TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Nikolai Kopa 164062IAPB

POSTGRESQL ANDMEBAASSÜSTEEMI PÕHISE METAANDMETEGA JUHITAVATE VEEBIRAKENDUSTE KIIRPROGRAMMEERIMISE KESKKONNA EDASIARENDUS

Bakalaureusetöö

Juhendaja: Erki Eessaar
PhD

Tallinn 2019

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Nikolai Kopa

23.05.2019

Annotatsioon

Selle bakalaurusetöö eesmärgiks oli pgApexi programmis puuduva funktsionaalsuse realiseerimine. pgApex on PostgreSQL andmebaaside veebipõhiste andmebaasirakenduste metaandmetega juhitav kiirprogrammeerimise keskkond. See programm on mõeldud Oracle Application Express (Oracle APEX) analoogina.

Töö käigus uurisin pgApexi programmi arhitektuuri ja selle andmebaasi, sest selle teadmine on uue funktsionaalsuse realiseerimiseks vajalik. Kasutuslugude loetelu alusel planeerisin programmi järgmine väljalaske ja iteratsioonides lahendatavad ülesanded. Samuti kirjeldasin töös arendusprotsessi. Töö käigus tegin süsteemi analüüsi, mis kirjeldab ainult lisatud funktsionaalsust ning esitasin andmebaasis, kasutajaliideses (*frontend*) ning tagasüsteemis (*backend*) tehtud muudatused.

Töö tulemusena realiseerisin järgmine funktsionaalsus: tabelivormid (*Tabular Forms*), detailvaated (*Detail View*), alamraportid (*Subreport*) ning APP_USER süsteemse muutuja väärtuse kuvamine HTML koodis ja selle kasutamine tabelivormi funktsioonide väljakutsetes.

Töö tulemuste valideerimiseks lõin näidisrakenduse, mis põhineb minu osalusel tehtud "Andmebaasid II" projektil "Spordiklubi infosüsteemi treeningute arvestus". Samuti viisin tulemuste valideerimiseks läbi kaks intervjuud üliõpilastega, kes olid varem kasutanud pgApexi esimest versiooni ja kellel ma palusin kasutada ka uut versiooni. Realiseerisin mõned nende tehtud ettepanekud ka pgApexi uues versioonis.

pgApexi uue versiooni kood on maailmale avaldatud ja on kättesaadav GitHub-is. https://github.com/nikopa96/pgapex2

Selle hoidla näol on tegemist programmi esimese versiooni hoidla hargmikuga (*fork*). Tulevikus ühendatakse see *pull request* abil pgApex algse hoidlaga ja siis on lähtekood leitav sealt: https://github.com/raitraidma/pgapex

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 59 leheküljel, 9 peatükki, 20 joonist, 14 tabelit.

Abstract

Further Development of a PostgreSQL-Based Rapid Development Environment for the Metadata-Driven Web Applications

The goal of this thesis was to implement some features that were missing in the pgApex program. The program is a PostgreSQL-based rapid development environment for the metadata-driven web applications. The user interface of the development environment is also web-based. The program uses internally PostgreSQL and is meant for creating web applications on top of PostgreSQL databases. pgApex is intended to be an analogy to the Oracle Application Express (Oracle APEX) development environment.

In order to implement new features, I researched the architecture of the program and its database. To decide what feautures to implement in the new version of the program I made a release plan based on the Tommy Norman article “Agile Release Planning 101”. As a result, I together with my supervisor decided that I should implement Tabular Forms, Detailed Views, and Subreports. Moreover, we decided that I should add support of the system variable APP_USER to HTML regions and Tabular Form Functions. The value of the variable is the username of the logged-in user. It should be possible to display this value in HTML forms or use it as an argument in invoking a function that is associated with a table form.

I implemented all the planned features. The implementation of the features affected the database, user interface, and backend. This work give an overview of all changes that I made in the system to implement the new features: new and changed database functions and tables, data types, as well as changes in user interface and in the backend. Moreover, this thesis gives an overview of system analysis and development process.

The research method of the thesis is design science. The result of the thesis is a technical artifact (a new version of pgApex). In order to validate the results, I developed a database application that uses all new features. I also conducted two interviews with users, who used previous pgApex version. I asked them to use the new version as well and collected

their feedback. I also implemented some suggested ideas from the interviews, e.g., automatic generation of sequence numbers and pagination query parameters on pages that are used to create regions. Moreover, I wrore unit tests for the new features.

The code of the new version of pgApex is published in GitHub. https://github.com/nikopa96/pgapex2

This repository is a fork of the repository that contains the code of the first version of pgApex (https://github.com/raitraidma/pgapex). In the future, I will merge my repository with this repository by using pull request. After this, the source code will be there.

Development of pgApex is not finished with this work. One could still do many things to extend and improve the system. Some ideas about this are in the development view that is a part of the thesis document.

The thesis is in Estonian and contains 59 pages of text, 9 chapters, 20 figures, 14 tables.

Lühendite ja mõistete sõnastik

APEX	*Application Express,* Oracle Application Express (Oracle APEX). Programm veebipõhiste andmebaasirakenduste loomiseks eeskätt Oracle andmebaasisüsteemis loodud andmebaasidele
CSS	*Cascading Style Sheets,* „Veebilehtede valmistajatele ja kasutajatele mõeldud laadistik“ [1]
DOM	*Document Object Model*, „Eeskiri selle kohta, kuidas objekte (tekst, pildid, pealkirjad, lingid jne.) veebilehel esitada. DOM määrab ära, millised atribuudid kuuluvad millise objekti juurde ning kuidas objekte ja atribuute käsitleda” [1]
HTML	*HyperText Markup Language*, „Enimlevinud kodeerimissüsteem (tekstivorming) veebidokumentide loomiseks. HTML koodid ehk märgendid määravad ära selle, kuidas veebileht arvutiekraanil välja näeb“ [1]
JSON	*JavaScript Object Notation,* „lihtsustatud andmevahetusvorming, mis põhineb JavaScripti programmeerimiskeele alamhulgal. JSON on tekstvormingus ja programmeerimiskeelest sõltumatu“ [2]
GET-päring	HTTP meetodi päring, mis küsib infot serverist ja saadab seda kliendile
PDO	*PHP Data Objects*, PHP programmeerimiskeele laiendus, mis kirjeldab andmebaasi kasutamiseks mõeldud liidest [3]
PHP	*PHP: Hypertext Preprocessor*, „PHP on platvormist sõltumatu skriptikeel“ [1]
PL/pgSQL	*Procedural Language/PostGres Structured Query Language*, PostgreSQL andmebaasisüsteemis kasutusel olev protseduurne programmeerimiskeel andmebaasiserveris talletatud rutiinide (funktsioonid ja protseduurid) loomiseks.

POST-päring	HTTP meetodi päring, mis saadab andmeid kliendist serverile
REST	*Representational State Transfer*, „tarkvaraarhitektuuri laad, mis seab veebirakenduse loomisele kindlad piirid. Tihti kutsutakse sellised veebirakendusi ka RESTful veebirakendusteks. Veebirakendused, mis on ehitatud REST arhitektuuril, tagavad internetis rakenduste koostoimimise“ [4]
SQL	*Structure Query Language*, relatsioonilise mudeli alusel välja töötatud andmebaasikeel, mida saab kasutada nii erinevate andmebaasiobjektide haldamiseks, andmete otsimiseks kui ka andmete muutmiseks
URL	*Uniform Resource Locator*, internetiaadress

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänapäeval eelistavad eraisikud ja ettevõted hoida informatsiooni arvutisüsteemide abil loodud andmebaasides. Andmebaasi kasutamine nõuab erioskuseid ja tavalisel kasutajal võivad need puududa. Selleks, et lihtsustada tööd andmebaasiga ja mugavalt andmetega töötada, võivad kasutajad kasutada spetsiaalseid rakendusi. Rakendus on lõppkasutaja jaoks loodud ning terviklik programm [1], mis on kas kirjutatud nullist mingis programmeerimiskeeles või loodud spetsiaalvahendi kasutamise abil (võimalik et) ilma programmeerimisoskusteta. Neid spetsiaalvahendeid nimetatakse kiirprogrammeerimise keskkondadeks ja üks tuntud kiirprogrammeerimise keskkonna näide on Oracle Application Express (APEX). Oracle APEX on mitmeotstarbeline vahend veebipõhiste andmebaasirakenduste loomiseks ja see töötab eeskätt Oracle andmebaasisüsteemi abil loodud andmebaasidega. Alates 2018. aasta algusest saab läbi REST veebiteenuste panna Oracle APEX rakenduse kasutama ka teiste andmebaasisüsteemide hoole all olevaid andmeid [5]. Selles allikas [6] on mängitud läbi Oracle APEX veebirakenduse loomine PostgreSQL andmebaasi põhjal. See on võimalik, kuid arendajalt oodatakse suure hulga koodi kirjutamist. Samuti on sellisel juhul vaja lisaks PostgreSQL andmebaasisüsteemile võtta kasutusele ka Oracle andmebaasisüsteem. Kuigi tegemist võib olla tasuta pakutava Oracle Express Edition andmebaasisüsteemiga tähendab see ikkagi suuremaid halduskulusid (õppimine, ülalhoid) ning mida rohkem on rakenduses kasutusel erinevaid tarkvarakomponente, seda keerukamaks ja raskemini hallatavaks see muutub. Seega on endiselt aktuaalne küsimus, kas saaks luua PostgreSQL jaoks Oracle APEX stiilis rakendusi lihtsamal viisil.

1.1 Taust ja probleem

2016. aastal kaitsti magistritöö “PostgreSQL andmebaasisüsteemi põhine metaandmetega juhitavate veebirakenduste kiirprogrammeerimise keskkond” [7], mille autor Rait Raidma arendas tarkvara nimega pgApex. pgApex on veebipõhine kiirprogrammeerimise tarkvara veebipõhiste PostgreSQL andmebaasirakenduste loomiseks. Seda oskaks rakenduste loomiseks kasutada isegi ilma põhjalike rakenduste

programmeerimisoskusteta kasutaja. Eelduseks on, et andmebaasi arendaja on andmebaasis loonud vaadete ja funktsioonide näol liidese (virtuaalse andmete kihi), läbi mille toimub andmete kasutamine, sest rakendus ei pöördu otse baastabelite (edaspidi tabelite) poole, vaid kasutab neid läbi eelnimetatud liidese. pgApex-i loomisel oli eesmärgiks pakkuda PostgreSQL andmebaasirakenduste loojatele samasugust veebirakenduste loomise võimalust nagu teeb Oracle oma Application Express (Oracle APEX) veebirakenduste kiirprogrammeerimise vahendiga. Mõlema vahendi eripäraks on, et rakenduse kirjeldus salvestatakse andmebaasis ning loetakse sealt kasutajale veebilehe koostamise käigus. Seega on need vahendid ka ise andmebaasirakendused, kus rakenduse kirjeldamise tulemus salvestatakse andmebaasis.

Rait Raidma magistritöö tulemusena loodi pgApex-i esimene versioon, kuid palju vajalikku funktsionaalsust jäi süsteemi mahukuse tõttu realiseerimata. Veendusin selles ise, kui kasutasin pgApexit „Andmebaasid II“ õppeaine aineprojekti tegemiseks [8]. Järgnevad on mõned näited pgApexi hetke puudustest, mis muuhulgas piirasid ka „Andmebaasid II“ aineprojekti tegijaid.

- Ei toeta rakenduse importimist (varundamist) ja eksportimist (taastamist).
- Ei ole võimalik luua tabelivorme, kus olemite nimekirjas saab valida hulga olemeid ning anda korralduse teha nendega mingit ühesugust tegevust (näiteks kustutada või muuta seisundit).
- Ei saa luua rakendust, kus ühel lehel näidatakse olemite nimekirja ning konkreetse olemi juures klõpsates avaneb teisel lehel vorm selle detailandmete vaatamiseks ning muutmiseks.
- Ei ole realiseeritud süsteemset muutujat, mille väärtuseks on sisseloginud kasutaja kasutajanimi. Selle muutuja väärtust võiks saada kuvada lehel või anda ette rakendusest väljakutsutavatele andmebaasi rutiinidele.

Oracle APEX vahendis saab kõike eelnevat ning ka palju muud teha.

1.2 Ülesandepüstitus

Selle bakalaurusetöö eesmärgiks on realiseerida pgApexis puuduvat funktsionaalsust. Juba ülesandepüstituse koostamise käigus andsin endale aru, et suure töömahu tõttu pole korraga võimalik realiseerida kogu funktsionaalsust, vaid tuleb teha valik.

1.3 Metoodika

Selle bakalaurusetöö tegemiseks kasutan ma disainiteaduse metoodikat (*Design Science*) [9]. See tähendab, et lähtuvalt kasutajate (antud juhul juhendaja ning tänu varasemale pgApex kasutamisele ka minu enda) nõuetest ja maailmas juba kogutud teadmistest, realiseerin ma tehnilise tehise e artefakti, milleks antud juhul on pgApexi uus versioon.

Kuna puuduvat funktsionaalsust on palju, aga töö maht on piiratud, siis on vaja otsustada, mida on vaja realiseerida esmalt ja mida hiljem (näiteks järgmistes lõputöödes). Selleks planeerin ma järgmise väljalaske (*release planning*). Alguses on mul vaja lähemalt uurida kuidas pgApex töötab. pgApex programmi arhitektuuri uurimiseks ma kasutan informatsiooni Rait Raidma magistritööst [7] ning programmi Xdebug, mis võimaldab siluda projekti kirjutatud PHP programmeerimiskeeles. Pärast seda on mul vaja valida väljalaske planeerimise meetod ja selle meetodi põhjal planeerida järgmine väljalase. Peale seda on vaja süsteemi täiendused kavandada, programmeerida ja kirjutada ühiktestid.

Töö tulemusena on vaja väljastada uus pgApex-i versioon (töös edaspidi pgApexi teine versioon), kus on realiseeritud kogu planeeritud funktsionaalsus. Samuti on vaja teha nii, et pärast uue pgApex-i versiooni väljastamist oleks järgmistel arendajatel võimalus seda projekti edasi teha. See tähendab, et uus versioon tuleb GitHubis maailmale avaldada. Samuti tuleb tarkvara uus versioon üliõpilaste jaoks arendusserverisse üles panna.

Disainiteaduse kohaselt peab loodud tehist valideerima. Valideerimiseks loon ma pgApexis veebirakenduse, mis kasutab kogu uut funktsionaalsust. Veebirakendus põhineb minu „Andmebaasid II“ aineprojektil [8]. Samuti on vaja käivitada uued ja juba loodud ühiktestid.

Lisaks kogun töö käigus statistikat pgApex lähtekoodis ja andmebaasis tehtud muudatuste kohta (näiteks lisandunud tabelite arv, lisandunud veergude arv, lisandunud funktsioonide arv). See iseloomustab üheltpoolt minu tehtud töö mahtu ja teisalt saab seda kasutada toormaterjalina rakenduste evolutsiooni uurimise juures (see pole antud töö teema) [10].

1.4 Ülevaade tööst

Kõigepealt kirjeldan pgApexi esimest versiooni, st kirjeldan seda, milline oli süsteemi olukord enne uue arendustsükli algust. Seejärel planeerin antud lõputöö väljalaske. Järgnevalt annan ülevaate süsteemi arendusprotsessist. Seejärel kirjeldan täpsemalt funktsionaalseid nõudeid, millest lähtuvalt loon pgApexi uue versiooni. Töö järgmised osad on pühendatud pgApexis tehtud muudatuste tehnilise täpsusega kirjeldamisele. Tulemuse valideerimiseks loon ma näiterakenduse. Samuti kirjeldan ma töö lõpus oma nägemust sellest, mida peaks tulevikus pgApexi edasiarendamiseks ette võtma.

2 Süsteemi olukord enne arendust

pgApex on kiirprogrammeerimise keskkond eeskätt PostgreSQL andmebaaside veebirakenduste loomiseks. PostgreSQL positsioneerib ennast Oracle APEX analoogina. Olgu märgitud, et tänu võimalusele luua PostgreSQLi andmebaasis väliseid tabeleid, on pgApex abil põhimõtteliselt võimalik luua veebirakendust ka andmebaasidele, mis ei ole PostgreSQL hoole all [11] (vt peatükk 1, kus kirjeldati, et ka Oracle APEXit saab kasutada veebirakenduste loomiseks mitte-Oracle andmebaasidele).

Selleks, et luua oma veebirakendust pgApex keskkonnas, ei ole vaja teada programmeerimiskeeli – veebirakenduse loomise protsess sarnaneb programmi loomisega visuaalse programmeerimise keskkonnas (näiteks vormide ja aruannete loomine MS Accessis).

Järgnevalt kirjeldatakse pgApexi üldist tööpõhimõtet ja süsteemi arhitektuuri. Kogu järgnev tekst selles peatükis kirjeldab ainult selle programmi esimest versiooni, mille autoriks oli Rait Raidma ja ei kirjelda selle lõputöö tulemusena lisatud funktsionaalsust.

2.1 Tööpõhimõte

pgApex kasutajaliides koosneb kahest osast – administraatori töökeskkond, kus arendaja loob veebirakendust ja veebirakenduste käituskeskkond, mille abil käitada selles keskkonnas loodud veebirakendusi. pgApexi keskkonnas saab luua null või rohkem veebirakendust. pgApex keskkond on serveris, kus on PostgreSQL andmebaaside kobar e klaster ning iga loodud veebirakendus peab kasutama täpselt ühte selles kobaras olevat PostgreSQL andmebaasi. Kui veebirakendusel oleks vajalik kasutada andmeid mitmest andmebaasist, siis oleks lahendus andmebaasi väliste tabelite lisamine (mis sisaldavad andmeid teistest andmebaasidest) ning rakenduse neid (vaadete ja funktsioonide kaudu) kasutama panemine.

pgApexi abil loodud veebirakenduse eesmärk on pakkuda kasutajale lihtsat ja mugavat ligipääsu andmebaasis hoitavatele andmetele ja anda võimalus neid lihtsalt töödelda.

Andmete esitamine veebirakendusele toimub vaadete kaudu (*View*), muutmine (näiteks andmete värskendamine, lisamine, kustutamine) toimub funktsioonide abil. Nii vaated kui ka funktsioonid peavad olema loodud arendaja poolt PostgreSQL andmebaasis. Veebirakendus ei saa kasutada andmeid otse baastabelitest. See on üks erinevus Oracle APEXist (kus see on lubatud) ja see oli pgApexi loomisel tehtud teadlik valik, et kasutada maksimaalselt ära andmebaasi avalikust liidesest (mis koosneb vaadetest ja funktsioonidest) tulenevaid eeliseid [12].

Veebirakendus on HTML lehtede kogum. Iga leht koosneb regioonidest. Regioon on lehe plokk, kus on paigutatud mingi element: navigatsioon, vorm, raport või HTML kood.

- **Navigatsiooni regioon** (*Navigation region*): navigatsiooni paneel, kus on paigutatud lingid selle rakenduse teiste lehtedele või globaalsed lingid (URL-i abil). Enne navigatsiooni regiooni kasutamist on vaja luua navigatsiooni menüüs uus navigatsioon ja selle sees kirjeldada linke ja nende nimesid.
- **Vormi regioon** (*Form region*): vorm, mis võib sisaldada üks või mitu sisendvälja, millest igaüks on mingit tüüpi (*Input types*). pgApex toetab järgmiseid sisendväljade tüüpe: tekst (*Text*), parool (*Password*), raadionupud (*Radio buttons*), märkeruudud (*Checkboxes*), rippmenüü (*Drop down*) ja tekstiala (*Text area*). Tööpõhimõte seisneb selles, et vajutades nuppule „Kinnita“ (*Submit*), käivitab pgApex arendaja poolt loodud andmebaasi funktsiooni ja funktsiooni parameetrite väärtustena kasutatakse sisendväljadesse sisestatud informatsiooni. Funktsiooni valida ja sisendväljadega siduda on võimalik ainult administraatori keskkonnas. Sisendväljade järjekorda vormis on võimalik muutuda.
- **Raporti regioon** (*Report region*): regioon, mis esitab veebirakenduse kasutajale andmed andmebaasist tema jaoks sobivas hulgas ja formaadis. Raporti kaudu võib kasutaja ainult lugeda andmeid, mitte neid manipuleerida. Raporti regioon võtab andmed arendaja poolt loodud andmebaasi vaadetest. See regioon näeb välja nagu tabel, kus on päis ja kehand. Päis kirjeldab veerud. Kehand koosneb ridadest. Raportisse on võimalik lisada eraldi veerge URL linkidega. Veergude järjekorda on võimalik muuta ja see ei mõjuta andmebaasis loodud vaadet. Selle regiooni sees võib ka häälestada paginatsiooni (tulemuste jaotumist portsudeks) – regioonis võib olla korraga esitatud piiratud arv ridu ja et vaadata järgmiseid ridu on vaja laadida leht uuesti. Selle jaoks on vaja kirjeldada regiooni

spetsifikatsioonis paginatsiooni päringu parameeter (*Pagination query parameter*).

- **HTML regioon** (*HTML region*): sisaldab HTML koodi. Arendaja saab seda kasutada staatilise teksti (ei võeta rakenduse andmebaasist) ilusal kujul esitamiseks.

Regioonide järjekorda lehel on võimalik muuta.

Administraatori keskkonnas võib arendaja lisada rakendusse lehti ja määrata nende jaoks aliasi ehk URL-is kasutamiseks mõeldud alternatiivse nime. Iga lehe jaoks võib arendaja määrata, kas ligipääs sellele peab olema ainult veebirakenduse autenditud kasutajatel või on juurdepääs piiramatu kõigile, kes teavad lehe URLi. Iga veebirakenduse leht jaguneb kaheks osaks – navigatsiooni ala ja regioonide ala. Igas regioonide alas on võimalik luua üks või mitu regiooni.

Igal regioonil võib olla oma mall. Mallid võimaldavad arendajale luua veebirakendusele oma unikaalset disaini e väljanägemist. Esimeses pgApexi versioonis on ainult navigatsiooni regioonil mitu erinevat malli. Teistele regioonidele on kahjuks loodud ainult üks mall.

2.2 Andmebaasi arhitektuur

pgApex kiirprogrammeerimise keskkond kasutab sisemiselt PostgreSQL andmebaasi, mis hoiab andmeid veebirakenduste ülesehitusest ja samuti teiste samas serveris asuvate ning rakendustes kasutatavate PostgreSQL andmebaaside metaandmeid. Need andmed on vajalikud keskkonna funktsioneerimiseks.

2.2.1 Metaandmed

Metaandmed on antud juhul informatsioon andmebaasi objektide kohta. pgApexi puhul asuvad need andmed materialiseeritud vaadetes (*Materialized View*) e hetktõmmistes ja need on ainult PostgreSQL andmebaaside kohta, mis asuvad samas serveris. Kõikides vaadetes peale *database* on andmed ainult nende andmebaaside kohta, mida kasutab mõni pgApex abil antud serveris loodud rakendus. Kõikides vaadetes on andmed ainult mitte-

süsteemsete skeemide ja nende skeemiobjektide kohta. Järgnevalt nimetatakse keskkonna poolt kasutatavad materialiseeritud vaated.

- **data_type**: informatsioon andmetüüpide kohta, mis on kasutusel mitte-süsteemsetes skeemides (andmebaasi nimi, skeemi nimi, andmetüüp).
- **database**: kõikide serveris olevate andmebaaside nimed.
- **function**: informatsioon andmebaasis loodud funktsioonide kohta (andmebaasi nimi, skeemi nimi, funktsiooni nimi, tagastatava väärtuse tüüp). Aktiivsel kasutusel vormide regioonide jaoks.
- **parameter**: informatsioon andmebaasis loodud funktsioonide parameetrite kohta (parameetri nimi, andmebaasi nimi, funktsiooni skeemi nimi, funktsiooni nimi, parameetri andmetüüp, parameetri järjekorranumber). Parameetrite järjekord on oluline informatsioon, sest sellest sõltub funktsiooni väljakutsel selle tulemus.
- **schema**: andmebaaside skeemide nimed.
- **view**: informatsioon andmebaasi vaadete kohta (skeem ja nimi). Aktiivsel kasutusel raportite regioonide jaoks.
- **view_column**: informatsioon vaadete veergude kohta (andmebaasi nimi, skeemi nimi, vaate nimi, veeru nimi, veeru tüüp).

Metaandmeid kasutatakse aktiivselt veebirakenduste loomiseks. Esitatud loetelu materialiseeritud vaadetest moodustab *Andmebaasiobjektide registri*. Metaandmete kasutamise näited on jaotises 2.3.

2.2.2 Veebirakenduste ülesehituse andmed

Need andmed kirjeldavad veebirakenduse ülesehitust – millistest lehtedest rakendus koosneb, millised regioonid asuvad igal lehel, regioonide kuvamise järjekord jne. Kõik need andmed on baastabelites. Kuna baastabeleid on palju, siis ma nimetan ainult registrid, millesse need kuuluvad. Igas registris on üks või mitu baastabelit.

- Rakenduste register.
- Lehtede register.
- Regioonide register.
- Navigatsioonide register.
- Mallide register.

Nendes registrides hoitakse vastavalt informatsiooni rakenduste, lehtede, regioonide, navigatsioonides ja mallide kohta. Nende registride kasutamise näiteid esitatakse jaotises 2.3.

2.3 Programmi arhitektuur

Selles jaotises antakse ülevaade pgApex programmi arhitektuurist.

2.3.1 Administreerimise keskkonna tööpõhimõte

Administreerimise keskkonas loob arendaja veebirakendusi, mille abil võib vaadata andmebaasis olevaid andmeid ja neid muuta. Tehnilises mõttes jaguneb administraatori keskkond kolmeks osaks – kasutajaliides (*front end*), tagasüsteem (*back end*) ja andmebaas. Süsteemis kasutatakse klient-server mudelit (*client-server model*), kusjuures süsteemis on mitu klient-server komponenti ning sama süsteemi osa võib olla nii kliendi kui serveri rollis.

pgApexi kasutajaliides vastutab väljanägemise eest ning koostab JSON objekte POST-päringute jaoks. Veebilehtede küljendamiseks kasutatud Bootstrap 3 raamistikku. Elementide ja lehtede sidumiseks kasutatakse JavaScript raamistikku AngularJS, versioon 1.4.0.

Jaotises 2.2 kirjutatakse, et pgApexi programmil on oma andmebaas, mis hoiab metaandmeid ja andmeid loodud veebirakenduste ülesehituse kohta (sh ka mallid). Lehe või lehe regiooni loomisel peab arendaja selle jaoks kohustuslikult valima andmebaasis loodud funktsiooni, vaate või vaate veeru ning malli. Metaandmeid funktsioonide, vaadete ja vaadete veergude kohta hoitakse *Andmebaasobjektide registris* ning mallide andmeid hoitakse *Mallide registris*. Andmebaasiobjektide metaandmeid ja mallide nimekirja saab programm tehes GET-päringut serverile kasutajaliidese *service* failide kaudu.

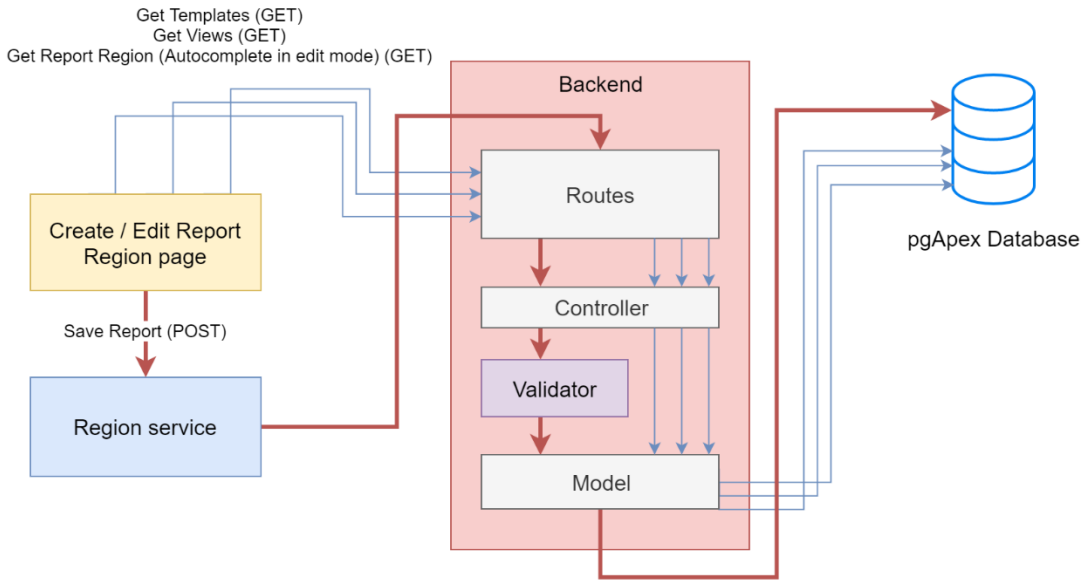
Tagasüsteemi osa on kirjutatud PHP programmeerimiskeeles koos Slim Framework 3 kasutamisega. Tagasüsteemis on realiseeritud GET- ja POST-päringud, mille abil arendaja, kasutades administraatori keskkonda, saab lugeda metaandmeid ja malle või salvestada veebirakenduse ülesehitust andmebaasi.

HTTP-päringuid kirjeldatakse routes.php failis ja sellise päringu käivitamisel alustab oma tööd kontroller (*Controller*).

pgApexis kasutatakse veebirakenduse ülesehituse salvestamiseks POST-päringuid. Pärast andmete sisestamist administraatori keskkonna vormi ja salvestamist, koostab AngularJS raamistikus kirjutatud kasutajaliidese osa JSON-objekti ja seda objekti kasutatakse POST-päringu argumendina. POST-päringu käivitamisel käivitub kontrolleris ka validaator, mis kontrollib, et JSON-objektis sisalduvad väärtused vastavad süsteemi nõuetele. Pärast valideerimist kutsub kontroller välja selle olemi mudeli (*Model*). Olemi mudelist teostatakse PDO abil päring andmebaasi ja see käivitab pgApexi andmebaasi funktsiooni, mis salvestab info andmebaasi. Pärast PDO töö lõpetamist tagastab server kliendile ehk administraatori keskkonnale vastuse staatuskoodiga 200, mis tähendab, et operatsioon täideti edukalt.

Arendajal on ka võimalus muuta juba loodud lehe, regiooni või navigatsiooni ülesehituse andmeid, kui ta vajutab nupule „Muuda“ (*Edit*). Avades muutmise lehe saab arendaja andmeid lehe, regiooni või navigatsiooni ülesehitusest ühe GET-päringu abil, mis tagastab pgApexi andmebaasis käivituva funktsiooni tulemuse JSON formaadis. Kasutajaliidese osa AngularJS raamistikus kasutab JSON dokumendis olevaid andmeid muutmiseks mõeldud vormide täitmiseks.

Joonis 1 kirjeldab administraatori keskkonna tööpõhimõtet. Raporti loomise lehel tehakse GET-päringud mallide ja vaadete nimekirja saamiseks. Raporti muutmise lehel tehakse veel üks GET-päring, mis tagastab Raporti JSON-kujul automaatseks lõpetamiseks. Raporti regiooni salvestamiseks tehakse POST-päring ning enne täitmist päring valideeritakse.



Joonis 1. Administreerimise keskkonna tööpõhimõte Rapoti regiooni loomise näitel.

2.3.2 Veebirakenduse tööpõhimõte

Veebirakendus on HTML lehtede kogum. Veebirakenduse lehed koosnevad regioonidest, mida arendaja ise kirjeldab ja ka määrab nende kuvamise järjekorra ning väljanägemise. Väljanägemise määrab mall e disain. pgApexi lähtekood ei sisalda HTML faile veebirakenduste jaoks ja kõiki malle hoitakse andmebaasis *Mallide registris*.

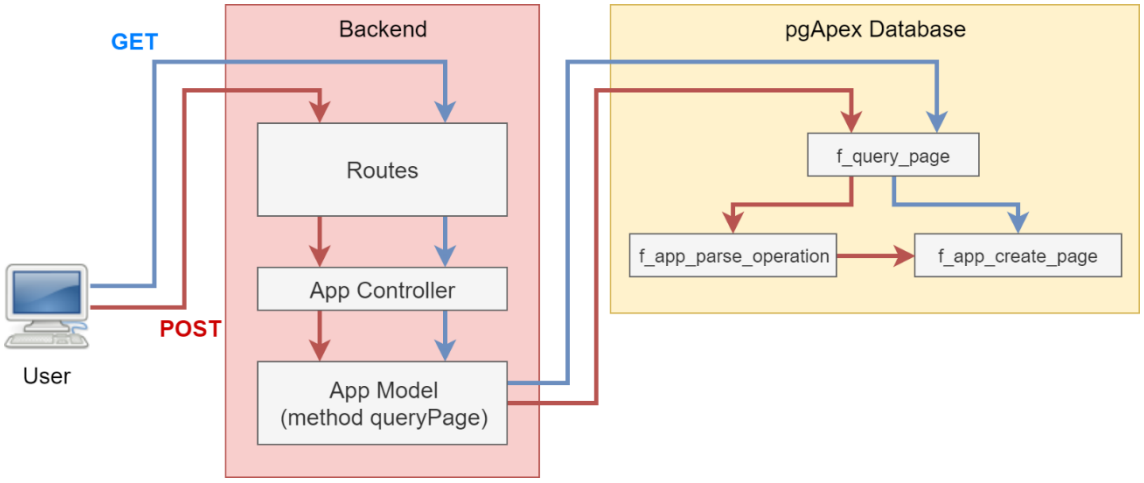
Lehe koostamise eest vastavad pgApexi andmebaasi funktsioonid, mis on kirjutatud SQL või PL/pgSQL programmeerimiskeeltes. Kui kasutaja küsib serverilt veebirakenduse lehte tagarakenduse funktsiooni *queryPage* kaudu, siis teeb server päringu andmebaasist ja käivitab pgApexi andmebaasi funktsiooni *pgapex.f_app_query_page*, mis alustab selle lehe koostamise protsessi. Võib öelda, et see on veebirakenduse alguspunkt. See funktsioon algatab sessiooni, loob ajutised tabelid, loob ühenduse *dblink* mooduliga ning käivitab GET- või POST-päringuid töötlevaid funktsioone.

Lehe regioonide koostamiseks kasutatakse GET-päringuid. Selle eest vastutab funktsioon *pgapex.f_app_create_page*, mis koostab lehe HTML koodi ja käivitab regioonide koostamiseks andmebaasi funktsioone (Joonis 2). Iga regiooni tüübi jaoks – navigatsiooni, raporti, vormi ning HTML regioon – on eraldi funktsioon. Regioonide koostamise funktsioonid saavad regioonides esitatavad andmed välisandmebaasist kasutades *dblink* moodulit ja paigutavad saadud info regiooni kirjeldavasse malli (Joonis

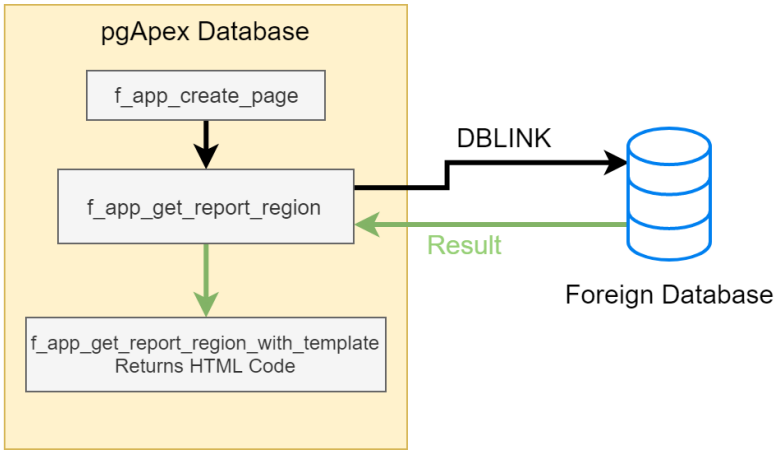
3). Pärast regioonide koostamist paigutab funktsioon *pgapex.f_app_create_page* nende funktsioonide töö tulemused oma tulemusse ja tagastab selle funktsioonile *pgapex.f_app_query_page*.

Joonis 3 kirjeldab, kuidas andmebaasi funktsioon *f_app_create_page* käivitab funktsiooni *f_app_get_report_region*, mis saab andmeid välisandmebaasist *dblink* mooduli abil ja annab saadud andmed edasi funktsioonile *f_app_get_report_region_with_template*, mis paigutab saadud andmeid nõutud malli.

POST-päringud on kasutusel juhul, kui regioonis on vorm kinnitamise nupuga. POST-päringu töötlemine algab funktsiooniga *pgapex.f_app_create_page*, mis alguses käivitab funktsiooni *pgapex.f_app_parse_operation*. See töötleb POST-päringu nõutud regiooni jaoks ja pärast seda käivitab eespool kirjeldatud funktsiooni *pgapex.f_app_create_page*, mis koostab POST-päringu lõpetamisel lehe.



Joonis 2. Veebirakenduse lehe koostamine raporti regiooni näitel, osa 1.



Joonis 3. Veebirakenduse lehe koostamine rapoti regiooni näitel, osa 2.

3 Väljalaske planeerimine

Kuna puuduvat funktsionaalsust on palju, aga töö maht on piiratud, siis on vaja otsustada, mida on vaja realiseerida esmalt ja mida hiljem (näiteks järgmistes lõputöödes). Selleks planeerin ma järgmise väljalaske (*release planning*).

"Tarkvara väljalaske planeerimine (*Software release planning*) on funktsionaalsuste või nõuete valimise probleem, mida peavad olema realiseeritud järgmises väljalaskes või väljalasetes. See on oluline samm tarkvara arendamises, sest on vaja kooskõlastada otsuste kinnitamise kriteeriumid (näiteks ärilised väärtused, pinge ja kulud) arvestades piiranguid (näiteks funktsionaalsuste prioriteet, ressursside kättesaadavus) " [13].

Selle lõputöö tulemusena valmiv pgApexi teine versioon on üks väljalase.

3.1 Kasutatud väljalaske planeerimise meetod

Väljalaske planeerimine on mittetriviaalne ülesanne. On olemas palju väljalaske planeerimise meetodeid. pgApexi teise versiooni väljalaske planeerimiseks otsustasin ma kasutada meetodit, mida kirjeldatakse detailselt Tommy Normani artiklis "Agile Release Planning 101" [14]. Valisin selle meetodi, sest see on piisavalt lihtne. See artikkel kirjeldab, kuidas koostada väljalaske plaani kasutades programmi Excel. Kohandasin seda meetodit enda ülesande jaoks.

Planeerimine algas täitmata ülesannete loendi koostamisest (*backlog*), mis koosneb kasutuslugudest (*User Story*). Alguses määrasin iga kasutusloo jaoks algprioriteedi. Prioriteete võib määrata arvudena ühest lõpmatuseni: mida väiksem on arv, seda suurem on selle kasutusloo prioriteet. Pärast prioriteetide seadmist hindasin iga kasutusloo oodatavat pingutust (*Estimated Effort*). Oodatava pingutuse hindamiseks võib kasutada suvalist mõõdikut (meetrikat), aga kõige populaarsem hindamise mõõdik on Fibonacci arvud. Selle arvude jada esimesed liikmed on 0, 1, 1, 2, 3, 5, 8, 13, 21 [15]. Mida suurem on pingutuse hinnang, seda rohkem aega nõuab selle ülesande täitmine. Kui algprioriteedid olid määratud ja oodatavad pingutused olid hinnatud, siis esitasin väljalaske plaani juhendajale kes antud juhul oli sisuliselt ka toote omanik (*Product*

Owner). Kuna toote omanik näeb iga kasutusloo pingutuse hinnangut, siis on tal võimalus muuta kasutuslugude täitmise prioriteete või jagada ülesanne mitmeks osaks [14].

Kui toote omanik tagastas mulle Excel tabelid ülevaadatud prioriteetidega, siis tegin selle põhjal esialgse otsuse, kui palju iteratsioone läheb arenduseks vaja ja milline on nende kestvus. Lisaks sellele pidin määrama, milliseid ülesandeid iga iteratsioon sisaldab. See sõltub kasutuslugude oodatavast pingutusest ja minu edenemise kiirusest – ülesannete oodatavate pingutuste summa ühes iteratsioonis ei peaks olema rohkem kui oodatav edenemise kiirus [14]. Edenemise kiirus (*Velocity*) selgub iteratsioonide läbimise käigus ja seega pidin töö käigus jooksvalt ümberhindama, millise hulga ülesandeid ma järgmises iteratsioonis proovin lahendada.

Iteratsiooni ajal ei tohi muuta ülesannete nimekirja, aga pärast selle iteratsiooni lõpetamist ja enne uue iteratsiooni algust võib olla vaja väljalaske plaan uuesti üle vaadata. See vastab Suletud akna reeglile (*Closed-window rule*) [16].

Kasutasin ka ajakarbi (*Timebox*) meetodit [17], mis tähendab, et kui ma ei jõudnud kõiki algselt plaanitud ülesandeid iteratsioonis täita, siis ma mitte ei teinud iteratsiooni pikemaks, vaid tõstsin need ülesanded täitmata ülesannete nimekirja, millega tegeleda mõnes järgmises iteratsioonis.

3.2 Uue pgApexi versiooni väljalaske planeerimine

Kuna olin projektis ainus arendaja, siis olin ka väljalaske plaani koostaja. Mul on õigus koostada väljalaske plaani, sealhulgas iseseisvalt määrata kasutuslugude oodatavaid pingutusi, iteratsioonide algust, lõppu ja edenemise kiirust.

Otsustasin teha neli iteratsiooni, igaühe pikkus kaks nädalat. Edenemise kiiruseks kõikides iteratsioonides hindasin 16 punkti. Väljalaske plaani koostamiseks panin funktsionaalsed nõuded kirja kasutuslugudena [18] (Tabel 1).

Tabel 1 Süsteemi kasutuslood.

Kasutusloo ID	Kasutuslugu
1000	Arendajan ma tahan võimalust kasutada süsteemset muutujat APP_USER, et personaliseerida veebirakendusi ja võimaldada tabelivormides muudatusi tehes registreerida, kes neid muudatusi tegi.
1001	Arendajana ma tahan rakendust eksportida ja importida, et teha rakenduse varukoopiat või taastada rakendust varem loodud varukoopia alusel.
1002	Arendajana ma tahan luua tabelivorme, et teha kerge vaevaga andmemuudatusi mitme samatüübilise olemi kohta.
1003	Arendajana ma tahan luua detailvaateid, et pakkuda rakenduse kasutajale paremat ülevaadet konkreetse olemi ja sellega seotud olemite andmetest.
1004	Arendajana tahan ma luua alamraporteid, et konkreetse olemi juures oleks võimalik vaadata sellega seotud olemite andmeid.

Tabel 2 on minu poolt koostatud esialgne iteratsioonide plaan.

Tabel 2. Esialgne iteratsioonide plaan.

Kasutuslood ID	Prioriteet tegemata tööde listis	Oodatav pingutus
1000	1	5
1001	2	8
1002	3	8
1003	4	8
1004	5	5

Tabel 3 on pärast suhtlemist juhendajaga minu poolt parandatud plaan, kus määrasin uued prioriteedid, aga oodatav pingutus ei muutunud. Selles tabelis on read sorteeritud tegemata tööde listi (*backlog*) prioriteedi järgi. Iga värv tähistab eraldi iteratsiooni. Tabelis on toodud ka iteratsiooni lõpp.

Tabel 3. Parandatud iteratsioonide plaan pärast suhtlemis juhendajaga.

Kasutusloo ID	Prioriteet tegemata tööde listis	Oodatav pingutus	Iteratsiooni lõpp
1002	1	8	
1003	2	8	24.03.2019

Kasutusloo ID	Prioriteet tegemata tööde listis	Oodatav pingutus	Iteratsiooni lõpp
1004	3	5	
1000	4	5	7.04.2019
1001	5	8	21.04.2019

Pärast suhtemist juhendajaga planeerisin kolm iteratsiooni, mille ajal ma pidin realiseerima uut funktsionaalsust ja viimane neljas iteratsioon jäi varuks. See tähendab, et kui ma ei jõua mõnda ülesannet täidetud (vt ajakarbi meetod), siis see ülesanne jaguneb osadeks ja tegemata osad siirduvad järgmisesse iteratsiooni.

Kuna algselt määrasin kasutuslugudele oodatavad pingutused kõhutunde järgi ja ülesannete tegemine nõudis planeeritust rohkem aega, siis ma korrigeerisin töö käigus väljalaske plaani, jagades ülesandeid mitmeks osaks. Vastavalt suletud akna ja ajakarbi reeglitele oli mul selleks õigus.

Tabel 4 esitab viimase iteratsioonide plaani, milleni jõudsin peale kolmandat iteratsiooni. Mahukad kasutuslood ID-ga 1003 ja 1004 (Tabel 1) jagatsin kaheks osaks. Kahjuks kõige väiksema prioriteediga “Rakenduse Import/Eksport” ülesanne ei jõudnud seetõttu väljalaske plaani ja see tuleb realiseerida järgmistes lõputöödes.

Tabel 4. Viimane iteratsioonide plaan. Oli korrigeeritud pärast 3.iteratsiooni. Iga värv on eraldi iteratsioon.

Kasutusloo ID	Kommentaar	Prioriteet tegemata tööde listis	Oodatav pingutus	Iteratsiooni lõpp
1002		1	8	24.03.2019
1003	Andmebaas ja tagarakenduse osa	2 (jagatud)	16	7.04.2019
1003	Kasutajaliidese osa	2 (jagatud)	8	
1004	Andmebaas ja tagarakenduse osa	3 (jagatud)	5	21.04.2019
1004	Kasutajaliidese osa	3 (jagatud)	8	
1000		4	3	5.04.2019

Tabel 2-Tabel 4 loodi originaalis Excelis.

4 Arendusprotsess

Selle peatüki eesmärk on kirjeldada, milline funktsionaalsus valiti realiseerimiseks pgApexi teises versioonis ja kuidas tekkisid ideed selle realiseerimiseks. Samuti annab see peatükk ülevaate arenduskeskkonnast ja tööristadest, mida kasutati pgApexi programmi uurimiseks.

4.1 Realiseerimiseks valitud funktsionaalsus

Peatükis 5 esitatakse tehnilist keelt kasutades kirja pannud kasutusjuhud funktsionaalsuse kohta, mis valiti realiseerimiseks jaotises 3.2. Need kasutusjuhud puudutavad järgnevaid funktsionaalsuseid.

- Tabelivormi regioon.
- Detailvaate regioon (sh ka üleminek raport regioonist).
- Alamraport.
- Süsteemse muutuja APP_USER väärtuse kuvamine HTML regioonis ja kasutamine teise parameetrina tabelivormi funktsioonides.

Kasutan kasutusjuhtude formaati, sest võrreldes kasutuslugudega võimaldab see rohkem infot kirja panna. Järgmistes lõputöö peatükkides kasutan ma neid kirjeldusi tehniliste muudatuste kirjeldamiseks.

4.2 Ideed funktsionaalsuse realiseerimiseks

Uue pgApexi versiooni arendusprotsessi käigus võeti vastu otsus “ära hakka jalgratast leiutama”. See tähendab, et uue funktsionaalsuse realiseerimiseks võidakse kasutada olemasolevaid komponente või võidakse need luua analoogselt teiste komponentidega, mille autoriks on Rait Raidma. See lihtsustab arendust ning teeb tundmatud komponente arusaadavaks teistele arendajatele, st toimub standardiseerimine.

Hea näide on tabelivormi regioon. Tegelikult on tabelivorm tavaline raport, kuhu on lisatud lisaveerg märkeruuduga ning lisatud ka nupud, millest igaühele vajutamine käivitab mingi arendaja poolt loodud andmebaasi funktsiooni. Funktsioon käivitatakse iga märgistatud rea korral ning funktsiooni väljakutses antakse ette märgistatud reas olev väärtus (olemi unikaalne identifikaator). Tabelivormi ja raporti tööpõhimõte on sarnane. Raporti regiooni ja tabelivormi regiooni andmebaaside tabelite ülesehitus on väikeste eranditega ühesugune. Nendele vastavate andmebaasi funktsioonide sisu ka on peaaegu samasugune. Funktsioonide erinevus on selles, et need võtavad andmeid erinevatest tabelitest. Seega, milleks on vajalik dubleerida raporti funktsiooni väikeste muudatusega ja teha sellest tabelivormi funktsiooni? Miks mitte teha kahest funktsioonist üks funktsioon? Põhjus on selles, et iga andmebaasi funktsioon peaks vastutama konkreetse ülesande eest ehk tuleb järgida otstarbe lahususe (*Separation of concens*) disainipõhimõtet [19].

Detailvaate regioon on sisuliselt ka tavaline raport, mis näitab andmeid ainult ühest vaate reast ja millel on teistsune mall. Detailvaate regiooni tööpõhimõte sarnaneb raportiga ja tabelivormiga (koosneb veergudest). Sellepärast on ka detailvaatega seotud andmebaasi tabelid ja funktsioonid eelnimetatutega sarnased.

Süsteemne muutuja APP_USER võtab oma väärtuse andmebaasi funktsioonist *f_app_get_setting*, mis loodi pgApexi esimeses versioonis.

4.3 Arenduskeskkond

pgApexi arendus toimus lokaalses arvutis, kus andmebaas ning PHP interpretaator oli käivitatud Ubuntu virtuaalmasinas. Virtuaalmasin loodi Vagrant programmi abil.

4.4 Programmi uurimine

pgApexi arhitektuuri uurimine toimus Xdebug programmi abil, mis on mõeldud PHP koodi silumiseks. Programmi arhitektuuri uurimine oli vajalik uue funktsionaalsuse kirjutamiseks. Detailsemalt kirjutatakse sellest jaotises 2.3.

4.5 Ühiktestimine

Selles projektis testitatakse JavaScript ja PHP koodi. JavaScript programmeerimiskeelt kasutatakse kasutajaliidese osas ja selle testide käivitamiseks on kasutusel Karma [7]. JavaScript ühiktestid testivad komponendide kontrollereid (*controller*) ja teenuseid (*service*).

PHP koodi testimiseks kasutatakse PHP Unit ja Mockery [7] programme. Need testid testivad peamiselt validaatoreid, mis kontrollivad, kas POST-päringute sisu vastab süsteemi nõuetele.

Kokku kirjutasin 19 ühiktesti.

4.6 Intervjuud (arendja rollis) kasutajatega

Viisin läbi kaks intervjuud. Intervjueeritavateks olid kaasüliõpilased informaatika erialalt, kes olid pool aastat varem kasutanud pgApex esimest versiooni oma „Andmebaasid II“ projekti jaoks.Intervjuud toimusid peale pgApex teise versiooni valmimist, sest tahtsin, et küsitletavad näeksid kogu funktsionaalsust. Intervjuude eesmärgiks oli välja selgitada, kas võimalikele kasutajatele meeldib tehtud töö ja kui ei meelid, siis parandada seda selles versioonis (kui see on võimalik) või järgmistes.

Intervjuud toimusid kohtumise vormis. Ma kohtusin kahe inimesega, kummagagi eraldi. Intervjuud käsitlesid ainult uue regioonide loomise protsessi administraatori keskkonnas. Intervjueeritavad said ülesandeks teha pgApexi teises versioonis kaks regiooni (tabelivorm, detailvaade) ja ühe alamaregiooni (alamraport). Nad tegisid ülesandeid iseseisvalt, aga kui nendel tekkisid küsimused, siis ma vastasin. Iga regiooni kohta küsisin ma neli küsimust.

- Kas regiooni (või alamregiooni) loomise protsess oli mugav?
- Kuidas võiks lihtsustada regiooni (või alamregiooni) loomise protsessi?
- Mis oli loomise protsessi ajal positiivne ja negatiivne?
- Mida oli keeruline luua?

Intervjuude lühikokkuvõte on jaotises 7.2. Intervjuude tekstid on Lisa 1 ja Lisa 2. Arvestasin intervjuudes väljapakutud ideedega arendusvaate koostamisel (peatükk 8) ning tegin ka jooksvaid parandusi tarkvaras (jaotis 8.1).

4.7 Tulemuste avaldamine

pgApexi uue versiooni kood on maailmale avaldatud ja on kättesaadav GitHub-is. https://github.com/nikopa96/pgapex2

Selle hoidla näol on tegemist programmi esimese versiooni hoidla hargmikuga (*fork*). Tulevikus ühendatakse see *pull request* abil pgApex algse hoidlaga ja siis on lähtekood leitav sealt: https://github.com/raitraidma/pgapex

Töö lähtekood on litsentseeritud MIT litsentsiga, sest ka esimene pgApex versioon kasutab seda litsentsi.

5 Süsteemi analüüs

Selles peatükis kirjeldatakse pgApexisse lisatavavat funktsionaalsust kasutusjuhtude mudeli ja kontseptuaalse andmemudeli abil. Kasutusjuhtude mudel väljendab nõudeid süsteemi funktsionaalsusele ning kontseptuaalne andmemudel funktsionaalsuse toetamiseks vajatavale andmebaasile. Kasutusjuhtude mudelis kirjutatakse kasutusjuhud lahti kõrgtaseme formaadis. Kasutan kasutusjuhtude mudelit (kasutuslugude kirjapaneku asemel) järjepidevuse huvides. Samasugusel viisil kirjeldati funktsionaalseid nõudeid ka Rait Raidma magistritöös [7]. Samuti võimaldavad need kirja panna rohkem infot kui kasutuslood (vt jaotis 3.2).

Kuna selle lõputöö eesmärgiks oli juba loodud programmi edasiarendamine, siis selles peatükis ei esita ma täielikku ülevaadet pgApexi funktsionaalsusest ja andmebaasist. Detailselt kirjeldatakse pgApexi tarkvara Rait Raidma magistritöös “PostgreSQL andmebaasisüsteemi põhine metaandmetega juhitavate veebirakenduste kiirprogrammeerimise keskkond” [7]. Samuti kirjeldatakse olemasolevat süsteemi antud töö peatükis 2.

Käesolevas töös ei lisata süsteemi uusi pädevusalasid (tegutsejaid e rolle), funktsionaalseid allsüsteeme ning registreid. Selle asemel tehakse muudatusi olemasolevates funktsionaalsetes allsüsteemides ja registrites ning ühtlasi muutub arendaja pädevusala vastutuspiirkond (st see saab uusi ülesandeid).

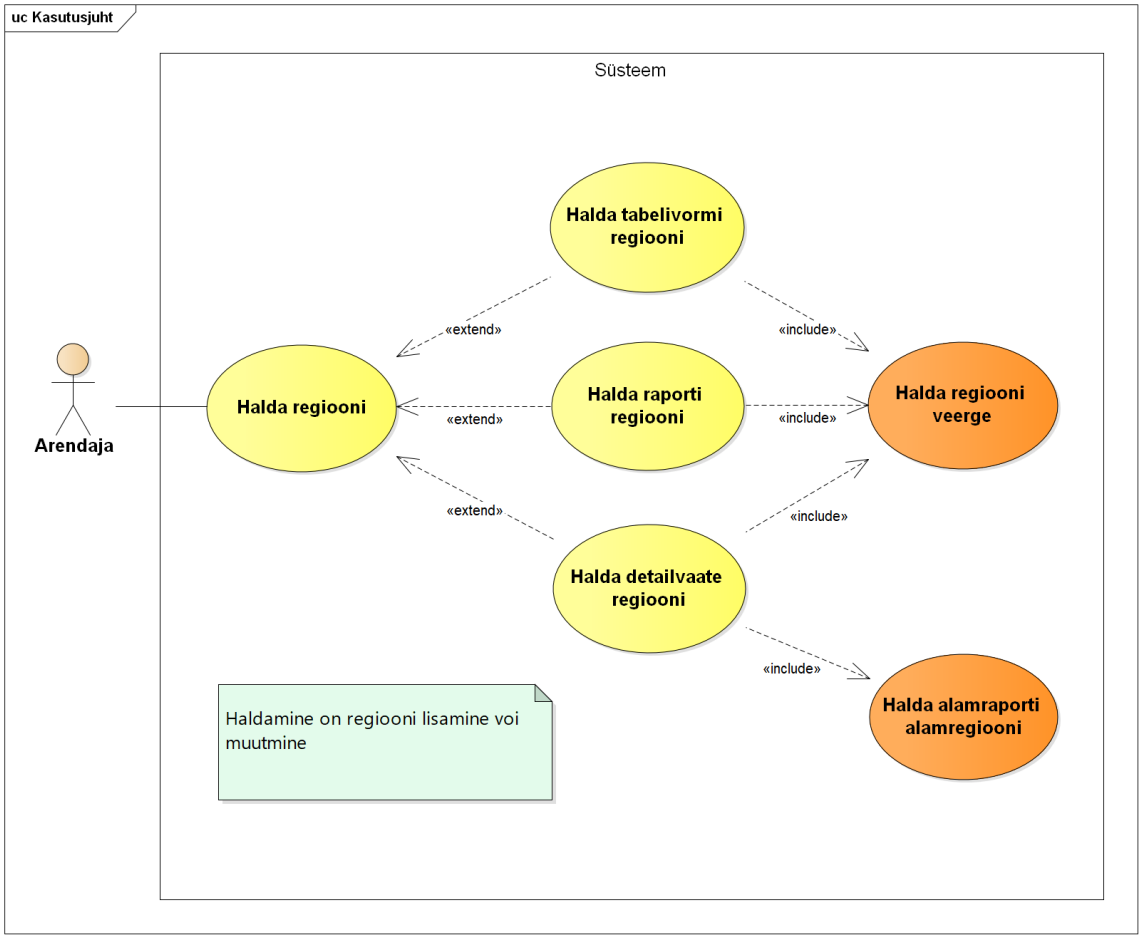
5.1 Regioonide funktsionaalne allsüsteem

Järgnevalt esitatakse kasutusjuhtude mudel, mis kirjeldab regioonide funktsionaalsesse allsüsteemi lisatud funktsionaalsust. Samuti esitatakse kahe registri täiendatud kontseptuaalsed andmemudelid, sest uue funktsionaalsuse toetamiseks peab regioonide funktsionaalne allsüsteem lugema nendest registritest täiendavaid andmeid.

5.1.1 Allsüsteemi kasutusjuhtude mudel

Joonis 4 esitatakse kasutusjuhtude diagramm, mis näitab seoseid tegutsejate ja kasutusjuhtude vahel ning kautusjuhtude omavahelisi seoseid. Kasutusjuht *Halda regiooni* (*Regiooni haldamine*) on kirjeldatud pgApexi esimeses versiooni dokumentatsioonis [7]. Ülejäänud diagrammil kujutatud kasutusjuhud lisandusid teise versiooni.

- Kollasega on tähistatud põhikasutusjuhud.
- Oranžiga on tähistatud alamkasutusjuhud, mida võidakse kasutada mitmes kasutusjuhtus.



Joonis 4. Regioonide funktsionaalse allsüsteemi kasutusjuhtude diagramm (ainult lisatava funktsionaalsuse kohta).

Kasutusjuht: Halda tabelivormi regiooni

Tegutsejad: Arendaja

Kirjeldus: Tabelivormi lisamisel või muutmisel kuvatakse arendajale vorm, millisse ta sisestab andmeid, mida kirjeldatakse *Halda regiooni* kasutusjuhus ning valib tabelivormi malli, andmebaasi vaate (mille põhjal süsteem teeb tabelivormi), valib, kas tabelivormis kuvataval tabelil peab olema päis või mitte, sisestab kui palju ridu peab olema ühel leheküljel ja sisestab paginatsiooni päringu parameetri nime (mille abil teostub minek selle regiooni järgmistele lehekülgedele). Pärast seda peab arendaja valima unikaalse identifikaatori (veeru, milles olevad väärtused antakse nupule vajutamise järel ette tabelivormiga seotud funktsioonile) ja lisama nuppusid, millest igaühele vajutades käivitub andmebaasis loodud funktsioon. Nuppude lisamise või muutmisel valib arendaja nupu malli, funktsiooni, sisestab nupu kuvamise järjekorra, nupu teksti ning funktsiooni täitmise õnnestumise ning ebaõnnestumise teated. Samuti võib kasutaja nupu lisamisel valida, et ta tahab kasutada süsteemse muutuja APP_USER väärtust funktsiooni teise parameetri väärtusena (oluline on parameetri positsioon, mitte nimi).

Kasutusjuht: Halda detailvaate regiooni

Tegutsejad: Arendaja

Kirjeldus: Detailvaate regiooni lisamine või muutmine toimub Raporti ja detailvaate haldamise lehel, kus neid kahte erinevat regiooni hallatakse koos ja nendes olevad andmed seotakse omavahel unikaalse identifikaatori vahendusel. Alguses valib arendaja unikaalse identifikaatori. Pärast seda sisestab arendaja detailvaate loomiseks vormi andmed, mis on esitatud *Halda regiooni* kasutusjuhus. Seejärel valib ta detailvaate malli ning veebirakenduse lehe, kuhu regioon paigutatakse. Samuti peab arendaja valima vaate veerud, millest kirjutatakse *Halda regiooni veerge* kasutusjuhus. Arendajal on ka võimalus teha alamraporteid, millest kirjutatuakse *Halda alamraporti alamregiooni* kasutusjuhus.

Kasutusjuht: Halda alamraporti alamregiooni

Tegutseja: Arendaja

Kirjeldus: Alamraporti alamregiooni lisamisel või muutmisel peab arendaja sisestama vormi alamregiooni nime, kuvamise järjekorra, paginatsiooni päringu parameetri nime, mida kasutakase selle regiooni järgmiste lehekülgede navigeerimiseks ning alamraporti ridade arvu ühel leheküljel. Pärast seda valib ta andmebaasi vaate, mille põhjal tuleb luua

alamraport ja selle vaate veeru, mille alusel alamraportis kuvatakse ainult neid andmeid, mis sisaldavad Detailvaate regiooni unikaalset identifikaatorit (näiteks detailvaate aluseks olevas vaates on veerg *treening_kood* ning alamraportis aluseks olevas vaates on veerg *treening*. Detailvaade ja alamraport seotakse nendes veergudes olevate väärtuste alusel). Edasi loob arendaja veerge, mille loomise protsessi kirjeldab *Halda regiooni veerge* kasutusjuht.

Kasutusjuht: Halda regiooni veerge

Tegutseja: Arendaja

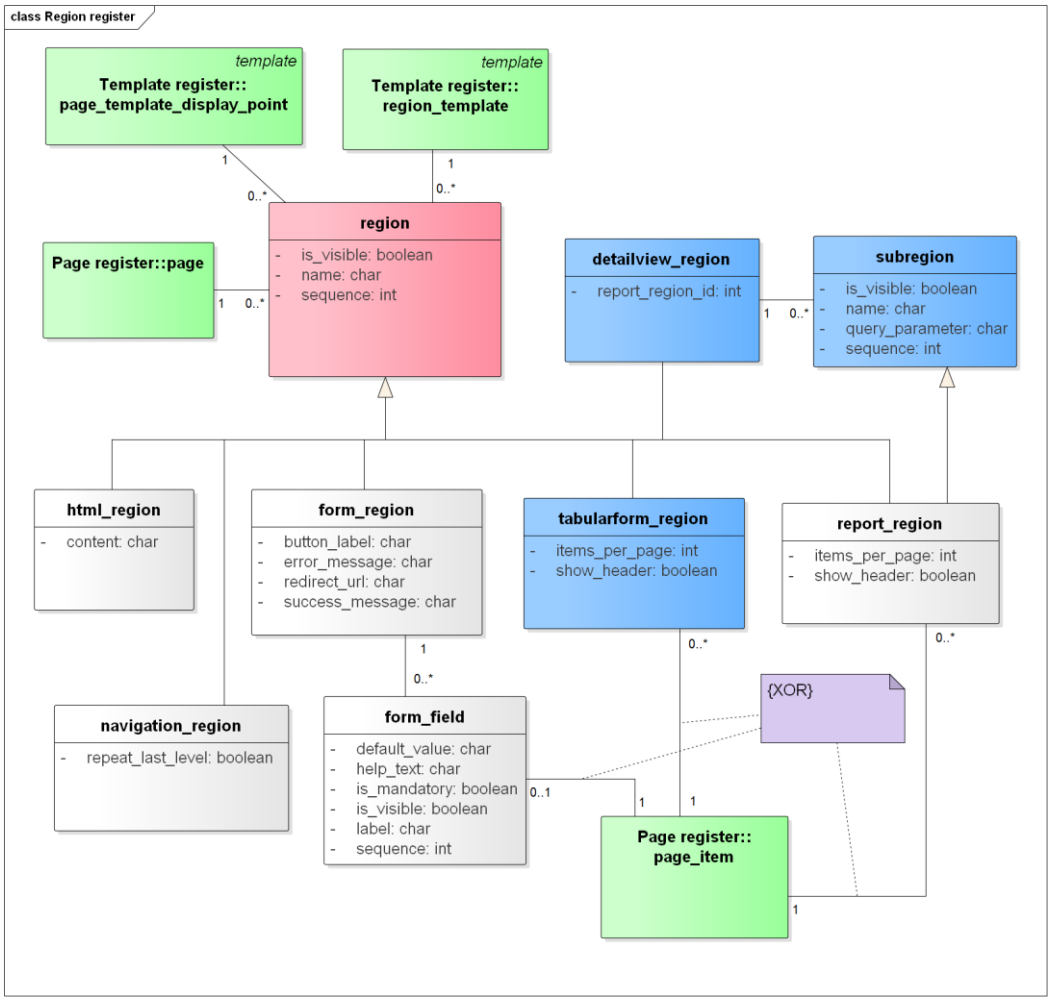
Kirjeldus: Regioonides on kahte liiki veerge: veerg milles olevad andmed pärinevad andmebaasist või lingiga veerg. Esimesel juhul peab arendaja sisestama veeru pealkirja, kuvamise järjekorra, valima vastava andmebaasi vaate veeru ja märkima, kas vaate veerus olevates väärtustes sisalduv HTML kood (kui see on olemas) tuleb rakenduse poolt teisendada tavaliseks tekstiks. Lingiga veeru puhul sisestab arendaja veeru pealkirja, kuvamise järjekorra, HTML koodi teisendamise tavatekstiks ning URL, lingi teksti ja atribuudid.

5.2 Regioonide register

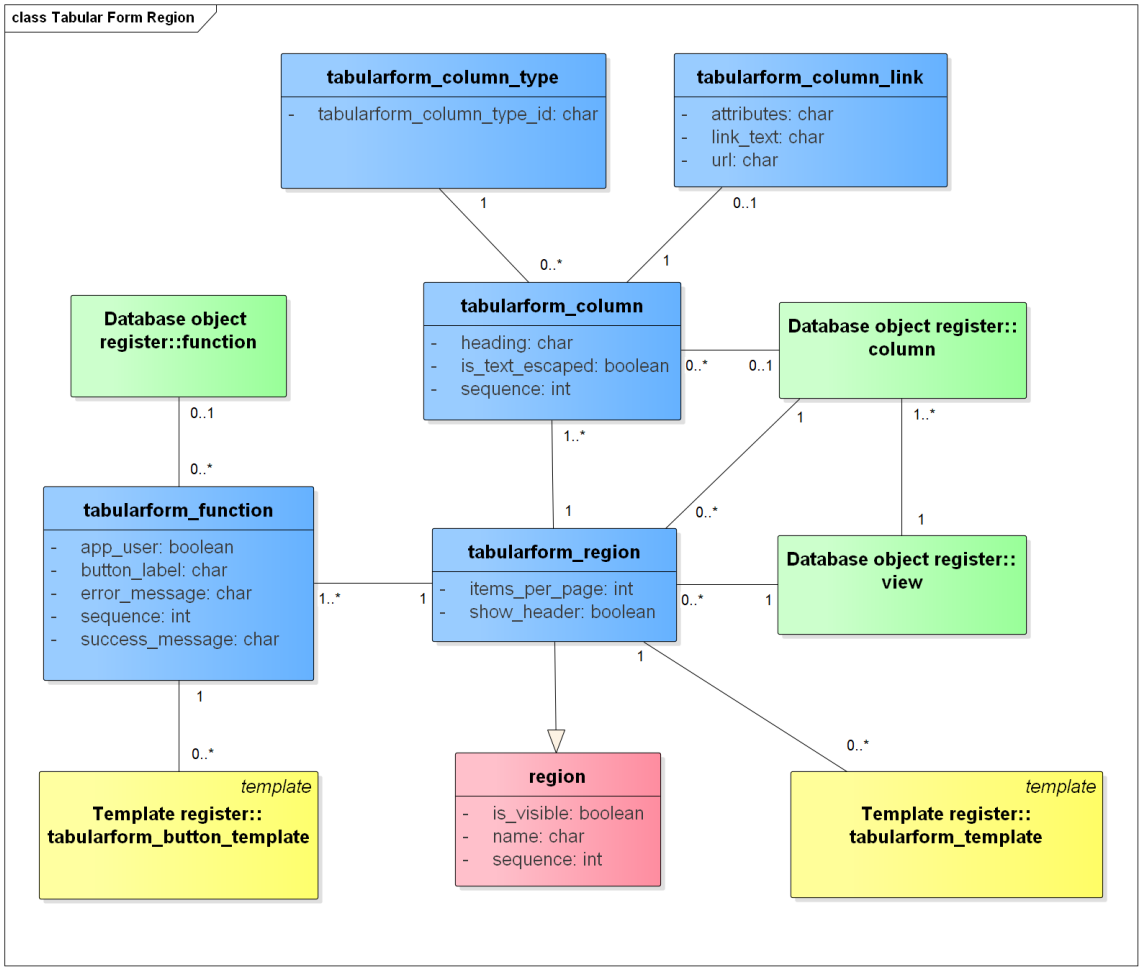
Joonis 5-Joonis 7 esitatakse *Regioonide registri* olemi-suhte diagrammi osad, kus on tehtud põhilised muudatused. Diagrammidel kujutatud elemente kirjeldavad Tabel 5 ja Tabel 6. Kirjeldan vaid neid elemente, mida pgApex teise versiooni kirjeldusse juurde lisasin.

- Punasega on tähistatud *Regioonide registri* põhiolemitüüp.
- Halliga on tähistatud *Regioonide registrisse* kuuluvad mitte-põhiolemitüübid, mida kirjeldati juba pgApexi esimeses versioonis.
- Sinisega on tähistatud *Regioonide registrisse* kuuluvad mitte-põhiolemitüübid, mille kirjeldus lisandus pgApexi teise versiooni.
- Rohelisega on tähistatud olemitüübid, mis ei kuulu *Regiooni registrisse*, aga on vajalikud *Regioonide funktsionaalse allsüsteemi* toimimiseks ja mis kirjeldati juba pgApexi esimeses versioonis.

- Kollasega on tähistatud olemitüübid, mis ei kuulu *Regiooni registrisse*, aga on vajalikud *Regioonide funktsionaalse allsüsteemi* toimimiseks ja mille kirjeldus lisandus pgApexi teise versiooni.



Joonis 5. Regioonide registri olemi-suhte diagramm, osa 1.

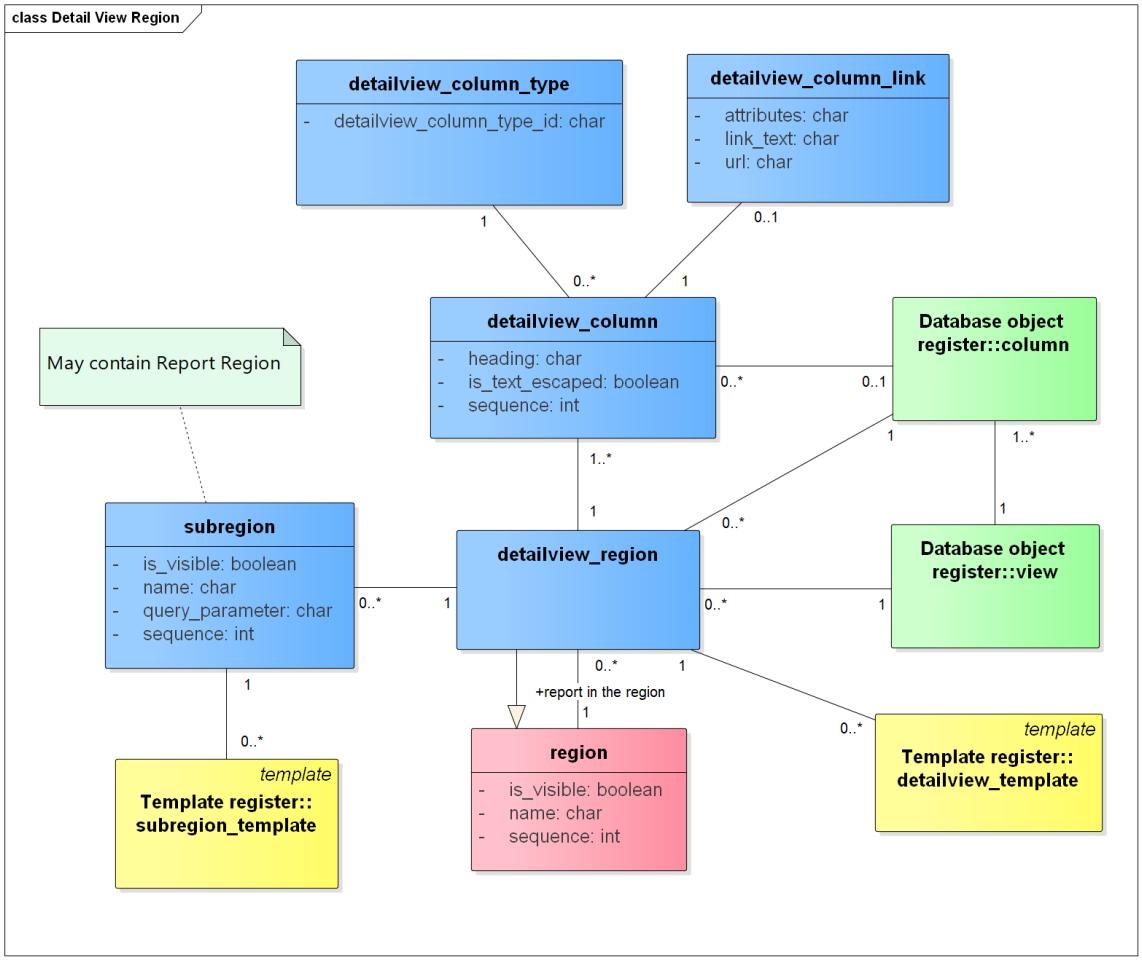


Joonis 6. Regioonide registri olemi-suhte diagramm, osa 2, Tabelivormi regioon.

Joonis 7. Regioonide registri olemi-suhte diagramm, osa 3, Detailvaate regioon.

Tabel 5. Regioonide registri olemitüüpide sõnalised kirjeldused.

Olemitüübi nimi	Definitsioon
detailview_column	Detailvaate veerg.
detailview_column_link	Detailvaate veerg, mis sisaldab linki.
detailview_column_type	Detailvaate veeru tüüp.
detailview_region	Detailvaate regioon. Lehe osa.
subregion	Alamregioon, mis asub Detailvaate regiooni sees ja võib sisaldada alamraportit.
tabularform_column	Tabelivormi veerg.
tabularform_column_link	Tabelivormi veerg, mis sisaldab linki.
tabularform_column_type	Tabelivormi veeru tüüp.
tabularform_function	Tabelivormi funktsiooni, mis käivitatakse nupu vajutamisel.
tabularform_region	Tabelivormi regioon. Lehe osa.



Tabel 6. Regioonide registri atribuutide sõnalised kirjeldused.

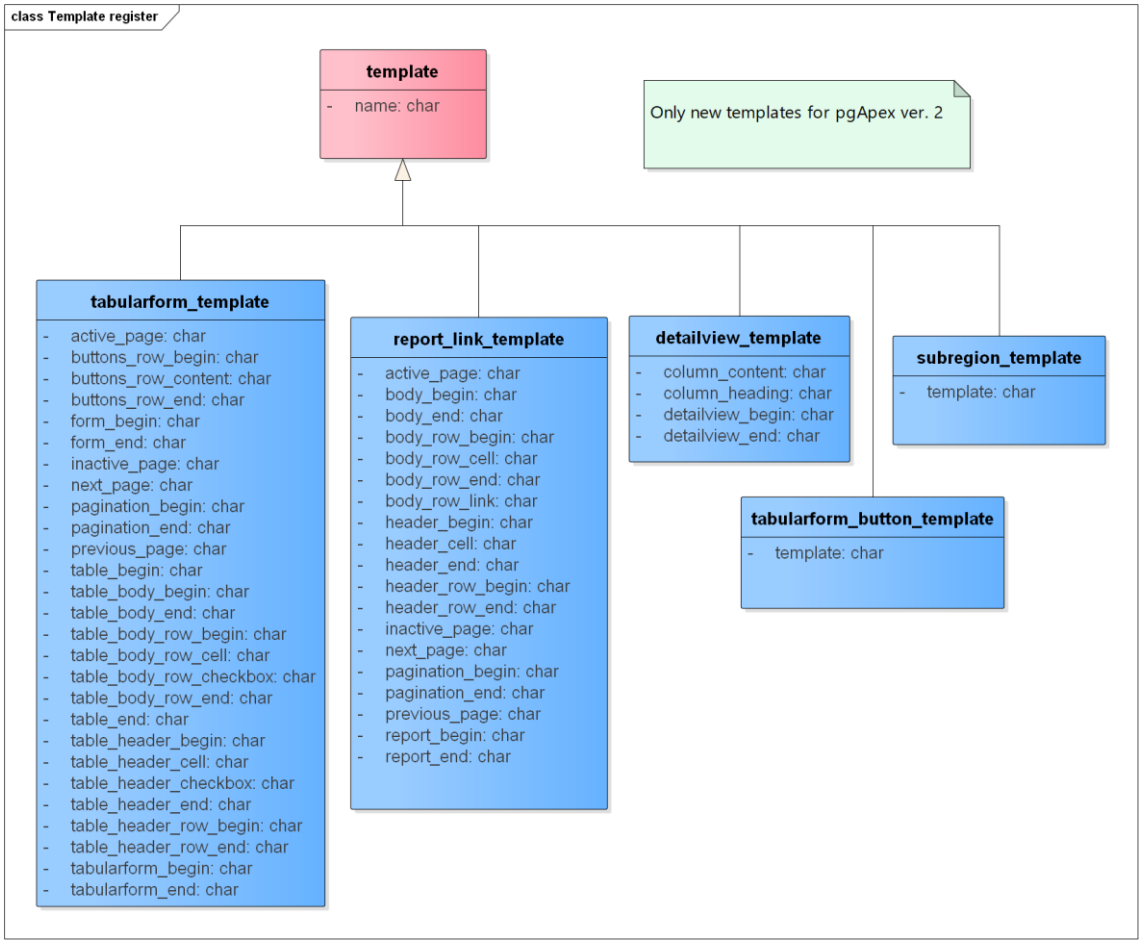
Olemitüübi nimi	Atribuudi nimi	Atribuudi definitsioon
detailview_column	heading	Detailvaate veeru pealkiri.
detailview_column	is_text_escaped	Kas teha detailvaate veeru lahtri sisu ohutuks (HTML koodi teisendamine tavatekstiks)?
detailview_column	sequence	Detailvaate veeru kuvamise järjekord.
detailview_column_link	attributes	Detailvaate lingi atribuut veerus.
detailview_column_link	link_text	Detailvaate lingi kuvatav tekst veerus.
detailview_column_link	url	Detailvaate lingi URL veerus.
detailview_column_type	detailview_column_type_id	Detailvaate veeru tüüp (kas selles on andmed andmebaasi vaate veerust või link). Tegemist on eraldi olemitüübiga, sest tulevikus võivad näiteks

Olemitüübi nimi	Atribuudi nimi	Atribuudi definitsioon
		detailvaates ja tabelivormis olla erinevat tüüpi veerud.
subregion	is_visible	Kas kuvada alamregiooni detailvaate regioonis?
subregion	name	Alamregiooni nimi.
subregion	query_parameter	Päringu parameeter. Võib olla kasutusel alamraporti lehekülgede navigeerimiseks.
subregion	Sequence	Alamregiooni kuvamise järjekord detailvaate regioonis.
tabularform_region	items_per_page	Kui palju tabelivormi ridasid peab olema ühel lehel.
tabularform_region	show_header	Kas kuvada tabelivormi päist?
tabularform_column	heading	Tabelivormi veeru pealkiri.
tabularform_column	is_text_escaped	Kas teha tabelivormi veeru lahtri sisu ohutuks (HTML koodi teisendamine tavatekstiks)?
tabularform_column	sequence	Tabelivormi veeru kuvamise järjekord.
tabularform_column_link	attributes	Tabelivormi lingi atribuut veerus.
tabularform_column_link	link_text	Tabelivormi lingi kuvatav tekst veerus.
tabularform_column_link	url	Tabelivormi lingi URL veerus.
tabularform_column_type	tabularform_column_type_id	Tabelivormi veeru tüüp (kas selles on andmed andmebaasi vaate veerust või link).
tabularform_function	app_user	Kas lisada funktsiooni väljakutsesse teine argument, mille väärtuseks on APP_USER süsteemse muutuja väärtus?
tabularform_function	button_label	Tabelivormi nupul kuvatav tekst.
tabularform_function	error_message	Veateade tabelivormi funktsiooni täitmise ebaõnnestuse puhul.
tabularform_function	sequence	Tabelivormi nupu kuvamise järjekord.
tabularform_function	success_message	Õnnestumise teade tabelivormi funktsiooni täitmise õnnestumise puhul.

5.3 Mallide register

Mall e kujundus, kirjeldab, kuidas andmebaasist pärit sisu mingis regioonis veebilehe kasutajale esitada. Joonis 8 esitatakse *Mallide registri* kontseptuaalne andmemudel, mis kirjeldab selle lõputöö raames tehtud muudatusi ja ei kirjelda kogu allsüsteemi. Diagrammidel kujutatud elemente kirjeldavad Tabel 7 ja Tabel 8. Kirjeldan vaid neid elemente, mida pgApex teise versiooni kirjeldusse juurde lisasin.

- Punasega on tähistatud *Mallide registri* põhiolemitüüp.
- Sinisega on tähistatud *Mallide registrisse* kuuluvad mitte-põhiolemitüübid, mille kirjeldus lisandus pgApexi teise versiooni.



Joonis 8. Mallide registri olemi-suhte diagramm.

Tabel 7. Mallide registri olemitüüpide sõnalised kirjeldused.

Olemitüübi nimi	Definitsioon
detailview_template	Detailvaate mall.

Olemitüübi nimi	Definitsioon
report_link_template	Raporti mall, mis sisaldab eraldi veergu lingiga Detailvaate regioonile.
subregion_template	Alamregiooni mall.
tabularform_template	Tabelivormi mall.
tabularform_button_template	Tabelivormi nupu mall.

Tabel 8. Mallide registri atribuutide sõnalised kirjeldused.

Olemitüübi nimi	Atribuudi nimi	Atribuudi definitsioon
detailview_template	detailview_begin	Detailvaate algus. Sisaldab avavat HTML märgendit <dl>.
detailview_template	detailview_end	Detailvaate lõpp. Sisaldab sulgevat HTML märgendit </dl>.
detailview_template	column_heading	Detailvaate pealkiri. Sõna #COLUMN_HEADING# asendatakse veeru nimega.
detailview_template	column_content	Detailvaate veeru sisu. Sõna #COLUMN_CONTENT# asendatakse veeru sisuga.
report_link_template	report_begin	Raporti algus. Sisaldab avavat HTML märgendit <table>.
report_link_template	report_end	Raporti lõpp. Sisaldab sõna #PAGINATION#, mida asendatakse paginatsiooni käigus.
report_link_template	header_begin	Raporti päise algus. Sisaldab avavat HTML märgendit <thead>
report_link_template	header_row_begin	Raporti päise rea algus. Sisaldab avavat HTML märgendit <tr>.
report_link_template	header_cell	Raporti päise rea kast, kus asub veeru pealkiri. Sõna #CELL_CONTENT# vastutab selle kasti sisu eest.
report_link_template	header_row_end	Raporti päise rea lõpp. Sisaldab sulgevat HTML märgendit </tr>.

Olemitüübi nimi	Atribuudi nimi	Atribuudi definitsioon
report_link_template	header_end	Raporti päise lõpp. Sisaldab sulgevat HTML märgendit </thead>.
report_link_template	body_begin	Raporti sisuosa algus. Sisaldab avavat HTML märgendit <tbody>.
report_link_template	body_row_begin	Raporti sisuosa rea algus.Sisaldab avavat HTML märgendit <tr>.
report_link_template	body_row_link	Raporti sisuosa rea kast, kus asub sõna #PATH#: Detailvaate regiooni asukoht, #UNIQUE_ID#: detailvaate parameeter, selle sõna abil avaneb detailvaade unikaalse sisuga, #UNIQUE_ID_VALUE#: parameetri väärtus.
report_link_template	body_row_cell	Raporti sisuosa rea kast, kus asub sõna #CELL_CONTENT#, mis vastutab kasti sisu eest.
report_link_template	body_row_end	Raporti sisuosa rea lõpp. Sisaldab sulgevat HTML märgendit </tr>.
report_link_template	body_end	Raporti sisuosa lõpp. Sisaldab sulgevat HTML märgendit </tdody>.
report_link_template	pagination_begin	Raporti paginatsiooni algus. Sisaldab avavaid HTML märgendeid <nav> ja .
report_link_template	pagination_end	Raporti paginatsiooni lõpp. Sisaldab sulgevaid HTML märgendeid </nav> ja .
report_link_template	previous_page	Raporti paginatsiooni link eelmisele raporti leheküljele. Sõna #LINK# asendatakse URL lingiga.
report_link_template	next_page	Raporti paginatsiooni link järgmisele raporti leheküljele.

Olemitüübi nimi	Atribuudi nimi	Atribuudi definitsioon
		Sõna #LINK# asendatakse URL lingiga.
report_link_template	active_page	Raporti paginatsiooni link kehtivale raporti leheküljele. Sõna #LINK# asendatakse URL lingiga.
report_link_template	inactive_page	Raporti paginatsiooni link teistele raporti lehekülgedele (v.a hetkel valitud lehekülg). Sõna #LINK# asendatakse URL lingiga. #NUMBER# on selle lehekülje number.
subregion_template	template	Alamregiooni mall. #SUBREGION_TITLE# on alamregiooni pealkiri ja #SUBREGION_BODY# on alamregiooni sisu.
tabularform_button_template	template	Tabelivormi nupu mall.
tabularform_template	tabularform_begin	Tabelivormi algus. Sisaldab avavat HTML märgendit .
tabularform_template	tabularform_end	Tabelivormi lõpp. Sisaldab sõna #PAGINATION# , mida asendatakse paginatsiooni käigus.
tabularform_template	form_begin	Tabelivormi vorm serverile saatmise jaoks. Sõna #TABULARFORM _FUNCTION_ID# on tabelivormi funktsiooni *tabularform_function* unikaalne identifikaator. Sisaldab avavat HTML märgendit <form>.
tabularform_template	buttons_row_begin	Tabelivormi nuppude rea algus. Sisaldab avavat HTML märgendit .
tabularform_template	buttons_row_content	Tabelivormi kinnitamise nupp serveris kirjeldatud malliga.
tabularform_template	buttons_row_end	Tabelivormi nuppude rea lõpp. Sisaldab sulgevat HTML märgendit .

Olemitüübi nimi	Atribuudi nimi	Atribuudi definitsioon
tabularform_template	table_begin	Tabelivormi tabeli algus. Sisaldab avavat HTML märgendit <table>.
tabularform_template	table_header_begin	Tabelivormi tabeli päise algus. Sisaldab avavat HTML märgendit <thead>.
tabularform_template	table_header_row_begin	Tabelivormi tabeli päise rea algus. Sisaldab avavat HTML märgendit <tr>.
tabularform_template	table_header_checkbox	Tabelivormi tabeli päise märkeruut. Kui selline märkeruut on märgistatud, siis on märgistatud kõik selle regiooni märkeruudud sellel leheküljel. Kõikide märkeruutude märgistamine toimub *checkAll* JavaScript funktsiooni abil.
tabularform_template	table_header_cell	Tabelivormi tabeli päise rea kast, kus asub veeru pealkiri. Sõna #CONTENT# asendatakse pealkirja tekstiga.
tabularform_template	table_header_row_end	Tabelivormi tabeli päise rea lõpp. Sisaldab sulgevat HTML märgendit </tr>.
tabularform_template	table_header_end	Tabelivormi tabeli päise lõpp. Sisaldab sulgevat HTML märgendit </thead>.
tabularform_template	table_body_begin	Tabelivormi tabeli sisuosa algus. Sisaldab avavat HTML märgendit <tbody>.
tabularform_template	table_body_row_begin	Tabelivormi tabeli sisuosa rea algus. Sisaldab avavat HTML märgendit <tr>.
tabularform_template	table_body_row_checkbox	Tabelivormi tabeli sisuosa rea märkeruut. #UNIQUE_ID_COLUMN#[] on massiiv märgistatud ridade unikaalsetest identifikaatoritest, mis on võrdne unikaalse sisuga veerus. #UNIQUE_ID_VALUE# on rea

Olemitüübi nimi	Atribuudi nimi	Atribuudi definitsioon
		unikaalne identifikaator, mis on võrdne unikaalse sisuga veerus.
tabularform_template	table_body_row_cell	Tabelivormi tabeli sisuosa rea kast sisuga. #CONTENT# vastutab kasti sisu eest.
tabularform_template	table_body_row_end	Tabelivormi tabeli sisuosa rea lõpp. Sisaldab sulgevat HTML märgendit </tr>.
tabularform_template	table_body_end	Tabelivormi tabeli sisuosa lõpp. Sisaldab sulgevat HTML märgendit </tbody>.
tabularform_template	table_end	Tabelivormi tabeli lõpp. Sisaldab sulgevat HTML märgendit </table>.
tabularform_template	form_end	Tabelivormi vormi lõpp. Sisaldab sulgevat HTML märgendit </form>.
tabularform_template	pagination_begin	Tabelivormi paginatsiooni algus. Sisaldab avavaid HTML märgendeid <nav> ja .
tabularform_template	pagination_end	Tabelivormi paginatsiooni lõpp. Sisaldab sulgevaid HTML märgendeid </nav> ja .
tabularform_template	previous_page	Tabelivormi paginatsiooni link eelmisele tabelivormi leheküljele. Sõna #LINK# asendatakse URL lingiga.
tabularform_template	next_page	Tabelivormi paginatsiooni link järgmisele tabelivormi leheküljele. Sõna #LINK# asendatakse URL lingiga.
tabularform_template	active_page	Tabelivormi paginatsiooni link kehtivale tabelivormi leheküljele. Sõna #LINK# asendatakse URL lingiga.
tabularform_template	inactive_page	Tabelivormi paginatsiooni link teistele tabelivormi leheküljele (v.a hetkel valitud lehekülg). Sõna #LINK# asendatakse URL lingiga. #NUMBER# on selle lehekülje number.

6 Muudatused pgApex arenduskeskkonnas

See peatükk kirjeldab muudatusi, mida tehti selle lõputöö raames pgApexi keskkonna tagarakenduses. Kuna pgApex on metaandmetega juhitav kiirprogrammeerimise vahend, mille puhul hoitakse rakenduste kirjeldusi andmebaasis ning koostatakse kasutajatele lehti andmebaasis loodud funktsioonide abil, siis tähendab tagarakenduses muudatuste tegemine andmebaasiobjektide muutmist.

Käesoleva lõputöö tulemusena saab pgApex keskkonnas luua andmebaasirakendusi, kus kasutatakse tabelivorme, detailvaateid ja alamraporteid detailvaadetes.

HTML regioonides võib kasutada süsteemset muutujat APP_USER, mille väärtuseks on sisselogitud kasutaja kasutajanimi. APP_USER muutuja sisu kuvamiseks peab arendaja HTML regiooni loomisel kirjutama tekstilahtrisse sõna #APP_USER# (sharp märkidega e trellidega). Selle muutuja väärtus peegeldub lehel ainult juhul, kui veebirakenduses kasutatakse autentimist. Lisakse sellele saab APP_USER muutujat kasutada koos tabelivormis andmete muutmiseks kasutatava funktsiooniga – selle väärtus võib olla selle funktsiooni teise parameetri väärtuseks.

Samuti lisasin ühe raporti malli: mall lingiga. Selle malli abil on rakenduse kasutajal võimalus klõpsata raporti veerus lingile ja avada selle kohta käiva detailvaate. Selle tõttu muutus osaliselt funktsioon *f_app_get_report_region_with_template*, mis vastutab raporti regiooni koostamise eest rakenduses.

Kuna rakendus on HTML lehtede kogum, mida koostavad SQL või PL/pgSQL programmeerimiskeeltes kirjutatud funktsioonid, siis detailsemalt saab nende muudatuste kohtalugeda Tabel 10-Tabel 13.

6.1 Tabelid

Siin kirjeldatakse muudatusi tabelites.

6.1.1 Uued tabelid

Tabel 9 esitab selle lõputöö raames loodud tabelite loetelu. Nendes tabelite loodi ka vajalikud kitsendused. Loodud tabelite vajadust kirjeldatakse peatükis 4. pgApex esimese versiooni andmebaasis oli 36 tabelit ja 176 veergu.Käesoleva lõputöö tulemusena loodi 16 tabelit, milles on kokku 119 veergu. Tabelite arv kasvas 41.7 protsenti. Veergude arv kasvas 67.6 protsenti. Käesoleva töö tulemusena loodi 15 indeksit.

Tabel 9. Uued pgApexi keskkonna tabelid ja nende kuuluvus registrisse.

Tabeli nimi	**Kuuluvus registrisse**
tabularform_region	Regioonide register
tabularform_column	Regioonide register
tabularform_column_link	Regioonide register
tabularform_column_type	Regioonide register
tabularform_function	Regioonide register
detailview_region	Regioonide register
detailview_column	Regioonide register
detailview_column_link	Regioonide register
detailview_column_type	Regioonide register
subregion	Regioonide register
subregion_template	Mallide register
report_link_template	Mallide register
tabularform_template	Mallide register
tabularform_button_template	Mallide register
detailview_template	Mallide register
subregion_template	Mallide register

6.1.2 Muudatused olemasolevates tabelites

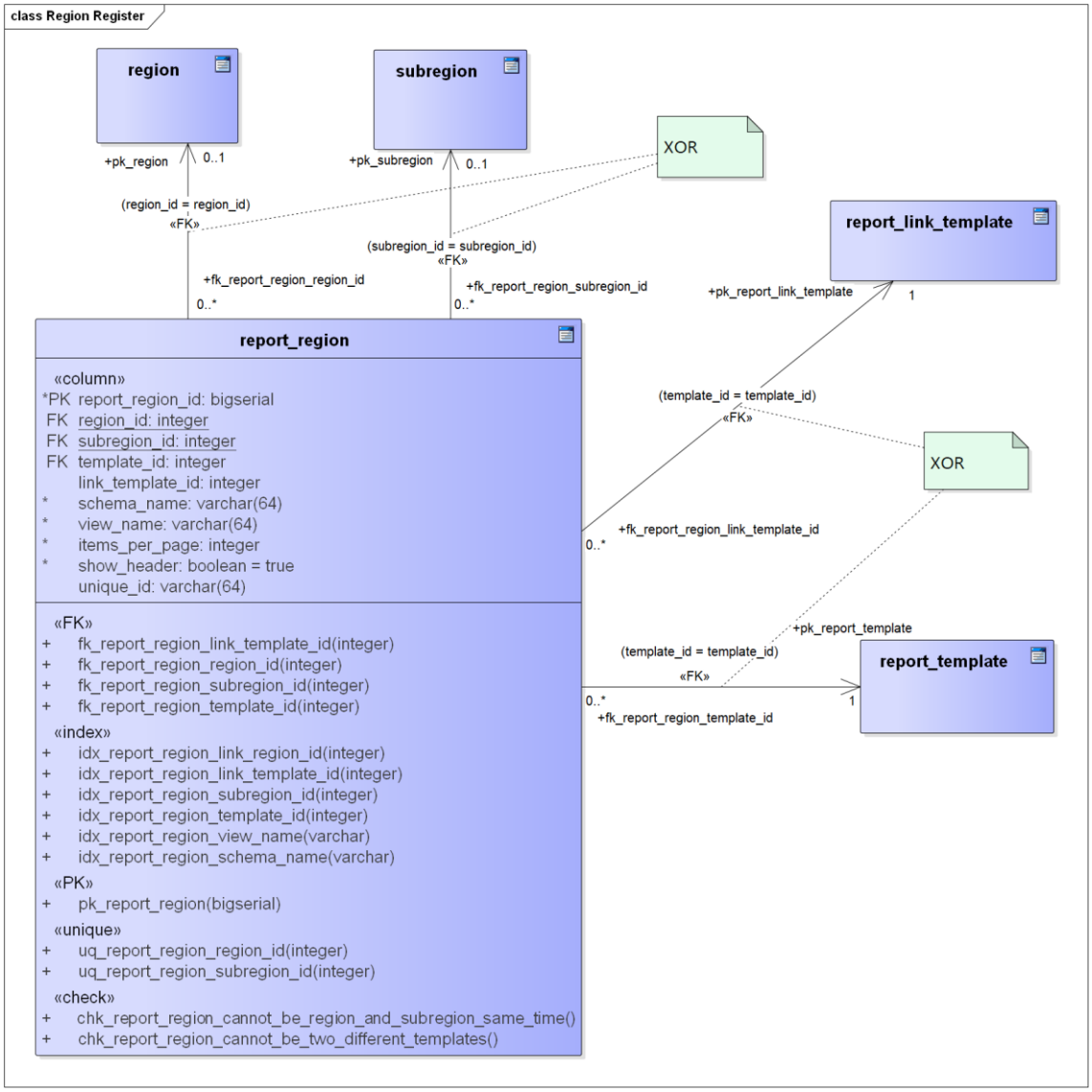
Käesoleva lõputöö käigus muudeti kahte tabelit. Muudatuste käigus lisati viis veergu. Muudatuste tulemusena lisati neli indeksit. Muudatuste tulemusena lisati kolm CHECK kitsendust ja kustutati üks CHECK kitsendus.

Kuna uue pgApexi versiooni eesmärkideks olid detailvaate ja alamraporti loomise võimalikuks muutmine, siis nende realiseerimiseks oli vaja teha muudatusi andmebaasis. Detailvaate valimine toimub raporti abil: see tähendab, et raporti mallis peab olema link detailvaatele ja selle realiseerimiseks loodi uus raporti mall. Alamraport on raport, mille kirjelduse üheks osaks on veeru nimi milles olevad andmed (unikaalsed identifikaatorid) võimaldavad siduda põhiraportis ja alamraportis olevad andmed (näiteks põhiraportis on andmed ühe treeningu kohta, selle alamraportis on andmed selle treeningu kategooriate omamiste kohta ning põhiraport ja alamraport on seotud treeningu koodi kaudu). Raportisse detailvaate lingi lisamise võimaldamine ja alamraporti realiseerimise võimaldamine tähendab *report_region* tabeli (vastutab raporti eest) muutumist.

Raporti lingi võimalikuks tegemiseks kustutati NOT NULL kitsendus veerul *template_id* ja lisati veerg *link_template_id*, mis viitab lingiga raporti mallile. Nüüd peab igas *report_region* tabeli reas olema kohustuslikult registreeritud kas *template_id* või *link_template_id*, aga mitte mõlemad samal ajal. Seda kontrollib CHECK kisendus *chk_report_region_cannot_be_two_different_templates* (Joonis 9). Analoogiline olukord on alamraportitega: iga *report_region rida* peab olema seotud kas reaga tabelis *region* või reaga tabelis *subregion*, aga mitte samaaegselt mõlemaga. Selle eest vastuab CHECK kitsendus *chk_report_region_cannot_be_region_and_subregion_same_time*. Nimetatud kitsendused esitavad kaart (*Arc*), mis realiseerib loogikaoperatsiooni XOR. Kokku loodi *report_region* tabeli jaoks kaks kaare kitsendust ja nende realiseerimiseks kasutati PostgreSQL funktsiooni *num_nonnulls*, mida PostgreSQL toetab alates versioonist 9.6 [20]. Samuti lisati tabelisse *report_region* veerg *unique_id*: selles veerus registreeritakse unikaalne identifikaator, mida alamraport vajab määratud ridade kuvamiseks.

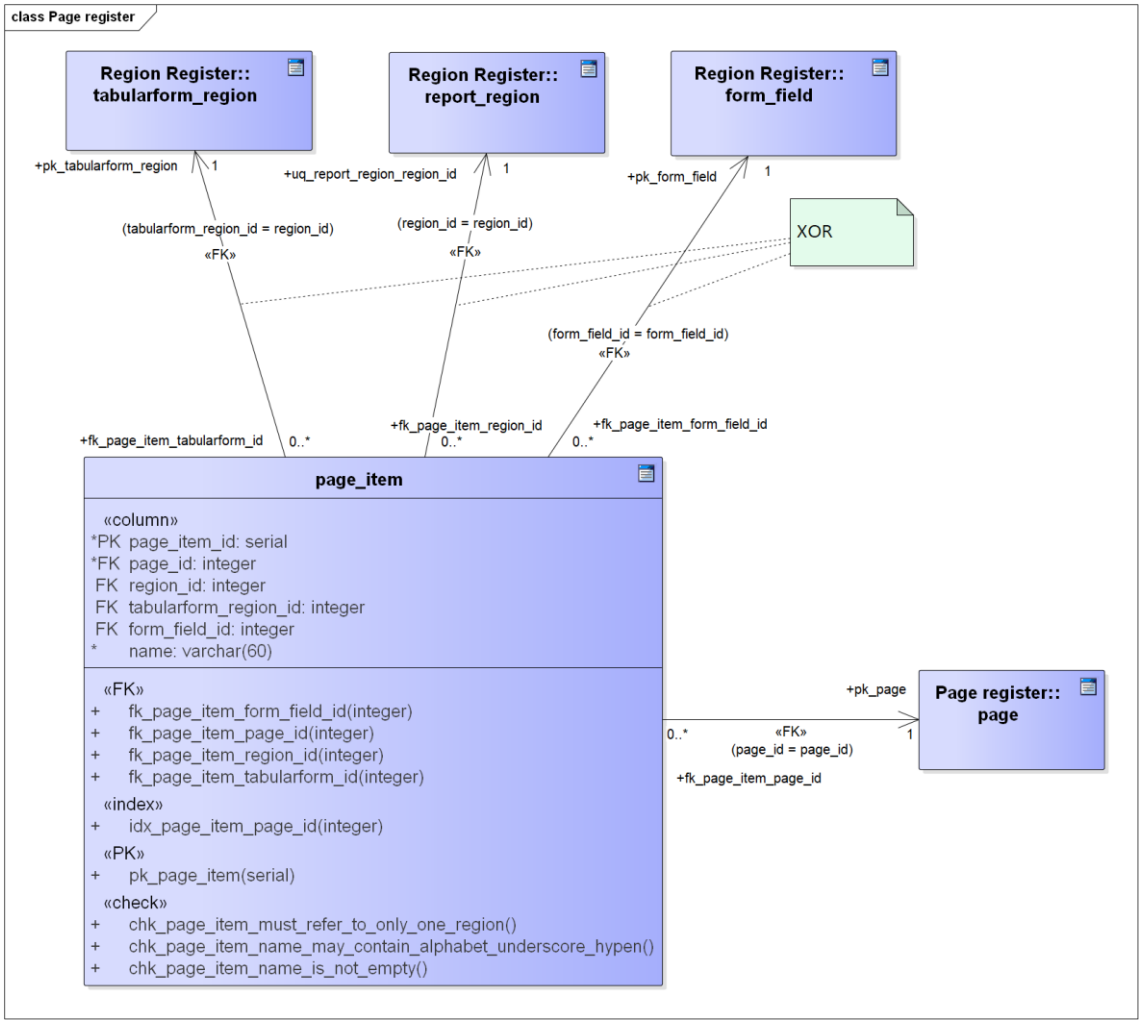
```
CHECK (num_nonnulls(template_id, link_template_id) = 1)
```

Joonis 9. CHECK kitsendus *chk_report_region_cannot_be_two_different_templates*.



Joonis 10. Muudetud Raporti regiooni füüsilise disaini andmebaasi diagramm.

Samuti tehti muudatusi lehtede registri tabelisse *page_item*: lisati veerg *tabelivorm_region_id*, mille abil võib viidata tabelivormi paginatsiooni päringu parameetrile, mis on kasutusel lehekülgede navigeerimiseks. Igas *page_item* tabeli reas peab olema täidetud üks veerg kolmest: kas *region_id*, *tabularform_region_id* või *form_field_id*. Seda tagab CHECK kitsendus *chk_page_item_must_refer_to_only_one_region*. Seda kitsendust muudeti pgApexi teises versioonis ja nüüd kasutab ka see PostgreSQL funktsiooni *num_nonnulls* (Joonis 11).



Joonis 11. Muudatused page_item tabelis, füüsilise disaini andmebaasi diagramm.

6.2 Funktsioonid

Selles jaotises kirjeldatakse uusi funktsioone.

6.2.1 Uued funktsioonid

Tabel 10 ja Tabel 11 nimetavad funktsioonid, mis oli kirjutatud pgApexi teise versiooni funktsionaalsuse realiseerimiseks. pgApex esimese versiooni andmebaasis oli 110 kasutaja defineeritud mitte-trigeri funktsiooni. Käesoleva lõputöö tulemusena loodi 35 funktsiooni, millest 9 on kirjutatud SQL keeles ja 26 on kirjutatud PL/pgSQL keeles. Funktsioonide arv kasvas 31.8 protsenti.

Tabel 10. pgApexi teise versiooni uued funktsioonid, mis on kasutusel administreerimise keskkonna funktsionaalsuse realiseerimiseks.

Nimi	**Kirjeldus**	**Funktsiooni keel**
f_template_get_report_link_templates	Tagastab raporti mallide (lingiga) nimekirja.	SQL
f_template_get_detailview_templates	Tagastab detailvaate mallide nimekirja.	SQL
f_template_get_tabularform_templates	Tagastab tabelivormi mallide nimekirja.	SQL
f_template_get_tabularform_button_templates	Tagastab tabelivormi nupu mallide nimekirja	SQL
f_region_save_tabularform_region	Salvestab tabelivormi andmebaasi.	PL/pgSQL
f_region_create_tabularform_region_column	Salvestab tabelivormi veeru andmebaasi.	PL/pgSQL
f_region_create_tabularform_region_link	Salvestab tabelivormi veeru lingi andmebaasi.	PL/pgSQL
f_region_delete_tabularform_region_columns	Kustutab tabelivormi veerge.	PL/pgSQL
f_region_create_tabularform_region_function	Salvestab tabelivormi nupu funktsiooni andmebaasi.	PL/pgSQL
f_region_delete_tabularform_region_functions	Kustutab ära kõik selle tabelivormi funktsioone.	PL/pgSQL
f_region_save_detailview_region	Salvestab detailvaate andmebaasi.	PL/pgSQL
f_region_create_detailview_region_column	Salvestab detailvaate veeru andmebaasi.	PL/pgSQL
f_region_create_detailview_region_link	Salvestab detailvaate veeru lingi andmebaasi.	PL/pgSQL
f_region_delete_detailview_region_columns	Kustutab ära kõik selle detailvaate veerge.	PL/pgSQL
f_region_save_report_subregion	Salvestab alamraporti andmebaasi.	PL/pgSQL
f_subregion_delete_subregion	Kustutab alamraporti andmebaasist.	PL/pgSQL
f_subregion_create_report_subregion_column	Salvestab alamraporti veeru andmebaasi.	PL/pgSQL

Nimi	Kirjeldus	Funktsiooni keel
f_subregion_create_report_subregion_link	Salvestab alamraporti veeru lingi andmebaasi.	PL/pgSQL
f_subregion_delete_report_subregion_columns	Kustutab ära kõik selle alamraporti veerge.	PL/pgSQL
f_region_get_tabularform_region	Tagastab tabelivormi andmed JSON kujul, et selle alusel vormi kirjeldust sisaldav vorm andmetega täita.	SQL
f_region_get_report_subregions	Tagastab alamraportide andmed JSON kujul automaatseks lõpetamiseks.	PL/pgSQL
f_region_get_detailview_region_id_by_ report_region_id	Tagastab detailvaate identifikaatori raporti regiooni identifikaatori abil, mis on seotud detailvaatega.	PL/pgSQL
f_region_get_report_and_detailview_ region_by_report_id	Tagastab raporti ja detailvaate andmed JSON-kujul raporti identifikaatori abil automaatseks lõpetamiseks.	SQL
f_region_get_report_and_detailview_ region_by_detailview_id	Tagastab raporti ja detailvaate andmed JSON-kujul detailvaate identifikaatori abil automaatseks lõpetamiseks.	SQL

Tabel 11. pgApexi teise versiooni uued funktsioonid, mis on kasutusel veebirakenduse lehtede koostamiseks.

Nimi	Kirjeldus	Funktsiooni keel
f_app_get_parent_region_subregions	Tagastab regiooni alamregioonid.	SQL
f_app_tabularform_region_submit	Käivitab tabelivormi funktsiooni. Loodi funktsiooni *f_app_form_region_submit* põhjal [21].	PL/pgSQL

Nimi	Kirjeldus	Funktsiooni keel
f_app_get_row_count_by_param	Tagastab andmebaasi vaate ridade arvu. Sisaldab neli sisseparameetrit: skeemi nimi, vaate nimi, vaate veeru nimi ja veeru väärtus. Teeb päring kujul: SELECT COUNT(1) FROM skeemi_nimi.vaate_nimi WHERE veeru_nimi = veeru_väärtus.	SQL
f_app_get_detailview_path	Tagastab detailvaate regiooni asukoha.	PL/pgSQL
f_app_get_report_subregion_with_template	Koostab lehele alamraporti. Loodi funktsiooni *f_app_get_report_region_with_template* põhjal [21] .	PL/pgSQL
f_app_get_tabularform_region_with_template	Koostab lehele tabelivormi. Loodi funktsiooni *f_app_get_report_region_with_template* põhjal [21].	PL/pgSQL
f_app_get_tabularform_region	Küsib andmebaasist andmeid *dblink* mooduli abil ja saadab neid funktsioonile, mis koostab tabelivormi. Loodi funktsiooni *f_app_get_report_region* põhjal [21].	PL/pgSQL
f_app_get_report_subregion	Küsib andmebaasist andmeid *dblink* mooduli abil ja saadab neid funktsioonile, mis koostab alamraporti. Loodi funktsiooni *f_app_get_report_region* põhjal [21].	PL/pgSQL
f_app_get_subregions	Koostab alamregioone.	PL/pgSQL
f_app_get_detail_view_with_template	Koostab detailvaate. Loodi funktsiooni *f_app_get_report_region_with_template* põhjal [21].	PL/pgSQL
f_app_get_detail_view	Küsib andmebaasist andmeid *dblink* mooduli abil ja saadab neid funktsioonile, mis koostab detailvaadet. Loodi funktsiooni *f_app_get_report_region* põhjal [21].	PL/pgSQL

6.2.2 Muutunud funktsioonid

Käesoleva lõputöö käigus muudeti seitset funktsiooni. Tabel 12 ja Tabel 13 loetlevad funktsioonid, mida selle töö käigus muudeti.

Tabel 12. pgApexi muutunud funktsioonid, mis on kasutusel administreerimise keskkonna funktsioneerimise tagamiseks.

Nimi	Kirjeldus
f_region_get_display_points_with_regions	Tagastab ühe lehe regioonide nimekirja. **Muudatused: nüüd tagastab ka tabelivorme, detailvaateid ja lingiga raporteid.**
f_region_get_region	Tagastab regiooni kirjelduse, et selle alusel kirjeldust sisaldav vorm andmetega täita. **Muudatused: nüüd tagastab ka tabelivorme, detailvaateid ja lingiga raporteid.**
f_region_save_report_region	Salvestab raporti regiooni andmebaasi. **Muudatused: nüüd salvestab unique_id ja link_template lingiga raporti ja alamraporti korral.**

Tabel 13. pgApexi muutunud funktsioonid, mis on kasutusel veebirakenduse lehtede koostamiseks.

Nimi	Kirjeldus
f_app_create_page	Koostab regioonidega lehte. **Muudatused: nüüd võib koostada tabelivorme ja detailvaateid.**
f_app_parse_operation	Käivitab vormi regiooni funktsiooni andmetega POST-päringust. **Muudatused: nüüd võib käivitada tabelivormi funktsiooni.**
f_app_get_html_region	Koostab HTML regiooni. **Muudatused: nüüd saab aru süsteemsest muutujast APP_USER.**
f_app_get_report_region_with_template	Koostab raporti lehte. **Muudatused: nüüd võib koostada ka lingiga raportit.**

6.3 Tüübid

pgApex esimese versiooni andmebaasis oli kasutaja-defineeritud tüüpi. Tabel 14 estitatakse kasutaja-defineeritud tüübid, mis lisati pgApexi teise versiooni andmebaasi funktsioonide töötamise võimaldamiseks.

Tabel 14. Funktsioonide töö tagamiseks pgApexi andmebaasi lisatud tüübid.

Nimi	**Kirjeldus**
t_column_with_link	Veerg lingiga.
t_tabularform_button	Tabelivormi nupp.

6.4 Muudatused arenduskeskkonna kasutajaliideses

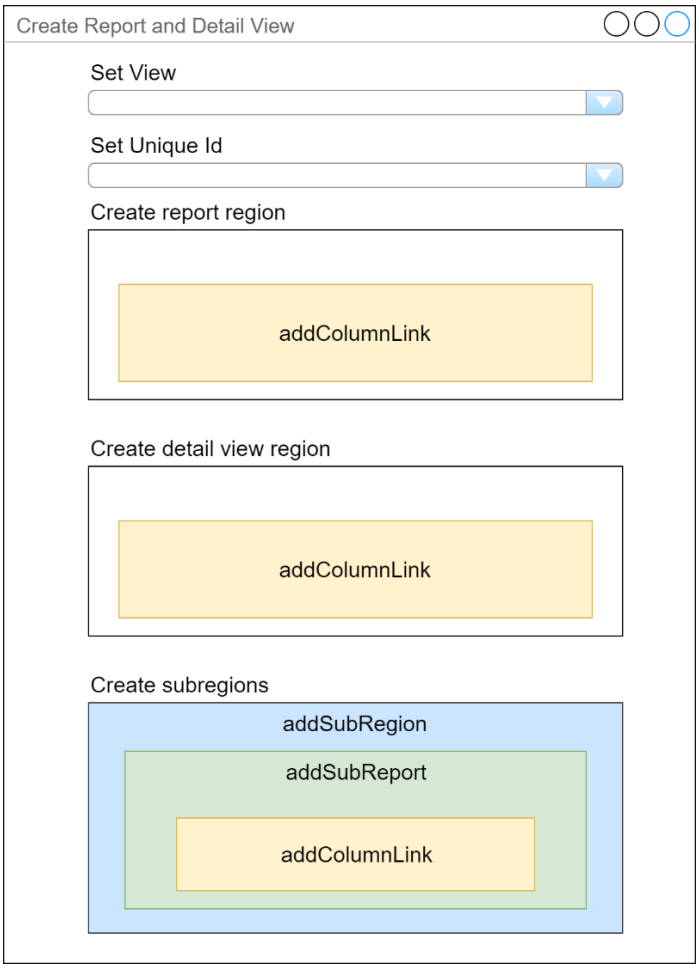
Uude pgApexi versiooni lisandusid tabelivormid, detailvaated ning alamraportid. Nende haldamine (lisamine, muutmine, kustutamine) toimub administreerimise keskkonnas ehk arendusvahendis. Selle jaoks loodi eraldi HTML lehed ja kontrollerid AngularJS raamistikus.

Jaotises 4.2 öeldakse, et tabelivormidel, detailvaadetel ning raportitel on palju ühist – need koosnevad veergudest. See tähendab, et tabelivormi, detailvaate ja alamraporti loomise lehtedel administreerimise keskkonnas peab olema võimalus kirjeldada veerge. Vältimaks HTML ja JavaScript koodi dubleerimist ühendati samasugused korduvad lehtede osad (nt. veergude loomine) direktiivideks (*Directives*). Direktiiviks on AngularJS v1.4.0 raamistikus DOM-elemendile lisatud märgend, mis annab raamistiku HTML kompilaatorile teada, et selle elemendiga on vaja siduda spetsiifiline käitumine või et näiteks elementi on vaja teisendada [22]. pgApexi administreerimise keskkonnas kasutatakse direktiive korduvate elementide ühendamiseks ning keeruliste elementide isoleerimiseks. Direktiivide kasutamine lihtsustab arendajate tööd, sest see teeb lehtede HTML koodi lihtsamaks ning muudatus direktiivis kajastub kõikides HTML lehtedes, mis seda direktiivi sisaldavad.

Selle lõputöö raames loodi neli direktiivi (Tabel 15). Iga direktiivi jaoks kirjutati HTML mall ja kontroller. Lisaks sellele suhtleb iga direktiivi kontroller lehe ülemkontrolleriga. Esimeses pgApexi versioonis direktiive ei kasutatud. Joonis 12 kirjeldatakse raporti ja detailvaate loomise lehe struktuuri. Sellel lehel kasutatakse *addColumnLink* direktiivi. Keerulised elemendid on isoleeritud direktiivideks *addSubRegion* ja *addSubReport*.

Tabel 15. Loodud direktiivid pgApexi 2. versioonis, nende kontrollerid, kirjeldus ja kasutamine lehtedel.

Direktiivi nimi	**Direktiivi kontroller**	**Kirjeldus**	**Kasutamine lehtedel**
addColumnLink	AddColumnLinkController	Lisab vaate veeru või veeru lingiga.	Tabelivormi loomine, raporti ja detailvaate loomine.
addButton	AddButtonController	Lisab nupu, mis käivitab funktsiooni	Tabelivormi loomine.
addSubRegion	AddSubRegionController	Komponent, kus asuvad erinevad alamregioonid. Selles versioonis see hoiab ainult alamraporteid, aga tulevikus võivad siin olla salvestatud ka alamvormid ja teised alamregioonid.	Raporti ja detailvaate loomine.
addSubReport	AddSubReportController	Selle komponendi sees toimub alamraporti loomine, mis omakorda sisaldab addColumnLink direktiivi.	Raporti ja detailvaate loomine.



Joonis 12. Raporti ja detailvaate loomise lehe struktuur.

6.5 Ühiktestimine

Ühiktestid kirjutati selle lõputöö raames loodud validaatorite kontrollimiseks: *TabularFormRegionValidator* ja *ReportAndDetailViewValidator*. Validaatorite ühiktestid kirjutati kasutades PHPUnit tarkvara ja sõltuvuse sisestamise (*Dependency Injection*) imiteerimiseks kasutatakse Mockery tarkvara.

Kõik PHP ühiktestid on võimalik käivitada, sisestades Vagranti virtuaalmasina terminalis käsu

`php "pgapex/vendor/phpunit/phpunit/phpunit"` kaustas `/vagrant`.

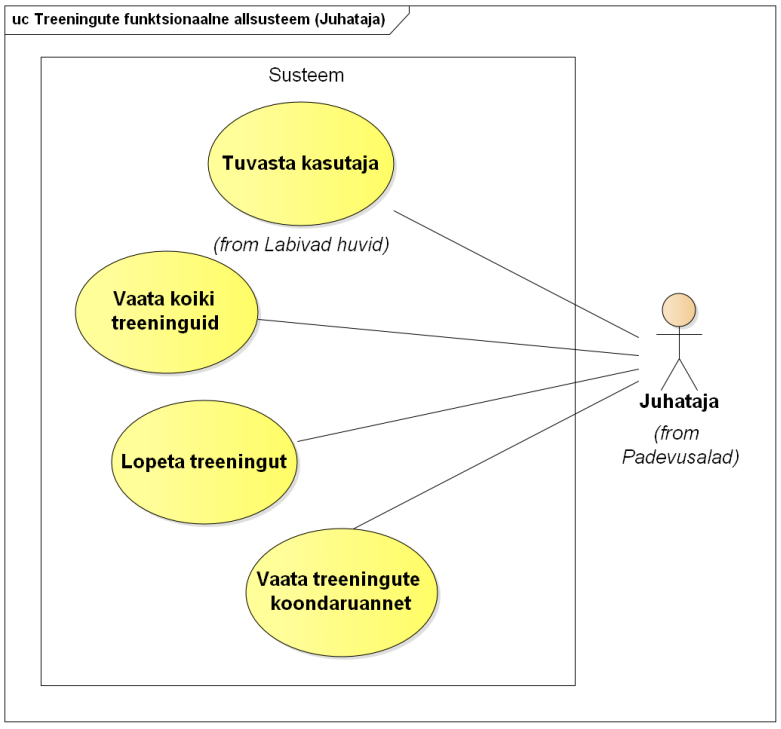
7 Valideerimine

Selles peatükis kirjutan ma töö tulemuste valideerimisest.

7.1 Näidisrakendus

Uue pgApexi versiooni valideerimiseks lõin ma rakenduse, mis kasutab kogu selle lõputöö raames realiseeritud uut funktsionaalsust. Loodud rakendus põhineb minu "Andmebaasid II" aineprojektil [8] ja realiseerib seal juhataja töökoha. Rakenduse realiseerimiseks vajalik andmebaas, sh andmebaasi avaliku liidese realiseerivad vaated ja funktsioonid, realiseerisin juba aineprojektis (koos kahe kaasautoriga). Näiterakenduse aadress on: `http://apex.ttu.ee/pgapex2/public/index.php/app/sport_club/homepage` (kasutajanimi: *louella.henson@colaire.com*, parool: *VZC%#88e*).

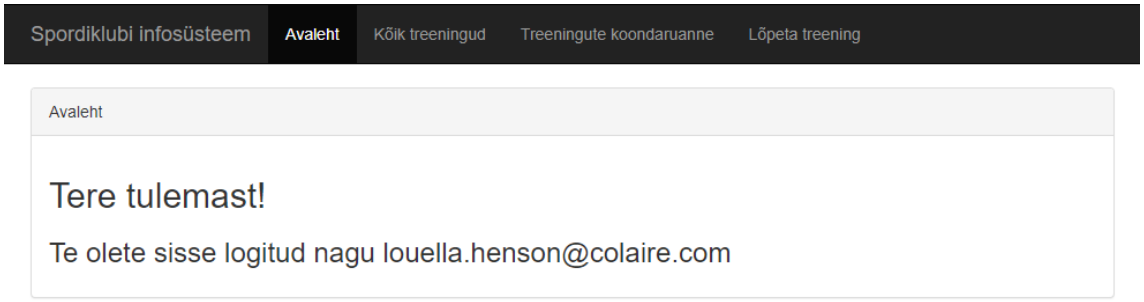
Selles rakenduses peab juhataja rakenduse kasutamiseks sisse logima. Ta võib vaadata kõiki treeninguid (sh nende detaile), saab treeninguid lõpetada ning saab vaadata treeningute koondaruannet.



Joonis 13. Treeningute funktsionaalse allsüsteemi kasutusjuhtude diagramm.

7.1.1 Kasutaja tuvastamine

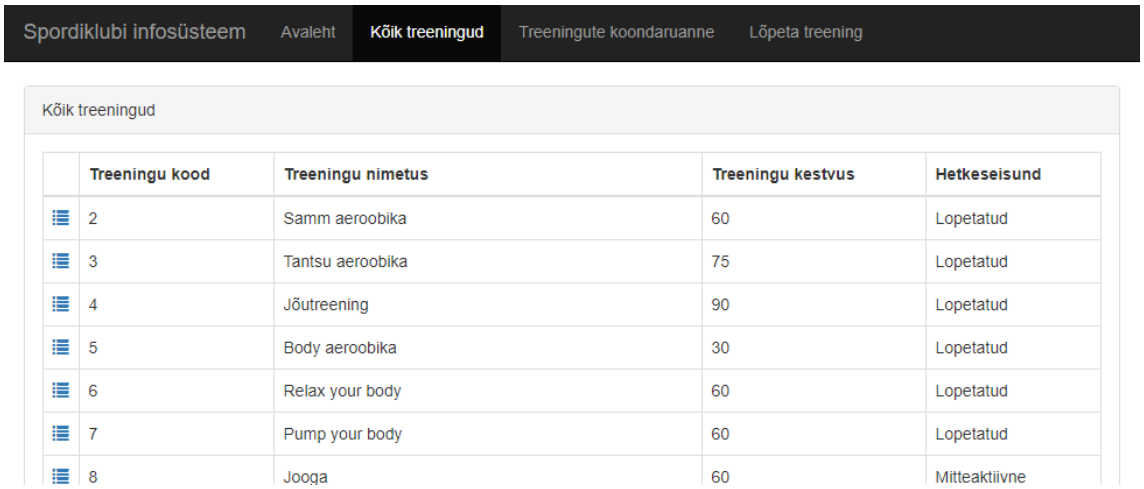
Treeningute juhtimissüsteem nõuab, et kasutaja tuleb autentida. Kasutaja sisestab autentimise lehel oma kasutajanime ja parooli ning arendaja poolt andmebaasis loodud funktsioon *f_on_juhataja* tuvastab kasutajat. Kui kasutaja tuvastatakse ja tegemist on juhatajaga, siis on ta autoriseeritud süsteemi kasutama. Kasutaja satub kodulehele ja näeb HTML regioonis kirja "Tere tulemast! Te olete sisse logitud nagu louella.henson@colaire.com" (Joonis 14). See regioon kasutab kasutajanime kuvamiseks süsteemset muutujat APP_USER. Antud juhul louella.henson@colaire.com on kasutajanimi ja see väärtus on omistatud APP_USER muutujale.



Joonis 14. Rakenduse avaleht. HTML regioon kuvab süsteemset muutujat APP_USER.

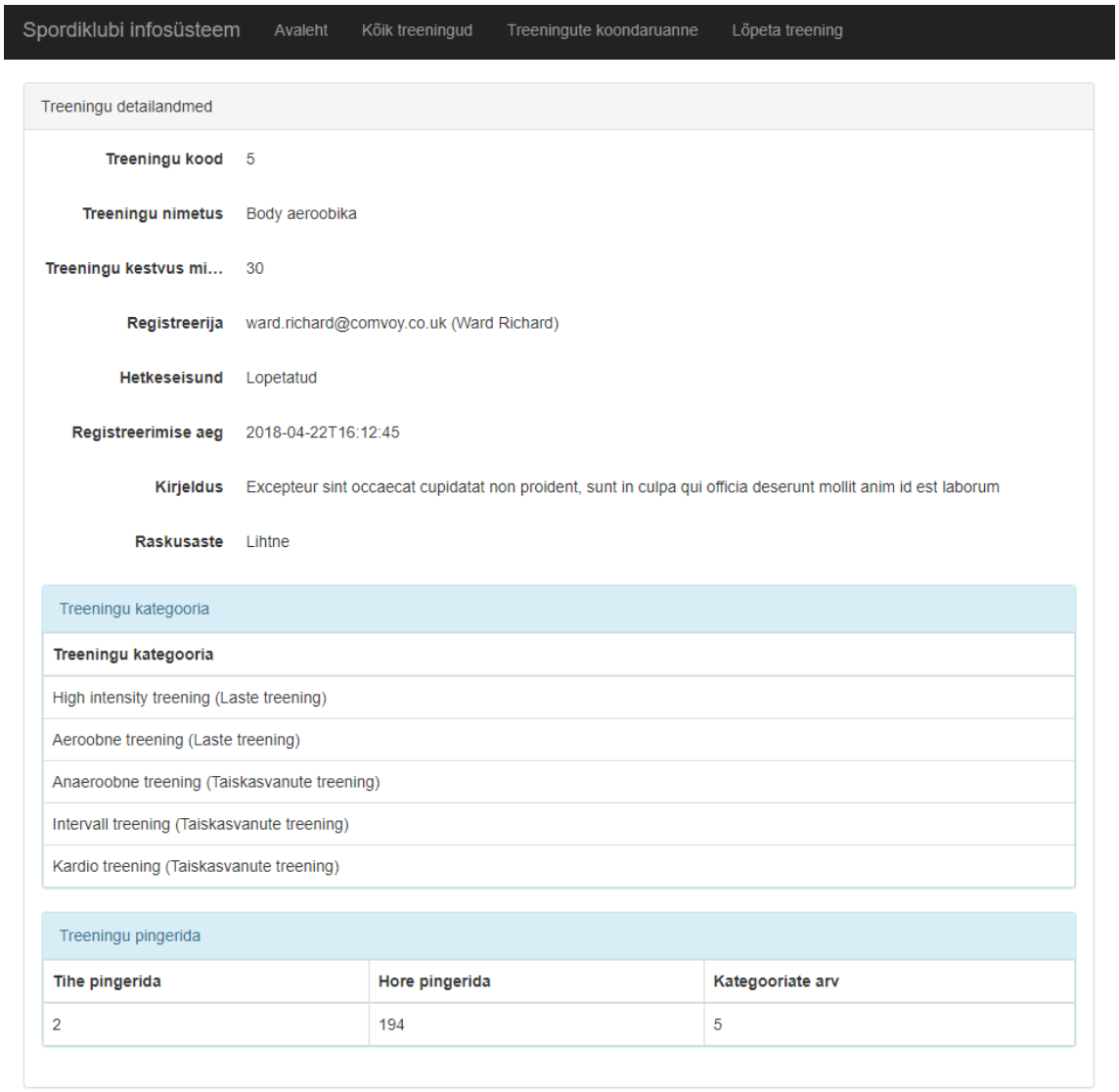
7.1.2 Kõikide treeningute vaatamine

Kui kasutaja avab lehe "Kõik treeningud", siis ta näeb raportit linkidega (Joonis 15). Raport esitab infot olemasolevate treeningute kohta. Klõpsates sinisele nupule esimeses raporti veerus jõuab ta järgmisele rakenduse lehele ja näeb detailvaadet, kus on esitatud detailandmed valitud treeningu kohta ning selle treeningu kategooriad ja pingerida (Joonis 16). Nii raport kui ka detailvaade võtavad infot andmebaasi vaatest nimega *treeningute_detailid*.



Joonis 15. Rakenduse leht “Kõik treeningud”.

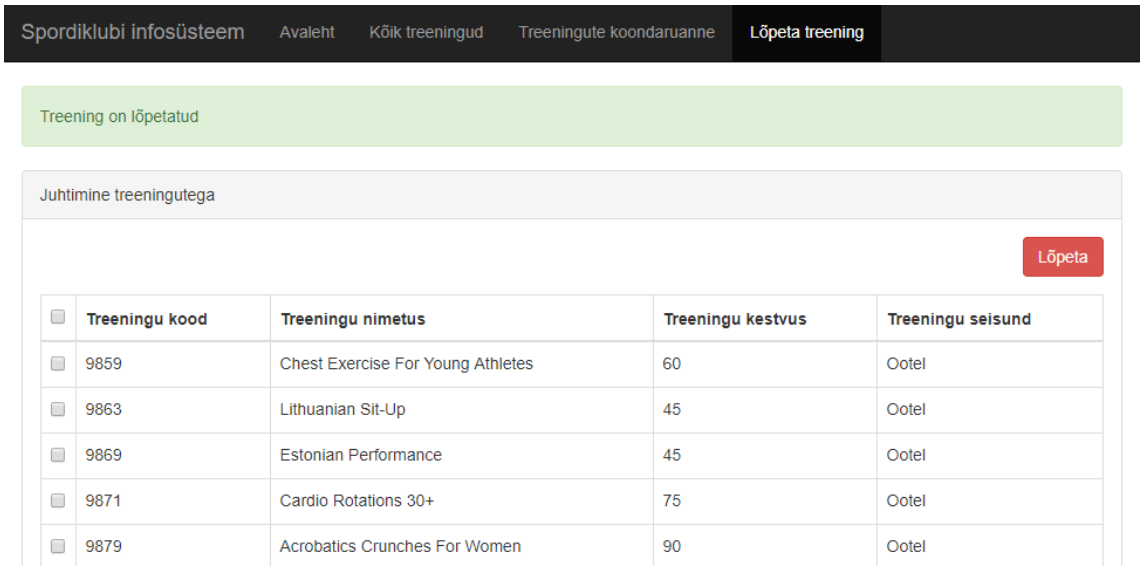
Treeningu kategooria ja treeningu pingerida on alamraportid, mis kuvavad ainult valitud treeningu kohta käivaid andmeid. Detailvaates ja alamraportites olevate andmete sidumiseks kasutatakse andmeid veerust, mis on detailvaates oleva iga rea unikaalseks identifikaatoriks (antud juhul tellimuse koodi).



Joonis 16. Treeningu detailandmeid, mida kuvatakse detailvaate ja alamraportite abil.

7.1.3 Treeningute lõpetamine

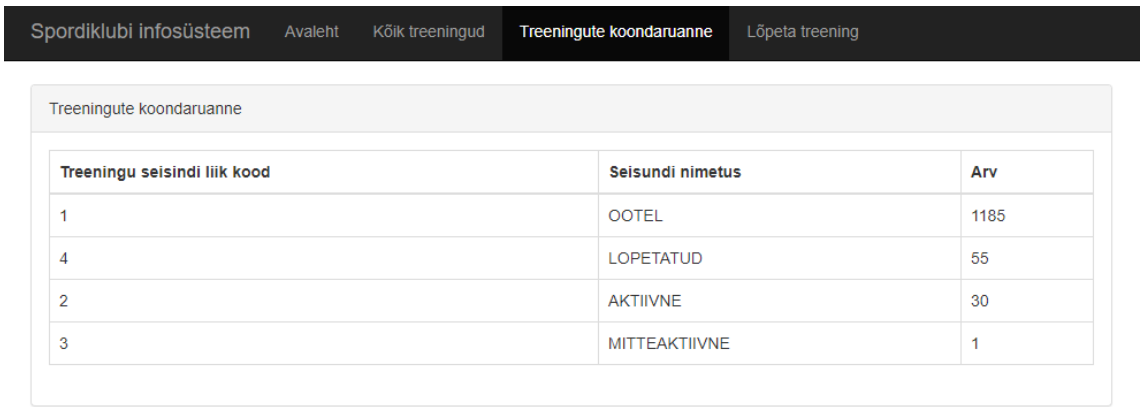
Lehel “Lõpeta treening” näeb kasutaja tabelivormi ühe nupuga nimega “Lõpeta”. Vajutades sellele nupule käivitab kasutaja kahe sisseparameetritega funktsiooni *f_lopeta_treening*, mis on tehtud andmebaasis arendaja poolt. See funktsioon lõpetab treeningut muutes treeningu seisundit ning ühtlasi registreerib treeningu lõpetaja. Tabelivormi olevad andmed leitakse andmebaasi vaatest *koik_treeningud*. Kui funktsioonil õnnestus treeningu lõpetamine, siis lehele ilmub õnnestumise kohta sõnum rohelises ristkülikus (Joonis 17). Kasutajal on korraga võimalik valida mitu treeningut ning anda nende kohta korraga lõpetamise korraldus.



Joonis 17. Treeningute lõpetamine lehel "Lõpeta treening".

7.1.4 Treeningute koondaruande vaatamine

Lehel "Treeningute koondaruanne" asub raport, mis näitab nõutud andmeid (iga võimaliku treeningu seisundi kohta selles olevate treenigute arvu) (Joonis 18). Raportid olid juba olemas Rait Raidma magistritöö tulemuses ja selles lõputöös nende funktsionaalsus ei muutunud. Kuid osaliselt muutusid andmebaasi funktsioonid, mis vatavad raportide esitamise eest. Sellepärast oli valideerimiseks vajalik ka tavalise raporti loomine.



Joonis 18. Raporti regioon, mis näitab treeningute koondaruannet.

7.2 Intervjuud

Intervjueerisin kahte kaasüliõpilast, kes võiksid olla sellises süsteemis arendaja rollis (vt jaotis 4.6). Intervjuude tekstid on Lisa 1 ja Lisa 2.

Mõlemad kasutajad vastasid, et regioonide ja alamregiooni loomise protsess oli mugav. Kasutajad pakkusid erinevaid meetodeid kuidas lihtsustada regioonide loomist. Esimene pakkus, et süsteem võiks automaatselt genereerida veergude järjekorra ning paginatsiooni päringu parameetri. Teine pakkus, et raporti ja detailvaate loomise lehel tuleks lihtsustada kasutajaliidest – jagada raporti, detailvaate ja alamregiooni loomine sakkidega kaartide vahel.

Kui rääkida keskkonna positiivsetest ja negatiivsetest külgedest, siis teisel kasutajal tekkis detailvaate loomise ajal viga (*Bug*) ja talle see loomulikult ei meeldinud. Teistes funktsionaalsustes oli tema hinnangul kõik positiivne. Mõlemad kasutajad vastasid, et regioonide / alamregiooni loomise protsess ei olnud keeruline, sest kogu süsteemis on kõik vormide elemendid (tekstilahtrid, mida on vaja täita) samad.

8 Arendusvaade

Minu arvates on järgmistes pgApexi versioonides vaja realiseerida mitte ainult uut funktsionaalsust. Muudatusi tuleb teha ka programmi "kapoti all". Järgmiseks esitatakse minu nägemus süsteemi edasistest arendusvajadustest, mis käsitleb nii funktsionaalsust kui ka tehnilist (programmi ülesehituse) osa.

Kui rääkida uuest funktsionaalsusest, siis pakun järgmiseid ideid.

- **Uued tabelivormide veergude tüübid.** Praegu võib tabelivormides luua kahte tüüpi veerge: veerg tekstiga ja lingiga. Tulevikus võiksid ilmuda ka uued veergude tüübid, nagu näiteks tekstiala, liitboks ja kalender.
- **Raport ja vorm.** Teises pgApexi versioonis ilmusid Raport (aruanne) ja Detailvaade, kus esimesel lehel asub raport linkidega ja klõpsates lingile avaneb teine leht detailvaatega. Sama asja võiks teha vormidega: kasutaja klõpsab raporti reale ja uuel lehel avaneb vorm (detailvaate asemel), kus kasutaja sisestab mingeid andmeid (näiteks muudab valitud olemi andmeid).
- **Raport ja detailvaade.** Nii raport kui detailvaade peavad praegu põhinema samal andmebaasi vaatel. Tulevikus võiks paindlikuse huvides lubada nende põhinemist erinevatel vaadetel. Siiski on oluline, et mõlemas vaates oleks sama unikaalne identifikaator, mille alusel saaks raportist tehtud valiku alusel avada lehe valitud olemi detailvaatega.
- **Alamvormi alamregioon.** Need võiksid olla realiseeritud analoogselt alamraportiga: tavaline vorm, mis asub alamregioonis.
- **Rakenduse import ja eksport.** Kahjuks ei jõudnud ma seda teises versioonis realiseerida.
- **Tabelivormide funktsioonid mitme parameetritega.** Praegu peab tabelivormide funktioonidel olema maksimaalselt üks sisseparameeter. Erandiks on funktsioonid, mis kasutavad sisendina süsteemse muutuja APP_USER väärtust, kus võib olla teine sisseparameeter. Tulevikus võiksid tabelivormid osata käivitada fuktsioone, millel on suurem hulk parameetreid.

- Pakkuda raportites automaatselt **otsimise** funktsionaalsust.
- Võimaldada kasutada rakendusse sisseloginud kasutaja kasutajanime (süsteemse muutuja APP_USER väärtus) vormides funktsiooni väljakutsel ühe argumendina ning raportites. Raportites kasutamine tähendaks seda, et kui raporti taga on päring vaate V (kus on veerud v1, v2, v3, v4) põhjal (Joonis 19):

```
SELECT v1, v2, v3 FROM V;
```

Joonis 19. Päring vaate põhjal.

, siis kasutajale lehe kuvamiseks täidetakse lause (Joonis 20):

```
SELECT v1, v2, v3 FROM V WHERE v4=<APP_USER väärtus>;
```

Joonis 20. Täiendatud päring vaate põhjal.

Näiteks kui klient logib süsteemi, siis näeb ta ainult enda tehtud tellimusi, mitte kõiki tellimusi.

- Võimaldada rakendusel kasutada **hetktõmmiseid** e materialiseeritud vaateid ning andmebaasis loodud **protseduure**. Hetkel võimaldab rakendus andmete lugemiseks kasutada vaateid ning muutmiseks funktsioone.

Intervjuus võimalike pgApexi kasutajatega tekkisid uued ideed, mida võiks samuti realiseerida järgmistes väljalasetes.

- **Regioonide loomise lehel vormi võiksid väljad Järjekord (*Sequence*) täituda automaatselt.** Samuti võiks süsteem täita ka täita paginatsiooni päringu parameetri (*Pagination query parameter*) välja. Need ideed ma realiseerisin peale intervjuud (vt jaotis 8.1)
- **Ümber kujundada Raporti ja Detailvaate loomise lehte.** Praegu peab arendaja sisestama raporti, detailvate ning alamregioonide kirjeldused ühel lehel. Kui arendaja tahab nendes regioonides luua palju veerge, siis leht läheb liiga pikkaks ja muutub infoga ülekoormatuks. Sellel lehel võiksid olla sakkidga kaardid (*Tabs*). Näiteks, on esimesel kaardil raporti loomine, teisel on detailvaate loomine ja kolmandal on alamregioonide loomine.

Kui rääkida programmi arhitektuuri muutmisest, siis palun mõelda järgmistest asjadest.

- **Vähendada andmebaasi funktsioonide arvu (kirjutatud SQL ja PL/pgSQL keeltes).** Veebirakenduse lehtede koostamine (renderdamine) toimub andmebaasi funktsioonide abil. Põhiprobleem on selles, et PL/pgSQL funktsioone on keeruline kirjutada ja siluda. Võib-olla selle tõttu ei taha arendajad osaleda pgApexi arendamises. Samuti ei toeta PL/pgSQL programmeerimiskeel objektorienteeritud programmeerimise printsiipe ja selle tõttu on vaja pidevalt koodi dubleerida. On vaja uurida, kas kasutaja jaoks lehtede koostamisel oleks võimalik osaliselt loobuda PL/pgSQL funktsioonidest ja teha seda, näiteks, JavaScript raamistiku abil. Autori arvates on see eraldi lõputöö teema.
- **Ümber kirjutada kasutajaliidese (*frontend*) osa uue Angulari versiooni jaoks.** Praegu kasutab pgApex AngularJS raamistikku v1.4.0. AngularJSi veel toetatakse, aga see on *de facto* vananenud. Pakun loobumist AngularJS v1.4.0 raamistikust ja kirjutada ümber kasutajaliidese osa uuemale versioonile: Angular 7.
- **Uuendada Bootstrap versiooni.** Praegu kasutab pgApex lehtede küljendamiseks Bootstrap 3, aga otstarbekas oleks minna üle Bootstrap 4 versioonile. See võimaldab teha ilusamaid lehti ja vähendada CSS koodi kirjutamist. Bootstrapi uuendamine nõuab ka programmi mõnede olemasolevate HTML lehtede ümberkirjutamist, sest uuest versioonist on ära kustutatud varem kasutatud CSS klassid.

Tahan märkida, et ma pakun ainult, et oleks vaja **uurida** nende ideede realiseerimise otstarbekust. Osaline loobumine PL/pgSQL funktsioonidest veebirakenduse lehtede koostamiseks ja kasutajaliidese täielik ülekirjutamine uuele Angular versioonile nõuab väga palju aega, aga see lihtsustab tulevikus programmi arendust ja teeb pgApexi kaasaegseks.

8.1 Intervjuude järel süsteemis tehtud muudatused

Intervjuud toimusid peale arenduse iteratsioonide lõppu. Arvestasin saadud tagasisidega ning tegin intervjuude alusel programmis järgmised parandused.

Regioonide loomise lehel täituvad järjekorra (*Sequence*) väljad automaatselt. Seda tehakse JavaSriptiga.

Süsteem täidab paginatsiooni päringu parameetri (*Pagination query parameter*) välja automaatselt. Siin ei genereeri programm uut välja väärtsust, pöördudes unikaalsuse kontrollimiseks andmebaasi poole. Selle asemel ma kehtestan selle välja vaikimisi väärtuse AngularJS ng-model sisseehitatud direktiivi jaoks. Uue regiooni loomisel on väli automaatselt täidetud, aga kasutaja peab ise jälgima, et paginatsiooni päringu parameeter oleks lehekülje piires unikaalne.

Samuti lisasin täienduse, et klõpsates pgApexi abil loodud veebirakenduses rakenduse nimele (nt fraasile „Spordiklubi infosüsteem“ akna ülemises vasakus nurgas Joonis 14-Joonis 18), siis viiakse kasutaja rakenduse kodulehele. See peaks oluliselt hõlbustama rakenduses navigeerimist.

Lõpuks parandasin ka ühe kasutaja poolt välja toodud vea, mis tekkis uue raporti ja detailvaate loomise ajal. Kirjutasin selleks otstarbeks PHPs validaatori.

9 Kokkuvõte

Selle bakalaurusetöö eesmärgiks oli realiseerida pgApexi programmist puuduvat funktsionaalsust.

Kuna puuduvat funktsionaalsust oli palju, aga aeg oli piiratud, siis oli vaja otsustada, mida peab realiseerima esimeses järjekorras ja mida hiljem (näiteks järgmistes lõputöödes). Selle probleemi lahendamiseks ja uue funktsionaalsuse valimiseks planeerisin ma järgmise väljalaske ja koostasin iteratsioonide plaani. Nende planeerimiseks kasutasin ma Tommy Normani artiklis "Agile Release Planning 101" kirjeldatud meetodit [14]. Lõppkokkuvõttes langetasin koos juhendajaga otsuse realiseerida väljalaskes tabelivormide, detailvaadete ja detailvaate alamraportite haldamise funktsionaalsuse. Samuti otsustasime, et realiseerin süsteemse muutuja APP_USER väärtuse kuvamise HTML koodis ja selle kasutamise tabelivormide funktsioonide väljakutsetes teise argumendina.

Kui uue versiooni funktsionaalsus oli valitud, siis oli vaja mõelda kuidas seda õigesti realiseerida. Selleks uurisin ma kogu programmi arhitektuuri. Vana funktsionaalsuse baasil realiseerisin ma uue funktsionaalsuse. pgApexi arhitektuuri uurimiseks ja koodi silumiseks kasutasin ma programmi Xdebug. Uue pgApexi versiooni arendus toimus lokaalses arvutis, kus oli Ubuntu virtuaalmasin, milles käivitasin PostgreSQL andmebaasisüsteemi ja PHP interpretaatori. Virtuaalmasina lõin kasutades programmi Vagrant.

Kuna pgApex jaguneb kaheks osaks – administraatori töökeskkonnaks, kus arendaja loob veebirakendusi ja veebirakenduste käituskeskkonnaks, siis oli mul vaja teha muudatusi andmebaasis, tagasüsteemis (*backend*) ja kasutajaliideses (*frontend*). Uue funktsionaalsuse realiseerimiseks lisandusid andmebaasis uued tabelid, funktsioonid ja andmetüübid ning samuti oli vaja teha muudatusi olemasolevates andmebaasobjektides. Näitena võib välja tuua, et andmebaasi lisandus 16 tabelit ja 35 funktsiooni. Administraatori töökeskkonda lisandusid lehed, kus arendaja võib luua tabelivormi, detailvaadet ning alamraportit. Need koosnevad AngularJS direktiividest, mida pgApexis

hakati esmakorselt kasutama käeoslevas (teises) versioonis. Tabelivormide, detailvaadete ning alamraportide jaoks kirjutasin ka tagasüsteemi osaks olevad validaatorid, mis kontrollivad uue regiooni salvestamisel POST-päringute sisu (kas need vastavad süsteemi nõuetele).

Seal kus võimalik kirjutasin uue funktsionaalsuse jaoks ühiktestid. Kirjutasin ühiktestid käesolevas versioonis loodud validaatoride jaoks (*TabularFormRegionValidator* ja *ReportAndDetailViewValidator*). Kuna need asuvad tagasüsteemis ja on kirjutatud PHP programmeerimiskeeles, siis testide kirjutamiseks kasutasin ma PHPUnit ja Mockery programme.

Lõputöö tulemuste valideerimiseks tegin näidisrakenduse, mis põhineb minu osalusel loodud õppeaine „Andmebaasi II“ projektil [8]. Näidisrakendus asub aadressil: `http://apex.ttu.ee/pgapex2/public/index.php/app/1/1`

kasutajanimi: *louella.henson@colaire.com*, parool: *VZC%#88e*

Lisaks sellele viisin läbi kaks intervjuud üliõpilastega, kes olid varem kasutanud esimest pgApexi versiooni ja kellel ma palusin proovida kasutada ka uut versiooni. Realiseerisin mõned nende tehtud ettepanekud pgApexi uues versioonis nagu näiteks järjekorranumbri ja paginatsiooni päringu parameetri genereerimine.

pgApexi uue versiooni kood on maailmale avaldatud ja on kättesaadav GitHub-is. https://github.com/nikopa96/pgapex2

Selle hoidla näol on tegemist programmi esimese versiooni hoidla hargmikuga (*fork*). Tulevikus ühendatakse see *pull request* abil pgApex algse hoidlaga ja siis on lähtekood leitav sealt: https://github.com/raitraidma/pgapex

pgApexi arendus ei ole selle tööga kaugeltki lõppenud. Palju on veel teha ning mõned ettepanekud edasiseks arenduseks tõin ma välja töö lõpus olevas arendusvaates.

Kasutatud kirjandus

[1] e-Teatmik: IT ja sidetehnika seletav sõnaraamat. [WWW] http://vallaste.ee/ (18.05.2019)

[2] JSON - Vikipeedia. [WWW] https://et.wikipedia.org/wiki/JSON (20.05.2019)

[3] PHP: Introduction - Manual. [WWW]. https://www.php.net/manual/en/intro.pdo.php (20.05.2019)

[4] REST – Vikipeedia. [WWW] https://et.wikipedia.org/wiki/REST (20.05.2019)

[5] Jellema, L. Oracle APEX: the low-code and low-cost application middle tier - AMIS Oracle and Java Blog. [WWW] https://technology.amis.nl/2018/11/01/oracle-apex-the-low-code-and-low-cost-application-middle-tier/ (18.05.2019)

[6] Savenko, D. Read-Write APEX application fully based on alien data or is it mandatory to use exactly Oracle Database? [WWW] https://dsavenko.me/read-write-apex-application-fully-based-on-alien-data/ (18.05.2019)

[7] Raidma, R. PostgreSQL andmebaasisüsteemi põhine metaandmetega juhitavate veebirakenduste kiirprogrammeerimise keskkond. Magistritöö, Tallinna Tehnikaülikool, 2016. [WWW] https://digi.lib.ttu.ee/i/?6787 (28.02.2019).

[8] Andrejev, V, Tšetšnev, V, Kopa, N. Spordiklubi infosüsteemi treeningute arvestus. Aineprojekt õppeaines „Andmebaasid II“, Tallinna Tehnikaülikool, 2018.

[9] Hevner, A. R, March, S. T., Park, J, Ram, S. (2004). Design Science in Information Systems Research – MIS Quarterly, 28 (1), 75-105. [Online] The ACM Digital Library (20.02.2019)

[10] D. Qiu, B. Li, and Z. Su, “An empirical analysis of the co-evolution of schema and code in database applications,” Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013, 2013.

[11] Foreign data wrappers - PostgreSQL wiki. [WWW] https://wiki.postgresql.org/wiki/Foreign_data_wrappers (18.05.2019)

[12] Burns, L. (2011). Building the Agile Database. How to Build a Successful Application Using Agile Without Sacrificing Data Management. Technics Publication, New Yersey. https://www.ester.ee/record=b4425682*est

[13] Ameller D., Farré C., Franch X., Rufian G. (2016) A Survey on Software Release Planning Models. In: Abrahamsson P., Jedlitschka A., Nguyen Duc A., Felderer M., Amasaki S., Mikkonen T. (eds) Product-Focused Software Process Improvement. PROFES 2016. Lecture Notes in Computer Science, vol 10027. Springer, Cham

[14] Norman, T. Agile Release Planning 101. [WWW]. http://tommynorman.blogspot.com/2012/09/agile-release-planning-101.html (3.05.2019)

[15] Fibonacci jada – Vikipeedia. [WWW] https://et.wikipedia.org/wiki/Fibonacci_jada (12.05.2019)

[16] Agile practices: managerial. [WWW] http://se.ethz.ch/~meyer/books/agile/manprac.pdf (12.05.2019)

[17] What is a Timebox? | Agile Alliance. [WWW] https://www.agilealliance.org/glossary/timebox/ (12.05.2019)

[18] User Stories and User Story Examples by Mike Cohn. [WWW] https://www.mountaingoatsoftware.com/agile/user-stories (20.05.2019)

[19] Separation of Concerns | DevIQ. [WWW] https://deviq.com/separation-of-concerns/ (20.05.2019)

[20] Waiting for 9.6 – Add num_nulls() and num_nonnulls() to count NULL arguments. – select * from depesz;. [WWW] https://www.depesz.com/2016/02/08/waiting-for-9-6-add-num_nulls-and-num_nonnulls-to-count-null-arguments/ (18.05.2019)

[21] pgapex/5_2_create_app_functions.sql at master · raitraidma/pgapex. [WWW] https://github.com/raitraidma/pgapex/blob/master/pgapex/evolutions/5_2_create_app_functions.sql (20.05.2019)

[22] AngularJS: Developer Guide: Directives. [WWW] https://code.angularjs.org/1.4.0/docs/guide/directive (13.05.2019)

Lisa 1 – Intervjuu 1

Tabelivormi regiooni loomine.

1. Kas regiooni loomise protsess oli mugav?

Mugav nagu teiste regioonide loomine, aga kahjuks on vaja täita tekstilahtreid nagu *sequence, Region Template* jne, mis võiksid olla täidetud automaatselt.

2. Kuidas võiks regiooni loomise protsessi lihtsustada?

Vaikimisi tekstilahtrid peavad olema täidetud automaatselt. Samuti *View* rippmenüü asukoht võiks olla pärast *Name* tekstilahtrit. *Region Template* rippmenüü asukoht võiks olla pärast *View*. Vormi elemendid *is visible, show header* lehe lõpus. *Sequence* ja *Pagination query parameter* väärtused võiksid olla genereeritud automaatselt kasutades JavaScript.

3. Mis oli positiivne ja negatiivne loomise protsessi ajal?

Kõik oli positiivne, aga mõnede vormi elementide loomine võiks toimuda otstarbekamalt.

4. Mida oli keeruline luua?

Oli keeruline aru saada, millist *sequence* ja *pagination query parameter* väärtust valida.

Raporti ja detailvaate loomine.

1. Kas regiooni loomise protsess oli mugav?

Oli mugav, see on sama kui eelmises regioonis. Nüüd ei ole vaja kirjutada HTML koodi, et realiseerida detailvaadet. Kahjuks on vaja sisestada järjekorra numbreid (*sequence*) nagu eelmises regioonis.

2. Kuidas võiks regiooni loomise protsessi lihtsustada?

Vahetada *name*, *view*, *unique ID* plokki ja *pagination query parameter* asukoht.

3. Mis oli loomise protsessi ajal positiivne ja negatiivne?

Positiivne on, et ühel lehel ma võin teha kohe kaks regiooni. Negatiivne on, et nende loomiseks on vaja läpida palju samme (nt *sequence* täitmine).

4. Mida oli keeruline luua?

Ei olnud keeruliseks.

Alamraporti alamregiooni loomine.

1. Kas regiooni loomise protsess oli mugav?

Jah, see oli intuitiivne. Hall ristkülik – see on hea!

2. Kuidas võiks regiooni loomise protsessi lihtsustada?

Ei ole vaja lihtsustada.

3. Mis oli loomise protsessi ajal positiivne ja negatiivne?

Positiivne on, et alamraporti loomise vorm asub hallis ristkülikus. Negatiivne on, et järjekorranumbri tekstilahter on liiga pikk.

4. Mida oli keeruline luua?

Ei olnud keeruline. Kõik ühes kohas – see on tore.

Lisa 2 – Intervjuu 2

Tabelivormi regiooni loomine.

1. Kas regiooni loomise protsess oli mugav?

Jah.

2. Kuidas võiks regiooni loomise protsessi lihtsustada?

Võiks teha lühikirjeldus ehk vihje mis on *sequence* või *Region template*.

3. Mis oli loomise protsessi ajal positiivne ja negatiivne?

Kõik oli korras.

4. Mida oli keeruline luua?

Kõik oli lihtne.

Raporti ja detailvaate loomine.

1. Kas regiooni loomise protsess oli mugav?

Oli mugav.

2. Kuidas võiks regiooni loomise protsessi lihtsustada?

Kõik on korras.

3. Mis oli positiivne ja negatiivne loomise protsessi ajal?

Uue raporti ja detailvaate loomise ajal tekkis viga: ma vajutasin nupule et salvestada, aga midagi ei salvestanud ja vea sõnumit ei ilmunud. Tuli välja, et ma juhuslikult kasutasin samu järjekorranumbreid raportil ja detailvaatel.

4. Mida oli keeruline luua?

Raporti ja detailvaate loomine ühel lehel on mugav. Vormi tekstilahtrid on mõlematel regioonidel identsed ja see lihtsustab tööd.

Alamraporti alamregiooni loomine.

1. Kas alamregiooni loomise protsess oli mugav?

Oli mugav.

2. Kuidas võiks lihtsustada regiooni loomise protsessi?

Loomise protsess on lihtne, aga kui raporti, detailvaate ja alamraportide jaoks kirjeldatakse palju veerge, siis leht muutub väga pikaks ja see on halb. Pakun idee: jagade see leht kolmeks osaks ehk kaardiks: esimeses on raporti loomine, teises on detailvaate loomine ja kolmandas on alamregioonid.

3. Mis oli loomise protsessi ajal positiivne ja negatiivne?

Kõik oli positiivne.

4. Mida oli keeruline luua?

Kõik oli lihtne.