TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut
Infosüsteemide õppetool

Andmebaasi disaineri töökoha realiseerimine PostgreSQL andmebaasi disaini kontrollimise süsteemi jaoks

bakalaureusetöö

Üliõpilane: Marko Vanaveski
Üliõpilaskood: 112810IAPB
Juhendaja: Erki Eessaar

Tallinn
2015

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.

--- ---

(*kuupäev*) (*allkiri*)

Annotatsioon

Andmebaasi disaineri töökoha realiseerimine PostgreSQL andmebaaside disaini kontrollimise süsteemi jaoks

Käesoleva töö eesmärgiks on disainida ning realiseerida veebipõhise kasutajaliidesega töökoht PostgreSQL andmebaaside potentsiaalsete disainiprobleemide ning vigade leidmiseks. Selleks loodi veebirakendus, kus kasutajal on võimalik tema poolt valitud andmebaasi peal käivitada erinevaid päringuid ning teste, mis annavad teavet võimalike disaini vigade kohta. Päringute tulemusi analüüsides saab hinnata olemasoleva andmebaasi disaini ning vajadusel saadud tulemuste põhjal viia sisse parandusi ning täiendusi. Süsteemi idee, metaandmete andmebaasi struktuur ja analüüsiks kasutatavad päringud olid autorile ette antud.

Töö tähtsaimaks tulemuseks on kasutusvalmis veebirakendus eelpool mainitud töökoha jaoks ja eesmärkide täitmiseks. Rõhku pani hea kasutatavusega kasutajaliidese disainimisele ja arusaadava ning lihtsalt laiendatava arhitektuuriga rakenduse loomisele. Loodud rakendus veebipõhise kasutajaliidesega asub leheküljel http://apex.ttu.ee/queryexecution.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 75 leheküljel, 6 peatükki, 51 joonist, 1 tabelit.

Abstract

Implementing Database Designer Workplace for a System that Checks the Design of PostgreSQL Databases

The goal of this thesis is to design and build a workplace with web-based user interface for a system that checks the design of PostgreSQL databases. For that purpose, web application was developed where user could execute different queries and tests on database that he/her selected. Returned results give information about possible design flaws in the database. By analyzing the results of the queries, one can assess the design on existing database and if needed make required changes and improvements. The idea behind the system, metadata database structure, and used queries were given to the author.

The most important result of this work is ready to use web application for achieving the previously mentioned goals. Emphasis was put on designing user interface with good usability and creating application with easy to understand and easily extendable architecture. The created application with web-based interface is located at http://apex.ttu.ee/queryexecution.

The thesis is in Estonian and contains 75 pages of text, 6 chapters, 51 figures, 1 table.

Lühendite ja mõistete sõnastik

Kasutajaliides ***User Interface***

„Kasutaja ja arvutiprogrammi vaheline ühenduslüli. Kasutajaliides kujutab endast käskude või menüüde komplekti, mille abil kasutaja saab programmiga suhelda.“ [1]

Sõrestikmudel ***Wireframe***

„Kest, mis näitab loodavaid veebilehti. Iga leht on joonistatud informatsiooniplokkidena, mis näitavad kus ja kuidas paiknevad erinevad komponendid. Selliselt esitatakse objektide visuaalne paigutus (*layout*), kuid mitte lõplik välimus.“ [2]

Prototüüp ***Prototype***

„Prototüüp on süsteemi algne versioon, mida kasutatakse disaini võimaluste katsetamiseks ning ideede demonstreerimiseks.“ [3]

CSS ***Cascading Style Sheets***

„Veebilehtede valmistajatele ja kasutajatele mõeldud laadistik. Laadilehed (*style sheets*) kirjeldavad, kuidas HTML dokumente esitada kuvaril, printeril või kõnesüntesaatorist kostva kõnena. Laadilehed lubavad kasutajal muuta sadade dokumendilehtede väljanägemist üheainsa CSS faili muutmise teel.“ [1]

HTML ***HyperText Markup Language***

„Enimlevinud kodeerimissüsteem (tekstivorming) veebidokumentide loomiseks. HTML koodid ehk märgendid määravad ära selle, kuidas veebileht arvutiekraanil välja näeb.“ [1]

Eelprotsessor ***Preprocessor***

„Tarkvara, mis töötleb toorandmed põhiprogrammile sobivale kujule.“ [1]

LESS ***LESS***

CSS-i eelprotsessor, mis laiendab CSS laadistikku. Lisab juurde uusi

võimalusi. Näiteks saab deklareerida muutujaid ja funktsioone, mis muudavad CSS-i kergemini hallatavamaks. [4] Less laiendiga faile saab kompileerida brauserile sobivaks CSS failiks.

AJAX ***Asynchronous JavaScript and XML***

„Lühend AJAX võeti kasutusele 2005.a. ja sellega tähistatakse selliste interaktiivsete veebirakenduste loomise meetodit, kus toimub jooksev kulissidetagune andmevahetus brauseri ja veebiserveri vahel, nii et kasutaja iga liigutuse peale pole vaja kogu veebilehte uuesti alla laadida.“ [1]

SQL ***Structured Query Language***

„Enimkasutatav päringukeel, mida toetavad kõik klient-server keskkonnale projekteeritud relatsioonandmebaasid. Päringukeeled kujutavad endast reeglite kogumit, mille alusel konstrueeritakse päringuid andmete otsimiseks andmebaasist.“ [1] Lisaks päringute tegemise ehk andmete otsimise võimalusele pakub SQL lauseid ka andmete muutmiseks, tehingute juhtimiseks, õiguste jagamiseks ning andmebaasiobjektide haldamiseks.

Eessüsteem ***Front end***

„Veebilehele ilmuv kasutajaliides, mis võimaldab veebisaidi külastajal kahepoolselt suhelda saidi dünaamiliste osadega nagu andmebaasid, ostukorviprogrammid ja onlain-ostutöötlustarkvara.“ [1]

Jooniste nimekiri

Joonis 1: Päringute ja testide käivitamise funktsionaalse allsüsteemi kasutusjuhtude eskiismudel 19
Joonis 2: Andmebaasi olemi-suhte diagramm 21
Joonis 3: Andmebaasi valimise eskiis 23
Joonis 4: Päringute või testide jooksutamise valimise eskiis 23
Joonis 5: Päringute valimise ning tulemuste esitamise eskiis 24
Joonis 6: Testide valimise eskiis 25
Joonis 7: Testi tulemuste eskiis 26
Joonis 8: Sisselogimise vaade 28
Joonis 9: Sisselogimise ebaõnnestumine 28
Joonis 10: Lehekülje päis sisselogimise lehel 30
Joonis 11: Lehekülje päis, kui andmebaas on valimata 30
Joonis 12: Lehekülje päis, kui andmebaas on valitud 30
Joonis 13: Lehekülje jalus 30
Joonis 14: Andmebaasi valimise vaade 32
Joonis 15: Tegevuse valimise vaade 33
Joonis 16: Andmebaasi valimine päringute või testide käivitamise vaates 34
Joonis 17: Testide valimise vaade 34
Joonis 18: Päringute grupeerimise sakid 35
Joonis 19: Päringute kuvamine ilma tulemusteta 35
Joonis 20: Päringute kuvamine väikese resolutsiooniga seadmetel 36
Joonis 21: Päringu tulemuste grupeerimine sakkide abil 37
Joonis 22: Külgriba päringute tulemuste näitamise puhul 38
Joonis 23: Päringu kuvamine koos tulemustega 39
Joonis 24: Päringu tulemuste detailaken 40
Joonis 25: Päringute valimise vaade 41
Joonis 26: Päringu tüübi kuvamine 42
Joonis 27: Bootstrapi muutujate muutmine 44
Joonis 28: Teenuste defineerimine konfiguratsioonifailis 46
Joonis 29: marsruutide määramine konfiguratsioonifailis 47

Joonis 30: Rakenduse kontrollerid 48
Joonis 31: Andmebaasi ühenduse seadistamise jadadiagramm DatabaseAccessInterface liidest realiseeriva kontrolleri puhul 50
Joonis 32: Eduka autentimise jadadiagramm 51
Joonis 33: PHP äriloogika olemite diagramm nr 1 53
Joonis 34: PHP äriloogika olemite diagramm nr 2 54
Joonis 35: Andmebaasi ja PHP objekti vastavuse määramise näide XML failis 55
Joonis 36: Doctrine'i konfiguratsiooni osa config.yml failis 55
Joonis 37: PHP hoidla objektid 56
Joonis 38: Objektide pärimine kasutades Doctrine ORM-i 57
Joonis 39: Päringu käivitaja ja temaga seotud objektid 59
Joonis 40: Andmebaaside otsimisega tegelevate objektide diagramm 61
Joonis 41: Andmebaasi otsimise jadadiagramm 62
Joonis 42: Mallid 63
Joonis 43: Mallile andmete andmine 64
Joonis 44: Eesti keelse tõlkesõnastiku faili sisu 64
Joonis 45: Tõlgete kasutamine mallides 65
Joonis 46: Päringute käivitamise jadadiagramm 66
Joonis 47: Testide käivitamise vana prototüüp 71
Joonis 48: Päringute käivitamise vana prototüüp 72
Joonis 49: Andmebaaside valimise mobiilivaade 73
Joonis 50: Tegevuse valimise mobiilivaade 74
Joonis 51: Päringute kuvamise mobiilivaade 75

Tabelite nimekiri

Tabel 1: süsteemi mittefunktsionaalsed nõuded 18

Sisukord

1. Sissejuhatus 12
1.1 Taust ja probleem 12
1.2 Ülesande püstitus 12
1.3 Metoodika 13
1.4 Ülevaade tööst 14
2. Andmebaasi disaini probleemide tuvastamine 15
2.1.1 Andmebaasi disaini vigade rohkus 15
2.1.2 Andmebaasi süsteemikataloog ja selle abil disainivigade tuvastamine 15
3. Süsteemi analüüs 17
3.1 Terviksüsteemi tükeldus allsüsteemideks 17
3.1.1 Funktsionaalsed allsüsteemid 17
3.2 Päringute ja testide käivitamise funktsionaalne allsüsteem 17
3.2.1 Süsteemi eesmärgid 17
3.2.2 Allsüsteemi kasutavad pädevusalad 18
3.2.3 Mittefunktsionaalsed nõuded 18
3.2.4 Allsüsteemi kasutusjuhtude eskiismudel 19
4. Disain 21
4.1 Andmebaasi disain 21
4.1.1 Olemi-suhte diagramm 21
4.2 Kasutajaliidese prototüüp 22
4.2.1 Loodud prototüüp 22
4.3 Kasutajaliidese disaini põhimõtted 26
4.3.1 Sisselogimise vaade 27
4.3.2 Päis ja jalus 29
4.3.3 Andmebaasi valimise vaade 31
4.3.4 Tegevuse valimise vaade 32
4.3.5 Testide valimise vaade 33
4.3.6 Päringu ja tema tulemuste kuvamine 35
4.3.7 Päringute valimise vaade 40
5. Rakenduse arhitektuur 43

5.1 Eessüsteemi ehitamine ... 43
5.1.1 Twitter Bootstrap ... 43
5.1.2 LESS eelprotsessor ... 43
5.2 Rakenduse serveripoolse osa arhitektuur ... 45
5.2.1 Symfony ... 45
5.2.2 Kontrollerid ... 46
5.2.3 Autentimine ... 50
5.2.4 Rakenduse andmebaasiga suhtlemine ... 51
5.2.5 Päringute käivitaja ... 57
5.2.6 Andmebaasi otsimine ... 60
5.2.7 Mallid ... 62
5.2.8 Kasutajaliidese keele valik ... 64
5.3 Näide komponentide omavahelisest koostööst ... 65
6. Kokkuvõte ... 67
Summary ... 68
Kasutatud kirjandus ... 69
Lisa 1 ... 71
Lisa 2 ... 72
Lisa 3 ... 73
Lisa 4 ... 74
Lisa 5 ... 75

1. Sissejuhatus

1.1 Taust ja probleem

Eelnevate tööde tulemusena on valminud mitmeid SQL päringuid, mis aitavad tuvastada võimalikke andmebaasi disaini probleeme. Päringud olid loodud pidades silmas eelkõige PostgreSQL andmebaasisüsteemi. Nende päringute käivitamiseks ja tulemuste kuvamiseks puudus kasutajasõbralik ning terviklik rakendus. Oli loodud küll prototüüp [5], kuid seal oli osa funktsionaalsusest puudu ning selle aluseks olev andmebaasidisain oli muutunud. Seega tekkis antud töö juhendajal vajadus uue kasutajaliidesega rakenduse arendamisele.

Loodav rakendus oleks kasulik eelkõige andmebaasi disainimist õppivatele isikutele. Näiteks TTÜ õppeainete „Andmebaasid I“ ning „Andmebaasid II“ raames peavad üliõpilased looma andmebaasi. Üheks võimaluseks on neil luua andmebaas kasutades PostgreSQL andmebaasisüsteemi. Loodud rakendusega saavad nad kontrollida enda andmebaasi disainini võimalike vigade suhtes. Lisaks saab õppejõud, kui rakenduse andmebaasi haldaja, luua teste, mis kontrollivad üliõpilaste andmebaaside vastavust teatud tingimustele. See lihtsustab nii õppejõu kui ka üliõpilaste tööd.

1.2 Ülesande püstitus

Antud lõputöö eesmärgiks on luua hea kasutatavusega veebipõhise kasutajaliidesega rakendus andmebaaside disaini kontrollimise süsteemi jaoks andmebaasi disaineri pädevusalale. Rakenduse serveripoolne osa peab olema loodud PHP programmeerimiskeeles ning kasutama PostgreSQL andmebaasisüsteemi. Rakenduse konfiguratsioonis määratakse ära andmebaasi server. Kasutajad peavad saama rakendusse sisse logida sama kasutajanime/parooliga, mis neil on andmebaasi serveril.

Läbi loodava liidese peavad rakenduse kasutajad saama eelnevalt kindlaksmääratud serveri peal olevate PostgreSQL andmebaaside hulgast valida endale sobiva andmebaasi. Valitud andmebaasi peal peab saama käivitada erinevatesse kategooriatesse kuuluvaid päringuid ning teste. Kasutaja peab saama vaadata käivitatud päringute tulemusi. Käivitatud päringute kohta tuleb rakenduse andmebaasi salvestada ka mõningat statistikat. Sinna hulka kuulub näiteks

päringu käivitaja kasutajanimi, kellaaeg, käivitatud päring ning päringu poolt leitud ridade arv. Lisaks peab saama määrata kasutajaliidese keelt. Esialgu on valikuteks eesti ning inglise keel.

Antud tööks olid ette antud mõned tulemid. Olemas oli rakenduse juhtandmeid sisaldav andmebaas, kus on disaini kontrolli päringud ja testid ning kuhu tuleb kirjutada andmeid päringute käivitamise kohta. Lisaks oli varem loodud ka prototüübid päringute [5] ning testide [6] käivitamiseks. Lisades 1-2 on esitatud antud prototüüpide ekraanipildid. Tegemist oli kahe erineva prototüübiga, mis polnud omavahel seotud ning asusid erinevatel lehekülgedel. Uue liidese loomisel ühendati need kaks erinevat funktsionaalsust ühte terviklikku rakendusse. Antud prototüübid ei olnud eriti kasutajasõbralikud ning sealt puudus osa vajalikku funktsionaalsusest ning info analüüsimine osutus selle tõttu keerulisemaks kui vaja. Seetõttu tekkis vajadus uue rakenduse järgi.

1.3 Metoodika

Töö eesmärkide saavutamiseks luuakse kõigepealt kasutajaliidese prototüüp kasutades tarkvara Balsamiq Mockups. Kasutajaliidese disainimisel kasutatakse mitmeid kasutajaliideste mustreid. Pärast prototüübi loomist luuakse täielikult toimiv veebirakendus kasutades serveri poolel programmeerimiskeelt PHP, Symfony2 raamistikku ja Doctrine ORM tarkvara. Serveripoolne osa kasutab andmete salvestamiseks ning lugemiseks PostgreSQL andmebaasisüsteemi.

Kliendipoolses osas kasutatakse HTML-i, JavaScripti skriptikeelt, CSS-i ja selle eelprotsessorit LESS, Twitter Bootstrap raamistikku, jQuery raamistikku ning mitmeid jQuery lisamooduleid.

Kasutaja suhtleb rakendusega läbi veebipõhise liidese ehk rakenduse kliendiks on veebibrauser. Server tagastab kliendile HTML faile koos sinna juurde kuuluvate piltide, CSS-i ja JavaScripti failidega. Osadel juhtudel kasutab klient serveriga suhtlemiseks ka AJAX-i põhist andmevahetust.

Lõputöö ise põhineb *design science* metoodikal, kus töö tulemusena luuakse teaduslikest ja insenerlikest põhimõtetest lähtuv töötav ja testitud artefakt (andmebaasi disainivigade otsimise süsteemi veebirakendus).

1.4 Ülevaade tööst

Antud töö koosneb kuuest osast. Teises peatükis tutvustatakse lühidalt antud töö aluseks olevat andmebaasi disaini probleeme ning nende vigade tuvastamist päringutega. Kolmandas peatükis analüüsitakse loodavat süsteemi. Neljandas peatükis uuritakse lähemalt kasutajaliidese disainimist, loodud liidesel kasutatud disainimustreid ning loodud kasutajaliidese prototüüpi. Viiendas peatükis toimub rakenduse arhitektuuri analüüs. Kuuendas peatükis esitatakse käesoleva töö kokkuvõte.

2. Andmebaasi disaini probleemide tuvastamine

Selles peatükis antakse lühiülevaade andmebaasi disainivigade tuvastamisest, mis on ka käesolevas töös esitletava veebirakenduse eesmärk. Kuigi käesolev töö keskendub SQL-andmebaasidele, on süsteemikataloogi põhjal võimalik leida disainivigu ka mitte-SQL andmebaaside korral kui andmebaasi tasemel kirjeldatakse andmebaasi skeem ilmutatud kujul ja see kirjeldus registreeritakse süsteemikataloogis.

2.1.1 Andmebaasi disaini vigade rohkus

Andmebaasi projekteerimise ja loomise protsess sisaldab ka andmebaasi disainimist ehk tehnilise kavandi loomist. Andmebaasi disaini võib jagada kaheks: loogiliseks ja füüsiliseks disainiks [7]. Loogilise disaini puhul kavandatakse andmebaas selle aluseks olevat andmemudelit silmas pidades, kuid mitte konkreetset andmebaasisüsteemi arvestades [7]. Füüsilise disaini puhul võetakse juba arvesse ka konkreetse andmebaasisüsteemi võimalused ja piirangud [7]. SQL-andmebaasi füüsilise disaini tegevuste hulka kuuluvad näiteks kitsenduste ja indeksite loomine ning tabelite valikuline denormaliseerimine.

Kui ei pöörata piisavalt tähelepanu andmebaasi disainile, võivad andmebaasi kasutamise käigus tekkida mitmed probleemid. Näiteks kui jätta deklareerimata andmebaasi tasemel kitsendused, siis võib sinna sattuda andmeid, mis on tegelikult teatud ärireeglitega vastuolus. Antud näite puhul oleks tegemist disainiprobleemiga. Antimustrid on tähtis disainiprobleemide kirjelduste allikas. Antimuster kirjeldab halba lahendust probleemile koos alternatiivse parema lahendusega. Seega oleks andmebaasi disaineri (ja tegelikult kõigi, kes andmeid tähtsaks peavad) seisukohalt kasulik teada, kas loodud andmebaasi disain sisaldab antimustreid või muid disainiprobleeme.

2.1.2 Andmebaasi süsteemikataloog ja selle abil disainivigade tuvastamine

SQL-andmebaasi süsteemikataloog sisaldab metaandmeid ehk andmeid hoitavate andmete kohta. Süsteemikataloog on osa igast SQL-andmebaasist ning selle baastabelites olevad andmed kirjeldavad andmebaasis olevaid skeeme [8]. SQL standard defineerib skeemi nimega INFORMATION_SCHEMA, mis peab sisaldama vaateid, mis põhinevad andmebaasi süsteemikataloogi baastabelitel [8].

Seega on võimalik koostada süsteemikataloogi põhjal päringuid, mis tagastavad infot andmebaasis hoitavate andmete ja andmebaasi struktuuri kohta. Näiteks saab INFORMATION_SCHEMA-sse kuuluvate vaadete põhjal luua päringuid, mis tagastavad teatud andmetüüpi olevate veergude nimed või tabelite nimed, kus puuduvad välisvõtme ja primaarvõtme kitsendused. Seda informatsiooni saab kasutada andmebaasist võimalike antimustrite ja disainivigade leidmiseks. Näiteks on PostgreSQL andmebaaside jaoks on Janina Voronova loonud info saamise eesmärgil mitmed päringud, millega saab täpsemalt tutvuda tema bakalaureusetöös [9]. Kuna INFORMATION_SCHEMA on standardiseeritud struktuuriga, siis saab selle põhjal loodud päringuid kasutada kõigis andmebaasisüsteemides, mis sellist skeemi toetavad.

Lisaks on loodud päringud kaheteistkümne erineva antimustri tuvastamiseks PostgreSQL andmebaasidest [8] [10]. Iga selline antimuster kirjeldab teatud andmebaasi disaini viga ning kirjeldatud päringute tulemused võivad viidata võimalikule disainiveale [10]. Tuleb arvestada, et kui päring tagastab ridu, siis ei tähenda see alati antimustrite olemasolu. TTÜ üliõpilaste andmebaaside peal läbiviidu eksperimendi tulemuste järgi tuvastasid antud 12 päringut kokku 81 juhtu, kus päring leidis antimustri olemasolu [11]. Ülevaatamisel leiti, et 23 juhul oli tegemist ehtsate antimustrite esinemisega. Kõige rohkem valesid tulemusi andsid päringud, mis põhinesid andmebaasiobjektide nimede ja baastabelites olevate andmete analüüsil.

3. Süsteemi analüüs

Selles peatükis esitatakse disainivigade otsimise süsteemi analüüsi mõned kõige olulisemad tulemused.

3.1 Terviksüsteemi tükeldus allsüsteemideks

Tegemist on andmebaasi disaini kontrollimise infosüsteemiga.

3.1.1 Funktsionaalsed allsüsteemid

- Päringute ja testide käivitamise funktsionaalne allsüsteem
- Päringute arvestuse funktsionaalne allsüsteem
- Testide arvestuse funktsionaalne allsüsteem
- Klassifikaatorite arvestuse funktsionaalne allsüsteem

Antud töös vaadeldakse täpsemalt **päringute ja testide käivitamise funktsionaalset allsüsteemi**. Teiste allsüsteemide haldamiseks on käesoleva töö juhendaja poolt loodud Microsoft Accessi töölauarakendus, mis kasutab ODBC draiveri vahendusel serveril olevat PostgreSQL andmebaasi.

3.2 Päringute ja testide käivitamise funktsionaalne allsüsteem

3.2.1 Süsteemi eesmärgid

Antud peatükis tähistatakse sõnaga **kasutaja** andmebaasi disaineri pädevusala. Käesolev süsteem sai loodud täitma järgnevaid eesmärke.

- Võimaldada kasutajatel valida rakendusega töötamiseks sobilik andmebaas eelnevalt kindlaksmääratud serveril eksisteerivate andmebaaside hulgast.
- Võimaldada kasutajatel käivitada tema poolt valitud andmebaasi peal erinevaid võimalikke andmebaasi disaini probleeme leidvaid päringuid.

- Võimaldada kasutajatel käivitada tema poolt valitud andmebaasi peal võimalikke andmebaasi disaini probleeme leidvaid eeldefineeritud teste, mis koosnevad ühest või mitmest kindlaksmääratud päringust.

- Võimaldada näidata kasutajatele päringute ja testide poolt leitud tulemusi.

- Süsteem peab salvestama statistikat käivitatud päringute poolt leitud tulemuste kohta. Sinna alla kuulub päringu poolt tagastatud ridade hulk, päringu käivitanud andmebaasi kasutaja nimi ning päringu käivitamise kellaaeg.

3.2.2 Allsüsteemi kasutavad pädevusalad

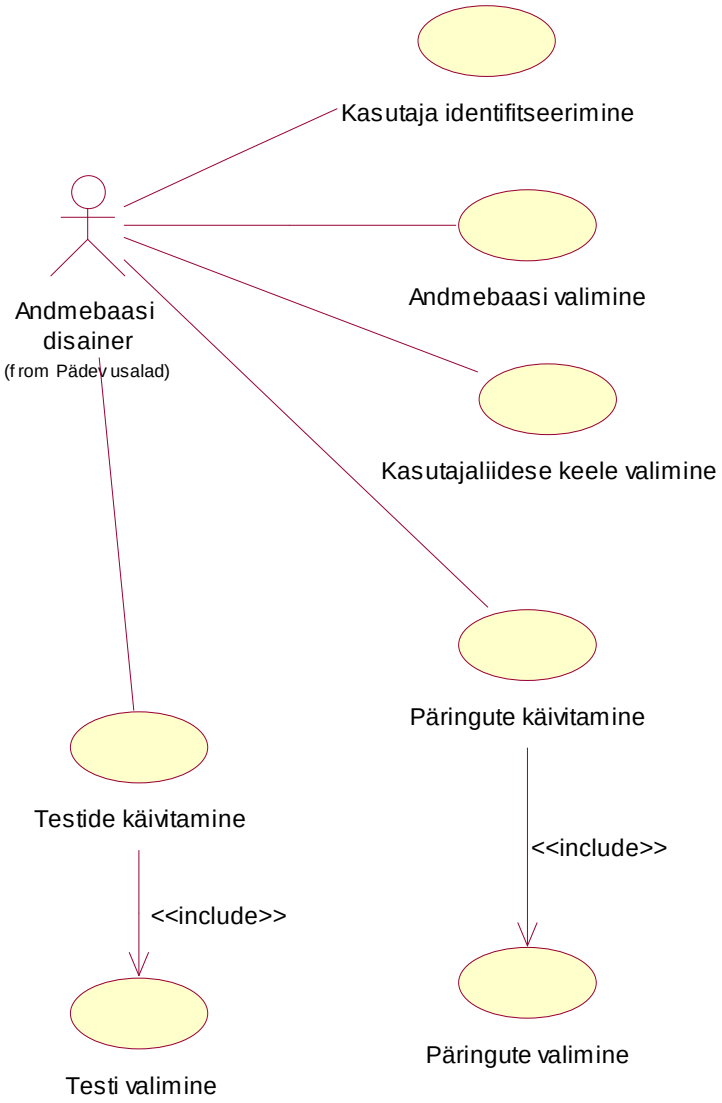
- Andmebaasi disainer

3.2.3 Mittefunktsionaalsed nõuded

Tabel 1: süsteemi mittefunktsionaalsed nõuded

Tüüp	*Nõude kirjeldus*
Andmebaasi-süsteem	Süsteem peab andmete hoidmiseks kasutama PostgreSQL andmebaasisüsteemi.
Arendusvahendid	Süsteemi kasutava rakenduse serveripoolne osa koodist peab olema kirjutatud programmeerimiskeeles PHP.
Kasutajaliides	Süsteemil peab olema veebipõhine kasutajaliides. Kasutajaliides peab olema selge ja arusaadav. Kasutajaliideses peab arvestama ka erinevate ekraani resolutsioonidega. Teisisõnu peab rakendus olema kasutatav ka mitte ainult suurte lauaarvuti monitoridega, vaid ka mobiilsete seadmetega. Kasutaja peab olulise info kätte saama ka väiksemate ekraanidega.
Keel	Süsteemi kasutatava rakenduse kasutajaliides peab olema nii eesti kui ka inglise keeles. Uute keelte lisamine peab olema tehtut võimalikult lihtsaks.
Veebibrauserite tugi	Süsteem peab töötama enimkasutatavatel veebibrauseritel. Sinna hulka kuulub Google Chrome, Internet Explorer ning Mozilla Firefox [12].
Laiendatavus	Süsteemi kasutava rakenduse arhitektuur peab võimaldama tulevikus võimalikult lihtsalt lisada juurde uut funktsionaalsust.

3.2.4 Allsüsteemi kasutusjuhtude eskiismudel



Joonis 1: Päringute ja testide käivitamise funktsionaalse allsüsteemi kasutusjuhtude eskiismudel

Järgnevalt esitatakse kasutusjuhtude tekstikirjeldused kõrgtaseme formaadis.

Kasutusjuht: Kasutaja identifitseerimine
Tegutsejad: Andmebaasi disainer
Kirjeldus: Subjekt identifitseerib ennast süsteemile. Ta sisestab antud serveris asuva PostgreSQL andmebaasi kasutaja kasutajanime ja parooli. Süsteem kontrollib, kas antud kasutajal on andmebaasi kasutaja olemas. Lisaks kontrollitakse, kas tal on olemas vajalikud õigused rakenduse enda andmebaasile, mis on vajalikud rakenduse korrektseks töötamiseks.

Kui subjekt on identifitseeritud ning vajalikud õigused on olemas, siis lubatakse tal süsteemi siseneda.

Kasutusjuht: Andmebaasi valimine
Tegutsejad: Andmebaasi disainer
Kirjeldus: Subjekt valib kõikide rakenduse serveril olevate andmebaaside hulgast teda huvitava andmebaasi. Vajadusel saab ta valida olevate andmebaaside nimekirja kitsendamiseks sisestada otsingusõna.

Kasutusjuht: Kasutajaliidese keele valimine
Tegutsejad: Andmebaasi disainer
Kirjeldus: Subjekt valib hetkel saadaolevate keele valikute seast talle sobivaima kasutajaliidese keele. Süsteem kuvab sama lehekülje subjekti poolt valitud keeles.

Kasutusjuht: Päringute valimine
Tegutsejad: Andmebaasi disainer
Kirjeldus: Subjekt valib teda huvitava päringute kategooria. Süsteem kuvab talle kõik antud kategooria päringud koos päringute infoga. Subjekt märgib teda huvitavad päringud antud kategooria alt valituks.

Kasutusjuht: Päringute käivitamine
Tegutsejad: Andmebaasi disainer
Kirjeldus: Subjekt annab süsteemile teada, et soovib valitud päringud käivitada. Süsteem käivitab valitud päringud subjekti poolt valitud andmebaasil ning kuvab talle päringute tulemused.

Kasutusjuht: Testide valimine
Tegutsejad: Andmebaasi disainer
Kirjeldus: Subjekt valib saadaval olevate testide hulgast teda huvitava. Süsteem kuvab antud testis sisalduvad päringud ning nende info subjektile.

Kasutusjuht: Testide käivitamine
Tegutsejad: Andmebaasi disainer
Kirjeldus: Subjekt annab süsteemile teada, et soovib valitud testi käivitada. Süsteem käivitab kõik antud testis sisalduvad päringud subjekti poolt valitud andmebaasil ning kuvab tulemused subjektile.

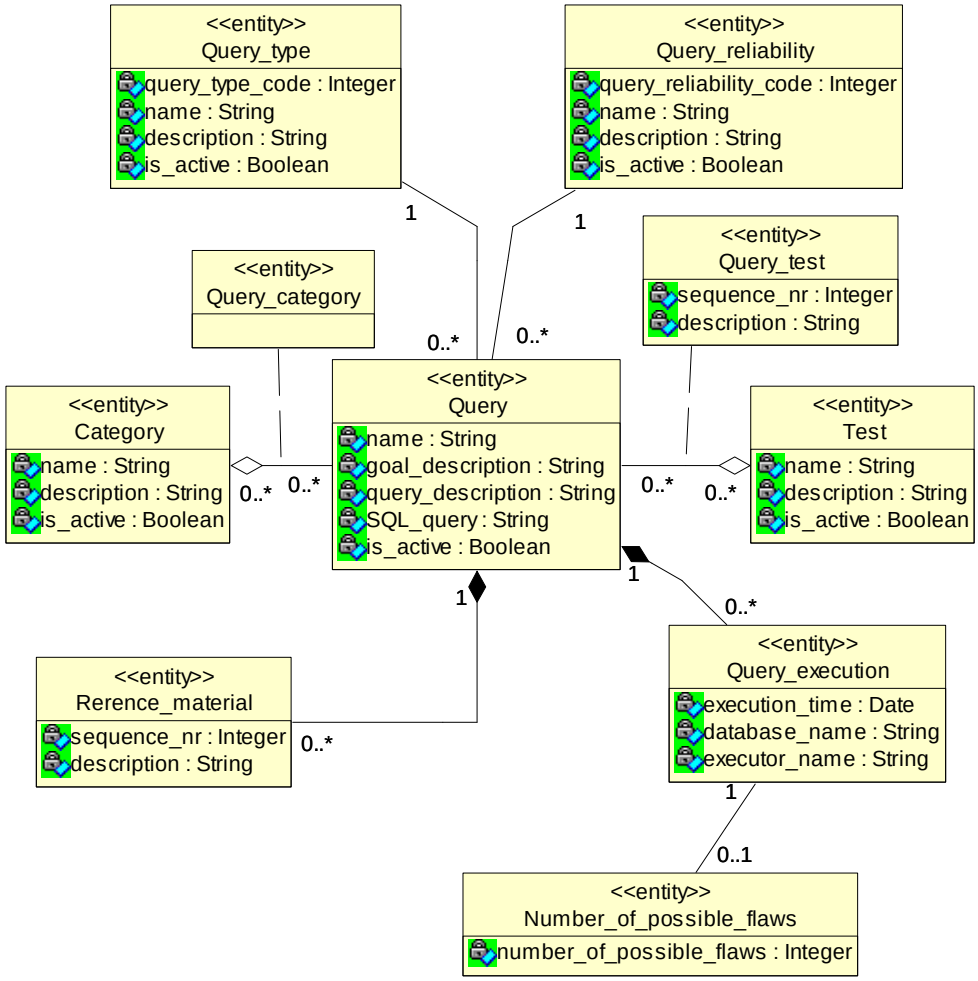
4. Disain

Selles peatükis tutvustatakse rakenduse tööks vajaliku andmebaasi struktuuri ning autori poolt loodud kasutajaliidese väljanägemist.

4.1 Andmebaasi disain

Antud töö jaoks oli antud kasutada varem valmis loodud andmebaas koos kõikide rakenduse töö toimimiseks vajalike andmetega. Töö parema loetavuse huvides on järgnevalt ära toodud etteantud andmebaasi olemi-suhte diagramm.

4.1.1 Olemi-suhte diagramm



Joonis 2: Andmebaasi olemi-suhte diagramm

Keskne olemitüüp on Päring (*Query*), mida saab koondada erinevatesse Testidesse (*Test*), ning otsimise ja rühmitamise huvides ka Kategooriatesse (*Category*). Testis on päringud kindlas järjekorras. Päringute käivitamised registreeritakse (*Query_execution*). Igal päringul on usaldusväärsuse hinnang (*Query_reliability*) ning mingit tüüpi (*Query_type*). Tüüpideks on „päring, mis leiab mõõdiku väärtuse“, „päring, mis leiab võimalikud disaini vead“ ja „päring, mis annab üldist infot“. Päringutega võib olla seotud abimaterjale (*Reference_material*).

4.2 Kasutajaliidese prototüüp

Rakenduse loomist alustati esmalt uue kasutajaliidese prototüübi loomisest. Antud etapis pandi rõhku sellele, kuidas kasutajale infot esitada (info esitamise viisidele ja paigutamisele) ning liidese elementide üldisele asukohale lehel.

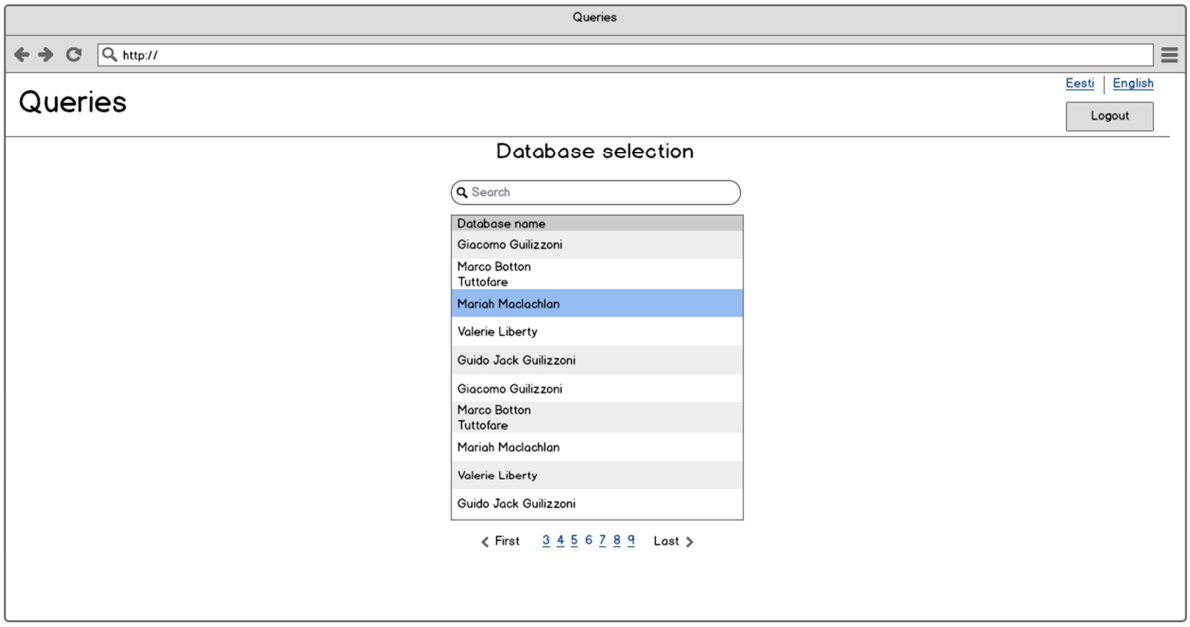
Tulevane kasutajaliides esitati staatiliste piltidena, *wireframe* ehk sõrestikmudeli [2] kujul. „*Wireframe* on ekraanipiltide kaudu rakendustarkvara baasfunktsionaalsuse kirjeldus ehk teisisõnu on tegemist tarkvara prototüübiga (esmase lahendusega) [13]“. Seega ei ole tegemist funktsionaalse prototüübiga, kus kasutaja saab interaktiivselt rakendust proovida. Loodud prototüübi staatilised pildid annavad selge ülevaate, milline loodav rakendus välja hakkab nägema, jättes arvestamata kasutajaliidese elementide konkreetseid värve, kujusid ning suurusi. Teisisõnu määratakse ära paigutus, mitte lõplik välimus [2].

Prototüüpide loomiseks kasutati tarkvara Balsamiq Mockups veebipõhist demoversiooni [14]. Valik langes Balsamiq’ule, kuna seal on esitatud paljud enamlevinud kasutajaliidese elemendid ning see omab väga lihtsat ning arusaadavat veebipõhist liidest, millega saab esialgsed eskiisid kiiresti tehtud. Töö autor kasutas teist versiooni, kuid nüüdseks on valmis ka Balsamiq Mockups kolmas versioon [15]. Seega võivad järgnevad joonistel olevad kasutajaliidese elemendid ning nende väljanägemine mõnevõrra erineda viimase versiooni poolt pakutavatest elementidest.

4.2.1 Loodud prototüüp

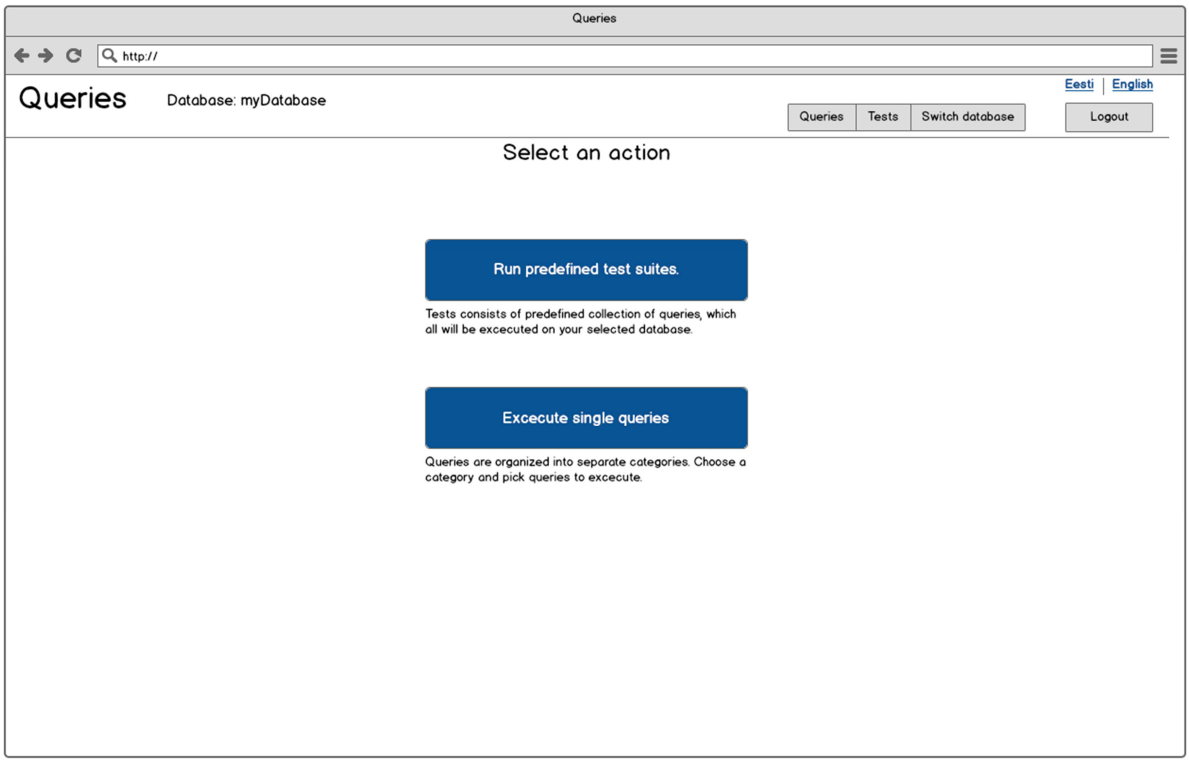
Järgnevalt esitatakse eelpool mainitud tarkvara abil loodud prototüüpide pildid. Kujutatud on olulisemaid rakenduse kasutamise aspekte, välja on jäetud sisselogimise funktsionaalsus. Prototüüpide disain arutati tagasiside saamiseks juhendajaga läbi, pärast mida alustati prototüübi realiseerimist.

Pärast sisselogimist avaneb kasutajale andmebaasi valimise lehekülg. Siin saab kasutaja valida ja otsida kõigi antud rakenduse serveris olevate andmebaaside seast.



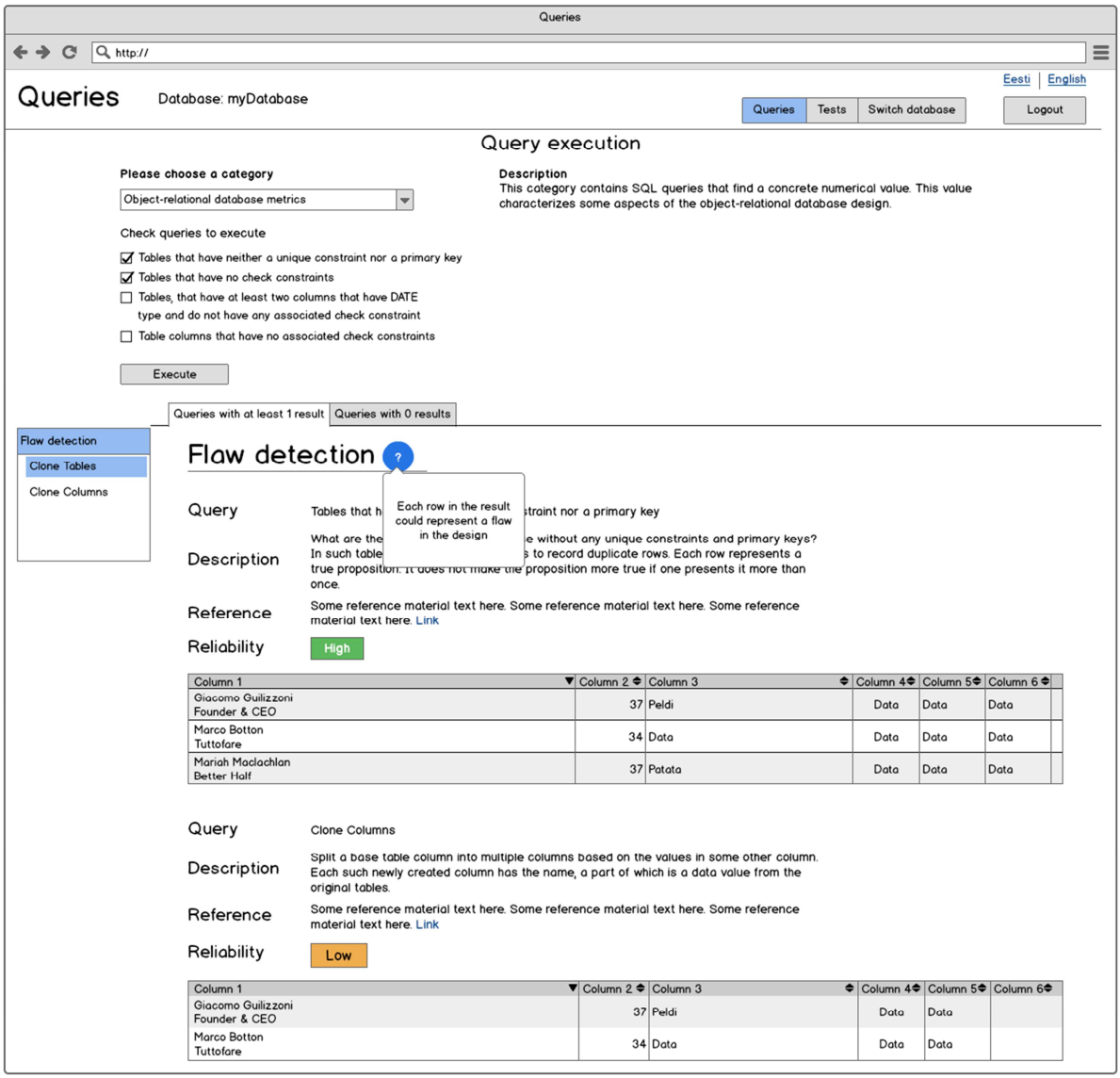
Joonis 3: Andmebaasi valimise eskiis

Pärast andmebaasi valimist kuvatakse kasutajale võimalike tegevuste valik. Tal on võimalik valida, kas soovitakse käivitada konkreetseid üksikuid päringuid või terveid teste.



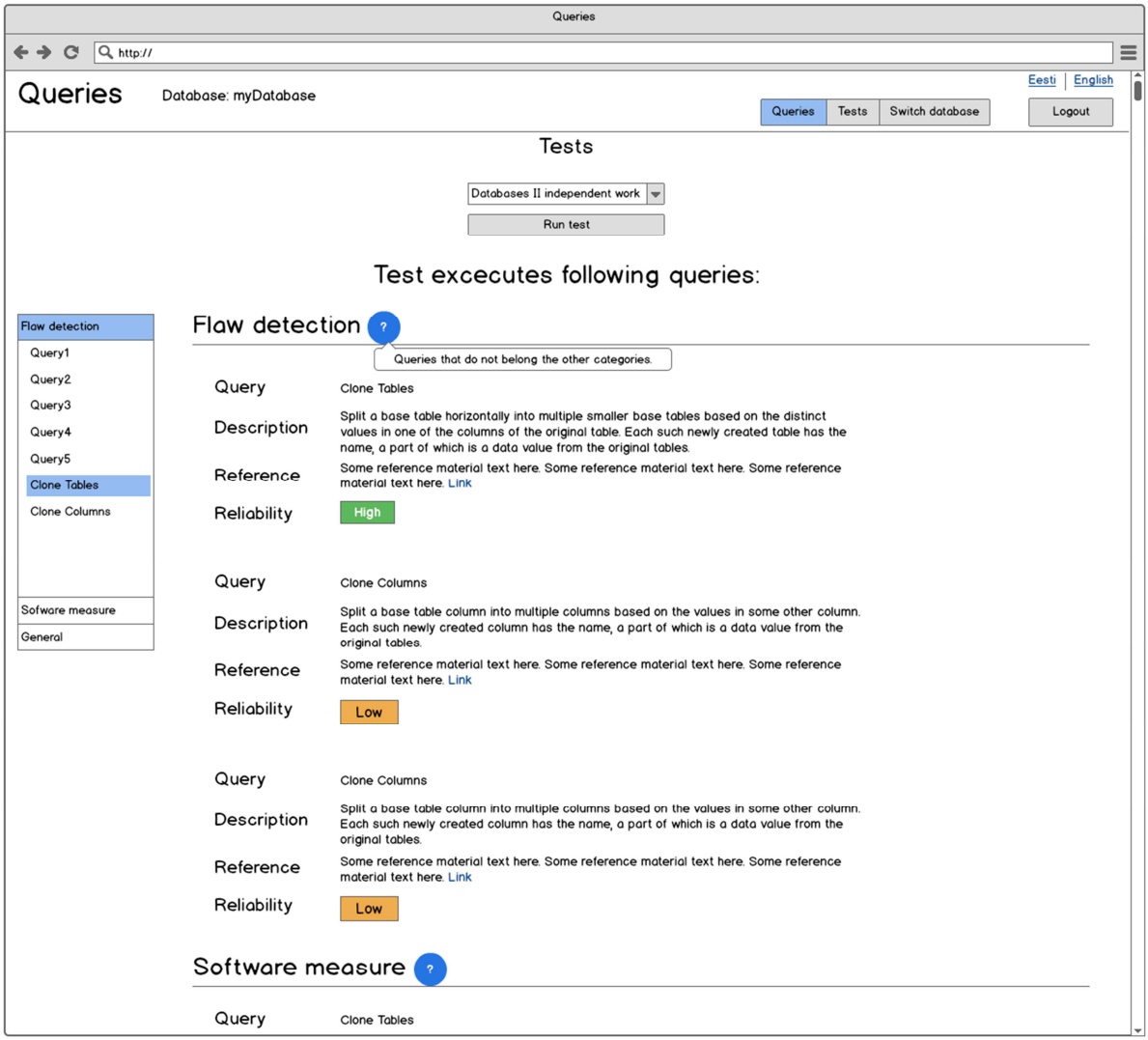
Joonis 4: Päringute või testide jooksutamise valimise eskiis

Järgnevalt on kujutatud üksikute päringute valimist ning nendega seotud info ja päringute tulemuste kasutajale kuvamist.



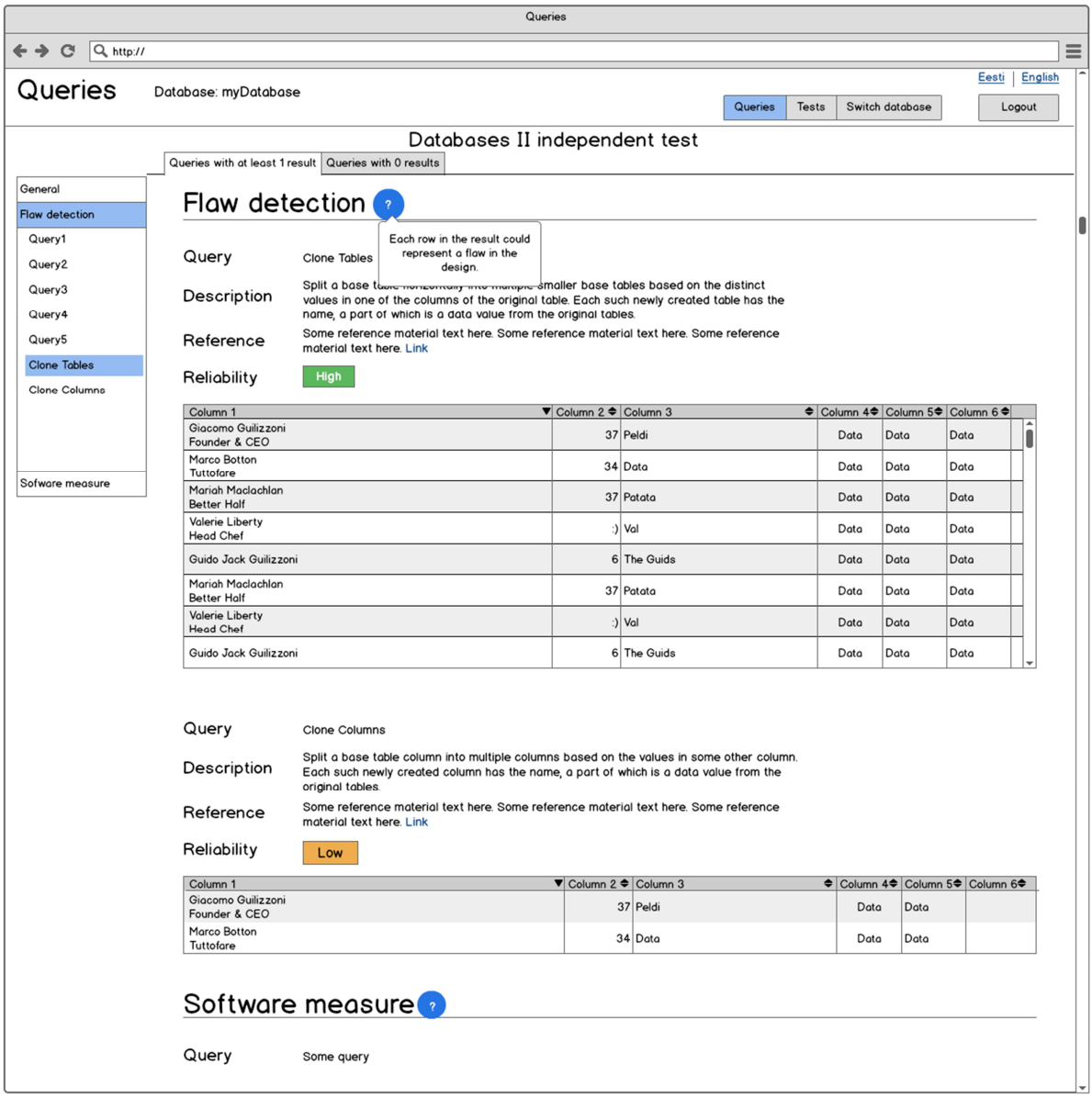
Joonis 5: Päringute valimise ning tulemuste esitamise eskiis

Siin on kujutatud testide valimist ning konkreetse valitud testi kohta käiva info ning selles sisalduvate päringute info kujutamine.



Joonis 6: Testide valimise eskiis

Pärast testi kävitamise lõppu kuvatakse kasutajale tulemuste aken. Tulemused on esitatud samasuguselt nagu üksikute päringute käivitamiselgi.



Joonis 7: Testi tulemuste eskiis

4.3 Kasutajaliidese disaini põhimõtted

Rakenduse esmamulje luuakse kasutajaliidese põhjal. Kui rakenduse kasutamine on keerukas ning kasutajale mitteintuitiivne ning raskesti arusaadav, siis suure tõenäosusega ta rakendust tulevikus kasutama ei hakka. Eriti on see oluline just veebipõhiste rakenduste juures, kus rakendust saab kasutada iga seadmega, millel on internetiühendus. See tähendab omakorda, et peab arvestama võimalikult paljude erinevate ekraaniresolutsioonidega, et liides oleks nendel

kõikidel võimalikult hästi kasutatav. Selleks on kasutusel paindlik veebidisain ehk *responsive design.*

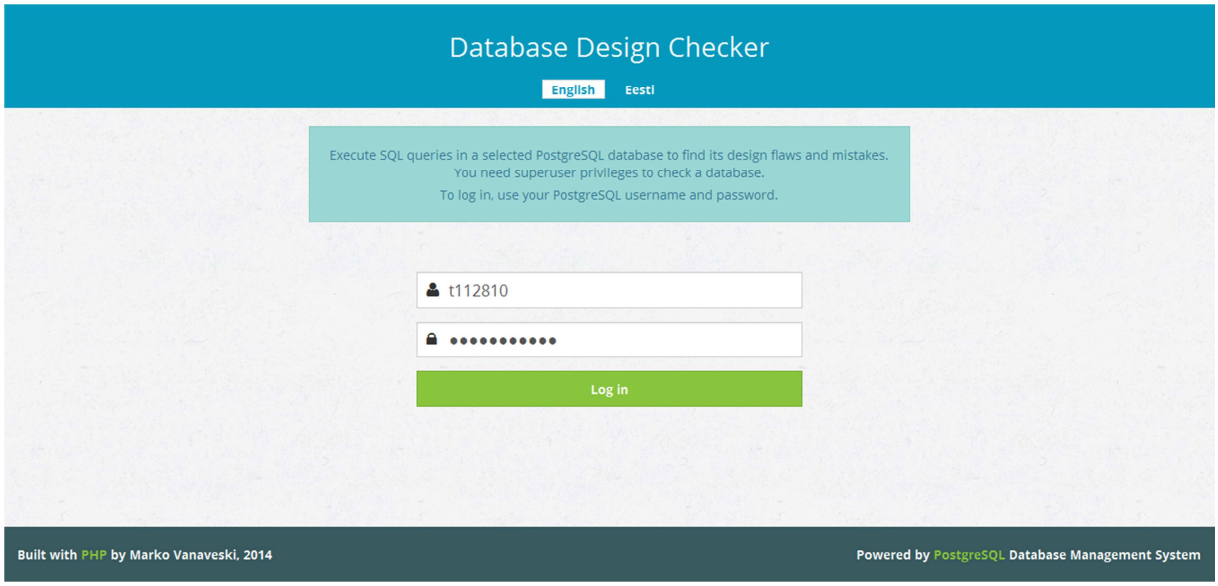
Paindliku veebidisaini kasutav lehekülg kohandub vastavalt seadme ekraani vaatealale. Kuna veebilehti kasutatakse läbi veebibrauserite, siis kohandub lehekülg vastavalt brauseriakna suurusele. Kuna väiksema vaateala laiusega ekraanile mahub vähem infot, siis tuleb valida, kas peita osa informatsioonist kasutajate eest ära või muuta info asukohta ehk paigutust väiksematel ekraanidel. Ka antud rakenduse veebiliides loodi paindlikku disaini silmas pidades. Lisades 3-5 on esitatud näited, kuidas käesoleva rakenduse kasutajaliides kasutab paindlikku disaini ning väiksema resolutsiooniga telefonidel välja näeb.

Järgnevalt põhjendatakse kasutajaliidese disainimisel tehtuid otsuseid, kasutatud kasutajaliideste mustreid ning põhimõtteid. Loodud veebiliidest testiti brauseritega Internet Explorer 11, Mozilla Firefox 38 ja Google Chrome 43.

4.3.1 Sisselogimise vaade

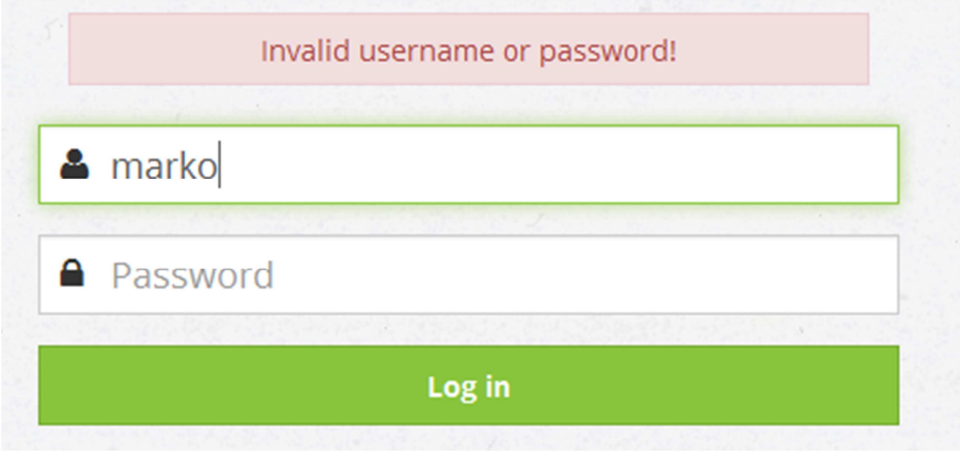
Loodud rakenduse avaleheks on sisselogimise lehekülg. Avaleheküljel esitatakse süsteemi tutvustus ning nõuded rakendusse sisenemiseks. Lehe kõige ülemises osas on olemas ka valik kasutajaliidese keele vahetamiseks.

Sisselogimise vorm on *gatekeeper form*, ehk vorm, mis seisab kasutaja ja tema eesmärgi saavutamise vahel [16]. Eesmärgiks on sellel juhul päringute ja testide käivitamine andmebaasi peal. Sellist tüüpi vormi puhul tuleks kasutada *center stage* mustrit ehk vorm on paigutatud lehekülje keskele, et kasutaja tähelepanu koonduks just sellel [16].



Joonis 8: Sisselogimise vaade

Kui kasutaja jätab kasvõi ühe väljadest täitmata, siis kuvatakse talle vastav info ning vormi ei saadeta serverile ära. See välistab serveri koormamist asjatute päringutega, sest kasutaja autentimine ebaõnnestuks sellisel juhul niikuini.



Joonis 9: Sisselogimise ebaõnnestumine

Kui sisestatud kasutajanime ja parooli kombinatsiooniga ei õnnestu siseneda, siis kuvatakse vastav veateade. Veateates ei täpsustada kas parool, kasutajanimi või need mõlemad olid valed. See välistab juhtumi, kus juhuslikult valitud kasutajanime ja parooli kombinatsiooni sisestamisega saab teada, kas sisestatud kasutajanimega kasutaja eksisteerib või mitte. Ebaõnnestunud sisselogimise korral jäetakse vormil alles eelnevalt sisestatud kasutajanimi. See väldib vajaduse, et peaks vale parooli sisestamise korral enda kasutajanime uuesti sisestama. Lisaks tühjendatakse automaatselt paroolivälli kuna sisestatud parooli kasutajale

teksti kujul ei esitata ja seega ei saaks kasutaja ka teada, millise parooli ta ebaõnnestunud sisselogimise korral sisestas.

Lisaks kontrollitakse sisselogimise puhul veel seda, kas kasutajal on vajalikud õigused rakenduse andmebaasile. Kui need puuduvad, siis kuvatakse nagu ebaõnnestunud sisselogimise korralgi talle sõnum. Sõnum aga seletab kasutajale, et tal puuduvad vajaliku õigused rakenduse andmebaasile.

Veateadete puhul on kasutatud *same-page error message* ehk sama lehekülje veasõnumite mustrit. Selle mustri puhul kuvatakse veasõnum vormi ülemises osas ning võimaluse korral tuuakse ka vea põhjustanud väljad mingitmoodi esile [16]. Käesoleval juhul, kui jäetakse näiteks parool sisestamata ja kasutaja vajutab sisselogimise nupule, siis muudetakse paroolivälja piirjooned punast värvi ning parooli väli muutub brauseris automaatselt aktiivseks. See annab kasutajale kohest visuaalset tagasisidet ning juhib ta tähelepanu vea põhjustanud väljadele.

Parooli ja kasutajanime tekstisisestusvälja juures on kasutatud *input prompt* ehk sisestusvälja vihje mustrit. Selle mustri järgi pannakse tekstisisestusväljale vihje, mis annab kasutajale teada, mida sinna välja peaks sisestama [16]. Antud muster on hea juhuks, kui konkreetse välja puhul puudub vaikeväärtus, mida süsteem saaks automaatselt alguses ära täita [16]. Parooli ning kasutajanime väli vastavad nendele tingimustele. *Input prompt* mustri kasutamisega saab sisselogimise vormilt jätta kuvamata iga vormivälja kohta sildid (*label*), kuna väljas olev vihje on piisav, et kasutaja saaks aru, mida antud välja tuleks sisestada. Lisaks on väljades ka informatiivsed ikoonid, mis muudavad välja eesmärgi visuaalselt arusaadavaks.

4.3.2 Päis ja jalus

Veebilehekülgedel on välja kujunenud sisu paigutamise osas teatud tavad, mida kasutajad erinevaid lehekülgi külastades eeldavad. Näiteks lehekülje globaalsed navigatsioonivalikud esitatakse tavaliselt alati lehekülje ülaosas [16]. Seega otsustati ka loodud veebiliideses esitada päises navigatsioonivalikud. Olenevalt sellest, kas kasutaja on sisse loginud või mitte ning kas tal on andmebaas valitud, kuvatakse päises infot erinevalt.

Sisselogimata kasutajatele kuvatavas päises puuduvad navigatsioonivalikud, sest järgnevad tegevused vajavad antud rakenduses sisselogimist. Sellisel juhul kuvatakse päises ainult rakenduse nimi ning kasutajaliidese keele valikud.

Joonis 10: Lehekülje päis sisselogimise lehel

Kui kasutaja on sisse loginud, kuid ta ei ole veel andmebaasi valinud, siis kuvatakse talle joonisel 11 nähtav päis. Päis on vertikaalselt kitsam kui sisselogimisel kuvatav päis, sest lehekülje sisu võtab rohkem ruumi. Kui kasutaja on sisseloginud, siis päise struktuur on igal leheküljel sama, erinevad ainult saadavalolevad valikud. See säilitab rakenduse erinevaid vaateid külastades ühtse lehekülje üldilme. Ilma andmebaasi valimata näidatakse lisaks keele valikule ka välja logimise nuppu ning kasutajanime.

Joonis 11: Lehekülje päis, kui andmebaas on valimata

Kui kasutaja on sisse loginud ja ka andmebaas on valitud, siis kuvatakse talle joonisel 12 nähtav päis. Sellisel juhul on päises esindatud kõik valikud, mida antud rakenduses teha saab. Lehekülg, kus kasutaja hetkel asub, on värviga teistest esile tõstetud. Kasutajanime kõrval kuvatakse nüüd ka kasutaja poolt valitud andmebaas. Sellised lahendused võimaldavad kasutajal päist vaadates kergesti rakenduses orienteeruda.

Joonis 12: Lehekülje päis, kui andmebaas on valitud

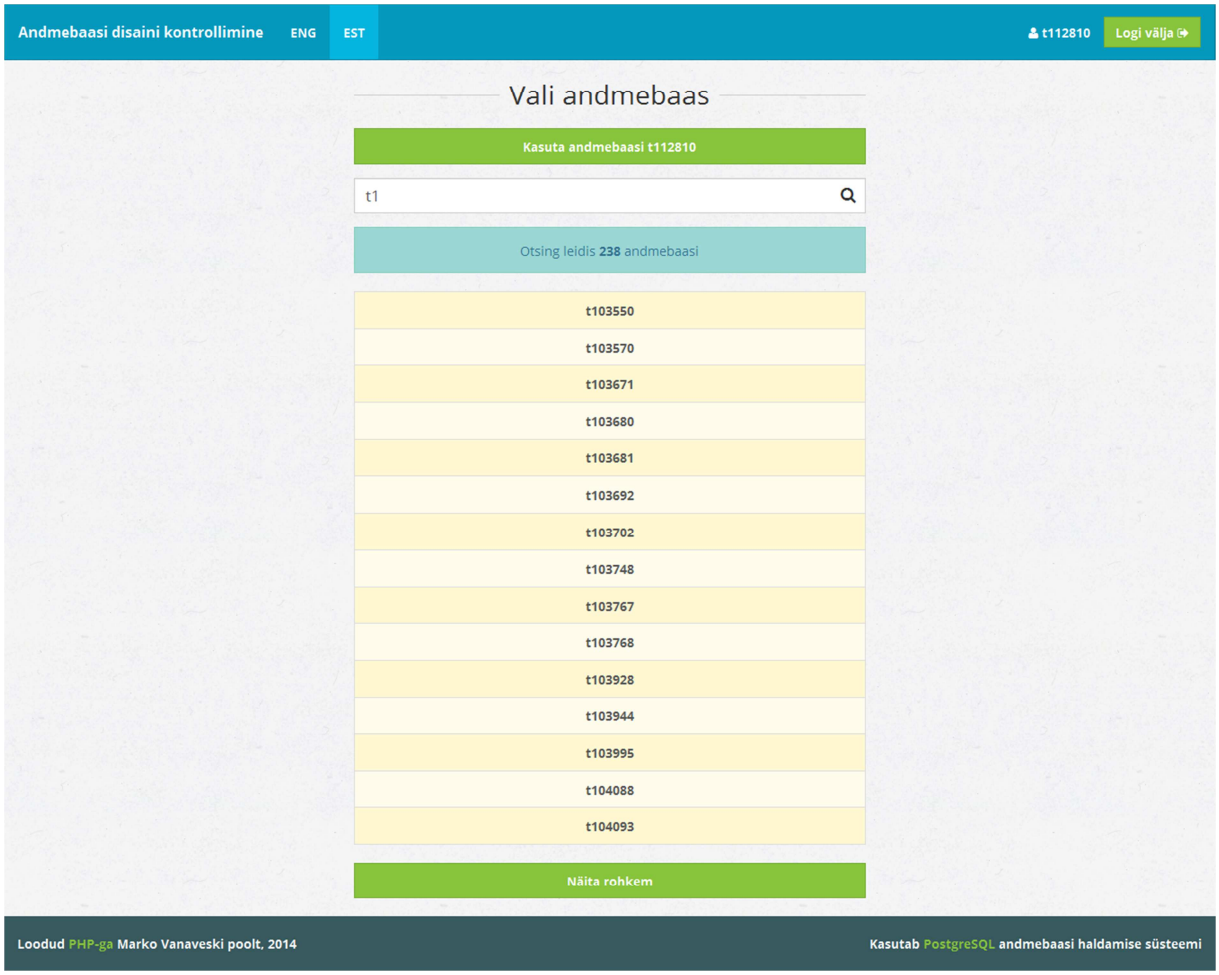
Lehekülje lõpus esitatakse igas vaates muutumatus jääv jalus ehk *footer*. Jalusel kuvatakse kasutajale infoks põhilised tehnoloogiad, mida rakendus kasutab ning samuti rakenduse autori nimi.

Joonis 13: Lehekülje jalus

4.3.3 Andmebaasi valimise vaade

Pärast sisselogimist kuvatakse kasutajale andmebaaside valimise lehekülg. Antud lehekülje eesmärgiks on minimeerida järgnevatel sammudel kasutaja poolset väljade täitmise ja muutmise vajadust. See tähendab, et alguses valitakse ära andmebaas, mille peal päringuid ja teste jooksutada. Hiljem liikudes päringute ning testide lehele on kasutaja poolt alguses valitud andmebaas juba eelnevalt andmebaasi vaikeväärtuseks. Loomulikult saab hiljem andmebaasi valikut muuta.

Loodud rakendus on mõeldud ka kasutamiseks TTÜ õppeaine Andmebaasid II õppevahendina. Igale selles osalevale üliõpilasele luuakse õppejõu poolt andmebaasiserverile kasutaja, kelle kasutajanimi on sama kui üliõpilase matriklinumber. Lisaks on väga paljudel üliõpilastel nende poolt loodud andmebaasi nimi sama, mis üliõpilase matriklinumber. Seetõttu otsustati luua andmebaasi valimiseks lahendus, kus kasutajale pakutakse kõigepealt tema kasutajanimega samanimelist andmebaasi, kui see eksisteerib. Selline lahendus võimaldab paljudel kasutajatel valida andmebaas üheainsa klõpsuga. Kui sama kasutajanimega andmebaasi ei eksisteeri, siis antud valik puudub.



Joonis 14: Andmebaasi valimise vaade

Realiseeritud on ka andmebaaside otsing. Kuna andmebaase võib serveril olla väga palju (töö kirjutamise hetkel oli apex.ttu.ee serveril 352 PostgreSQL andmebaasi), siis otsustati piirata algselt korraga nähtavate andmebaaside arvu viieteistkümneni. Kui kasutaja soovib, saab ta kuvada rohkem andmebaase. Iga nupuvajutusega lisatakse nimekirja järgnevad 15 andmebaasi. Tegemist on *infinite list* ehk lõpmatu nimekirja mustriga, kus korraga kuvatakse osa andmetest ning kasutaja saab soovi korral andmeid juurde kuvada [16]. Ka andmebaasi otsimise vormi puhul on kasutatud *input prompt* kasutajaliidese mustrit nagu oli kasutatud sisselogimise vormil peatükis 4.3.1.

4.3.4 Tegevuse valimise vaade

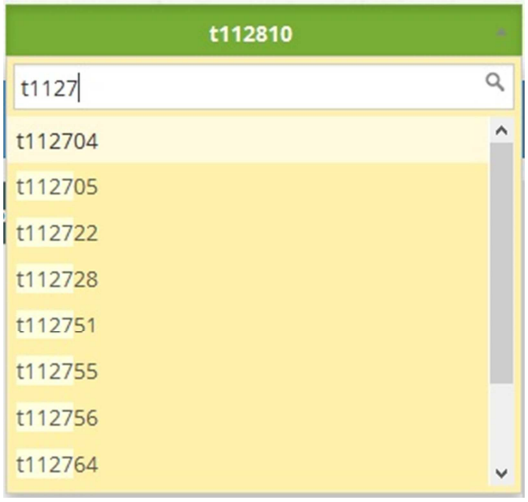
Pärast andmebaasi valimist kuvatakse kasutajale tegevuste valimise vaade. Tegevuse valimise lehekülg on ka kogu rakenduse avaleheks, kui kasutaja on sisse loginud ning andmebaas on valitud. Ehk kui kasutaja klõpsab päise vasakus nurgas oleva rakenduse nime peal, siis kuvatakse talle tegevuste valimise vaade. Antud leheküljel saab kasutaja valida, kas ta soovib

käivitada üksikudi päringuid või teste. Saadavalolevad valikud on esitatud suurte ristkülikukujuliste elementidena ning järgivad *grid of equals* mustrit. Selle mustri eesmärk on objektide, millel on sarnane välimus ning tähtsus, esitamine ruudustiku või maatriksina [16]. See muster sobib just olukorda, kus kasutaja peab tegema valiku võrdväärsete valikute vahel, milleks antud juhul on edasise tegevuse valimine. Ruudustik annab igale valikule võrdselt ruumi ning seetõttu tekitab kasutajas tunde, et mõlemad valikud on samaväärsed [16]. Tegevuste nimed on värviga esile toodud ning nimede all järgneb lühike informatiivne kirjeldus antud tegevusest.

Joonis 15: Tegevuse valimise vaade

4.3.5 Testide valimise vaade

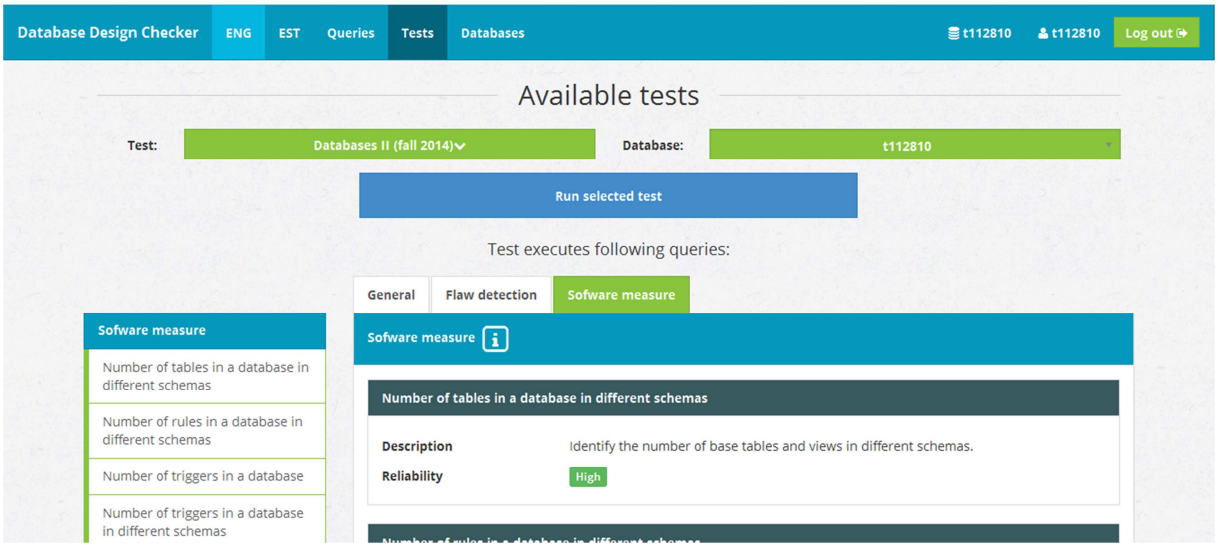
Testide käivitamise alal on juurdepääs andmebaaside valimise funktsionaalsusele. Antud väljas on vaikimisi väärtuseks kasutaja poolt peale sisselogimist valitud andmebaasi väärtus. Suur osa kasutajatest seda väärtus selles vaates enam ei muuda, vaid käivitavad päringuid ja teste algselt valitud andmebaasi peal. Antud funktsionaalsus on aga oluline neile, kes soovivad otsida disainivigu mitmest erinevast andmebaasist. Siinkohal saab näiteks tuua jälle „Andmebaasid II“ õppeaine, kus õppejõud soovib kontrollida mitme üliõpilase andmebaaside vastavust teatud tingimustele. Sellisel juhul saab ta otse testide leheküljelt valida uue üliõpilase andmebaasi, ilma andmebaasi valimise leheküljele minemata.



Joonis 16: Andmebaasi valimine päringute või testide käivitamise vaates

Rippmenüüs kuvatakse kõik saadaolevad andmebaasid antud serveril. Sisestades otsinguvälja teksti, filtreeritakse tulemused automaatselt. Lisaks tuuakse värviga esile kattuv osa sisestatud otsingusõnast ning andmebaasi nimest.

Andmebaasi ning kategooriate valimise juures on kasutatud *dropdown chooser* mustrit. Tegemist on mustriga, kus nupuvajutuse peale avaneb rippmenüü, kust kasutaja saab valida talle sobiva väärtuse [16]. Selline lahendus võimaldab muidu pika ning keerulise valiku esitada kompaktsemalt väiksemal ekraanipinnal ning peidab kasutaja eest pikad valikud ära seni, kuni kasutaja ise otsustab seda muuta [16].



Joonis 17: Testide valimise vaade

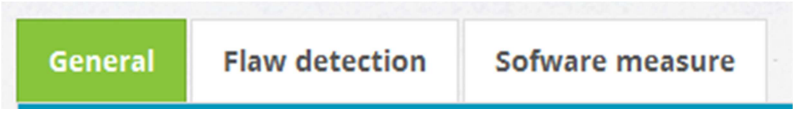
Päringute kuvamist ja tulemuste näitamist tutvustatakse järgmises peatükis.

4.3.6 Päringu ja tema tulemuste kuvamine

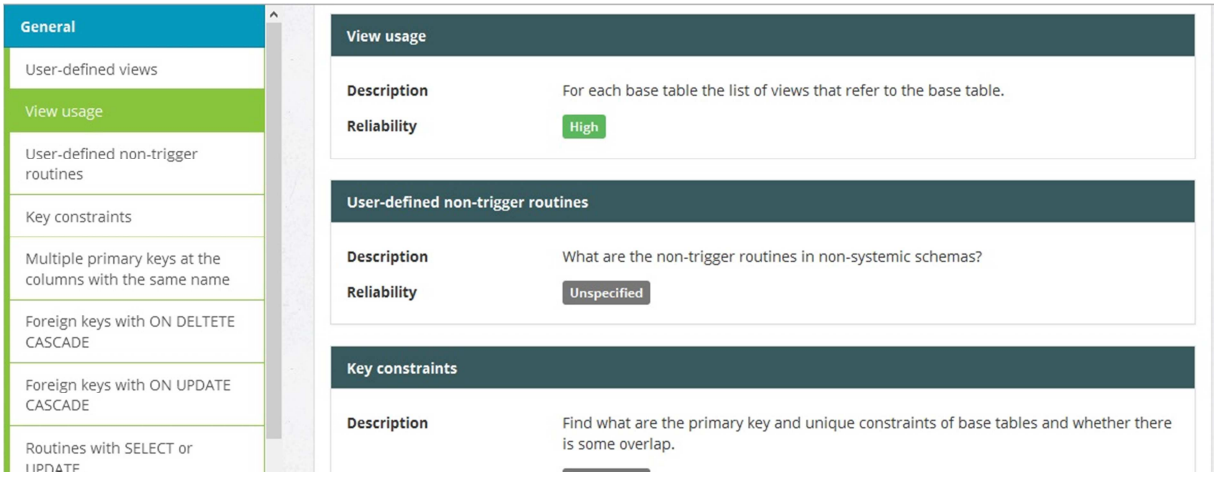
Päringute ja tema tulemuste kuvamiseks on loodud ühine vaade. Seda vaadet kasutatakse nii testide leheküljel seal sisalduvate päringute tutvustamiseks kui ka üksikute päringute valimise lehel ja testide lehel käivitatud päringute tulemuste esitamiseks. Erinev on lihtsalt päringute grupeerimise alus. Üksikute päringute käivitamise leheküljel toimub päringu kuvamine teistmoodi, seda seletatakse jaotises 4.3.6.

4.3.6.1 Kuvamine ilma tulemusteta

Testide valimise leheküljel kuvatakse kõigepealt päringud koos nende infoga. Testis sisalduvad päringud on jaotatud päringu tüübi alusel erinevatesse paneelidesse, mida saab valida vastava tüübi saki ehk *tab*'i peale vajutades. Sakkide ehk *tab*'ide kasutamine vastab mustrile *module tabs*. Idee seisneb selles, et sisu jaotatakse mingi kindla kriteeriumi alusel osadeks ning korraga kuvatakse kasutajale ainult üks osa [16]. Kuvatavat osa saab kasutaja vahetada sakkide abil.



Joonis 18: Päringute grupeerimise sakid

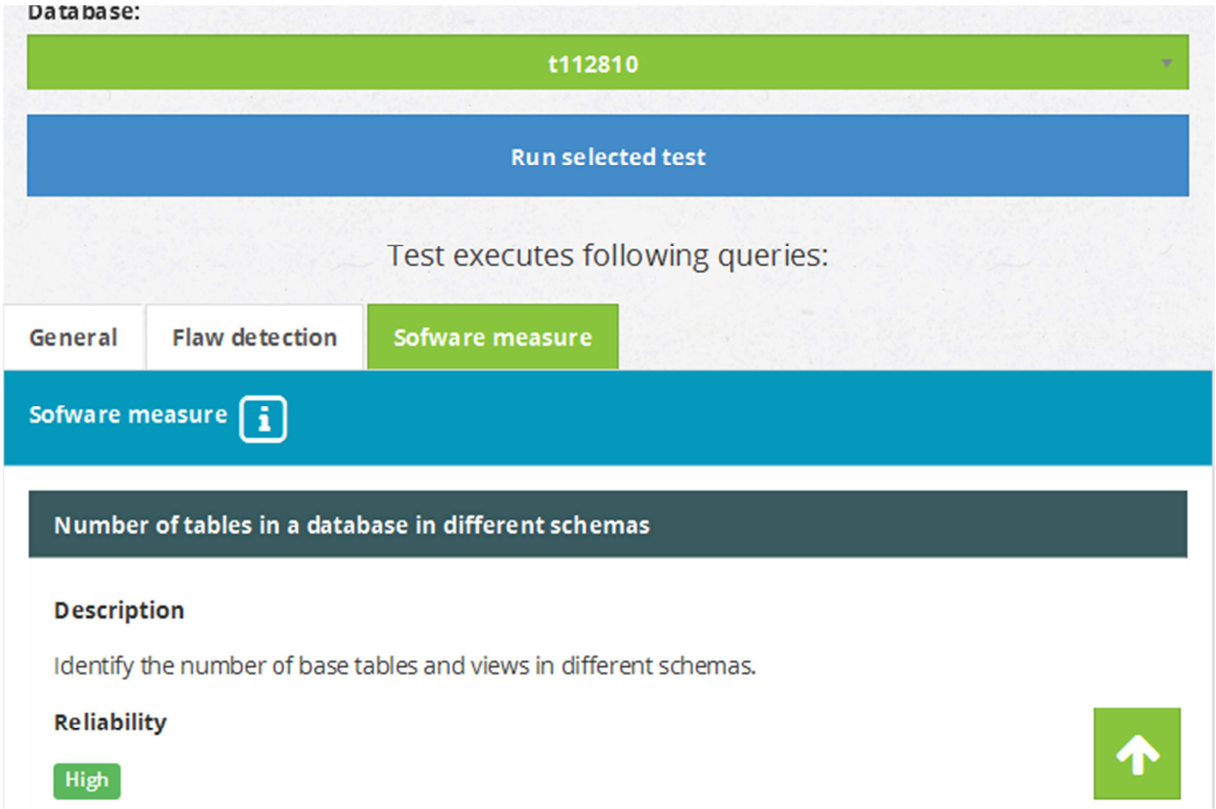


Joonis 19: Päringute kuvamine ilma tulemusteta

Paremal pool kuvatakse külgriba päringute nimega. Isegi kui päringuid on palju, võimaldab selline lahendus kergesti aru saada, millise päringu juures kasutaja parajasti on, sest ekraani vaateväljas olev aktiivne ülemine päring on külgribal värviga kergesti eristatav. Kasutaja võib

külgribal klõpsata ükskõik millise päringu nime peal ning süsteem kerib lehekülje automaatselt valitud päringu juurde. Isegi kui päringuid on palju, on külgriba alati nähtaval ja liigub koos lehekülge kerimisega kaasa. See võimaldab kasutajal kergemini orienteeruda.

Väiksemate resolutsioonidega ekraanide puhul ei mahuks külgriba ning päringute info ala korraga kõrvuti ekraanile ära. Seetõttu otsustati väiksemate resolutsioonide puhul külgriba ära peita.



Joonis 20: Päringute kuvamine väikese resolutsiooniga seadmetel

Külgribast vasakul pool kuvatakse päringute detailinfot. Seal sisaldub näiteks nende kirjeldus ja usaldusväärsus. Usaldusväärsus on tähistatud ka värviga, mis vastab tema kontekstile. Vähese usaldusväärsusega päringud on tähistatud punasega, mis seostub kasutajatel tavaliselt ohuga või sellega, et midagi on valesti. Kõrge usaldusväärsusega päringud on tähistatud rohelisega ning keskmise usaldusväärsusega oranžiga. Päringud, mille usaldusväärsus pole teada, on tähistatud neutraalse halliga.

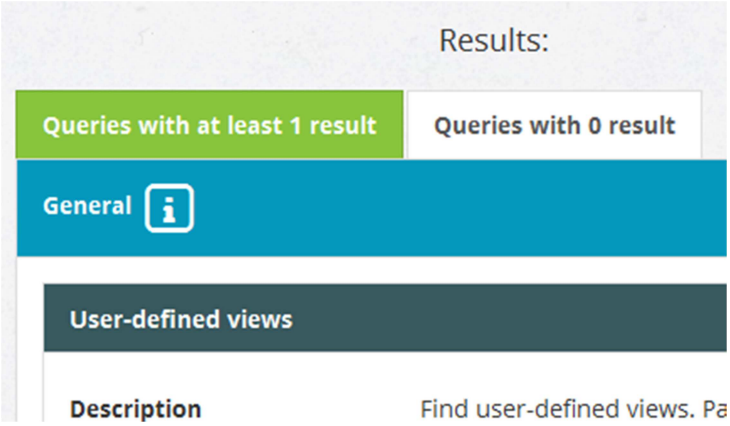
Päringute kuvamise puhul on kasutatud *titled section* ehk pealkirjastatud sektsioonide mustrit. Selle mustri põhjal jaotatakse sisu sektsioonideks, millele antakse visuaalselt eristatav

pealkiri, mis aitab kasutajatel selgelt eristada erinevaid sektsioone [16]. Igal päringu on visuaalselt selgelt eristatav element leheküljel.

4.3.6.2 Kuvamine koos tulemustega

Kui kasutaja poolt valitud päringud või tema poolt valitud testis sisalduvad päringute käivitamine on lõppenud, kuvatakse talle nende päringute poolt leitud tulemused. Tulemuste kuvamine on lahendatud samamoodi nii üksikute päringute kui ka testide puhul. Päringud on jaotatud sakkide abil kahte alasse:

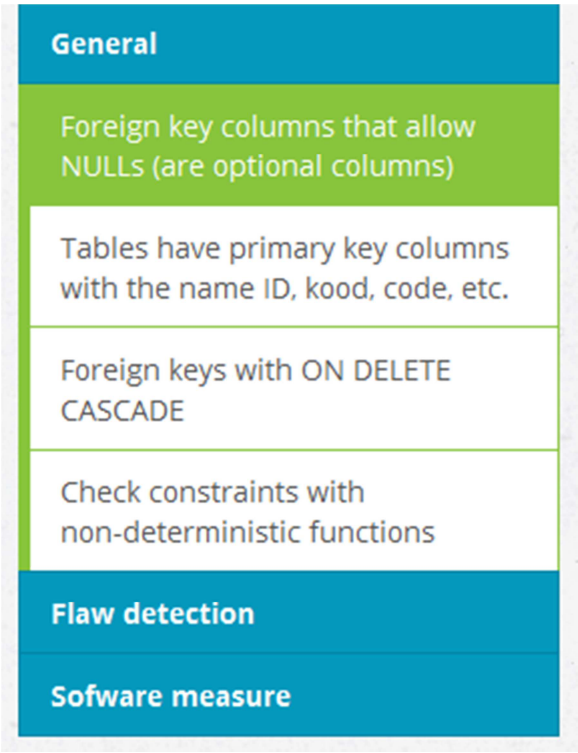
- päringud, mis tagastasid vähemalt ühe tulemuse
- päringud, mis ei tagastanud ühtegi tulemust



Joonis 21: Päringu tulemuste grupeerimine sakkide abil

Sellise jaotamise abil leiab kasutaja kergesti ülesse päringud, mis võivad viidata mingile konkreetsele andmebaasi disaini veale.

Külgribal on tulemuste näitamise puhul grupeeritud päringud tüübi järgi. Joonis 18 järgi on näha ka, et jaotises 4.3.6.1 mainitud *titled section* mustrit on kasutatud ka päringutüüpide puhul, kus iga tüüp on selgelt eristatav sektsioon leheküljel.



Joonis 22: Külgriba päringute tulemuste näitamise puhul

Päringu poolt leitud tulemuste kuvamine kasutab samasugust elementi nagu ilma tulemusteta päringu detailinfo kuvamine. Päringu info lõppu on lisandunud aga tulemuste tabel. Käivitatud päringute tulemused kuvatakse samamoodi nii üksikute päringute kui ka testide lehel. Kuna päring võib tagastada väga palju ridu ning igas reas võib olla teadmata arv veerge, siis otsustati kõikide tulemuste kuvamiseks luua eraldi aken.

Constraint naming

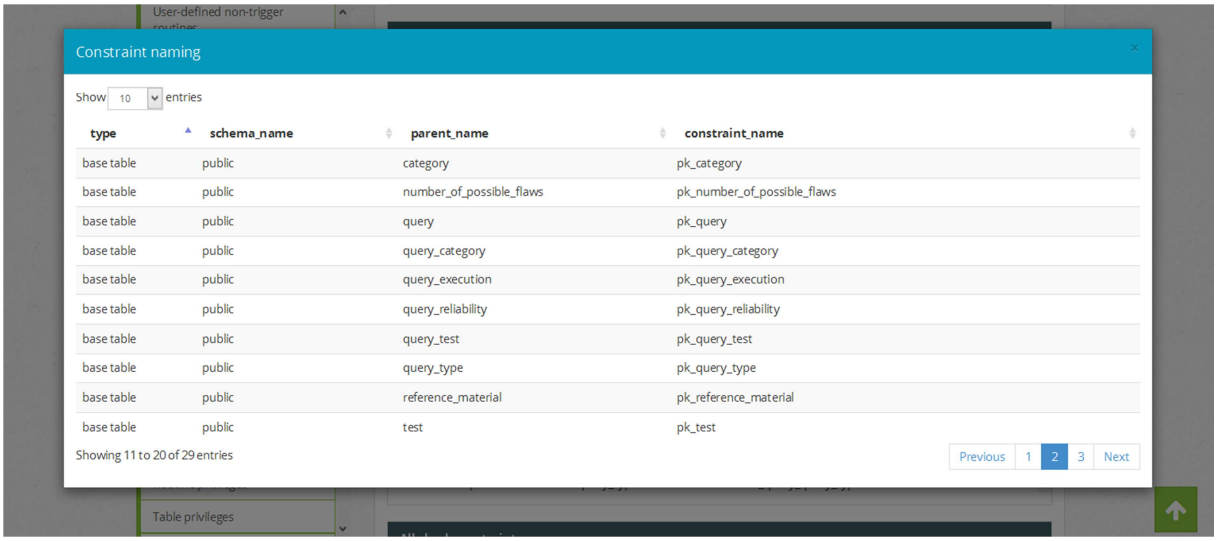
Description	To find out how consistent are the constraint names and whether users have given themselves the names.
Reliability	High
Results	Show all results

type	schema_name	parent_name	constraint_name
domain	public	domeen_nimetus	chk_ei_ole_tyhi
domain	public	d_name	chk_name_emptystring
base table	public	number_of_possible_flaws	chk_number_of_possible_flaws
domain	public	d_desc	chk_text_emptystring
base table	public	query_execution	fk_number_of_possible_flaws_query_execution
base table	public	category	fk_query_category_category
base table	public	query	fk_query_category_query
base table	public	query	fk_query_execution_query
base table	public	query_reliability	fk_query_query_reliability
base table	public	query_type	fk_query_query_type

Joonis 23: Päringu kuvamine koos tulemustega

Tulemuste esitamise tabelis on kasutatud andmete paremaks eristamiseks *row striping* mustrit ehk tabeli read on vaheldumisi kahe sarnase varjundiga värviga [16]. Selline lahendus aitab kasutajal visuaalselt eristada tabeli erinevaid ridu ja lugeda ridu vasakult paremale ilma ridu segamini ajamata [16].

Üldvaates on samuti esitatud kõik päringu tulemused, kuid nad on piiratud kindla suurusega ala sisse ning ala suuruse ületamisel tekivad kerimisribad. Kui kasutaja soovib päringu tulemusi täpsemalt sorteerida ning paremat ülevaadet saada, siis saab nupuvajutuse peale avada konkreetse päringu tulemuste akna. Aken on modaalaken ehk see avaneb olemasoleva sisu peale.



Joonis 24: Päringu tulemuste detailaken

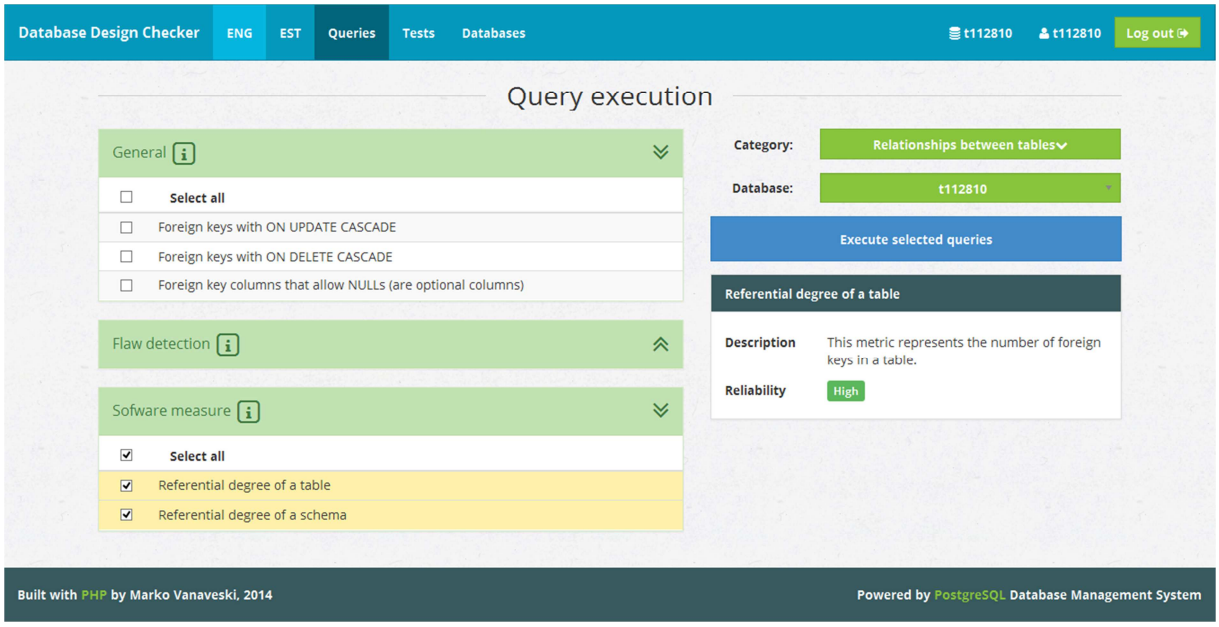
Tulemused esitatakse jälle tabeli kujul. Erinevus eelmise tabeliga seisneb selles, et nüüd on kasutajal suurem kontroll tulemuste sorteerimise ja kuvamise üle. Kuna tulemusi võibolla väga palju, siis otsustati kasutada *pagination* ehk lehekülgedeks jagamise mustrit [16]. Ühe leheküljel kuvatavate tulemuste arvu saab kasutaja ülevalt vasakult nurgast valida. Lehekülgedeks jagamine annab kasutajatele suurema kontrolli info kuvamise üle.

Antud tabeli juures otsustati kasutada ka *sortable table* ehk sorteeritava tabeli mustrit. Selle mustri põhjal saab kasutaja tabeli ridu sorteerida vastavalt veergude väärtuste järgi [16]. See võimaldab kasutajal näiteks tulemusi sorteerida andmebaasi tabelite nimede järgi.

Sorteerimine ja lehekülgedeks jagamine toimub kõik JavaScripti abil ilma serveri poole pöördumata. See tagab väga sujuva ja kiire sorteerimise ning lehekülgedeks jagamise.

4.3.7 Päringute valimise vaade

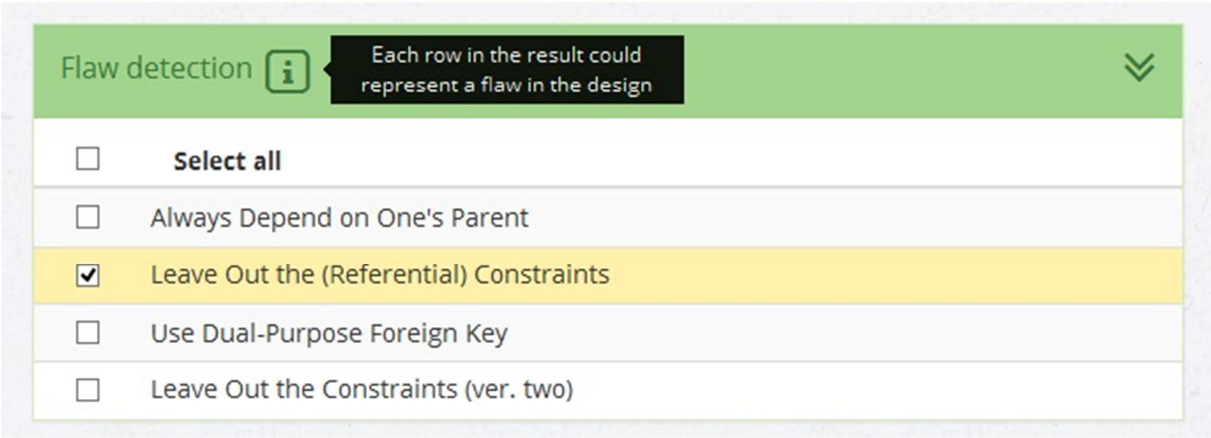
Päringute valimise vaates kuvatakse kindla kategooria alla kuuluvaid päringuid. Lehekülg on jagatud kaheks osaks: vasakpoolne päringute nimekiri ning parempoolne info/käivitamise ala.



Joonis 25: Päringute valimise vaade

Vasakpoolses päringute nimekirja alas kuvatakse valitud kategooria päringuid, mis omakorda on grupeeritud päringute tüübi järgi. Iga päringu tüüp on omaette plokk, mida on visuaalselt lihtne eristada. Päringu tüübi rohelist värvi nime ala peal klõpsates on võimalik antud tüüpi päringud peita. Tegemist on seega *collapsible panel* mustriga, kus kasutaja saab paneele enda tahte järgi sulgeda ning avada [16]. See võimaldab kasutajal peita teda mitte huvitavat tüüpi päringud näiteks kompaktsema vaate saamise eesmärgil ning see annab kasutajale suurema kontrolli teda huvitava info filtreerimiseks. Klõpsatavaks on tehtud terve päringu tüübi päis, mitte ainult noolte ikooni osa. Tänu sellele ei pea kasutajad hiirega klõpsama ainult väikese pindalaga ikoonil ning see parandab kasutajamugavust. Iga tüübi nime kõrval on infoikoon, millele hiirega liikudes või klõpsates ilmub nähtavale tüübi kirjelduse tekst.

Päringu tüübi plokis on kasutatud samuti jaotises 4.3.6.2 mainitud *row striping* mustrit tabeli andmete paremaks eristamiseks.



Joonis 26: Päringu tüübi kuvamine

Liikudes hiirega üle päringu, kuvatakse lehekülje paremal pool detailinfot valitud päringu kohta. Selline lahendus võimaldab kasutajal ise valida, milliste päringute kohta ta detailsemat informatsiooni näha soovib. Lehekülje paremal pool asuv päringute info ja käivitamise plokk liigub lehekülge kerides kaasa. See võimaldab kasutajatel valida uut kategooriat ning lugeda päringu detailinfot isegi siis, kui päringuid on palju ning need ei mahu korraga ekraani vaatevälja ära. Päringu info esitamine on visuaalselt samasugune nagu jaotises 4.3.6.1 esitatud kirjelduses.

Päringute valimise leheküljel asub sama andmebaasi valimise rippmenüü, mis testide valimise lehelgi. Testide valimise rippmenüü on asendatud kategooriate valimise omaga. Nende põhimõtted ja kasutatud mustrid on samad, mis peatükis 4.3.5 kirjeldatud juhul.

5. Rakenduse arhitektuur

Selles peatükis kirjutatakse rakenduse tehnilisest arhitektuurist.

5.1 Eessüsteemi ehitamine

Eessüsteem ehk *front-end* on veebilehele ilmuv kasutajaliides, mis võimaldab veebilehe külastajal suhelda lehekülje dünaamiliste osadega nagu näiteks andmebaasid [1]. Eessüsteemi arendamiseks võeti lisaks tavapärastele HTML-i, JavaScripti ning JavaScripti raamistiku jQuery ja tema moodulite kasutamisele tarvitamisele ka Twitter Bootstrap ning LESS eelprotsessor. JavaScript'i ja selle raamistikku jQuery kasutatakse lehekülje interaktiivsemaks tegemiseks. Näiteks on realiseeritud nende abil laadimisanimatsioonid, elementide dünaamiline nähtavaks muutmine jne. Lisaks toimub osa andmevahetust serveri poolega asünkroonselt läbi JavaScripti, kasutades selleks AJAX tehnoloogiat ehk asünkroonset andmevahetust JavaScripti abil.

5.1.1 Twitter Bootstrap

Käesoleva töö kasutajaliidese loomiseks otsustati kasutada Twitter Bootstrap 3 raamistikku. Tegemist on HTML, CSS ja JavaScripti avatud lähtekoodiga raamistikuga, mis on mõeldud just paindliku disainiga lehekülgede ehitamiseks [17]. Seega sobis raamistik antud rakenduse jaoks. Bootsrap raamistik sisaldab mitmeid kasutusvalmis komponente, mis lihtsustavad kasutajaliidese arendamist. Esimene versioon Bootstrapist avalikustati 2010. aastal.

5.1.2 LESS eelprotsessor

Töös kasutatakse ka LESS eelprotsessorit, mis laiendab CSS laadistikku. LESS loodi 2009. aastal Alexis Sellier'i poolt ning oli esialgselt kirjutatud Ruby programmeerimiskeeles, kuid hiljem kirjutati JavaScripti peale ümber [4].

LESS lisab juurde uusi võimalusi CSS-i kirjutamiseks ning muudab haldamise lihtsamaks. See on täielikult tagurpidi ühilduv CSS süntaksiga. See tähendab, et iga korrektne CSS koodijupp sobib kasutamiseks ka LESS failides. Näiteks saab deklareerida muutujaid. Selle asemel, et mitmes kohas kirjutada uuesti vajalikku RGB värvikoodi, saab defineerida muutuja ning väärtustada see vajaliku värvikoodiga. Kui mitmetes elementides on kasutusel sama värvikood, siis saab selle asemel kirjutada eelnevalt defineeritud muutuja nime. Kui tulevikus

tekib vajadus elementide värvi muutmiseks, piisab ainult muutuja väärtuse ülekirjutamisest ning ei pea nägema vaeva, et otsida kõik antud värvikoodi esinemised failidest ülesse.

Twitter Bootstrapi CSS on loodud LESS failide põhjal [17]. Antud rakenduse puhul kasutatakse Bootstrapi puhul mitte valmiskompileeritud *.css* faile, vaid *.less* faile. Selline lahendus võimaldab kergesti modifitseerida Bootstrapi stiili konkreetse rakenduse jaoks sobivamaks. Antud rakenduses on mitmed Bootstrapi poolt defineeritud muutujad uute väärtustega ülekirjutatud.

```
@import "../bootstrap/bootstrap.less";

@import "../bootstrap/bootstrap-chosen-variables.less";
@import "../bootstrap/bootstrap-chosen.less";

//------------------------Bootstrap variable override-----------
@icon-font-path:          "/queries/fonts/";

@border-radius-base:        0px;
@border-radius-large:       0px;
@border-radius-small:       0px;
```

Joonis 27: Bootstrapi muutujate muutmine

Joonis 27 on näha autori poolse *.less* faili sisu algus. Faili imporditakse vajalikud Bootstrapi muutujaid sisaldavad failid ning muutujate väärtused kirjutatakse üle. Sellele järgnevad juba autori poolt kirjutatud CSS stiilid ja muutujad. Fail *bootstrap.less* on baasfail Bootsrapi *.less* failide kompileerimiseks üheks *.css* failiks. Selles failis sisalduvad ainult import laused, mis lisavad juurde kõik Bootsrapi poolt kasutatavad *.less* failid. Pärast kompileeritakse kogu fail CSS formaati. Kompilaator kasutab viimasena ülekirjutatud muutujate väärtusi. Bootstrapi *.less* failid asuvad eraldi kataloogis muutumatuna ning nendes failides muutujate väärtusi ei muudeta. Sellise struktuuriga saab lihtsasti uuendada Bootstrapi faile uuele versioonile ning kõik üleväärtustaud muutujad jäävad samaks.

LESS eelprotsessori *.less* laiendiga faile saab kompileerida brauseritele sobivaks *.css* laiendiga failideks. Selleks on olemas nii käsurea põhine vahend, mitmeid kolmanda osapoolte loodud eraldi installitavaid graafilise kasutajaliidesega programme, kui ka veebirakendusi [4].

5.2 Rakenduse serveripoolse osa arhitektuur

Rakenduse serveripoolne osa on kirjutatud PHP programmeerimiskeeles. Kasutusele võeti Symfony PHP raamistik, täpsemalt teine versioon ehk Symfony2. Raamistik otsustati kasutusele võtta eelkõige selle pärast, et see lihtsustab rakenduse loomist. Raamistikku kasutades on rakendus kirjutatud kindlat arhitektuuri järgides ning raamistiku komponentide abil on lihtsamad probleemid kergesti lahendatavad. Lisaks on Symfony raamistik väga põhjalikult dokumenteeritud ning väga lihtsasti jälgitav. Seetõttu, kui tulevikus on vaja programmis muudatusi teha, saab keegi kolmas isik Symfony dokumentatsiooni järgides kergemini rakenduse arhitektuurist aru.

5.2.1 Symfony

Symfony autori F.Potencier'i sõnul on Symfony korduvkasutatavate ja eraldiseisvate PHP komponentide kogum [18]. Antud komponendid moodustavadki raamistiku. Symfony pakub nii kontrolleri (*controller*) kui ka vaate (*view*) osa. Kontroller juhib rakenduse tööd. Symfony raamistikus on kontrolleri ülesanne tagastada *Response* objekt, mis saadetakse tagasi kliendile [19]. *Response* objekt on Symfony raamistiku objekt ning sisaldab muuhulgas HTTP päiseid ning kliendile tagastatavat sisu. Sisuks võib olla näiteks HTML lehekülg, pildifail, JSON objekt või mingi muu fail [19].

Üldiselt saab iga Symfony rakenduse elutsükli jagada dokumentatsioonis kirjeldatud viisil neljaks osaks [19].

1. Eeskontroller (*front controller*), milleks on *app.php* või *app_dev.php*, mis võtab kliendilt tulnud päringu vastu ja käivitab kõik Symfony rakenduse tööks vajalikud komponendid.

2. Marsruuter (*Router*) inspekteerib tulnud päringut (URL aadress) ja otsib sobiva kontrolleri ja vastava funktsiooni antud kontrolleris, mis selle päringu jaoks on eelnevalt defineeritud.

3. Leitud kontroller käivitab vastava funktsiooni ning loob lõpuks *Response* objekti.

4. *Response* objektis olevad HTTP päised ja sisu saadetakse kliendile tagasi. Sisuks võib olla HTML, XML või mingis muus formaadis fail.

Symfony raamistikku kasutades on kasutatavad paljud Symfony teenused. Teenus on tavaline PHP objekt, mis täidab mingit kindlat ülesannet [19]. Lisaks saab kasutada ka sõltuvuste sisestamise ehk *dependency injection* komponenti. See tähendab, et objekti loomisel antakse objektile automaatselt kaasa mingid ressursid [19]. Sellisel juhul saab teenuste konfigureerimisel ära määrata argumendid, mis antakse automaatselt objekti loomisel konstruktori argumentidena kaasa. Argumentideks saab olla näiteks mingi teine teenus. Ka loodud rakenduses defineeriti osad objektid teenustena ning kasutati sõltuvuste sisestamist, et anda objektile tööks vajalikud ressursid kohe kaasa.

```
services:
    query_authenticator:
        class:     t112810\QueriesBundle\Security\QueryAuthenticator
        arguments: [@translator, @database_provider_factory]
```

Joonis 28: Teenuste defineerimine konfiguratsioonifailis

5.2.2 Kontrollerid

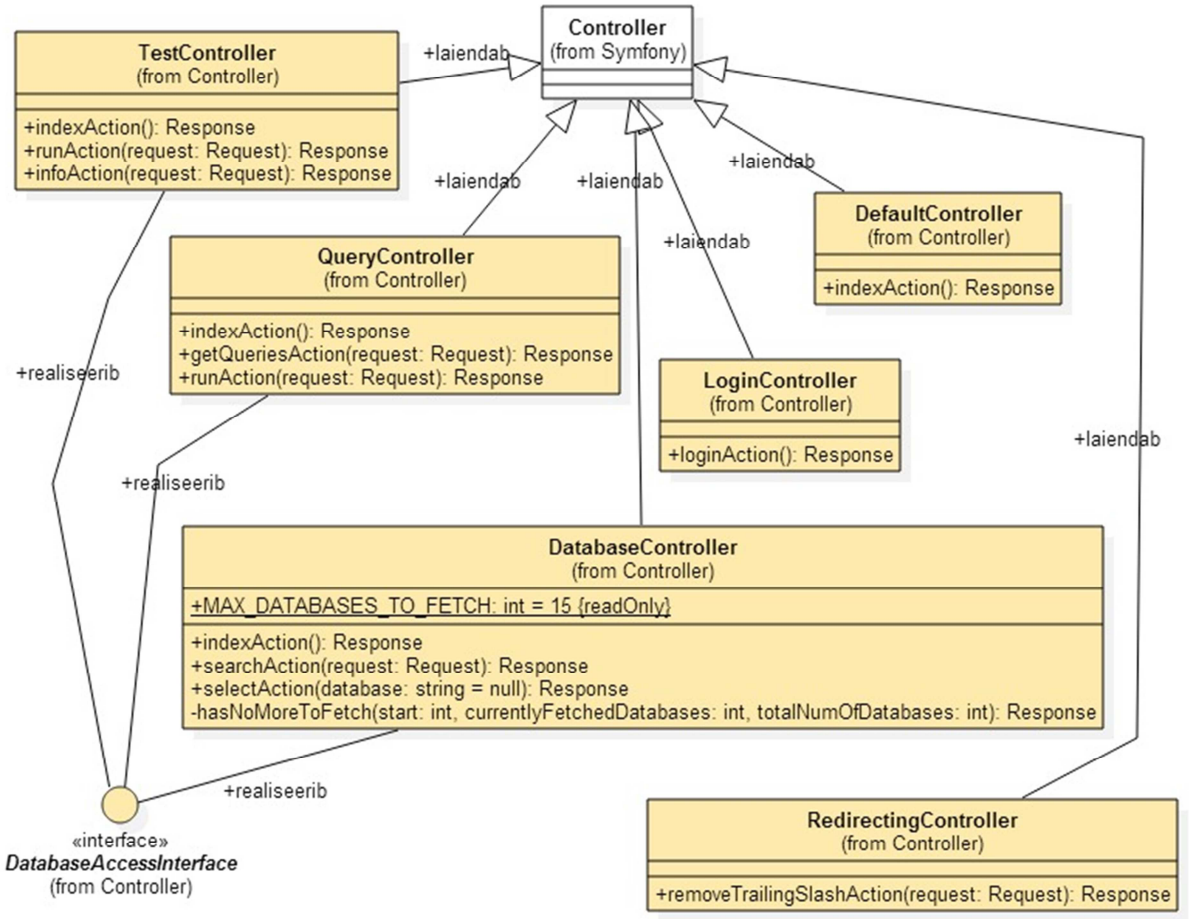
Iga kliendilt tulnud päringuga seotakse üks kindla kontrolleri funktsioon. Seda juhul, kui vastava päringu uurimisel leitakse eelnevalt konfigureeritud marsruutidest kontroller, mis peab seda päringut täitma. Selleks tuleb ära defineerida marsruudid, mis määravad ära, millise kontrolleri milline funktsioon käivitatakse teatud tingimustele vastava HTTP päringu puhul. Marsruutide leidmisega tegeleb Symfony *Route* objekt. Marsruutide ja kontrollerite vastavus on defineeritud *routing.yml* konfiguratsioonifailis.

```
39  tests:
40      path:       /{_locale}/tests
41      defaults:   { _controller: t112810QueriesBundle:Test:index, _locale: en }
42      methods:    [GET]
43      requirements:
44          _locale: en|et
45
46  run_test:
47      path:       /{_locale}/tests/run
48      defaults:   { _controller: t112810QueriesBundle:Test:run, _locale: en }
49      methods:    [POST]
50      requirements:
51          _locale: en|et
52
53  test_info:
54      path:       /{_locale}/tests/info
55      defaults:   { _controller: t112810QueriesBundle:Test:info, _locale: en }
56      methods:    [GET]
57      requirements:
58          _locale: en|et
```

Joonis 29: marsruutide määramine konfiguratsioonifailis

Näiteks, kui klient teeb HTTP GET päringu aadressile */et/tests*, siis käivitatakse *TestController* kontrolleri meetod *indexAction*. Symfony nimetamistavade puhul on kontrolleri klassi nime lõpus alati sõna *Controller* ning kontrolleri meetodi nime lõpus sõna *Action*. Kui järgida neid tavasid, siis võib marsruutide defineerimisel sõnad *Controller* ja *Action* ära jätta, nagu ülevalolevalt pildilt näha võib. See võimaldab lühemalt ning arusaadavamalt marsruudid ära seadistada. Pildi pealt on ka näha, et saab ära määrata, milliseid HTTP meetodeid toetatakse (näiteks GET või POST). Lisaks on ära defineeritud ka *_locale* muutuja, millel on kaks võimalikku väärtust: *en* ja *et*. Tegemist on kasutajaliidese keelt määrava parameetriga. Kui külastada URL-i ilma selle parameetrita, siis vaikimisi määratakse *_locale* parameetri väärtuseks *en*, ehk kasutajale kuvatakse inglise keelse kasutajaliidesega lehekülg.

Antud rakenduses loodi 6 kontrollerit, mis kõik laiendavad Symfony *Controller* baasklassi. Kontrollereid saab luua ka ilma Symfony kontrollerit laiendamata, kuid laiendamise puhul on tagatud ligipääs lisafunktsionaalsusele ja meetoditele, mis muudavad rakenduse programmeerimise lihtsamaks.



Joonis 30: Rakenduse kontrollerid

Testide valimise ja käivitamisega on seotud *TestController*. Üksikute päringute valimise ja käivitamisega on seotud *QueryController*. Andmebaaside valimise ja otsimisega on seotud *DatabaseController* ja kasutajate sisselogimisega on seotud *LoginController*.

Lisaks on loodud ka *DefaultController*, mis käivitub siis, kui kasutaja on sisse loginud ja külastab rakenduse juuraadressi. Antud kontroller kontrollib, kas kasutaja on andmebaasi valinud või mitte. Kui andmebaas on valimata, siis suunatakse päring edasi *DatabaseController*'ile ja tema *indexAction* meetodile. Selle meetodi töö tulemusena kuvatakse kasutajale andmebaaside valimise lehekülg. Kui andmebaas on valitud, siis suunatakse samuti edasi *DatabaseController*'ile, kuid *selectAction* meetodile. Antud meetodi töö tulemusega kuvatakse kasutajale valik, kas ta soovib käivitada teste või üksikuid päringuid.

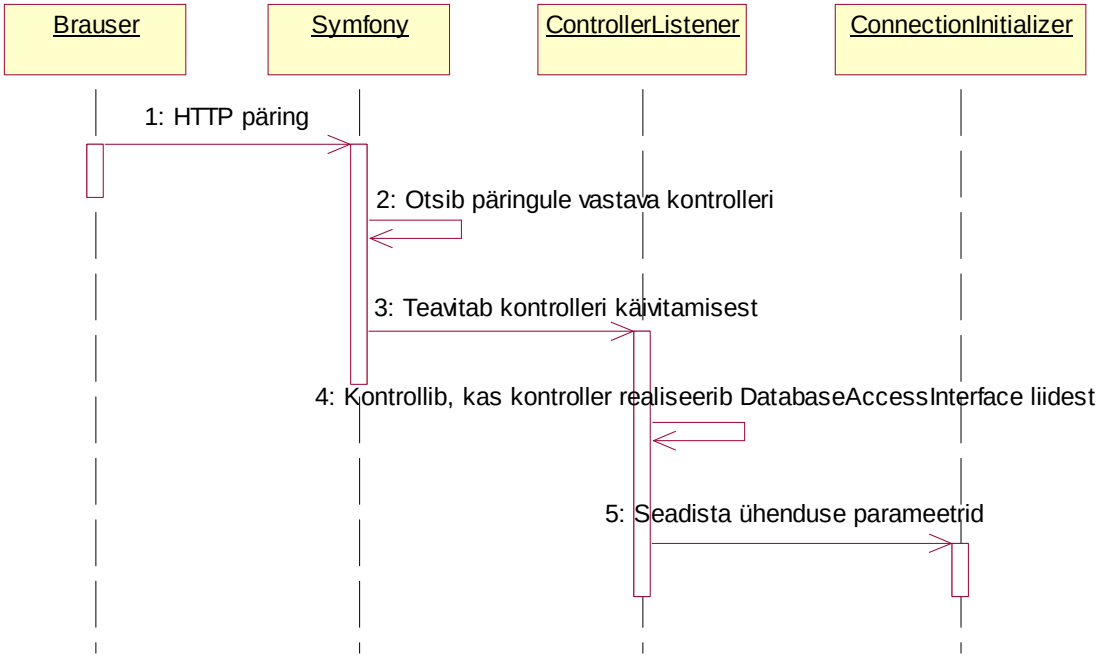
Kontrollerina on eraldi loodud ka *RedirectingController*, mille ainus eesmärk on URL-ist eemaldada viimane kaldkriips ja päring suunata saadud URL-ile. See on vajalik sellepärast, et

kasutajad saaksid lehekülgi külastada olenemata sellest, kas nad kirjutavad URL-i lõppu kaldkriipsu või mitte.

5.2.2.1 Andmebaasiühenduse seadistamine kontrollerite jaoks

Osad mainitud kontrolleritest realiseerivad liidest *DatabaseAccessInterface*. Tegemist on tühja liidesega ehk see ei sisalda mitte ühtegi meetodit ega välja. Antud liides on vajalik, et eristada kontrollereid, mis vajad enda tööks andmebaasiühendust. Kuna andmebaasiga ühendutakse sisse logitud kasutaja kasutajanime ja parooliga, mitte konfiguratsioonifailis kindlaksmääratud kasutajana, siis peab kuskil andmebaasiühenduse loomiseks vajalikud kasutajanime ja parooli ära määrama.

Selleks on olemas kontrolleri kuulaja objekt *ControllerListener*. Kuulaja objektis on defineeritud funktsioon, mida Symfony raamistik kutsub alati enne iga kontrolleri käivitamist. Antud funktsioonis kontrollitakse, kas realiseerib *DatabaseAccessInterface* liidest. Kui see seda teeb ja kasutaja on sisse loginud, siis võetakse kasutaja andmed ning määratakse andmebaasiga ühendumise kasutajanimeks sisse logitud kasutaja kasutajanimi ning parooliks tema poolt sisestatud parool. Antud lahendus tagab, et andmebaasiühenduse parameetrid on enne kontrolleri funktsiooni väljakutset alati seadistatud ning ei pea hakkama mitmes kohas andmebaasiga ühendumiseks vajalikku kasutajanime ning parooli määrama. Järgneval joonisel on ära toodud andmebaasiühenduse seadistamise jadadiagramm.



Joonis 31: Andmebaasi ühenduse seadistamise jadadiagramm DatabaseAccessInterface liidest realiseeriva kontrolleri puhul

5.2.3 Autentimine

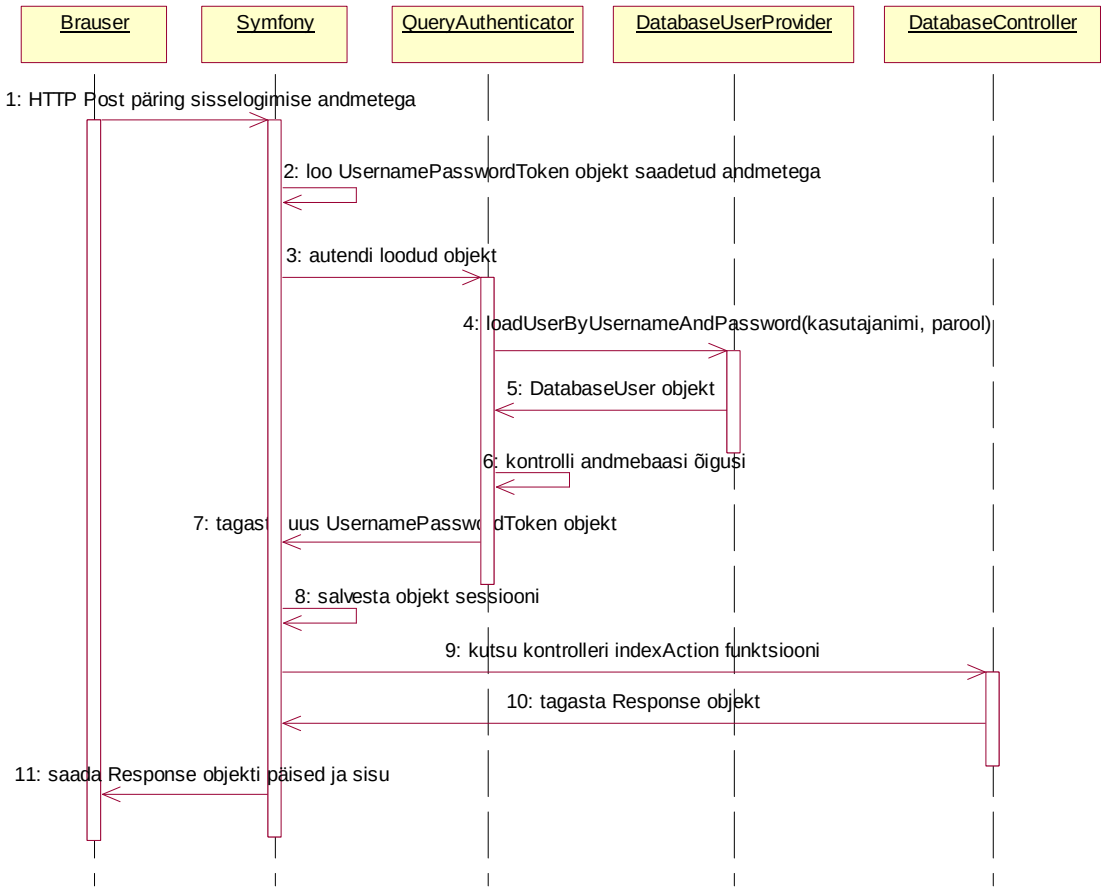
Kasutaja saab rakendusse siseneda kasutades rakenduse konfiguratsioonis määratud andmebaasiserveril asuva mingi PostgreSQL andmebaasi kasutaja kasutajanime ja parooli. Kasutajaks registreerimist antud rakenduses ei toimu.

Kasutaja autentimine on teostatud Symfony turvalisuse komponendi (*security component*) abil, mis toetab ka sisselogimisvormi abil kasutaja autentimist. Konfiguratsioonifailis on ära määratud millistele lehekülgedele saavad kasutajad anonüümselt ligi ning milliste jaoks peab ta olema autenditud. Ainukesena pääseb anonüümselt ligi sisselogimise leheküljele.

Kui kasutaja on sisestanud enda kasutajanime ja parooli ja need serverile saatnud, siis Symfony loob nende andmete põhjal žetooni ehk *token*'i. Seejärel kutsub Symfony välja autentimisega tegelevalt objektilt meetodit, mille eesmärk on tagastada žetoon koos kasutaja objektiga või erind ehk *exception*, kui kasutajat ei õnnestunud autentida. Autentimiseks üritatakse kõigepealt kasutaja poolt sisestatud kasutajanime ja parooliga ühenduda rakenduse andmebaasiga. Seejärel kontrollitakse, kas kasutajal on olemas kõik vajalikud õigused rakenduse andmebaasiga töötamiseks. Sinna hulka kuulub näiteks kõikidest tabelitest

lugemisõigus ning teatud tabelite puhul ka andmete sisestamise õigus. Kui kõik vajalikud õigused on olemas, siis loetakse kasutaja edukalt autendituks. Loodud žetoon kasutajainfoga salvestatakse sessiooni ajaks. Selle žetooni abil saab Symfony teada, kas kasutaja on juba autenditud või mitte.

Kui kasutajat ei õnnestunud autentida, siis kuvatakse uuesti sisselogimisvorm veateatega. Sisselogimisvormi kuvamise ülesanne on *LoginController* kontrolleril.



Joonis 32: Eduka autentimise jadadiagramm

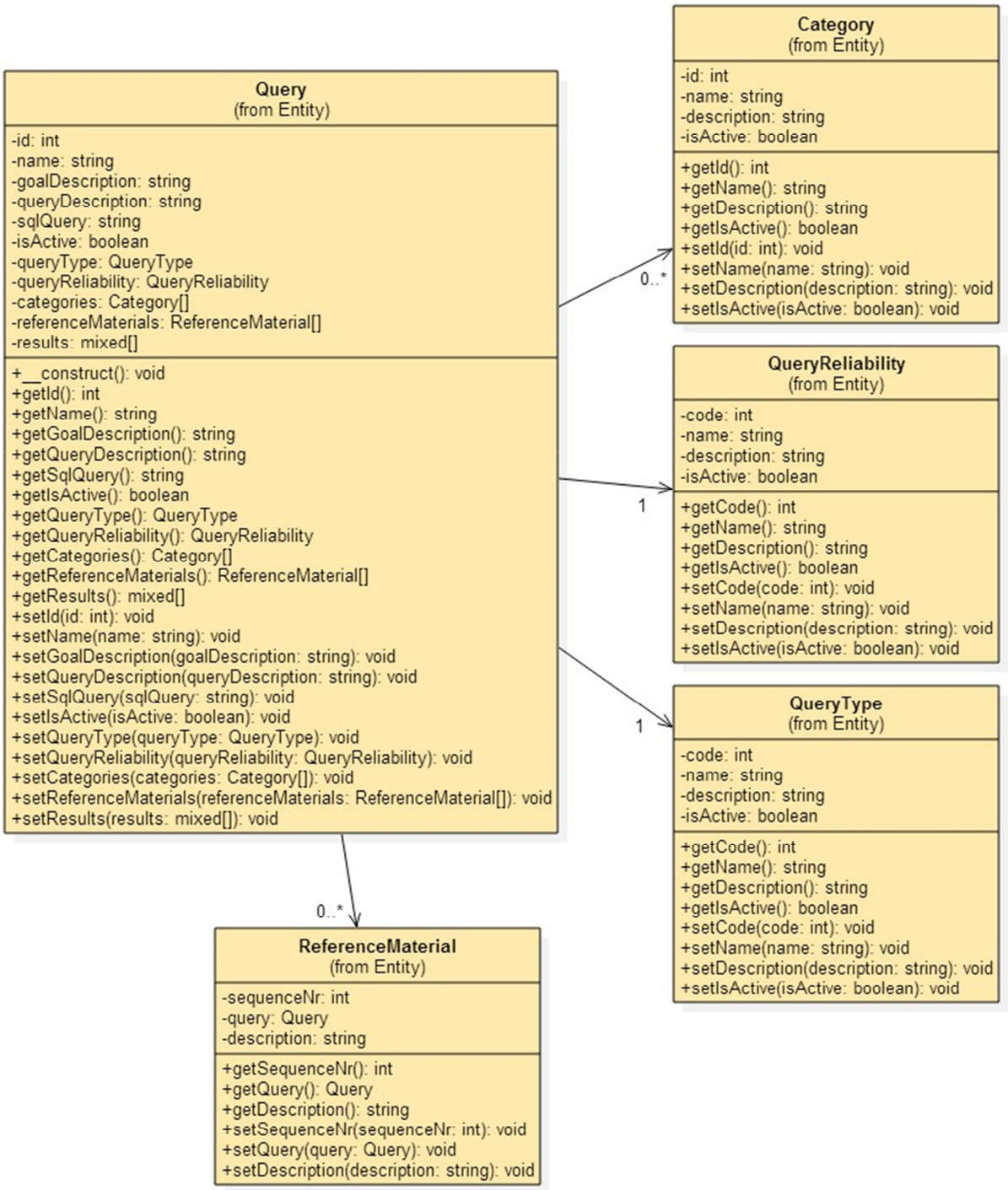
5.2.4 Rakenduse andmebaasiga suhtlemine

Rakenduse andmebaasiga suheldakse läbi Doctrine ORM-i. ORM ehk *object relational mapper* on tarkvara, mis seab vastavusse andmebaasis hoitavad andmed ning rakenduses kasutavad objektid ning hoolitseb andmete püsivuse eest [20]. Selle tarkvara kasutamise eeliseks on, et programmeerija saab keskenduda objektorienteeritud äriloogika objektidele

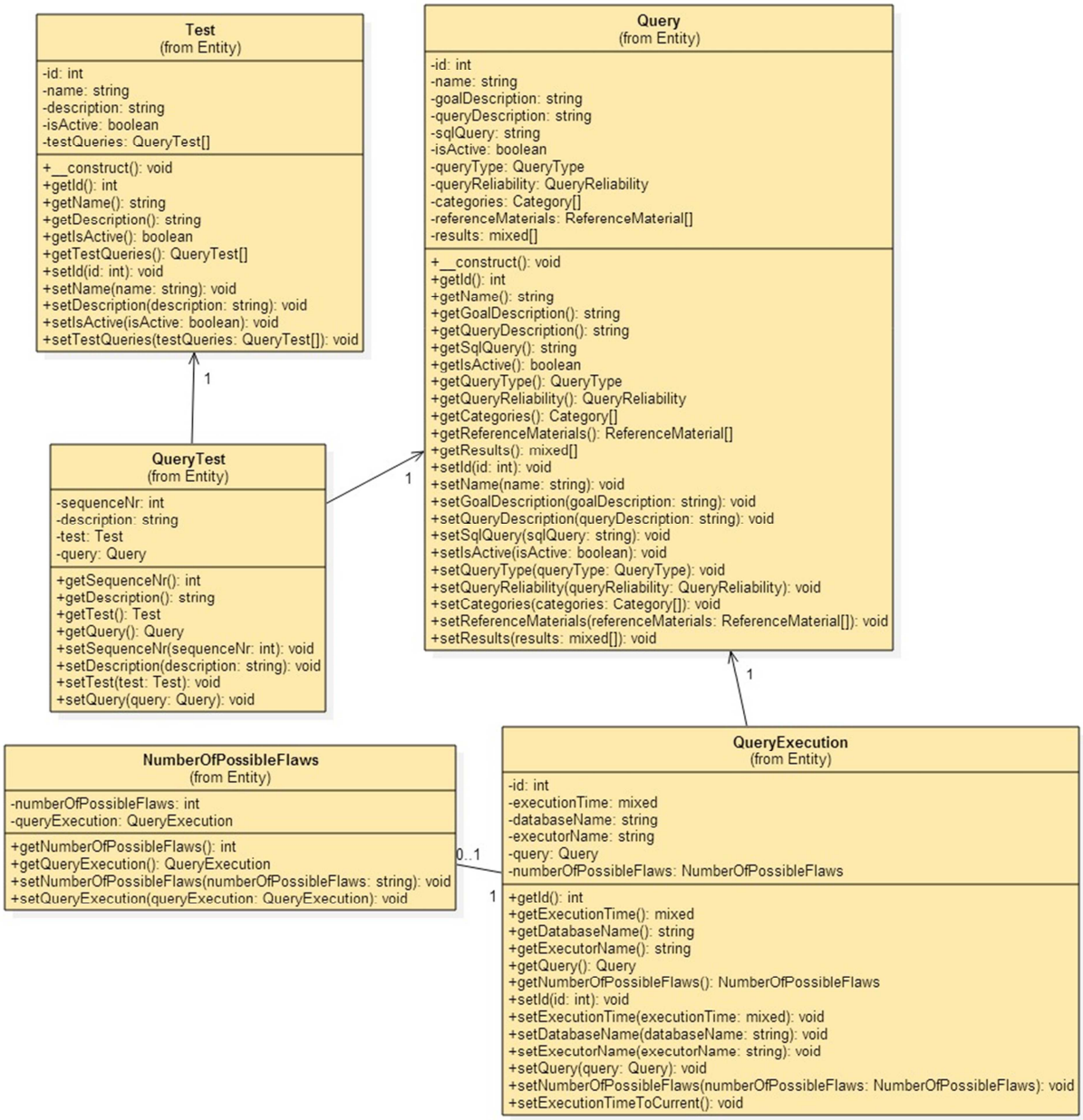
ning jätta püsivus ehk *persistence* suures osas Doctrine'i hoolde [20]. Doctrine'i eelis on ka see, et ta toetab mitmeid erinevaid andmebaasisüsteeme. Näiteks on toetatud PostgreSQL, MySQ, Oracle ja SQLite andmebaasid [21].

5.2.4.1 Doctrine ja objektide seosed andmebaasi tabelitega

Doctrine'i abil saab tavalisi PHP objekte seostada andmebaasi tabelitega. Neid kutsutakse olemiteks. Olemid on PHP objektid, mis on identifitseeritavad unikaalse identifikaatori või primaarvõtme järgi ning nad sisaldavad välju, mida salvestatakse ja päritakse andmebaasist [20]. Kõigepealt loodi rakenduse tarbeks äriloogika mudelid ehk olemid, mis põhinevadki andmebaasis hoitavatel andmetel.



Joonis 33: PHP äriloogika olemite diagramm nr 1



Joonis 34: PHP äriloogika olemite diagramm nr 2

PHP olemite ja nende väljade ning andmebaasi tabelite ja nende veergude vastavus määratakse ära eraldi XML konfiguratsioonifailides. Iga olemi kohta on eraldi fail. Järgnev on näide *Test* olemi konfiguratsioonifailist.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<doctrine-mapping
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns='http://doctrine-project.org/schemas/orm/doctrine-mapping'
    xsi:schemaLocation='http://doctrine-project.org/schemas/orm/doctrine-mapping
        https://raw.githubusercontent.com/doctrine/doctrine2/master/doctrine-mapping.xsd'>

    <entity
        name="t112810\QueriesBundle\Entity\Test" table="test"
        repository-class="t112810\QueriesBundle\Entity\Repository\TestRepository">
        <id name="id" type="integer" column="test_id">
            <generator strategy="AUTO" />
        </id>
        <field name="name" type="string" column="name" />
        <field name="description" type="string" column="description" />
        <field name="isActive" type="boolean" column="is_active" />
        <one-to-many field="testQueries" target-entity="QueryTest" mapped-by="test">
            <join-column name="test_id" referenced-column-name="test_id" />
        </one-to-many>
    </entity>
</doctrine-mapping>
```

Joonis 35: Andmebaasi ja PHP objekti vastavuse määramise näide XML failis

XML formaadis vastavuse määramine on ainult üks võimalikke viise Doctrine'i puhul. Lisaks on toetatud YAML formaadis, PHP kommentaarides annotatsioonidega ning eraldi PHP failidega defineeritavad vastavused. Antud juhul valiti XML formaat, kuna siis on programmi kood eraldatud andmebaasi vastavuse konfiguratsioonidest. Kui soovida teha muudatusi vastavustes, siis piisab ainult konfiguratsioonifaili muutmisest ning ei pea hakkama PHP failides muudatusi tegema.

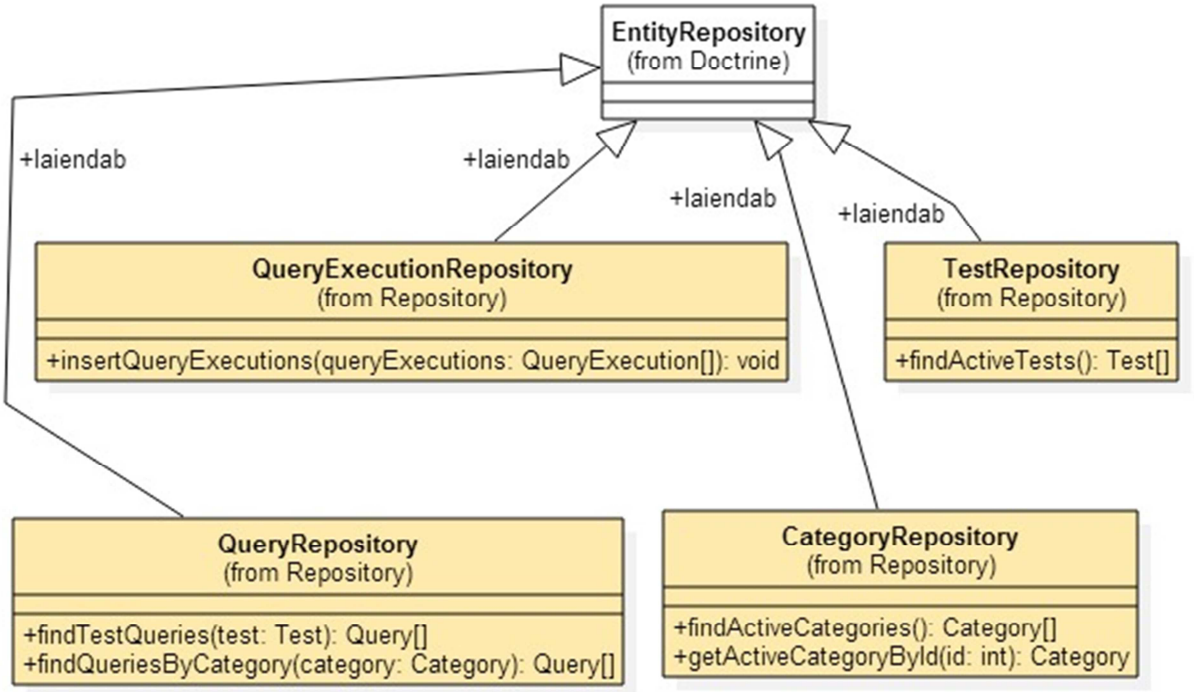
See võimaldab lihtsalt vahetada ka rakenduse andmebaasisüsteemi. PHP olemid jäävad täpselt samaks, kuid konfiguratsioonifailides saab ära määrata teised vastavused, mis on spetsiifilised mingi muu andmebaasisüsteemi andmebaasi struktuurile. Nii saab ära defineerida mitme andmebaasisüsteemi jaoks omad konfiguratsioonifailid. Seega saab antud rakenduse panna kergesti kasutama näiteks Oracle andmebaasi, ilma programmikoodi muutmata. Symfony konfiguratsioonifailis *config.yml* on võimalik täpsustada kausta, kust Doctrine'i vastavuste faile otsitakse.

```yaml
mappings:
    t112810QueriesBundle:
        mapping:            true
        type:               xml
        dir:                Resources/config/doctrine
```

Joonis 36: Doctrine'i konfiguratsiooni osa config.yml failis

5.2.4.2 Objektide hoidlad

Andmete pärimine andmebaasist ning samuti salvestamine andmebaasi käib läbi hoidla ehk *repository*.



Joonis 37: PHP hoidla objektid

Hoidla üheks ülesandeks on teatud tüüpi olemite tagastamine. Olulisemate olemite jaoks on olemas eraldi hoidla objekt, läbi mille päritakse andmebaasist selle olemiga seotuid andmeid. Nendes hoidlaobjektides on meetodid, mis tagastavad teatud tingimustele vastavad olemid koos nendega seotud olemitega. Kõik need hoidlad laiendavad Doctrine'i *EntityRepository* klassi. See klass sisaldab üldist baasfunktsionaalsust, näiteks olemi leidmist primaarvõtme järgi ja kõikide antud tüüpi olemite leidmist. Symfony raamistiku abil saab Doctrine olemeid pärida ka üldisest hoidlast, ilma spetsiifilisi hoidlaobjekte loomata. Hoidlad kasutavad andmebaasiga suhtlemiseks Doctrine'i objekti nimega olemihaldur ehk *entity manager*.

Päringuid on koostatud Doctrine'i päringuehitaja objekti abil. Sellel objektil on mitmeid meetodid päringute programmi abil koostamiseks. Päringute koostamine põhineb PHP objektidel ning selle väljadel, mitte andmebaasi tabelite ja selle veergude nimedel. Doctrine teisendab koostatud päringu andmebaasile sobivaks SQL päringuks ning tagastatud tulemuste põhjal loob uued PHP objektid vastavalt defineeritud konfiguratsioonile.

```
$qb = $this->getEntityManager()->createQueryBuilder();
$qb->select('Category')
    ->from('t112810QueriesBundle:Category', 'Category')
    ->where('Category.isActive = true')
    ->addOrderBy('Category.name');
$categories = $qb->getQuery()->getResult();

return $categories;
```

Joonis 38: Objektide pärimine kasutades Doctrine ORM-i

Kasutajapoolsed laiendused antud hoidla baasklassile lisavad juurde täiendavat funktsionaalsust. Keerulisemad päringud ja andmete salvestamine on paigutatud eraldi hoidlaobjektidesse. See muudab defineeritud keerulisemad päringud korduvkasutatavateks ning hoiab rakenduse andmebaasiga suhtlemise loogika ühes kohas. Lisaks on juurde lisatud näiteks andmebaasi *QueryExecution* olemi salvestamine. Kuna selle olemiga on seotud ka *NumberOfPossibleFlaws* olem, siis salvestab Doctrine *QueryExecution* olemi salvestamisel ka temaga seotud *NumberOfPossibleFlaws* olemi.

Doctrine toetab ka niinimetatud laiska laadimist ehk päring tehakse andmebaasi alles siis, kui üritatakse kasutada olemi mingit välja ja see ei ole see hetk veel väärtustatud. Samas kui on teada, et olemi juures kasutatakse ka alati temaga seotuid olemeid, siis saab need esialgse päringuga koos andmebaasist võtta. See väldib mitmete asjatute päringute tegemist andmebaasi. Nii on toimitud näiteks *QueryRepositiry* olemite funktsioonide puhul, mis tagastavad päringud koos kõigi seotud olemitega. See on vajalik selleks, sest päringute puhul kasutatakse alati ka näiteks tema tüüpi ning usaldusväärsust.

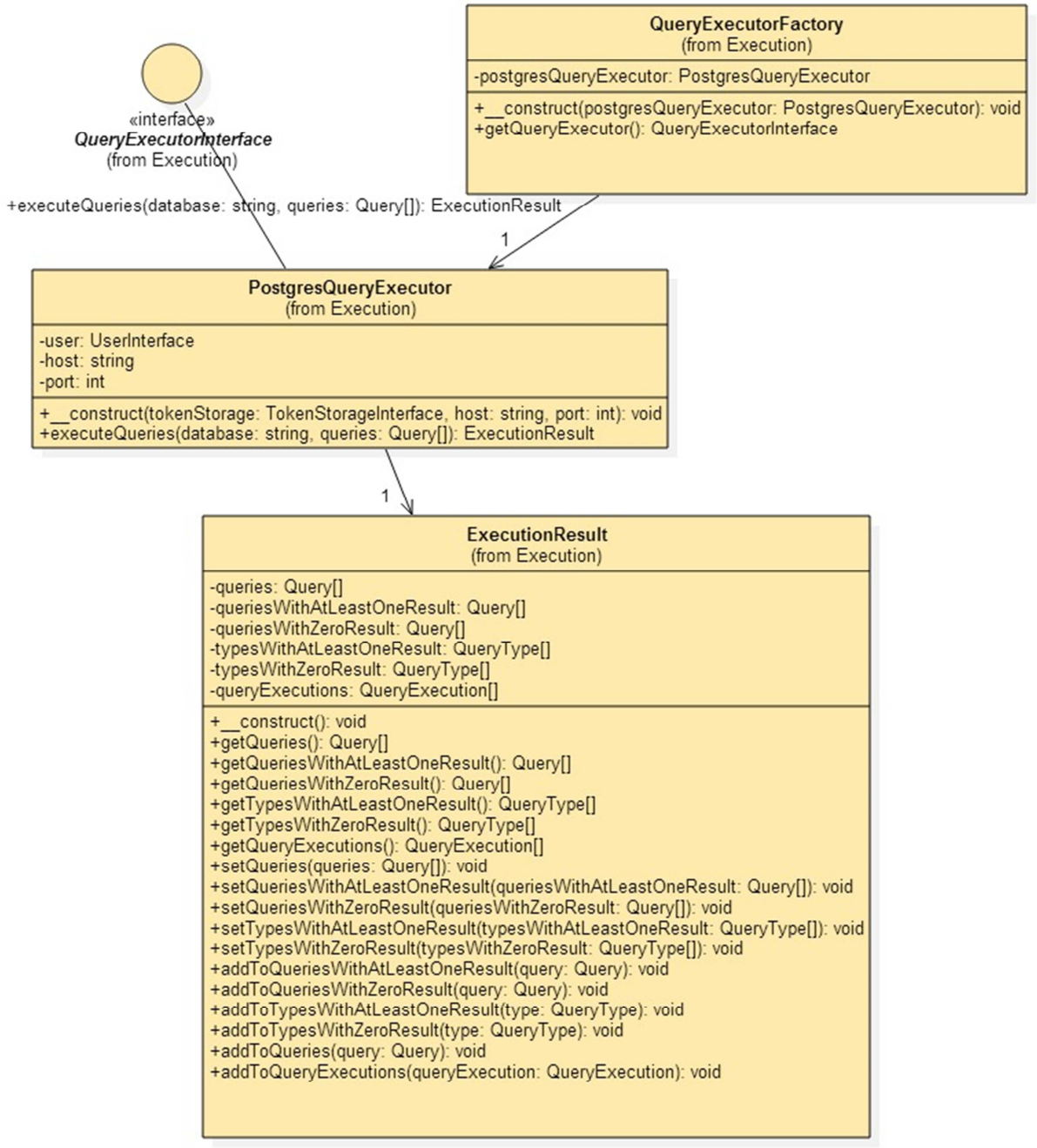
5.2.5 Päringute käivitaja

Kasutaja poolt valitud päringute või tema valitud testis sisalduvate päringute käivitamine tema valitud andmebaasi peal toimub läbi päringu käivitaja objekti, mis realiseerib *QueryExecutorInterface* liidest. Antud liidesel on ainult üks funktsioon *ExecuteQueries*, mis võtab sisendiks päringute objektide massiivi ning tagastab päringu tulemuste objekti. Päringu tulemuste hoidmiseks on loodud eraldi objekt *ExecutionResult*. Antud objekt sisaldab mitmeid välju, mis lihtsustavad päringute tulemuste kasutajale kuvamist.

Päringu käivitaja ehk objekt, mis implementeerib *QueryExecutorInterface* liidest, saadakse läbi *QueryExecutorFactory* objekti. *QueryExecutorFactory* on niiöelda tehase tüüpi objekt, mis tagastab vastavalt teatud tingimustele kindlat tüüpi eelpoolmainitud liidest realiseeriva

objekti. Selline lahendus võimaldab tulevikus luua teiste andmebaasisüsteemide spetsiifilisi päringu käivitajaid. Piisab vaid ühe meetodi realiseerimisest ning seejärel tuleb muuta ka *QueryExecutorFactory* päringu käivitaja tagastamise loogikat. Kuna hetkel on olemas ainult PostgreSQL andmebaasisüsteemidele mõeldud päringu käivitaja, siis käivitaja valimise loogika puudub ning tehaseobjekt tagastab alati ühe ja sama realisatsiooni liidesest, mis on mõeldud spetsiaalselt PostgreSQL jaoks.

Antud päringu käivitajas ehk *PostgresQueryExecutor* objektis kasutatakse Doctrine'i päringukeele asemel PHP PostgreSQL andmebaasiga ühendumise spetsiifilisi funktsioone. See on sellepärast, et päringuid hoitakse teksti kujul andmebaasis ning üks päring võib koosneda mitmetest keerukatest päringutest ja käskudest, mis on näiteks semikooloniga eraldatud. Doctrine'i abil saab korraga aga käivitada ühe SQL lause. Sellepärast kasutataksegi PHP andmebaasiga ühendumise funktsioone, et vältida teksti kujul oleva päringu parsimist alampäringuteks.



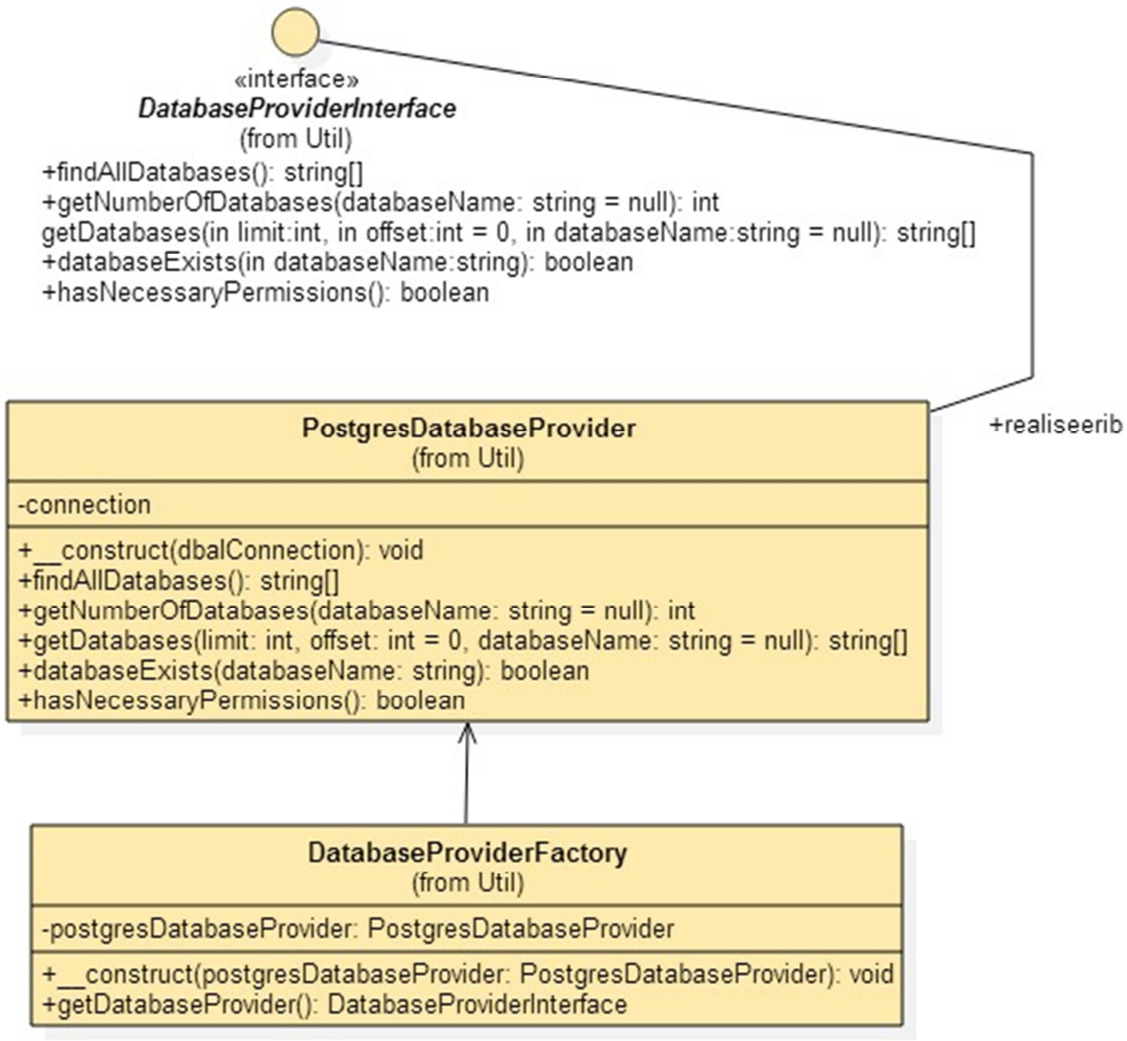
Joonis 39: Päringu käivitaja ja temaga seotud objektid

Objekt *tokenStorage*, mis on *PostgresQueryExecutor* konstruktori parameeter, on Symfony raamistiku *TokenStorageInterface* liidest realiseeriv objekt. Antud objekti kaudu saab kätte *UserInterface* liidest realiseeriva objekti, mis salvestatakse *user* välja. Sellel objektil on olemas meetodid sisse logitud kasutaja kasutajanime ja parooli saamiseks, mida kasutatakse andmebaasiühenduse loomiseks. Lisaks antakse konstruktorisse kaasa ka andmebaasi host ja pordi number.

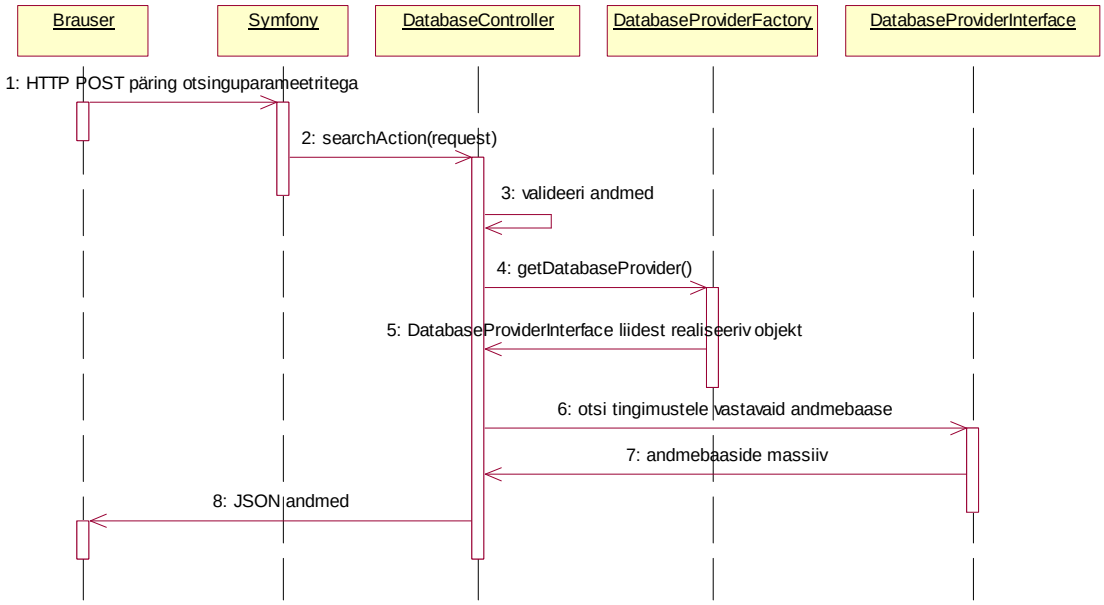
5.2.6 Andmebaasi otsimine

Nagu eelnevalt on mainitud, saab kasutaja valida teda huvitava andmebaasi kõikide andmebaaside hulgast, mis andmebaasiserveril asuvad. Andmebaasiserver määratakse kindlaks rakenduse konfiguratsioonifailis. Serveril asuvate andmebaaside otsimiseks kasutatakse PostgreSQL süsteemikataloogis asuvat *pg_database* kataloogi.

Andmebaaside otsimine on realiseeritud AJAX tehnoloogia abil. JavaScriptiga tehakse serverile päring ning server tagastab JSON formaadis andmed. Andmebaaside otsimise jaoks on loodud eraldi objekt *PostgresDatabaseProvider*, mis realiseerib liidest *DatabaseProviderInterface*. Kui tulevikus tekib vajadus rakenduse andmebaas mingi muu andmebaasisssteemi (näiteks Oracle) peale üle viia, siis piisab ainult uue klassi loomisest, mis seda liidest realiseerib. Andmebaaside otsimisega tegelevat objekti küsitakse tehase tüüpi objekti käest, mis tagastab teatud tingimustele vastava realisatsiooni *DatabaseProviderInterface* liidesest. Kuna hetkel on loodud ainult PostgreSQL jaoks andmebaaside otsija, siis tagastatakse alati see objekt. Kui tulevikus lisandub muude andmebaasisüsteemide tugi, siis tuleb ainult loogika lisada, mis teeks kindlaks, millist andmebaasisüsteemi kasutatakse ning tagastaks õige objekti.



Joonis 40: Andmebaaside otsimisega tegelevate objektide diagramm



Joonis 41: Andmebaasi otsimise jadadiagramm

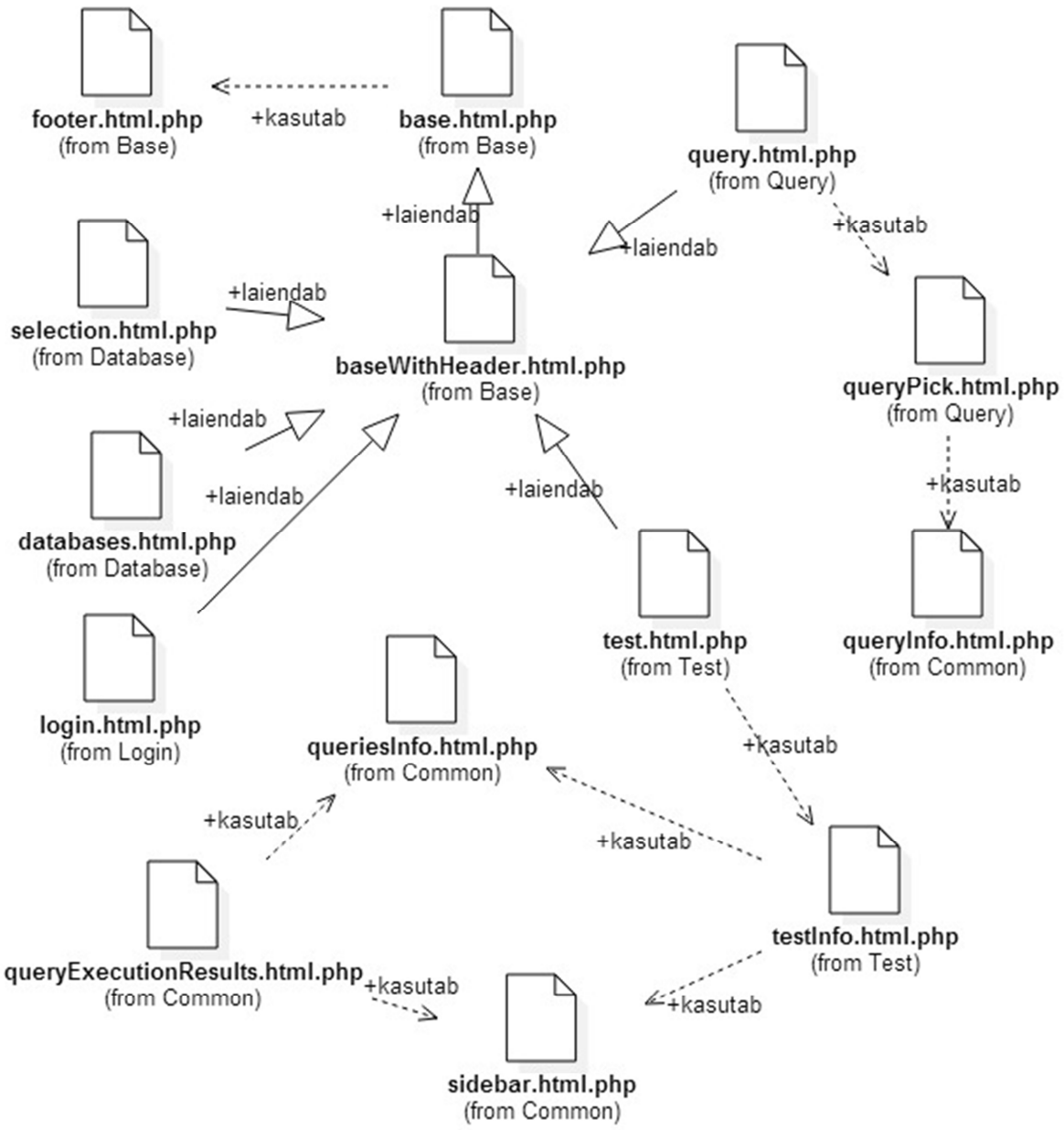
Nagu jooniselt näha, siis Symfony suunab saabunud HTTP päringu täitmise konkreetsele kontrollerile. Päringu parameetriteks on otsitava andmebaasi nimi ja startpositsioon, mis näitab mitmendast tulemusest alates andmebaase tagastada. Korraga tagastatakse kuni 15 andmebaasi. Näiteks, kui start on 20, siis tagastatakse leitud andmebaaside massiivi hulgast need, mis on positsioonil 21-35. Brauserile saadetavad JSON formaadis andmed sisaldavad leitud andmebaaside nimesid, kogu andmebaaside arvu antud serveris ning infot selle kohta, kas antud otsingutingimustele vastavaid andmebaase leidub veel.

5.2.7 Mallid

Kliendile ehk antud juhul veebibrauserile tagastatav HTML moodustatakse mallidest. Mall on tekstifail, mis genereerib väljundit, milleks on ükskõik milline tekstipõhine formaat (HTML, XML jne) [19]. Käesoleval juhul on tegemist PHP mallidega – tekstifailidega, mida parsitakse PHP poolt ning see sisaldab PHP koodi ning teksti ning PHP parsimise tulemusena tekib HTML formaadis tekst. Vaade ehk HTML lehekülg, mida klient läbi veebibrauseri näeb, on kokku pandud mitmetest mallidest. Kuna mitmel kasutajale kuvataval leheküljel on korduvad elemendid, näiteks päringute tulemuste kuvamise osa, siis saab need ära kirjeldada ühes mallis ning lisada neid vajadusel mitmesugustele teistele mallidele.

Symfony raamistiku abil saavad mallid nagu PHP klassidki üksteist laiendada. See võimaldab kogu lehekülje üldise paigutuse nagu päise ja jaluse defineerida ühes baasmallis. Selles

baasmallis saab määrata plokid, mida seda malli laiendavad teised mallid saavad ülekirjutada. Nii ongi defineeritud lehekülje üldine paigutus baasmalli (eraldi päisega ja päiseta versioon) ning teised mallid lisavad selle sisuplokki omapoolse sisu.



Joonis 42: Mallid

Mallile andmete edastamine toimub kontrolleris. Selleks on kasutusel Symfony baaskontrolleri funktsioon *render*, mille argumentideks on soovitava malli nimi ning assotsiatiivne massiiv. Massiiv koosneb võtme-väärtus paaridest. Võtmeks on vabalt valitud nimi ja väärtuseks ükskõik milline PHP objekt.

```
return $this->render('t112810QueriesBundle:Query:query.html.php',
    array(
        'database' => $database,
        'navigation' => true,
        'active_menu' => 'queries',
        'databases' => $databases,
        'categories' => $categories,
        'queries' => $queries,
        'types' => $queryTypes,
    )
);
```

Joonis 43: Mallile andmete andmine

Symfony mallimootor (*templating engine*) muudab edastatud andmed mallis lihtsasti kättesaadavaks. Massiivi elemendi väärtuseks olev objekt on kättesaadav läbi võtme nimega muutuja. Näiteks ülaltoodud pildi põhjal saab *query.html.php* mallis kasutada muutujat *$queries*, milleks on päringute objektide massiiv; või muutujat *$active_menu*, mille väärtuseks on joonis 43 põhjal sõna *queries*.

5.2.8 Kasutajaliidese keele valik

Käesoleva töö üheks nõudeks oli, et rakenduse liides peaks olema nii eesti kui ka inglise keeles. Üks põhjus, miks otsustati kasutada Symfony raamistikku oli ka see, et antud raamistikul on hea mitmekeelsuse tugi. Selleks on kasutusel Symfony tõlkekomponent.

Tõlkimiseks kasutatavat sõnastikku hoitakse *.yml* failiformaadis. Iga keele kohta on loodud oma fail. Uue keele lisamiseks on vaja lisatava keele kohta luua ainult uus sõnastikufail.

```
tests:
  title:
    availableTests: 'Saadavalolevad testid'
    testExecution: 'Test käivitab järgmised päringud'
    testResults: 'Tulemused'
  form:
    label:
      test: 'Test'
      database: 'Andmebaas'
    action:
      runSelectedTest: 'Käivita valitud test'
```

Joonis 44: Eesti keelse tõlkesõnastiku faili sisu

Nagu pildilt näha, siis tõlked on grupeeritud erinevate võtmete kaupa. Võtmesõnad on valitud võimalikult informatiivsed ning nad peaksid kirjeldama tõlgitava fraasi eesmärki. Sõnastiku

failides võib võtmeks olla ükskõik milline sõna, kaasa arvatud ka terve tõlgitav fraas originaalkeeles. Selline hierarhiline võtmete struktuur nagu ülevaloleval pildil näha valiti seetõttu, et siis saab tõlgetesse kergesti muutusi teha, ilma malle muutmata. Kui mallides oleks originaalkeeles fraas ja soovitakse tõlget muuta, siis tuleks muudatusi teha nii mallides kui ka sõnastiku enda failides. Hierarhiline võtmete lahendus lihtsustab seega tõlgete haldamist.

Tõlgete kasutamiseks mallides on Symfony jaoks olemas tõlketeenus *trans*, mis on kasutatav läbi globaalse muutuja *$view*, mis on igas mallis saadaval. Antud teenus tagastab argumendiks antud võtme järgi leitud sõnumi soovitud keeles. Näiteks joonisel 45 kujutatud tõlkefunktsiooni väljakutse tagastab eesti keele puhul vastavalt joonisel 44 kujutatud sõnastikule teksti „Saadavalolevad testid“.

```
<?php echo $view['translator']->trans('tests.title.availableTests') ?>
```

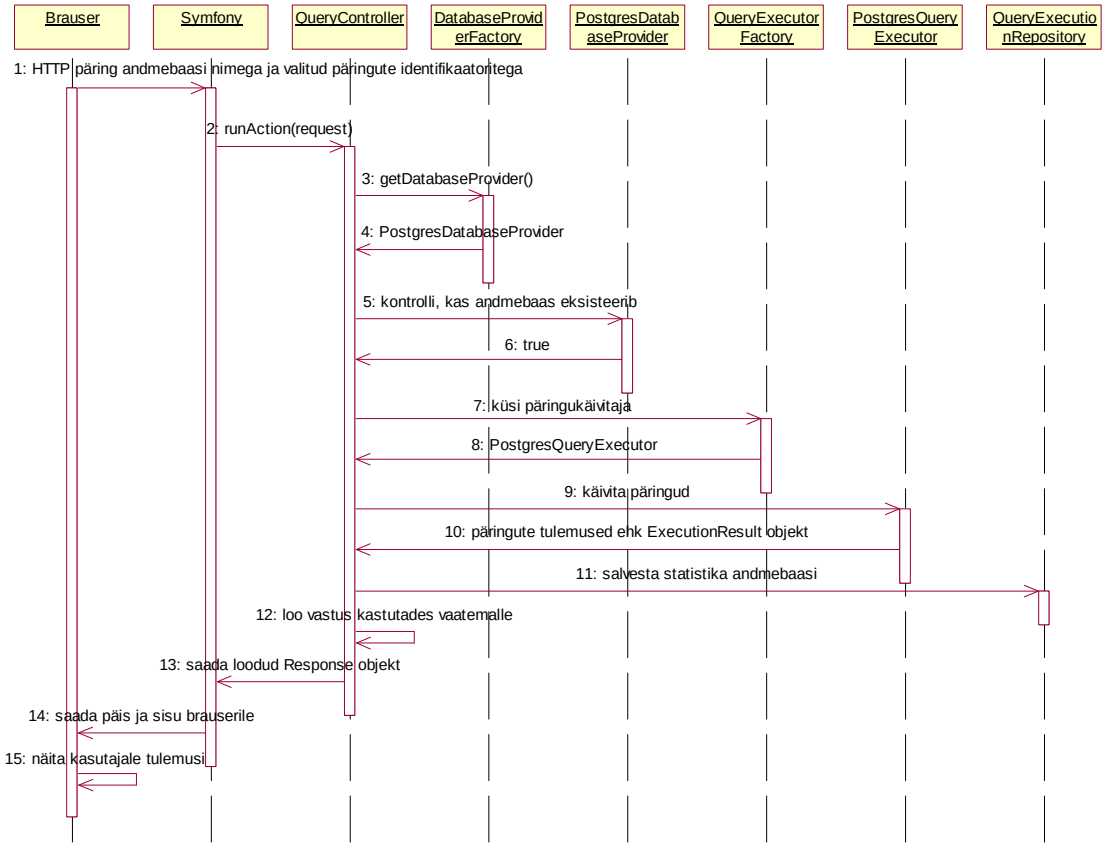
Joonis 45: Tõlgete kasutamine mallides

Soovitud kasutajaliidese keele valik antakse kaasa URL-iga. Tegemist on alati esimese URL-i osaga juuraadressi järel ning kui see puudub, siis pannakse vaikimisi keele väärtuseks inglise keel. Näiteks */et/queries/* puhul on keele valikuks *et* ehk eesti keel. Symfony oskab antud URL-i osa järgi määrata ära, millisest failist tõlkeid loetakse.

Rakenduse konfiguratsioonifailis on ära määratud, milline on põhikeel. Kui üritatakse tõlkida mingit sõnumit võtme järgi näiteks eesti keelde ja seda võtit ei leita eesti keele sõnastikust, siis kasutatakse põhikeele sõnastikku, milleks on antud rakenduses inglise keel. Kui otsitavat sõnumit või sõnumi võtit ei leita sõnastikust, siis tõlkekomponent tagastab sõnastiku võtme.

5.3 Näide komponentide omavahelisest koostööst

Järgnevalt esitatakse näide päringute käivitamisest, mis näitab kuidas eelnevates jaotistes kirjeldatud komponendid omavahel suhtlevad.



Joonis 46: Päringute käivitamise jadadiagramm

Kui mingis etapis tekib viga, siis saadetakse brauserile tagasi HTTP vastus vastava staatusekoodiga. Päringute käivitamine toimub asünkroonselt AJAX tehnoloogia abil. Kui brauser saab tagasi veateate, siis kuvatakse kasutajale sellekohane teade ning soovi korral saab seejärel uuesti päringuid käivitada.

6. Kokkuvõte

Käesoleva töö eesmärgiks oli disainida ning realiseerida andmebaasi disaineri töökoht PostgreSQL andmebaaside disainivigade kontrollimise süsteemile. Loodava lahenduse näol pidi olema tegemist veebipõhise kasutajaliidesega ning veebirakendus pidi olema kirjutatud PHP programmeerimiskeeles. Lisaks pidi kasutajaliidese keel olema nii inglise kui ka eesti keeles.

Tööks seatud eesmärgid täideti. Läbi analüüsi ja prototüübi arendamise koguti kokku piisavalt infot, et arendada valmis kasutuskõlblik veebirakendus päringute ja testide käivitamiseks. Päringute tulemused annavad kasutajale väärtuslikku infot võimalike disainivigade kohta.

Kasutajaliidese disainimisel põhjendati tehtud valikuid ning toodi välja kasutatud kasutajaliidese mustrid. Rakenduse loomiseks kasutati mitmeid tehnoloogiad ning raamistikke, mis muudavad rakenduse arhitektuuri arusaadavamaks ning kergemini laiendatavaks. Rakendusse loodi ka mitmekeelsuse tugi. Loodud lahendus võimaldab tulevikus vajadusel lihtsalt kasutajaliidesesse uusi keeli juurde lisada.

Loodud rakendus toetab hetkel PostgreSQL andmebaase. Edasiarendusena saaks kirjutada juurde vajalikud komponendid ka teiste andmebaasisüsteemide toetamiseks. Tugi selle jaoks on rakenduses loodud. Teiste andmebaasisüsteemide toetamiseks oleks veel vajalik kirjutada spetsiaalselt andmebaasisüsteemi silmas pidades ka päringud, mis tuvastavad võimalikke disainivigu konkreetset andmebaasisüsteemi silmas pidades. Samuti saaks täiustada rakenduse tööks vajaliku andmebaasi struktuuri, muutes võimalikuks hoida samas andmebaasis osasid andmeid tõlgituna mitmesse keelde, näiteks päringute kirjeldust nii inglise kui ka eesti keeles.

Summary

The goal of this thesis was to design and implement a database designer workplace for a system that checks the design of PostgreSQL databases. The solution had to be a web-based user interface and the web application itself had to be written in PHP programming language. Furthermore, the user interface should be available both in Estonian and in English.

The goal that was set for this thesis was completely fulfilled. Through analysis and prototyping enough information was gathered to develop usable web application for executing queries and tests. The results of the queries give users valuable information for detecting possible design flaws in the selected database.

In the designing part of user interface there were explanations given why it was designed that way and patterns described that were used in the designing process. In order to create this application many technologies and frameworks were used which made the architecture of the application more understandable and easily expandable. There support for multiple languages for user interface was also built. Created solution makes it easy to add new languages to the user interface in the future if necessary.

The created application currently supports only PostgreSQL databases. A possible future improvement would be creating necessary components to support other database management systems as well. Support for that is provided in the application. In order to support other database management systems, necessary queries have to be written, which would find possible design flaws in databases while taking into account the specifics of the database management system. In addition, database structure modification is possible in order to make it possible to store translations of certain data in the same database. For example, to store query description both in English and in Estonian.

Kasutatud kirjandus

[1] Heikki Vallaste. Vallaste IT ja sidetehnika seletav sõnaraamat. [WWW] http://www.vallaste.ee (20.05.2015)

[2] Veebi disainimise meetodid. [WWW] http://www.e-uni.ee/e-kursused/eucip/arendus/4410_veebi_disainimise_meetodid.html (20.05.2015)

[3] Prototüüpimine. [WWW] http://www.e-uni.ee/e-kursused/eucip/arendus/1224_prototpimine.html (20.05.2015)

[4] Less eelprotsessor http://lesscss.org/ (20.05.2015)

[5] Päringute käivitamise prototüüp. [WWW] http://apex.ttu.ee/queries/ (20.05.2015)

[6] Testide käivitamise prototüüp. [WWW] http://apex.ttu.ee/queries2/ (20.05.2015)

[7] Erki Eessaar, *Süsteemiarendus ja andmebaasi koht selles. TTÜ Andmebaasid II õppeine materjal*, 2013

[8] Erki Eessaar, "On Query-based Search of Possible Design Flaws of SQL Databases," *Innovation and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering Lecture Notes in Electrical Engineering*, vol. 313, International Conference on Systems, Computing, Sciences and Software Engineering (SCSS 12), lk. 53-60, 2015

[9] Janina Voronova, *PostgreSQL andmebaasi kohta süsteemikataloogi abil info leidmine*, 2009, Bakalaureusetöö, TTÜ Informaatikainstituut

[10] Erki Eessaar. Design flaws queries. [WWW] http://staff.ttu.ee/~eessaar/files/Design_flaws_queries.pdf (20.05.2015)

[11] Erki Eesaar, Janina Voronova, "Using SQL Queries to Evaluate the Design of SQL Databases," *Innovation and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering Lecture Notes in Electrical Engineering*, vol. 312, International Conference on Systems, Computing, Sciences and Software Engineering (SCSS 13), lk. 179-186, 2015

[12] StatCounter brauseri kasutamise statistika. [WWW] http://gs.statcounter.com/#browser-ww-monthly-201502-201504 (20.05.2015)

[13] EAS. Korduma kippuvad küsimused. [WWW] http://www.eas.ee/et/avalikule-ja-mittetulundussektorile/turism/106-ettevotjale/perioodi-2007-2013-stardi-ja-kasvutoetuse-programm-/351-korduma-kippuvad-kuesimused3 (20.05.2015)

[14] Balsamiq Mockups. [WWW] http://webdemo.balsamiq.com/ (20.05.2015)

[15] Introduction to Balsamiq Mockups 3. [WWW] http://support.balsamiq.com/customer/portal/articles/1844131 (20.05.2015)

[16] Jenifer Tedwell, *Designing Interfaces*, 2010

[17] Twitter Bootstrap. [WWW] http://getbootstrap.com/ (20.05.2015)

[18] Fabien Potencier arvamus Symfoni raamistikust. [WWW].

http://fabien.potencier.org/article/49/what-is-symfony2 (20.05.2015)

[19] Symfony the book. [WWW]
http://symfony.com/pdf/Symfony_book_2.6.pdf?v=4 (20.05.2015)

[20] Doctrine. [WWW]
http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/tutorials/getting-started.html (20.05.2015)

[21] Doctrine konfiguratsioon. [WWW]
http://doctrine-dbal.readthedocs.org/en/latest/reference/configuration.html

[22] Postgresql information schema. [WWW]
http://www.postgresql.org/docs/9.4/static/information-schema.html (20.05.2015)

Lisa 1

Logout

Database: **121060_disain1** Select a new database

Please select a test: Databases II (fall 2014) Select

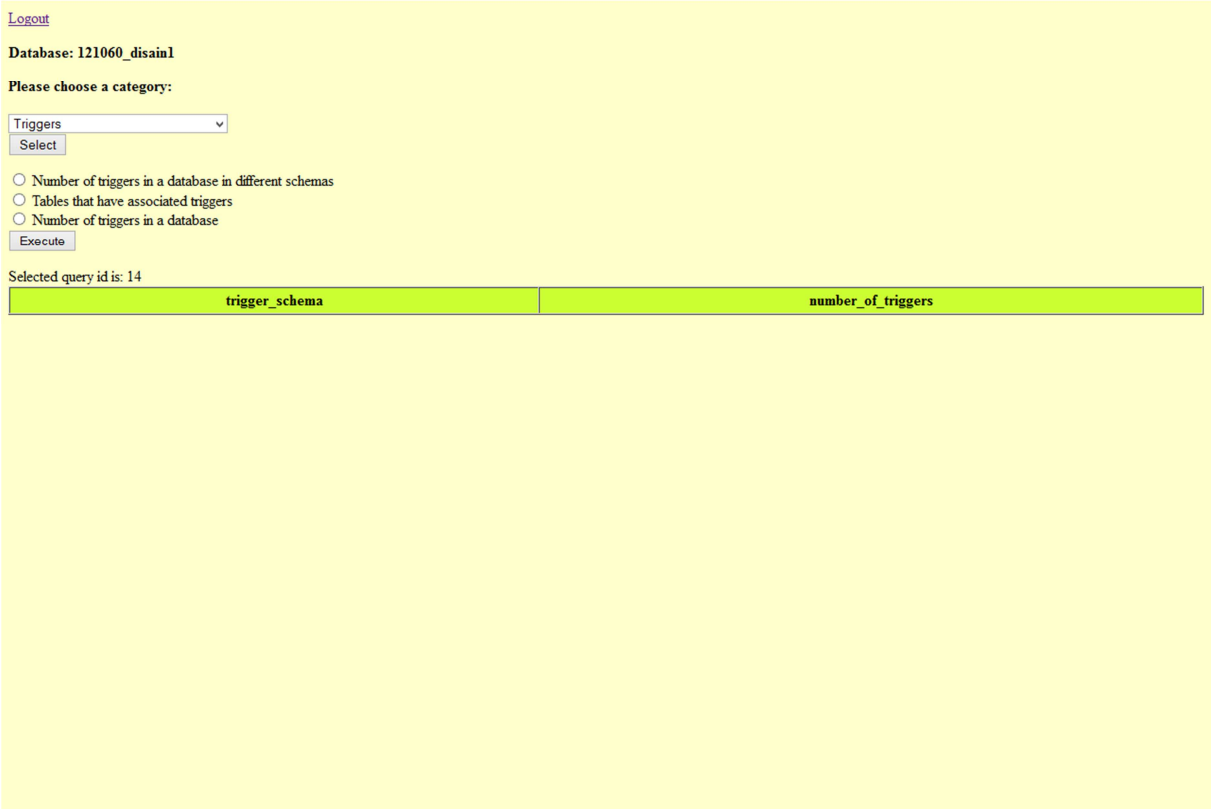
Press **Execute** to execute all the queries in the test.

Execute

Number of tables in a database in different schemas	Identify the number of base tables and views in different schemas.
User-defined views	Find user-defined views. Pay attention to the outer join operations. One should use these if and only if there is real need for them. Otherwise they just reduce performance. On the other hand pay attention that outer join is used where it is logically needed.
View usage	For each base table the list of views that refer to the base table.
User-defined non-trigger routines	What are the non-trigger routines in non-systemic schemas?
Number of rules in a database in different schemas	Rules can be used to maintain data integrity in a database by causing rejection of incorrect insertions and updates. Therefore, the number of rules in a database gives some information about general data integrity.
Number of triggers in a database	Triggers are used to maintain data integrity in a database by causing rejection of incorrect insertions and updates. Therefore, the number of triggers in a database gives some information about general data integrity.
Number of triggers in a database in different schemas	Triggers are used to maintain data integrity in a database by causing rejection of incorrect insertions and updates. Therefore, the number of triggers in a database gives some information about general data integrity.
Tables that have associated triggers	Find information about tables that are associated with triggers.
Base tables that have neither a unique constraint nor a primary key	What are the base tables in a database without any unique constraints and primary keys? In such tables there are no restrictions to record duplicate rows. Each row represents a true proposition. It does not make the proposition more true if one presents it more than once.
Key constraints	Find what are the primary key and unique constraints of base tables and whether there is some overlap.
Overlapping keys	Find completely overlapping primary keys and unique constraints.
Base tables that have surrogate key as the primary key and do not have any unique constraints	A surrogate key is a primary key that consist of one column. The values of this column do not represent any meaning for the user and the system usually generates the values automatically. In case of defining a surrogate key in a table it is common mistake to forget declaring existing natural keys in the table.
Referential degree of a schema	Referential degree is defined as the number of foreign keys in the database schema.
Referential degree of a table	This metric represents the number of foreign keys in a table.
Leave Out the (Referential) Constraints	Database developers do not define referential (foreign key) constraints. For instance, they could reason that it helps them to reduce performance problems in case of some update operations. However, in this case the enforcement of referential integrity becomes a task of applications. The query operates as follows: Find base tables that do not participate in any referential constraint (as the referenced table or as the referencing table).
Constraint naming	To find out how consistent are the constraint names and whether users have given themselves the names.

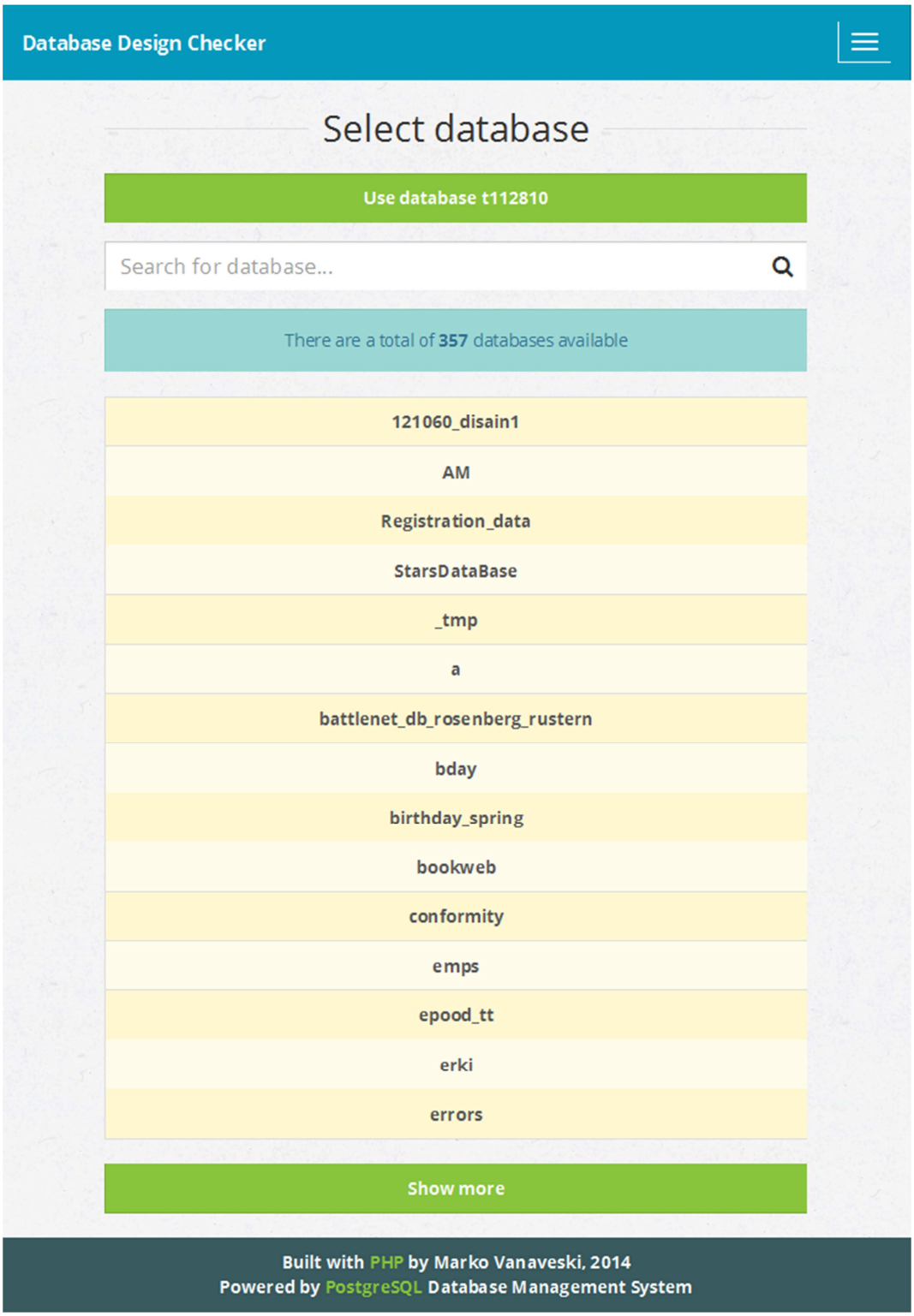
Joonis 47: Testide käivitamise vana prototüüp

Lisa 2



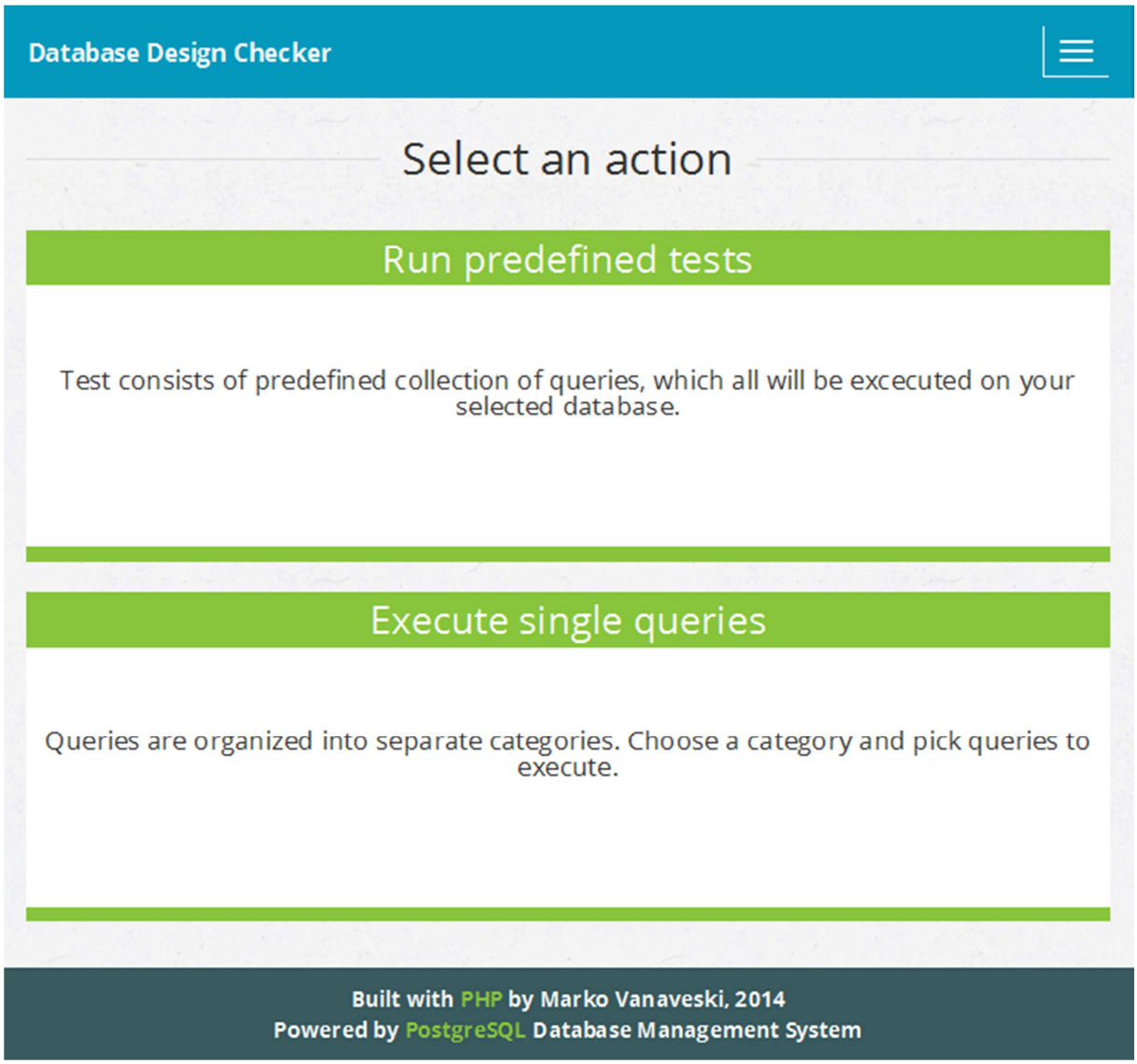
Joonis 48: Päringute käivitamise vana prototüüp

Lisa 3



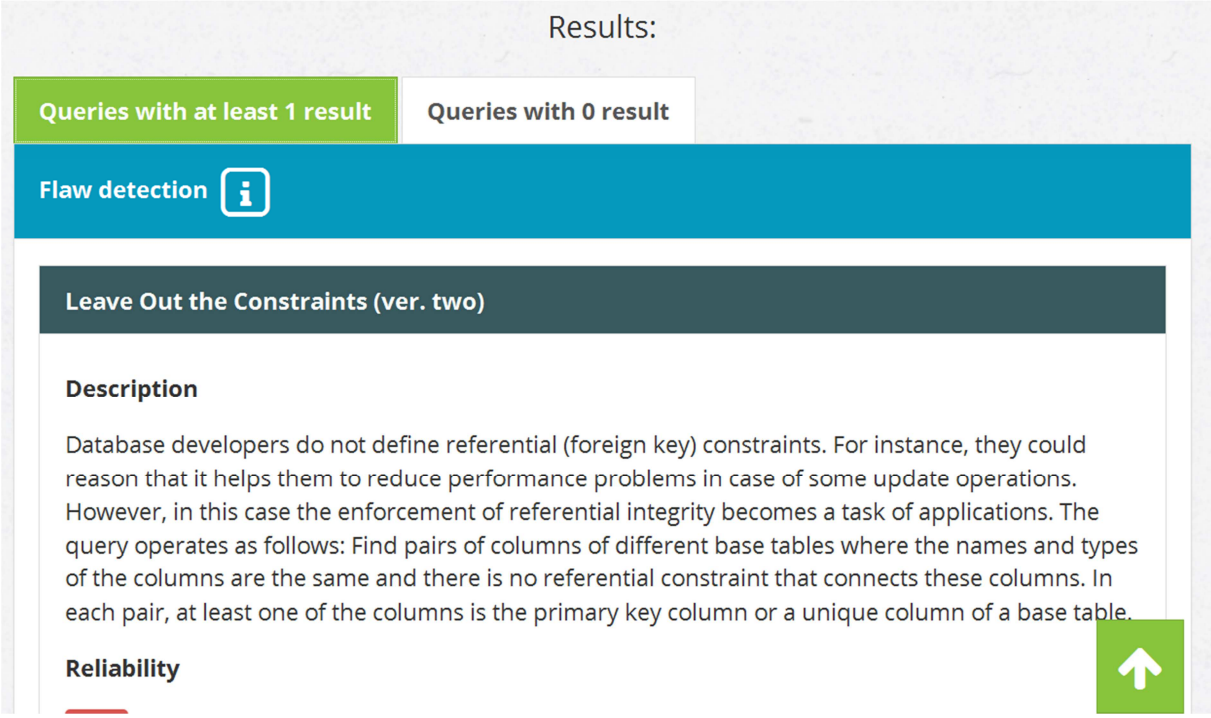
Joonis 49: Andmebaaside valimise mobiilivaade

Lisa 4



Joonis 50: Tegevuse valimise mobiilivaade

Lisa 5



Joonis 51: Päringute kuvamise mobiilivaade