavust nõuetele ning salvestab kasutaja kirje andmebaasi.

- **Kasutusjuht**: Muuda kasutajat

  Tegutsejad: Tavakasutaja, toimetaja või administraator

  Kirjeldus: Tegutseja alustab kasutaja muutmist. Tegutseja muudab soovitud andmed ja salvestab need. Süsteem kontrollib andmete vastavust nõuetele ning salvestab muudetud andmed andmebaasi.

  Märkused: Toimetaja ja tavakasutaja saavad muuta ainult enda parooli. Administraator saab muuta kõikide kasutajate kõiki andmeid. Rolli muutmisel kontrollib süsteem, et peale muutmist jääks andmebaasi vähemalt üks aktiivne administraatori rolliga kasutaja (vastasel juhul ei lubata rolli muuta).

- **Kasutusjuht**: Vaata kasutajat

  Tegutsejad: Tavakasutaja, toimetaja või administraator

  Kirjeldus: Tegutseja saab vaadata kasutajaid nimekirjana. Nimekirjast saab avada kasutaja detailandmete vaate. Toimetaja ja administraator saavad vaadata ka kasutaja sündmusi.

  Märkus: Tavakasutaja ja toimetaja saavad vaadata ainult enda kasutajaga seotud andmeid. Administraator saab vaadata kõikide kasutajate andmeid.

- **Kasutusjuht**: Deaktiveeri kasutaja

  Tegutsejad: Administraator

  Kirjeldus: Tegutseja alustab kasutaja muutmist. Tegutseja eemaldab märke „on aktiivne“ ja salvestab muudatuse. Süsteem kontrollib andmete vastavust nõuetele ning salvestab muudetud andmed andmebaasi.

  Märkus: Deaktiveeritud kasutaja ei saa süsteemi sisse logida. Süsteem kontrollib, et peale kasutaja deaktiveerimist jääks andmebaasi vähemalt üks aktiivne administraatori rolliga kasutaja (vastasel juhul ei lubata kasutajat deaktiveerida).

- **Kasutusjuht**: Aktiveeri kasutaja

  Tegutsejad: Administraator

  Kirjeldus: Tegutseja alustab kasutaja muutmist. Tegutseja lisab märke „on aktiivne“ ja salvestab muudatuse. Süsteem kontrollib andmete vastavust nõuetele ning salvestab muudetud andmed andmebaasi.

- **Kasutusjuht**: Tuvasta kasutaja

  Tegutsejad: Tavakasutaja, toimetaja või administraator

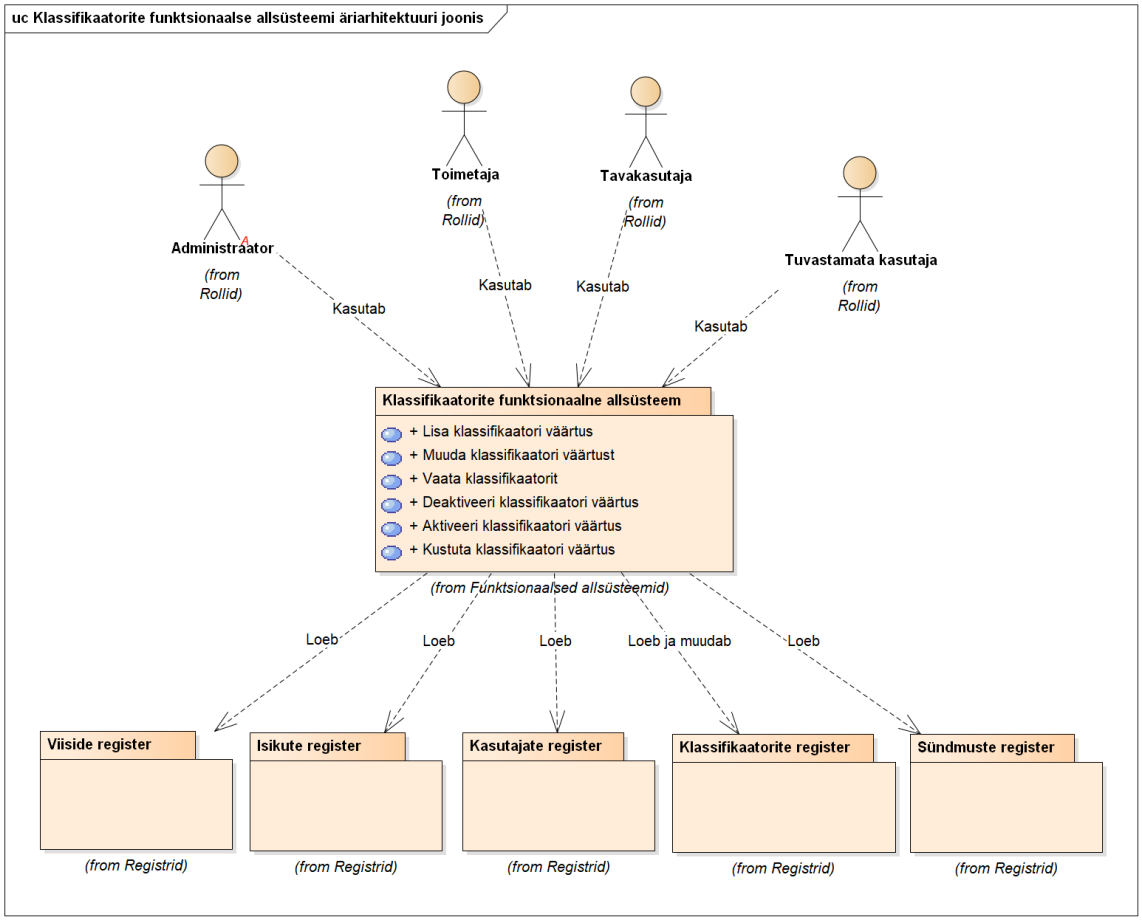
  Kirjeldus: Tegutseja alustab süsteemi sisselogimist sisestades enda kasutajanime ja parooli. Süsteem autendib kasutaja, st kontrollib tema identiteeti ja kas kasutaja seisund lubab süsteemi kasutada (kasutaja peab olema aktiivne). Peale edukat autentimist autoriseerib süsteem kasutaja, st kontrollib tema kasutajarolli ja võimaldab kasutajal teha temale määratud rollile lubatud tegevusi.

  Märkus: Kasutusjuhtu „Tuvasta kasutaja“ kasutatakse kõikides funktsionaalsetes allsüsteemides.

3.1.4 Klassifikaatorite allsüsteem

Klassifikaatorite allsüsteemi eesmärgid on järgmised.

- Võimaldada hallata (lisada, muuta, kustutada) klassifikaatorite väärtuseid, mida kasutatakse teiste andmete liigitamiseks.
- Saada ülevaadet olemasolevatest klassifikaatoritest ja nende väärtustest.



Joonis 5. Klassifikaatorite funktsionaalse allsüsteemi äriarhitektuuri joonis.

Järgnevalt on toodud klassifikaatorite allsüsteemi kasutusjuhtude kirjeldused.

- **Kasutusjuht**: Lisa klassifikaatori väärtus

  Tegutsejad: Toimetaja või administraator

  Kirjeldus: Tegutseja alustab uue klassifikaatori väärtuse sisestamist. Tegutseja sisestab nõutud andmed ning salvestab need. Süsteem kontrollib sisestatud andmete vastavust nõuetele ning salvestab klassifikaatori väärtuse kirje andmebaasi.

- **Kasutusjuht**: Muuda klassifikaatori väärtust

  Tegutsejad: Toimetaja või administraator

  Kirjeldus: Tegutseja alustab klassifikaatori väärtuse muutmist. Tegutseja muudab soovitud andmed ja salvestab need. Süsteem kontrollib andmete vastavust nõuetele ning salvestab muudetud andmed andmebaasi.

  Märkused: Peale klassifikaatori väärtuse nimetuse muutmist kuvatakse uus nimetus igal pool, kus see konkreetne klassifikaatori väärtus kasutusel on. Muuta

ei saa viisi seisundi liikide ja rollide väärtuseid. Nende muutmine tähendab, et on toimunud muudatus infosüsteemi toimimises, mis tähendab ka muudatusi tarkvaras. Vastavate klassifikaatorite väärtused muudetakse siis arendajate poolt otse andmebaasis.

- **Kasutusjuht**: Vaata klassifikaatorit
  Tegutsejad: Tuvastamata kasutaja, tavakasutaja, toimetaja või administraator
  Kirjeldus: Tegutseja saab vaadata klassifikaatoreid nimekirjana. Nimekirjast saab avada klassifikaatori väärtuste nimekirja vaate. Sellest nimekirjast saab avada iga klassifikaatori väärtuse detailandmete vaate. Toimetaja ja administraator saavad vaadata ka klassifikaatori sündmusi.

- **Kasutusjuht**: Deaktiveeri klassifikaatori väärtus
  Tegutsejad: Toimetaja või administraator
  Kirjeldus: Tegutseja alustab klassifikaatori väärtuse muutmist. Tegutseja eemaldab märke „on aktiivne“ ja salvestab muudatuse. Süsteem kontrollib andmete vastavust nõuetele ning salvestab muudetud andmed andmebaasi.
  Märkus: Deaktiveeritud klassifikaatori väärtust ei kuvata tegutsejale vastavas valikmenüüs ja seda ei ole võimalik enam andmete liigitamiseks kasutada. Kohtades, kus klassifikaatori väärtust on liigitamiseks kasutatud, kuvatakse seda endiselt.

- **Kasutusjuht**: Aktiveeri klassifikaatori väärtus
  Tegutsejad: Toimetaja või administraator
  Kirjeldus: Tegutseja alustab klassifikaatori väärtuse muutmist. Tegutseja lisab märke „on aktiivne“ ja salvestab muudatuse. Süsteem kontrollib andmete vastavust nõuetele ning salvestab muudetud andmed andmebaasi.

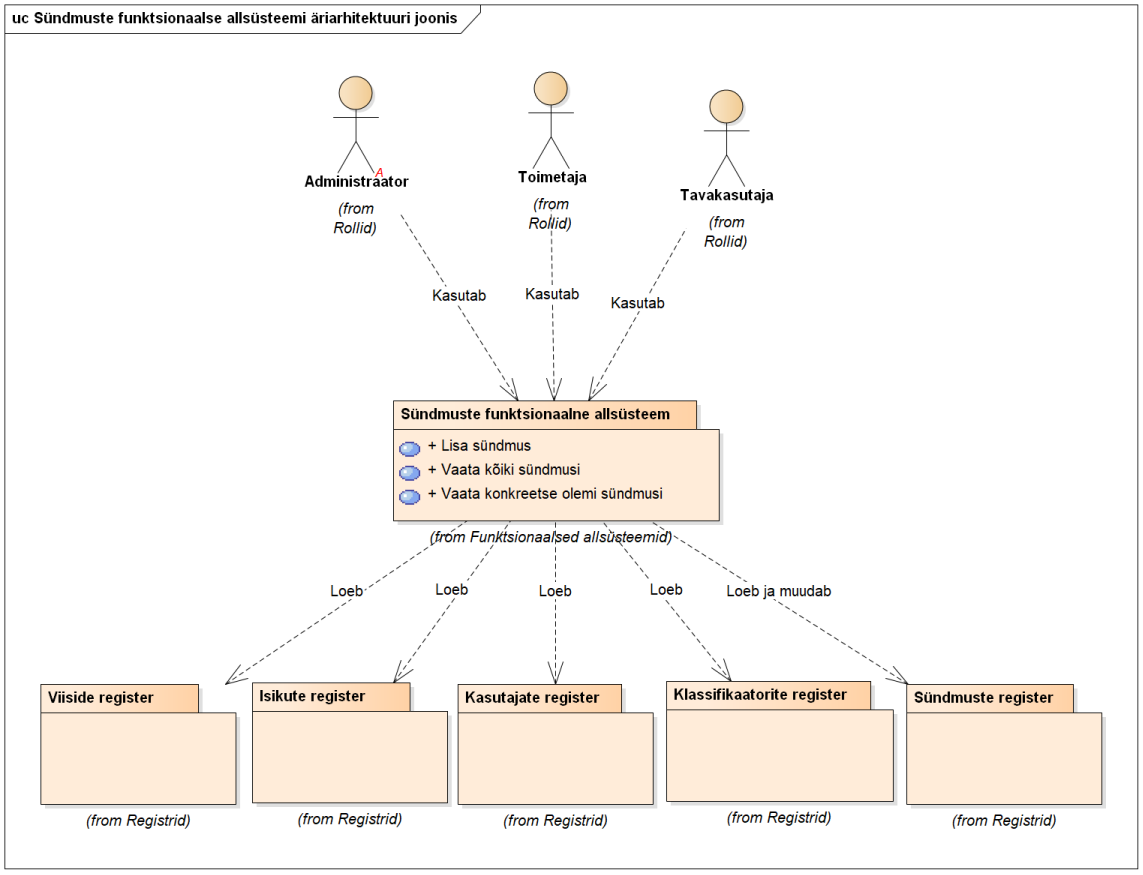
- **Kasutusjuht**: Kustuta klassifikaatori väärtus
  Tegutsejad: Administraator
  Kirjeldus: Tegutseja alustab klassifikaatori väärtuse kustutamist. Süsteem küsib tegutsejalt täiendavat kinnitust kustutamise soovi kohta. Tegutseja kinnitab kustutamise soovi. Süsteem kustutab klassifikaatori väärtuse andmebaasist.

Märkused: Kustutada saab ainult neid klassifikaatorite väärtuseid, mida ei kasutata andmebaasis ühegi olemi iseloomustamiseks (st mida pole kordagi kasutatud). Kustutada ei saa viisi seisundi liikide ja rollide väärtuseid.

3.1.5 Sündmuste allsüsteem

Sündmuste allsüsteemi eesmärgid on järgmised.

- Hoida kasutaja tegevuste (andmete lisamine, muutmine ja kustutamine) ajalugu.
- Võimaldada tuvastada ja vaadata, kes ja millal on andmeid lisanud või muutnud.



Joonis 6. Sündmuste funktsionaalse allsüsteemi äriarhitektuuri joonis.

Järgnevalt on toodud sündmuste allsüsteemi kasutusjuhtude kirjeldused. Andmete all on mõeldud viiside ja viisi alamobjektide, isikute, kasutajate ning klassifikaatorite andmeid. Tegevused on lisamine, muutmine (sh deaktiveerimine ja aktiveerimine) ja kustutamine.

- **Kasutusjuht**: Lisa sündmus

  Tegutsejad: Tavakasutaja, toimetaja või administraator

  Kirjeldus: Tegutseja teeb tegevuse andmetega. Süsteem salvestab lisaks andmetele ka tehtud tegevusega seotud sündmuse logi.

- **Kasutusjuht**: Vaata kõiki sündmusi

  Tegutsejad: Toimetaja või administraator

  Kirjeldus: Tegutseja saab vaadata sündmuste ajalugu koondina põhiobjekti (põhiolemitüübi) kõikide sündmuste kohta: viiside sündmused, isikute sündmused, kasutajate sündmused ja klassifikaatori sündmused. Tegutseja näeb lisamise, muutmise ja kustutamise sündmusi. Näiteks viiside sündmuste all kuvatakse kõikide viisidega ja viisi alamobjektidega seotud sündmused. Iga sündmuse kirje juurest saab avada sündmuse detailandmete vaate.

- **Kasutusjuht**: Vaata konkreetse olemi sündmusi

  Tegutsejad: Toimetaja või administraator

  Kirjeldus: Tegutseja saab vaadata sündmuste ajalugu olemi detailandmete vaatest. Tegutseja näeb ainult lisamise ja muutmise sündmusi. Näiteks viisi detailandmete vaates näeb selle viisiga ja viisi alamobjektidega seotud sündmusi. Iga sündmuse kirje juurest saab avada sündmuse detailandmete vaate.

  Märkus: Kustutatud olemite sündmusi tegutseja ei näe, kuna kustutatud olemi kirjet tegutsejale ei kuvata.

3.1.6 Mittefunktsionaalsed nõuded

Tabel 2 esitab viiside infosüsteemi mittefunktsionaalsed nõuded.

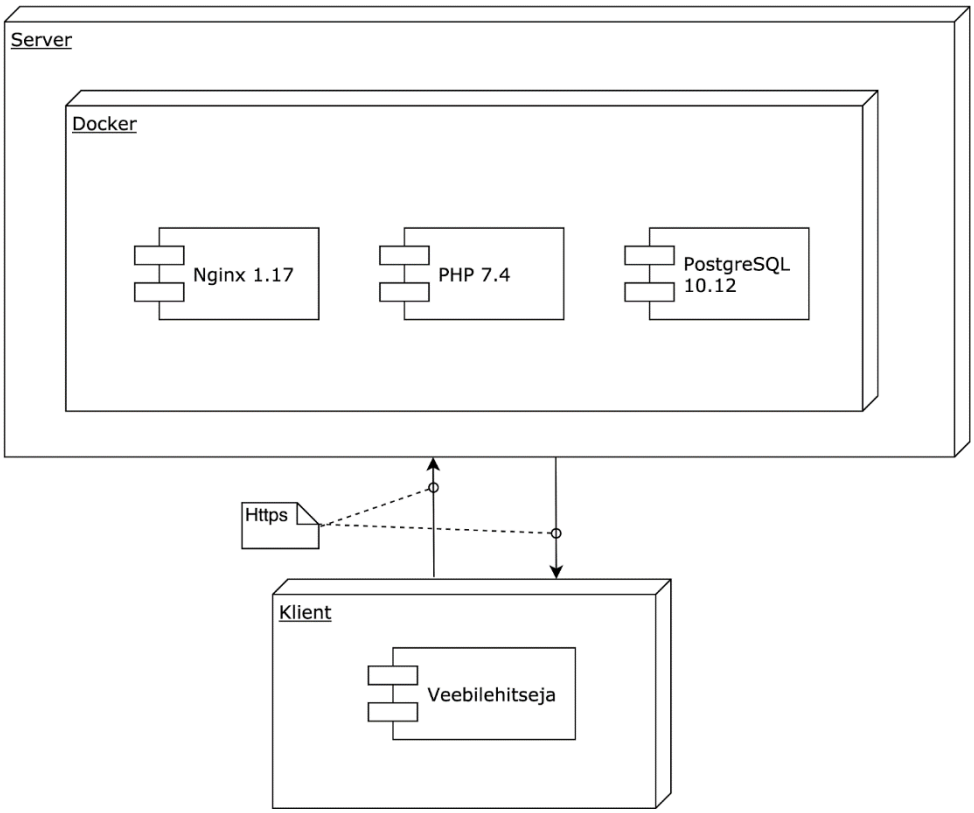
Tabel 2. Viiside infosüsteemi mittefunktsionaalsed nõuded.

Tüüp	**Nõude kirjeldus**
andmekvaliteet	Rakendus peab salvestama automaatselt informatsiooni kirjete sisestamise ja muutmise aja ning kasutajate tehtud tegevuste kohta.
arhitektuur	Andmebaas, SQL-skriptid ja rakendus peavad kasutama UTF-8 kodeeringut.
arhitektuur	Rakenduse äriloogika tuleb võimalusel realiseerida andmebaasist eraldi, sõltumatus rakenduskihis.
dokumentatsioon	Kogu rakenduse dokumentatsioon peab olema kirjutatud eesti keeles.
kasutajaliides	Kasutajaliides peab olema veebipõhine.

Tüüp	Nõude kirjeldus
kasutajaliides	Kasutajaliides peab olema eesti keeles ja lihtsalt tõlgitav teistesse keeltesse.
kasutajaliides	Kasutajaliides peab olema kasutatav enamlevinud veebilehitsejatega, muuhulgas nutiseadmetel (Android ja iOS).
kasutajaliides	Rakendus peab olema graafiliselt skaleeruv ja kasutatav ilma horisontaalsete kerimisribadeta kõigi enamlevinud arvutimonitoride resolutsioonidega.
kasutajaliides	Kui on salvestamata andmeid, siis kasutajaliides peab enne lehelt lahkumist alati küsima kinnitust.
kasutajaliides	Kasutajaliides peab alati küsima kinnitust enne andmete kustutamist.
kasutajaliides	Päringu vastusena kuvatud tabeli veerge peab olema võimalik andmete järjekorras sorteerida. Sorteerimine peab olema Eesti tähestikule vastav.
serveri tarkvara	Süsteem peab andmete hoidmiseks kasutama PostgreSQL andmebaasisüsteemi abil loodud andmebaasi.
turvalisus	Süsteem peab olema kaitstud SQL-süstide ja skriptisüstide vastu.
turvalisus	Kliendi ja serveri vahel peab autenditud kasutajasessioonide korral olema sessioon krüpteeritud HTTPS protokolli kasutades.
turvalisus	Andmebaasis hoitakse parooli räsiväärtus, mis on leitud selle parooli ja soola põhjal.
turvalisus	Andmete loomise, muutmise ja kustutamise tegevused peab logima.

3.2 Tehniline arhitektuur

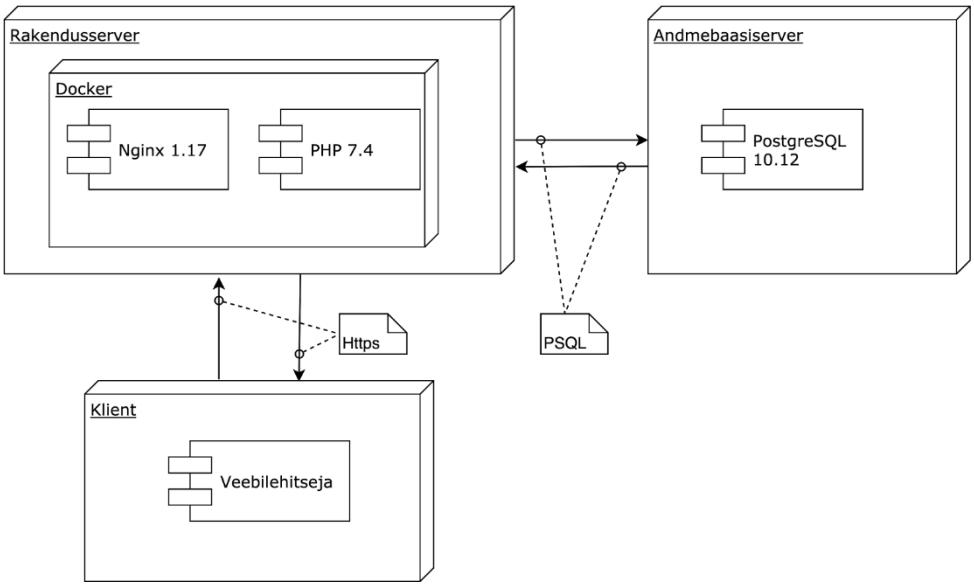
Arenduskeskkond on üles ehitatud konteineritel põhinevale lähenemisele. Docker platvormi tööriista Compose abil loodi keskkond, kuhu paigutati veebiserver Nginx, PHP ning PostgreSQL andmebaas. Suhtlus serveri ja kliendi vahel käib HTTPS protokolli kasutades. Arenduskeskkonna tehniline arhitektuur on esitatud Joonis 7.



Joonis 7. Viiside infosüsteemi arenduskeskkonna tehniline arhitektuur.

Serveris paiknev rakendus loodi CakePHP raamistiku abil [11]. CakePHP on avatud lähtekoodiga PHP-põhine veebirakenduste arendamise raamistik. Töö autorid kasutasid CakePHP versiooni 3.8.11. Raamistik kasutab MVC (*Model-View-Controller*) disainimustrit [41]. Seda mustrit kasutades jagatakse rakenduse loogika kolme komponendi vahel – mudel, vaade ja kontroller. Mudelis kirjeldatakse äriloogika rakenduses kasutatavatele andmetele. Vaate abil kuvatakse kasutajaliides. Kontroller juhib suhtlust kasutajaga – kuidas kasutada mudeleid ning kuidas edastada mudelites olevaid andmeid vaadetele.

EKM kasutab viiside infosüsteemi juurutamisel samuti Dockeri konteineripõhist lähenemist, sarnaselt sellele nagu tegid seda töö autorid arenduskeskkonnas. Peamine erinevus on see, et andmebaas tõstetakse konteinerist üle EKM-i andmebaasiserverisse. Tehniline arhitektuur tellija juures on esitatud Joonis 8.



Joonis 8. Viiside infosüsteemi tehniline arhitektuur peale juurutamist.

3.3 Disain

Järgnevalt kirjeldatakse andmebaasi ja kasutajaliidese disaini põhimõtteid.

3.3.1 Andmebaasi disain

Andmebaasi füüsilise disaini loomisel võeti aluseks registrite olemi-suhte diagrammid ja olemitüüpide atribuutide juures toodud lisakitsendused.

Füüsilise disaini andmebaasi diagrammid on toodud registrite kaupa Lisa 10 – Lisa 14. Diagrammidel ei esitata CHECK kitsendusi ja domeene.

Atribuutide alusel loodud veergudele jõustatud lisakitsendused on kirjeldatud üldistatud kujul ning iga tingimuse kohta on toodud koodi näide (vt Joonis 9).

Teksti hoidmiseks mõeldud atribuutide kitsendused.

- Ei tohi olla tühi string.
- Ei tohi olla ainult tühimärkidest koosnev string.
- Kui olemitüüpi peamiselt iseloomustavatest väärtustest ei ole ükski kohustuslik, siis vähemalt üks atribuutide väärtustest peab olema registreeritud. Näiteks olemitüübi *Isik* juures peab olema registreeritud vähemalt üks kolmest – eesnimi või perekonnanimi või hüüdnimi.

Temporaalsete (ajaandmete) hoidmiseks mõeldud atribuutide kitsendused.

- Peab olema mingis kindlas vahemikus (otspunktid kaasa arvatud).
- Ei tohi olla varasem selle atribuudi väärtusest, mille esitatavale sündmusele see kronoloogiliselt järgneb.

```
CONSTRAINT CK_title_not_empty_string CHECK (VALUE <> '')
CONSTRAINT CK_title_not_only_whitespace CHECK (VALUE !~ '^[[:space:]]+$')
CONSTRAINT CK_persons_name_exist CHECK (
    (given_name IS NOT NULL) OR
    (surname IS NOT NULL) OR
    (nickname IS NOT NULL)
)
CONSTRAINT CK_timestamp_check CHECK (
    VALUE BETWEEN '2020-01-01' AND '2200-01-01'
)
CONSTRAINT CK_tunes_modified_no_earlier_than_created CHECK (
    modified >= created
)
```

Joonis 9. Atribuutide kitsenduste koodi näited.

Andmebaasi loodi järgmised indeksid:

- indeksid primaar- ja unikaalsuse kitsendustega hõlmatud veergudele (loodi automaatselt andmebaasisüsteemi süsteemi poolt);
- indeksid välisvõtmetele;
- täiendavad sekundaarsed indeksid hierarhiliste klassifikaatorite *lft* ja *rght* veergudele;
- unikaalsed UPPER funktsioonil põhinevad indeksid klassifikaatorite nimetuste ja kasutajanime veergudele (sellistele veergudele unikaalsuse kitsendusi ei loodud).

Indeksi tüübina kasutati PostgreSQL poolt vaikimisi kasutatavat B-puu indeksi tüüpi.

Andmebaasi loodi trigeri funktsioonid ja trigerid:

- viisi seisundite muutmiseks. Triger käivitub pärast tabeli *tunes* (Viis) veeru *verified_by* (kontrollija) lisamist või muutmist. Kui kontrollija on määratud, muudetakse viisi seisundiks „Kontrollitud“, vastasel juhul muudetakse viisi seisundiks „Sisestatud“;

- vähemalt ühe aktiivse administraatori rolliga kasutaja tagamiseks. Triger käivitub enne tabeli *users* (Kasutaja) veergude *user_role_type_id* (roll) ja *is_active* (on aktiivne) muutmist. Kui peale muudatust ei jääks andmebaasi ühtegi aktiivset administraatori rolliga kasutajat, siis tagastatakse vastav veateade ja muudatust ei tehta.

Kõik loodud andmebaasiobjektid paigutati ühte skeemi. Loodud andmebaasis on 51 baastabelit, milles on kokku 354 veergu. Veergude omaduste kirjeldamiseks kasutati 12 domeeni. Andmebaasis on loodud kaks trigerit. Vaateid, hetktõmmiseid ehk materialiseeritud vaateid ja mitte-trigeri funktsioone/protseduure töö tulemusena ei loodud.

Andmebaasi loodi rakendusele vastav kasutaja, millele anti tööks minimaalselt vajalik õiguste kogum.

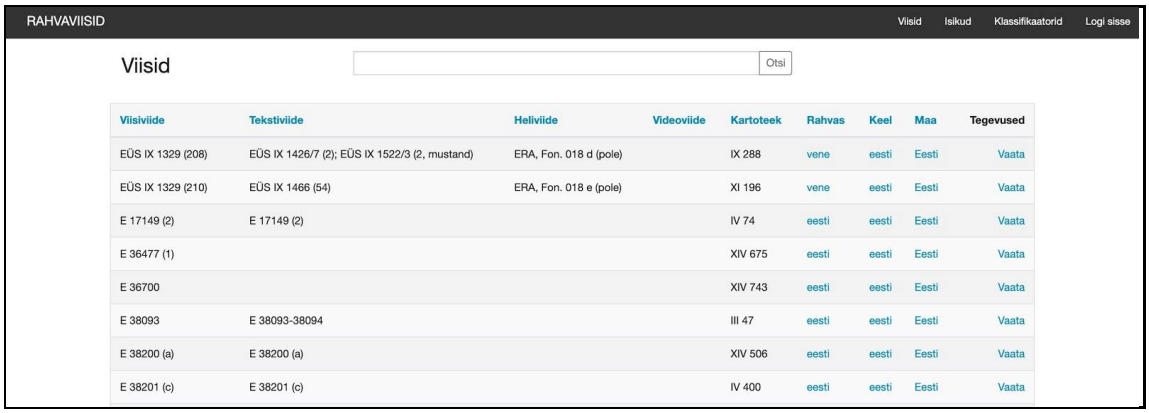
3.3.2 Kasutajaliidese disain

Kasutajaliidese aluseks on CakePHP raamistiku abil genereeritud vaated. CakePHP genereerib vaikimisi kõikidele tabelitele neli vaadet:

- *index* – nimekiri tabeli ridadest (olemitest),
- *view* – tabeli rea (olemi) detailandmed,
- *add* – tabeli rea (olemi) lisamine,
- *edit* – tabeli rea (olemi) muutmine.

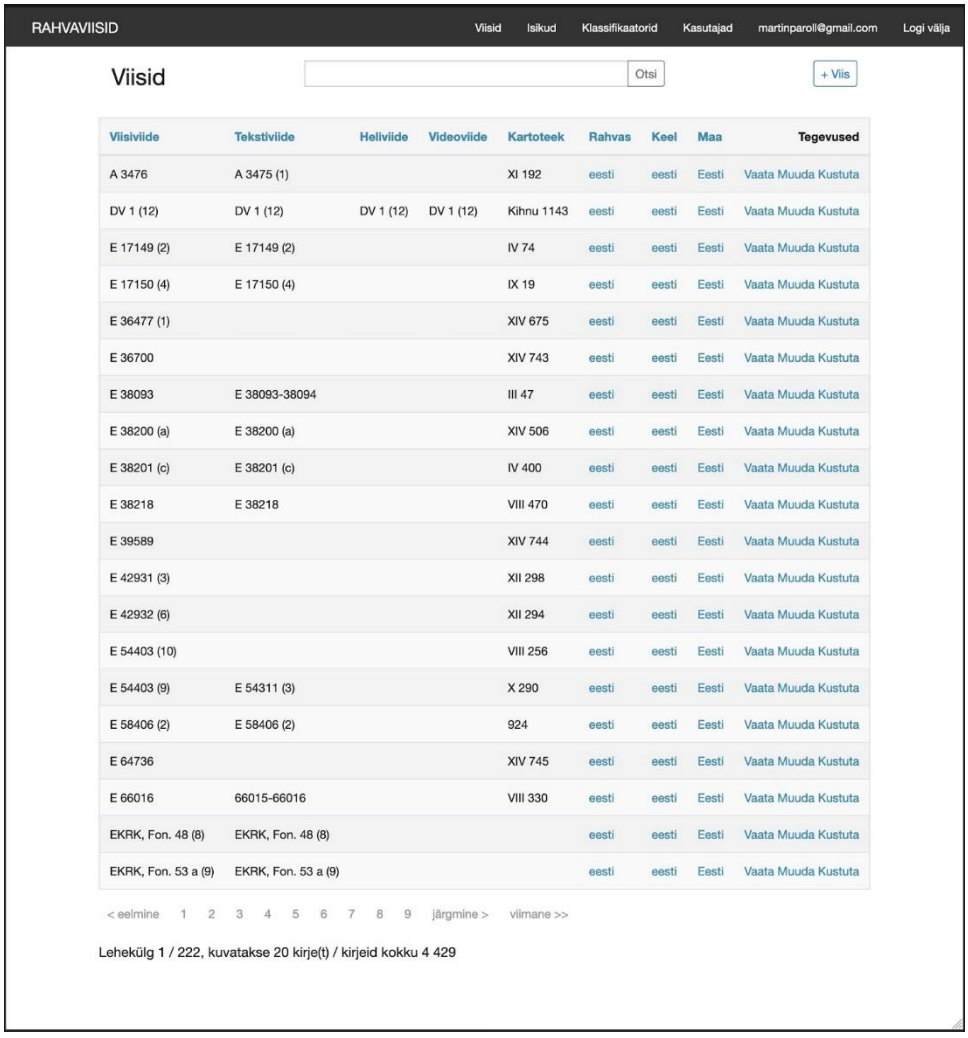
CakePHP kasutab veebilehtede kujundamisel Zurb Foundation veebiraamistikku. Töö autorid asendasid selle Bootstrap 4 veebiraamistikuga, kuna nad olid sellega varasemalt kokku puutunud. Selleks kirjutati skriptide abiga ümber kõikide vaadete HTML (*Hypertext Markup Language*) kujunduse klassid.

Kasutajaliidese navigatsiooni loomisel võeti aluseks viiside infosüsteemi jaotus allsüsteemideks. Päises olevas horisontaalses menüüs (vt Joonis 10) on lisaks sisselogimisele kolm menüüpunkti: *Viisid*, *Isikud*, *Klassifikaatorid*. Kasutajaliidese avaleheks on viiside menüüpunkt.



Joonis 10. Kasutajaliidese avaleht.

Peale sisselogimist antakse kasutajale ligipääsud menüüpunktidele ja tegevustele, mis on lubatud kasutajale määratud rollile. Näiteks administraatori rollis kasutajale tekib juurde menüüpunkt *Kasutajad* ning võimalus lisada, muuta või kustutada viisi (vt Joonis 11).



Joonis 11. Kasutajaliidese avaleht administraatori rollis.

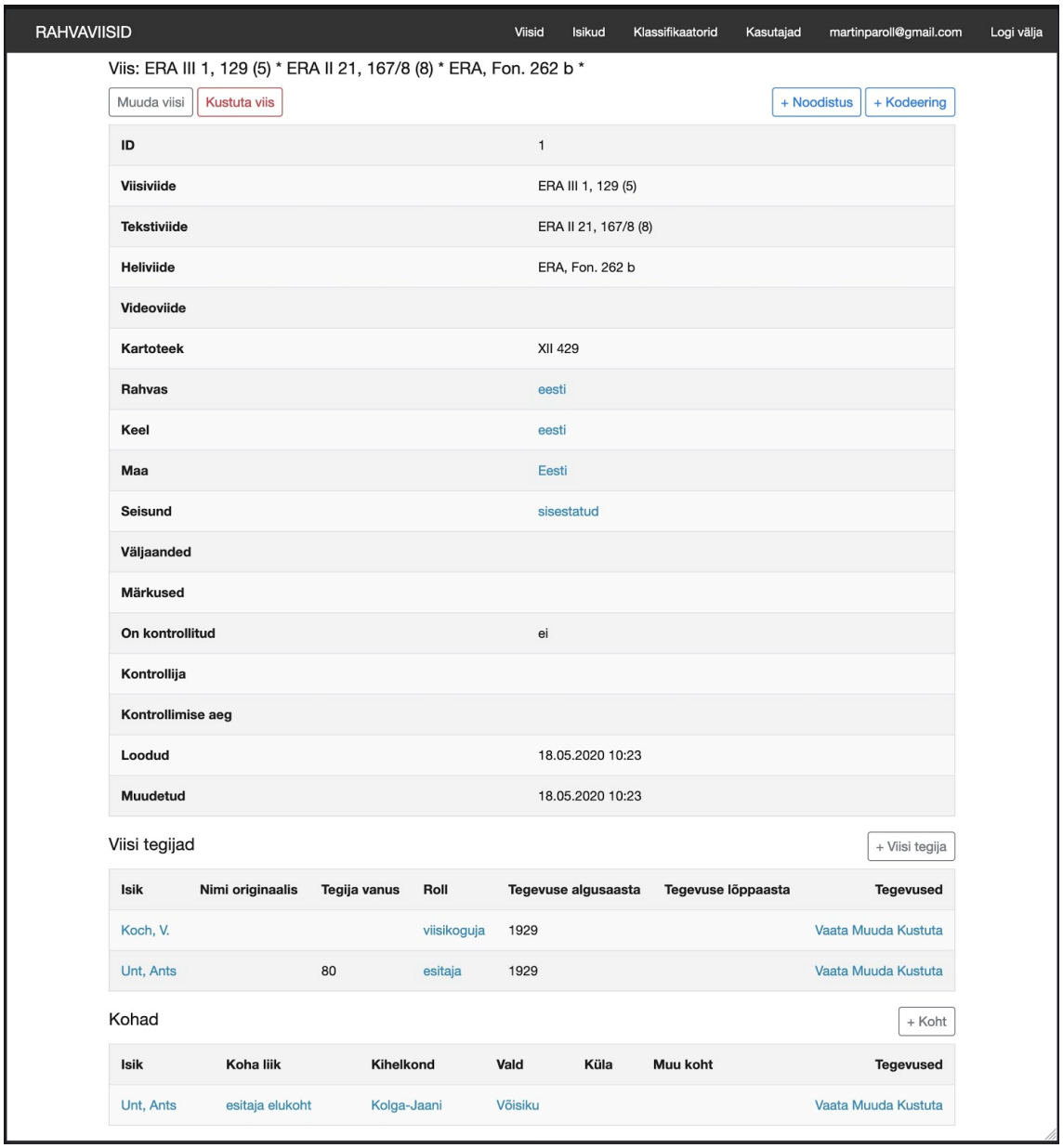
Viiside lehel on horisontaalse tabelina esitatud nimekiri andmebaasis olevatest viisidest koos võimalusega viisi otsida. Viise on võimalik veergude pealkirjade järgi sorteerida. Ühel lehel kuvatakse korraga 20 viisi. Iga viisi kirje lõpus on tegevused, mida selle viisiga teha saab.

Viisi detailandmed (vt Joonis 12) on esitatud vertikaalse tabelina, mille all kuvatakse nimekirjadena viisi alamobjektid. Viiside lehtedel olevatel nuppude stiilidel on järgmine tähendus:

- *primary* (sinine nupp) – järgmine loogiline tegevus;
- *danger* (punane nupp) – kustutamise tegevus;
- *secondary* (hall nupp) – muud tegevused.

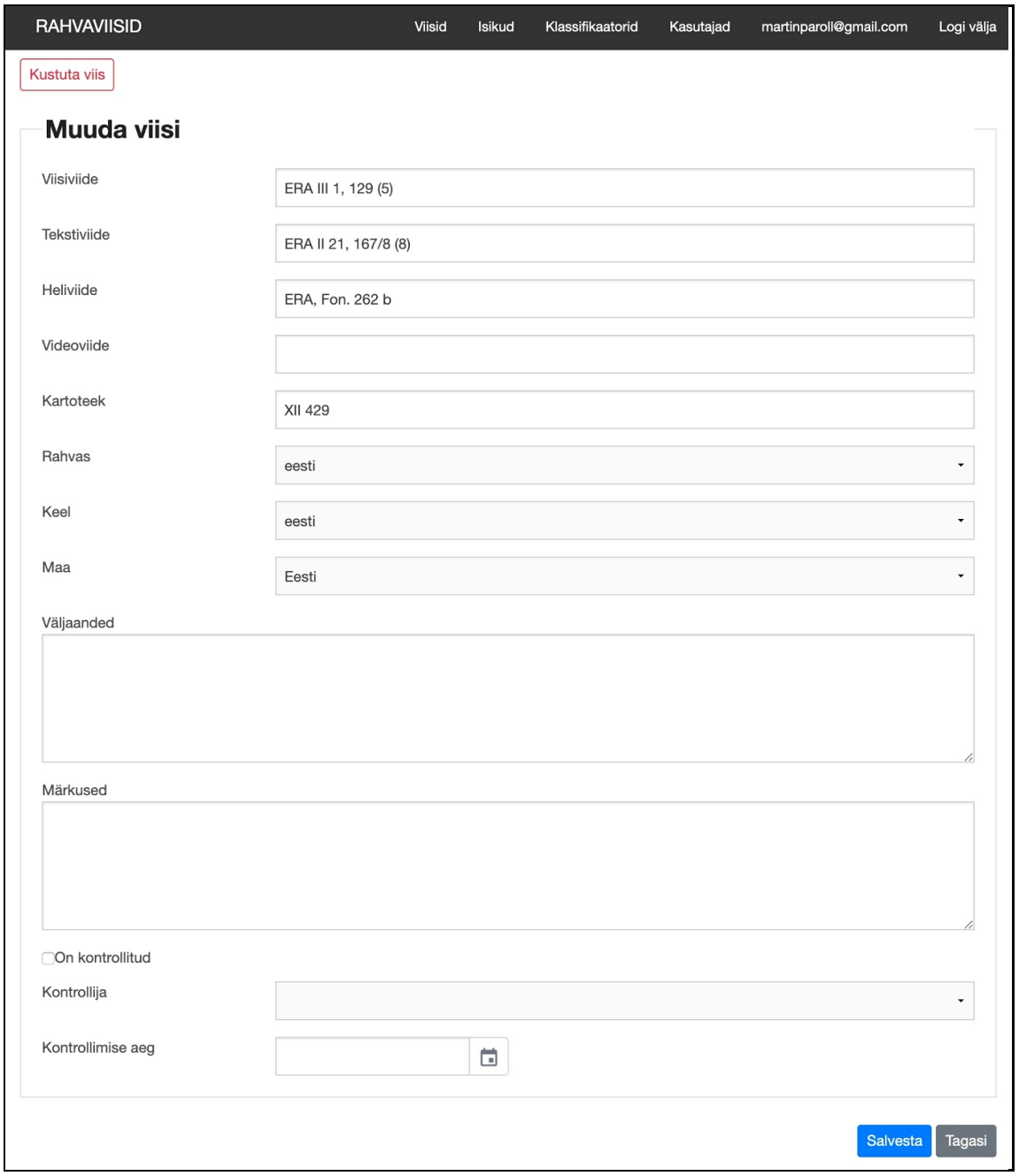
Isikute, kasutajate ja klassifikaatorite menüüpunktid on disainitud sarnaselt viiside menüüpunktile.

Enamik genereeritud vaadetest on loodud tabelitena. Kasutajaliides kohandab automaatselt kuvatava informatsiooni asetust vastavalt ekraani laiusele tänu Bootstrap 4 skaleeruvale ruudustikule ning tabeli stiilidele. Kõik vaated kasutavad veebilehitseja laiusel 992px ja laiema puhul mõlemas vaateakna ääres nihet 1/12 ekraani suurusest (vt Joonis 12).



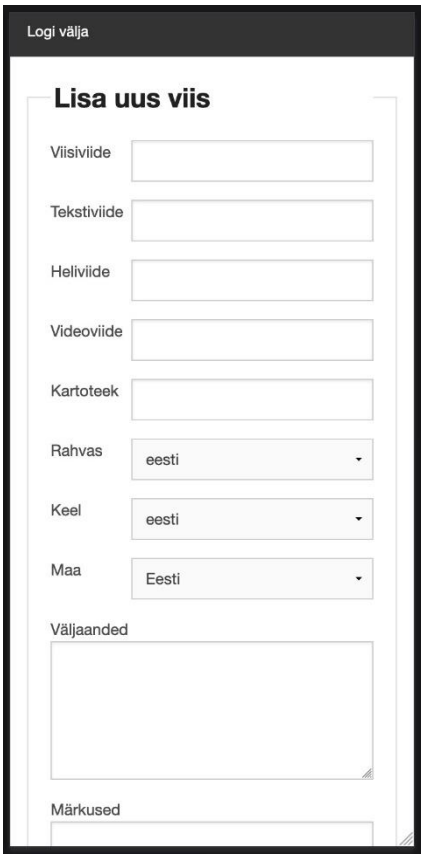
Joonis 12. Viisi detailandmete vaade veebilehitseja suurusel 1000px.

Kitsamate ekraanide puhul võetakse kasutusele kogu veebilehitseja akna laius (vt Joonis 13).



Joonis 13. Viisi muutmise vaade veebilehitseja suurusel 960px.

Nii andmete kuvamine kui ka sisestamine toimivad kasutajasõbralikult ka mobiililt (vt Joonis 14).



Joonis 14. Viisi lisamise vaade veebilehitseja suurusel 360px (Samsung Galaxy 9 mobiiltelefon).

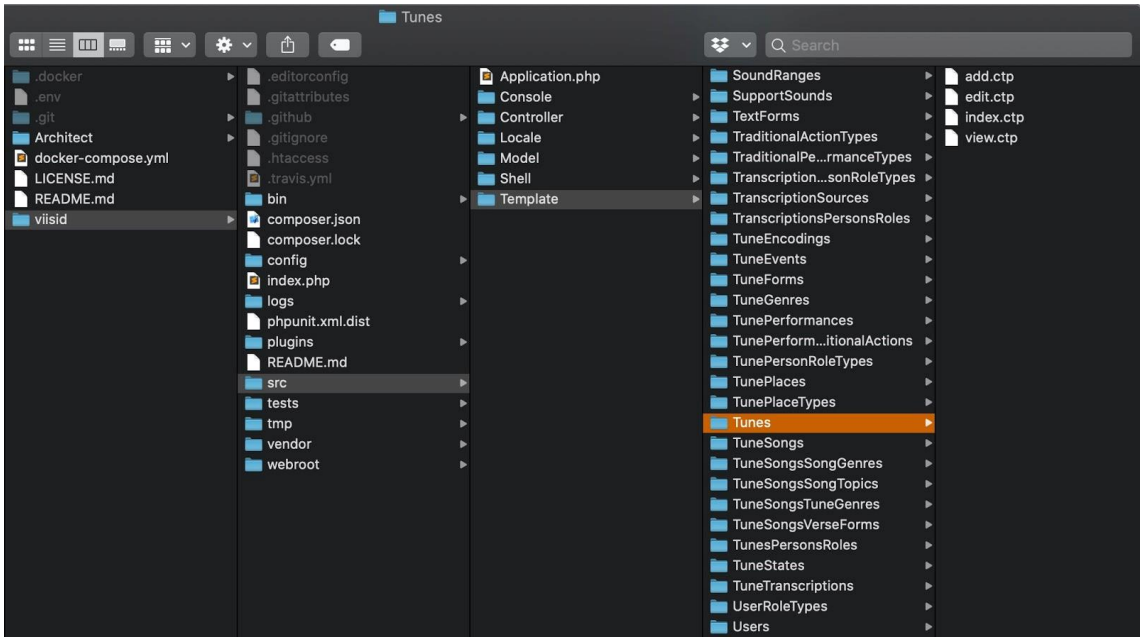
3.4 Kood

Rakenduse baaskood on enamuses genereeritud CakePHP raamistiku poolt, mis kasutab MVC disainimustrit. Rakenduses on kasutusel 51 baastabelit (tabelit). Raamistiku abil genereeriti tabelitele vastavad mudelid, vaated ja kontrollerid ning rakenduse töötamiseks vajalik baasstruktuur: baasrakendus (sh selle tööks vajalikud funktsioonid), vahekiht, konfiguratsiooni failid, marsruudid, automaattestid jms.

Koodi struktuur repositooriumis (vt Joonis 15) on järgnev.

1. Esimesel tasemel on EA mudelid (kaust *Architect*), rakenduse lähtekood (kaust *Viisid*) ning Docker Compose konfiguratsioon, litsents ning README fail.
2. Teisel tasemel asuvad CakePHP raamistiku ning kõikide konteinerite (Nginx, PHP, PostgreSQL) seadistused ning lähtestamise ja käivitamise jaoks vajalikud failid. Samuti asuvad siin kihis rakenduse logifailid ning automaattestid.
3. Viisid\src kataloogis asub rakenduse lähtekood. Seal on konsooli tööriistad (*Console*), kontrollerid (*Controller*), tõlkefailid (*Locale*), mudelid (*Model*) ja vaated (*Template*).

Joonis 15 esitab näitena objekti *Viis* (*Tunes*) vaated: lisa (*add.ctp*), muuda (*edit.ctp*), indeks (*index.ctp*) ja vaata (*view.ctp*).



Joonis 15. Koodi struktuur repositooriumis.

Järgnevalt on täpsemalt kirjeldatud mudeleid, vaateid ja kontrollereid ning töö autorite poolt juurde kirjutatud osa.

3.4.1 Mudelid

CakePHP poolt genereeritud mudelid baseeruvad andmebaasil. Mudelid vahendavad suhtlust veebirakenduse ja andmebaasi vahel. CakePHP raamistik jagab mudelid kaheks: *Table* ja *Entity*. *Table* mudel määrab ära, kuidas ORM (*Object-Relational Mapping*) kiht läheneb tabelile. *Entity* määrab selle, kuidas rakendus peab käituma ühe (või mitme) tabeli reaga.

Rakenduses korrektse andmesisestuse jaoks jõustati mudelites valideerimisreeglid, mis genereeriti andmebaasi põhjal. Näiteks kui andmebaasis on veerule jõustatud kitsendus: `username varchar(40) NOT NULL CHECK (VALUE <> '')`, siis mudelis on järgmised valideerimisreeglid (vt Joonis 16).

```
$validator
    ->scalar('username')
    ->maxLength('username', 40)
    ->requirePresence('username', 'create')
    ->notEmptyString('username');
```

Joonis 16. Valideerimisreegli näide.

Kasutati valideerimisreegleid, mida CakePHP genereeris ning ise uusi ei lisatud. Tuleb ka märkida, et andmebaasis regulaaravaldiste abil jõustatud reeglite kohta (näiteks `CHECK (VALUE !~ '^[[:space:]]+$')`) CakePHP valideerimisreeglit ei koostanud. Töö autorid uusi mudeleid juurde ei lisanud.

3.4.2 Vaated

CakePHP genereerib kõikidele andmebaasi tabelitele vaikimisi neli vaadet: *index*, *view*, *add* ja *edit*.

Index vaade kuvab tabelina nimekirja tabeli kõikidest ridadest. Näiteks on rakenduse avaleht tegelikult tabeli *Viis index* vaade, mis kuvab kõik viisid. Ühel lehel kuvatakse korraga 20 viisi.

Töö autoritel tuli neid vaateid oluliselt muuta. Vaikimisi genereeritud vaates kuvatakse tabeli rea kohta täpselt samad andmed nagu andmebaasis, sh tabeli primaarvõtme väärtus ja kõik välisvõtme väärtused (süsteemis sisemiseks kasutamiseks mõeldud identifikaatorid). Kasutaja jaoks tuli osa andmeid ära jätta ja osa asendada seotud tabelitest võetud arusaadavate andmetega.

View vaade kuvab tabeli ühe rea detailvaate ning sellele tabelile välisvõtme kaudu viitavate seotud tabelite *index* vaated. Näiteks ühe konkreetse viisi kohta kuvatakse selle detailandmed viiside tabelist ning kõik sellele reale välisvõtme kaudu viitavad read (viisi tegijad, kohad, esitused, laulud, muusikalised tunnused, noodistused ning kodeeringud) teistest tabelitest.

View vaateid oli vaja muuta samadel põhjustel kui eelmainitud *index* vaateid. Lisaks ei olnud alati vaja kasutajale kõiki seotud ridu kuvada.

Add vaade on tabelisse uue rea sisestamiseks. Näiteks on ühe uue viisi sisestamise vaade.

Edit vaade on tabeli juba varasemalt sisestatud rea andmete muutmiseks. Näiteks on sisestatud viisi mõne viite parandamiseks või andmete kontrolli märke lisamiseks.

Add ja *edit* vaateid muutsid töö autorid vaikimisi genereeritud vaadetest kõige vähem. Need on mõlemad oma olemuselt sisestusvormid, *edit* vaate korral on vorm lihtsalt eeltäidetud andmebaasist võetud väärtustega. Seepärast käsitleme neid koos.

Väiksemaid muudatusi oli kasutusmugavuse tõstmiseks siiski vaja teha. Näiteks genereeris CakePHP kuupäeva sisestusväljad kolme eraldi väljana: aasta, kuu ja päev. Selle asemel võtsid töö autorid kasutusele Gijgo kuupäeva valimise teegi [16]. Kuupäeva teegi valikukriteeriumiteks olid aktiivne kasutajatugi ja arendus teegi repositooriumis ning juurutamise lihtsus.

Kõikidel vaikimisi genereeritud vaadetel oli vasakul ka vertikaalne navigatsioonimenüü. Selle asendasid töö autorid lihtsustatud horisontaalse menüüga rakenduse üleval ääres ning erinevatele vaadetele lisati vajalike tegevuste läbiviimiseks nupud (vt Joonis 12).

Töö autorid lisasid juurde vaated klassifikaatorite nimekirja ja kasutaja autentimise jaoks.

3.4.3 Kontrollerid

CakePHP genereeris kõikidele tabelitele kontrollerid meetoditega *index*, *view*, *add*, *edit*. Igale vaatele on vaja kontrolleris vastavat meetodit. Töö autorid muutsid kõiki raamistiku poolt genereeritud meetodeid andmete kuvamiste järjestuste, seotud tabelite andmete kuvamise ning sündmuste logi salvestamise jaoks. Juurde oli vaja lisada otsing, autentimise ja autoriseerimise meetodid ning sündmuste logi andmete võrdlemiseks ja salvestamiseks.

3.4.4 Kasutaja autentimine ja autoriseerimine

Autentimata kasutajal on lubatud rakenduses vaadata viise, viisi alamobjekte, isikuid ning klassifikaatoreid. Kui süsteemi kasutaja soovib viia läbi lisamise, muutmise või kustutamise operatsioone, peab ta end autentima. Peale autentimist annab süsteem kasutajale juurdepääsu operatsioonidele, mida ta on autoriseeritud tegema.

Kasutajate autentimiseks kasutasid töö autorid CakePHP raamistiku komponenti *Auth* [4]. Antud komponent kontrollib, kas vastav kasutajakonto eksisteerib ning kas parooli räsiväärtus konto juures on vastavuses kasutaja sisestatud parooliga. Paroolide räsiväärtuste jaoks kasutatakse raamistiku klassi *DefaultPasswordHasher* [15], mis omakorda kasutab PHP-ga kaasasolevat *password_hash* [40] funktsiooni. Viimane kasutab Blowfish krüpteerimisalgoritmile baseeruvat *bcrypt* räsifunktsiooni 512-bitise soolaga.

Kasutajate kontrollerisse lisasid töö autorid juurde järgmised meetodid:

- *login* – kasutaja autentimine *Auth* komponendi abil;
- *logout* – kasutaja sessiooni lõpetamine;
- *password* – kasutaja enda parooli muutmine;
- *beforeFilter* – autentimata kasutajate õiguste haldus;
- *isAuthorized* – autenditud kasutajate autoriseerimise kontroll.

Autentimata kasutajate õigused on defineeritud meetodiga *beforeFilter* (vt Joonis 17).

```
public function beforeFilter(Event $event)
{
    $this->Auth->deny();
    $this->Auth->allow(['index', 'view', 'display']);
}
```

Joonis 17. Kontrolleri *AppController* meetod *beforeFilter*.

Autenditud kasutaja autoriseerimist kontrollitakse rakenduse keskses kontrolleris *AppController*, mille kaudu suunatakse kõikide teiste kontrollerite pöördumised. Kui tegemist on administraatoriga, siis on lubatud kõik toimingud. Kui tegemist on toimetajaga, siis on lubatud vaatamise, lisamise ning muutmise toimingud. Kui tegemist on tavakasutajaga, siis on lubatud ainult vaatamise toiming. Seda meetodit (vt Joonis 18) kasutavad 48 erinevat kontrollerit.

```
public function isAuthorized($user)
{
    $action = $this->request->getParam('action');

    if (isset($user['user_role_type_id'])) {
        $role = $user['user_role_type_id'];

        if ($role == 1) {
            return true;
        } elseif ($role == 2
            && ($action === 'add' OR $action === 'edit')) {
            return true;
        } elseif ($role == 3 && $action === 'view') {
            return true;
        }

    }
    return false;
}
```

Joonis 18. Kontrolleri *AppController* meetod *isAuthorized*.

Eelnevast erinevat õiguste kontrollisüsteemi kasutavad sündmustega, kasutajatega ning viisi seisundite ja kasutajate rollide klassifikaatoritega seotud kontrollerid. Sellisel puhul on kasutusel alternatiivne, tabeli-spetsiifiline *isAuthorized* meetod, mis on ülimuslik võrreldes kontrolleris *AppController* defineerituga.

Tabeli *Kasutaja* andmeid tohib muuta, aga mitte kustutada, vaid administraatori õigustes autenditud kasutaja (vt Joonis 19). Administraatori rolli mitteomavad kasutajad tohivad vaadata vaid enda kasutajakonto infot. Meetodeid *login*, *logout* ja *password* tohivad kasutada kõik.

```
public function isAuthorized($user)
{
    $action = $this->request->getParam('action');

    if (isset($user['user_role_type_id'])) {

        $role = $user['user_role_type_id'];
        if ($action === 'delete') {
            return false;
        } elseif ($role == 1) {
            return true;
        } elseif ($action === 'view') {
            if ($this->request->getParam('pass')[0]
                && ($user['id'] == $this->request->getParam('pass')[0])) {
                return true;
            }
            return false;
        } elseif ($action === 'logout' OR $action === 'password') {
            return true;
        }

    }
    return false;
}
```

Joonis 19. Kasutajate kontrolleri *UsersController* meetod *isAuthorized*.

Andmete terviklikkuse tagamiseks ei tohi saada kasutajaliidese kaudu sündmusi muuta ega kustutada. Erandkorras saab kustutamist teha andmebaasi administraator otse andmebaasis. Seetõttu on sündmustega seotud kontrollerites keelatud kõik meetodid peale *view* (vt Joonis 20).

```
public function isAuthorized($user)
{
    $action = $this->request->getParam('action');

    if ($action === 'view') {
        return true;
    }
    return false;
}
```

Joonis 20. Klassifikaatorite sündmuste kontrolleri *ClassifierEventsContoller* meetod *isAuthorized*.

3.4.5 Sündmused

Nõuetest tulenevalt peab süsteem salvestama kõik andmete lisamise, muutmise ja kustutamise sündmused. Selleks loodi eraldi sündmuste allsüsteem koos teenindatava registriga.

Iga allsüsteemi jaoks realiseeriti eraldi mudel vastava allsüsteemi põhiobjekti ja alamobjektidega seotud sündmuste salvestamiseks. Nendes mudelites võrreldakse andmeid salvestamisel ja enne kustutamist andmebaasis olevate andmetega kasutades CakePHP funktsioone *beforeSave* (vt Joonis 21), *beforeDelete* ja *afterSave* [51].

```
public function beforeSave(Event $event, EntityInterface $entity, ArrayObject
$options)
{
    if (!empty($entity->id)) {
        $this->oldState = $this->get($entity->id)->toArray();
    } else {
        $this->oldState = [];
    }

    parent::beforeSave($event, $entity, $options);
}
```

Joonis 21. Isikute tabeli *ComparePersonEventsTable* meetod *beforeSave*.

3.4.6 Tõlked

CakePHP raamistik toetab automaatselt tõlgete ning lokalisatsioonide lisamist [29]. Rakenduse kasutajaliideses kuvatavad tõlked asuvad eraldi eesti keele tõlkefailis. Rahvaviiside rakenduse kasutajaliides antakse tellijale üle vaikeseadistuses eestikeelsena.

Andmebaasi tasemel rakendatav andmete kontroll tagastab veateated kasutajaliidesesse inglise keeles. Ingliskeelsed veateated püütakse kontrolleris kinni. Kasutajale

arusaadavaks tõlgitakse need samuti raamistikuga kaasas oleva tõlke funktsionaalsuse abil.

Sündmuste logi salvestatakse andmebaasi ingliskeelsena, sest rakendus ja andmebaas on kirjutatud samuti ingliskeelsena (st identifikaatorid on ingliskeelsed). Eesti keele seadete kasutamisel tõlgib rakendus tõlkefaili järgi ka sündmuste logis olevaid väärtuseid. See võimaldab lisada tõlke keeli vaid uusi tõlkefaile luues.

3.5 Testid

Viiside infosüsteemi tarkvara testimise eesmärgiks oli tuvastada süsteemis olevaid vigu ja tagada süsteemi nõuetele vastavus. Töö käigus muudeti ja täiendati nõudeid, eesmärgiga arendada tellija vajadustele vastav süsteem.

Testimise skoopi kuulusid nii funktsionaalsed kui ka mittefunktsionaalsed nõuded (vt alapeatükk 3.1).

Testimist viisid läbi nii teostaja kui ka tellija meeskonnaliikmed. Tellija poolelt osalesid EKM-i töötajad, kes saavad olema süsteemi põhikasutajad.

Testijate ressurss oli ajaliselt piiratud. Tuli leida eesmärgi saavutamiseks kõige efektiivsem lähenemine. Kuna projekti viidi läbi pika perioodi vältel, siis vajadusel võeti arvesse ja oodati ressursi vabanemist.

Testimisel kasutati järgmisi testimismeetodeid [45].

- Valge kasti testimine (*White Box Testing*) – üksikasjalik testimine, mille raames vaadati süsteemi sisemisi struktuure ja loogikat, vajadusel analüüsiti lähtekoodi.
- Musta kasti testimine (*Black Box Testing*) – testiti ilma lähtekoodi ja süsteemi sisemisi struktuure vaatamata. Eelduseks olid teadmised süsteemi käitumuslikust toimimisest.
- Halli kasti testimine (*Gray Box Testing*) – kombinatsioon valge kasti ja musta kasti testimismeetoditest.
- *Ad hoc* testimine (*Ad hoc Testing*) – testimisel improviseeriti ja tehti süsteemis tegevusi juhuslikkuse alusel.

Rakenduse ja andmebaasi valideerimiseks viidi läbi järgmist tüüpi käsitsi testimised:

- andmebaasi testimine (teostaja ja tellija);
- funktsionaalne testimine kasutajaliidese kaudu, mille käigus valideeriti ka kasutatavust (teostaja ja tellija);
- mittefunktsionaalne testimine (teostaja).

Automaatteste rakenduses ei realiseeritud. Antud teema on lisatud tuleviku arendusvaatesse.

Andmebaasi testimise käigus jälgiti andmebaasi loomise ja näiteandmete sisestamise skripti tulemit. Docker konteinerite käivitamisel luuakse andmebaasi konteineri esimesel käivitamisel andmebaas nullist koos andmemudeli ja näiteandmetega. Andmebaasi muudatuste jõustamiseks kustutati ja taasloodi andmebaas alati nullist. Vigaste andmete sisestamist selle käigus ei proovitud, kuid andmete sisestamisel valideeriti andmebaasi kitsendusi.

Funktsionaalsete testide käigus testiti kõiki kasutusjuhte, sh rollidele vastavaid õigusi, ning andmetele seatud nõudeid. Testiti nii positiivseid kui ka alternatiivseid (sh veaolukordadega) stsenaariumeid. Testimine toimus kasutajaliidese kaudu. Samade testide raames valideeriti ka süsteemi kasutatavust. Suur osa testimist tehti musta kasti meetodil, kuid vastavalt vajadusele rakendati ka valge kasti ja halli kasti meetodit.

Protsessi käigus kirjeldati testjuhtudena kasutusjuhtude positiivsed stsenaariumid ning ainult olulisemad alternatiivsed stsenaariumid. Sündmuste lisamine ja vaatamine valideeriti teiste kasutusjuhtude raames, nt viisi sisestamisel kontrolliti ka vastava sündmuse lisamist. Suur osa testjuhte tehti *ad hoc* meetodil, proovides läbi erinevaid tegevusi ning erinevate andmete ja kombinatsioonide sisestamist. Enamik testjuhte läbiti korduvalt. Joonis 22 – Joonis 23 on toodud testjuhtude kirjeldamise näited.

	Kasutusjuht		Testjuhu nimetus	Tegutseja roll		Tegevuse kirjeldus	Oodatav tulemus
1	**Sisesta viis**						
		1.1	Korrektse viisi sisestamine	Administraator	1	+Viis. Sisestab korrektsed andmed. Salvestab.	Viis salvestatakse ja kuvatakse vastav teade. Saab vaadata lisamise sündmust. Viisi juurest saab sisestada alamobjektid (viisi tegija, koht, esitus, laul, muusikalised tunnused, noodistus, noodistuse tegija, kodeering).
					2	+Viisi tegija Sisestab korrektsed andmed. Salvestab.	Viisi tegija salvestatakse ja kuvatakse vastav teade. Viisi andmete juures kuvatakse sisestatud viisi tegija. Saab vaadata lisamise sündmust. Saab lisada veel viisi tegijaid.
					3	+Koht Sisestab korrektsed andmed. Salvestab.	Koht salvestatakse ja kuvatakse vastav teade. Viisi andmete juures kuvatakse sisestatud koht. Saab vaadata lisamise sündmust. Saab lisada veel kohti.
					4	+Esitus Sisestab korrektsed andmed. Salvestab.	Esitus salvestatakse ja kuvatakse vastav teade. Viisi andmete juures kuvatakse sisestatud esitus. Saab vaadata lisamise sündmust. Rohkem esitusi ei saa lisada.
					5	+Laul Sisestab korrektsed andmed. Salvestab.	Laul salvestatakse ja kuvatakse vastav teade. Viisi andmete juures kuvatakse sisestatud laul. Saab vaadata lisamise sündmust. Rohkem laule ei saa lisada.
					6	+Muusikalised tunnused Sisestab korrektsed andmed. Salvestab.	Muusikalised tunnused salvestatakse ja kuvatakse vastav teade. Viisi andmete juures kuvatakse sisestatud muusikalised tunnused. Saab vaadata lisamise sündmust. Rohkem muusikalisi tunnuseid ei saa lisada.
					7	+Noodistus Sisestab korrektsed andmed. Salvestab.	Noodistus salvestatakse ja kuvatakse vastav teade. Noodistusele saab lisada Noodistuse tegija. Viisi andmete juures kuvatakse sisestatud noodistus. Saab vaadata lisamise sündmust. Saab lisada veel noodistusi.
					8	+Noodistuse tegija Sisestab korrektsed andmed. Salvestab.	Noodistuse tegija salvestatakse ja kuvatakse vastav teade. Noodistuse andmete juures kuvatakse sisestatud noodistuse tegija. Saab vaadata lisamise sündmust. Saab lisada veel samale noodistusele noodistuse tegijaid.
					9	+Kodeering Sisestab korrektsed andmed. Salvestab.	Kodeering salvestatakse ja kuvatakse vastav teade. Viisi andmete juures kuvatakse sisestatud kodeering. Saab vaadata lisamise sündmust. Saab lisada veel kodeeringuid.

Joonis 22. Viisi sisestamise positiivse stsenaariumi testjuht.

	Kasutusjuht		Testjuhu nimetus	Tegutseja roll		Tegevuse kirjeldus	Oodatav tulemus
1	**Sisesta viis**						
		1.2	Sama viis on olemas (ei ole unikaalne)	Administraator	1	+Viis. Sisestab andmed (sama viisiviite+tekstiviite+heliviite+videoviite kombinatsioon peab enne olemas olema). Salvestab.	Kuvatakse veateade, viisi ei salvestata. Peale andmete muutmist saab salvestada.
6	**Lisa isik**						
		6.2	Sama PIDiga isik on olemas (ei ole unikaalne)	Administraator	1	+Isik. Sisestab andmed (sama PID peab enne olemas olema). Salvestab.	Kuvatakse veateade, isikut ei salvestata. Peale andmete muutmist saab salvestada.
16	**Lisa klassifikaatori väärtus**						
		16.2	Sama klassifikaatori väärtus on olemas (ei ole unikaalne)	Administraator	1	+Klassifikaatori väärtus. Sisestab andmed (sama nimi peab enne olemas olema). Salvestab.	Kuvatakse veateade, klassifikaatori väärtust ei salvestata. Peale andmete muutmist saab salvestada.

Joonis 23. Unikaalsuse valideerimise testjuhud.

Testimisel ilmnenud vigade parandamisel ja muudatusettepanekute realiseerimisel oli esmaseks prioriteediks funktsionaalsus. Nii kasutatavus kui ka kasutajaliidese kujundus ja stiilid olid teisejärgulised.

Mittefunktsionaalsel testimisel kontrolliti kasutajaliidesele seatud nõuete täidetust. Veenduti, et kasutajaliidese kuvad on eesti keeles ning tarkvara saab kasutada veebilehitsejates Google Chrome, Firefox, Safari ja Brave ning nutiseadmetel Android ja iOS. Lisaks kontrolliti äriprotsessidega seotud nõuded (nt kasutajaliides peab alati küsima kinnitust enne andmete kustutamist).

Rakenduse koodi kirjutamisel tegi arendaja esmase valideerimise lisatud või muudetud koodi osas. Lisaks tegi teine arendaja koodi läbivaatuse ja lisatud funktsionaalsuse valideerimise kasutades valge kasti meetodit. Tuvastatud vead parandati enne edasist testimist.

Peale rakenduse koodi või andmebaasi muudatusi viidi teostaja poolt läbi osalisi regressiooniteste kogu valmisoleva tarkvara ulatuses. See tähendas kasutajaliidese kaudu funktsionaalset testimist.

Teostaja meeskonnaliikmed testisid arendusi enda lokaalsetes masinates. Tellija juures oli üles seatud testkeskkond. Rakenduste muudatuste korral teavitati tellija IT-töörühma töötajat, kellel oli ligipääs viiside projekti koodirepositooriumisse. Tema poolt uuendati testkeskkonnas rakenduse kood ja taaskäivitati see uute muudatustega. Peale igat muudatust tegi esmase valideerimise süsteemi toimimise osas alati teostaja. Peale seda teavitati tellija testijaid, sh anti ülevaade valminud funktsionaalsustest ja parandatud vigadest.

3.6 Tulemi üleandmine

Projekti tulemina antakse EKM-ile üle järgnevad tehised.

- Toimiv ja kasutusele võtmiseks valmis tarkvara koos lähtekoodiga. Juurutamine sõltub EKM-ist.
- Projekti dokumentatsioon, mis kirjeldab loodud infosüsteemi tarkvara, sh realiseeritud funktsionaalsed ja mittefunktsionaalsed nõuded, kasutusjuhud, funktsionaalsed allsüsteemid, kasutusõiguste kirjeldus, kontseptuaalne

andmemudel, andmebaasi füüsilise disaini mudel, süsteemi arhitektuur, andmebaasi ja kasutajaliidese disain ning koodi põhimõtted.

- Tarkvara kasutusjuhend.

Tarkvara ja keskkonnad on kirjeldatud koodina, mis on kättesaadavad töö autorite repositooriumis. Üleandmiseks kloonib EKM-i IT-töörühma töötaja repositooriumis oleva rahvaviiside projekti koodi EKM-i sisekasutuseks olevasse repositooriumisse. Töö autorite repositooriumis suletakse rahvaviiside projekt. Tarkvara juurutusjuhend asub repositooriumi juurkataloogis failis README.md

Tulemi üleandmine toimub hiljemalt 30. juunil 2020.

3.7 Arendusvaade

Käesoleva projekti realiseerimine oli aluseks viiside infosüsteemi tekkimisele. Infosüsteemi oluliste eesmärkide (vt alapeatükk 1.2) saavutamiseks peab seda aga edasi arendama.

Järgnevalt on toodud nimekiri tellija poolt esitatud funktsionaalsustest, mida käesoleva projekti käigus ei realiseeritud, kuid mis on vaja süsteemi eesmärkide saavutamiseks ja paremaks kasutamiseks realiseerida.

- **Andmete migratsioon**
  - MS Access andmebaasis ja Google Sheetsis olevate andmete korrastamine ja migreerimine viiside infosüsteemi.

- **Viiside haldamine**
  - Viiside liitotsing erinevate tunnuste järgi.
  - Uue viisi lisamine olemasoleva viisi andmete alusel.
  - Viisi andmete muutmine mitme viisi juures korraga.
  - Viisi tekstile versioonide lisamine, kui sama tekstiviitega tekst on üles kirjutatud mitu korda ja nendes on erinevused.
  - Mallide loomine viisi kodeeringute jaoks.
  - Kasutajaliidese kasutajakogemuse analüüsimine ja parendamine.
  - Automaattestide realiseerimine.

- **Failide hoidmine**
  - Noodifailide hoidmine.
  - Helifailide ettemängimine.

- **Liidestus seotud süsteemidega**
  - Tekstide kuvamine Eesti regilaulude andmebaasist.
  - Failide küsimine failirepositooriumist Kivike.
  - Andmete vahetamine Kivikese juures oleva isikute mooduliga.

- **Andmete kättesaadavaks tegemine**
  - Andmete ja failide allalaadimine.
  - Kasutajaliidesesse teiste keelte lisamine, sh klassifikaatorite väärtuste tõlkimine vastavatesse keeltesse.

Kõiki funktsionaalsuseid tuleb enne realiseerimist koos tellijaga analüüsida, et tuvastada realiseerimise täpne vajadus ja sisu. Töö autorite hinnangul on järgmisteks prioriteetsemateks töödeks andmete migratsioon ja liidestused seotud süsteemidega.

4 Analüüs ja järeldused

Järgnevalt kirjeldatakse eelmises peatükis esitatud töö tulemuste valikute põhjendused ja analüüs ning antakse hinnang projekti tegemisele.

4.1 Töö tulemuste põhjendus

Töö tulemuste põhjenduste all on analüüsitud ja põhjendatud valikuid, mida töö autorid tegid nõuete, tehnilise arhitektuuri, andmebaasi disaini ja testimise juures.

Kasutajaliidese disain ning kood põhinesid peamiselt raamistikul ning nende osas töö autorid olulisi valikuid tegema ei pidanud ning lähtuti sellest, mida raamistik valmis genereeris. Koodi täiendamisel võeti aluseks CakePHP koodi kirjutamise standardid [17].

4.1.1 Nõuded

Süsteemi analüüsimisel kasutasid töö autorid andmekeskset lähenemist järgmistel põhjustel.

- Tellija kasutas viisist ja viisi andmetest rääkimisel keerulisi ja spetsiifilisi termineid, mille sisu ei olnud teostajale arusaadav.
- Polnud selge, mida mainitud andmetest oleks parem väljendada kontseptuaalses andmemudelis olemitüüpidena ja mis nende atribuutidena ning millised on nende omavahelised seosed.
- Andmete hulk oli suur. Andmete kirjeldamine ja andmemudeli loomine olid esmahinnangul kõige töömahukam osa projektist.

Analüüsi käigus kirjeldati olemitüüpide ja atribuutide nimetused, definitsioonid, andmetüübid (nt tekst, number, valikväärtus), pikkused ja esinemise sagedus (kohustuslikkus, ühe või mitme väärtuse lubamine).

Andmekeskse analüüsi olulisemad väärtused olid järgmised.

- Võimaldas grupeerida viisi andmed loogilisse struktuuri ja aru saada andmete omavahelistest seostest.
- Tagas ühtse ja arusaadava viise puudutava terminoloogia. Olemitüüpide ja atribuutide kokkulepitud sõnalised selgitused, mis sisuliselt moodustavad süsteemi valdkonna mõistete sõnastiku (Lisa 5 – Lisa 9), aitasid teostajatel valdkonnast paremini aru saada.

Analüüsi käigus tuvastati, et andmed jagunevad nelja põhiobjekti vahel: viis koos alamobjektidega (koht, tegija, esitus, laul, muusikalised tunnused, noodistus, kodeering), klassifikaator, isik ja kasutaja.

Paljud paindmetoodikad näevad ette funktsionaalsuse kirja panemise kasutuslugude abil [18], millega antakse edasi ainult süsteemi kasutaja vaatenurk. Töö autorid otsustasid analüüsi käigus kirjeldada aga kasutusjuhud (koos tegutsejatega), kuna lisaks oli vaja saada ülevaade ka süsteemi käitumisest kasutaja tegevuste korral.

Kasutusjuhud andsid arusaamise funktsionaalsustest, mida tarkvara pidi toetama. Selle eelised olid järgmised.

- Andis lihtsa ja selge arusaama projekti skoobist, mis oli arusaadav ka ilma tehniliste teadmisteta.
- Lihtsustas arendamist ja otsuste tegemist, kuna kõik meeskonnaliikmed said aru tegutseja vaatest.
- Võimaldas paremini prioritiseerida tööde järjekorda, nt kõigepealt realiseerida viisi sisestamise funktsionaalsus ja peale seda kasutaja lisamise oma. Selline lähenemine oli vajalik ka selleks, et anda tarkvara järk-järgult tellijale testimiseks.
- Kasutusjuhtude alusel lähenemist sai kasutada ka testimisel.

Kuna nii andmed kui kasutusjuhud jaotusid nelja põhiobjekti vahel, siis otsustati töö paremaks organiseerimiseks jagada viiside infosüsteem põhiobjektide alusel funktsionaalseteks allsüsteemideks. Viiside, klassifikaatorite, isikute ja kasutajate logi kasutamise kirjeldamiseks defineeriti lisaks sündmuste funktsionaalne allsüsteem, mis teenindab sündmuste registrit, kus vastavat logi hoitakse. Selline lähenemine võimaldas

tükeldada arenduse väiksemateks osadeks ning realiseerida süsteem järk-järgult funktsionaalsete allsüsteemide kaupa. Näiteks kõigepealt arendati viiside allsüsteemi funktsionaalsus ja seejärel klassifikaatorite allsüsteemi oma. See lihtsustas arendusprotsessi ning aitas hoida funktsionaalsuste iteratiivse valmimise tähelepanu keskmes.

4.1.2 Tehniline arhitektuur

Infosüsteemi disainimisel lähtusid töö autorid sellest, et sarnaste nõuetega ülesandeid lahendatakse tavaliselt kasutades andmebaasi ning baasis olevat infot vahendavat veebirakendust.

Andmebaasi ning veebirakenduse kombinatsioon valiti ka seetõttu, et EKM-is on sarnane arhitektuur juba kasutusel, nt Eesti Regilaulude andmebaas [24] ja failirepositoorium Kivike [23]. EKM-i eesmärgiks on, et nende andmekogud oleksid avalikult kättesaadavad ning see laieneb ka viiside infosüsteemile.

Avaliku teabe seaduse [5] kohaselt on keelatud asutada ühtede ja samade andmete kogumiseks eraldi andmekogusid. Samuti soovitab Euroopa Komisjoni *The Once-Only Principle Project* kasutada avalikes andmekogudes *Once-Only* printsiipi [36], kus kasutaja peab enda valduses olevat infot süsteemiga vaid üks kord jagama ning süsteem peab enda inforessurssidele võimaldama ühiskasutuse. Seda arvesse võttes on andmebaasi kaudu ülesandele lähenemine ainuõige otsus.

Serveri andmebaasi ja selle kasutamiseks mõeldud rakenduste abil on võimalik teha andmed kättesaadavaks paljudele inimestele korraga. Sisestama peab sellisesse andmebaasi andmeid vaid korra, et need kõikidele kättesaadavaks (ühiskasutatavaks) muutuks.

Veebirakenduse ehitamise eeliseks platvormipõhisele (kasutaja arvutisse installeeritud) rakendusele on see, et veebilehitsejad on juba kõikidel platvormidel olemas. See tähendab, et arendajal ei ole vaja kõikidele kasutajate platvormidele eraldiseisvat klient-rakendust ehitada. Samuti vähendab see hooldustööde mahtu rakenduse elutsükli vältel.

Töö autorid kasutasid arenduskeskkonnana erinevaid operatsioonisüsteeme. Kuna tellija poolt puudus algselt info, mis platvormil uus tarkvara tööle pannakse, otsustasid töö autorid nii arenduse kui ka juurutuse lihtsustamiseks kasutada Docker konteineritel

põhinevat lähenemist. Töö autorid kasutasid Docker platvormi tööriista Compose, millel on mitmeid eeliseid varasemate, virtualiseerimise põhiste lähenemistega [6], [52].

- Kogu rakenduse platvorm on võimalik kirjeldada koodina (vt Joonis 24). See võimaldab platvormi versioneerida ning salvestada repositooriumisse.
- Platvormi seadete halduse lihtsus – kogu platvormi parameetrid on äärmiselt lihtsalt muudetavad.
- Keskkonna ühilduvus operatsioonisüsteemiga – ei ole tähtis, kas Docker Server on tööle pandud Linux, Unix, MacOS või Windows platvormil. Kui see käivitus ühel, käivitub see ka teisel.
- Iga süsteemi komponent käivitatakse Compose abil eraldi konteineris. Iga moodulit on võimalik eraldi uuendada või seadistada. Iga mooduli käitumist saab eraldi analüüsida ning võimalikke vigu leida ja parandada. Kui kõik moodulid on üksteisest isoleeritud, siis juhtub süsteemis vähem võimalikke konflikte.
- Docker loob konteinerite vahele automaatselt võrgud. Juurutatava rakenduse haldur ei pea sellega eraldi tegelema. Lisaks on väga täpselt võimalik kontrollida, mis informatsioon moodulite vahel liigub.

```
version: '3'
services:
  web:
    image: nginx
    volumes:
      -./.docker/conf/nginx/default.conf:/etc/nginx/conf.d/default.conf
      - ./viisid:/var/www/html
    ports:
      - 80:80
    restart: always
    depends_on:
      - php
      - db
  php:
    build: .docker
    restart: always
    volumes:
      - ./.docker/conf/php/php.ini:/usr/local/etc/php/conf.d/php.ini
      - ./.docker/conf/php/xdebug.ini:/usr/local/etc/php/conf.d/xdebug.ini
      - ./viisid:/var/www/html
    environment:
      URL: "${URL:-http://localhost}"
  composer:
    image: composer
    volumes:
      - ./viisid:/app
    command: install
  db:
    image: postgres:10.12
    restart: always
    environment:
      - POSTGRES_DB=viisid
      - POSTGRES_USER=local_dev_username
      - POSTGRES_PASSWORD=local_dev_password
    ports:
      - 5555:5432
    volumes:
      - ./.docker/conf/postgres/:/docker-entrypoint-initdb.d/
```

Joonis 24. Rakenduse platvormi kirjeldus koodina.

PHP serveris kasutasid töö autorid CakePHP raamistikku. Raamistik valiti seetõttu, et töö autorid soovisid oma töös keskenduda rohkem analüüsile ja andmestruktuuridele kui rakenduse programmeerimisele. CakePHP raamistik võimaldab genereerida andmebaasi alusel baasrakenduse. Seda võimalust töö autorid ka kasutasid. Valik kasutada CakePHP raamistikku langetati otsingutulemuste ning ekspertarvamuste kogumise järgselt. Lisaks kaaluti Slim [44], Lumen [33] ja Yii [58] raamistikke.

CakePHP raamistiku poolt genereeritud kood kasutab MVC [34] disainimustrit. MVC võimaldab arendada erinevad rakenduse kihid erinevate arendajate poolt. See vähendab võimalusi koodi konfliktideks olukorras, kui erinevate arendajate kood repositooriumis liidetakse. Samuti on võimalik vaadete kihti (kasutajaliidest) ülejäänud rakendusest eraldiseisvalt arendada. Seda saab teha näiteks meeskonnaliige, kes ei pea oskama keerukamat arendust, vaid on keskendunud kasutajaliidese tehnoloogiatele.

Töö autorite hinnangul oli käesolevas projektis tarkvara arendamiseks CakePHP raamistiku kasutamine sobiv valik. Lähtuvalt eelmainitust vähendas ja lihtsustas raamistik oluliselt tarkvara funktsionaalsuste programmeerimist. Lisaks aitas see kaasa osade mittefunktsionaalsete nõuete täitmisele, nt rakenduse äriloogika realiseerimine andmebaasist eraldi, süsteemi kaitsmine SQL-süstide ja skriptisüstide vastu, kasutajaliideses kuvatavate tabelite veergude sorteerimine, parooli ja soola põhjal leitud räsiväärtuse loomine.

EKM kasutab asutuse veebirakenduste jaoks ühtset PostgreSQL serverit – seega olid andmebaasi aluseks olev andmemudel (SQL andmemudel) ja kasutatav andmebaasisüsteem (PostgreSQL) teostajatele ette kirjutatud. Kuna PostgreSQL on tasuta, avatud lähtekoodiga, võimalusterohke, populaarne ja järgib hästi SQL standardit ei heida teostajad sellele valikule midagi ette. Juurutamise käigus muudetakse rakenduse konfiguratsiooni nii, et eraldi Docker PostgreSQL konteineri asemel võtab rakenduse konteiner edaspidi ühendust EKM-i keskse andmebaasiserveriga (vt Joonis 8). Sellisel juhul saab rakendada viiside andmebaasile samu haldus- ja varundusreegleid, mis rakenduvad EKM-i teistele veebirakenduste andmebaasidele.

4.1.3 Andmebaasi disain

Andmebaasiobjektide nimed loodi ingliskeelsena tulenevalt CakePHP raamistiku vaikimisi seadetest [13].

Andmebaasiobjektide nimetamisel järgisid töö autorid CakePHP raamistiku poolt andmebaasile esitatavaid nõudeid [12], [50]. Seal, kus nõudeid ei esitatud, võtsid töö autorid aluseks mõned olemasolevad nimede andmise mustrid [3] ja kohandasid need endale sobivaks. Nimede andmisel püüti olla järjekindlad.

CakePHP andmebaasiobjektide nimetamise põhimõtted.

- Tabelite nimed on mitmuses ja nime komponendid on eraldatud alakriipsuga (_): `tunes`, `tune_places`.
- Veergude nimed on kahe ja rohkema sõna korral eraldatud alakriipsuga (_): `tune_reference`, `melostrophe_num_audio`.
- Igal tabelil on primaarvõtme veerg nimega `id`. Raamistiku nõuded olid põhjuseks miks antud juhul kasutati sellist primaarvõtit (sh sellise nimega), mille kasutamise probleemidele viitab SQL antimuster „Üks suurus kõigile“ (*One Size Fits All*) [32].
- Välisvõtme veeru nime alguses on seotud tabeli nimi ainsuses, millele järgneb `_id`: `tune_id`, `tune_performance_id`.
- Mitu-mitmele seose tulemusena loodud vahetabelite nimed sisaldavad seotavate tabelite nimesid: `tune_songs_song_genres`, `tune_songs_song_topics`.

Tabel 3 esitab töö autorite poolt kasutamiseks valitud andmebaasiobjektide nimede mustrid.

Tabel 3. Andmebaasiobjektide nimede mustrid.

Andmebaasiobjekti tüüp	**Nime muster**	**Näide**
Primaarvõtme kitsendus	`PK_{table}`	`PK_tune_performances`
Välisvõtme kitsendus	`FK_{child_table}_{primary_table}`	`FK_tune_performances_tunes`
Unikaalsuse kitsendus	`UQ_{table}_{column}`	`UQ_tune_performances_tune_id`
Domeen	`D_{name}`	`D_title`
CHECK kitsendus (domeenis)	`CK_{domain_name}_{description}`	`CK_title_not_empty_string`
CHECK kitsendus (tabelis)	`CK_{table_name}_{description}`	`CK_tune_performances_modified_no_earlier_than_created`
Indeks (välisvõtme veerule)	`IX_{child_table}_{primary_table}`	`IX_tune_performances_actual_action_types`
Unikaalsuse indeks	`IX_{table}_{column}`	`IX_actual_action_types_title`
Trigeri funktsioon	`F_{name}`	`F_change_tune_state()`

Andmebaasiobjekti tüüp	Nime muster	Näide
Triger	`TR_{table}_{function_name}`	`TR_tunes_ change_tune_state`

PostgreSQLis saab andmebaasiobjekti nime pikkus olla vaikimisi seadete korral maksimaalselt 63 baiti. Seda nõuet ei saanud eelpool toodud mustrite korral alati täita. Sellises olukorras lühendati mitmesõnalise tabeli nime nii, et jäeti alles iga sõna esimene täht. Näiteks tabeli `tune_songs_song_topics` välisvõtme kitsenduse nimi on `FK_tsst_tune_songs.`

Domeenide kasutamisel baastabelite veergude kirjeldamiseks lähtuti üldjuhul põhimõttest, et domeen on taaskasutatav tehis ja selle loomise pingutus tasub ära, kui domeene erinevate veergude puhul taaskasutada. Seega kaaluti domeeni loomist juhul, kui seda sai kasutada vähemalt kahe veeru puhul ning sellega oli seotud CHECK kitsendusi. Paaril korral on loodud domeen, mida kasutatakse vaid ühe veeru puhul, et lihtsustada hilisemat kitsenduste lisamist ja luua tuleviku jaoks võimalus selle domeeniga määratud omaduste taaskasutamiseks.

Klassifikaatorite realiseerimisel tuli arvestada nii lineaarsete kui ka hierarhiliste klassifikaatoritega.

Lineaarsete klassifikaatorite võimaliku realiseerimise lahenduse valikul võtsid töö autorid aluseks Aleksandra-Salome Vellemaa poolt Tallinna Tehnikaülikoolis kaitstud bakalaureusetöö „Mõned disainimustrid klassifikaatorite esitamiseks SQL andmebaasides“ [56]. Antud bakalaureusetöös võrdles Vellemaa omavahel kolme disainimustrit: „Iga klassifikaator eraldi tabelis“, „Kõik klassifikaatorid ühises tabelis“ ja „Kunstlik ühendaja“ (iga klassifikaator eraldi tabelis, millele luuakse lisaks kunstlik klassifikaatorite tabel metaandmete hoidmiseks). Käesoleva töö autorid valisid oma projektis sobivaimaks lahenduseks disainimustri „Iga klassifikaator eraldi tabelis“. Valiku peamiste põhjustena võib välja tuua päringute tegemise lihtsuse, suhteliselt väikese ridade arvu tabelites ning võimaluse kasutada klassifikaatori identifikaatoreid otse päringutes ja välisvõtmete veergudel vaikimisi väärtustena. Päringutes kasutatud klassifikaatorid lisati andmebaasi kindlas järjekorras andmebaasi esmasel loomisel.

Hierarhiliste klassifikaatorite realiseerimisel lähtuti CakePHP raamistiku reeglitest [53]. Reeglid nägid ette hierarhilisi klassifikaatoreid sisaldavatele tabelitele järgmiste veergude lisamise:

- *parent_id* – viitab vanemale,
- *lft* – puustruktuuri hoidmiseks,
- *rght* – puustruktuuri hoidmiseks.

Indeksite loomisel lähtuti põhimõttest, et tegemist on andmebaasiga, kus toimub palju (väikesemahulisi) andmemuudatusi, seega nende jõudluse huvides ei saa indekseid olla liiga palju (sest andmete muudatused nõuavad süsteemilt ka indeksite muutmist).

Kitsenduste kontroll (primaarvõtme kitsendused, unikaalsuse kitsendused, NOT NULL kitsendused, CHECK kitsendused, triger) realiseeriti andmebaasi tasemel, et tagada andmebaasis olevate andmete reeglitele vastavus ning saavutada päringute ja andmemuudatuste kiirem täitmine. Andmebaasis realiseeritud kitsenduste alusel genereeris CakePHP valideerimisreegleid ka rakenduse mudelitesse, mis võimaldab teha andmete kontrolli juba rakenduse tasemel ning mitte saata ebakorrektseid andmeid andmebaasi. Sellisel viisil kitsenduste kontrolli loomine vastab „Ära korda ennast“ (*Don't Repeat Yourself*) disainimustrile [20], mis soovitab, kuidas tarkvaras dubleerimist vältida. Andmetele kehtivad kitsendused on keskselt ja autoriteetselt ära kirjeldatud andmebaasis ning rakendustes kasutusmugavuse huvides tehtava kontrolli kood on tekitatud selle põhjal automaatselt – koodi genereerides. Probleemiks on, et kitsenduste lisamisel, muutumisel või eemaldamisel tuleb rakenduse koodi muuta käsitsi, kuna algselt automaatselt genereeritud rakenduse koodi on juba liiga palju käsitsi muudetud, et saaks lubada selle asendamist uue genereeritud koodiga. Samuti on probleemiks, et arendusvahend (CakePHP) ei suutnud regulaaravaldisi sisaldavate CHECK kitsenduste põhjal koodi genereerida.

Andmebaasis ei loodud andmete lugemise toetamiseks vaateid ega andmemuudatuste tegemiseks funktsioone/protseduure, sest töö autorid kasutasid selleks CakePHP raamistiku sisseehitatud ORM lahendust. Raamistiku enda ORM lahenduse ning sisseehitatud funktsioonide kasutamine tõstis rakenduse turvalisust. Raamistik pakub kaitset SQL-süstide ja skriptisüstide vastu, kui kasutada raamistiku sisemisi funktsionaalsusi. Kasutatud lahenduse probleemina tuleb esile tõsta, et iga tabelite

struktuuris tehtav muudatus tingib vajaduse muuta ka rakenduse koodi, sest seda muudatust ei saa peita vaadete ja funktsioonide/protseduuride liidese taha.

4.1.4 Testid

Testimismeetoditest kasutati agiilset testimist, kuna see tulenes iteratiivset mudelit järgiva arendusmetoodika valikust. Töö autorite arvates olid antud lähenemise eelised järgmised.

- Võimaldas keskenduda töötavale ja ootustele vastava tarkvara loomisele.
- Võimaldas teha jooksvalt muudatusi ja otsuseid, kui tellija vajadused seda nõudsid.
- Võimaldas testida järk-järgult vastavalt arenduste valmimisele ja saada ka tellijalt kiiremini tagasisidet loodava tarkvara sobivuse osas.

Antud lähenemise juures oli väljakutseks lõplike muudatuste piiritlemine. Kuna eeldati et muudatuste tegemine oli pidevalt võimalik, siis projekti valmimiseks oli vajalik ühel hetkel tõmmata piir realiseeritava ja mitterealiseeritava vahele. Näiteks otsustati töö käigus jätta tuleviku arenduseks edasine kasutajaliidese kasutajakogemuse analüüs ja parendamine.

Käesoleva projekti algusfaasis otsustati automaatteste mitte realiseerida ning testida käsitsi kasutajaliidese kaudu järgmistel põhjustel.

- Leiti, et efektiivne oli kasutada uurimuslikku katsetamise (*Exploratory Testing*) [25] lähenemist, kuna valdkond oli teostajale võõras. Testjuhte ei loodud ette, vaid valideerimise käigus kontrolliti süsteemi erinevaid funktsionaalsusi ja võimalusi. Jooksvalt kirjeldati vajadustele vastavad testjuhud.
- Võimaldas keskenduda kasutaja töövoole ja selle parandamisele aktsepteeritava taseme saavutamiseks. Kuna andmevälju oli palju, siis tuli töö käigus leida lahendus nende mõistlikul viisil esitamiseks. Samuti oli oluline realiseerida kõik kasutajale kuvatavad tekstid ja info arusaadavalt.
- Käsitsi testimine oli vaja mingis ulatuses igal juhul läbi viia. Täiendav automaattestide kirjutamine oleks tähendanud hinnanguliselt liiga suurt täiendavat ajakulu. Töö autorid ei tegele igapäevaselt automaattestide kirjutamisega, mistõttu oleks pidanud kulutama aega ka õppimisele.

- Käesoleva projekti käigus valmis viiside infosüsteemi esimene etapp, mille eesmärgiks oli luua kasutamiseks valmis tarkvara väga piiratud hulgale kasutajatele. Automaattestid ei olnud eesmärgi täitmiseks kohustuslikud.
- Rakendus on äriloogika vaates lihtsa ülesehitusega ja keerulisi ärireegleid ei ole. Seetõttu tundus lihtne kasutajaliidese kaudu vajalikud testjuhud üle käia.
- Puudus laialdane kogemus automaattestimise rakendamisest tööalastes projektides.

Projekti lõppedes leiavad töö autorid, et antud tarkvara arendamise puhul oleks siiski pidanud rakendama automaattestimist järgmistel põhjustel.

- Regressiooniteste tuli teha palju kordi, kuna nõudeid otsustati korduvalt muuta (sh tehti muudatusi ka andmebaasi osas). See testimine oli töömahukas ning üldjuhul jõuti valideerida ainult olulisemad testjuhud.
- Andmeväljade hulk oli suur, aga äriloogika lihtne. Testide koodi kirjutamine ei oleks olnud ilmselt väga keeruline.
- Realiseeritud automaatteste saaks kasutada ka järgmiste arenduste puhul, mis hoiaks tulevikus aega kokku.

Teostaja poolel ei olnud testkeskkonna ülesseadmine vajalik, sest kõik meeskonnaliikmed oskasid ise enda masinas koodi uuendada. Kuna tellija testijad ei oma vastavaid teadmisi ja tööriistu, siis neile oli vajalik eraldi testkeskkond, mis seati üles tellija juures.

Testjuhud kirjeldati jooksvalt iteratsioonide käigus. Kuna juba esimese iteratsiooni jooksul olid töö autorid omandanud suhteliselt kõrge teadlikkuse nii rahvaviisi sisust kui ka süsteemi eesmärkidest, siis otsustati kõiki testjuhte mitte kirja panna. See võimaldas panustada rohkem aega reaalsesse testimisse, et tagada võimalikult kõrge testidega kaetus.

Arendatud ja testimisse jõudnud tarkvara kvaliteeti hindavad töö autorid heaks. Kuna alati rakendati koodi ülevaatamist teise, teemaga detailselt kursis oleva, arendaja poolt, siis avastati palju vigu juba arendusetapis. Seetõttu hindavad töö autorid funktsionaalsete testide käigus raporteeritud vigade arvu pigem väiksemaks nende keskmisest tööalasest praktikast.

Tellija testimise tulemusena saadi pigem kinnitust realiseeritud lahenduse sobivuse osas ning nende poolt edastati parandamiseks üksikud vead.

Töö autorite hinnangul on testidega kaetus 90% tarkvara funktsionaalsusest.

Töö autorid plaanivad peale käesoleva töö esitamist parandada kõik olulised vead, et süsteemi oleks võimalik kokkulepitud funktsionaalsuses kasutada. Enne projekti tulemi üleandmist tellijale viiakse läbi ulatuslik vastuvõtu testimine kogu funktsionaalsuse osas, veendumaks, et tarkvara vastab kokkulepitud nõuetele. Tarkvara kvaliteedi lõplikuks kinnituseks on tellija valmisolek projekti tulemi vastuvõtmiseks.

4.2 Tehtud tööde detailne kirjeldus

Tehtud tööd jagasid töö autorid järgmisteks kategooriateks:

- analüüs – eeltöö, nõuete kogumine ja muutumine, tehnilise lahenduse ja tööriistade valimine, modelleerimine;
- andmebaas – kontseptuaalse andmemudeli tõlkimine inglise keelde (et genereerida selle põhjalt ingliskeelne andmebaasi disaini mudel), andmebaasi loomise ja näiteandmete sisestamise skriptide koostamine, andmebaasisüsteemi versioonide uuendamine, muutunud nõuete jõustamine;
- dokumentatsioon – eeltöö ja nõuete dokumenteerimine, lõputöö kirjutamine, tellijale antava dokumentatsiooni koostamine;
- projekt – projektijuhtimine, tööriistade valik meeskonnatööks ning pikemad koosolekud;
- rakendus – arenduskeskkonna loomine, rakenduse genereerimine ja arendamine, esmane testimine, teise arendaja koodi ülevaatamine ja testimine, muutunud nõuete arendamine;
- testimine – testimine, testjuhtude loomine ja täiendamine;
- õppimine – tööriistade kasutamine, veebikursused, loengute kuulamine, keerulisemate programmeerimisülesannete lahendamine, andmebaasi disainimise põhimõtete uurimine.

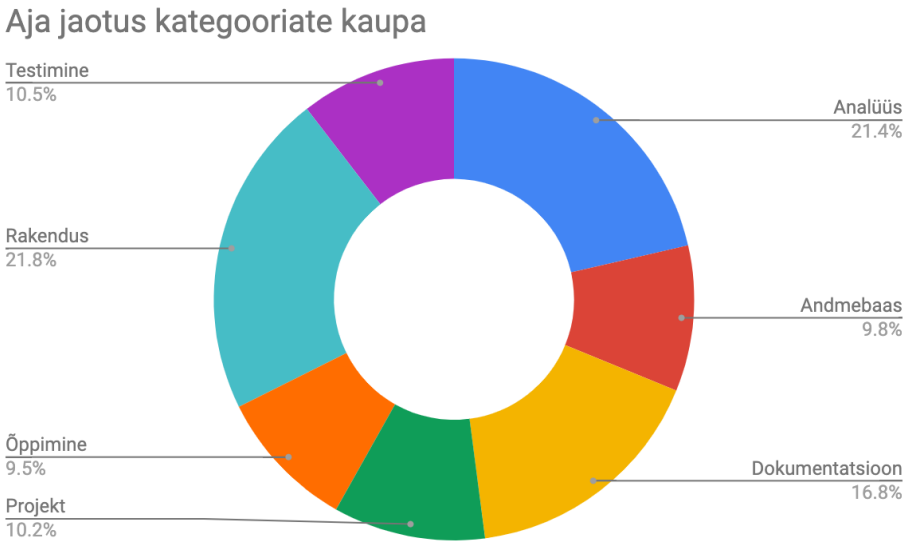
Järgnevalt on toodud väljavõtted käsitsi koostatud tegevuste logist, projektihaldussüsteemist Jira ning Bitbucket koodirepositooriumi projektidest.

4.2.1 Käsitsi koostatud tegevuste logi

Logifailis oli kokku 420 sissekannet tööde mahuga 1 136 tundi. See jaotub järgnevalt:

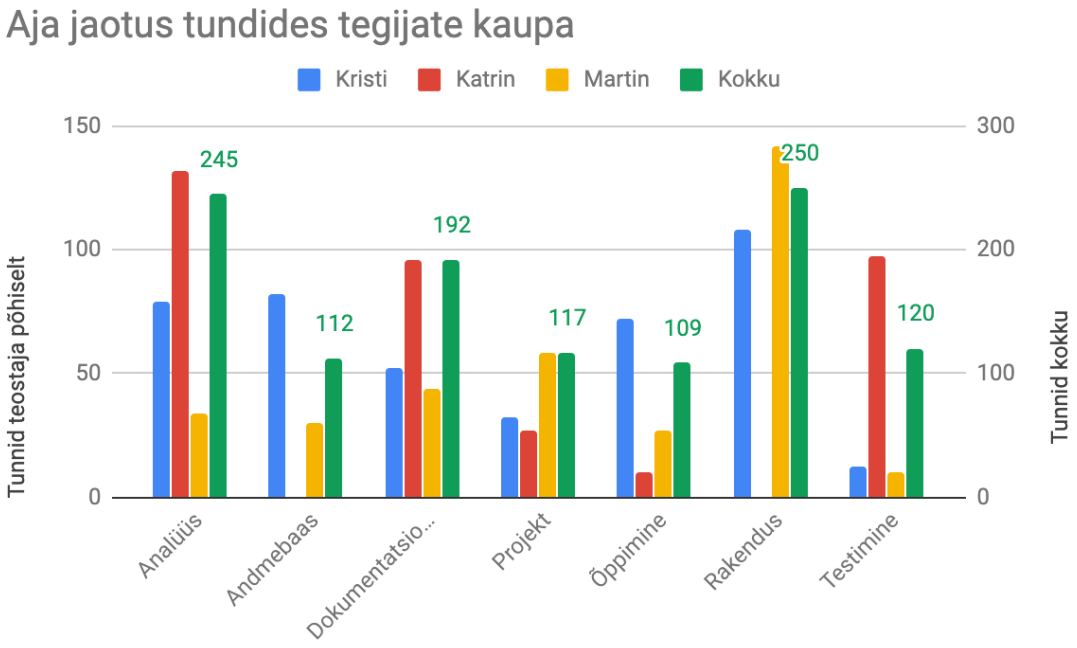
- Martin Paroll: 345 tundi,
- Katrin Avloi: 353 tundi,
- Kristi Seemen 438 tundi.

Joonis 25 esitab tegevuste mahud protsentuaalselt kategooriate kaupa.



Joonis 25. Aja jaotus protsentuaalselt kategooriate kaupa.

Joonis 26 esitab tegevuste mahud tundides kategooriate ja tegijate kaupa. Iga töö autori isiklikud tunnid on kuvatud vasaku X-telje skaala alusel ning kokku tundide arv paremal pool oleva X-telje skaala alusel.

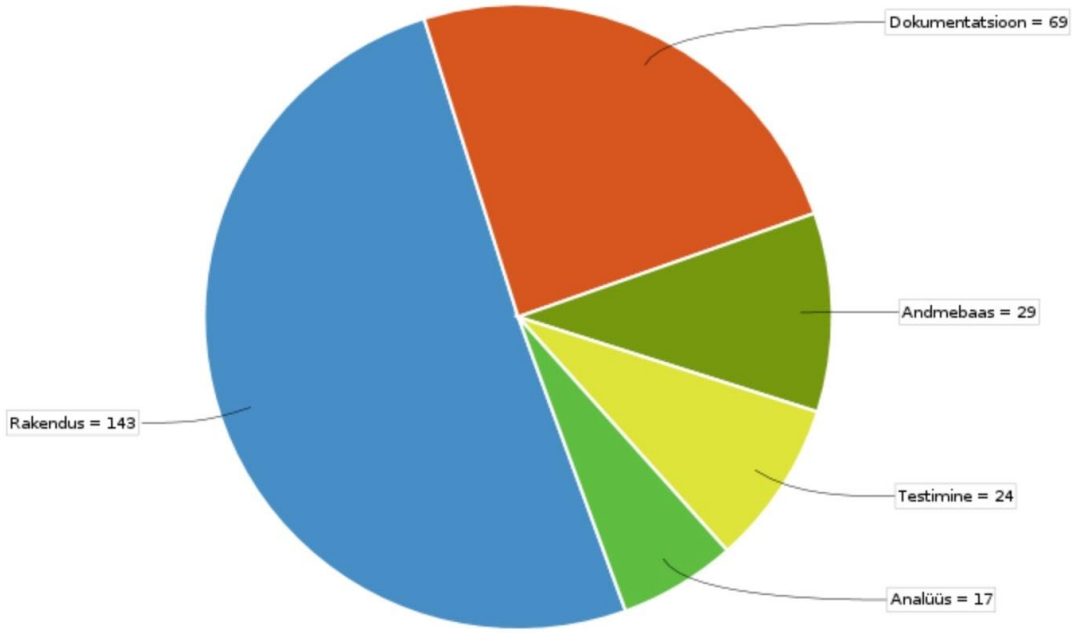


Joonis 26. Aja jaotus tundides kategooriate ja tegijate kaupa.

4.2.2 Projektihaldussüsteemi väljavõte

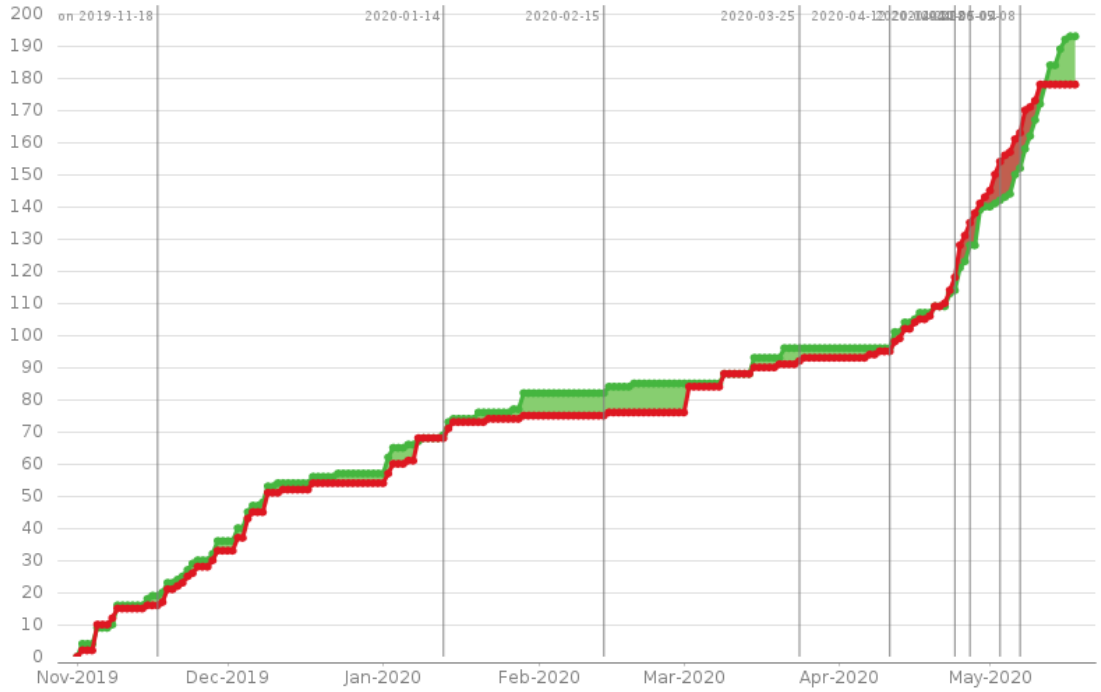
Järgnevalt on toodud väljavõtted projektihaldussüsteemist Jira, kus kokku oli 251 ülesannet.

Joonis 27 on toodud väljavõte ülesannetest kategooriate kaupa. Osa ülesandeid kuulus korraga mitmesse kategooriasse. Analüüsi (6%) ja rakenduse (56%) ülesannete arv protsentuaalselt erineb märgatavalt sellest, mis on toodud käsitsi koostatud tegevuste logi juures (vt Joonis 25). See on tingitud ülesande tükeldamise ja selle täitmiseks kuluva aja erinevusest. Analüüsi ülesanded olid sõnastatud üldiselt, mistõttu nende sisu oli laiem ja ühe ülesande täitmiseks kulus palju tunde. Rakenduse tööd olid seevastu väga detailsed, mistõttu oli ka nende tükkide arv suurem, samas aga kulus ühe ülesande täitmiseks vähem aega.



Joonis 27. Ülesannete jaotus kategooriate kaupa.

Joonis 28 on esitatud punasega loodud ülesannete arv ning rohelisega lahendatud ülesannete arv koos iteratsioonidega (vertikaalsed jooned). Iteratsiooni lõpp tähendas tavaliselt ka tellijale uue versiooni üleandmist. Joonise ajateljel on viimased 200 päeva, illustreerimaks kuidas projekti lõpu faasides iteratsioonide aeg vähenes ning koos sellega versioonide üleandmine tellijale sagenes. Esitatud perioodil enamus aega oli varasemalt püstitatud ülesannete lahendamise arv suurem kui uute ülesannete püstitamine (roheline joon kõrgemal kui punane).



Joonis 28. Loodud *vs.* lahendatud ülesanded koos iteratsioonidega.

4.2.3 Koodirepositooriumi väljavõte

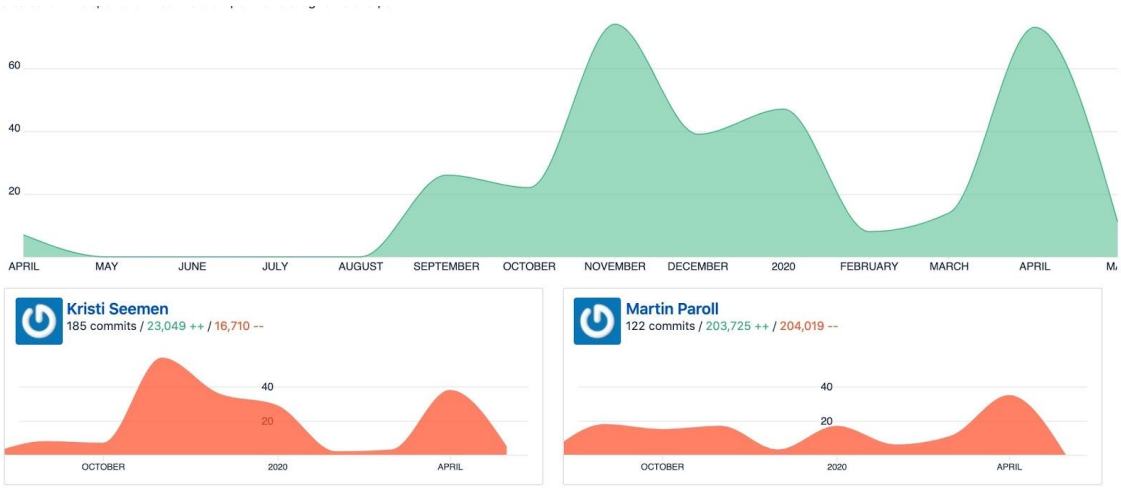
Repositooriumi esimene *commit* on kuupäevaga 9. november 2018. Projekti alguses prooviti erinevaid PHP raamistikke ning esimeses repositooriumi projektis *Kivike* asub ka kõige esimene tellijale tehtud näiterakendus. 20. aprillil 2019 alustasid töö autorid tööd nii öelda nullist. Loodi samasse repositooriumisse täna kasutusel olnud projekt *Viisid* ning genereeriti sinna uus rakendus uue ja põhjalikuma andmebaasi mudeli põhjal.

Tabel 4 on toodud väljavõte projekti *Viisid* koodimuudatuste kohta. Suur erinevus väljavõttes toodud numbrites tuleneb sellest, et suuremate andmebaasi muudatuste korral genereeris Martin Paroll andmebaasi põhjal rakenduse taas nullist, st et repositooriumis kirjutati üle kõik andmebaasiobjektide kontrollerid, mudelid ja vaated.

Tabel 4. Projekti *Viisid* väljavõte koodimuudatustest.

Tegevus	**Martin Paroll**	**Kristi Seemen**
Lisatud ridu	87 453	10 447
Kustutatud ridu	58 440	10 166
Muudetud faile	3 290	1 178
Commit'e	97	153
Muudetud ridu kokku	145 893	20 613

Projekti *Viisid* koodimuudatuste väljavõte ajateljel on toodud Joonis 29. Antud joonis näitab teostatud koodimuudatusi kokkuvõtvalt ning tegijate kaupa.



Joonis 29. Projekti *Viisid* väljavõte ajateljel.

4.3 Hinnang projekti tegemisele

Projekti alustati 2018. aasta oktoobris ning tulem antakse EKM-ile üle hiljemalt 30. juunil 2020. Viiside infosüsteemi realiseerimine pika perioodi vältel oli võimalik, kuna puudusid piirangud eelarve osas ja tähtaeg oli paindlik. Lisaks tuli arvestada, et töö autorid täitsid vajalikke ülesandeid muude põhikohustuste kõrvalt.

Ülesanne lahendati esitatud tellimuse alusel meeskondliku projekti vormis. Töö autorid olid varasemalt koos projekte teinud nii tööalaselt kui ka äriinfotehnoloogia bakalaureuseõppe jooksul, mistõttu oli enne käesoleva projekti alustamist välja kujunenud usalduslik tööalane suhe. Oli teada, et kõik panustavad ühise eesmärgi saavutamisse, peavad kinni lubadustest ja kokkulepitud töödistsipliinist. See võimaldas projekti alustamisel rollide jaotust meeskonnas mitte kokku leppida ning otsustada tööülesannete jagunemise jooksvalt.

Meeskonnas töö tegemist hindavad töö autorid väga suureks eeliseks. Küsimuste korral ja probleemide tekkimisel oli alati võimalik teiste süvitsi teemat valdavate liikmetega nõu pidada, mistõttu jõuti töö autorite arvates paremate lahendusteni. Samuti saab positiivse aspektina välja tuua ühise meeskondliku motiveerimise, sest ühe liikme panus aitas kaasa ka teiste liikmete panustamisele. Meeskondliku töö kitsaskohana saab välja

tuua suurema ajakulu, kuna aega kulus ühistele koosolekutele ning olulisemad asjad tuli teistega kokku leppida.

Meeskondliku töö organiseerimiseks tööriistade valikul lähtuti senisest kogemusest ning need toetasid eesmärgi saavutamist. Kõige olulisemaks peavad töö autorid Kanbani tahvlit ja regulaarseid projekti koosolekuid, mis aitasid loodava tarkvara valmimist tähelepanu keskmes hoida.

Projekti tegemisel lähtuti enda kokku pandud arendusmetoodikast, kuna see võimaldas kasutada erinevate meetodite elemente ning neid igal hetkel sobivamaks kohandada. Töö autorite hinnangul oli selline lähenemine ainuõige ning sobis antud projekti ja meeskonda väga hästi.

Projekti alguses lepiti kokku üldine arendusprotsess, kuid detaile parendati jooksvalt vastavalt vajadusele, nt loodi juurde eraldi rakenduse teemade kanal Slacki, hakati tegema koodi ülevaatust (päris alguses ei tehtud), kaasati tellijat rohkem protsessi (sh korraldati tihedamalt koosolekuid). Iteratsioone planeeriti paindlikumalt kui Scrum seda ette näeb. Iteratsiooni pikkus ei olnud fikseeritud ning vajadusel tehti muudatusi tööde nimekirjas. Kõige olulisem oli tellijale uute funktsionaalsuste tarnimine. Selle saavutamiseks toimis meeskonnas väga hästi Kanbani lähenemine, mille kohaselt tuleb keskenduda ühele pooleli olevale ülesandele korraga. Tööde kuhjumist ühele inimesele välditi vajadusel ülesannete ümberjagamisega.

Koostööd projekti tellijaga hindavad töö autorid väga heaks. Kuigi asuti erinevates linnades, siis füüsiline kaugus ei olnud projekti valmimisel kuidagi takistuseks. Tellija oli alati valmis kaasa mõtlema ja pidama videokoosolekuid ning vajaduse korral leiti võimalus ka füüsiliseks kohtumiseks. Kuna töö autorite jaoks oli valdkond võõras, siis suurimaks väljakutseks oli tellijaga selles osas ühise keele leidmine ja põhiobjekti keerukuse mõistmine. Tellija oli siinkohal väga kannatlik ning valmis selgitama teemasid korduvalt ja erinevatest aspektidest lähtuvalt. Koostöö käigus tuli arvestada, et infosüsteemide arendusprotsessis osalemine ei ole tellija põhikompetents. Oluline oli tellija poolt testimisse panustamine, kuna selle käigus selgusid mõned vead, mida töö autorid ei olnud osanud ise arvesse võtta.

Koostööd lõputöö juhendajaga hindavad töö autorid suurepäraseks. Ta oli alati valmis kohtuma ning talle esitatud küsimused said kiire ja põhjaliku vastuse. Ta süvenes

kõikidesse tõstatud teemadesse väga detailselt. Juhendaja tegi ettepanekuid keerulisemate tekkinud probleemide lahendamiseks ning jagas soovitusi erinevate allikmaterjalide osas. Kuna projekti käigus realiseeriti EKM-is kasutusele võetav tarkvara, siis oli väga oluline tagada selle jätkusuutlikkus. Valitud juhendaja sobis oma teadmiste, põhjalikkuse ja kaasamõtlemise valmidusega selle ülesande lahendamist väga hästi toetama. Retrospektiivis leiavad töö autorid, et koostööd ja juhendaja abi kasutamist oleks saanud veelgi enam efektiivistada regulaarsete kohtumiste kokku leppimisega, kuna see oleks tõenäoliselt motiveerinud valmimise aja osas rohkem pingutama.

Töö autorid omasid varasemat kogemust tarkvara arendusprojektides projektijuhi, analüütiku ja testija rollis. Analüüsi osale kulus küll palju aega, aga protsessi mõttes see raskusi ei valmistanud, va väikeseks väljakutseks oli infosüsteemi andmebaasi struktuuri loomine. Programmeerimisega ei olnud keegi tööalaselt tegelenud, mistõttu oli sellega seotud ülesannete täitmine ja otsuste tegemine aeganõudev. Samuti tuli kulutada aega täiendavate oskuste õppimisele. Tulenevalt oma vähestest kogemustest küsisid töö autorid tarkvara lahenduse osas eksperthinnanguid ka pikaajalise töökogemusega tarkvarainseneri ja IT-juhi taustaga kolleegidelt.

Töö autorite konsensuslikul hinnangul oli kõikide liikmete panus oluline ja määrav projekti eesmärgi täitmiseks. Meeskonnasisene tööde jaotus otsustati ühiselt tagamaks piisavalt efektiivne ülesannete täitmine, arvestades sealjuures ka iga liikme isiklikke huve. Kokkuvõttes koostöö meeskonnasiseselt toimis suurepäraselt.

Töö autorite isikliku panuse kirjeldus ja eneseanalüüs on toodud Lisa 1 – Lisa 3.

4.4 Tellija hinnang

Projektile andis oma hinnangu ka tellija esindaja, ERA vanemteadur Taive Särg.

„Eesti Kirjandusmuuseumi Eesti Rahvaluule Arhiivi eesmärk on luua kasutajasõbralik rahvaviiside andmebaas koos metaandmetega. Tallinna Tehnikaülikooli üliõpilased Katrin Avloi, Martin Paroll ja Kristi Seemen lõid andmebaasi struktuuri ja andmete sisestamise keskkonna, kahjuks ei jõudnud luua otsingusüsteemi. Arvestades siiski, et tegemist oli üliõpilastele senitundmatu materjaliga ja töökeskkonnaga, on nad ära teinud suure ja vajaliku töö. Kõige rohkem võttiski alguses aega meie materjali iseärasuste

mõistmine. Tulemus töötab korralikult ja üliõpilased olid töö juures loomingulised, püüdlikud ja kohusetundlikud. Üksikasjades saaks ehk teha mugavamaks andmete sisestamise keskkonda ja kuvamist ekraanil, kuid saame aru, et see ei ole proportsioonis üliõpilastöö oodatava töö- ja ajakuluga.“

5 Kokkuvõte

Käesoleva töö eesmärgiks oli arendada puuduv viiside infosüsteemi tarkvara Eesti Kirjandusmuuseumile. Ülesanne lahendati esitatud tellimuse alusel meeskondliku projekti vormis.

Töö tegemisel järgisid töö autorid enda kokku pandud arendusmetoodikat, mis kasutas iteratiivset arendusmudelit ja laenas nii Kanbani kui ka Scrumi elemente. Tarkvara analüüsimisel ja loomisel kasutati andmekeskset lähenemist. Projekti skoobis olev funktsionaalsus realiseeriti järk-järgult ning rakenduse valideerimisel kasutati agiilset testimist.

Projekti käigus koguti nõudeid, loodi rahvaviiside infosüsteemi andmebaas ja veebirakendus ning testiti realiseeritud süsteemi vastavust nõuetele.

Töö autorite hinnangul oli ainuõige otsus lahendada infosüsteemi disain kasutades andmebaasi ning baasis olevat infot vahendavat veebirakendust. Arenduskeskkonna platvormina kasutati Docker konteineritel põhinevat lähenemist, mis lihtsustas nii arendust kui ka juurutamist. Projekti algusfaasis otsustati läbi viia ainult käsitsi testimine, kuid järeldusena leiavad töö autorid, et suure testimise töömahu ja andmeväljade hulga tõttu oleks pidanud rakendama automaattestimist.

Projekti tulemina antakse Eesti Kirjandusmuuseumile üle toimiv ja kasutusele võtmiseks valmis tarkvara koos lähtekoodi ja projekti dokumentatsiooniga. Töö autorid soovitavad valminud tarkvara edasi arendada, et saavutada kõik viiside infosüsteemi eesmärgid.

Kasutatud kirjandus

[1] 13th Annual State of Agile Survey. – CollabNet VersionOne. [WWW] https://stateofagile.com/home (18.05.2020)

[2] Ahmad, M. O., Markkula, J., Oivo, M. Kanban in software development: A systematic literature review. – *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, sept 2013, 9–16. [*Online*] IEEE Xplore, doi: 10.1109/SEAA.2013.28

[3] Aibast, K. Baastabelite kitsenduste nimede ühtlustamiseks mõeldud PostgreSQL laienduse loomine : bakalaureusetöö. Tallinn : Tallinna Tehnikaülikool, 2018. [WWW] https://digikogu.taltech.ee/et/Item/05157bb4-763e-4a9d-bc50-2025692b8e1d (06.05.2020)

[4] AuthComponent. – CakePHP 3.8. [WWW] https://book.cakephp.org/3/en/controllers/components/authentication.html (08.05.2020)

[5] Avaliku teabe seadus. (Vastu võetud 15.11.2000) – Riigi Teataja. [WWW] https://www.riigiteataja.ee/akt/122032011010?leiaKehtiv (07.05.2020)

[6] Bauer, R. What's the Diff: VMs vs Containers. – *Backblaze Blog*, 28.06.2018. [WWW] https://www.backblaze.com/blog/vm-vs-containers/ (07.05.2020)

[7] Bitbucket | The Git solution for professional teams. – Atlassian. [WWW] https://bitbucket.org (30.04 2020)

[8] Bootstrap. [WWW] https://getbootstrap.com/ (30.04.2020)

[9] Botero *et al*. Drawing Together, Infrastructuring and Politics for Participatory Design. Finland : University of Oulu, 2019. [WWW] http://hdl.handle.net/2142/103012 (11.05.2020)

[10] Brinkkemper, S. Method engineering: engineering of information systems development methods and tools. – *Information and Software Technology*, 1996, 38 (4), 275–280. [*Online*] ScienceDirect, doi: 10.1016/0950-5849(95)01059-9

[11] CakePHP - Build fast, grow solid | PHP Framework | Home. – CakePHP - The rapid development php framework. [WWW] https://cakephp.org/ (27.04.2020)

[12] CakePHP Conventions: Database Conventions. – CakePHP 3.8. [WWW] https://book.cakephp.org/3/en/intro/conventions.html#database-conventions (06.05.2020)

[13] CakePHP Conventions: View Conventions. – CakePHP 3.8. [WWW] https://book.cakephp.org/3/en/intro/conventions.html#view-conventions (06.05.2020)

[14] Chat, Meetings, Calling, Collaboration | Microsoft Teams. [WWW] https://www.microsoft.com/en-us/microsoft-365/microsoft-teams/group-chat-software (08.05.2020)

[15] Class Cake\Auth\DefaultPasswordHasher. – CakePHP 3.8. [WWW] https://api.cakephp.org/3.8/class-Cake.Auth.DefaultPasswordHasher.html (08.05.2020)

[16] codedance/jquery.AreYouSure. – GitHub. [WWW] https://github.com/codedance/jquery.AreYouSure (30.04.2020)

[17] Coding Standards. – CakePHP 3.8. [WWW] https://book.cakephp.org/3/en/contributing/cakephp-coding-conventions.html (11.05.2020)

[18] Cohn, M. User Stories and User Story Examples. – Mountain Goat Software. [WWW] https://www.mountaingoatsoftware.com/agile/user-stories (18.05.2020)

[19] Diagram Software and Flowchart Maker. [WWW] https://www.diagrams.net/ (04.05.2020)

[20] Dont Repeat Yourself. [WWW] http://wiki.c2.com/?DontRepeatYourself (18.05.2020)

[21] Eessaar, E. A Set of Practices for the Development of Data-Centric Information Systems. – *Information System Development*, Cham, 2014, 73–84. [*Online*] Springer Link, doi: 10.1007/978-3-319-07215-9_6.

[22] Eessaar, E. Ülikooli infosüsteemi vastuvõtuaegade allsüsteem. Näiteprojekt õppeaines "Andmebaasid II". [*Online*] TalTech Tarkvarateaduse instituut (22.04.2020)

[23] Eesti Kirjandusmuuseumi failirepositoorium Kivike. [WWW] http://kivike.kirmus.ee/index.php?dok_id=1&module=2&op=/ (07.05.2020)

[24] Eesti regilaulude andmebaas. [WWW] http://www.folklore.ee/regilaul/andmebaas (07.05.2020)

[25] Exploratory Testing. – *Agile Alliance*, 17.12.2015. [WWW] https://www.agilealliance.org/glossary/exploratory-testing/ (11.05.2020)

[26] Extreme Programming: A Gentle Introduction. [WWW] http://www.extremeprogramming.org/ (18.05.2020)

[27] Full Lifecycle Modeling for Business, Software and Systems | Sparx Systems. [WWW] https://sparxsystems.com/products/ea/ (30.04.2020)

[28] Grid system. – Bootstrap. [WWW] https://getbootstrap.com/docs/4.0/layout/grid/ (30.04.2020)

[29] Internationalization & Localization. – CakePHP 3.8. [WWW] https://book.cakephp.org/3/en/core-libraries/internationalization-and-localization.html (10.05.2020)

[30] Jira | Issue & Project Tracking Software. – Atlassian. [WWW] https://www.atlassian.com/software/jira (30.04.2020)

[31] jQuery Datepicker for Bootstrap and Material Design. – Gijgo. [WWW] https://gijgo.com/datepicker (30.04.2020)

[32] Karwin, B. SQL Antipatterns. Pragmatic Bookshelf, 2010. [*Online*] O'Reilly (18.05.2020)

[33] Lumen - PHP Micro-Framework By Laravel. [WWW] https://lumen.laravel.com/ (18.05.2020)

[34] McArthur, K. Pro PHP: Patterns, Frameworks, Testing and More. Apress, 2008. [*Online*] O'Reilly (01.05.2020)

[35] Notepad++. [WWW] https://notepad-plus-plus.org/ (09.05.2020)

[36] Once-Only | TOOP.EU. [WWW] https://toop.eu/once-only (07.05.2020)

[37] Oracle VM VirtualBox. [WWW] https://www.virtualbox.org/ (15.05.2020)

[38] Overview of Docker Compose. – Docker Documentation. [WWW] https://docs.docker.com/compose/ (30.04 2020)

[39] pgAdmin - PostgreSQL Tools. [WWW] https://www.pgadmin.org/ (09.05.2020)

[40] PHP: password_hash. – PHP Manual. [WWW] https://www.php.net/manual/en/function.password-hash.php (08.05.2020)

[41] Porebski, B., Przystalski, K., Nowak, L. Building PHP Applications with Symfony™, CakePHP, and Zend® Framework. Wrox Press, 2011. [*Online*] O'Reilly (27.04.2020)

[42] Rising, L., Janoff, N. S. The Scrum software development process for small teams. – *IEEE Software*, 2000, 17 (4), 26–32. [*Online*] IEEE Xplore, doi: 10.1109/52.854065

[43] Sein *et al*. Action Design Research. – *MIS Quarterly*, 2011, 35 (1), 37–56. [*Online*] JSTOR, doi: 10.2307/23043488

[44] Slim Framework. [WWW] http://www.slimframework.com/ (18.05.2020)

[45] Software Testing Methods. – *Software Testing Fundamentals*, 12.01.2011. [WWW] http://softwaretestingfundamentals.com/software-testing-methods/ (11.05.2020)

[46] Sotsiaalkaitse infosüsteemi arendustööd. – Riigihangete register. [WWW] https://riigihanked.riik.ee/rhr-web/#/procurement/1543287/documents?group=B (11.05.2020)

[47] Sourcetree | Free Git GUI for Mac and Windows. – Atlassian. [WWW] https://www.sourcetreeapp.com (30.04.2020)

[48] Sublime Text - A sophisticated text editor for code, markup and prose. [WWW] https://www.sublimetext.com/ (09.05.2020)

[49] Sõnaveeb. [WWW] https://sonaveeb.ee/ (22.04.2020)

[50] Table Objects: Basic Usage. – CakePHP 3.8. [WWW] https://book.cakephp.org/3/en/orm/table-objects.html#basic-usage (06.05.2020)

[51] Table Objects: Event List. – CakePHP 3.8. [WWW] https://book.cakephp.org/3/en/orm/table-objects.html#event-list (08.05 2020)

[52] Top 10 Benefits of Docker - DZone DevOps. – dzone.com. [WWW] https://dzone.com/articles/top-10-benefits-of-using-docker (07.05.2020)

[53] Tree. – CakePHP 3.8. [WWW] https://book.cakephp.org/3/en/orm/behaviors/tree.html (04.05.2020)

[54] Unified Modeling Language. [WWW] https://www.uml-diagrams.org/ (15.05.2020)

[55] Valentina Studio Overview. – Paradigma Software. [WWW] https://www.valentina-db.com/en/valentina-studio-overview (09.05.2020)

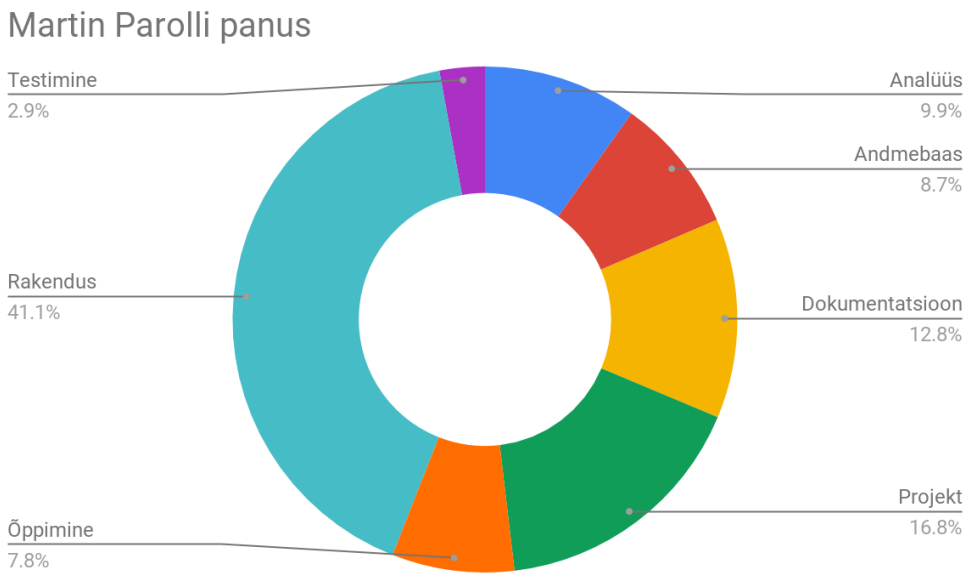
[56] Vellemaa, A.-S. Mõned disainimustrid klassifikaatorite esitamiseks SQL andmebaasides : bakalaureusetöö. Tallinn : Tallinna Tehnikaülikool, 2015. [WWW] https://digikogu.taltech.ee/et/Item/fb7e7116-a202-4fd3-a7bc-be5134d9a975 (04.05.2020)

[57] Where work happens. – Slack. [WWW] https://slack.com/ (01.05.2020)

[58] Yii Framework. [WWW] https://www.yiiframework.com/ (18.05.2020)

Lisa 1 – Martin Parolli panuse kirjeldus ja eneseanalüüs

Minu peamisteks rollideks käesoleva projekti läbiviimisel olid projektijuhtimine, tööriistade valik nii meeskonnatöö kui tehnilise arhitektuuri osas ning rakenduse programmeerimine. Kokku märkisin käsitsi kirjutatud logisse 345 tundi, mille protsentuaalset jaotust esitab Joonis 30.



Joonis 30. Martin Parolli ajaline panus protsentuaalselt kategooriate kaupa.

Projektijuhina oli minu ülesandeks koosolekute ja infovahetuse korraldamine, samuti meeskonnatööks sobivate tööriistade valik ja seadistamine. Kuna ma olen projekti- ja tootejuhina töötanud erinevates tehnoloogiasektori ettevõtetes, siis mul on hea ülevaade nii tarkvara arendusprotsesside juhtimise kui ka tööriistade parimatest praktikatest. Kokkuvõttes projektijuhtimine ja koostöö suunamise rolli kandmine vastas ootustele ning suuremaid väljakutseid ei esitanud.

Minu suurim roll projektis oli seotud rakenduse arendamisega. Valisin selle oma peamiseks teemaks, kuna see tundus põnev väljakutse. See osutus keerulisemaks, kui ma esialgu arvasin.

Minu esimeseks ülesandeks oli valida infosüsteemi tehniline arhitektuur ning otsustada millise programmeerimiskeele me ülesande lahendamiseks valime ja sõltuvalt sellest, millise raamistiku valime. Nende teemadega ma varasemalt kokku puutunud ei olnud. Seetõttu lugesin sel teemal palju ning küsisin tarkvara lahenduse osas eksperthinnanguid ka pikaajalise töökogemusega tarkvarainseneri ja IT-juhi taustaga kolleegidelt. Umbes kolmandik õppimiseks märgitud tunde kulus selle valdkonna tundmaõppimiseks. Keerukusest hoolimata oli see minu jaoks väga põnev ning arendav osa projektist.

Arenduse tööriistade valikuga olen ma väga rahul, kuna peale esmast keskkondade loomist said projekti teised liikmed kohaliku arenduskeskkonna seadistatud vähese vaevaga. Samuti oli tellijale testkeskkonna ülesseadmine ja versiooniuuenduste tegemine valitud tööriistadega lihtne ja kiire.

Programmeerimine nõudis oodatust oluliselt rohkem aega, seda isegi PHP raamistiku genereeritud baaskoodi kasutades. Arenduse käigus tekkinud vigade leidmine ja silumine võttis vahest äärmiselt kaua aega. PHP keelega olime õppekava jooksul kokku puutunud vaid loetud tunnid. Kaks kolmandikku logitud õppimise ajast kulus erinevatele PHP kursustele.

Rakenduse arendamisel oli mul mitmeid ülesandeid. Lisasin raamistiku genereeritud koodibaasile raamistiku enda tööriistadel baseeruva autentimise ning autoriseerimise. Kirjutasin nullist rakendusse sündmuste salvestamise. Realiseerisin andmebaasis jõustatud kitsenduste täidetuse kontrollist tulenevate vigade kinnipüüdmise ning kasutajale esitamise, kasutades ära raamistiku tõlkefunktsionaalsust. Lisasin kasutajaliidese skaleeruvaks esitamiseks Bootstrap stiilid ja ruudustiku. Viimase rakendamise jaoks kirjutasin kõikides vaadetes skriptiga ringi HTML klassid.

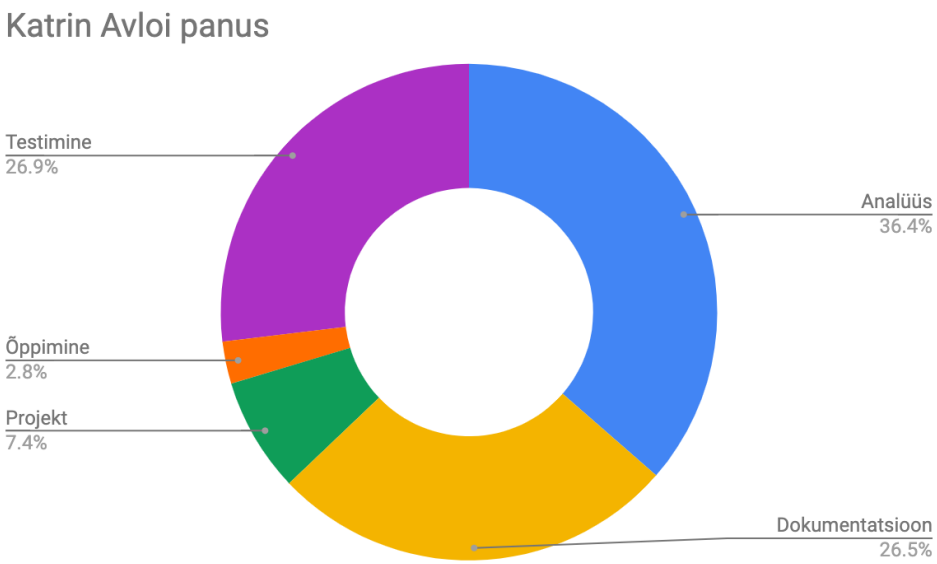
Lisaks oli minu üheks ülesandeks ka teise arendaja koodi ülevaatus ja testimine. Nii käis üks ülesanne vahest korduvalt kahe arendaja vahel edasi-tagasi. Arendasime peamiselt repositooriumi projekti harus nimega *prelive*. Iga iteratsiooni lõpus liitsin ma *prelive* haru koodi haruga *master*, valideerisin põhifunktsionaalsused ning seejärel võeti see kasutusse tellija juures asuvas testkeskkonnas, kus ma taaskord rakenduse korrektse toimimise valideerisin. Nende kahe teema tõttu kulus mul antud valdkonnale kokku palju tunde.

Kokkuvõttes olid minu jaoks kõige keerukamateks töödeks programmeerimine ning lõputöö kirjutamine. Tänu varasemale tööalasele kogemusele, oli minu vedada olnud valdkondadest mulle kõige lihtsam projektijuhtimine. Programmeerimise ning platvormide osa arendas mind käesoleva töö juures kõige enam. Kõige olulisem kogu projekti eduka valmimise juures oli minu jaoks aspekt, et saime selle teha meeskonnatööna. Toetusime üksteise tugevustele ning motiveerisime üksteist panustades töösse. Samuti motiveeris mõte, et loome reaalselt kasutusse võetavat infosüsteemi tarkvara, mis tulevikus väärtust loob.

Lisa 2 – Katrin Avloi panuse kirjeldus ja eneseanalüüs

Eeltöö käigus oli minu ülesandeks koostada visiooni ja skoobi dokument leppimaks tellijaga üldisel tasandil kokku selles, millise lahenduse ja funktsionaalsusega edasi liigume. See oli oluline, et tagada ühtne arusaam kogu projekti meeskonna siseselt.

Projekti käigus kujunes minu põhilisteks rollideks analüüs ja testimine ning lisaks kulutasin suure hulga aega dokumentatsiooni koostamisele. Käsitsi koostatud tegevuste logi alusel kulus mul nende tööde tegemiseks 353 tundi. Joonis 31 on toodud täpsem ülevaade minu ajalise panuse kohta protsentuaalselt kategooriate kaupa.



Joonis 31. Katrin Avloi ajaline panus protsentuaalselt kategooriate kaupa.

Analüüsi tegemine on üheks minu tugevuseks, kuna ärivajaduste selgitamine ja nõuete kirjeldamine on kuulunud pikalt minu töökohustuste hulka. Seetõttu näitasin ma hea meelega initsiatiivi selle protsessi alustamisel ja juhtimisel. Rahvaviiside valdkond oli minu jaoks võõras ning analüüsi algusfaasis selgus, et tegemist on veel keerulisema ja andmemahukama objektiga, kui ma olin arvanud. Sel põhjusel kulus nõuete kogumiseks oodatust oluliselt rohkem aega.

Teema ja detailide mõistmiseks tundus loogiline alustada andmete analüüsist, et saada ülevaade rahvaviisi detailidest. Varasema kogemuse alusel lõin eraldi töödokumendi

süsteemi hallatavate andmete kirjeldamiseks ja tellijaga nõuete kokku leppimiseks. Protsessi käigus panin ma kirja andmeväljade nimetused, definitsioonid, andmetüübid (nt tekst, number, valikväärtus), pikkused ja esinemise sageduse (kohustuslikkus, ühe või mitme väärtuse lubamine). Üritades kogutud info alusel andmeid loogiliselt grupeerida, jõudsime ühiste arutelude tulemusel töös toodud andmeobjektide ja funktsionaalseteks allsüsteemideks jagamiseni.

Lisaks kirjeldasin ma ka funktsionaalsed allsüsteemid ja nende eesmärgid ning panin kirja kasutusjuhud. Juba eelmainitud töödokumendi alusel koostasin registrite olemitüüpide ja atribuutide sõnalised kirjeldused. EA-d kasutades modelleerisin kasutusjuhtude diagrammid ja funktsionaalsete allsüsteemide äriarhitektuuri joonised. Kuna antud CASE vahendiga mul kogemus puudus, siis pidin enne natukene õppima, kuidas seal projekte ja diagramme teha. Selleks kasutasin nii juhendaja andmebaaside aine materjale kui ka CASE vahendi ametlikku veebilehte.

Kuna teised töö autorid olid valmis koodi kirjutamise ja rakenduse realiseerimisega tegelema, siis konsensuslikult leppisime kokku, et mina sinna ei panusta. Mulle see sobis, kuna koodi kirjutamisega ma tööalaselt tegelenud ei ole ning see võimaldas tegeleda projektis mulle rohkem huvipakkuvate teemadega.

Testimisega olin ma eelnevalt kokku puutunud eelkõige kasutaja funktsionaalsete testide tegemisel. Käesolevas projektis tegin ma testijana nii funktsionaalseid kui ka mittefunktsionaalseid teste läbi kasutajaliidese. Ma kirjeldasin olulisemad testjuhud ning tegin ka *ad hoc* testimist.

Automaattestide praktika mul puudus, kuna minu osalusel projektides pole neid üldjuhul rakendatud. Sellest tulenevalt ei osanud ma testimise algusfaasis juhtida tähelepanu nende vajalikkusele. Tagantjärele analüüsides näen, et automaattestide rakendamine oleks valdkond, kus ma oleksin saanud oma testimise oskusi arendada.

Testimise keskkond oli minu jaoks oluline ja seetõttu hindan kõrgelt selle mugavust. Kasutades Sourcetree tarkvara sain ma väga kiiresti uuendada viimase seisu koodist enda isiklikku arvutisse ja taaskäivitada Dockeri konteinerid, peale mida sain koheselt alustada testimist.

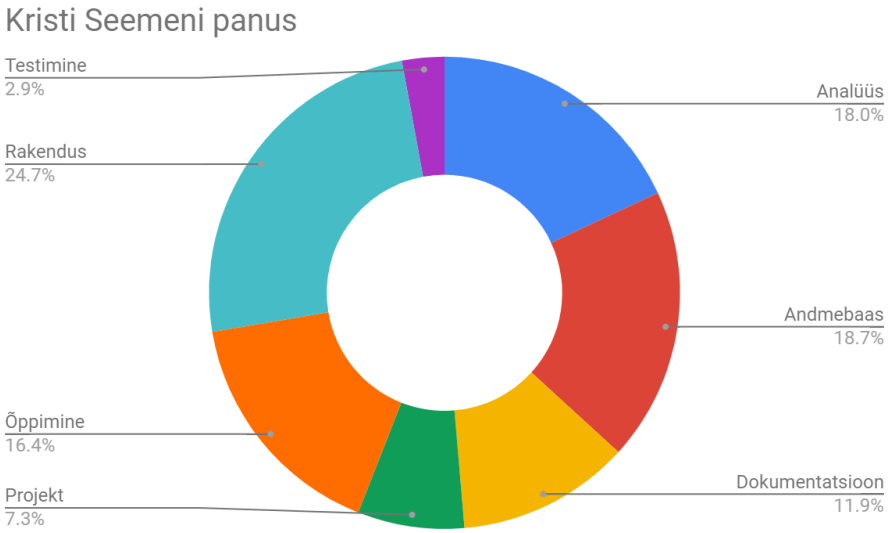
Dokumentatsiooni osas on olnud minu rolliks analüüsi käigus kogutud nõuete kirjapanemine. Samuti on olnud märkimisväärne käesoleva töö kirjutamise maht.

Üheks positiivseks aspektiks oli antud projekti puhul iteratiivse arendusmudeli järgimine. Senine praktika on mul olnud pigem kose mudeli suunitlusega, mis näeb ette, et kindla aja ja eelarvega tuleb saavutada projekti alguses kokkulepitud funktsionaalsus. Hindan väga saadud kogemust. Vähetähtis ei ole ka see, et sai arendada sellist tarkvara, mida tellijal päriselt vaja on.

Kogu projekti jooksul oli minu jaoks väga suur väärtus suhtlus teiste meeskonnaliikmetega ning võimalus ühiselt teemasid arutada. Samuti aitas teiste olemasolu hoida motivatsiooni eesmärgi täitmiseks. Ma kindlasti teeksin lõputööd ka edaspidi meeskondlikus vormis, kui saaksin sinna valida samad liikmed.

Lisa 3 – Kristi Seemeni panuse kirjeldus ja eneseanalüüs

Käesoleva projekti läbiviimisel sain osaleda projekti kõikides etappides, kuna meeskonnaliikmete arv oli väike ja tegemist vajavate tegevuste hulk suur. Kokku kulus mul käsitsi koostatud tegevuste logi alusel 438 tundi, mille jagunemine protsentuaalselt kategooriate kaupa on esitatud Joonis 32.



Joonis 32. Kristi Seemeni ajaline panus protsentuaalselt kategooriate kaupa.

Suurem osa tehtud töödest jagunes analüüsi, andmebaasi ja rakenduse vahel. Kuna kõikides nendes valdkondades tuli ette palju uut ja väljakutset pakkuvat, siis kujunes ka õppimise osakaal päris märgatavaks.

Alguses tundus, et analüüsi etapp võiks olla meie jaoks kõige lihtsam, kuna me oleme kõik seda päris palju teinud. Tegelikkuses selgus, et analüüsis mängib lisaks analüüsioskustele suurt rolli ka valdkonna tundmine. Kuna ma ise olen töötanud ainult finantssektoris, siis kõik arhiivide ja selles hoitavate rahvaviisidega seonduv oli mulle täiesti tundmatu valdkond. Analüüsi etapis oli minu üheks mahukamaks ülesandeks teha EA-s valmis kontseptuaalne andmemudel. Seda olin ma varasemalt teinud ainult andmebaasi ainetes ja siis ka teise CASE vahendiga. Seetõttu tuli mudeli valmimiseks vaadata andmebaaside loengute lindistusi ja EA õppevideoid.

Kontseptuaalse andmemudeli alusel genereerisin EA-s füüsilise disaini andmemudeli PostgreSQL andmebaasisüsteemi jaoks, mille pealt genereerisin omakorda SQL laused baastabelite loomiseks. Kuna olen töötanud ka andmeanalüütikuna, siis kokkupuude andmebaasidega oli olemas, aga andmebaase ma varasemalt loonud ei ole. Ka siin tuli palju juurde õppida, nt kuidas nimetada andmebaasiobjekte, luua domeene ja jõustada kitsendusi.

Kõige suuremaks väljakutseks oli programmeerimine, mida ma olin varasemalt teinud ainult koolitööde jaoks. Minu peamiseks ülesandeks oli vaadete muutmine kasutajale arusaadavaks ja loetavaks. Sellega seoses tuli muuta ka kontrollereid ning vähesel määral ka mudeleid. Kui algselt tundus, et see ei tohiks olla eriti keeruline, siis pidin üsna kiiresti tõdema, et siiski on. Iga üksiku ülesande lahendamiseks oli vaja vaadata CakePHP dokumentatsiooni või pöörduda Google'i poole. Ning kui nendest abi ei olnud, tuli katsetada. Lisaks enda ülesannete lahendamisele ja testimisele, vaatasin üle koodi ja testisin teise arendaja poolt lahendatud ülesandeid.

Kuna minu programmeerimise ülesanded olid üldises plaanis lihtsamat laadi (mitte küll minu jaoks) ning ajaressurss nende lahendamiseks oli suhteliselt piiratud, siis jäi suur osa raamistiku võimalustest avastamata ja kasutamata. Sellest tekkis suurem huvi programmeerimise ja kasutajaliidese disaini (kuna vaadete abil kuvatakse kasutajaliides) vastu.

Kuigi eelpool kirjutatust võib välja lugeda, et projektiks ettevalmistus oli lünklik ja projekti käigus tuli palju juurde õppida, siis pigem ütleksin, et kõik varasemad õpingud ja kogemused võimaldasid üldse selles projektis osaleda ja oma panus anda.

Väga kasulik oli ka tehtud projekti kirjapanek lõputöösse. See on pannud meid analüüsima kõike seda, mida me tegime, kuidas me seda tegime ja miks me seda tegime.

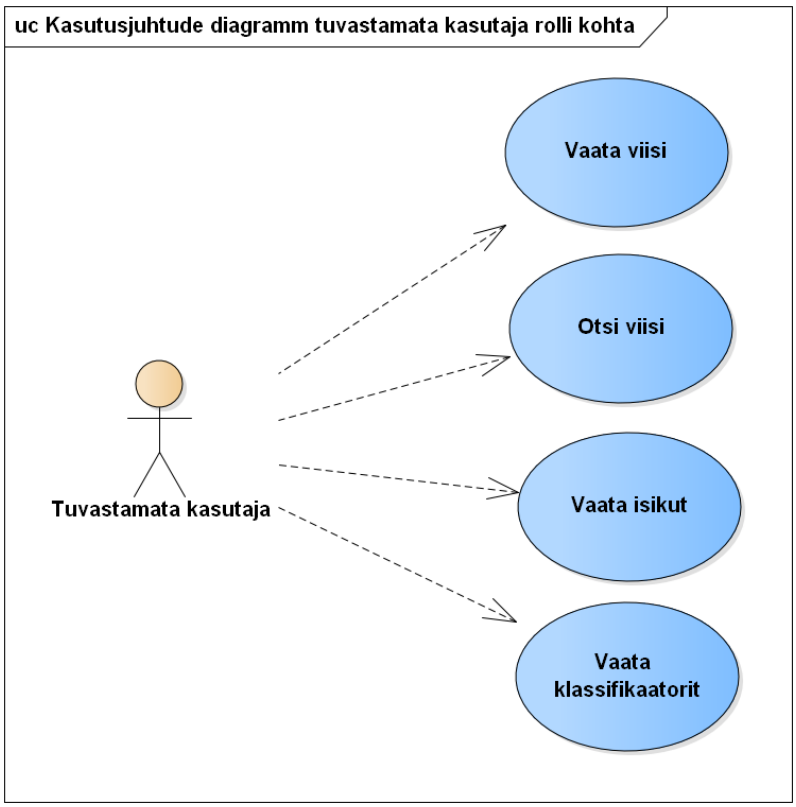
Kogu projektil oli minu jaoks lisaks väga heale meeskonnale ning erinevate oskuste ja teadmiste arendamisele kaks väga olulist lisaväärtust. Esimene neist oli võimalus luua nullist uus süsteem ning osaleda selle kõikides etappides. Seda võimalust mul varasemalt ei ole olnud. Teiseks ja vahest ka olulisemaks väärtuseks oli see, kelle ja mille jaoks me selle süsteemi lõime. Oli väga hea tunne teha ära oluline samm rahvaviiside infosüsteemi arendamisel ja anda oma panus, et sellest saab ühel päeval

andmebaas, mida me kõik saame kasutada. Kui vähegi võimalik, siis osaleksin ka infosüsteemi järgmistes arendusprojektides.

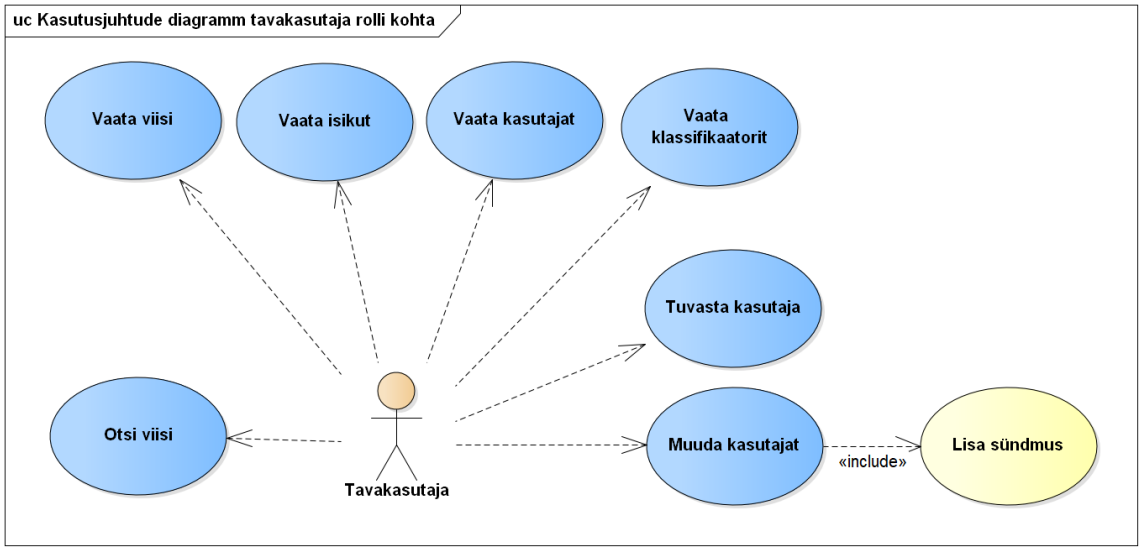
Lisa 4 – Kasutusjuhtude diagrammid

Joonis 33 – Joonis 36 on esitatud kasutusjuhtude diagrammid rollide kaupa. Joonistel toodud kasutusjuhtude värvidel on järgmine tähendus.

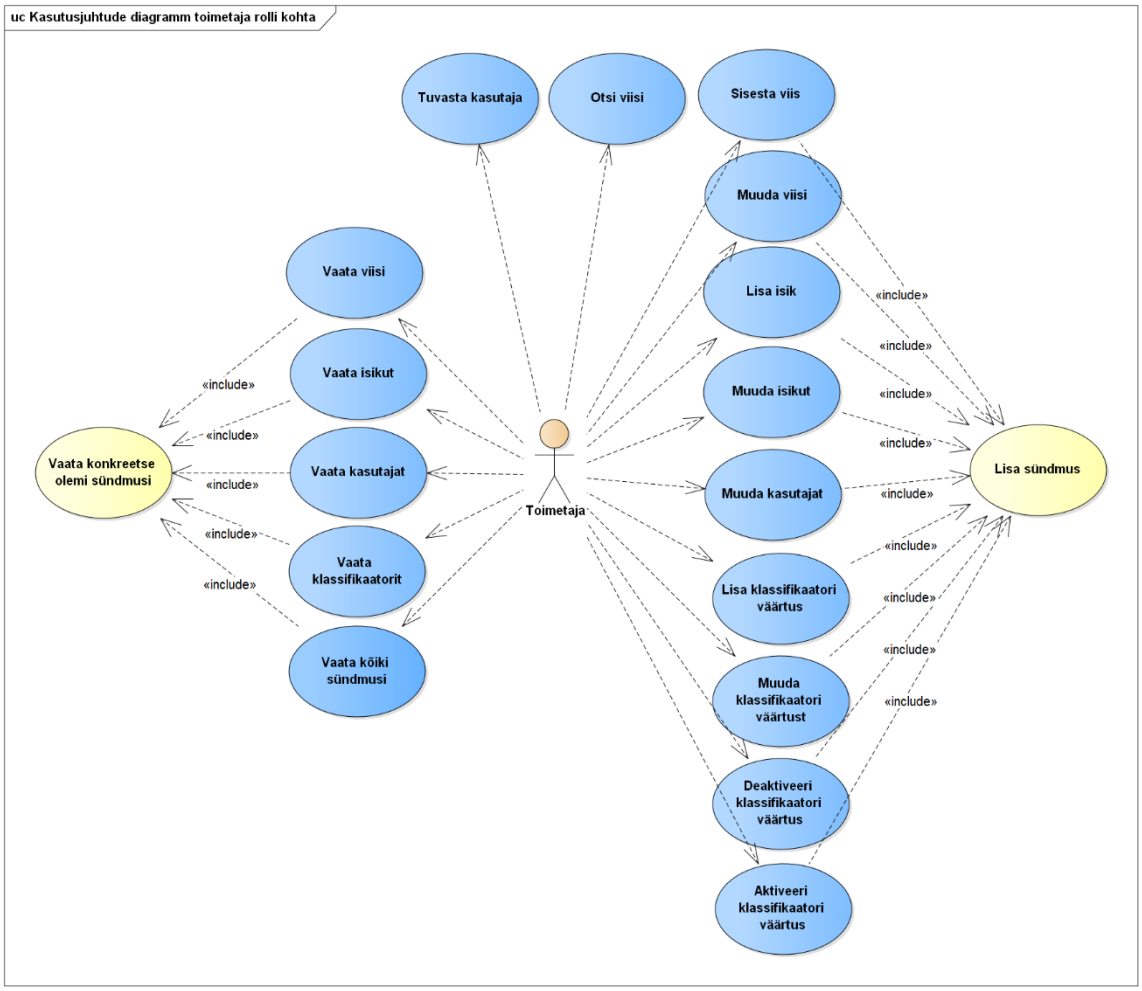
- Sinisega on tähistatud põhikasutusjuhud.
- Kollasega on tähistatud abistavad kasutusjuhud (sisuliselt kasutusjuhu fragmendid), mis on kirja pandud selleks, et mitte kirjeldada mitmekordselt erinevates kasutusjuhtudes esinevat ühesugust funktsionaalsust.



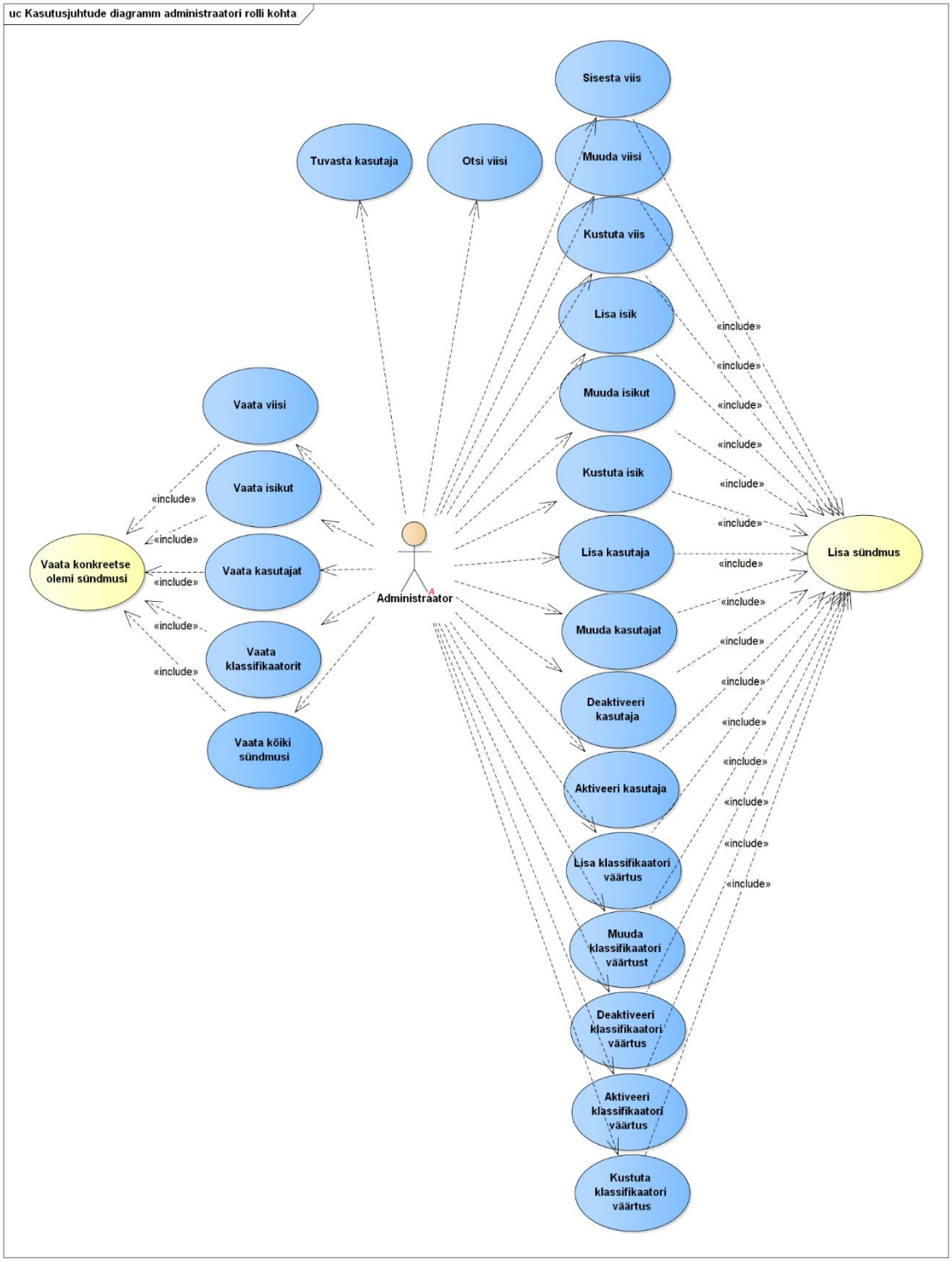
Joonis 33. Kasutusjuhtude diagramm tuvastamata kasutaja rolli kohta.



Joonis 34. Kasutusjuhtude diagramm tavakasutaja rolli kohta.



Joonis 35. Kasutusjuhtude diagramm toimetaja rolli kohta.

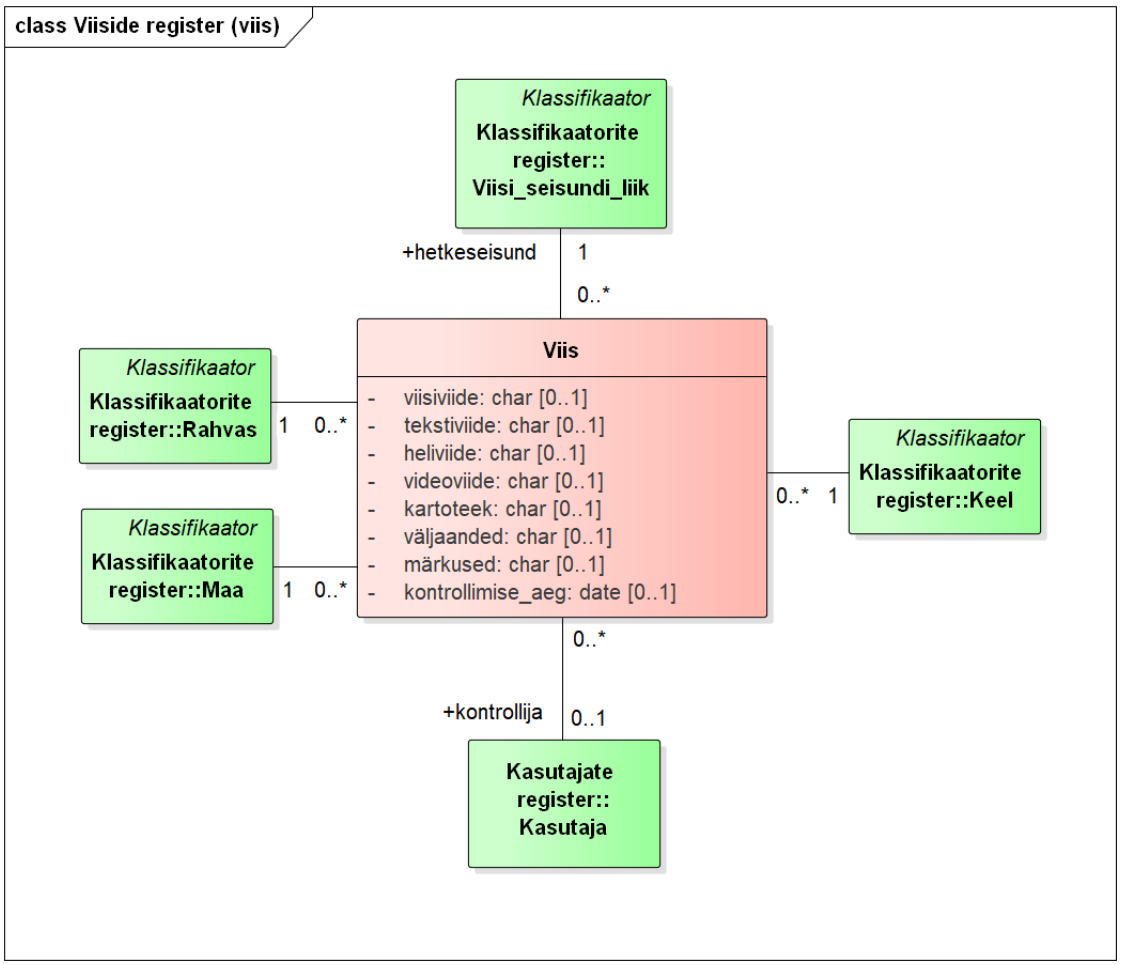


Joonis 36. Kasutusjuhtude diagramm administraatori rolli kohta.

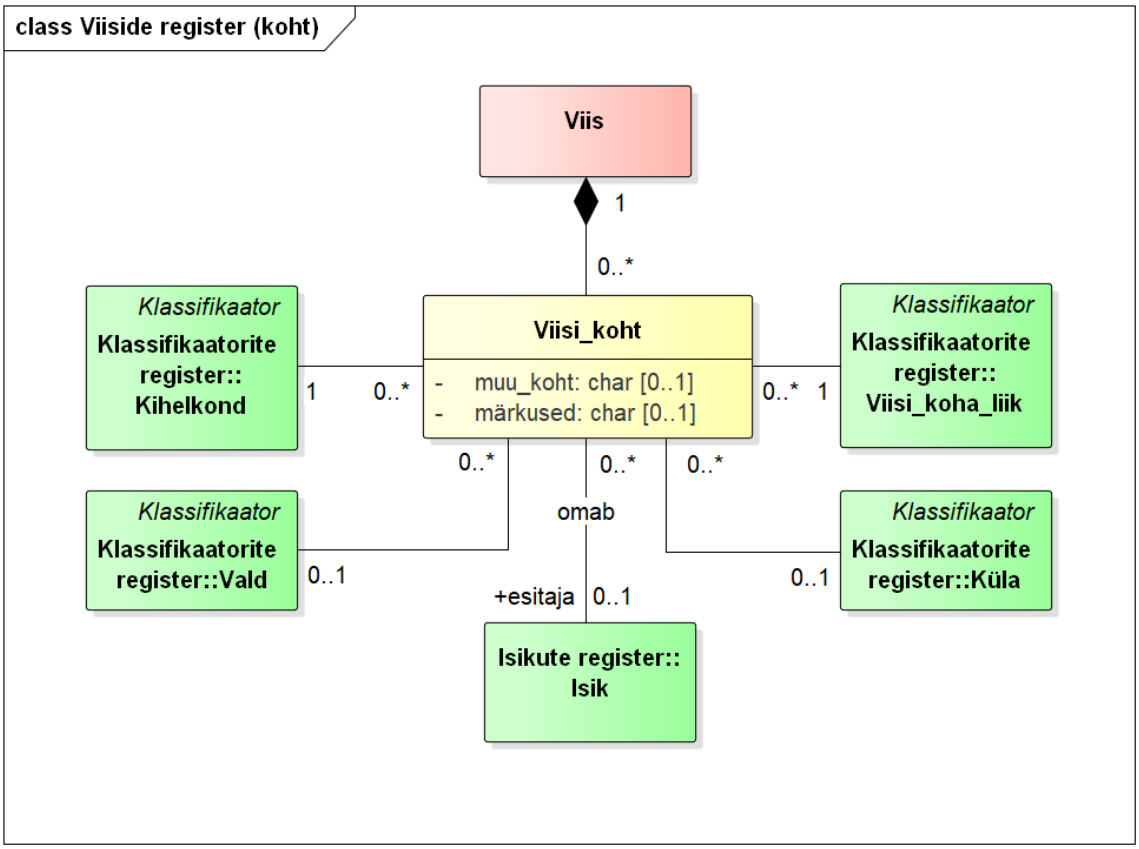
Lisa 5 – Viiside registri analüüs

Joonis 37 – Joonis 43 esitatud olemi-suhte diagrammidel on värvidel järgmine tähendus.

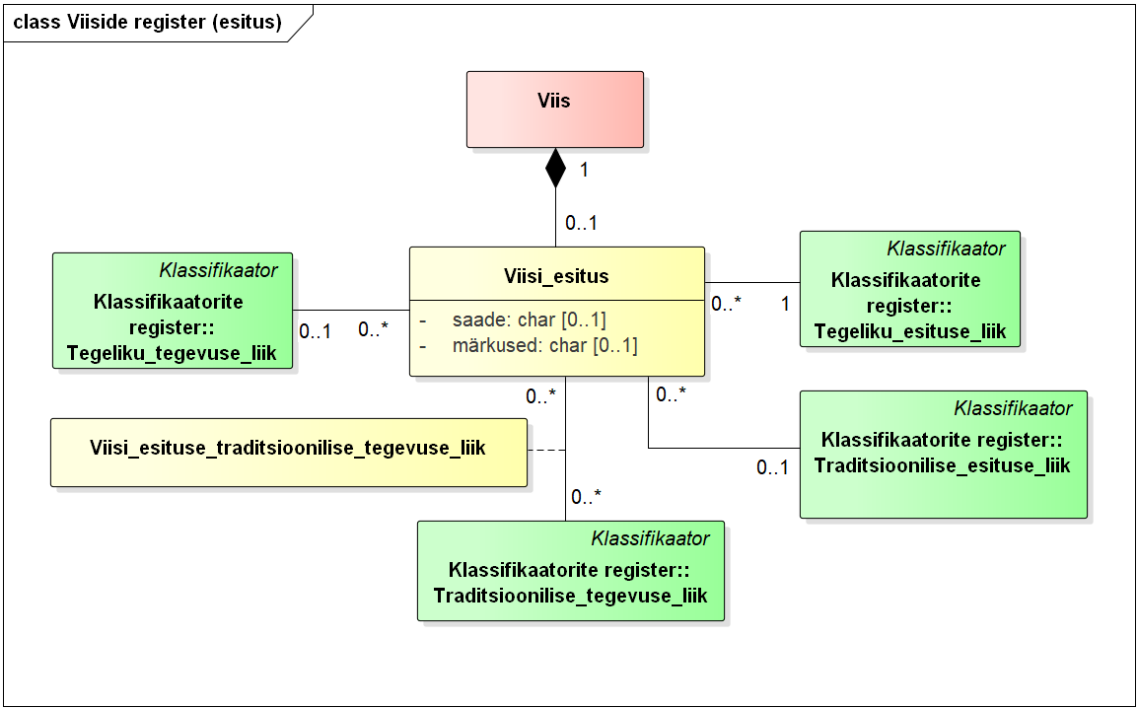
- Punasega on tähistatud registri põhiobjekt.
- Kollasega on tähistatud registrisse kuuluvad alamobjektid (mitte-põhiobjektid).
- Rohelisega on tähistatud teistesse registritesse kuuluvad objektid, mida on vaja viiside allsüsteemi toimimiseks.



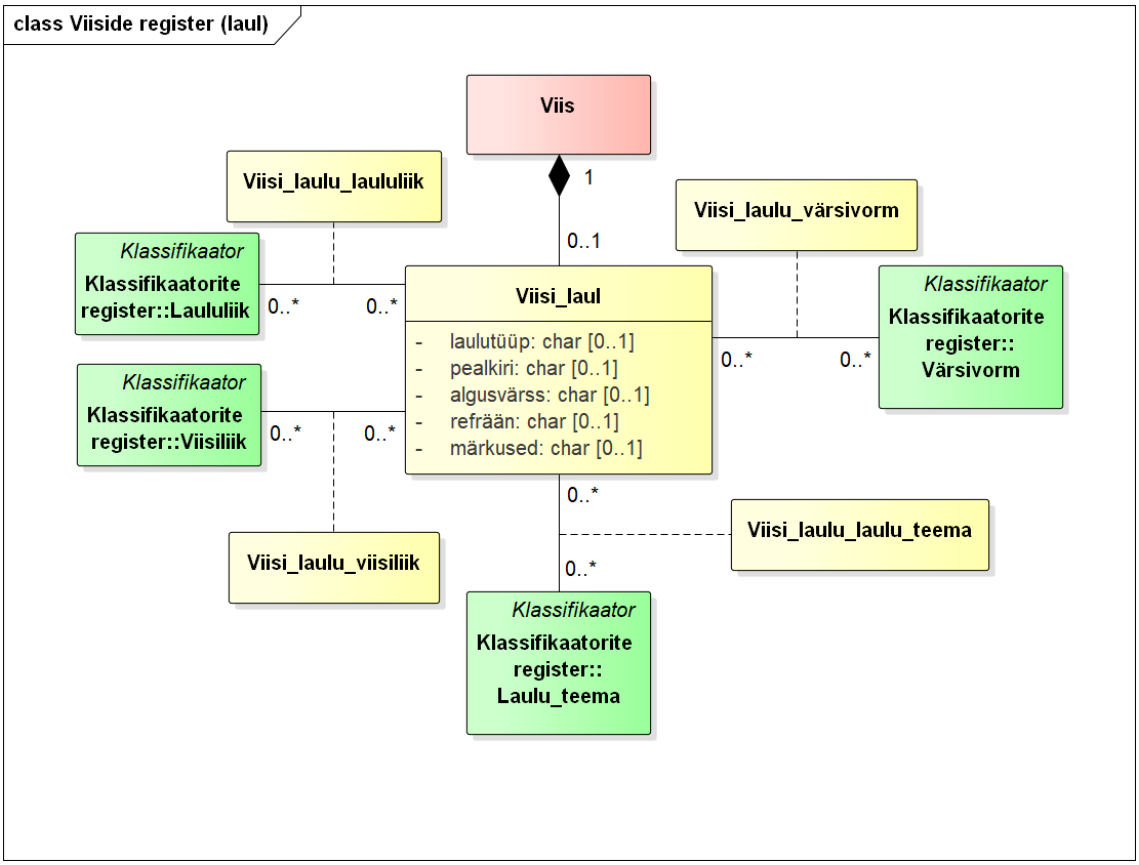
Joonis 37. Viiside registri olemi-suhte diagramm viiside kohta.



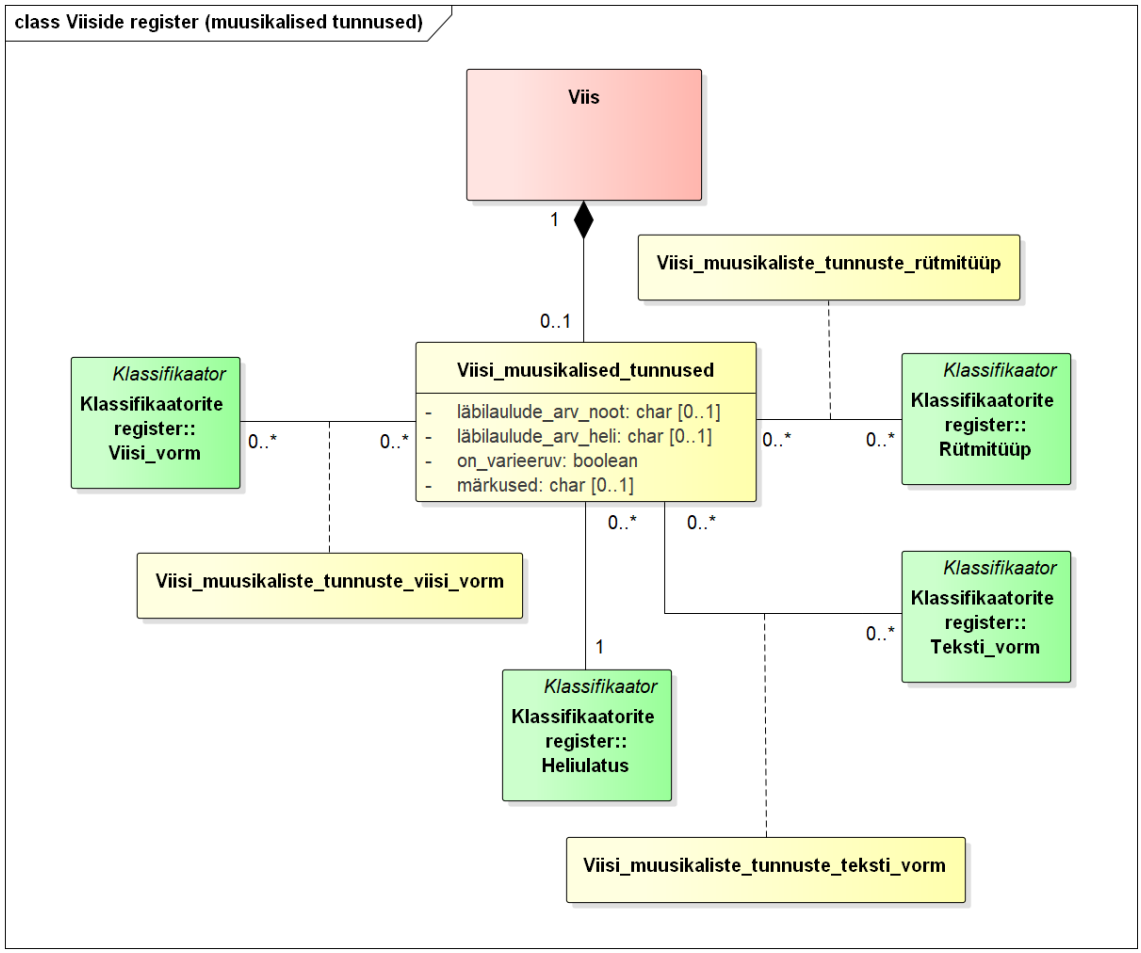
Joonis 38. Viiside registri olemi-suhte diagramm kohtade kohta.



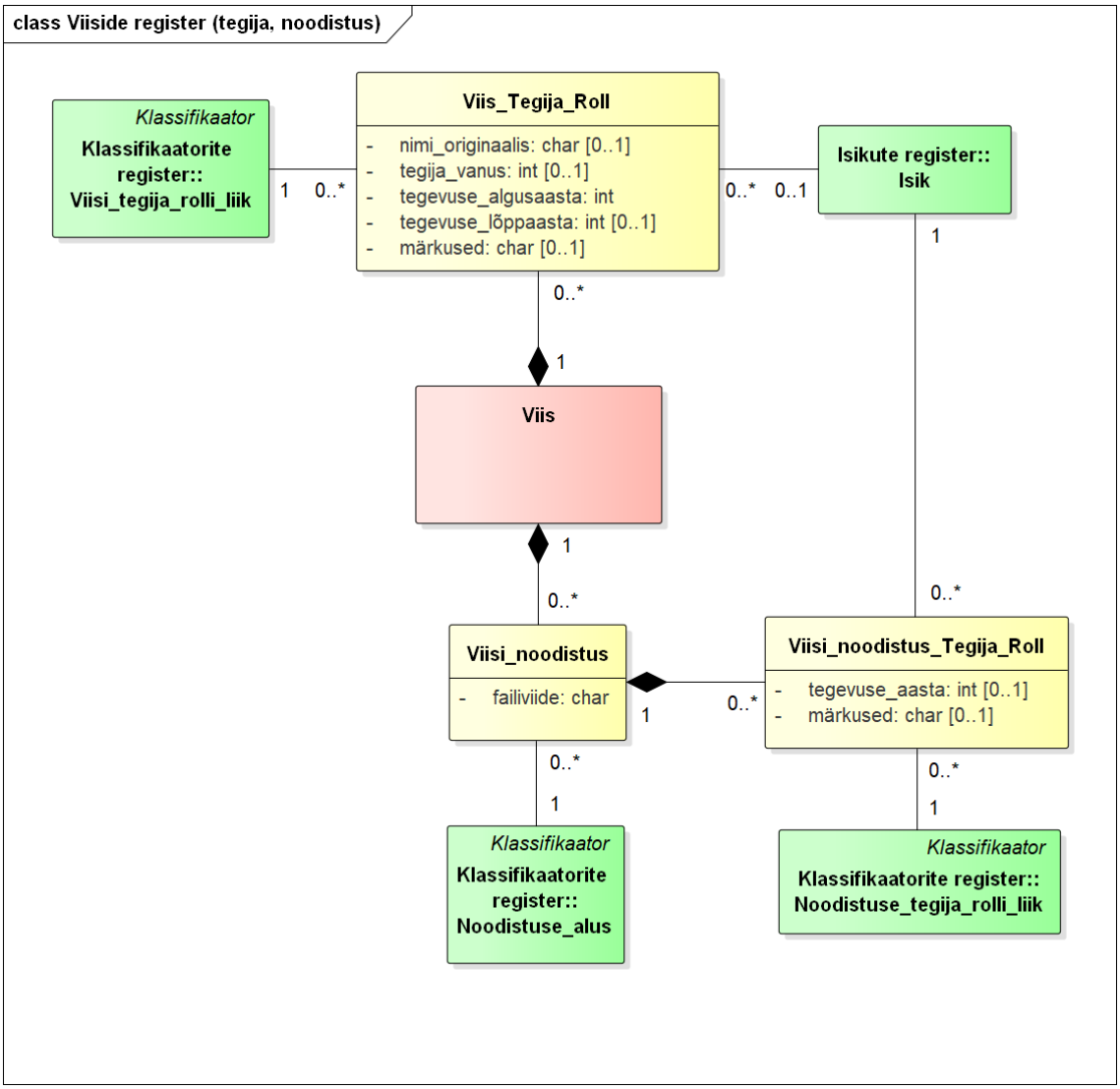
Joonis 39. Viiside registri olemi-suhte diagramm esituste kohta.



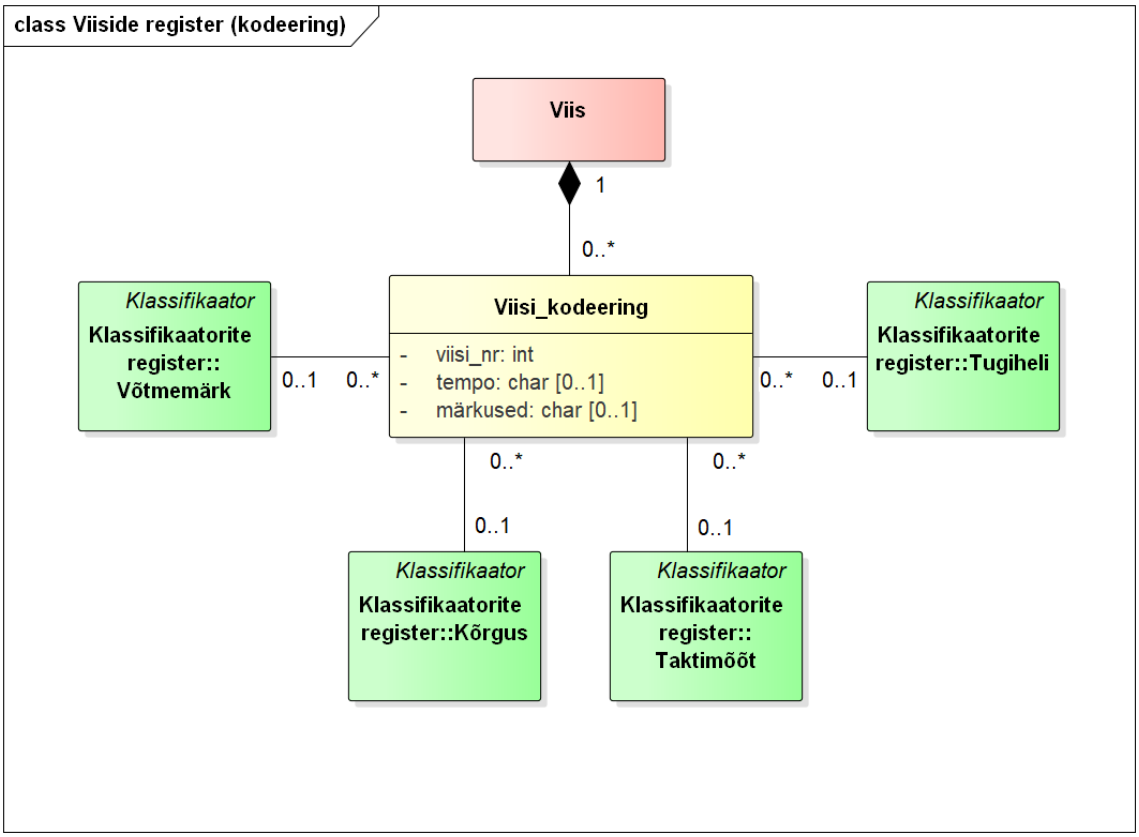
Joonis 40. Viiside registri olemi-suhte diagramm laulude kohta.



Joonis 41. Viiside registri olemi-suhte diagramm muusikaliste tunnuste kohta.



Joonis 42. Viiside registri olemi-suhte diagramm tegijate ja noodistuste kohta.



Joonis 43. Viiside registri olemi-suhte diagramm kodeeringute kohta.

Tabel 5 esitab viiside registri olemi-suhte diagrammidel esitatud olemitüüpide sõnalised kirjeldused.

Tabel 5. Viiside registri olemitüüpide sõnalised kirjeldused.

Olemitüübi nimi (inglise keeles)	**Kuuluvus registrisse**	**Definitsioon**
Viis (tunes)	Viiside register	Viis vastab ühe esituse jäädvustusele (võib olla jäädvustatud mitme eri meediumiga) või mitme esituse üldistatud jäädvustusele.
Viis_Tegija_Roll (tunes_persons_roles)	Viiside register	Viisiga seotud isikud ja tegevused.
Viisi_esitus (tune_performances)	Viiside register	Viisi kogumisel jäädvustatud ettekanne.
Viisi_esituse_ traditsioonilise_tegevuse_liik (tune_performances_ traditional_actions)	Viiside register	Viisi esituse ja traditsioonilise tegevuse liigi seos.

Olemitüübi nimi (inglise keeles)	**Kuuluvus registrisse**	**Definitsioon**
Viisi_kodeering (tune_encodings)	Viiside register	Noodikirja sümbolite tähistused.
Viisi_koht (tune_places)	Viiside register	Viisi esitaja elukoht või päritolukoht.
Viisi_laul (tune_songs)	Viiside register	Laul on rahvaviisi esitus koos tekstiga.
Viisi_laulu_laulu_teema (tune_songs_song_topics)	Viiside register	Viisi laulu ja laulu teema seos.
Viisi_laulu_laululiik (tune_songs_song_genres)	Viiside register	Viisi laulu ja laululiigi seos.
Viisi_laulu_viisiliik (tune_songs_tune_genres)	Viiside register	Viisi laulu ja viisiliigi seos.
Viisi_laulu_värsivorm (tune_songs_verse_forms)	Viiside register	Viisi laulu ja värsi vormi seos.
Viisi_muusikalised_tunnused (musical_characteristics)	Viiside register	Viisi muusikaga seotud omadused.
Viisi_muusikaliste_ tunnuste_rütmitüüp (musical_characteristics_ rhythm_types)	Viiside register	Viisi muusikaliste tunnuste ja rütmitüübi seos.
Viisi_muusikaliste_tunnuste_ teksti_vorm (musical_characteristics_ text_forms)	Viiside register	Viisi muusikaliste tunnuste ja teksti vormi seos.
Viisi_muusikaliste_tunnuste_ viisi_vorm (musical_characteristics_ tune_forms)	Viiside register	Viisi muusikaliste tunnuste ja viisi vormi seos.
Viisi_noodistus (tune_transcriptions)	Viiside register	Originaalse helisalvestuse transkriptsioon või käsikirja toimetatud versioon.
Viisi_noodistus_Tegija_Roll (transcriptions_persons_roles)	Viiside register	Viisi noodistusega seotud isikud ja tegevused.

Tabel 6 esitab viiside registri olemi-suhte diagrammidel esitatud atribuutide sõnalised kirjeldused.

Tabel 6. Viiside registri atribuutide sõnalised kirjeldused.

Olemitüübi nimi	**Atribuudi nimi (inglise keeles)**	**Definitsioon**	**Näiteväärtus**
Viis	viisiviide (tune_reference)	Arhiiviviide noodile {Vähemalt üks neljast – viisiviide või tekstiviide või heliviide või videoviide peab olema registreeritud. Viisiviide ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	ERA III 4, 11 (2)
Viis	tekstiviide (text_reference)	Arhiiviviide tekstile {Vähemalt üks neljast – viisiviide või tekstiviide või heliviide või videoviide peab olema registreeritud. Tekstiviide ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	ERA II 30, 275/9 (5)
Viis	heliviide (sound_reference)	Arhiiviviide helisalvestisele {Vähemalt üks neljast – viisiviide või tekstiviide või heliviide või videoviide peab olema registreeritud. Heliviide ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	ERA, Fon. 258 a
Viis	videoviide (video_reference)	Arhiiviviide videole {Vähemalt üks neljast – viisiviide või tekstiviide või heliviide või videoviide peab olema registreeritud. Videoviide ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	DV 1 (12)

Olemitüübi nimi	Atribuudi nimi (inglise keeles)	Definitsioon	Näiteväärtus
Viis	kartoteek (catalogue)	Viisi kartoteegikaardi number {Kartoteek ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	XIV 506
Viis	väljaanded (publications)	Trükis avaldatud teosed [49], kus viis on ära märgitud. {Väljaanded ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	VK XI, lk 330
Viis	märkused (remarks)	Vabas vormis kommentaarid viisi kohta. {Märkused ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	Lisasin teksti viite.
Viis	kontrollimise_aeg (verified)	Kuupäev, millal on toimunud viisi kontrollimine. {Kui kontrollimise aeg on registreeritud, siis peab ka kontrollija olema registreeritud. Kontrollimise aeg peab olema vahemikus 01.01.2000 ja 01.01.2200 (otspunktid kaasa arvatud).}	20.03.2020
Viis_Tegija_Roll	nimi_originaalis (name_origin)	Nimi, mis on märgitud arhiivi dokumendis. {Nimi originaalis ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	Morritson, Pärnust

Olemitüübi nimi	Atribuudi nimi (inglise keeles)	Definitsioon	Näiteväärtus
Viis_Tegija_Roll	tegija_vanus (person_age)	„Aeg (aastates), mille keegi või miski on käesoleva või mingi teatava hetkeni elanud või olemas olnud [49].“ {Vanus peab olema vahemikus 1 ja 150 (otspunktid kaasa arvatud).}	79
Viis_Tegija_Roll	tegevuse_ algusaasta (action_start_year)	Aasta, millal tegevust alustati. {Tegevuse algusaasta peab olema registreeritud. Tegevuse algusaasta peab olema vahemikus 0 ja 2200 (otspunktid kaasa arvatud).}	1880
Viis_Tegija_Roll	tegevuse_ lõppaasta (action_end_year)	Aasta, millal tegevus lõpetati. {Tegevuse lõppaasta peab olema vahemikus 0 ja 2200 (otspunktid kaasa arvatud). Tegevuse lõppaasta peab olema suurem kui tegevuse algusaasta.}	1890
Viis_Tegija_Roll	märkused (remarks)	Vabas vormis kommentaarid tegevuse ja tegija kohta. {Märkused ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	Olnud kuulus punutud mööbli meister ja oma kaubaga kaunis laialt ringi rännand.

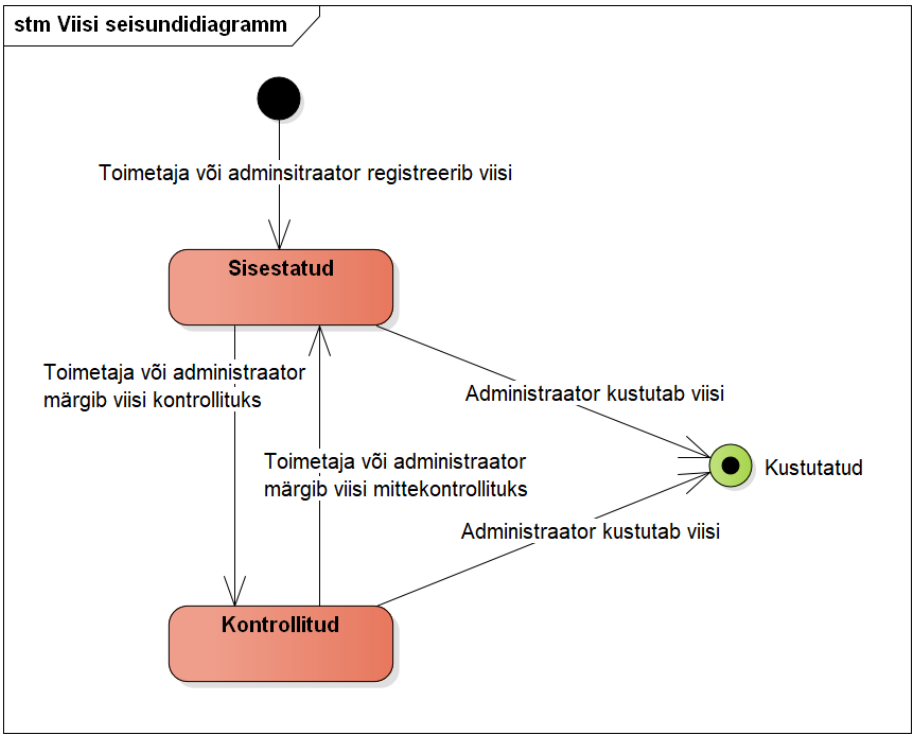
Olemitüübi nimi	**Atribuudi nimi (inglise keeles)**	**Definitsioon**	**Näiteväärtus**
Viisi_esitus	saade (accompaniment)	Laulu saatev instrumentaalne kaasmäng [49]. {Saade ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	moldpill
Viisi_esitus	märkused (remarks)	Vabas vormis kommentaarid esituse kohta. {Märkused ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	Väga ilus esitus, selge, noodist pikem, veidi erineb tekstist.
Viisi_kodeering	viisi_nr (tune_ encoding_num)	Järjekorranumber. {Viisi number on unikaalne ühe viisi lõikes. Peab olema registreeritud.}	1
Viisi_kodeering	tempo (tempo)	Muusika esitamise kiirus. {Tempo ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	Parajalt
Viisi_kodeering	märkused (remarks)	Vabas vormis kommentaarid kodeeringu kohta. {Märkused ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	alguses tempo: Parajas rutulises tempos. Refrääniks sama viis tempoga Ruttu.
Viisi_koht	muu_koht (other_place)	Mingi täpsem punkt või piirkond [49], mis täpsustab viisi esitaja elukohta või päritolukohta. {Muu koht ei tohi olla tühi string ja ainult	Kõutsi talu

Olemitüübi nimi	Atribuudi nimi (inglise keeles)	Definitsioon	Näiteväärtus
		tühimärkidest koosnev string.}	
Viisi_koht	märkused (remarks)	Vabas vormis kommentaarid koha kohta. {Märkused ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	Helis kuulen: Aru vallas.
Viisi_laul	laulutüüp (song_type)	Laulu tüübinimetus, mis tähistab enam-vähem jääva sisuga teksti või tekstiosa. {Laulutüüp ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	Heinast hobu
Viisi_laul	pealkiri (song_title)	Laulu nimetus, mis on märgitud kogumise andmete juurde. {Pealkiri ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	Lapse kiigutamise laul
Viisi_laul	algusvärss (first_verse)	Laulu esimene rütmiline üksus [49]. {Algusvärss ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	Tuleb see õnnis õhtuke
Viisi_laul	refrään (refrain)	Laulus iga salmi lõpus või järel (varieeritult) korduv osa [49]. {Refrään ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	Oh ale, oh ale, oh mis ale on see lugu!

Olemitüübi nimi	**Atribuudi nimi (inglise keeles)**	**Definitsioon**	**Näiteväärtus**
Viisi_laul	märkused (remarks)	Vabas vormis kommentaarid laulu kohta. {Märkused ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	Teksti juures märkus värsipaari erandliku kordamise kohta.
Viisi_ muusikalised_ tunnused	läbilaulude_arv_ noot (melostrophe_num_ score)	Näitab, mitmel korral on viisitervik noodis esitatud. {Läbilaulude arv ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	7 (AB) / 3 (ABA)
Viisi_ muusikalised_ tunnused	läbilaulude_arv_ heli (melostrophe_num_ audio)	Näitab, mitmel korral on viisitervik helisalvestisel esitatud. {Läbilaulude arv ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	5
Viisi_ muusikalised_ tunnused	on_varieeruv (is_variable)	Näitab, kas viisi muusikalised tunnused on varieeruvad. {Peab olema registreeritud.}	FALSE
Viisi_ muusikalised_ tunnused	märkused (remarks)	Vabas vormis kommentaarid muusikaliste tunnuste kohta. {Märkused ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	Noodistuses üks varieeruv noot, helis mitmekesisem varieerumine, murtud värsse ka.
Viisi_noodistus	failiviide (file_reference)	Noodistuse faili nimetus. {Failiviide ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	fon_306_f.jpg

Olemitüübi nimi	Atribuudi nimi (inglise keeles)	Definitsioon	Näiteväärtus
Viisi_noodistus_ Tegija_Roll	tegevuse_aasta (action_year)	Aasta, millal tegevus tehti. {Tegevuse aasta peab olema vahemikus 0 ja 2200 (otspunktid kaasa arvatud).}	1912
Viisi_noodistus_ Tegija_Roll	märkused (remarks)	Vabas vormis kommentaarid tegevuse ja tegija kohta. {Märkused ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	Noodistuse juurde on originaalis märgitud vale helisalvestuse number.

Joonis 44 esitab viiside registri põhiobjekti *Viis* võimalikud seisundid.

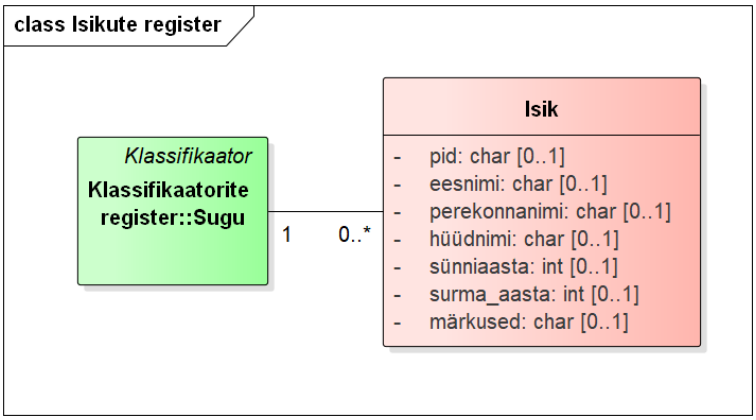


Joonis 44. Viisi seisundidiagramm.

Lisa 6 – Isikute registri analüüs

Joonis 45 esitatud olemi-suhte diagrammil on värvidel järgmine tähendus.

- Punasega on tähistatud registri põhiobjekt.
- Rohelisega on tähistatud teistesse registritesse kuuluvad objektid, mida on vaja isikute allsüsteemi toimimiseks.



Joonis 45. Isikute registri olemi-suhte diagramm.

Tabel 7 esitab isikute registri olemi-suhte diagrammil esitatud olemitüüpide sõnalised kirjeldused.

Tabel 7. Isikute registri olemitüüpide sõnalised kirjeldused.

Olemitüübi nimi (inglise keeles)	**Kuuluvus registrisse**	**Definitsioon**
Isik (persons)	Isikute register	Viisi ja viisi toimetamisega seotud füüsiline isik.

Tabel 8 esitab isikute registri olemi-suhte diagrammil esitatud atribuutide sõnalised kirjeldused.

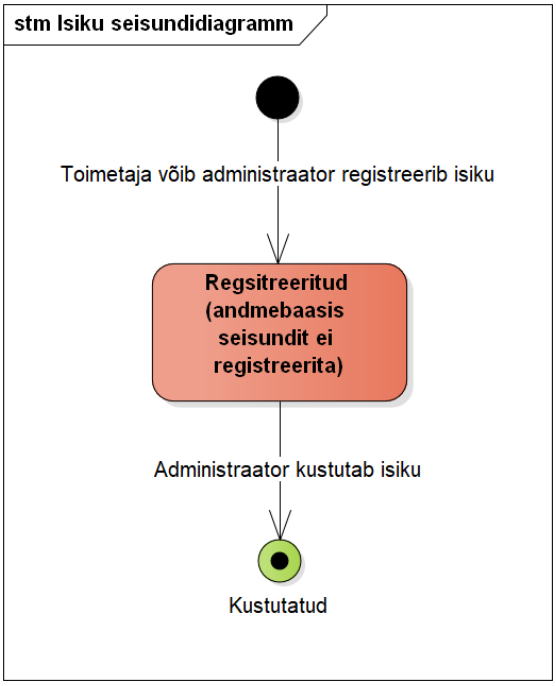
Tabel 8. Isikute registri atribuutide sõnalised kirjeldused.

Olemitüübi nimi	**Atribuudi nimi (inglise keeles)**	**Definitsioon**	**Näiteväärtus**
Isik	pid (pid)	Isiku unikaalne identifikaator Kivikese allsüsteemis. {PID on unikaalne. PID ei tohi	KM-13105-62389-15920

Olemitüübi nimi	Atribuudi nimi (inglise keeles)	Definitsioon	Näiteväärtus
		olla tühi string ja ainult tühimärkidest koosnev string.}	
Isik	eesnimi (given_name)	„Lapsele tema sünni registreerimisel antav nimi (eesti keeles perekonnanime ees) [49].“ {Vähemalt üks kolmest – eesnimi või perekonnanimi või hüüdnimi peab olema registreeritud. Eesnimi ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	Karl
Isik	perekonnanimi (surname)	„Vanemailt lapsele kanduv või abiellumise teel saadav nimi (eesti keeles eesnime järel) [49].“ {Vähemalt üks kolmest – eesnimi või perekonnanimi või hüüdnimi peab olema registreeritud. Perekonnanimi ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	Kask
Isik	hüüdnimi (nickname)	„Igapäevases kõnes tarvitatav isiku mitteametlik nimi, mis on talle külge jäänud tema välimuse, tegevuse või iseloomu järgi või kujunenud tema nime moonutusel [49].“ {Vähemalt üks kolmest – eesnimi või perekonnanimi või hüüdnimi peab olema registreeritud. Hüüdnimi ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	Tälle Madli
Isik	sünniaasta (birth_year)	Aasta, millal isik on sündinud. {Sünniaasta peab olema vahemikus 0 ja 2200 (otspunktid kaasa arvatud).}	1845

Olemitüübi nimi	Atribuudi nimi (inglise keeles)	Definitsioon	Näiteväärtus
Isik	surma_aasta (death_year)	Aasta, millal isik on surnud. {Surma-aasta peab olema vahemikus 0 ja 2200 (otspunktid kaasa arvatud). Surma-aasta ei saa olla väiksem kui sünniaasta.}	1912
Isik	märkused (remarks)	Vabas vormis kommentaarid isiku kohta. {Märkused ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	preester

Joonis 46 esitab isikute registri põhiobjekti *Isik* võimalikud seisundid.

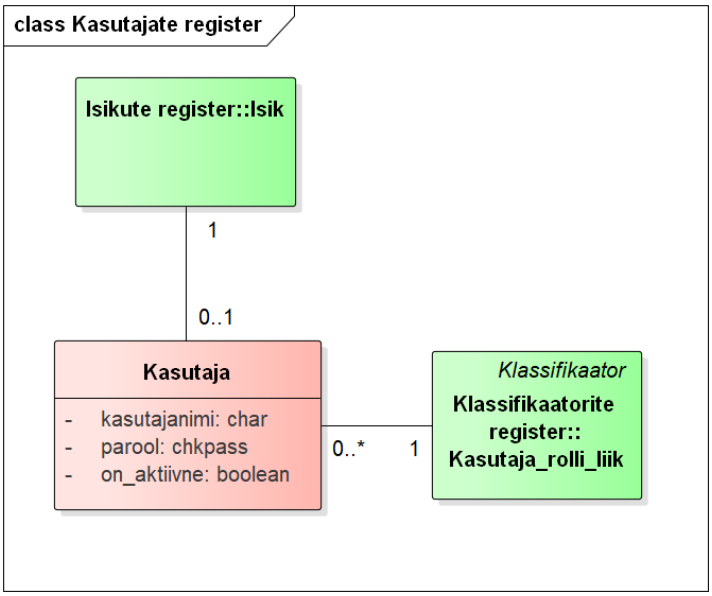


Joonis 46. Isiku seisundidiagramm.

Lisa 7 – Kasutajate registri analüüs

Joonis 47 esitatud olemi-suhte diagrammil on värvidel järgmine tähendus.

- Punasega on tähistatud registri põhiobjekt.
- Rohelisega on tähistatud teistesse registritesse kuuluvad objektid, mida on vaja kasutajate allsüsteemi toimimiseks.



Joonis 47. Kasutajate registri olemi-suhte diagramm.

Tabel 9 esitab kasutajate registri olemi-suhte diagrammil esitatud olemitüüpide sõnalised kirjeldused.

Tabel 9. Kasutajate registri olemitüüpide sõnalised kirjeldused.

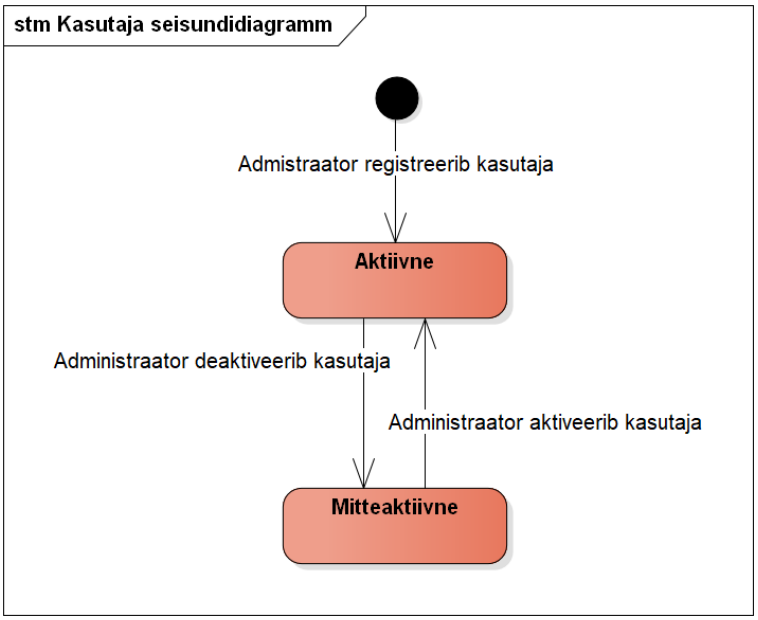
Olemitüübi nimi (inglise keeles)	**Kuuluvus registrisse**	**Definitsioon**
Kasutaja (users)	Isikute register	„See, kes kasutab mingit (arvuti)süsteemi, teenust vm vahendit, kes selle abil midagi teeb või sellest abi saab [49].“

Tabel 10 esitab kasutajate registri olemi-suhte diagrammidel esitatud atribuutide sõnalised kirjeldused.

Tabel 10. Kasutajate registri atribuutide sõnalised kirjeldused.

Olemitüübi nimi	**Atribuudi nimi (inglise keeles)**	**Definitsioon**	**Näiteväärtus**
Kasutaja	kasutajanimi (username)	„Arvutisüsteemile, andmebaasile vm ressursile juurdepääsu tagav nimi [49].“ {Kasutajanimi (tõstutundetu) on unikaalne. Kasutajanimi peab olema registreeritud. Kasutajanimi ei tohi sisaldada tühikuid ega tohi olla ainult tühimärkidest koosnev string.}	kasutaja@test.ee
Kasutaja	parool (password)	„Turvalisust tagav sõna vm märgijada, mis lubab süsteemi siseneda [49].“ Andmebaasi salvestatakse parooli ja soola põhjal leitud räsiväärtus. {Parool peab olema registreeritud. Parool ei tohi olla ainult tühimärkidest koosnev string.}	$2y$10$zEXJBsFKXPQbIKaSUzKcn.YDCiRNaWAAPjiPGgC2.ZyxHMjdGxCPS
Kasutaja	on_aktiivne (is_active)	Näitab, kas kasutaja on aktiivne ja tohib süsteemi siseneda. {Peab olema registreeritud. Süsteemis peab olema vähemalt üks aktiivne administraatori rolliga kasutaja.}	TRUE

Joonis 48 esitab Kasutajate registri põhiobjekti *Kasutaja* võimalikud seisundid.

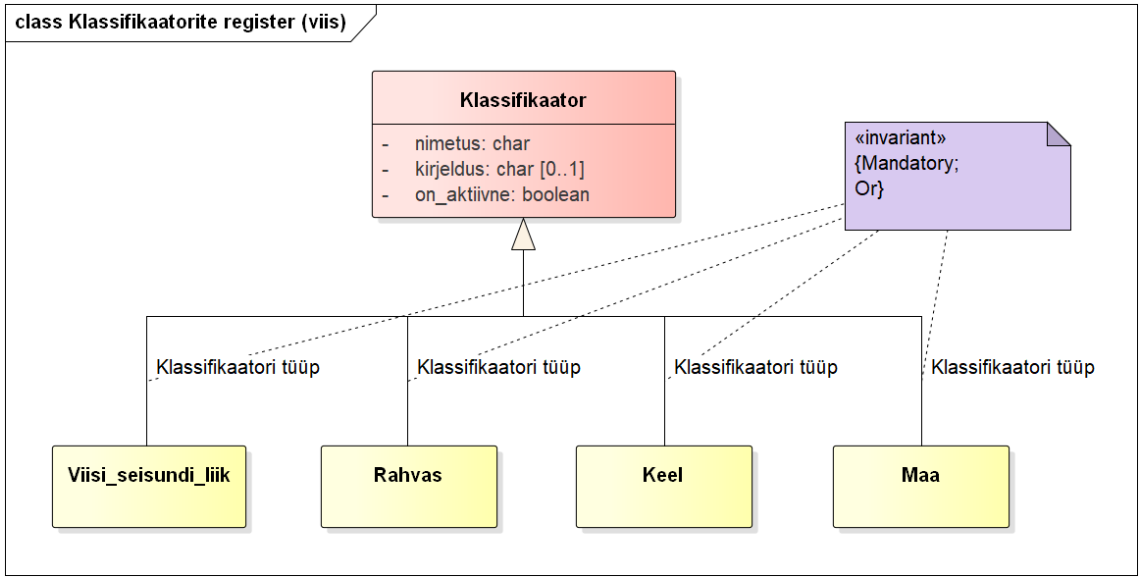


Joonis 48. Kasutaja seisundidiagramm.

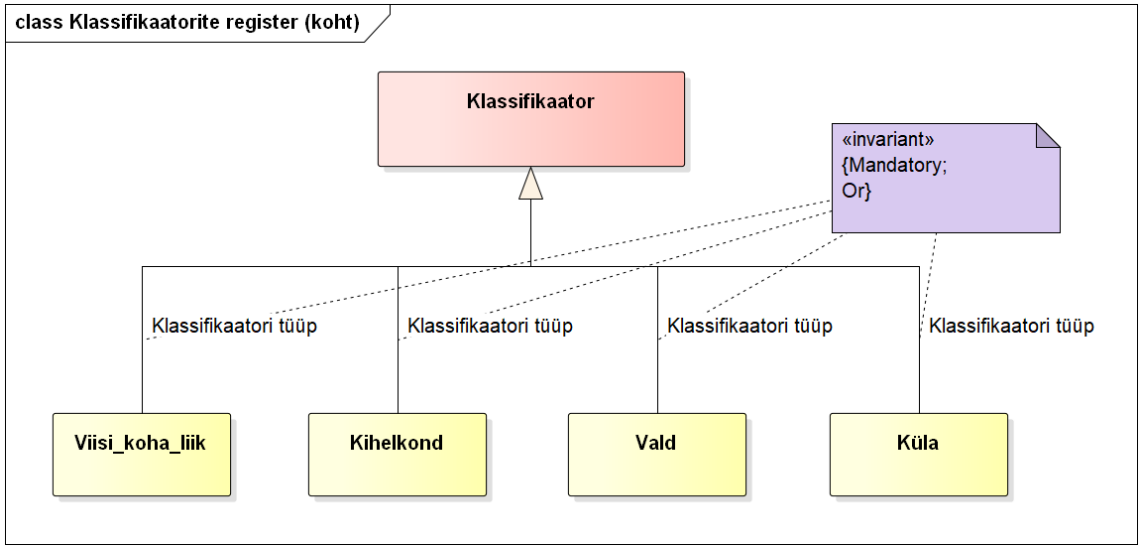
Lisa 8 – Klassifikaatorite registri analüüs

Joonis 49 – Joonis 56 esitatud olemi-suhte diagrammidel on värvidel järgmine tähendus.

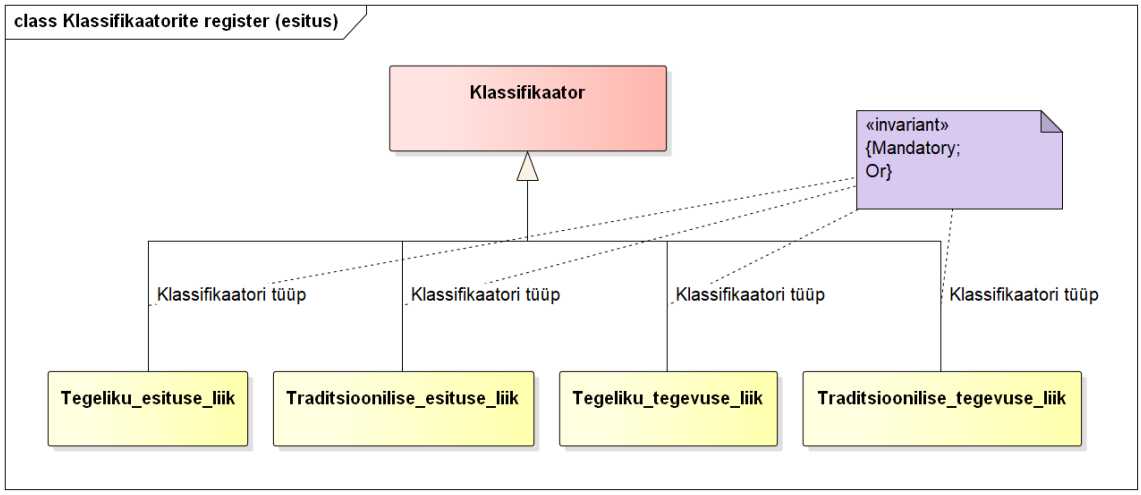
- Punasega on tähistatud registri põhiobjekt.
- Kollasega on tähistatud registrisse kuuluvad alamobjektid (mitte-põhiobjektid).



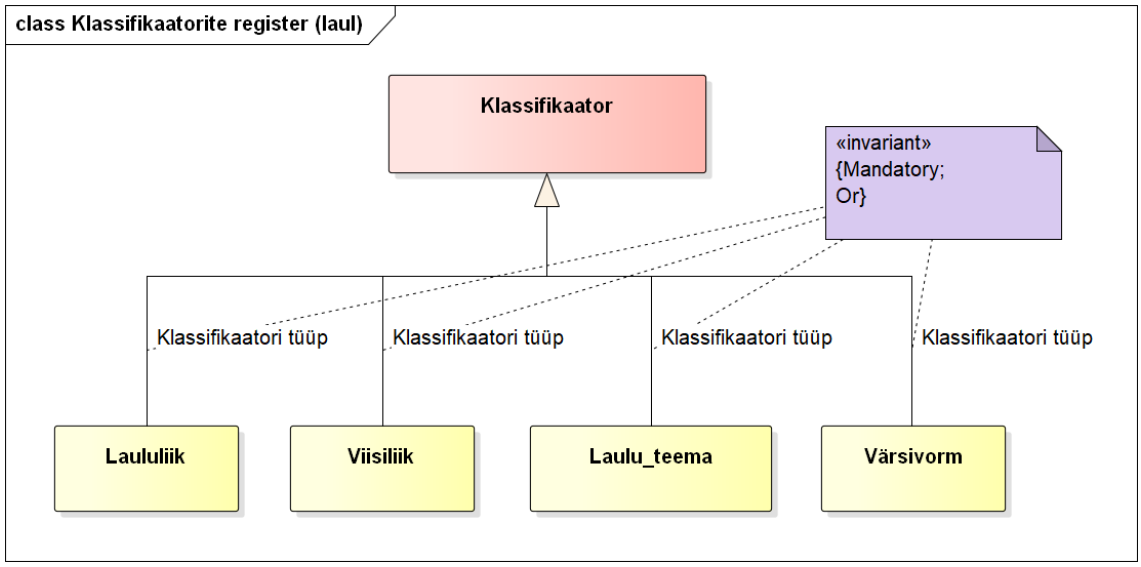
Joonis 49. Klassifikaatorite registri olemi-suhte diagramm viiside klassifikaatorite kohta.



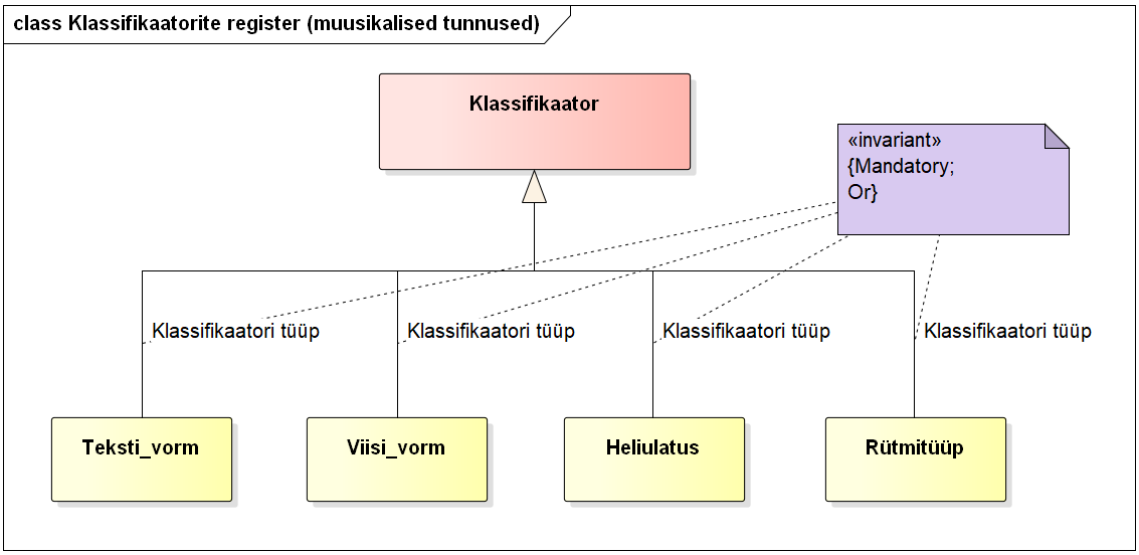
Joonis 50. Klassifikaatorite registri olemi-suhte diagramm kohtade klassifikaatorite kohta.



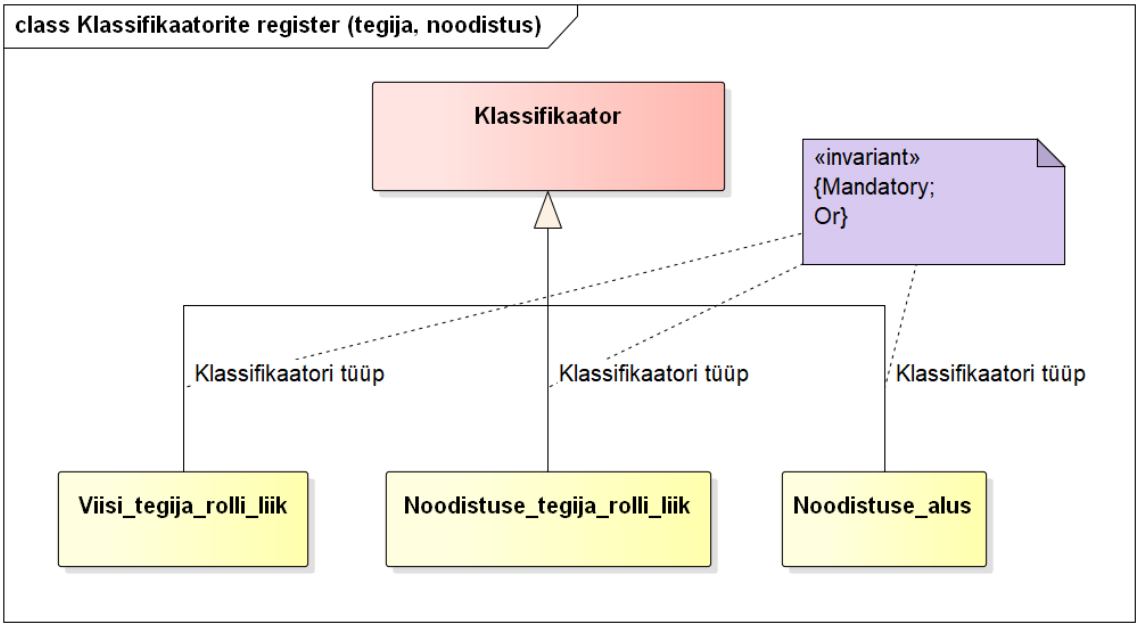
Joonis 51. Klassifikaatorite registri olemi-suhte diagramm esituste klassifikaatorite kohta.



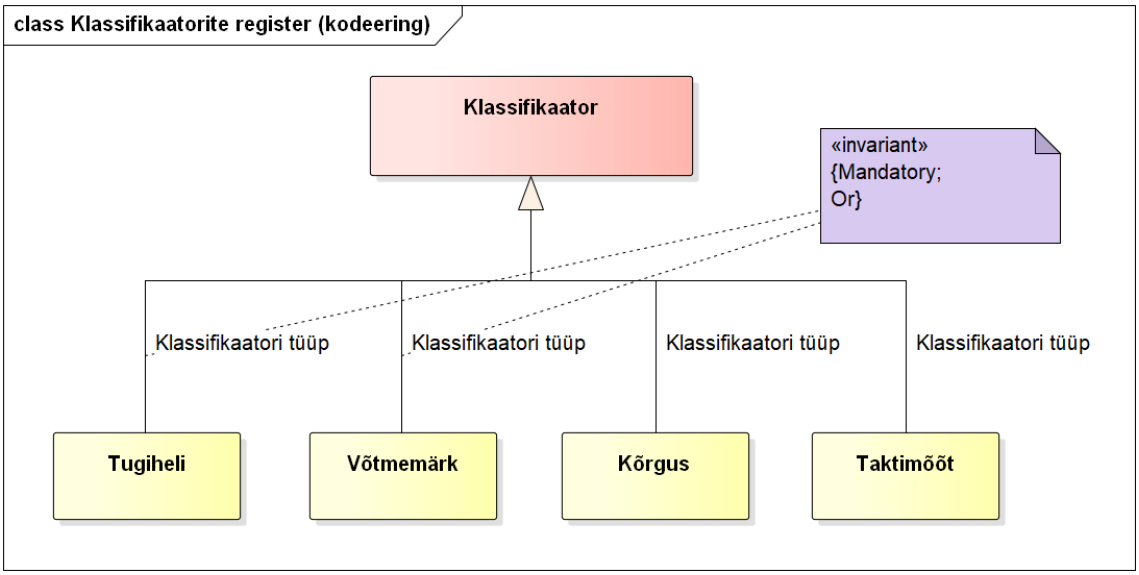
Joonis 52. Klassifikaatorite registri olemi-suhte diagramm laulude klassifikaatorite kohta.



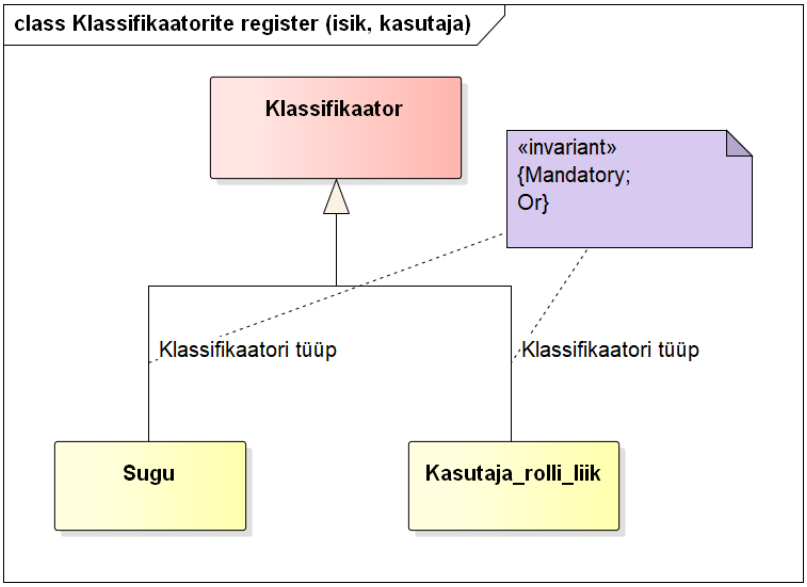
Joonis 53. Klassifikaatorite registri olemi-suhte diagramm muusikaliste tunnuste klassifikaatorite kohta.



Joonis 54. Klassifikaatorite registri olemi-suhte diagramm tegijate ja noodistuste klassifikaatorite kohta.



Joonis 55. Klassifikaatorite registri olemi-suhte diagramm kodeeringute klassifikaatorite kohta.



Joonis 56. Klassifikaatorite registri olemi-suhte diagramm isikute ja kasutajate klassifikaatorite kohta.

Tabel 11 esitab klassifikaatorite registri olemi-suhte diagrammidel esitatud olemitüüpide sõnalised kirjeldused.

Tabel 11. Klassifikaatorite registri olemitüüpide sõnalised kirjeldused.

Olemitüübi nimi (inglise keeles)	**Kuuluvus registrisse**	**Definitsioon**
Heliulatus (sound_ranges)	Klassifikaatorite register	Vahe viisi alumisest helist ülemiseni, tähistatud traditsiooniliste intervallide numbritega. Näiteks: v3, v3-4.

Olemitüübi nimi (inglise keeles)	**Kuuluvus registrisse**	**Definitsioon**
Kasutaja_rolli_ liik (user_role_types)	Klassifikaatorite register	Näitab, millises rollis kasutaja tegutseb ja millised õigused rollist tulenevad. Näiteks: administraator, toimetaja.
Keel (languages)	Klassifikaatorite register	„Inimese olulisim suhtlemisvahend, mis mõtete ja tunnete väljendamiseks kasutab sõnu ja väljendeid ning mida hoiab koos teatav struktuur ehk grammatika [49].“ Näiteks: eesti.
Kihelkond (parishes)	Klassifikaatorite register	„Maakiriku koguduse piirkond (Eestis 13. sajandist kuni aastani 1925); ajaloolis-etnograafiliste (ka keeleliste) iseärasustega ala ka hiljem [49].“ Näiteks: Kuusalu, Tarvastu.
Klassifikaator	Klassifikaatorite register	„Täpselt kirjeldatud, üksteist välistava tähisega kategooria, mida teatavas andmekogus kasutatakse millegi liigitamiseks või rühmadeks jaotamiseks [49].“
Kõrgus (pitches)	Klassifikaatorite register	Tugiheli absoluutkõrgus (tähtnimega) algsel laulmisel. Näiteks: g1=a, g1=c.
Küla (villages)	Klassifikaatorite register	„Väike maa-asula, mille moodustavad üksteisega lähestikku asuvad talud või hooned [49].“ Näiteks: Uusküla, Eigla.
Laulu_teema (song_topics)	Klassifikaatorite register	Laulu peamine sisu. Liigitus on kahetasemeline. Näiteks: töö->lõikus, kal->jõulud.
Laululiik (song_genres)	Klassifikaatorite register	Laulu liik funktsionaal-temaatilises liigituses. Liigitus on kahetasemeline. Näiteks: kal->vastla, kal->jõulu.
Maa (countries)	Klassifikaatorite register	Suurem territoriaalne üksus (riik), kus viis koguti. Näiteks: Eesti, Läti.
Noodistuse_alus (transcription_ sources)	Klassifikaatorite register	Näitab, mille alusel noodistus on tehtud. Näiteks: heli, video.
Noodistuse_ tegija_rolli_liik (transcription_person_ role_types)	Klassifikaatorite register	Näitab, millise tegevuse tegijana isik noodistuse juures osales. Näiteks: noodistaja, noodigraafik.
Rahvas (nations)	Klassifikaatorite register	Ühtse keelelise ja kultuurilise identiteediga rahvas [49]. Näiteks: eesti, vene.

Olemitüübi nimi (inglise keeles)	**Kuuluvus registrisse**	**Definitsioon**
Rütmitüüp (rhythm_types)	Klassifikaatorite register	Näitab üldistatult laulu rütmi. Näiteks: 1, 5.
Sugu (sexes)	Klassifikaatorite register	Kumbki neist kahest rühmast, kelleks isendid sugurakkude ja -elundite kahetaolisuse põhjal jagunevad [49]. Näiteks: mees, naine.
Taktimõõt (measures)	Klassifikaatorite register	„Kahe kohakuti numbriga tähistatav struktuur, mis määrab takti rõhuliste ja rõhutute osade arvu ja vältuse [49].“ Näiteks: 3/8, 4/8.
Tegeliku_ esituse_liik (actual_ performance_types)	Klassifikaatorite register	Mitu lauljat ja kuidas oli nende laulmine korraldatud esitamise ajal – viisi kogumissituatsioonis. Näiteks: Ü, E+Ü.
Tegeliku_ tegevuse_liik (actual_action_types)	Klassifikaatorite register	Näitab, kuidas viisi kogumisel laulu esitus oli seotud tegevusega. Näiteks: istudes, hällitades.
Teksti_vorm (text_forms)	Klassifikaatorite register	Näitab, kuidas viisi ja teksti laulmisel kombineeriti. Näiteks: AA, AB.
Traditsioonilise_ esituse_liik (traditional_ performance_types)	Klassifikaatorite register	Mitu lauljat ja kuidas oli nende laulmine korraldatud esitamise ajal – traditsiooniliselt. Näiteks: R, Ka.
Traditsioonilise_ tegevuse_liik (traditional_ action_types)	Klassifikaatorite register	Näitab, kuidas traditsiooniliselt laulu esitus oli seotud tegevusega. Näiteks: tantsides, hüpitades (last).
Tugiheli (support_sounds)	Klassifikaatorite register	Tugiheli on nagu põhitoon, tavaliselt alumine ja sagedane heli, sellel põhineb viiside võrdlemine ja analüüs. Näiteks: a, b.
Viisi_koha_liik (tune_place_types)	Klassifikaatorite register	Kohta iseloomustav liigitus. Näiteks: esitaja elukoht.
Viisi_seisundi_liik (tune_states)	Klassifikaatorite register	Viisi hetkeseisundi iseloomustus. Näiteks: sisestatud, kontrollitud.
Viisi_tegija_ rolli_liik (tune_person_ role_types)	Klassifikaatorite register	Näitab, millise tegevuse tegijana isik viisi juures osales. Näiteks: viisikoguja, esitaja.

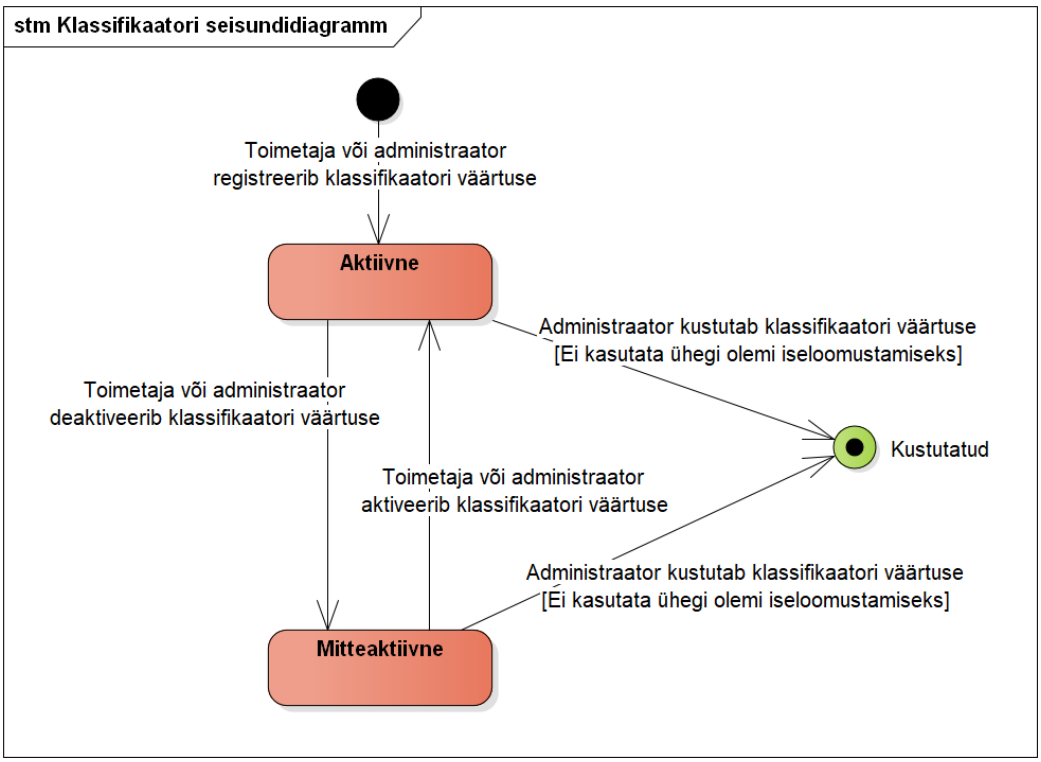
Olemitüübi nimi (inglise keeles)	**Kuuluvus registrisse**	**Definitsioon**
Viisi_vorm (tune_forms)	Klassifikaatorite register	Kirjeldab viisi struktuursete osade (viisiridade, refräänide) arvu ja asetust. Näiteks: 2, 4.
Viisiliik (tune_genres)	Klassifikaatorite register	Viisi liik, mis kogumisel viisi kohta öeldi. Liigitus on kahetasemeline. Näiteks: töö->karja, töö->lõikus.
Võtmemärk (key_signatures)	Klassifikaatorite register	„Noodivõtme järel paiknev alteratsioonimärk, mis määrab helistiku ja kehtib kogu helitöö ulatuses [49].“ Näiteks: #f, bh.
Värsivorm (verse_forms)	Klassifikaatorite register	Näitab värsiehitust, see on olemuslik tunnus, mis teeb vahet sellistel suurtel rühmadel nagu regilaulud (runo) ja uuemad lõppriimilised laulud (riim).

Tabel 12 esitab klassifikaatorite registri olemi-suhte diagrammidel esitatud atribuutide sõnalised kirjeldused.

Tabel 12. Klassifikaatorite registri atribuutide sõnalised kirjeldused.

Olemitüübi nimi	**Atribuudi nimi (inglise keeles)**	**Definitsioon**	**Näiteväärtus**
Klassifikaator	nimetus (title)	Tähistab klassifikaatori väärtust. {Nimetus (tõstutundetu) on ühe klassifikaatori piires unikaalne, v.a hierarhilise klassifikaatori korral, kus nimetus on unikaalne vastava taseme piires. Nimetus peab olema registreeritud. Nimetus ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	saun
Klassifikaator	kirjeldus (description)	Klassifikaatori väärtuse selgitus. {Kirjeldus ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	saun, leil, saunas käimine, vihtlemine
Klassifikaator	on_aktiivne (is_active)	Näitab, kas klassifikaatori väärtus on kasutatav või mitte. Kui väärtus on tõene, siis saab seda objekti lisamise ja muutmise juures kasutada.	TRUE

Joonis 57 esitab klassifikaatorite registri põhiobjekti *Klassifikaator* võimalikud seisundid.

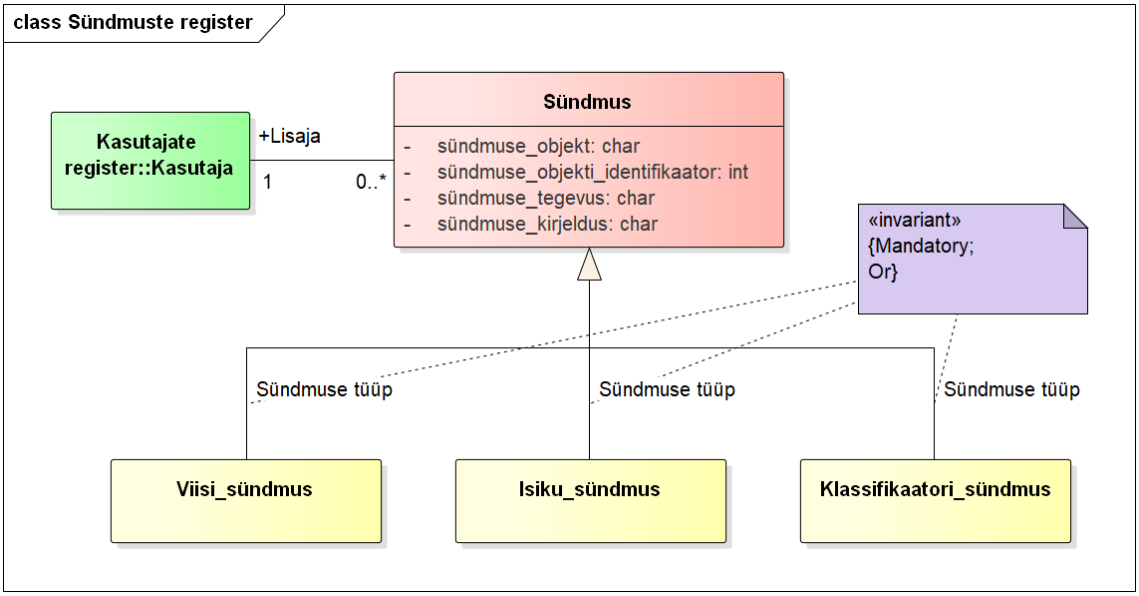


Joonis 57. Klassifikaatori seisundidiagramm.

Lisa 9 – Sündmuste registri analüüs

Joonis 58 esitatud olemi-suhte diagrammil on värvidel järgmine tähendus.

- Punasega on tähistatud registri põhiobjekt.
- Kollasega on tähistatud registrisse kuuluvad alamobjektid (mitte-põhiobjektid).
- Rohelisega on tähistatud teistesse registritesse kuuluvad objektid, mida on vaja sündmuste allsüsteemi toimimiseks.



Joonis 58. Sündmuste registri olemi-suhte diagramm.

Tabel 13 esitab sündmuste registri olemi-suhte diagrammil esitatud olemitüüpide sõnalised kirjeldused.

Tabel 13. Sündmuste registri olemitüüpide sõnalised kirjeldused.

Olemitüübi nimi (inglise keeles)	**Kuuluvus registrisse**	**Definitsioon**
Isiku_sündmus (person_events)	Sündmuste register	Midagi, mis on süsteemis juhtunud isiku või kasutaja andmetega.
Klassifikaatori_ sündmus (classifier_events)	Sündmuste register	Midagi, mis on süsteemis juhtunud klassifikaatori andmetega.
Sündmus	Sündmuste register	Juhtum süsteemis seoses põhiobjektide andmetega, mille talletamist peetakse oluliseks. Nendeks juhtumiteks on andmete lisamine, muutmine ja kustutamine.

Olemitüübi nimi (inglise keeles)	Kuuluvus registrisse	Definitsioon
Viisi_sündmus (tune_events)	Sündmuste register	Midagi, mis on süsteemis juhtunud viisi või mõne alamobjekti andmetega.

Tabel 14 esitab sündmuste registri olemi-suhte diagrammil esitatud atribuutide sõnalised kirjeldused.

Tabel 14. Sündmuste registri atribuutide sõnalised kirjeldused.

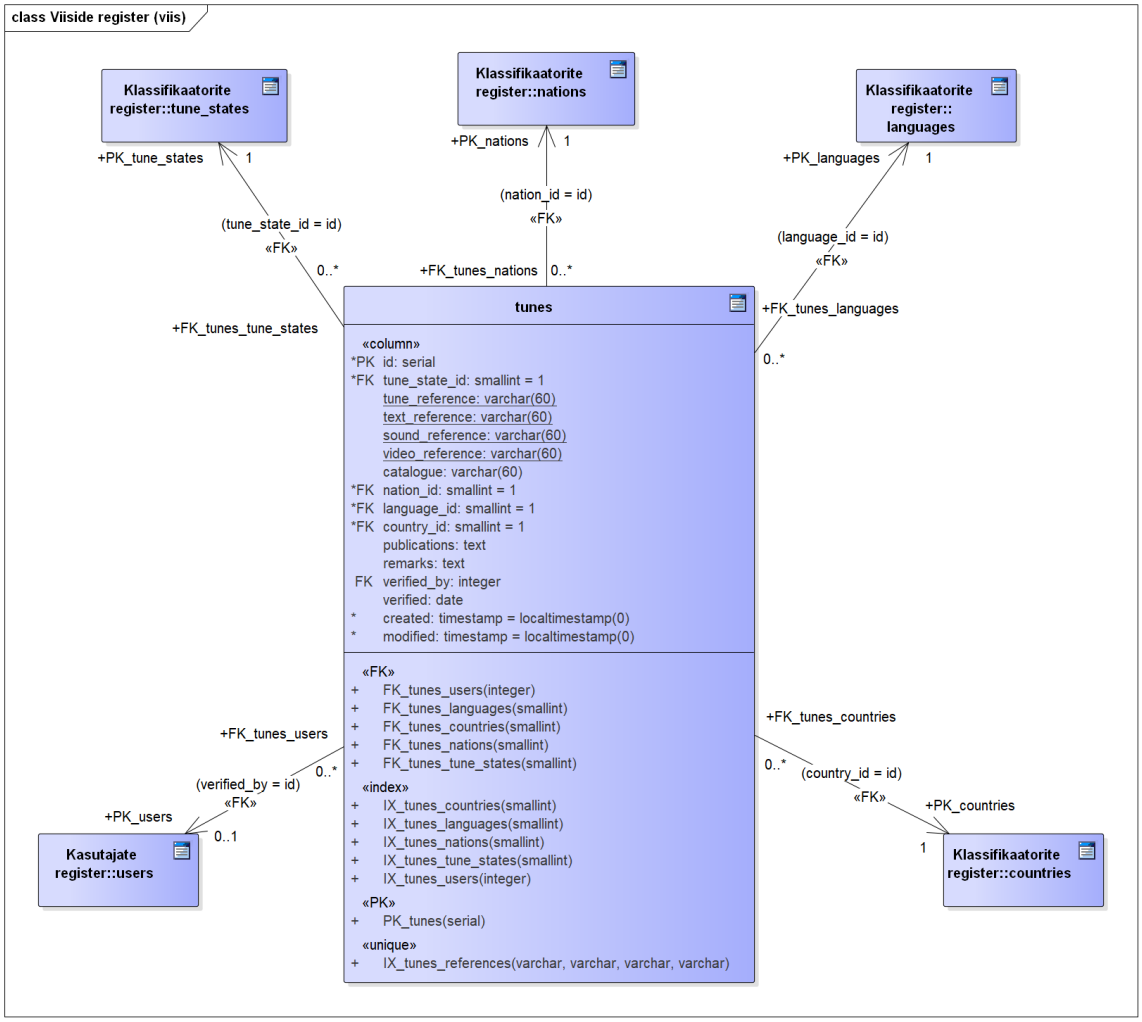
Olemitüübi nimi	Atribuudi nimi (inglise keeles)	Definitsioon	Näiteväärtus
Sündmus	sündmuse_ objektitüüp (event_object)	Sündmuse poolt mõjutatud objektitüübi nimetus. {Sündmuse objektitüüp peab olema registreeritud. Sündmuse objektitüüp ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	Viisid
Sündmus	sündmuse_objekti_ identifikaator (event_object_ ident)	Sündmuse poolt mõjutatud objekti identifikaator andmebaasis. Kui sündmuseks oli objekti kustutamine, siis viitab identifikaator objektile, mida andmebaasis enam ei eksisteeri. {Sündmuse objekti identifikaator peab olema registreeritud.}	3
Sündmus	sündmuse_tegevus (event_action)	Tegevus, mida sündmuse poolt mõjutatud objektiga tehti. {Sündmuse tegevus peab olema registreeritud. Sündmuse tegevus ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	Muuda
Sündmus	sündmuse_kirjeldus (description)	Kirjeldab sündmuse poolt mõjutatud objekti andmeid, mis tegevuse käigus lisandusid, muutusid või kustusid. Andmete lisamise korral on	kataloog: vana väärtus: tühi; uus väärtus: „XI 192“

Olemitüübi nimi	Atribuudi nimi (inglise keeles)	Definitsioon	Näiteväärtus
		kirjelduses objekti kõikide atribuutide nimed koos väärtustega. Andmete muutmise korral on kirjelduses objekti atribuutide nimed koos vanade ja uute väärtustega. Andmete kustutamise korral on kirjelduses objekti kõikide atribuutide nimed koos väärtustega. {Sündmuse kirjeldus peab olema registreeritud. Sündmuse kirjeldus ei tohi olla tühi string ja ainult tühimärkidest koosnev string.}	

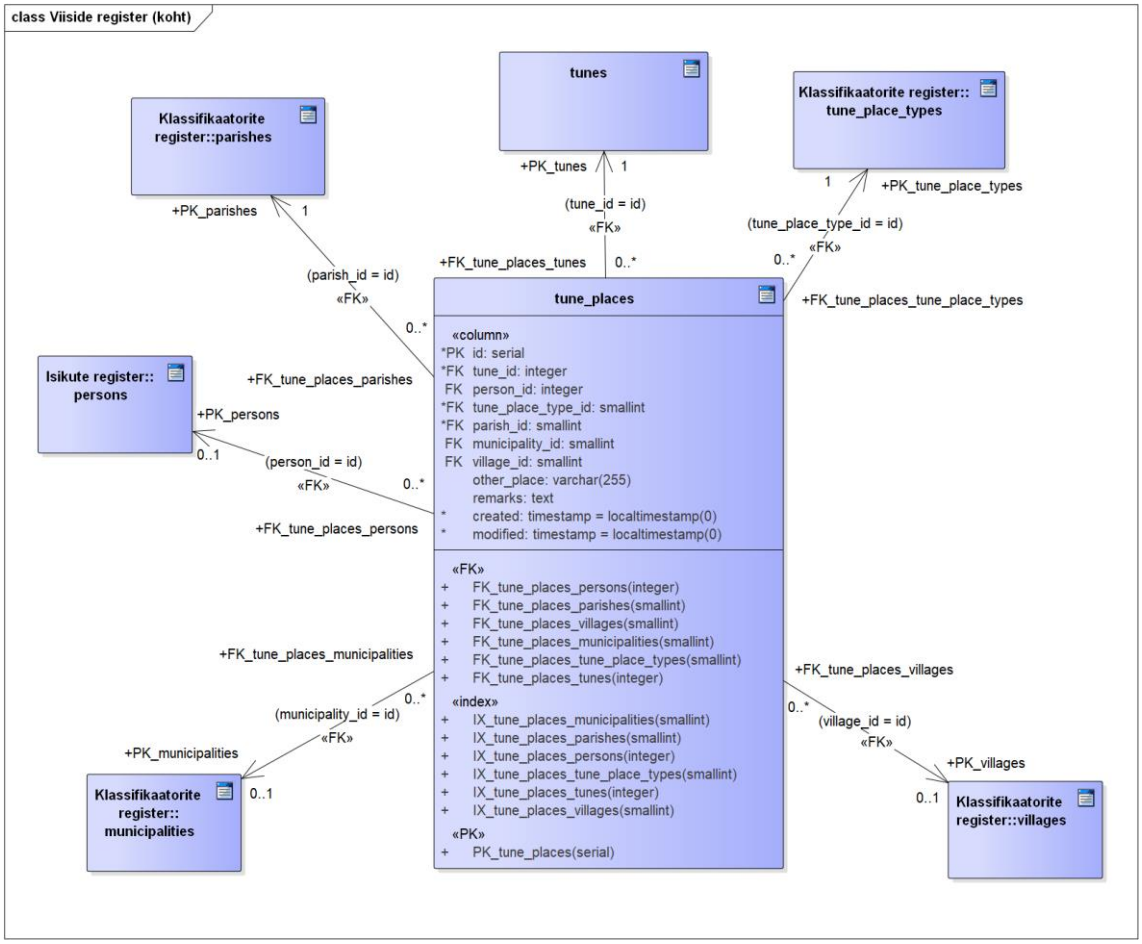
Sündmuste registri põhiobjektil *Sündmus* pole erinevaid infosüsteemile huvipakkuvaid seisundeid ja seega seisundidiagrammi ei esitata.

Lisa 10 – Viiside registri disain

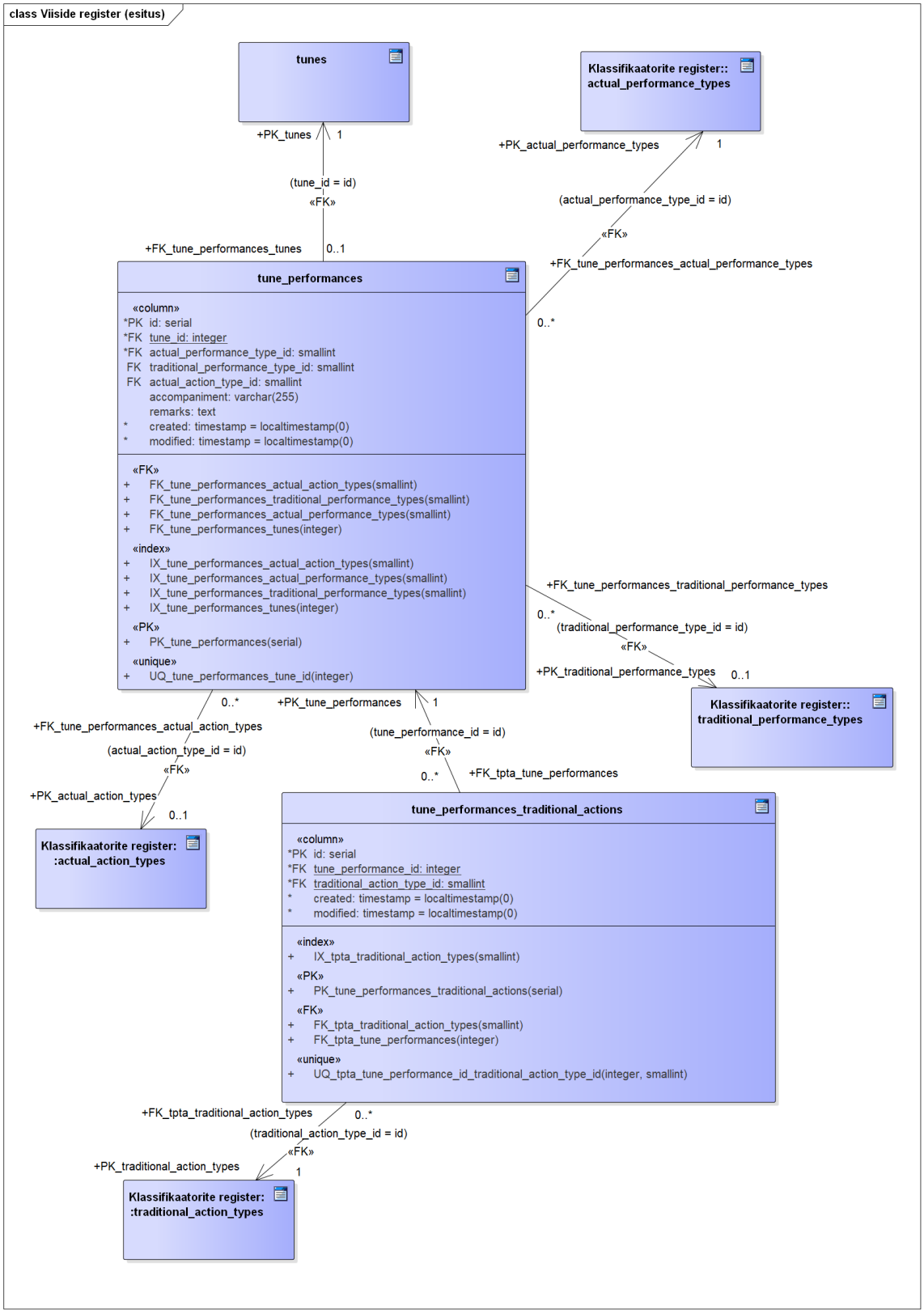
Joonis 59 – Joonis 68 esitavad viiside registri füüsilise disaini.



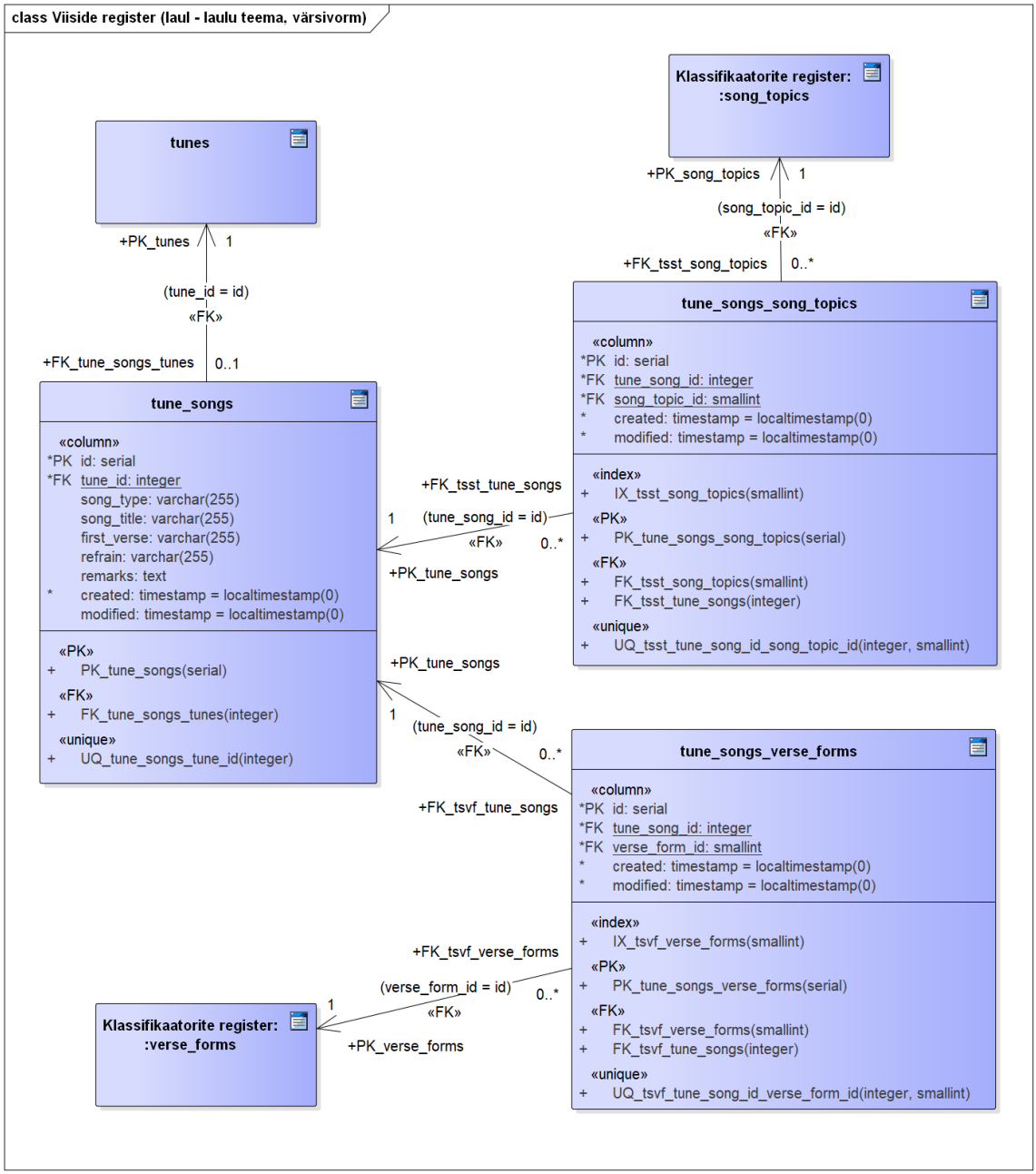
Joonis 59. Viiside registri füüsilise disaini andmebaasi diagramm viiside kohta.



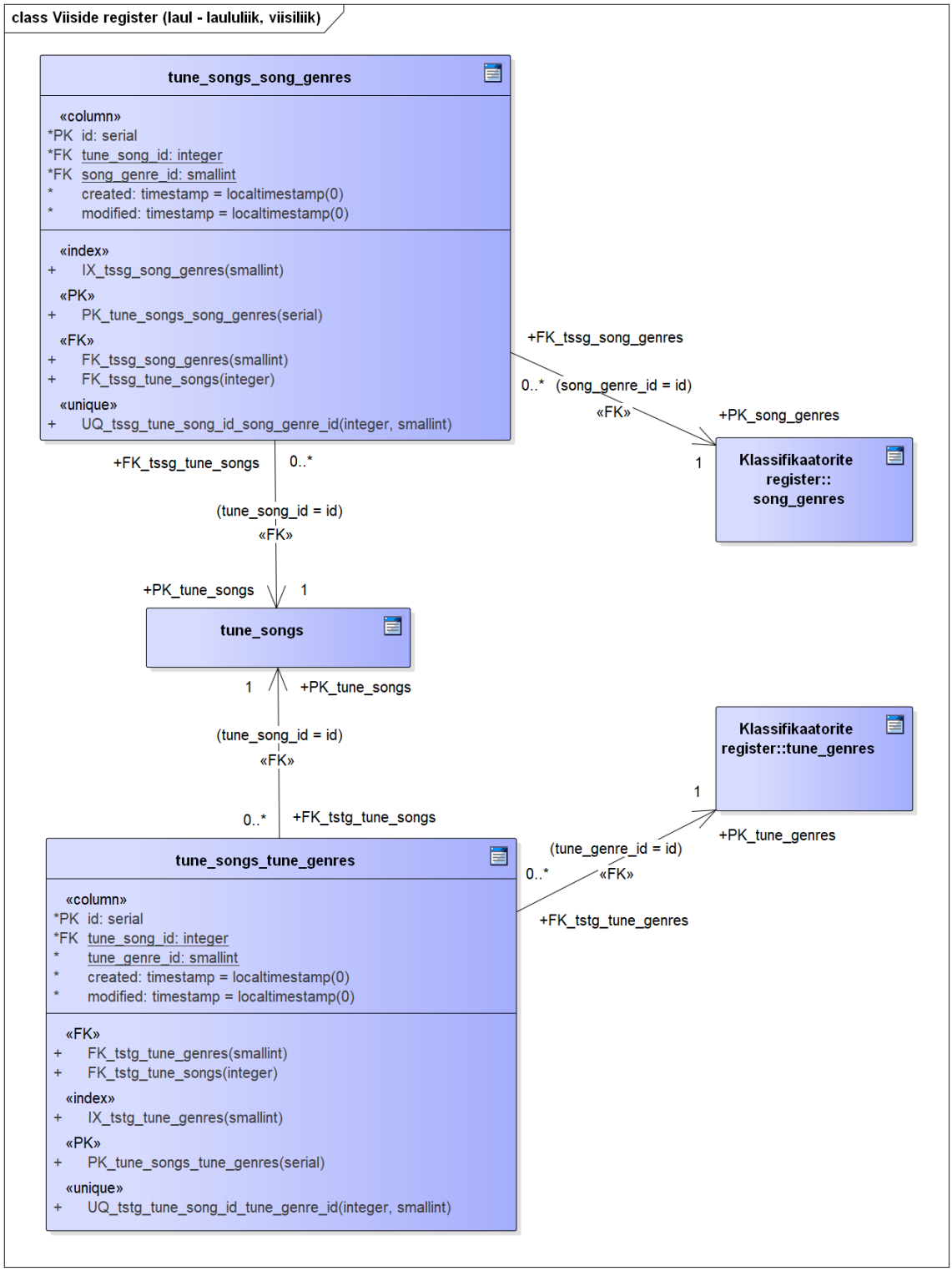
Joonis 60. Viiside registri füüsilise disaini andmebaasi diagramm kohtade kohta.



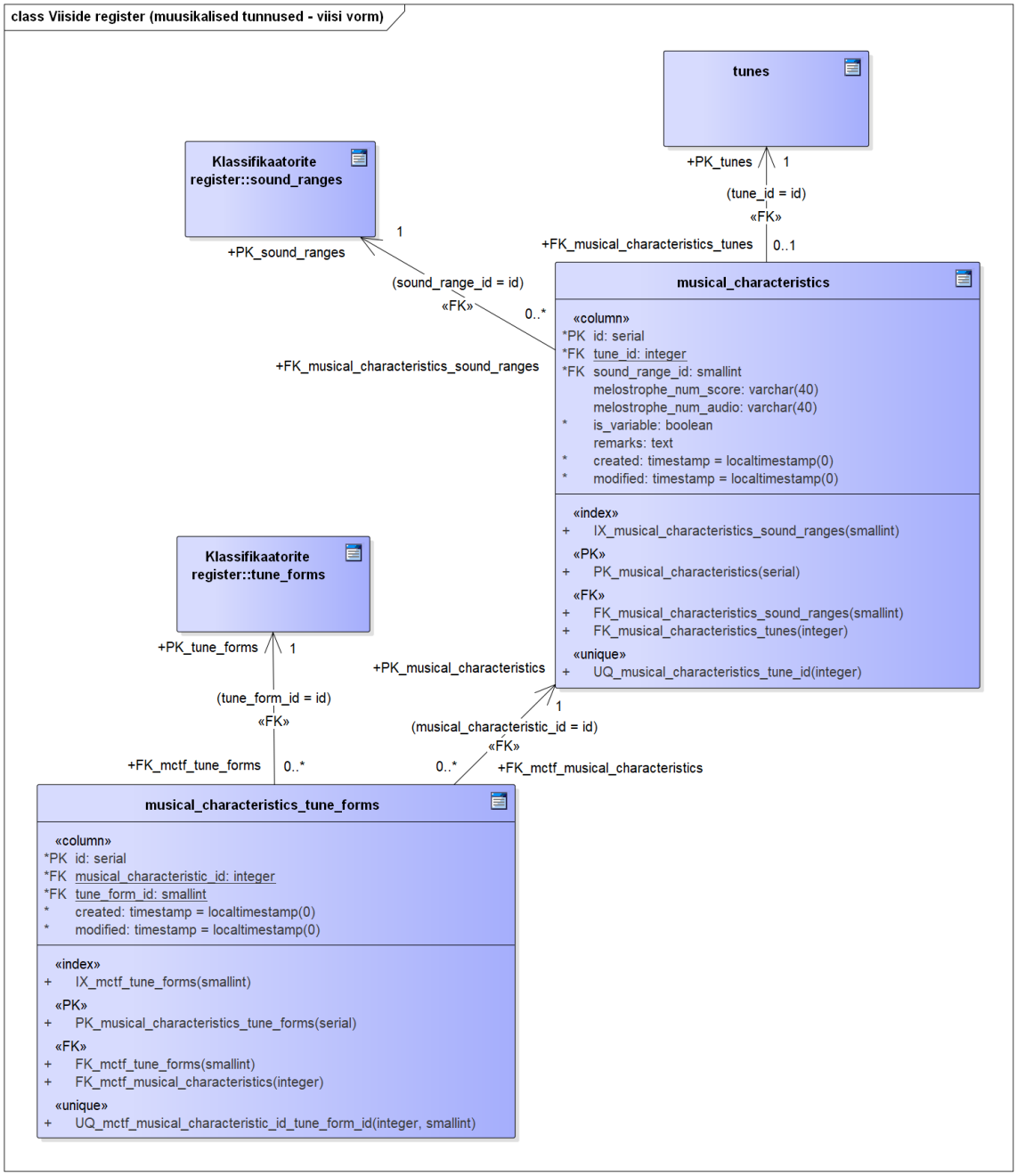
Joonis 61. Viiside registri füüsilise disaini andmebaasi diagramm esituste kohta.



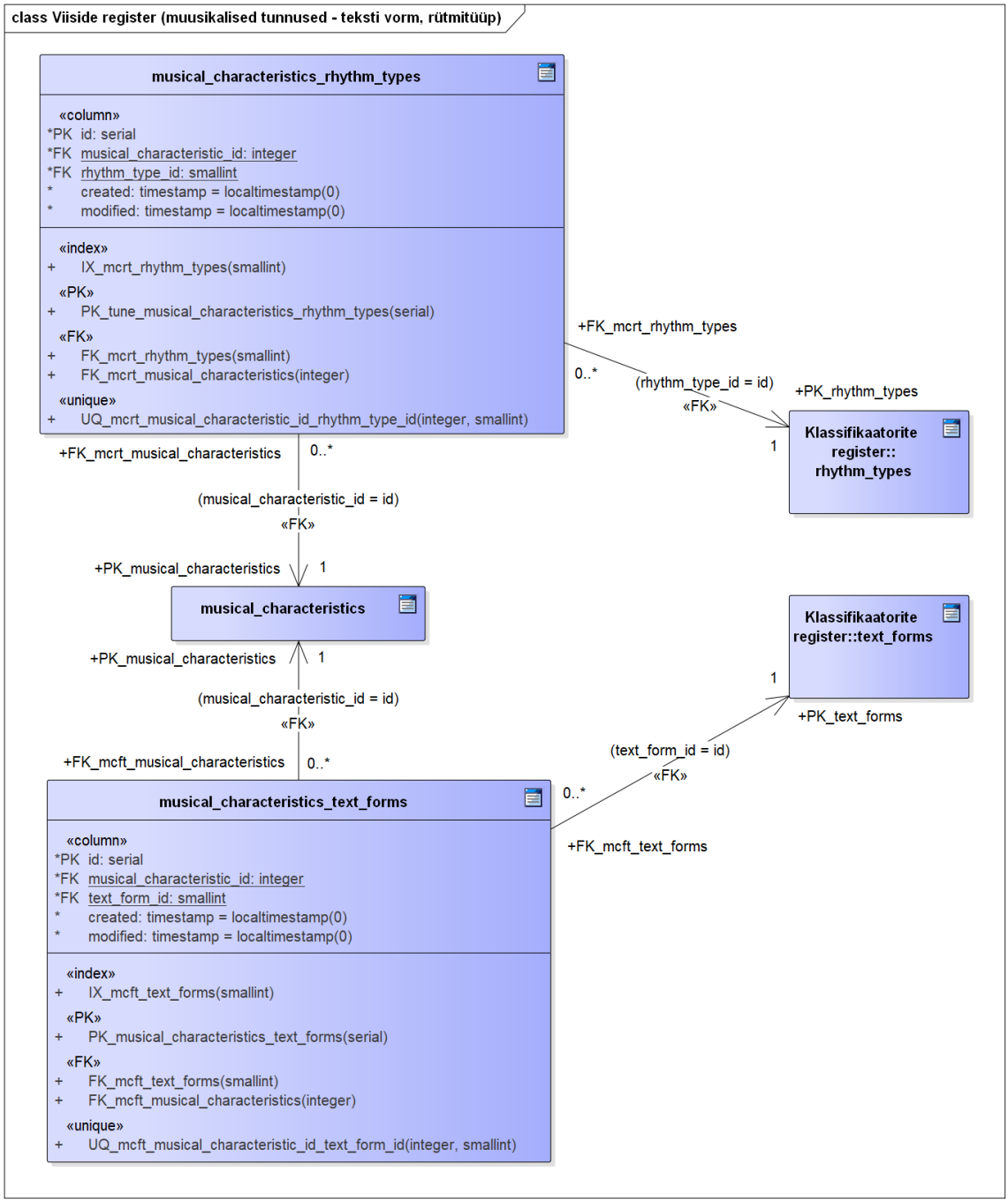
Joonis 62. Viiside registri füüsilise disaini andmebaasi diagramm laulude (laulu teema, värsivorm) kohta.



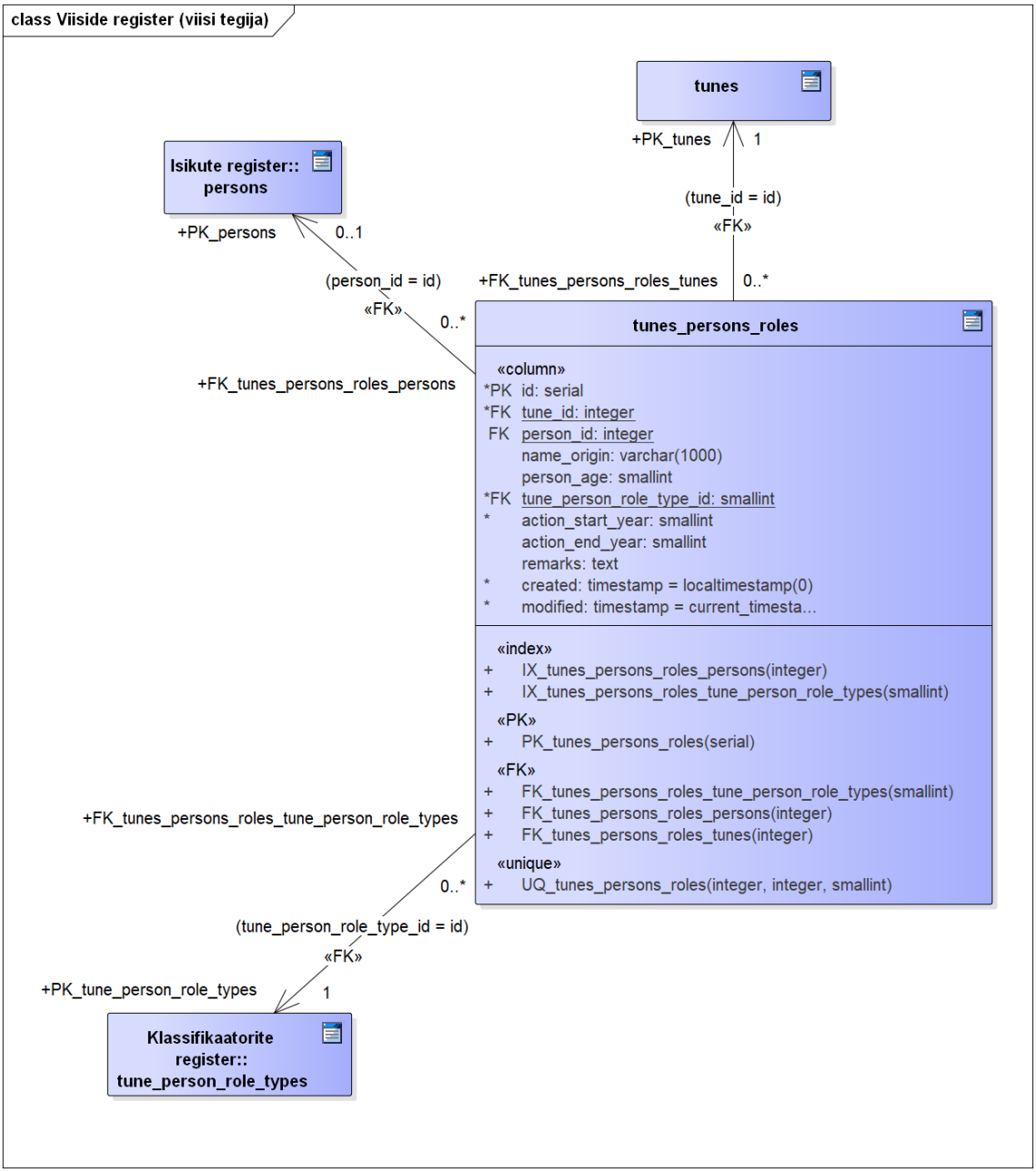
Joonis 63. Viiside registri füüsilise disaini andmebaasi diagramm laulude (laululiik, viisiliik) kohta.



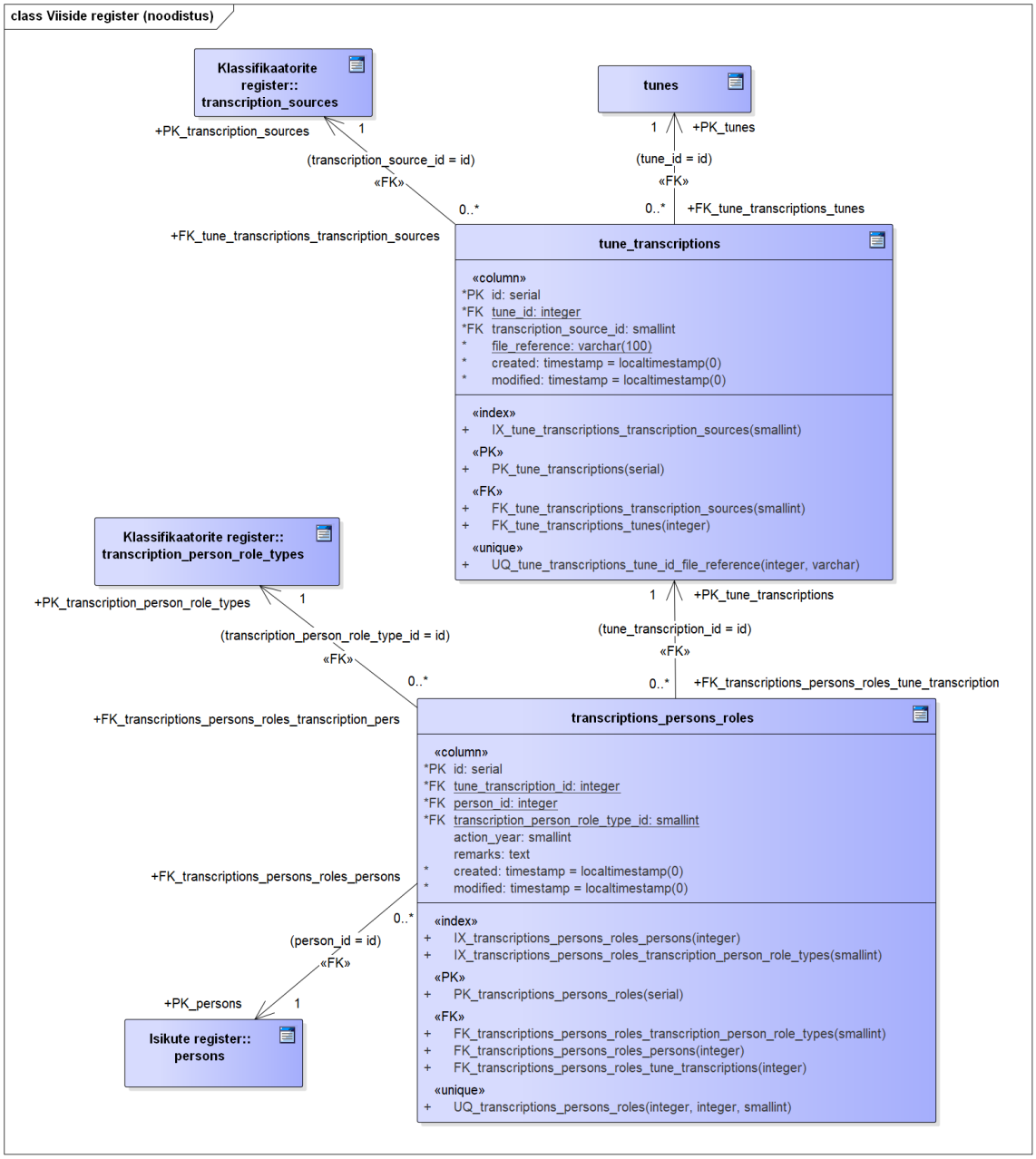
Joonis 64. Viiside registri füüsilise disaini andmebaasi diagramm muusikaliste tunnuste (viisi vorm) kohta.



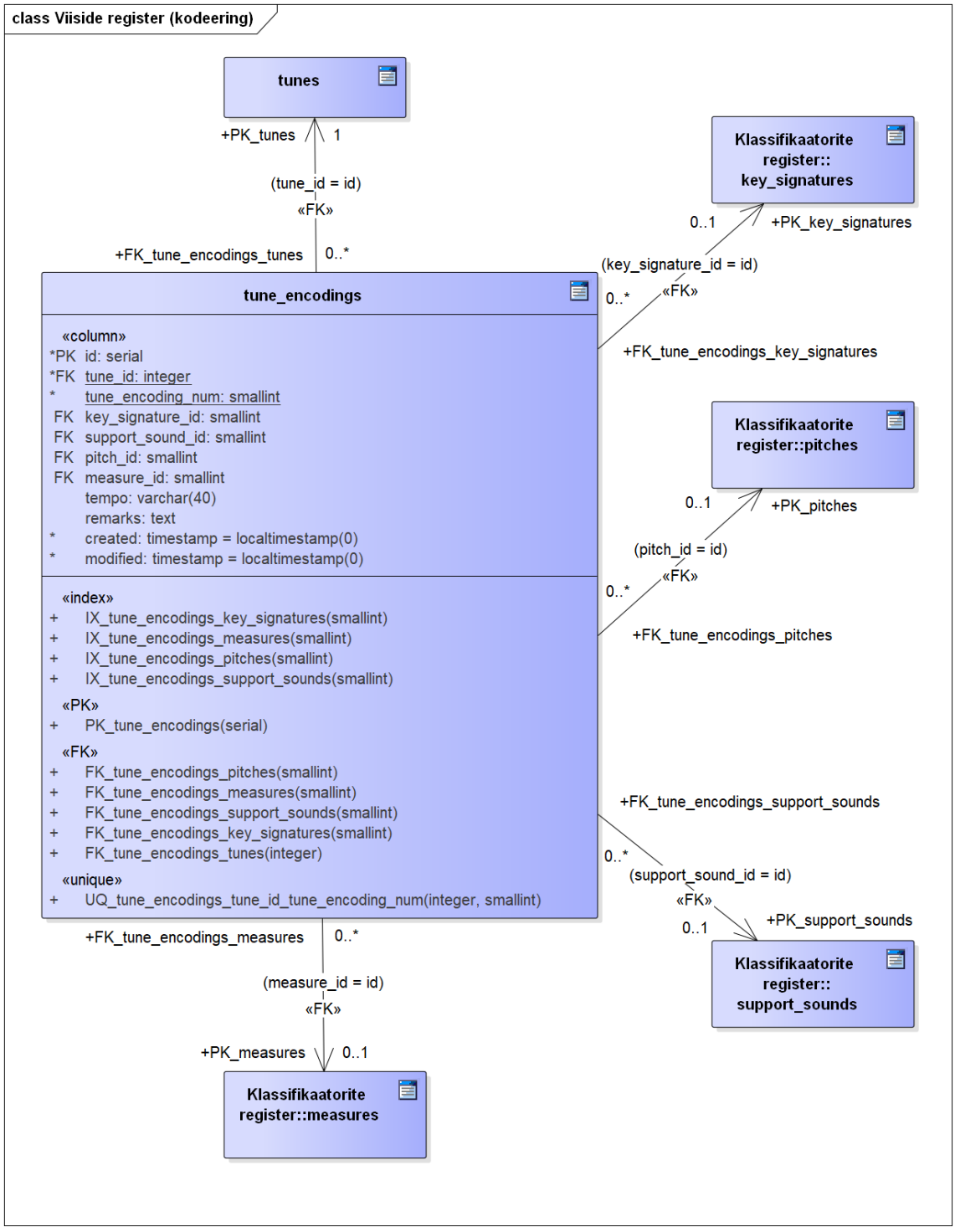
Joonis 65. Viiside registri füüsilise disaini andmebaasi diagramm muusikaliste tunnuste (teksti vorm, rütmitüüp) kohta.



Joonis 66. Viiside registri füüsilise disaini andmebaasi diagramm viisi tegijate kohta.



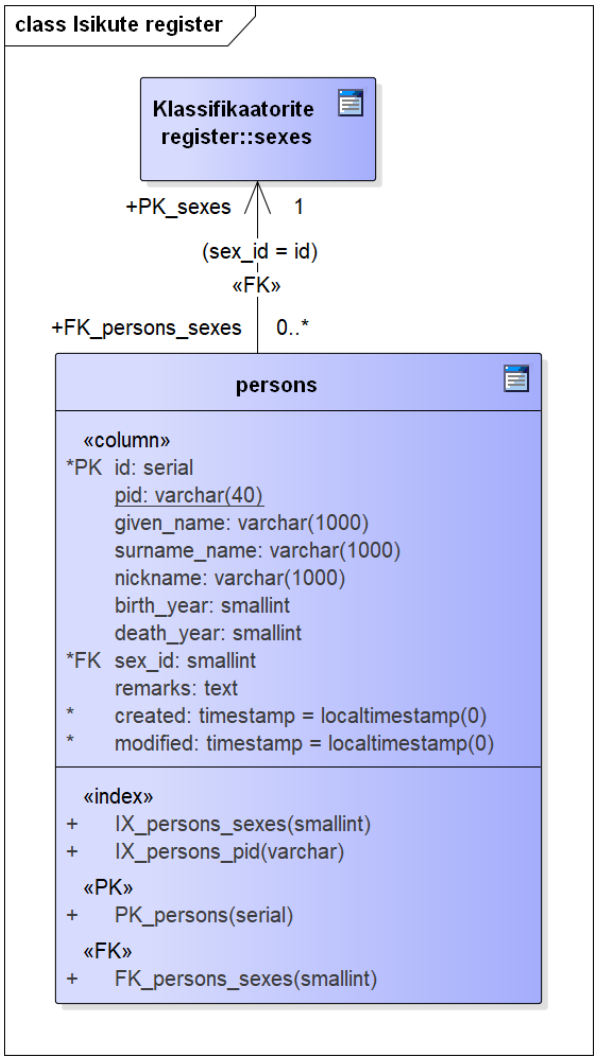
Joonis 67. Viiside registri füüsilise disaini andmebaasi diagramm noodistuste kohta.



Joonis 68. Viiside registri füüsilise disaini andmebaasi diagramm kodeeringute kohta.

Lisa 11 – Isikute registri disain

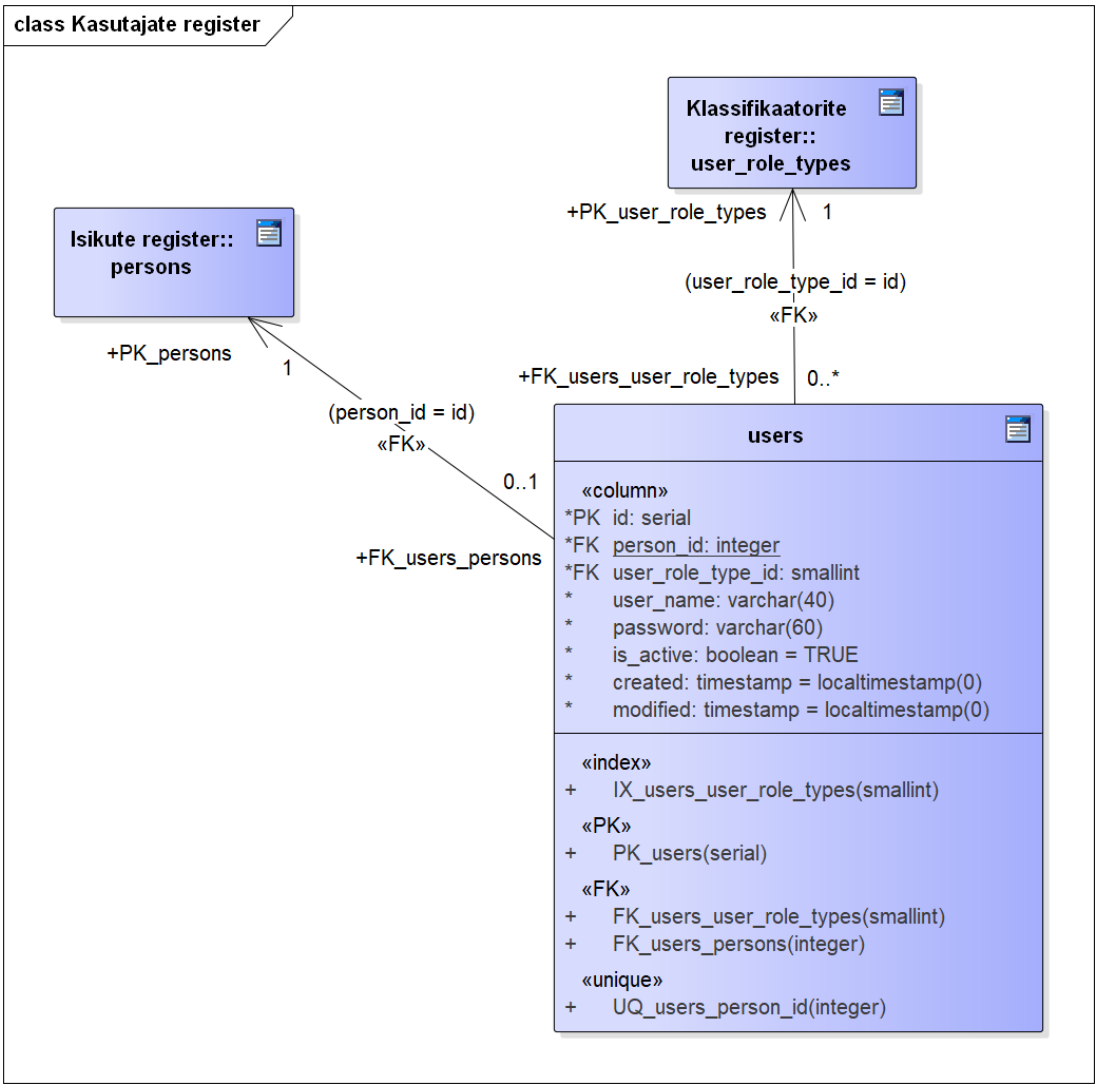
Joonis 69 esitab isikute registri füüsilise disaini.



Joonis 69. Isikute registri füüsilise disaini andmebaasi diagramm.

Lisa 12 – Kasutajate registri disain

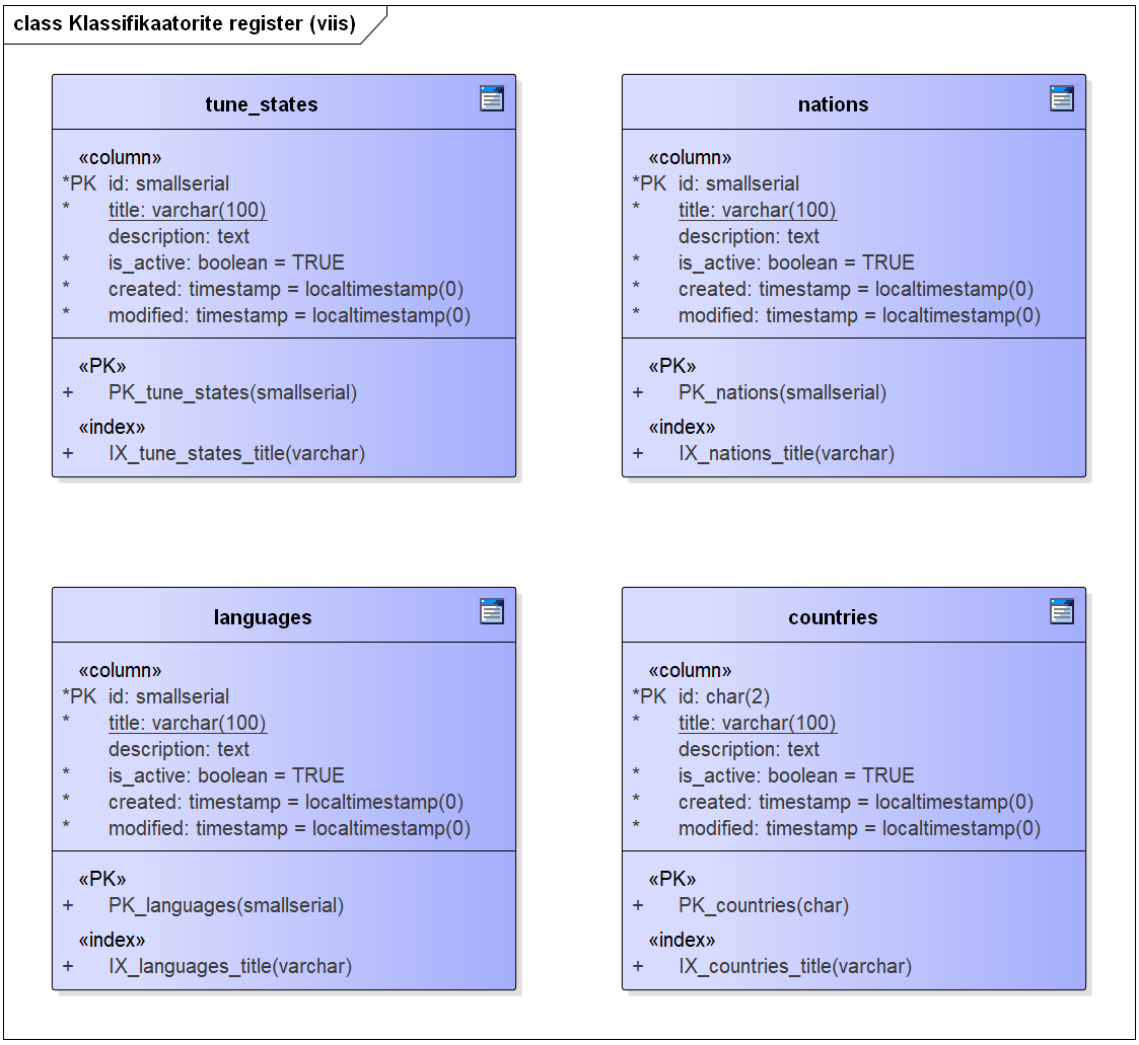
Joonis 70 esitab kasutajate registri füüsilise disaini.



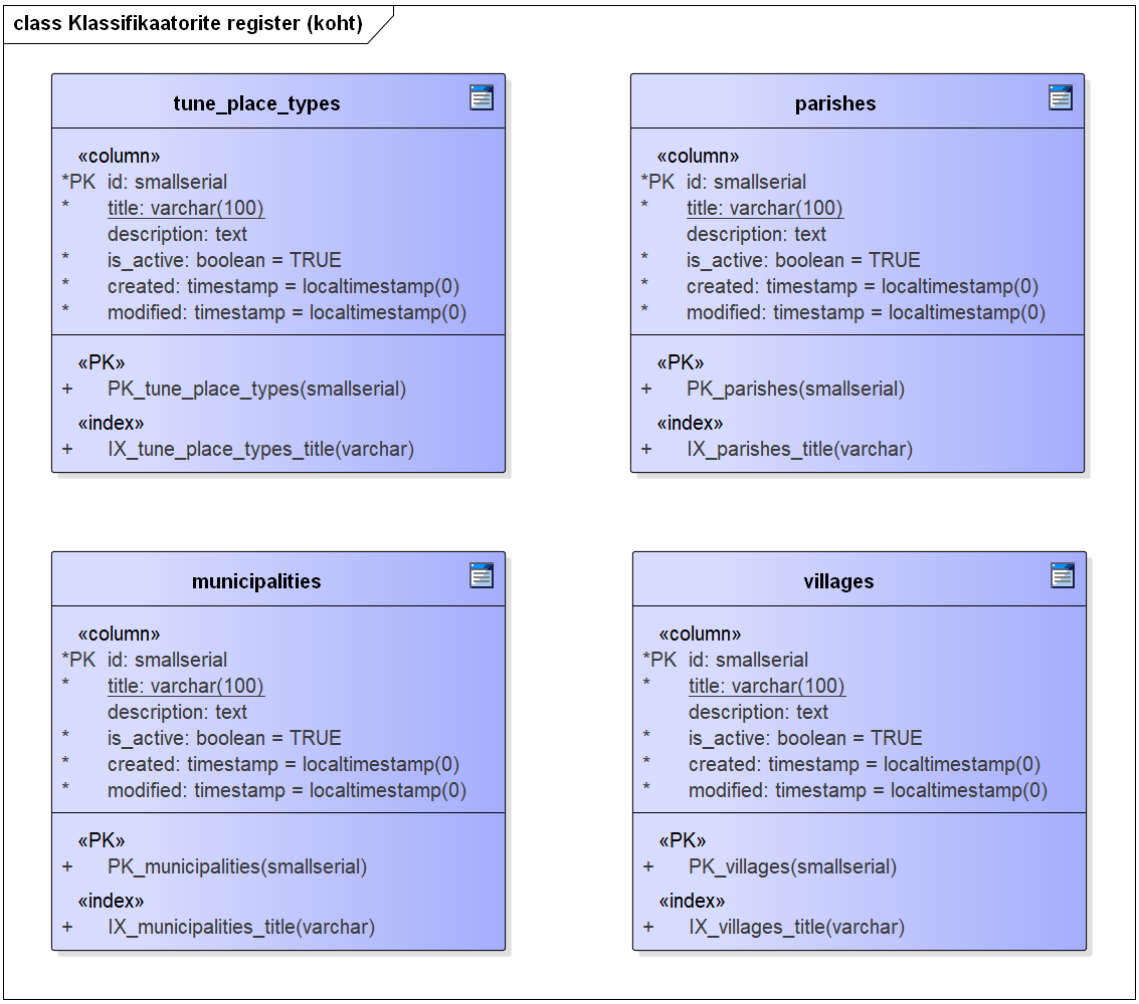
Joonis 70. Kasutajate registri füüsilise disaini andmebaasi diagramm.

Lisa 13 – Klassifikaatorite registri disain

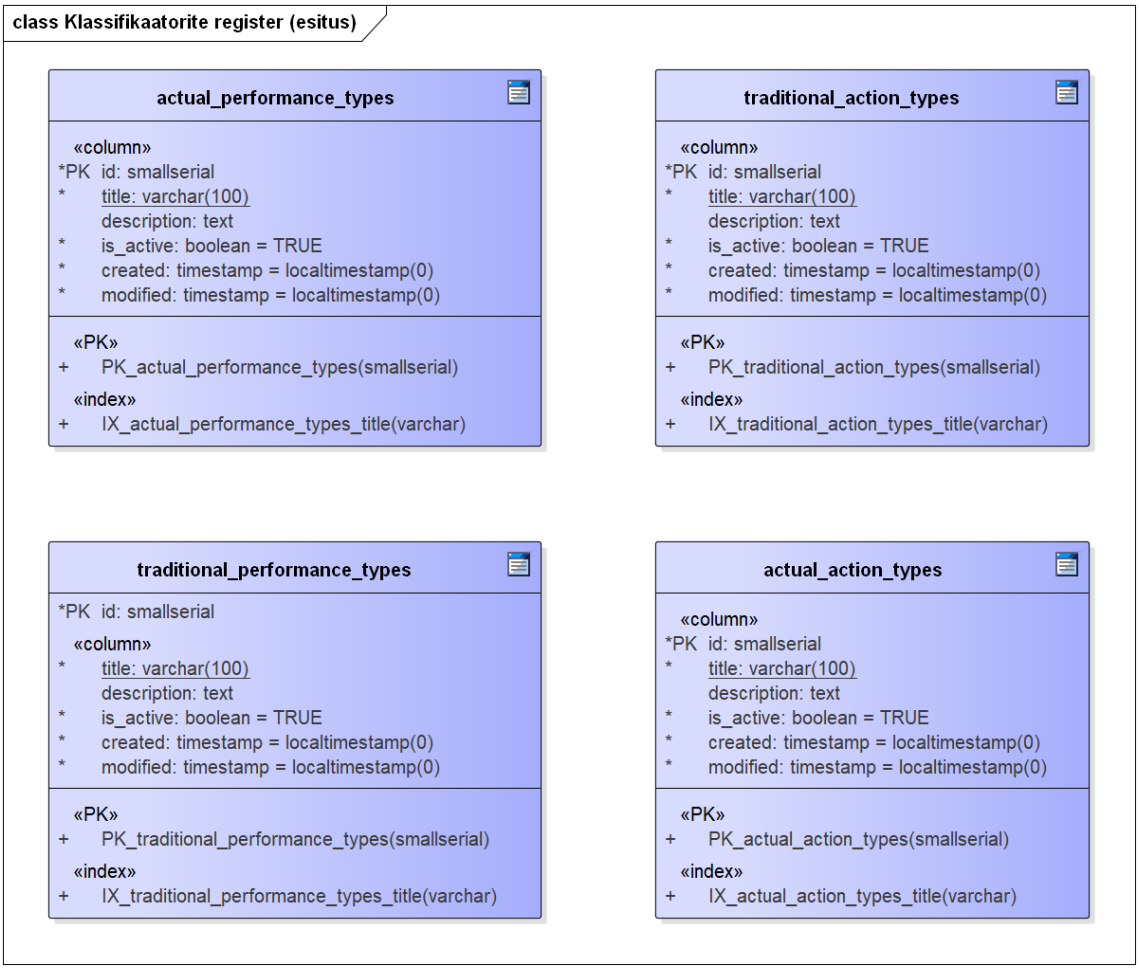
Joonis 71 – Joonis 78 esitavad klassifikaatorite registri füüsilise disaini.



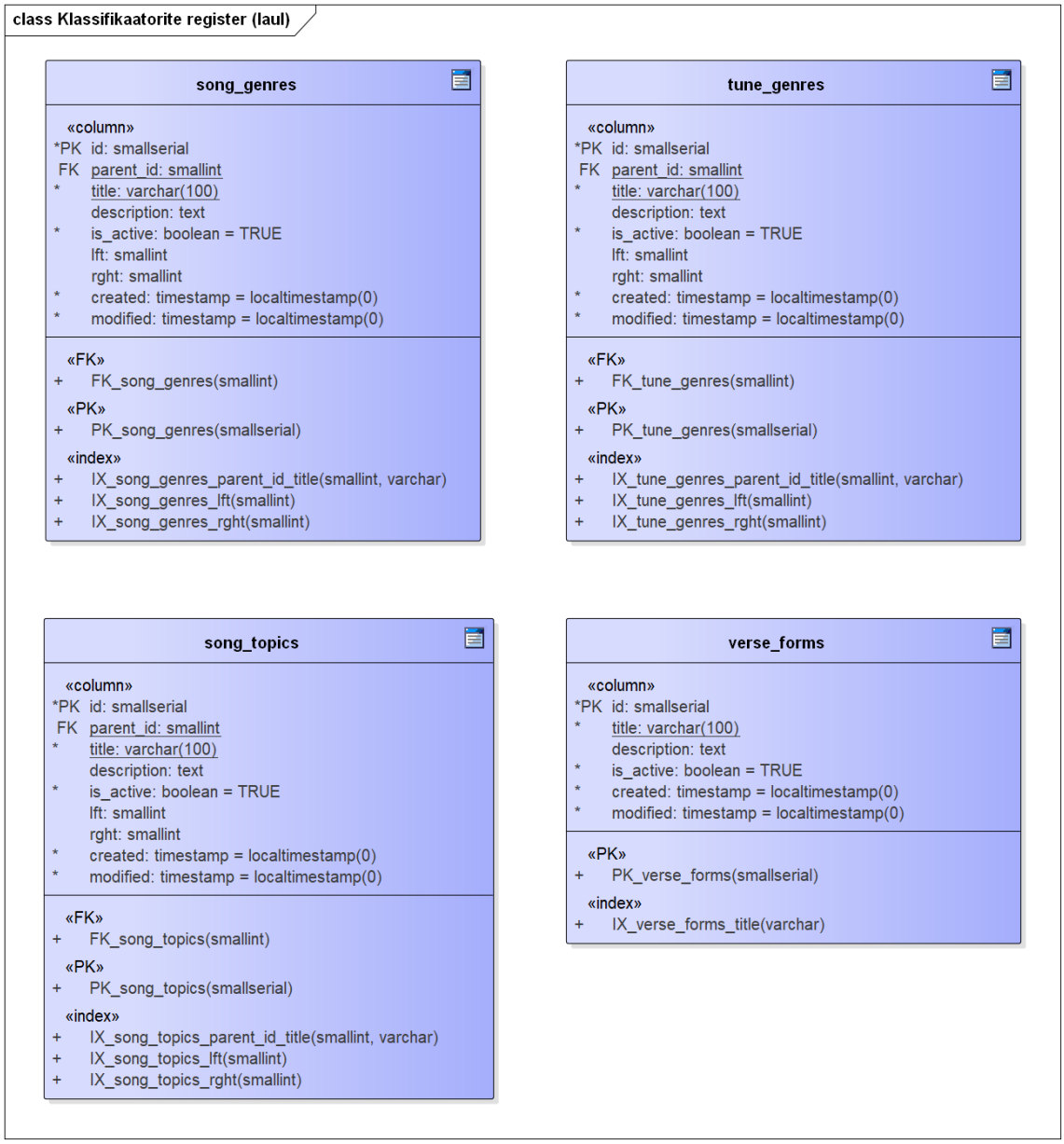
Joonis 71. Klassifikaatorite registri füüsilise disaini andmebaasi diagramm viiside klassifikaatorite kohta.



Joonis 72. Klassifikaatorite registri füüsilise disaini andmebaasi diagramm kohtade klassifikaatorite kohta.

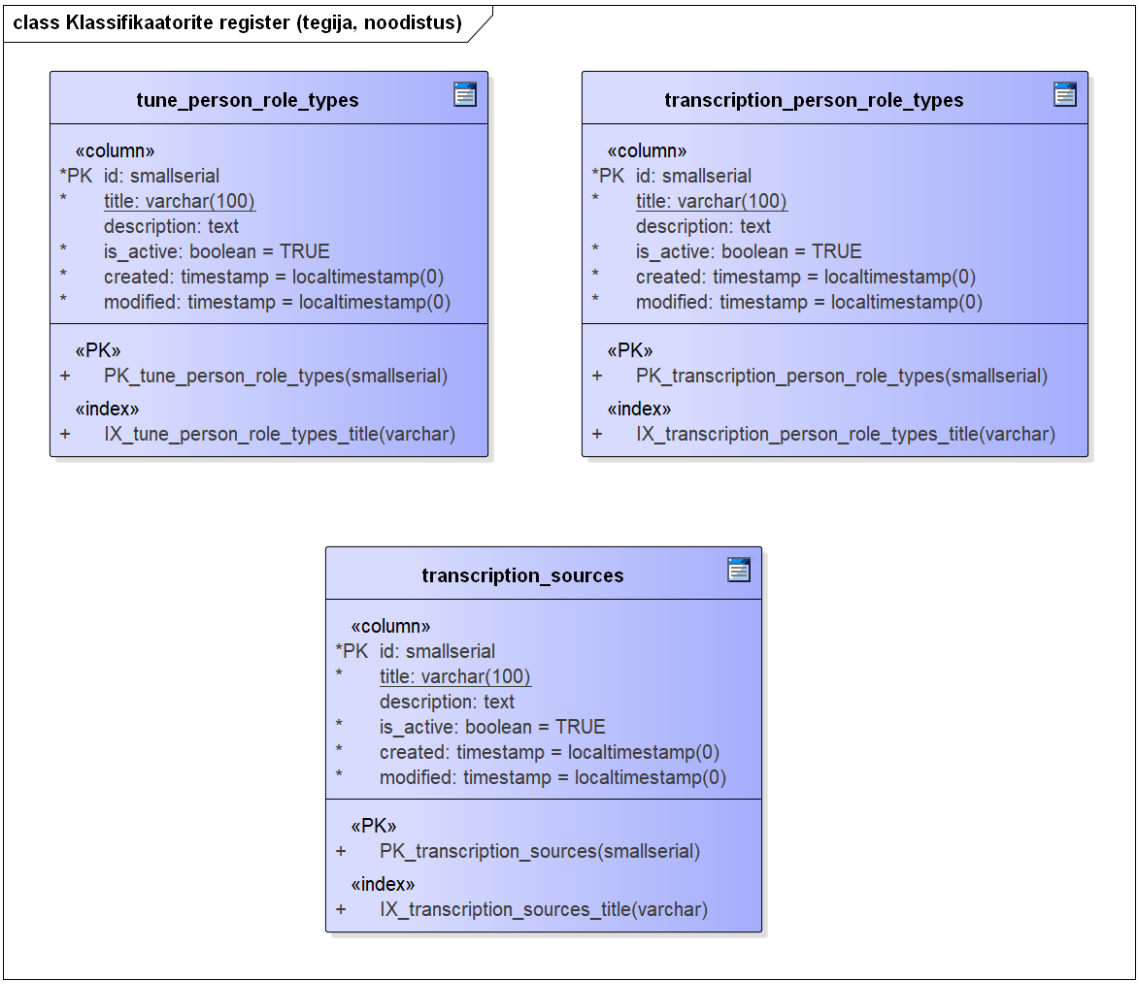


Joonis 73. Klassifikaatorite registri füüsilise disaini andmebaasi diagramm esituste klassifikaatorite kohta.

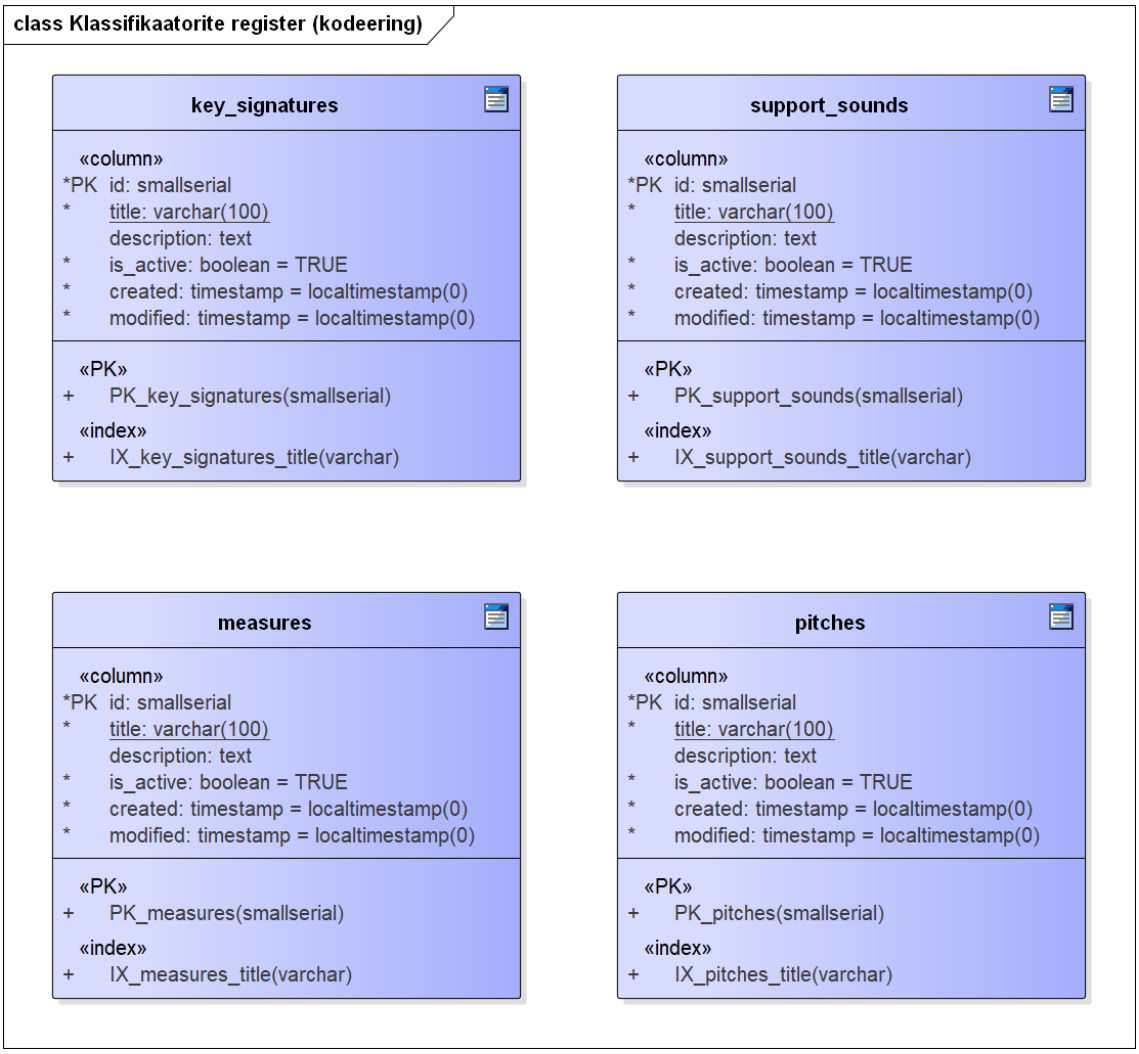


Joonis 74. Klassifikaatorite registri füüsilise disaini andmebaasi diagramm laulude klassifikaatorite kohta.

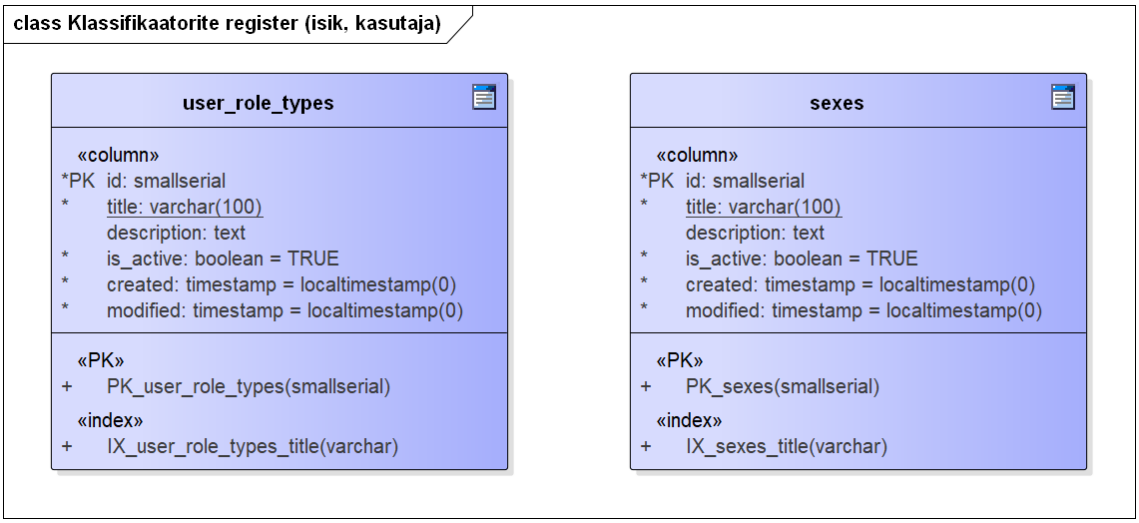
Joonis 75. Klassifikaatorite registri füüsilise disaini andmebaasi diagramm muusikaliste tunnuste klassifikaatorite kohta.



Joonis 76. Klassifikaatorite registri füüsilise disaini andmebaasi diagramm tegijate ja noodistuste klassifikaatorite kohta.



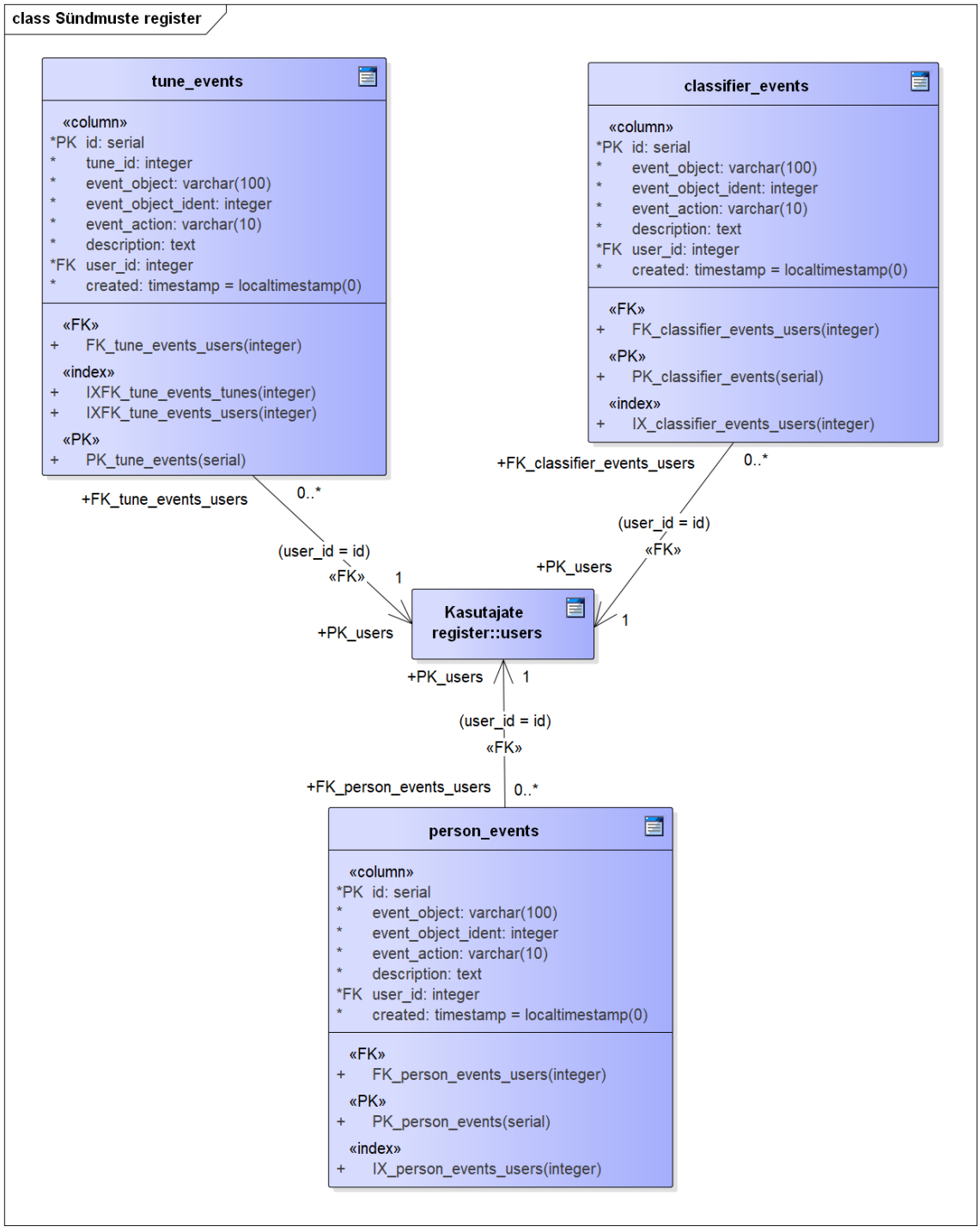
Joonis 77. Klassifikaatorite registri füüsilise disaini andmebaasi diagramm kodeeringute klassifikaatorite kohta.



Joonis 78. Klassifikaatorite registri füüsilise disaini andmebaasi diagramm isikute ja kasutajate klassifikaatorite kohta.

Lisa 14 – Sündmuste registri disain

Joonis 79 esitab sündmuste registri füüsilise disaini.



Joonis 79. Sündmuste registri füüsilise disaini andmebaasi diagramm.