KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Ernestas Burokas

PHP PROGRAMAVIMO KALBOS KARKASŲ CHARAKTERISTIKŲ TYRIMAS

Baigiamasis magistro projektas

Vadovas
Dr. Dominykas Barisas

KAUNAS, 2016

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

PHP PROGRAMAVIMO KALBOS KARKASŲ CHARAKTERISTIKŲ TYRIMAS

Baigiamasis magistro projektas
Programų sistemų inžinerija (kodas M4046M21)

Vadovas
Dr. Dominykas Barisas

Recenzentas
Doc. Dr. Jonas Čeponis

Projektą atliko
Ernestas Burokas

KAUNAS, 2016

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

(Fakultetas)

ERNESTAS BUROKAS

(Studento vardas, pavardė)

PROGRAMŲ SISTEMŲ INŽINERIJA (M4046M21)

(Studijų programos pavadinimas, kodas)

„PHP programavimo kalbos karkasų charakteristikų tyrimas“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

20 16 m. Gegužės 17 d.

Kaunas

Patvirtinu, kad mano, **Ernesto Buroko**, baigiamasis projektas tema „PHP programavimo kalbos karkasų charakteristikų tyrimas“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka) *(parašas)*

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

Burokas, Ernestas. Comparison Of PHP Frameworks Characteristics *Master's* / supervisor doc. Dominykas Barisas. The Faculty of informatics, Kaunas University of Technology.

Research area and field: Software Engineering

Key words: PHP frameworks, Zend2, Yii, Laravel, Symfony2, CodeIgniter, CakePHP.

Kaunas, 2016. 61 p.

SUMMARY

An increasing number of websites are created using a wide variety of frameworks that facilitate the programming work and provide various functions (code and menu generation, querying the database, validation etc.). Manual implementation of these functions is time consuming. However, for developers, which intend to create website based on PHP framework or just want to test a new framework, it is hard to pick right framework because of wide selection.

Moreover, it is not always easy to find technical characteristics of PHP frameworks. This tool is created to solve that issue. It is simple website for PHP frameworks comparison and it helps to do this research. With this program user can select which frameworks and characteristics will be compared between each other. Such comparison might help user to decide which framework is the most appropriate for him. Biggest advantage of this program is that user do not have to install all frameworks to test them, it can be done using PHP frameworks comparison tool.

During the research four characteristics of frameworks comparison were selected. First one is speed, this metric determines how quickly page is loading using one of the frameworks. Second is number of class loaded, this characteristics shows how many classes or libraries are used in framework. Third comparison metric is number of code lines, which is needed to perform particular operation. Last characteristic is cyclomatic complexity, this metric determines all execution paths of the code. During the course 6 frameworks were measured: Zend2, Symfony2, CakePHP, Laravel, CodeIgniter, Yii. In order to make more detailed research website programmed with plain PHP (no framework used), was included into comparison.

1. ĮVADAS

1.1. Dokumento paskirtis

Šis dokumentas yra skirtas aprašyti sukurtos sistemos architektūrą bei atlikti jos analizę. Taip pat dokumente yra atliekama teorinė technologijų analizė susijusi su visais *PHP* programavimo kalbos karkasais. Eksperimento dalyje yra pateikti atlikto tyrimo rezultatai bei jų analizė.

1.2. Santrauka

Burokas, Ernestas. *PHP* programavimo kalbos karkasų charakteristikų tyrimas. Magistro baigiamasis projektas / vadovas dr. Dominykas Barisas; Kauno technologijos universitetas, informatikos fakultetas.

Mokslo kryptis ir sritis: Programų sistemų inžinerija

Reikšminiai žodžiai: *php programvimo kalbos karkasai, Zend2, Yii, Laravel, Symfony2, CodeIgniter, CakePHP*.

Kaunas, 2016. 61 p.

Vis daugiau ir daugiau interneto tinklalapių yra kuriama naudojant įvairiausius karkasus, kurie palengvina programavimo darbus ir suteikia įvairiausių funkcijų (kodo ir meniu generavimas, apsauga, užklausų į duomenų bazę sukūrimo įrankis ir pan.), kurias programuoti pačiam užimtų daug laiko. Tačiau programuotojui, kuris ketina kurti tinklalapį panaudodamas naują karkasą arba tiesiog nori išbandyti tinklalapio kūrimą panaudojant karkasą, yra gana sunku išsirinkti, kurį karkasą naudoti, kadangi jų yra labai daug.

Taip pat ne visada lengva rasti karkasų technines charakteristikas. Šiam tikslui ir yra kuriamas šis įrankis, kuris yra realizuotas kaip interneto svetainė, skirta *PHP* programavimo kalbos karkasų tyrimui atlikti. Šios programos pagalba vartotojas galės pasirinkti, kuriuos esamus karkasus norėtų palyginti bei pagal kokius parametrus. Toks palyginimas galėtų padėti vartotojui apsispręsti, kuris karkasas jam yra tinkamiausias. Naudojant šią programą vartotojui nereikėtų pačiam išbandyti visus esamus karkasus, o tiesiog pasinaudojus įrankiu įvertinti įvairiausius parametrus.

Atliekant tyrimą buvo pasirinktos 4 charakteristikos *PHP* programavimo kalbos karkasų palyginimui. Tai – greitis, kuris nusako, kaip greitai yra užkraunamas puslapis naudojantis tam tikru karkasu. Užkrautų klasių kiekis, kuris nusako naudojamų klasių skaičių kiekviename karkase. Kodo eilučių kiekis, reikalingas atlikti tam tikrai funkcijai. Ciklomatinis sudėtingumas, nusakantis galimus įvykdymo kelius funkcijoje. Tyrimo metu buvo nagrinėjami 6 *PHP* programavimo kalbos karkasai: *Zend2, Symfony2, CakePHP, Laravel, CodeIgniter, Yii*. Siekiant atlikti detalesnį tyrimą į palyginimą buvo įtraukta ir programa, kuri nenaudoja jokio karkaso.

2. ANALITINĖ DALIS

2.1. *PHP* programavimo kalba

2.1.1. Įvadas

Šiame magistratūros darbe, yra nagrinėjami *PHP* programavimo kalbos karkasai ir jų charakteristikos. Ši kalba buvo sukurta paversti interneto puslapių turinį dinaminiu, kadangi prieš tai svetainių turinys buvo statinis. Iki šių dienų tai yra pagrindinis šios kalbos tikslas, ir ji yra populiariausia tarp kalbų dinaminiam turiniui kurti. *PHP* programavimo kalba turi labai daug įrankių, su ja praktiškai galima padaryti viską. Ji yra labai populiari norint sugeneruoti *XML* tipo dokumentus, *flash* animacijas, *PDF* formato failus ir dar daugiau. Ši programavimo kalba gali paleisti įvairius scenarijus naudojant komandinę eilutę. Ši funkcija yra naudojama norint padaryti sistemos atsarginę kopiją (angl. *back up*) ar įvairių log'ų nuskaitymą. *PHP* programavimo kalba veikia įvairiausiose platformose, tokiose kaip: *Linux*, *FreeBSD*, *Ubuntu*, Debian, *Solaris*, *Mac OS*, *Windows*. Taip pat gali būti vykdoma daugelyje žiniatinklio serverių: *Apache*, *Microsoft IIS*, *Netscape/iPlanet*. Maža to, ši kalba turi integruotas funkcijas *PDF*, *GIF*, *JPEG*, *PNG* failų generavimui ir *flash* filmukų generavimui. Vienas iš didžiausių *PHP* programavimo kalbos pliusų yra didelis duomenų bazių palaikymas, ši kalba palaiko tokias duomenų bazes: *MySQL*, *PostgreSQL*, *Oracle*, *Sybase*, *MS-SQL*, *DB2*, *ODBC*, *SQLite*, *MongoDB* [1, 1-2 p.].

2.1.2. Istorija

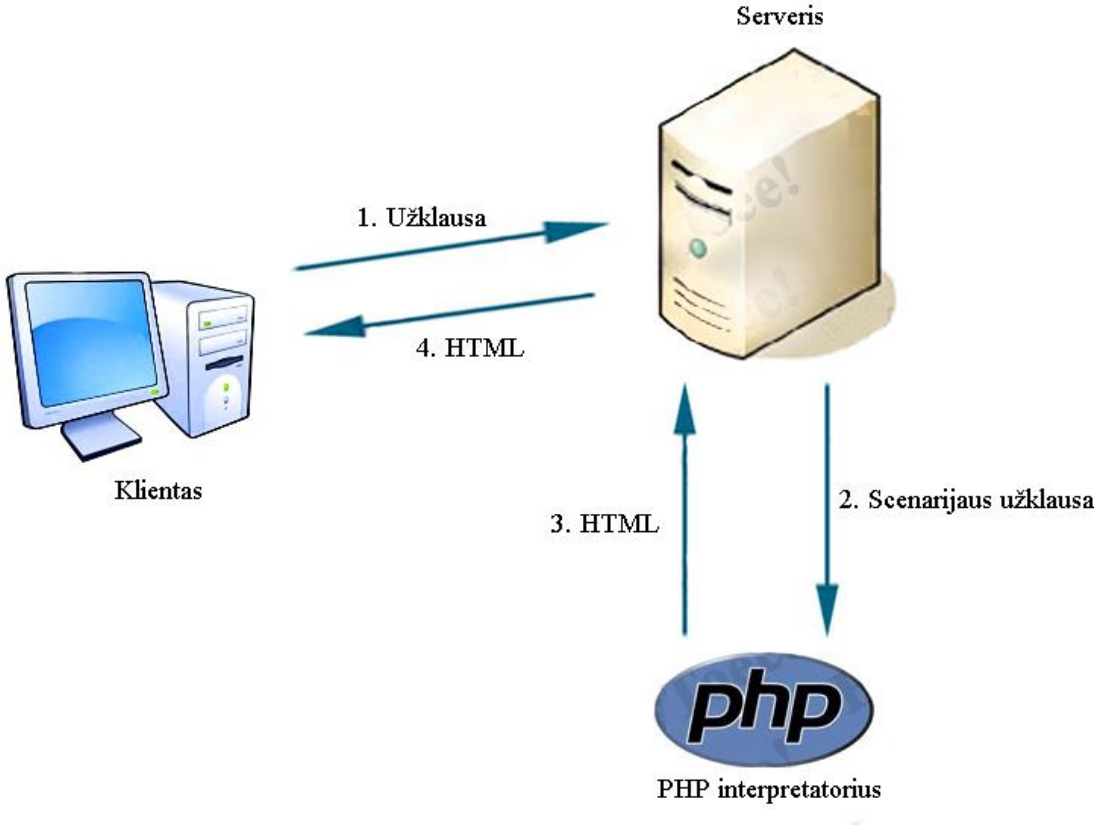
PHP programavimo kalbą sukūrė *Rasmus Lerdorf* 1994 metais. Dabartinė naudojama *PHP* programavimo kalbos versija gerokai skyrėsi nuo pirmosios. Pirmoji šios kalbos versija buvo sukurta net neplanuojant kurti scenarijų kalbos, pagrindinė logika buvo parašyta *C* kalba. Pagrindinė problema, su kuria susidūrė *Rasmus Lerdorf* norėdamas sukurti įrankį Universitetui, kuris gauna duomenis iš skirtingų šaltinių, buvo *PHP* programavimo kalbos integravimas į serverį. Norint tai padaryti reikėjo nulaužti serverį ir taip integruoti kalbą, taigi norint naudotis šia kalba reikėdavo naudotis nulaužtu serveriu. Vėliau šią problemą ėmėsi spręsti *Apache* ir palengvino *PHP* programavimo kalbos integravimą į serverį [1, 2-4 p.].

1996 metais buvo išleista antroji kalbos versija, kuri jau buvo traktuojama kaip scenarijaus kalba. Pirmosios versijos supaprastintas žymų pakeitimo kodas buvo pakeistas analizatoriumi, kuris palyginus su pirmąja versija galėjo apdoroti sudėtingesnes žymas. Pagrindinė antros versijos kūrimo priežastis buvo kelių žmonių, kurie naudojo pirmąją *PHP* programavimo kalbos versiją, susidomėjimas *C* kalba grįstų karkasų panaudojimas kurti įvairius papildinius. Taigi šie žmonės nuolatos reikalaudavo galimybės siųsti *HTML* blokus su įvairiausiomis sąlygomis. Taip buvo sukurta *if* sąlyga. Kai jau buvo ši sąlyga, automatiškai atsirado *else* ir tai paskatino sukurti scenarijų kalbą. Įpusėjus 1997 metams buvo nemažai šios kalbos vartotojų, tačiau PHP programavimo kalbos antra versija nebuvo tobula, joje pastebėta pagrindinio analizatoriaus stabilumo problemų ir tai tebuvo vieno žmogaus projektas [1, 4-5 p.].

Nuo trečiosios versijos, kuri buvo išleista 1998 metais, šis projektas tapo atviro kodo programavimo kalba, kadangi prie jo prisijungė *Zeev Suraski* ir *Andi Gutmans*, kurie pasisiūlė perrašyti pagrindinį analizatorių. Vėliau kurti kitas dalis prisidėjo ir daugiau savanorių, taip *PHP* programavimo kalba tapo atviro kodo. Po šios versijos kalba sparčiai išpopuliarėjo visame pasaulyje. Kuriant kitą versiją jau buvo atlikti tik fundamentalūs *PHP* programavimo kalbos architektūros pakeitimai, tokie kaip: atskiras sluoksnis tarp kalbos ir žiniatinklio serverio, pridėtas gijų saugumo mechanizmas, taip pat patobulinta žymų analizatoriaus sistema. Po daugelio kūrėjų pastangų ketvirtoji *PHP* programavimo kalbos versija buvo išleista 2000 metais. Penktoji šios kalbos versija pasižymi stabilumu, tapo objektiškai orientuota. Taip pat integruota *XML* ir *SQLite* [1, 5-6 p.].

2.1.3. Veikimo principas

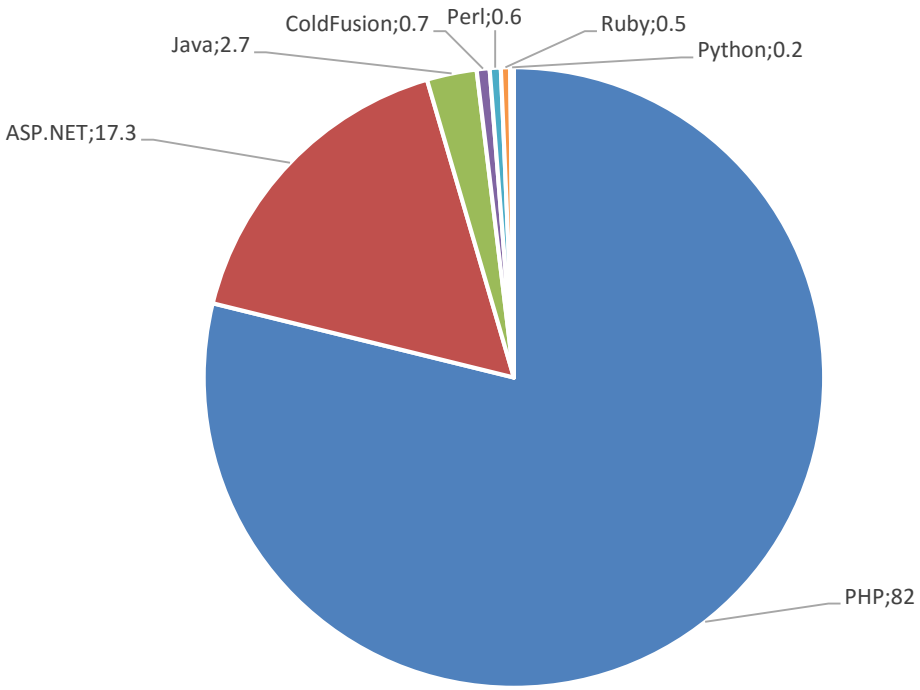
PHP programavimo kalbos veikimo principas yra parodytas toliau esančiame paveikslėlyje (žr. 2.1.3.1 pav.). Viskas prasideda, kai vartotojas atsidaro interneto puslapį naudodamas bet kurią naršyklę. Kai svetainės adresas yra suvedamas į naršyklę, paspaudus ,,užkrauti puslapį" iš karto siunčiama užklausa į serverį (1 žingsnis), kuriame yra patalpinta interneto svetainė. Serveris savo ruožtu pateikia užklausą *PHP* interpretatoriui pradėti vykdyti pageidaujamus veiksmus (2 žingsnis). Interpretatorius vykdo veiksmus ir grąžina *HTML* kodą serveriui (3 žingsnis), o šis į kliento kompiuterį (4 žingsnis), atlikus šiuos veiksmus puslapis yra užkraunamas vartotojo kompiuteryje. Norint atlikti šiuos veiksmus reikia laiko, laikas priklauso nuo to, kokiu atstumu nuo jūsų yra serveris. Šis principas skiriasi nuo statinių puslapių principo, kuris buvo naudojamas anksčiau. Ankstesniu principu serveris grąžindavo *HTML* duomenis iš karto, kai gaudavo vartotojo užklausą. Toks principas tikdavo statiniams puslapiams, kaip pavyzdžiui: paprastam tekstui, nuotraukoms vaizduoti. Atsiradus poreikiui registruoti vartotojus sistemoje ar parašyti komentarus po straipsniu atsirado dinaminiai puslapiai [2].



2.1.3.1 pav. *PHP* programavimo kalbos veikimo principas

2.1.4. PHP programavimo kalbos populiarumas

Karkasų analizei atlikti buvo pasirinkta *PHP* programavimo kalba, pasirinkimą didžiąja dalimi lėmė milžiniškas šios kalbos populiarumas. Jos populiarumas lyginant su kitomis serverio pusės programavimo kalbomis yra pateiktas toliau esančiame paveikslėlyje (žr. 2.1.4.1 pav.). Remiantis http://w3techs.com/ svetainės duomenimis *PHP* programavimo kalbą 2014 metų lapkričio mėnesį naudojo 82 procentai interneto svetainių, kas parodo visišką dominavimą. Antra pagal populiarumą liko *ASP.Net* programavimo kalba su 17,3 procento vartotojų. Trečioje vietoje buvo *Java*, kurią naudojo 2.7 procento interneto puslapių. Likusios serverio pusės programavimo kalbas net nesiekia 1 procento panaudojimo [3].



2.1.4.1 pav. Serverio pusės programavimo kalbų populiarumas

Lyginant *PHP* programavimo kalbą su jos artimiausiu konkurentu *ASP.NET*, *PHP* programavimo kalba yra greitesnė, taip pat yra atviro kodo, o tai reiškia, jog prieinamas šios programavimo kalbos kodas. Dar vienas svarbus šios kalbos pliusas yra tai, jog ji visiškai nemokama bei palaiko labai daug duomenų bazių valdymo sistemų. Šią kalbą labai lengva išmokti, tačiau lyginant kalbą su *ASP.NET* galima įžvelgti ir pastarosios kalbos pranašumų. ASP.NET yra saugesnė programavimo kalba, taip pat turi derinimo įrankį, šia kalba paprasčiau kurti įvairiausius automatinius testus [5].

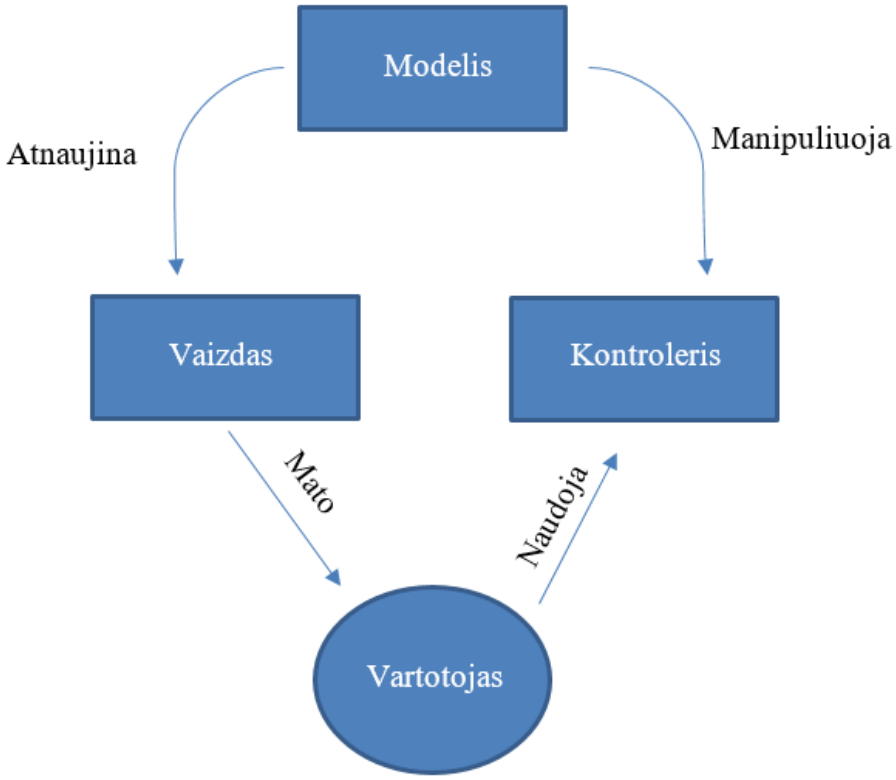
2.2. *MVC* šablono architektūra

Bet kurios programavimo kalbos karkasas tai yra bibliotekų rinkinys, skirtas padidinti programuojamo kodo panaudojimą daug kartų, taip didinant sistemų kūrimo laiką ir mažinant išlaidas. PHP programavimo kalbos karkasai savyje turi kodą, skirtą pagrindinėms funkcijoms atlikti, tokioms kaip: prieiga prie duomenų bazės, sesijos valdymas, apsauga ir panašiai. Šios funkcijos bei jų gausa kiekviename karkase gali skirtis. Šios funkcijos padeda sumažinti programavimo ir testavimo laiką, kadangi nereikia iš naujo programuoti jau esamo funkcionalumo ir jį testuoti. Dauguma karkasų taip pat palaiko tvarkingą aplankų struktūrą, siekiant atskirti karkaso dalis taip palengvinant kodo taisymą programuotojui, kadangi paprasčiau rasti failą, kurį reikia pataisyti [20].

MVC šablonas gerai atskiria vartotojo sąsajos, biznio logikos ir duomenų modelio komponentus, nustatydamas ryšius tarp jų. Šiame šablone kiekvieno komponento atsakomybės yra gerai apibrėžtos, tai lemia geresnį kodo suskaidymą į modulius [20].

Beveik visi populiariausi *PHP* programavimo kalbos karkasai naudoja *MVC* modelį, tad skyrelyje bus nagrinėjamas būtent šis modelis. *MVC* modelio užuomazgos buvo dar 1970 metais, ir tai yra modelis, atskiriantis duomenis, skirtus atvaizdavimui nuo metodų, kurie bendrauja su duomenimis. Iš esmės gerai suprojektuotas MVC projektas turi leisti dirbti svetainės išorės (angl. *front-end*) ir vidaus (*back-end*) programuotojams prie to paties projekto netrukdant vienas kitam, tai reiškia, nepriklausomai vienam nuo kito [4].

Šį modelį sudaro trys pagrindinės dalys: modelis, vaizdas, kontroleris. *MVC* modelio schema yra pavaizduota toliau esančiame paveikslėlyje (žr. 2.2.1 pav.). Jame pavaizduota, kaip duomenys keliauja tarp komponentų ir kokie ryšiai sieja komponentus. Pirmiausia vartotojas siunčia užklausą kontroleriui, kontroleris analizuoja užklausą ir iškviečia modelį, kuris atlieka reikalingą logiką ir jungimąsi su duomenų baze. Atlikęs veiksmus modelis grąžina rezultatus kontroleriui, kontroleris persiunčia rezultatą vaizdui, užklausa yra baigiama, kai rezultatas pasiekia vartotoją. Apžvelkime *MVC* modelio sudedamąsias dalis plačiau [4].



2.2.1 pav. *MVC* modelio schema

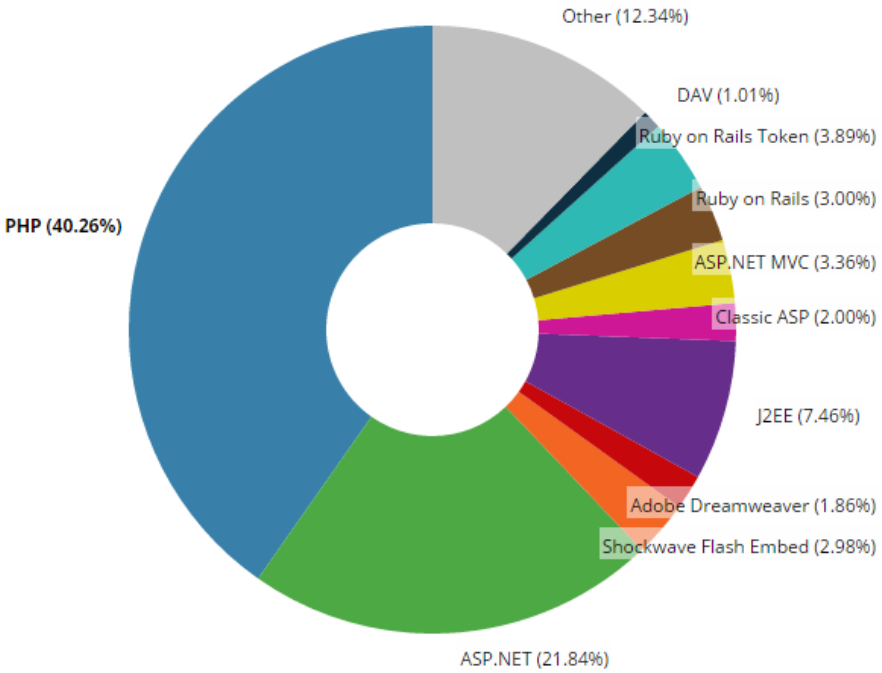
Šiame šablone modelis yra tarsi jungiamoji dalis tarp kontrolerio ir vaizdo. Vienas iš esminių aspektų yra tas, jog modelis iš esmės „aklas“, jis nežino, kas atsitinka su duomenimis, kai juos perduoda kitam komponentui. Jis negauna jokio atsakymo apie tai. Vienintelis modelio tikslas yra paruošti duomenis ir perduoti kitam komponentui. Tačiau modelio negalime laikyti kaip duomenų bazės, nes tai skirtingi dalykai. Modelis veikia kaip duomenų apsaugininkas, kuris priima visas jam skirtas užklausas ir apdoroja jas. Dažniausiai ši šablono dalis yra sudėtingiausia lyginant su kitomis [4].

Vaizdas yra galutinis duomenų, gautų iš modelio, pavaizdavimas naršyklėje. Tradiciškai vaizdas interneto puslapiuose, kurie naudoja *MVC* šabloną, yra dalis *HTML* kodo. Pavyzdžiui, mygtukas puslapyje, kuris yra sugeneruotas vaizdo, paspaudus mygtuką vartotojas iškviečia kontrolerio įvykį. Dažniausia daroma programuotojų klaida, kuri pažeidžia šį šabloną, yra ta, kad vaizdas neturi jokios jungties su modeliu. Visi duomenys vaizdui yra perduodami iš kontrolerio. Tai prieštarauja šablono teorijai [4].

Paskutinis *MVC* šablono komponentas yra kontroleris. Pagrindinis jo darbas yra apdoroti duomenis, kuriuos pateikė vartotojas, ir atitinkamai atnaujinti modelį. Tai yra vienintelė šablono dalis, kuri bendrauja su vartotoju, be kurio veiksmų kontroleris neturėtų prasmės egzistuoti. Kontroleris gali būti traktuojamas kaip duomenų surinkėjas, kuris surinktus duomenis perduoda modeliui, kur jie laikomi. Ši dalis turėtų turėti tik logiką, skirtą informacijos surinkimui. Taip pat kontroleris yra sujungtas tik su vienu vaizdu ir vienu modeliu, kad užtikrintų vienos krypties duomenų srauto sistemą [4].

2.3. *PHP* programavimo kalbos karkasai

Remiantis karkasų populiarumo statistika (žr. 2.3.1 pav.), akivaizdžiai matosi, jog populiariausi yra PHP programavimo kalbos karkasai, kurie sudaro net 40 procentų visų interneto puslapiuose naudojamų karkasų, antri pagal populiarumą yra *.NET* programavimo kalbos karkasai. Tai puikiai atspindi, jog kuriant interneto puslapius didžiausia konkurencija yra būtent tarp šių programavimo kalbų [20].



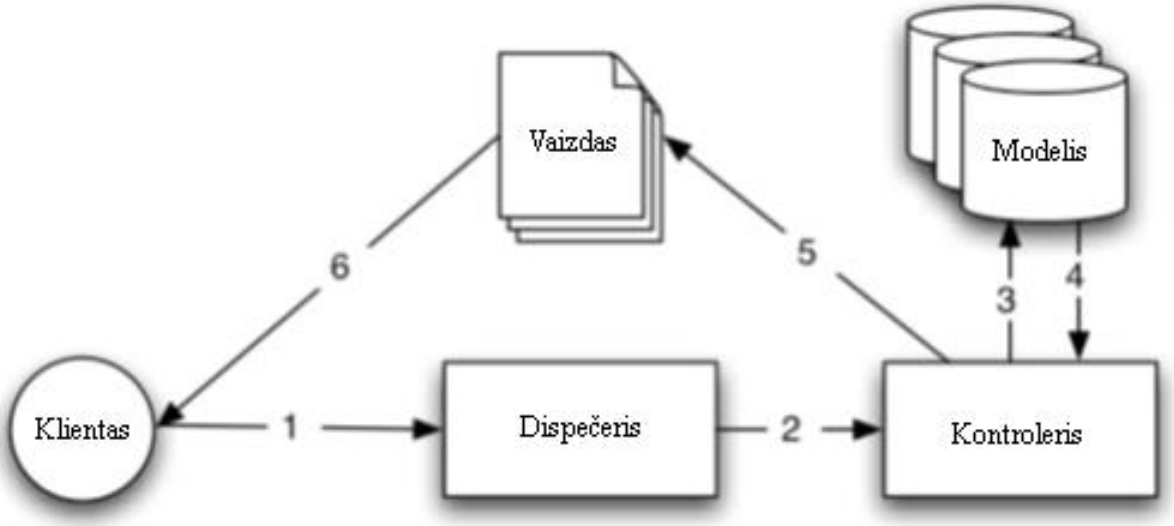
2.3.1 pav. Karkasų naudojimo populiarumas interneto svetainėse [20]

PHP programavimo kalbos karkasų analizei buvo pasirinkti 6 karkasai (*CakePHP, Laravel, Symfony, Yii, CodeIgniter, Zend*), kurie yra populiariausi šiuo metu. Visi šie programavimo kalbos karkasai naudoja *MVC* šabloną.

2.3.1. *CakePHP* karkasas

Pirmoji šio karkaso versija buvo išleista dar 2006 metais. Šis karkasas yra nemokamas bei atviro kodo, sukurtas greitam interneto svetainių kūrimui. *CakePHP* karkaso savybės: lanksti licencijavimo tvarka, suderinta su 5.2.8 ir vėlesne *PHP* programavimo kalbos versija, integruota duomenų bazių *CRUD* (pagrindinės funkcijos, tokios kaip: duomenų gavimas, įrašymas, redagavimas ir trynimas) sistema, vartotojas gali pats nurodyti, kaip jo programa bendraus su duomenų baze (angl. *application scaffolding*), kodo generavimas, *MVC* architektūra, užklausų dispečeris, sudarantis švarias nuorodas ir maršrutus, integruota patikra, greitas ir lankstus šablonų panaudojimas, vaizdo sluoksnio padėjėjas, skirtas *AJAX* tipo užklausoms, *Javascript* programavimo kalbai ir *HTML* žymų kalbai, elektroninio pašto, sausainiukų, sesijos ir užklausų komponentai, lankstus priėjimas prie sistemoje naudojamų failų (angl. *ACL*), duomenų valymas, lanksti podėliavimo sistema, lokalizacija ir kitų [6].

Užklausos vykdymo etapai *CakePHP* karkase yra pavaizduoti paveikslėlyje (žr. 2.3.1.1 pav.). Visas procesas prasideda, kai vartotojas pateikia užklausą užkrauti naują puslapį ar prašo kokių nors resursų ir panašiai. Pirmiausia atėjusią užklausą apdoroja dispečeris, parinkdamas tinkamą kontrolerį užklausai įvykdyti. Kai užklausa pasiekia tinkamą kontrolerį, kontroleris komunikuoja su modelio sluoksniu, kad atliktų tam tikrus veiksmus su reikalingais duomenimis. Po šių veiksmų kontroleris grąžina rezultatą reikiamam vaizdo objektui, kuris sugeneruoja pagrindinį vaizdą. Visiškai sugeneruotas vaizdas pateikiamas vartotojui, o tai reiškia, jog puslapis yra užkraunamas interneto naršyklėje. Beveik kiekviena užklausa vykdoma pateiktu modeliu [6].

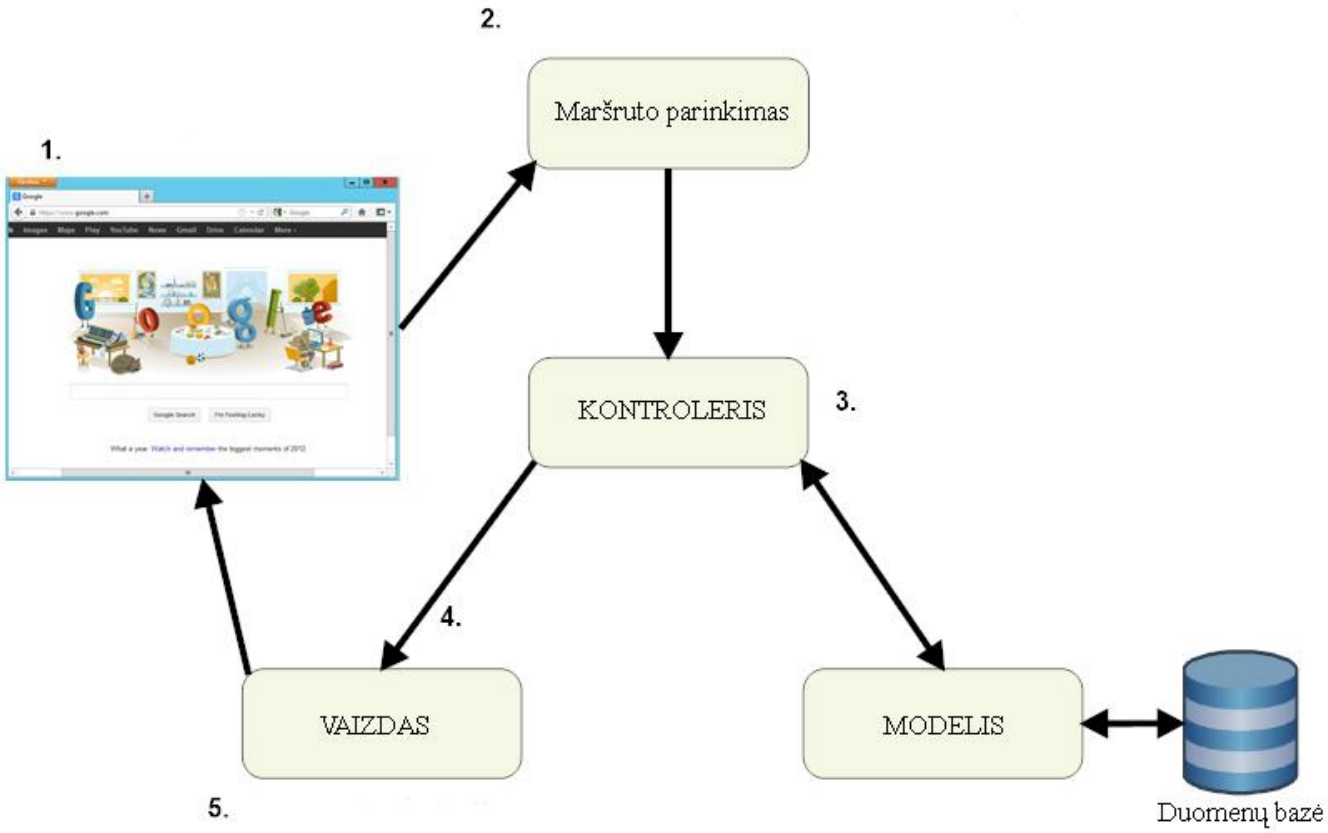


2.3.1.1 pav. Tipinė užklausa *CakePHP* karkase

2.3.2. *Laravel* karkasas

Pirmoji šio karkaso versija buvo išleista dar prieš kelis metus. Šis karkasas taip pat naudoja *MVC* modelį. *Laravel* karkasas vadovaujasi principu *DRY* (*don't-repeat-yourself*), tai reiškia, kad funkcionalumas yra parašytas vieną kartą ir niekur kitur nesidubliuoja. Šis karkasas siūlo aplinką, kurioje lengva sujungti kodą, esantį skirtinguose komponentuose. Šis karkasas taip pat daug dėmesio skiria kodo testavimui. Kodo rašymas bei funkcijų kūrimas vyksta lygiagrečiai su testavimu: parašius tam tikrą funkcionalumą jis yra testuojamas naudojant *UNIT* (nedalomus) testus, taip užtikrinama, kad funkcionalumas veikia gerai ir galima judėti toliau žinant, kad prieš tai atlikti veiksmai yra teisingi. Programuotojas bendrauja su šiuo karkasu per komandinės eilutės įrankį, kuris generuoja ir valdo visą karkaso aplinką. Šis įrankis vadinamas *Artisan*, jis gali būti naudojamas generuojant kodo griaučius ar duomenų bazės schemas [7].

Tipinis *Laravel* programos, kuri naudoja *MVC* modelį, veikimo principas yra pavaizduotas toliau pateiktame paveikslėlyje (žr. 2.3.2.1. pav.). Principas yra identiškas anksčiau minėtam *CakePHP* karkasui. Interneto naršyklė komunikuodama su *Laravel* karkaso programa siunčia užklausą, kuri yra gaunama žiniatinklio serveryje ir iš karto perduodama *Laravel* karkaso maršrutų parinkimo įrankiui. Šis įrankis gauna užklausą ir ją nukreipia į tinkamą kontrolerio klasę. Kai kuriais atvejais kontroleris iš karto sugeneruoja vaizdą ir išsiunčia į interneto naršyklę. Dažniausiai dinaminiuose interneto puslapiuose kontroleris bendrauja su modeliu, kuris yra *PHP* objektas, nurodantis programos elementą, taip pat komunikuoja su duomenų baze. Panaudojus modelį kontroleris sugeneruoja galutinį vaizdą (*HTML, CSS* ir paveiksliukai) ir grąžina galutinį puslapį į vartotojo interneto naršyklę. Šis karkasas atskiria vaizdą, modelį ir kontrolerį laikydamas jų kodą atskiruose failuose ir direktorijose [7].

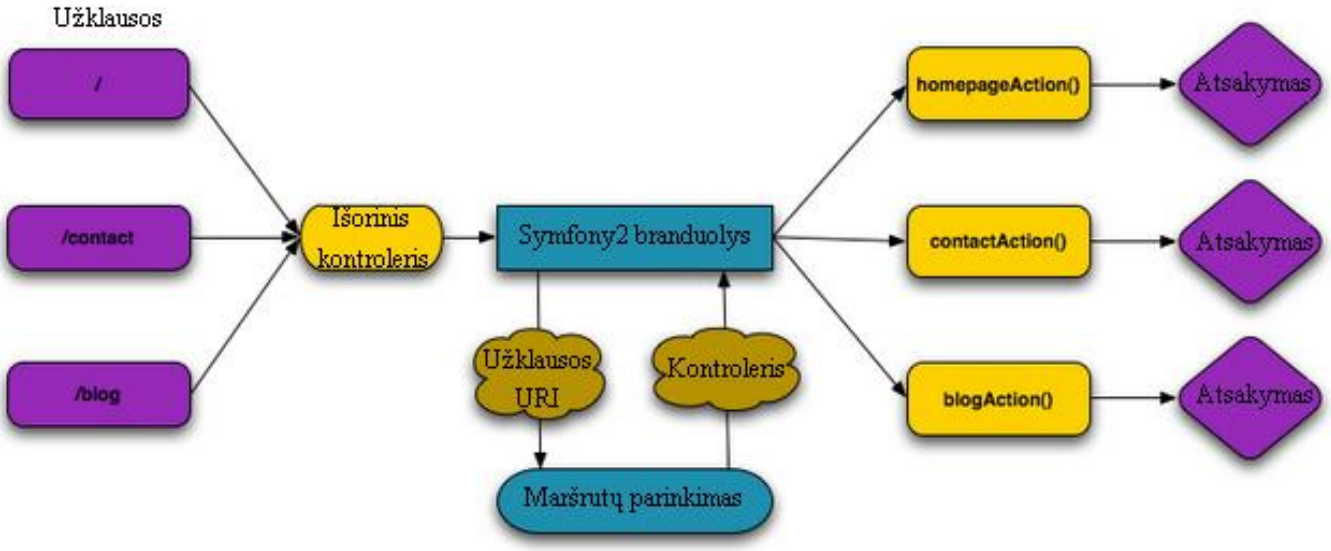


2.3.2.1 pav. Tipinė užklausa Laravel karkase.

2.3.3. *Symfony* karkasas

Šis karkasas yra *PHP* biblioteka, kuri remiasi dviem taisyklėmis: pirmoji – suteikti vartotojui pasirinkimą tarp įvairiausių komponentų įskaitant trečiųjų šalių komponentus bei antroji taisyklė – pateikti protingą konfigūraciją bei biblioteką, kuri tarsi sujungia visas sistemos dalis į vieną. Pagrindinis *Symfony* karkaso tikslas yra integruoti daug nepriklausomų įrankių. Nors šis karkasas vadinamas *Symfony*, jis gali būti visiškai pakeičiamas. *Symfony* karkasas jungia virš 20 bibliotekų, kurios gali būti panaudojamos bet kuriamame projekte. Šios bibliotekos yra vadinamos *Symfony* komponentais. Keletas pagrindinių *Symfony* karkaso komponentų: *HttpFoundation*, susidedanti iš užklausų ir atsakymų klasių, taip pat turinti klasių, kurios skirtos valdyti sesijas ir įkelti failus; *Routing*, galingas ir greitas įrankis maršrutų parinkimo sistemai, kuris leidžia sujungti konkrečią *URI* (pavyzdžiui, /kontaktai) su tam tikra informacija, kaip užklausa turėtų būti apdorota; *Form*, pilnas ir lankstus karkasas, skirtas kurti formoms ir jas apdoroti; *Validator*, sistema, skirta kurti taisyklėms (pvz: elektroninio pašto adreso įvedimo lauke turi būti @ simbolis) apie vienokio ar kitokio formato duomenis ir jų laikytis patvirtinant formą; *ClassLoader*, automatiškai užsikraunanti biblioteka, kuri leidžia naudoti *PHP* programavimo kalbos klases, nereikalaujant, kad jos būtų įtrauktos tame faile; *Templating*, įrankių rinkinys, naudojamas generuoti šablonus, tvarkyti šablonų paveldimumą, kai vienas šablonas naudoja kitą ir atlikti kitas pagrindines šablonų funkcijas; *Security*, galinga biblioteka, skirta atlikti saugumo funkciją programoje; *Translation*, karkasas, skirtas versti įvairiausius žodžius, vartojamus programoje. Reikėtų paminėti, kad išvardinti komponentai yra nepriklausomi ir jie gali būti naudojami nepaisant to, ar *Symfony* karkasas yra naudojamas toje programoje [8].

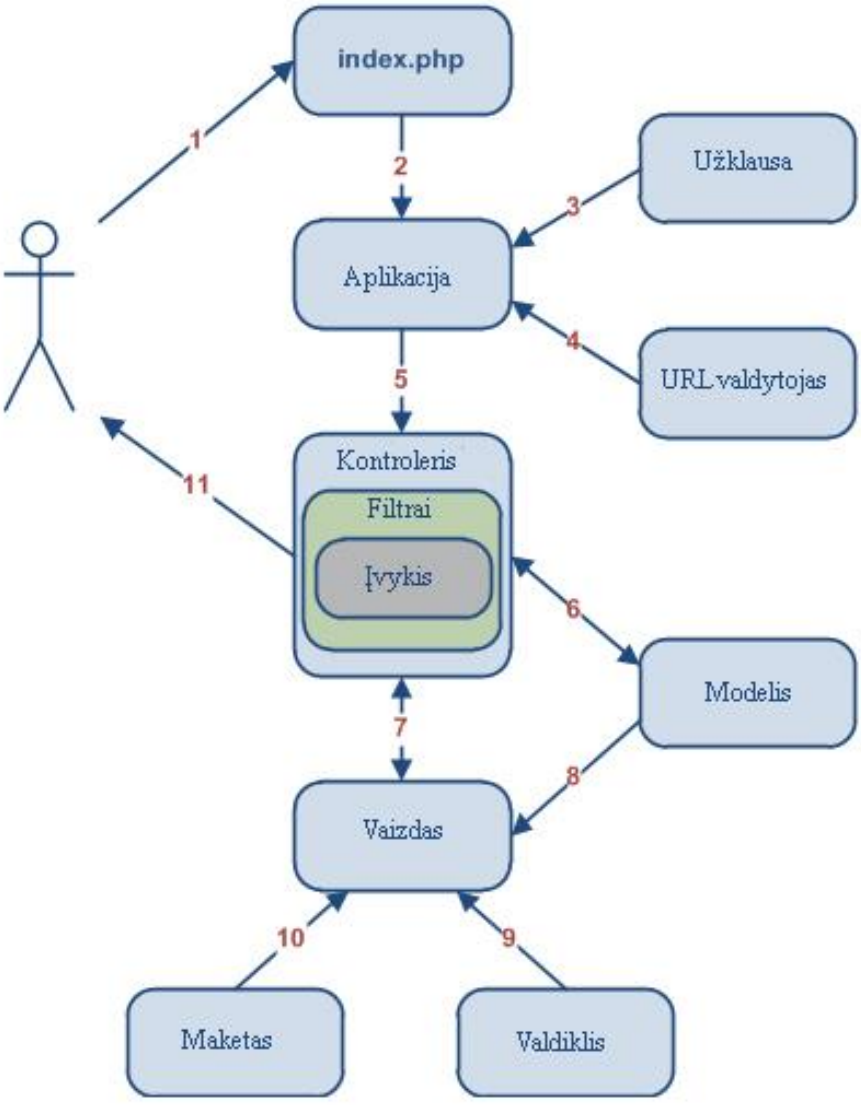
Symfony karkasas naudoja toliau pateiktą schemą (žr. 2.3.3.1 pav.) kiekvienai užklausai apdoroti. Ateinančios užklausos yra interpretuojamos maršrutų parinkimo įrankio ir paduodamos kontroleriui, kuris grąžina atsakymo objektą. Kiekvienas vartotojo programos puslapis yra nurodytas maršrutų konfigūracijos faile, kuris sujungia skirtingas nuorodas su skirtingomis *PHP* funkcijomis. Pagrindinė užduotis kiekvienai funkcijai, arba kontroleriui, yra panaudoti informaciją, atėjusią iš užklausos ir sukurti bei grąžinti atsakymo objektą.



2.3.3.1 pav. *Symfony* karkaso veiksmų sekos schema

2.3.4. *Yii* karkasas

Šis karkasas yra nemokamas ir atviro kodo sukurtas penktai *PHP* programavimo kalbos versijai. Kaip ir *Laravel* karkasas, *Yii* karkasas vadovaujasi *DRY* principu. Šis karkasas buvo pradėtas kurti 2008 metais, jį kūrė *Qiang Xue* [9]. *Yii* karkasas susideda iš šių įrankių: *MVC* modelis; Duomenų priėjimo objektai (angl. *Data Access Objects*), kurie leidžia prisijungti prie skirtingų duomenų bazių valdymo sistemų naudojant tik viena sąsają; Užklausų statytojas (angl. *query builder*), kuris siūlo objektiškai orientuotą metodą kurti *SQL* tipo užklausoms taip išvengiant *SQL* įterpimo (ang. *injection*) atakų; Formų kūrimas ir patikra; *AJAX* tipo užklausų valdiklis, kuris yra integruotas su *jQuery* biblioteka, šis valdiklis susideda iš automatinio laukų užpildymo, medžio tipo duomenų vaizdavimo, duomenų tinklo ir kitų; Autentifikavimas ir autorizacija; Stilių ir temų keitimas, leidžiantis greitai pakeisti interneto puslapio išvaizdą; Žiniatinklio servisai, šis karkasas palaiko *WSDL* tipo servisų specifikacijų kūrimą ir užklausų apdorojimą; Lokalizacija, kuri leidžia atlikti žinučių vertimą, datos ir skaičių formatavimą; Kelių sluoksnių podėliavimo sistema, kuri palaiko duomenų, puslapių bei fragmentų podėliavimą, paspartinantį programos veikimą; Klaidų tvarkymas ir registravimas; Apsauga, karkasas apsaugo nuo įvairių tipų atakų, tai *SQL* įterpimas, *XSS*, *CSRF* ir slapukų klastojimo; *Unit* ir funkcionalumo testai, kurie naudoja *PHPUnit* ir *Selenium* įrankius, skirtus atlikti testavimą; Automatinis kodo generavimas, kuris padeda sugeneruoti kodą įvedimo laukams ir *CRUD* funkcijoms; Draugiškas trečiųjų šalių kodui, pavyzdžiui, naudojant *Yii* karkasą programoje taip pat galima naudoti kodą iš *Zend* karkaso; Detali dokumentacija; Komponentų biblioteka; [10].



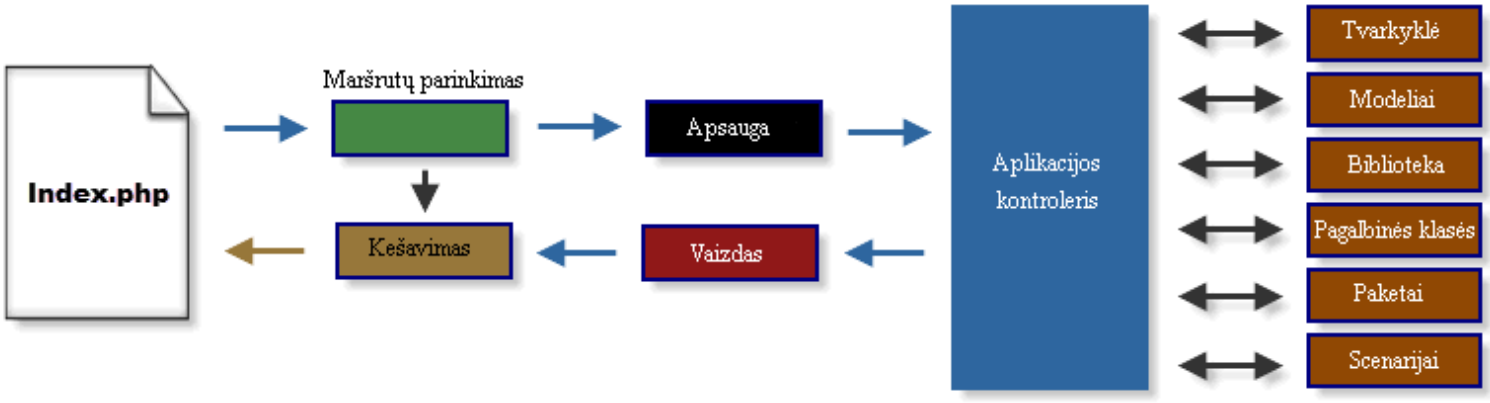
2.3.4.1 pav. Tipinė *Yii* karkaso darbo eiga

Tipinė *Yii* karkaso veikla yra pavaizduota schemoje (žr. 2.3.4.1 pav.). Pirmiausia vartotojas atlieka užklausą kreipdamasis į tam tikrą interneto puslapį, atėjusią užklausą pirmiausia gauna žiniatinklio serveris, kuris apdoroją užklausą sukurdamas *bootstrap* tipo scenarijų (schemoje 1 žingsnis). Šis scenarijus sukuria programą ir vykdo ją (schemoje 2 žingsnis). Programa pasiekia vartotojo pateiktą užklausą iš programos komponento schemoje pavadinta užklausa (schemoje 3 žingsnis). Programa nustato, kurį kontrolerį ir kokį jo veiksmą reiks panaudoti norint atlikti užklausą (schemoje 4 žingsnis). Nustačius kontrolerį sukuria jo objektą, kuris toliau apdoroja užklausą. Kontroleris nustato, kuris veiksmas bus naudojamas ir sukuria filtrus, skirtus tam veiksmui. Jeigu filtrai leidžia atlikti veiksmą, jis atliekamas (schemoje 5 žingsnis). Veiksmas skaito *Post* tipo modelį, kurio *ID* sutampa su duomenų bazėje esančiu (schemoje 6 žingsnis). Veiksmas sugeneruoja vaizdą su nuskaityto modelio duomenimis (schemoje 7 žingsnis). Vaizdas skaito ir pavaizduoja atributus iš modelio (schemoje 8 žingsnis). Taip pat vaizdas įvykdo kelis valdiklius (schemoje 9 žingsnis). Vaizdo generuojamas rezultatas yra apvelkamas į tam tikrą maketą (schemoje 10 žingsnis) ir galiausiai pavaizduojamas vartotojui (schemoje 11 žingsnis) [11].

2.3.5. *CodeIgniter* karkasas

Kaip ir prieš tai aprašyti karkasai, šis taip pat naudoja *MVC* modelį, kuris padeda atskirti sistemos dalis. *CodeIgniter* sukūrė *Rick Ellis* iš *EllisLab* korporacijos. Minėtas karkasas pasižymi tokiomis savybėmis: pilnai įgyvendintos duomenų bazių klasės, kurias palaiko keletas platformų; aktyvių duomenų bazės įrašų palaikymas; formų ir duomenų patikra; Apsauga ir *XSS* filtravimas; sesijos valdymo sistema; elektroninių laiškų siuntimo klasė, kuri palaiko priedus, *HTML*/teksto elektroninius laiškus bei daug protokolų (*sendmail, SMTP, mail*); paveikslėlių redagavimo (kirpimo, dydžio keitimo, sukimosi ir panašiai) biblioteka, palaiko *GD, ImageMagick* ir *NetPBM*; failų įkėlimo klasė; *FTP* klasė; Lokalizacija; Puslapiavimas; Duomenų kodavimas; Lyginamoji analizė (angl. *Benchmark*); Pilnas puslapio podėliavimas; Klaidų registracija; Programos profiliavimas; Kalendoriui skirta klasė; Vartotojų agento klasė; Pašto kodavimo klasė; Šablonų variklio klasė; *XML-RPC* biblioteka; *Unit* testų klasė; paieškos variklis; lankstus maršrutų parinkimas; Didelė pagalbinių klasių biblioteka [12].

Duomenų srautų judėjimas *CodeIgniter* karkase yra pavaizduotas toliau esančiame paveikslėlyje (žr. 2.3.5.1 pav.). *Index.php* veikia kaip išorinis kontroleris, inicijuodamas pagrindinius resursus, reikalingus šiam karkasui. Maršrutų parinkimo įrankis analizuoja *HTTP* užklausą įvertindamas, ką toliau su ja daryti. Jeigu podėliavimo failas egzistuoja šiai užklausai jis iš karto yra nusiunčiamas į interneto naršyklę praleisdamas visus kitus veiksmus. Toliau, prieš užkraunant programos kontrolerį, užklausos duomenys yra patikrinami, kad atitiktų saugumo reikalavimus. Kontroleris užkrauna modelį, pagrindines bibliotekas, pagalbines klases ir kitus resursus, reikalingus įvykdyti užklausai. Tada yra sugeneruojamas vaizdas ir nusiunčiamas į interneto naršyklę. Jeigu podėliavimo funkcija yra įjungta, tai puslapis yra podėliuojamas, kad ateityje vykdant tą patį veiksmą nereikėtų visų etapų įvykdyti iš naujo [13].

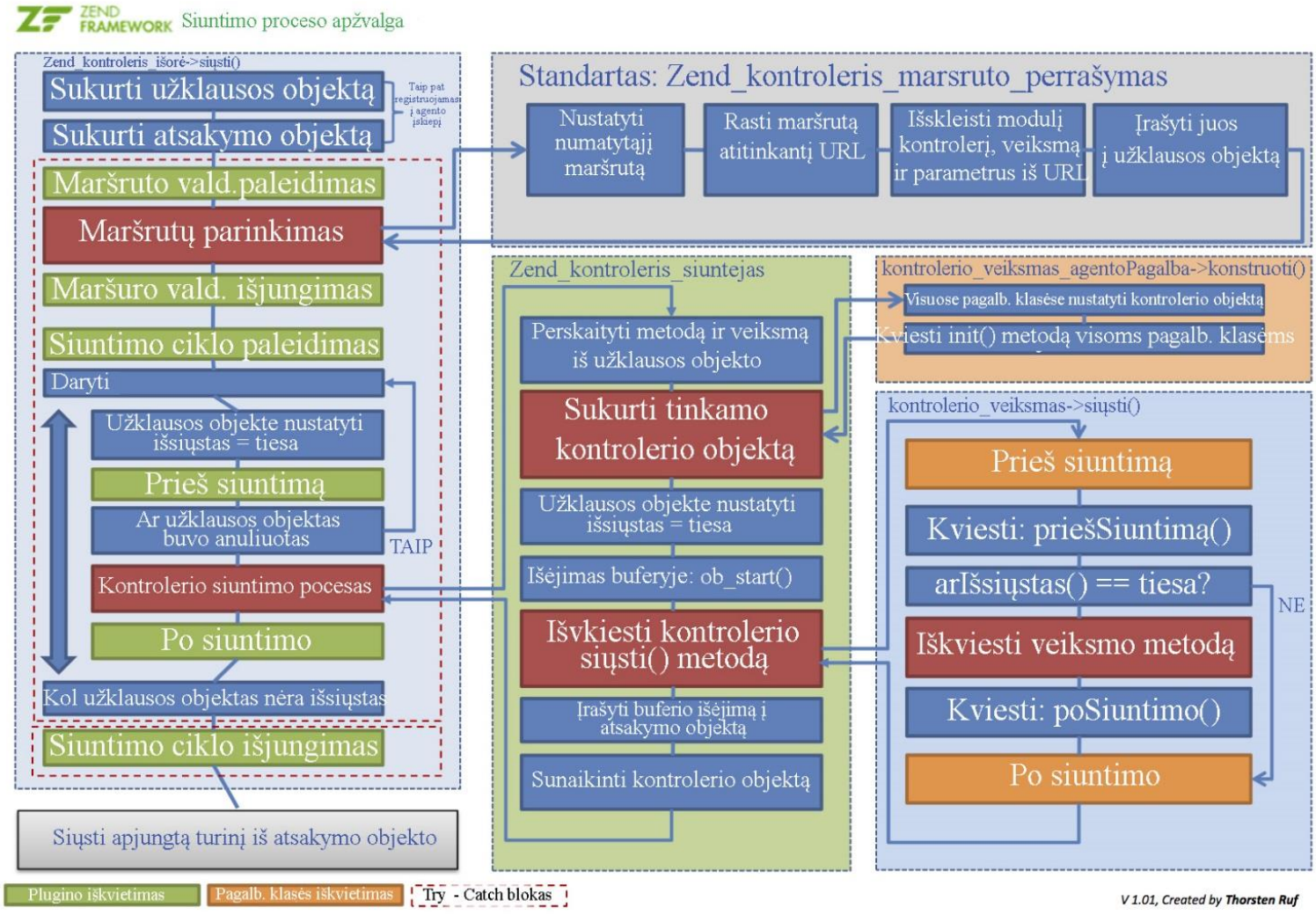


2.3.5.1 pav. Tipinė *CodeIgniter* karkaso duomenų srauto schema

2.3.6. *Zend* karkasas

Zend yra atviro kodo karkasas, skirtas kurti interneto programas su *PHP* programavimo kalbos 5.3 ir naujesne versijomis. Šis karkasas naudoja visiškai pilnai objektiškai orientuotą kodą. Taip pat turintis unikalią struktūrą, kuri yra sukurta vadovaujantis *SOLID* objektiškai orientuoto dizaino principu. Tokia architektūra leidžia vartotojams naudoti tokius komponentus, kokius jie tik nori. *Zend* karkasas palaiko *Pyrus* ir *Composer* diegimo sistemas. Testavimui naudoja *PHPUnit* įrankį ir *Travis CI* kaip nuolatinio komponentų integravimo įrankį. Be abejo, šie komponentai gali būti naudojami atskirai ir be *Zend* karkaso, tačiau pastarasis apjungia visus komponentus. Šis karkasas taip pat siūlo tvirtą, geros greitaveikos *MVC* modelio įgyvendinimą, taip pat duomenų bazės abstrakcijas, kurias lengva panaudoti. Formų komponentas, kuris įgyvendina *HTML5* formų generavimą, patikrą ir filtravimą. Kiti komponentai, tokie kaip *Zend* autentifikacija ir *Zend* leidimai, leidžia atlikti veiksmus su vartotojų rolėmis, teisėmis ir autorizacija [14].

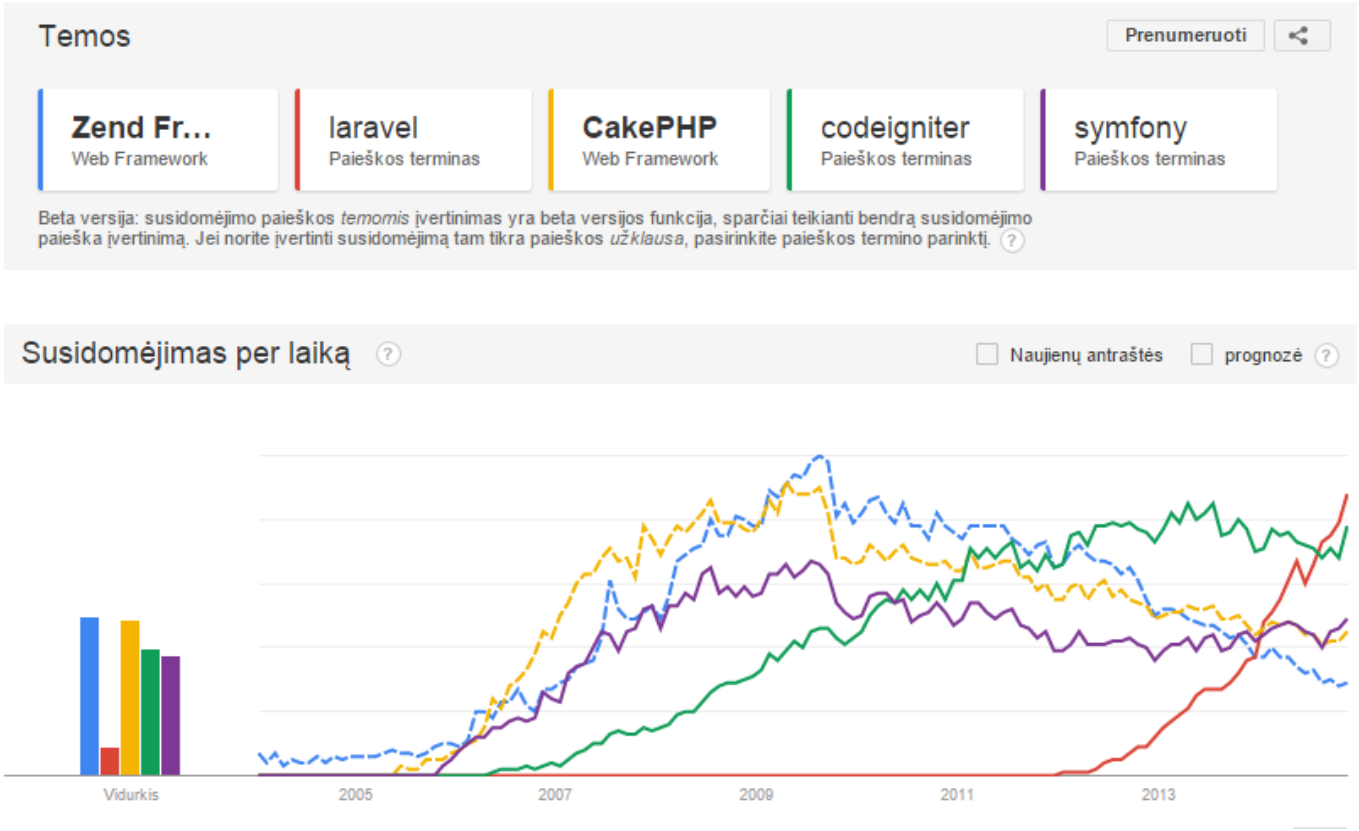
Išsami *Zend* karkaso siuntimo proceso schema yra pateikta toliau esančiame paveikslėlyje (žr. 2.3.6.1 pav.) [15].



2.3.6.1 pav. *Zend* karkaso siuntimo proceso schema

2.4. *PHP* programavimo kalbos karkasų palyginimas

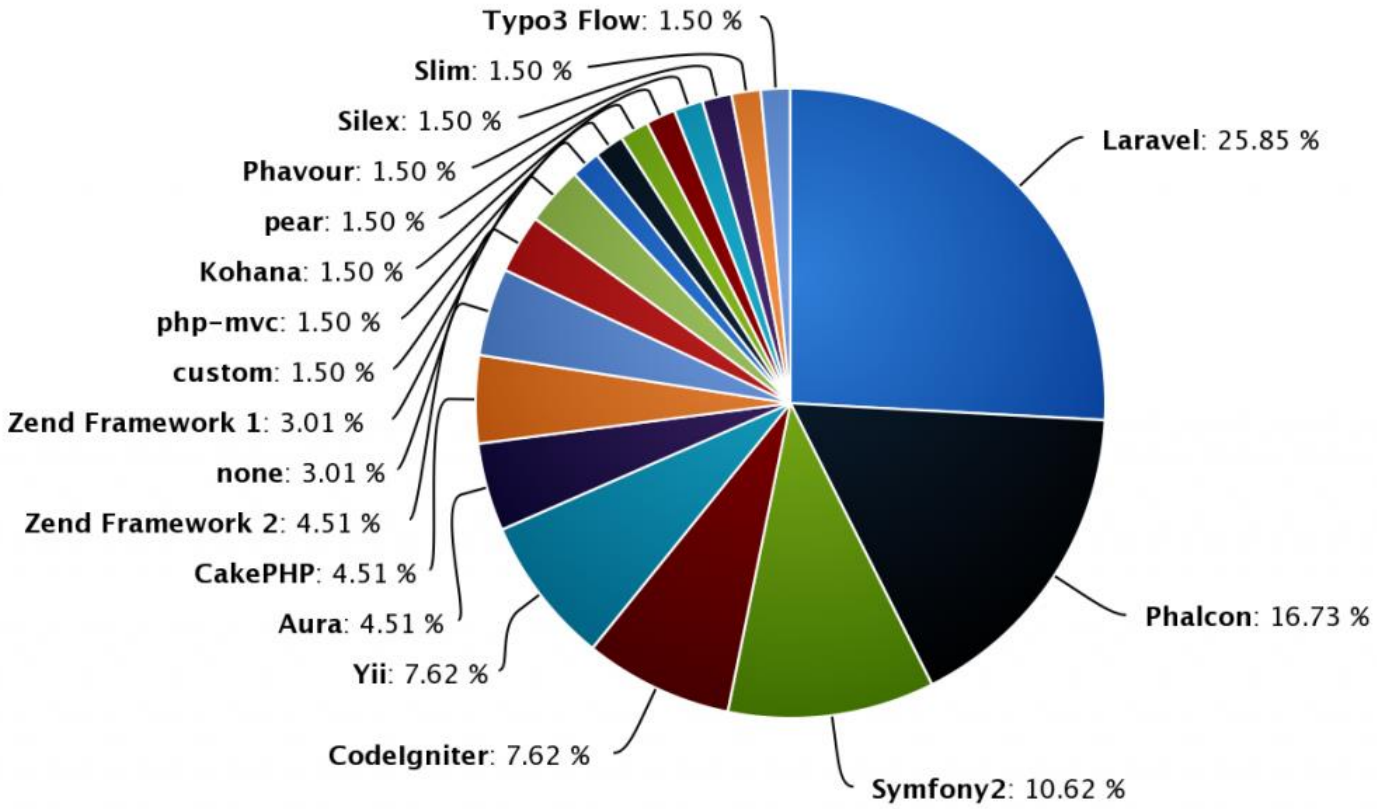
Vienas iš aspektų, galimų palyginti *PHP* programavimo kalbos karkasus, yra populiarumas. Naudojantis *Google Trends* įrankiu buvo nustatytas 5 karkasų populiarumas pagal užklausas *Google* paieškos sistemoje, rezultatai yra pateikti toliau esančiame grafike (žr. 2.4.1 pav.). Šiame grafike paieškų rezultatai yra vaizduojami nuo 2004 metų iki dabar. Galima pastebėti, jog pirmiausiai iš paminėtų karkasų atsirado *Zend* karkasas, jo paieškos jau aptinkamos 2004 metais, be to, šis karkasas buvo pasiekęs didžiausią populiarumą tarp tiriamų karkasų, tai įvyko 2009 metų pabaigoje, tačiau nuo tų metų šio karkaso populiarumas tik mažėja. Toliau pagal senumą yra *Symfony* karkasas, kurio populiarumas pradėjo kilti apie 2006 metus ir apie 2010 metus pasiekė savo piką, nuo to laiko šio karkaso populiarumas truputį sumažėjo, bet išliko gana stabilus. *CakePhp* karkasas pradėjo populiarėti dar 2005 metų viduryje ir didžiausią populiarumą pasiekė apie 2010 metus, po to laiko populiarumas sumažėjo ir po truputėlį mažėja. *CodeIgniter* karkasas pradėjo kelti vartotojų susidomėjimą 2006 metų viduryje ir stabiliai populiarėjo iki 2013 metų pirmo ketvirčio, nuo tada šio karkaso populiarumas šiek tiek yra sumažėjęs. Paskutinis iš tiriamų karkasų *Laravel* yra vėliausiai pradėjęs populiarėti (apie 2012 metus) ir sparčiai populiarumas kyla iki šiol. Apibendrinant matyti, kad pagal vartotojų paieškos rezultatus populiariausias *PHP* programavimo kalbos karkasas yra *Laravel*, kuris sparčiausiai kyla šiuo metu, o *CodeIgniter*, *Symfony* ir *CakePHP* karkasų populiarumas pastaruoju metu taip pat didėja. Paskutinėje vietoje šiuo metu yra *Zend* karkasas, kurio populiarumas vis dar ritasi žemyn. Šiame grafike nepavyko įkelti visų 6 karkasų, kadangi įrankis palaiko tik 5 temų susidomėjimą. Tačiau *Yii* karkaso populiarumas pagal paieškų skaičių yra trečioje vietoje (pradėjo populiarėti apie 2009 metus) ir šiuo metu kyla [16].



2.4.1 pav. *PHP* programavimo kalbos karkasų paieškų rezultatai

Remiantis sitepoint.com atlikta žmonių apklausa apie populiariausius *PHP* programavimo kalbos karkasus buvo sudaryta toliau pavaizduota diagrama (žr. 2.4.2 pav.). Šis tyrimas buvo atliktas 2013 metų pabaigoje. Dauguma atsakymų buvo atmesti dėl to, kad žmonės dažnai minėjo *Wordpress* įrankį, kuris yra turinio valdymo sistema, o ne *PHP* karkasas. Atsakymai taip pat buvo atmesti žmonių, kurie yra naudoję tik vieną karkasą. Taigi apklausos rezultatai yra pakankamai objektyvūs [17].

Pagal pateiktus rezultatus populiariausi ir geriausi karkasai yra *Laravel*, *Phalcon*, *Symfony2*, *CodeIgniter* ir *Yii*. Panaikinus rezultatus, kurie yra neadekvatūs, *Laravel* neteko apie pusės vartotojų balsų, kadangi šie vartotojai negalėjo įrodyti savo tinkamos kvalifikacijos ir darbo su šiuo karkasu, nepaisant to šis karkasas vis tiek liko populiariausiu pasirinkimu. Vieną iš pranašumų, kodėl buvo pasirinktas būtent *Laravel* karkasas, vartotojai minėjo kaip lengvai išmokstamą karkasą, kitas privalumas, kuris buvo paminėtas, tai gerai veikiantis pokalbių langas, kuriame galima pasiklausinėti apie šį karkasą ir gauti greitus ir tikslius atsakymus. *Phalcon* karkaso didžiausiu privalumu buvo įvardintas greitis, lyginant su kitais karkasais, taip pat buvo paminėtas šio karkaso paketas, kuris yra sujungtas iš kelių komponentų (*ORM*, šablonų variklio, *PHQL* ir daugiau), vadinasi, nebereikia naudotis trečiųjų šalių bibliotekomis ir viskas lieka atmintyje, todėl pagreitėja karkaso veikimas. Vartotojai taip pat paminėjo, jog šis karkasas yra įrašomas kaip plėtinys. *Symfony2* karkasas buvo paminėtas kaip labiausiai modalinis ir praplečiamas karkasas, taip pat turintis daugiausiai savybių. Tačiau šio karkaso vartotojai pripažino, kad *Symfony2* yra kiek išsipūtęs ir lėtas karkasas dėl savo gausaus kiekio funkcionalumo [17].



2.4.2 pav. *PHP* programavimo kalbos karkasų populiarumas 2013 metų pabaigoje

2013 metais Pietų Konektikuto valstijos universitete buvo atliktas tyrimas, kurio metu buvo sukurtos trys vienodo funkcionalumo interneto svetainių versijos. Jas sukūrė universiteto absolventas, kuris gerai mokėjo *PHP* programavimo kalbą, bet nebuvo susidūręs su šios kalbos karkasais. Interneto programos buvo sukurtos naudojant *Eclipse PDT* programavimo aplinką [20].

Pirmoji sistemos versija sukurta nenaudojant jokių karkasų (*PHP* programavimo kalbos versija 5.3.10). Taip pat šioje versijoje nebuvo realizuota *MVC* šablono architektūra. Kitos dvi versijos buvo sukurtos naudojantis karkasais *CakePHP* ir *CodeIgniter*. Visos programos versijos buvo įdėtos vietiniame *WAMP* serveryje. Programos buvo analizuojamos pagal tris skirtingas charakteristikas: programavimo sudėtingumą (kodo eilučių skaičių), greitį ir apsaugą [20].

Kodo eilučių kiekis buvo nustatomas naudojant *CLOC* (*Count Lines of Code*) programą. Gautas kodo eilučių kiekis kiekvienai programos versijai atspindi faktą, jog naudojant paprastą *PHP* implementaciją kodas nėra perpanaudojamas, priešingai negu naudojant *PHP* programavimo kalbos karkasus. Rezultatai yra pavaizduoti toliau esančioje lentelėje (žr. 2.4.1 lent.), kurioje matyti, jog *CakePHP* ir *CodeIgniter* naudoja bibliotekas ir pagalbines klases, kurios programoje yra naudojamos daug kartų. CakePHP karkasas naudoja mažiausiai kodo eilučių, kadangi šiame karkase priėjimas prie duomenų bazių yra padarytas automatiškai naudojant modelio susiejimą su duomenų lentelėmis, todėl sutaupomas kodo eilučių skaičius. Akivaizdu, jog programą rašant paprastu *PHP* kodu, kodo eilučių skaičius išauga dėl to, kad visas bibliotekas ir įvairiausias pagalbines klases reikia pasirašyti pačiam, taip pat yra mažiau kodo perpanaudojimo atvejų. Galima daryti išvadą, jog rašant programą, naudojantis *CakePHP* karkasu, programuotojams reikia parašyti mažiau kodo eilučių ir tai lemia greitesnį sistemos kūrimą, lengvesnį palaikymą ir mažesnę tikimybę atsirasti klaidoms [20].

2.4.1 lentelė. Kiekvienos programos kodo eilučių skaičius

Pavadinimas	***PHP***	***CSS***	***PHP* failų skaičius**	***CSS* failų skaičius**
Paprastas *PHP*	3002	650	42	13
CodeIgniter	2321	621	36	13
CakePHP	1408	680	16	13

Kiekvienos versijos greitis buvo nustatomas naudojantis *PHP* plėtiniu *XDebug*. Šis įrankis suteikia galimybę gauti funkcijų trasavimo, atminties paskirstymo informaciją bei laiką, reikalingą atlikti įvairioms užduotims. Greičio rezultatai yra pateikti lentelėje (žr. 2.4.2 lent.). Kaip matyti iš rezultatų, yra labai ryškus skirtumas tarp paprasto *PHP* kodo ir karkasų. Kiekvienam funkciniam komponentui įvykdyti, programos, parašytos naudojantis *PHP* karkasais, užtruko žymiai ilgiau nei be karkasų parašyta programa. Pavyzdžiui, naudojant programą su paprastu *PHP* kodu, prisijungimo ir atsijungimo funkcijos buvo įvykdytos per 96 milisekundes, tuo tarpu naudojant *CodeIgniter* tai truko 23 kartus ilgiau, o naudojant *CakePHP* - 16 kartų. Tai susiję su faktu, jog karkasams atliekant bet kokią funkciją, reikia įkrauti tam tikras bibliotekas, todėl reikia papildomo laiko [20].

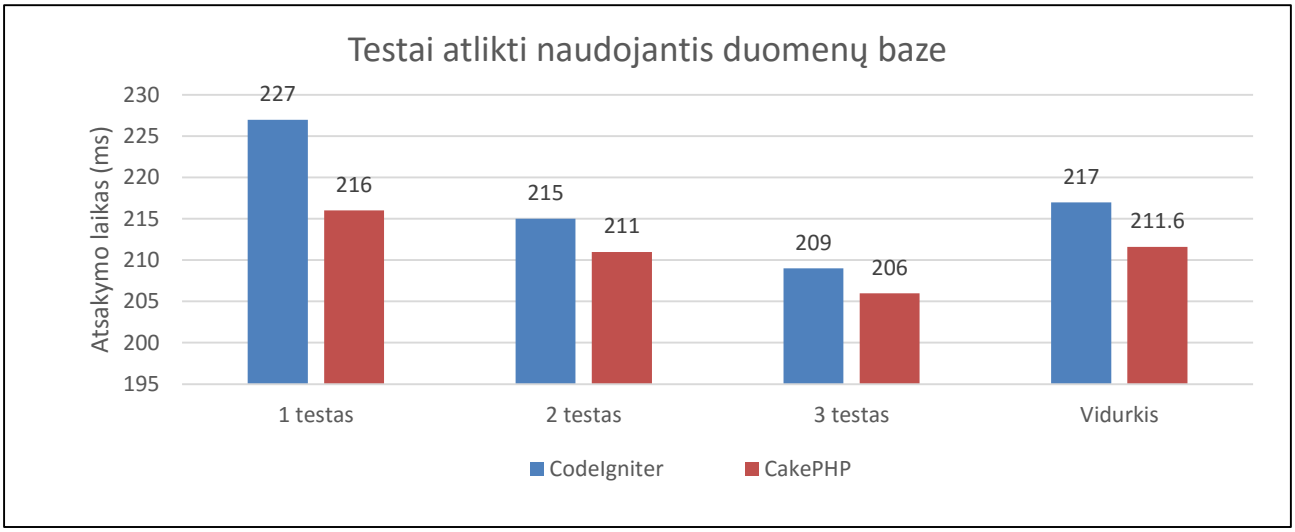
2.4.2 lentelė. Kiekvienos programos greičio palyginimas

Funkcionalumas	**Paprastas *PHP***		***CodeIgniter***		***CakePHP***	
	Laikas (ms)	**Kodo eil.**	**Laikas (ms)**	**Kodo eil.**	**Laikas (ms)**	**Kodo eil.**
Prisijungti/Atsijungti	96	121	2291	192	1559	83
Sesijos valdymas	91	132	739	118	1309	102
Duomenų bazė	143	83	762	89	1492	52
Formų naudojimas	487	209	1338	385	2262	191
Formų patikra	321	292	966	406	2411	310

Atliekant apsaugos analizę buvo panaudotas *Nessus* įrankis. Pagal rezultatus *CakePHP* karkasas pasirodė saugesnis už *CodeIgniter* ir paprastą *PHP* kodą. *CakePHP* karkasas turi apsaugos komponentus, kurie gali užtikrinti apsaugą nuo pagrindinių apsaugos atakų. Tuo tarpu *CodeIgniter* karkasas turi panašias apsaugos priemones (*CSRF* apsaugą ir *XSL*), tačiau tyrimo rezultatai parodė, kad šis karkasas su numatyta konfigūracija nesugebėjo užtikrinti apsaugos nuo *SQL* įterpimo atakos [20].

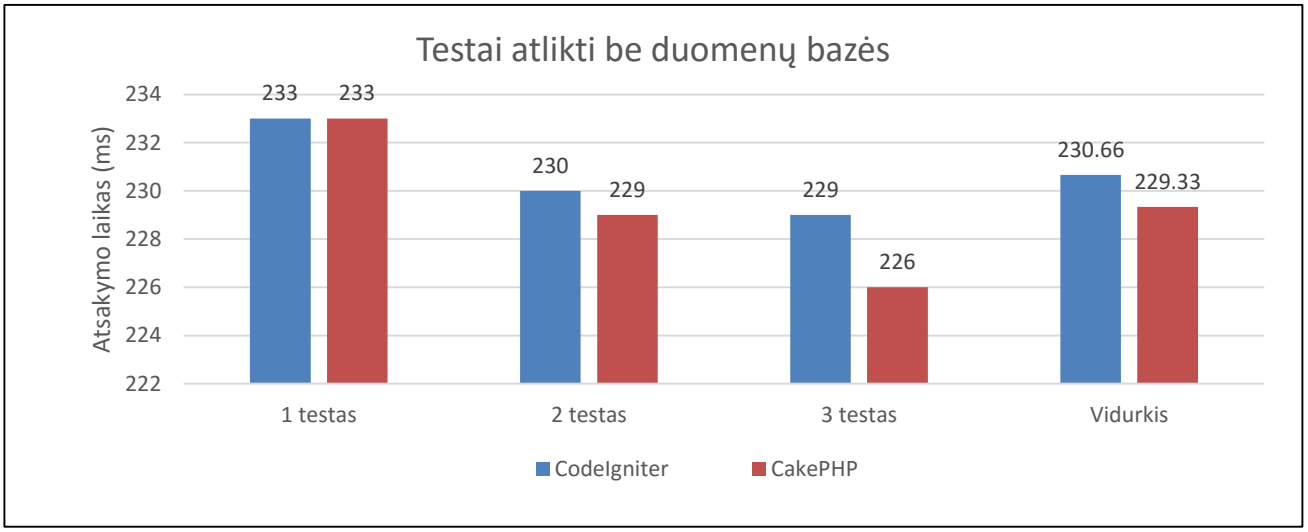
Panašų tyrimą karkasų greičiui matuoti 2012 metais atliko studentas iš Blekingo technologijos instituto, esančio Švedijoje. Jo tyrime taip pat buvo lyginami *CakePHP* ir *CodeIgniter* karkasai. Tikslas buvo išsiaiškinti, kuris iš karkasų greičiau įkrauna internetinį puslapį. Iš viso buvo paleidžiami 6 testai, 3 buvo paleisti naudojantis duomenų baze, kiti trys be jos. Buvo atliekama 20000 užklausų į *Apache* serverį, kur ir buvo įkeltos sistemos [21].

Atliekant testus ir kartu naudojant duomenų bazę, buvo atsitiktinai nustatyta, kurį karkasą testuoti. Toliau esančioje diagramoje (žr. 2.4.3 pav.) yra pavaizduoti šių testų rezultatai. Iš rezultatų matyti, jog skirtumas nėra labai didelis, tačiau *CakePHP* karkasas yra greitesnis 5,34 ms per užklausą, lyginant su *CodeIgniter* karkasu [21].



2.4.3 pav. Testų vidurkiai naudojantis duomenų baze užklausai [21]

Kiti 3 testai buvo atlikti nenaudojant duomenų bazes, gauti rezultatai (žr. 2.4.4 pav.) parodė, jog atsakymo laikas yra beveik identiškas naudojant abu karkasus. Visgi labai nežymiai, vos 1,3 ms, *CakePHP* karkasas pranoko *CodeIgniter*. Taigi tiek naudojant duomenų bazę, tiek be jos, CakePHP karkasu parašyta programa užklausas apdoroja šiek tiek greičiau nei kito tirto karkaso [21].



2.4.4 pav. Testų vidurkiai be duomenų bazės užklausai [21]

Lyginant PHP programavimo kalbos karkasus pagal jų savybes, kurios yra pateiktos toliau esančioje lentelėje (žr. 2.4.3 lent.), galima pastebėti, jog vienintelis *CodeIgniter* palaiko senesnę, ketvirtąją, PHP versiją. Taip pat objektų vaidmenų modeliavimą (angl. *ORM*) palaiko visi, išskyrus *Zend* karkasą. Autentifikacijos nepalaiko vienintelis *CodeIgniter* karkasas. Tokias funkcijas, kaip podėliavimas ir patikra, palaiko visi nagrinėjami *PHP* programavimo kalbos karkasai [18]. Kodo generavimo funkcijos nepalaiko *Zend* ir *CodeIgniter* karkasai. Visi karkasai turi testavimo įrankį *PHPUnit*, tačiau čia išsiskiria *Yii* karkasas, kuris be minėto įrankio dar naudoja ir *Selenium*. Žiūrint į tolimesnes savybes, matyti *Symfony* karkaso pranašumas, kuris vienintelis turi sluoksnių ir *CRUD* generatorių bei kartu su *Laravel* karkasu atlieka pranešimų registravimo funkciją. Kalbant apie apsaugą, *CakePHP*, *Yii* ir *Zend* neturi jokios integruotos apsaugos, kai tuo metu *Laravel*, *CodeIgniter* ir *Symfony* naudoja apsaugą nuo populiariausių atakų. Apibendrinant Symfony ir Laravel atrodo patraukliausiai lyginant karkasų savybes ir populiarumą. Pagal savybes prasčiausiai atrodo *Zend* karkasas, tai taip pat atitinka populiarumo tyrimą [19].

2.4.3 lentelė. PHP karkasų savybės

Požymis	***CakePHP***	***Yii***	***Zend***	***Symfony***	***Laravel***	***CodeIgniter***
Vėliausia versija	2.4.6 (2014-03)	1.1.14 (2013-08)	2.2.0 (2013-05)	2.4 (2013-12)	4.1.24 (2014-03)	2.1.4 (2013-07)
PHP4	✗	✗	✗	✗	✗	✓
PHP5	✓	✓	✓	✓	✓	✓
Moduliai	✓	✓	✓	✓	✗	✓
ORM	✓	✓	✗	✓	✓	✓
EDP	✗	✓	✗	✗	✓	✗
Autentifikacija	✓	✓	✓	✓	✓	✗
Podėliavimas	✓	✓	✓	✓	✓	✓
Patikra (angl. Validator)	✓	✓	✓	✓	✓	✓
Kodo generavimas	✓	✓	✗	✓	✓	✗
Servisai	Trečiųjų šalių	✗	✗	✗	✗	*XML-PRC*
Testavimo biblioteka	*PHPUNIT*	*PHPUNIT, Selenium*	*PHPUNIT*	*PHPUNIT*	*PHPUNIT*	*PHPUNIT*
Sluoksnių generatorius	✗	✗	✗	✓	✗	✗
Meniu generatorius	✗	✗	✗	✗	✗	✗
CRUD generatorius	✗	✗	✗	✓	✗	✗
Pranešimų registravimas (ang. *Logging*)	✗	✗	✗	✓	✓	✗
Apsauga nuo *XSS*	✗	✗	✗	✓	✓	✓
Apsauga nuo *XSRF*	✗	✗	✗	✓	✓	✓
Apsauga nuo *SQL* įterpimo	✗	✗	✗	✗	✓	✓

3. PROJEKTINĖ DALIS

3.1. Architektūros pateikimas

Pilnam architektūros pateikimui naudojami vaizdai yra pateikti lentelėje (žr. 3.1.1 lent.). Programinės įrangos architektūra yra pateikiama naudojantis *UML* standartu [1].

3.1.1 lentelė. Architektūrai patekti naudojami vaizdai.

Vaizdas	Aprašymas	Modeliavimo elementai
Panaudojimo atvejų	Apibūdina sistemos panaudojimo atvejus ir jų scenarijus.	Panaudojimo atvejų diagrama (angl. *use case*)
Sistemos statinis	Apibūdina sistemoje naudojamus objektus ir sąryšius tarp jų.	Klasių diagrama (angl. *class*)
Sistemos dinaminis	Apibūdina sistemos dinaminį vaizdą.	Sąveikos diagrama (angl. *interaction*), būsenos diagrama (angl. *state*), veiklos diagrama (angl. *activity*)
Išdėstymo	Apibūdina aplinką, kurioje sistema veiks, bei sąryšius tarp skirtingų aplinkų.	Išdėstymo modelis
Duomenų	Apibūdina sistemos duomenų objektus bei sąryšius tarp jų.	Duomenų modelis

3.2. Architektūros tikslai ir apribojimai

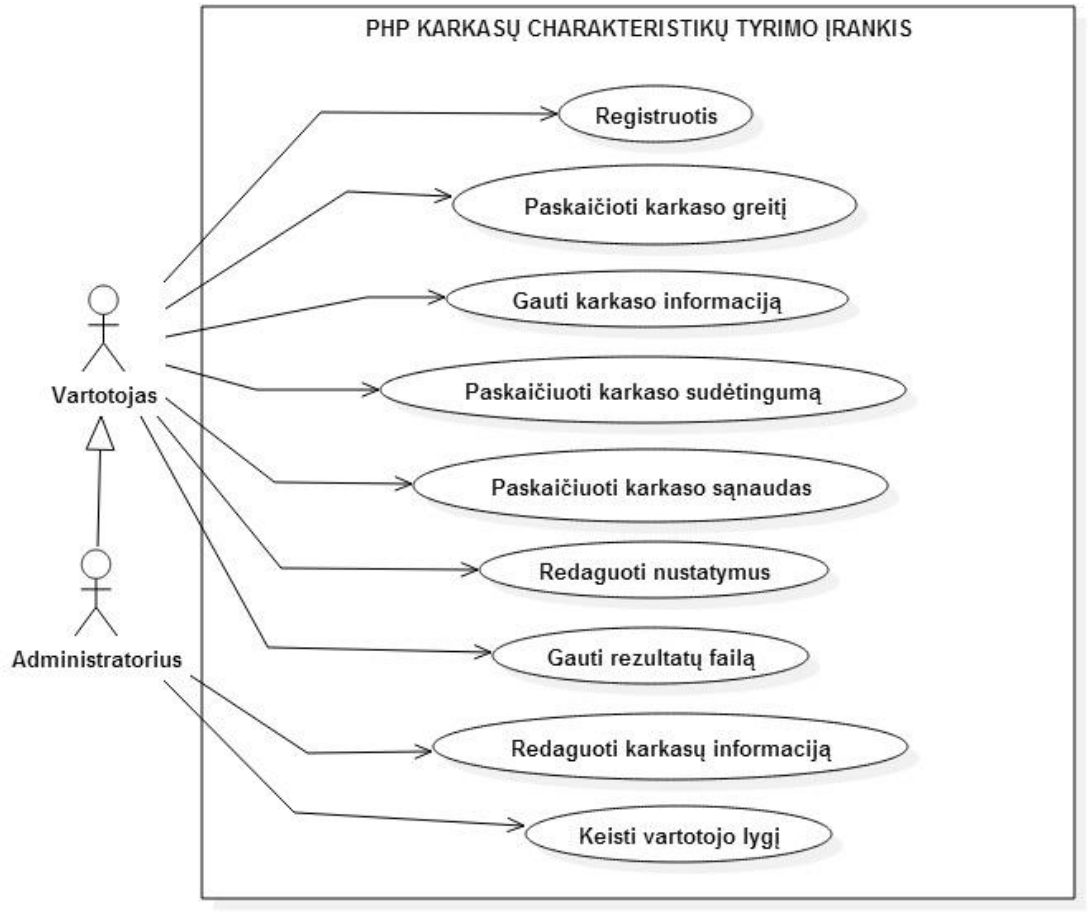
Reikalavimai, kurie turi įtakos sistemos architektūrai:

- ✓ Visa sistema bus realizuota kliento-serverio principu. Kliento pusės kodas bus vykdomas kliento kompiuteryje, likęs kodas – serverio dalyje.
- ✓ Sistemos dalys (pagrindinė sistema ir sistemos kiekvienam *PHP* programavimo kalbos karkasui) bus įkeltos viename serveryje.
- ✓ Duomenų bazė taip pat bus tame pačiame serveryje, tačiau pagrindinė sistema naudosis skirtinga duomenų baze negu likusios, skirtos kiekvienam karkasui, kurios naudos vieną duomenų bazę.
- ✓ Kiekvieno karkaso sistemos dalis turi būti realizuota identiškai, siekiant korektiškai atlikti tyrimus. Pavyzdžiui, jeigu viename karkase yra realizuotas duomenų gavimo ir jų pavaizdavimo lentelėje puslapis, tai kiti karkasai turi turėti tokį pat funkcionalumą su tokiu pačiu pavaizdavimu.
- ✓ Sistema turi suteikti vartotojui naudotis produktu dviem kalbomis (anglų ir lietuvių).
- ✓ Visų funkcijų vykdymas, išskyrus karkasų charakteristikų skaičiavimą, turi būti įvykdytas per 2–4 sekundes.
- ✓ Sistema turi užtikrinti, kad vartotojo duomenų teisės nebus pažeidžiamos ir jie nebus viešinami ar kitaip panaudojami. Šiam tikslui reikia duomenų bazėje saugoti koduotus slaptažodžius.
- ✓ Dingus interneto ryšiui tarpiniai rezultatai turi būti išsaugoti vartotojo kompiuteryje ir atsiradus ryšiui perkeliami į serverį.
- ✓ Sistema turi gebėti aptarnauti 100 vartotojų vienu metu.
- ✓ Sistema turi gebėti dirbti su duomenų baze, turinčia iki pusės milijono įrašų.
- ✓ Sistema turi funkcionuoti nepriklausomai nuo naudojamos platformos.
- ✓ Sistemoje turi būti realizuota apsauga nuo internetinių pažeidžiamumų, tokių kaip *SQL* injekcija, *XSS* ir t.t.

3.3. Panaudojimo atvejų vaizdas

3.3.1. Panaudojimo atvejų modelis

Toliau esančiame panaudojimo atvejų modelyje (žr. 3.3.1.1. pav.) galima matyti du aktorius (vartotojas, administratorius), kurie yra sujungti paveldėjimo ryšiu, o tai reiškia, jog administratorius paveldi visus panaudojimo atvejus, kuriuos naudoja vartotojas.



3.3.1.1 pav. Panaudojimo atvejų modelis

3.3.2. Panaudojimo atvejų scenarijai

Toliau pateiktose lentelėse yra pateikiama kiekvieno panaudojimo atvejo specifikacija su pagrindiniu scenarijumi.

3.3.2.1 lentelė. Panaudojimo atvejo: Registruotis, specifikacija

Panaudos atvejis	Registruotis
Tikslas	Sukurti naują vartotoją sistemoje
Aktoriai	Svečias
Ryšys su kitais PA	-
Nefunkciniai reikalavimai	Patogus ir intuityvus registracijos procesas
Prieš-sąlyga	-
Sužadinimo sąlyga	,,Meniu" mygtuko paspaudimas
Po-sąlyga	Vartotojas yra sukuriamas sistemoje
Pagrindinis scenarijus	Vartotojas užpildo registracijos formą; Paspaudžia ,,registruotis"; Vartotojas sėkmingai priregistruojamas prie sistemos;
Alternatyvus scenarijus	Vartotojas neteisingai užpildo registracijos duomenis; Paspaudžia ,,registruotis"; Naujas vartotojas nesukuriamas; Parodomas klaidos pranešimas.

3.2.2.2 lentelė. Panaudojimo atvejo: Paskaičiuoti karkaso greitį, specifikacija

Panaudos atvejis	Paskaičiuoti karkaso greitį
Tikslas	Atlikti karkasų palyginimą pagal greitį
Aktoriai	Svečias, Vartotojas, Administratorius
Ryšys su kitais PA	-
Nefunkciniai reikalavimai	Greitas veikimas
Prieš-sąlyga	-
Sužadinimo sąlyga	,,Meniu" mygtuko paspaudimas
Po-sąlyga	Vartotojui parodomi karkasų palyginimo pagal greitį rezultatai
Pagrindinis scenarijus	Vartotojas paspaudžia mygtuką; Ekrane pateikiami rezultatai;
Alternatyvus scenarijus	Vartotojas paspaudžia mygtuką; Dingsta interneto ryšis; Vartotojui yra parodomi daliniai rezultatai.

3.3.2.3 lentelė. Panaudojimo atvejo: Gauti karkaso informaciją, specifikacija

Panaudos atvejis	Gauti karkaso informaciją
Tikslas	Pateikti vartotojui išsamią informaciją apie karkasą
Aktoriai	Svečias, Vartotojas, Administratorius
Ryšys su kitais PA	-
Nefunkciniai reikalavimai	Patogus, patrauklus ir lengvai suprantamas karkaso informacijos pateikimas
Prieš-sąlyga	-
Sužadinimo sąlyga	,,Meniu" mygtuko paspaudimas
Po-sąlyga	Vartotojui pateikiama pasirinkto karkaso informacija
Pagrindinis scenarijus	Vartotojas pasirenka karkasą iš sąrašo; Pateikiama karkaso informacija;
Alternatyvus scenarijus	-

3.3.2.4 lentelė. Panaudojimo atvejo: Paskaičiuoti karkaso sudėtingumą, specifikacija

Panaudos atvejis	Paskaičiuoti karkaso sudėtingumą
Tikslas	Atlikti karkasų palyginimą pagal sudėtingumą
Aktoriai	Vartotojas, Administratorius
Ryšys su kitais PA	-
Nefunkciniai reikalavimai	Greitas veikimas
Prieš-sąlyga	Sistemos naudotojas turi būti prisijungęs kaip vartotojas arba administratorius
Sužadinimo sąlyga	„Meniu" mygtuko paspaudimas
Po-sąlyga	Vartotojui parodomi karkasų palyginimo pagal sudėtingumą rezultatai
Pagrindinis scenarijus	Vartotojas paspaudžia mygtuką; Ekrane pateikiami rezultatai;
Alternatyvus scenarijus	Vartotojas paspaudžia mygtuką; Dingsta interneto ryšis; Vartotojui yra parodomi daliniai rezultatai.

3.3.2.5 lentelė. Panaudojimo atvejo: Paskaičiuoti karkaso sąnaudas, specifikacija

Panaudos atvejis	Paskaičiuoti karkaso sąnaudas
Tikslas	Atlikti karkasų palyginimą pagal sąnaudas
Aktoriai	Vartotojas, Administratorius
Ryšys su kitais PA	-
Nefunkciniai reikalavimai	Greitas veikimas
Prieš-sąlyga	Sistemos naudotojas turi būti prisijungęs kaip vartotojas arba administratorius
Sužadinimo sąlyga	„Meniu" mygtuko paspaudimas
Po-sąlyga	Vartotojui parodomi karkasų palyginimo pagal sąnaudas rezultatai
Pagrindinis scenarijus	Vartotojas paspaudžia mygtuką; Ekrane pateikiami rezultatai;
Alternatyvus scenarijus	Vartotojas paspaudžia mygtuką; Dingsta interneto ryšis; Vartotojui yra parodomi daliniai rezultatai.

3.3.2.6 lentelė. Panaudojimo atvejo: Redaguoti nustatymus, specifikacija

Panaudos atvejis	Redaguoti nustatymus
Tikslas	Pakeisti sistemos numatytuosius nustatymus į vartotojo norimus
Aktoriai	Vartotojas, Administratorius
Ryšys su kitais PA	-
Nefunkciniai reikalavimai	-
Prieš-sąlyga	Sistemos naudotojas turi būti prisijungęs kaip vartotojas arba administratorius
Sužadinimo sąlyga	„Meniu" mygtuko paspaudimas
Po-sąlyga	Nauji vartotojo nustatymai yra išsaugomi sistemoje
Pagrindinis scenarijus	Vartotojas pasirenka/įveda nustatymus; Paspaudžia juos išsaugoti; Nustatymai išsaugoti sistemoje;
Alternatyvus scenarijus	Vartotojas įveda klaidingus duomenis; Paspaudžia juos išsaugoti; Nauji nustatymai neišsaugomi; Parodomas klaidos pranešimas.

3.3.2.7 lentelė. Panaudojimo atvejo: Gauti rezultatų failą, specifikacija

Panaudos atvejis	Gauti rezultatų failą
Tikslas	Sugeneruoti rezultatų failą ir leisti jį parsisiųsti
Aktoriai	Vartotojas, Administratorius
Ryšys su kitais PA	-
Nefunkciniai reikalavimai	-
Prieš-sąlyga	Sistemos naudotojas turi būti prisijungęs kaip vartotojas arba administratorius. Turi būti atlikta viena iš karkasų palyginimo funkcijų.
Sužadinimo sąlyga	Mygtuko paspaudimas
Po-sąlyga	Vartotojas gauna rezultatus faile
Pagrindinis scenarijus	Vartotojas pasirenka vieną iš funkcijų karkasų palyginimui atlikti; Sistema atlieka skaičiavimus; Vartotojas pasirenka gauti rezultatų failą; Sugeneruojamas failas.
Alternatyvus scenarijus	-

3.3.2.8 lentelė. Panaudojimo atvejo: Redaguoti karkasų informaciją, specifikacija

Panaudos atvejis	Redaguoti karkasų informaciją
Tikslas	Pridėti, keisti ir trinti karkasų informaciją
Aktoriai	Administratorius
Ryšys su kitais PA	-
Nefunkciniai reikalavimai	-
Prieš-sąlyga	Sistemos naudotojas turi būti prisijungęs kaip administratorius
Sužadinimo sąlyga	Meniu mygtuko paspaudimas
Po-sąlyga	Sistemoje redaguojama karkasų informacija
Pagrindinis scenarijus	Administratorius pasirenka karkasų informaciją; Pasirenka pridėti naują karkaso savybę; Įveda savybės pavadinimą ir reikšmę; Paspaudžia išsaugoti; Nauja karkaso savybė yra sukuriama sistemoje.
Alternatyvus scenarijus	-

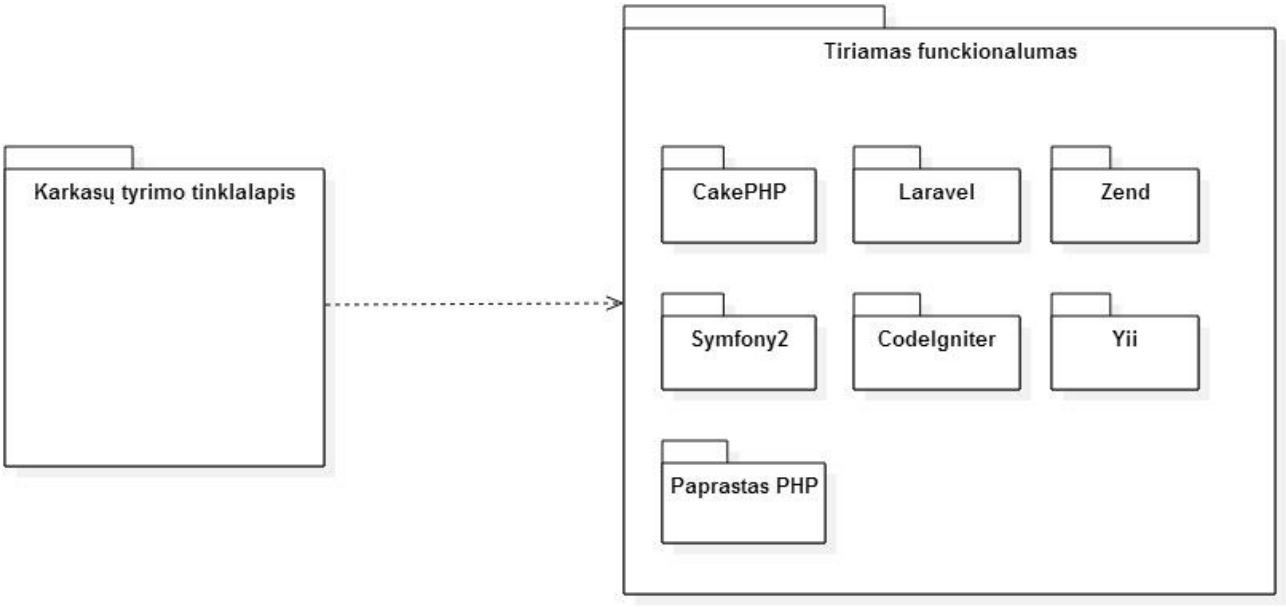
3.3.2.9 lentelė. Panaudojimo atvejo: Keisti vartotojo lygį, specifikacija

Panaudos atvejis	Keisti vartotojo lygį
Tikslas	Pakeisti vartotojo lygį iš administratoriaus į vartotoją ir atvirkščiai
Aktoriai	Administratorius
Ryšys su kitais PA	-
Nefunkciniai reikalavimai	-
Prieš-sąlyga	Sistemos naudotojas turi būti prisijungęs kaip administratorius
Sužadinimo sąlyga	„Meniu" mygtuko paspaudimas
Po-sąlyga	Sistemoje redaguojamas vartotojo lygis
Pagrindinis scenarijus	Administratorius pasirenka redaguoti vartotojus; Paspaudžia ant norimo vartotojo eilutės; Parenka naują lygį vartotojui; Paspaudžia „išsaugoti"; Vartotojo lygis yra pakeičiamas.
Alternatyvus scenarijus	Administratorius bando keisti savo lygį; Parodomas klaidos pranešimas.

3.4. Sistemos statinis vaizdas

3.4.1. Sistemos paketai

Sistemą sudaro du pagrindiniai paketai (žr. 3.4.1.1. pav.). Pirmasis yra visos sistemos tinklalapis, kuriame vartotojas atlieka tam tikrus veiksmus. Šis paketas siekdamas atlikti visus reikalingus veiksmus naudoja kitą paketą – tiriamo funkcionalumo. Jis sudarytas iš standartinių funkcijų, kurios yra taikomos visiems tiriamiems *PHP* programavimo kalbos karkasams. Taigi visi karkasai, kaip paketai, yra patalpinti tiriamo funkcionalumo pakete.



3.4.1.1 pav. Sistemos išskaidymas į paketus

3.4.2. Karkasų tyrimo tinklalapio komponentas

Klasifikacija

Paketas.

Apibrėžimas

Karkasų tyrimo tinklalapio komponentas yra pagrindinė sistemos svetainė, kuri patalpinta serveryje. Jos pagrindinis tikslas – suteikti vartotojui funkcijas, kurių pagalba jis galėtų atlikti *PHP* programavimo kalbos karkasų charakteristikų skaičiavimus.

Atsakomybės

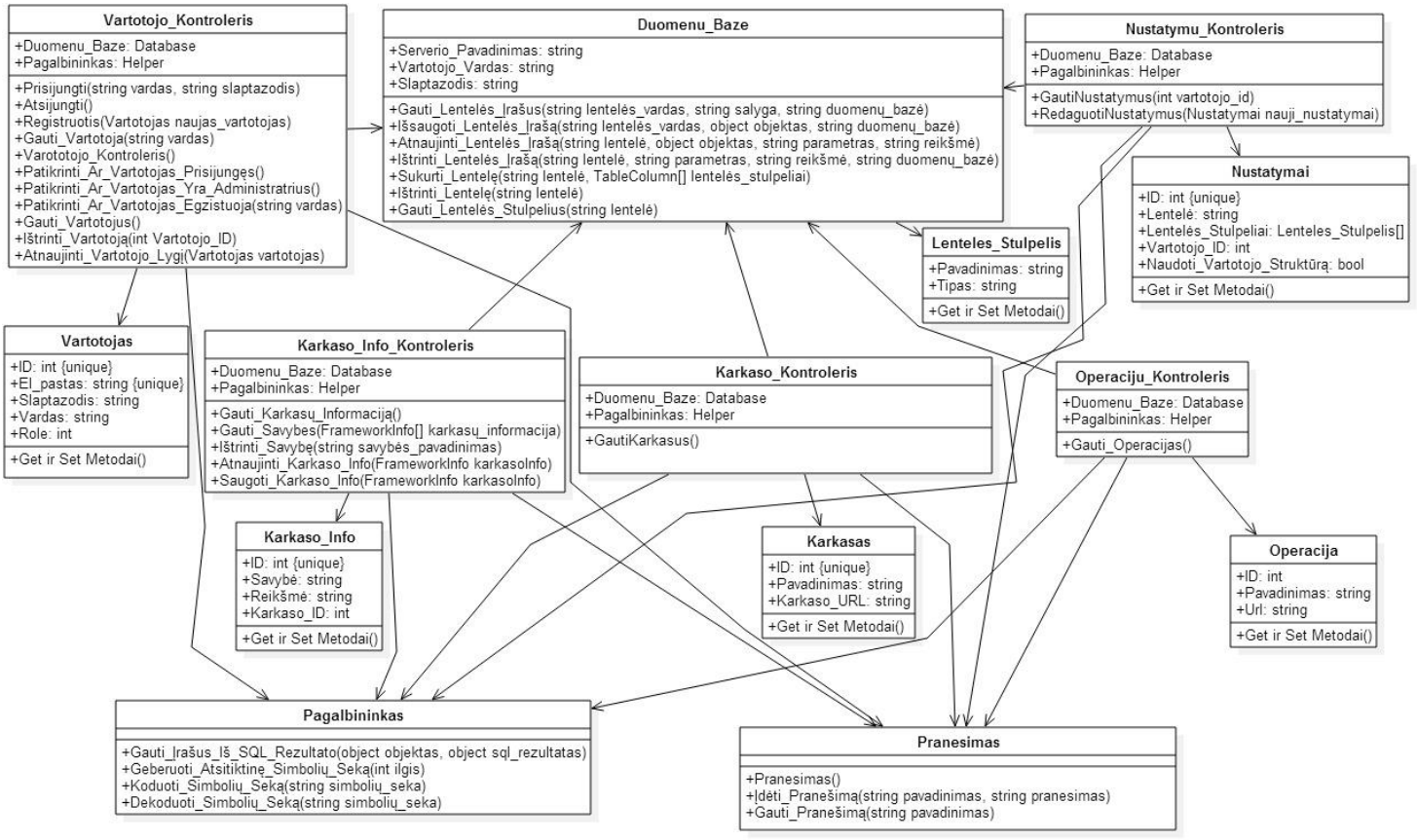
Šis komponentas yra atsakingas už vartotojo sąsajos pateikimą bei visų tinklalapyje esančių funkcijų realizavimą. Tai apima vartotojo prisijungimą, karkasų charakteristikų skaičiavimą, rezultatų išsaugojimą, nustatymų keitimą ir pan. Šis komponentas taip pat yra atsakingas už patogų ir teisingą informacijos pateikimą vartotojui.

Apribojimai

- Šis komponentas turi sąveikauti su tiriamo funkcionalumo komponentu.
- Šis komponentas turi būti prieinamas visą parą.
- Šiame komponente esantys puslapiai turi užsikrauti 2-4 sekundžių intervalu.

Struktūra

Karkasų tyrimo tinklalapio paketo klasių diagramą sudaro šešios pagrindinės klasės (žr. 3.4.2.1. pav.), kurios atlieka visą sistemos funkcionalumą. Vartotojo kontroleris yra atsakingas už vartotojo veiksmus, tokius kaip prisijungimas, registracija ir pan. Nustatymų kontroleris apibrėžia vartotojo nustatymus ir metodus, skirtus gauti esamus nustatymus bei redaguoti juos. Karkaso kontroleris yra skirtas pagrindiniams veiksmams su karkasais atlikti. Karkaso informacijos kontroleris yra atsakingas už karkaso savybių redagavimą. Operacijų kontroleris yra atsakingas už operacijų gavimą. Bendravimui su duomenų baze yra skirta duomenų bazės klasė. Kitos klasės reikalingos papildomam funkcionalumui, tokiam kaip pranešimų valdymas (pranešimo klasė), ar pagalbinėms funkcijoms kitoms klasėms (pagalbininko klasė), likusios klasės yra skirtos apibrėžti naudojamų duomenų struktūrą.



3.4.2.1 pav. Paketo: karkasų tyrimo tinklalapis klasių diagrama

Karkasų tyrimo tinklalapio klasės

Karkaso kontrolerio klasė

3.4.2.1 lentelė. Karkaso kontrolerio klasės specifikacija

Komponento pavadinimas	*Karkaso_Kontroleris*
Klasifikacija	Klasė
Apibrėžimas	Klasė, kurios tikslas yra atlikti veiksmus su karkasais.
Atsakomybės	Atsakingas už karkasų gavimą.
Apribojimai	-
Sąveikavimas	Šis komponentas sąveikauja su duomenų bazės, karkaso, pagalbininko ir pranešimų komponentais.
Resursai	Komponentas naudoja duomenų bazės resursus.
Skaičiavimai	-
Sąsaja	Šis komponentas yra sužadinamas tiesiogiai tinklalapio komponento.

Karkaso kontrolerio komponento struktūra

3.4.2.2 lentelė. Karkaso kontrolerio komponento – *Gauti_Karkasus* specifikacija

Komponento pavadinimas	*Gauti_Karkasus*
Klasifikacija	Metodas
Apibrėžimas	Metodas, kurio tikslas gauti visus sistemoje esančius karkasus.
Apribojimai	Šiame komponente vykdomas karkasų gavimas negali trukti ilgiau nei 3 sekundes.
Resursai	Komponentas naudoja duomenų bazės resursus.
Skaičiavimai	-

Vartotojo kontrolerio klasė

3.4.2.3 lentelė. Vartotojo kontrolerio klasės specifikacija

Komponento pavadinimas	*Vartotojo_Kontroleris*
Klasifikacija	Klasė
Apibrėžimas	Klasė, kurios tikslas yra atlikti veiksmus su vartotoju.
Atsakomybės	Atsakingas už vartotojo prisijungimą, registraciją bei atsijungimą iš sistemos. Taip atsakingas už prisijungusio vartotojo gavimą.
Apribojimai	Šis komponentas turi užtikrinti vartotojo duomenų saugumą bei neleisti jų nutekinti trečiosioms šalims.
Sąveikavimas	Šis komponentas sąveikauja su duomenų bazės, vartotojo, pagalbininko ir pranešimų komponentais.
Resursai	Komponentas naudoja duomenų bazės resursus bei vartotojo sesiją.
Skaičiavimai	Šiame komponente yra atliekami skaičiavimai, susiję su vartotojo duomenimis (pavyzdžiui, slaptažodžio kodavimas).
Sąsaja	Šis komponentas yra sužadinamas tiesiogiai tinklalapio komponento.

Vartotojo kontrolerio komponento struktūra

3.4.2.4 lentelė. Vartotojo kontrolerio komponento – *Prisijungti* specifikacija

Komponento pavadinimas	*Prisijungti*
Klasifikacija	Metodas
Apibrėžimas	Metodas, kurio tikslas yra atlikti vartotojo prisijungimą prie sistemos.
Apribojimai	Vartotojo duomenys turi būti saugomi sesijoje, taip pat sesija turi būti apsaugota nuo jos pasisavinimo.
Resursai	Komponentas naudoja duomenų bazės resursus bei vartotojo sesiją.
Skaičiavimai	-

3.4.2.5 lentelė. Vartotojo kontrolerio komponento – *Registruotis* specifikacija

Komponento pavadinimas	*Registruotis*
Klasifikacija	Metodas
Apibrėžimas	Metodas, kurio tikslas yra pridėti naują vartotoją į sistemą.
Apribojimai	Šis veiksmas neturi trukti ilgiau nei 4 sekundės. Vartotojo slaptažodis turi būti koduotas.
Resursai	Komponentas naudoja duomenų bazės resursus.
Skaičiavimai	Pagrindiniai skaičiavimai yra skirti slaptažodžio kodavimo funkcijai, kuri užtikrintų, kad slaptažodžiui atkoduoti neužtektų vieno algoritmo.

Nustatymų kontrolerio klasė

3.4.2.6 lentelė. Nustatymų kontrolerio klasės specifikacija

Komponento pavadinimas	*Nustatymu_Kontroleris*
Klasifikacija	Klasė
Apibrėžimas	Klasė, kurios tikslas yra atlikti veiksmus su vartotojo nustatymais.
Atsakomybės	Atsakingas už vartotojo nustatymų gavimą iš duomenų bazės bei jų atnaujinimą.
Apribojimai	Vartotojo nustatymai turi būti išsaugoti 2-4 sekundžių intervale.
Sąveikavimas	Šis komponentas sąveikauja su duomenų bazės, nustatymų, pagalbininko ir pranešimų komponentais.
Resursai	Komponentas naudoja duomenų bazės resursus.
Skaičiavimai	Šiame komponente specifinių skaičiavimų nėra atliekama.
Sąsaja	Šis komponentas yra sužadinamas, tinklalapio komponento tiesiogiai.

Nustatymų kontrolerio komponento struktūra

3.4.2.7 lentelė. Nustatymų kontrolerio komponento – *Redaguoti_Nustatymus* specifikacija

Komponento pavadinimas	*Redaguoti_Nustatymus*
Klasifikacija	Metodas
Apibrėžimas	Metodas, kurio tikslas yra atnaujinti vartotojo nustatymus.
Apribojimai	-
Resursai	Komponentas naudoja duomenų bazės resursus.
Skaičiavimai	-

Karkaso informacijos kontrolerio klasė

3.4.2.8 lentelė. Karkaso informacijos kontrolerio klasės specifikacija

Komponento pavadinimas	*Karkaso_Info_Kontroleris*
Klasifikacija	Klasė
Apibrėžimas	Klasė, kurios tikslas atlikti veiksmus su karkasų savybėmis.
Atsakomybės	Atsakingas už PHP karkasų savybių gavimą bei jų redagavimą.
Apribojimai	Šiam komponentui nėra taikoma jokių apribojimų.
Sąveikavimas	Šis komponentas sąveikauja su duomenų bazės, karkaso informacijos, pagalbininko ir pranešimų komponentais.
Resursai	Komponentas naudoja duomenų bazės resursus.
Skaičiavimai	-
Sąsaja	Šis komponentas yra sužadinamas tiesiogiai tinklalapio komponento.

Karkaso informacijos kontrolerio komponento struktūra

3.4.2.9 lentelė. Karkaso informacijos kontrolerio komponento – *Saugoti_Karkasu_Info* specifikacija

Komponento pavadinimas	*Saugoti_Karkasu_Info*
Klasifikacija	Metodas
Apibrėžimas	Metodas, kurio tikslas yra išsaugoti visų karkasų reikšmes pagal vieną savybę.
Apribojimai	Saugojimas neturi viršyti 4 sekundžių.
Resursai	Komponentas naudoja duomenų bazės resursus.
Skaičiavimai	-

Operacijų kontrolerio klasė

3.4.2.10 lentelė. Operacijų kontrolerio klasės specifikacija

Komponento pavadinimas	*Operaciju_Kontroleris*
Klasifikacija	Klasė
Apibrėžimas	Klasė, kurios tikslas gauti visas sistemoje esančias operacijas.
Atsakomybės	Atsakingas už operacijų gavimą.
Apribojimai	Šiam komponentui nėra taikoma jokių apribojimų.
Sąveikavimas	Šis komponentas sąveikauja su duomenų bazės, operacijos, pagalbininko ir pranešimų komponentais.
Resursai	Komponentas naudoja duomenų bazės resursus.
Skaičiavimai	-
Sąsaja	Šis komponentas yra sužadinamas tiesiogiai tinklalapio komponento.

Operacijų kontrolerio komponento struktūra

3.4.2.11 lentelė. Operacijų kontrolerio komponento – *Gauti_Operacijas* specifikacija

Komponento pavadinimas	*Gauti_Operacijas*
Klasifikacija	Metodas
Apibrėžimas	Metodas, kurio tikslas yra gauti visas sistemoje esančias operacijas
Apribojimai	Saugojimas neturi viršyti 2 sekundžių.
Resursai	Komponentas naudoja duomenų bazės resursus.
Skaičiavimai	-

Duomenų bazės klasė

3.4.2.12 lentelė. Duomenų bazės kontrolerio klasės specifikacija

Komponento pavadinimas	*Duomenu_Baze*
Klasifikacija	Klasė
Apibrėžimas	Klasė, kurios tikslas atlikti veiksmus su duomenų baze.
Atsakomybės	Atsakingas už duomenų gavimą, atnaujinimą, saugojimą, trynimą duomenų bazėje, taip pat redaguoti lentelių struktūrą, kurti naujas lenteles bei jas trinti.
Apribojimai	Šiam komponentui nėra taikoma jokių apribojimų.
Sąveikavimas	Šis komponentas sąveikauja lentelės stulpelio komponentu.
Resursai	Komponentas naudoja duomenų bazės resursus.
Skaičiavimai	-
Sąsaja	Šis komponentas yra sužadinamas kitų sistemos komponentų.

Duomenų bazės komponento struktūra

3.4.2.13 lentelė. Duomenų bazės kontrolerio komponento – *Gauti_Lentelės_Irasus* specifikacija

Komponento pavadinimas	*Gauti_Lenteles_Irasus*
Klasifikacija	Metodas
Apibrėžimas	Metodas, kurio tikslas gauti nurodytos lentelės įrašus iš duomenų bazės
Apribojimai	-
Resursai	Komponentas naudoja duomenų bazės resursus.
Skaičiavimai	Metodui gali būti perduota sąlyga, pagal kurią yra atliekamas filtravimas (pvz. *Where* sąlyga), šiame metode yra sukurta dinamiškos užklausos kūrimo logika.

3.4.2.14 lentelė. Duomenų bazės kontrolerio komponento – *Sukurti_Lentele* specifikacija

Komponento pavadinimas	*Sukurti_Lentele*
Klasifikacija	Metodas
Apibrėžimas	Metodas, kurio tikslas yra sukurti duomenų lentelę kartu su jos stulpeliais.
Apribojimai	-
Resursai	Komponentas naudoja duomenų bazės resursus.
Skaičiavimai	-

3.4.2.15 lentelė. Duomenų bazės kontrolerio komponento – *Gauti_Lentelės_Stulpelius* specifikacija

Komponento pavadinimas	*Gauti_Lenteles_Stulpelius*
Klasifikacija	Metodas
Apibrėžimas	Metodas, kurio tikslas gauti nurodytos lentelės stulpelius bei jų tipus.
Apribojimai	Stulpelių gavimas neturi viršyti 2 sekundžių.
Resursai	Komponentas naudoja duomenų bazės resursus.
Skaičiavimai	-

Likę komponentai: *Nustatymai, Vartotojas, Karkasas, Karkaso_Info, Operacija, Lenteles _Stulpelis*. Yra skirti apibrėžti duomenų struktūromis, dėl to jie susideda tik iš atributų bei *Get()* ir *Set()* metodų. *Pagalbininko* komponentas yra skirtas atlikti pagalbinėms sistemos funkcijoms, tokiomis kaip slaptažodžio kodavimas. *Pranešimų* komponentas yra atsakingas už pranešimų, kurie rodomi vartotojui, tvarkymą. Jis naudoja vartotojo sesiją veiksmams su pranešimais atlikti.

SĄVEIKAVIMAS

Karkasų tyrimo tinklalapio komponentas sąveikauja su funkcionalumo komponentu, iš kurio gauna visą informaciją apie *PHP* programavimo kalbos karkasų charakteristikas. Šie komponentai bendrauja per *HTTP* protokolą, t. y., tinklalapio komponentas siunčia užklausas karkasų funkcionalumo komponentui ir gautus rezultatus pateikia vartotojui.

RESURSAI

Karkasų tinklalapių komponentas yra patalpintas serveryje, jo puslapiai yra saugomi *.php* failo formatu. Šis komponentas naudoja duomenų resursus (duomenų bazę), kad galėtų įgyvendinti visas specifikacijoje aprašytas funkcijas.

SKAIČIAVIMAI

Komponento būsena tampa aktyvi, kai vartotojas interneto naršyklėje įveda svetainės adresą. Šiame komponente yra gaunami rezultatai iš funkcionalumo komponento. Po informacijos gavimo vyksta rezultatų apdorojimas, t.y., surenkami visų karkasų rezultatai pagal tam tikrą charakteristiką ir atliekamas jų matematinis palyginimas bei rezultatai yra pateikiami vartotojui tiek tinklalapyje, tiek faile. Rezultatų analizei yra naudojamas matematinis principas. Pavyzdžiui, gaunami karkaso funkcijos užkrovimo rezultatai, jie yra grąžinami skaitine forma. Šis komponentas gautą skaitinę formą apdoroja taip, kad vartotojui būtų aiškūs rezultatai. Tokiu atveju visų funkcijų įvykdymo greitis yra pavaizduojamas grafike. Naudodamas palyginimo operatorius atlieka charakteristikų palyginimą.

SĄSAJA

Interneto svetainėje įvesti duomenys *HTTP POST* metodu yra perduodami komponentui tolimesniam apdorojimui. Pavyzdžiui, vartotojas įveda prisijungimo informaciją (vartotojo vardą ir slaptažodį), kai vartotojas pateikia formą paspausdamas mygtuką. Formos duomenys *HTTP POST* metodu yra perduodami komponentui ir šis patikrina, ar toks vartotojas yra duomenų bazėje ir ar vartotojo slaptažodis yra teisingas. Vartotojo sąsają taip pat sudaro vartotojo registracijos, nustatymų pakeitimų, karkasų charakteristikų skaičiavimo ir t.t.

3.4.3. Tiriamo funkcionalumo komponentas

KLASIFIKACIJA

Paketas.

APIBRĖŽIMAS

Tiriamo funkcionalumo komponentas yra tarsi pagrindinio komponento dalis, be kurios pastarasis negalėtų atlikti *PHP* programavimo kalbos karkasų charakteristikų skaičiavimo. Šis komponentas taip pat yra patalpintas serveryje, jo pagrindinis tikslas yra įgyvendinti įvairiausias funkcijas (manipuliacija su duomenų bazėje esančiais duomenimis, vartotojo prisijungimas, kompleksinių funkcijų skaičiavimas, vartotojo sukurtų funkcijų skaičiavimas) tiriamuose karkasuose bei pateikti funkcijų vykdymo charakteristikas.

ATSAKOMYBĖS

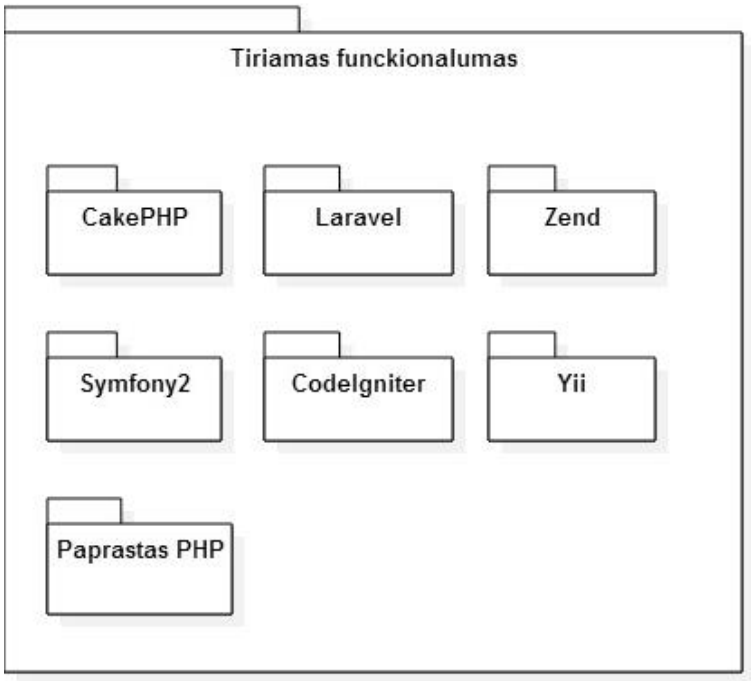
Šis komponentas yra atsakingas už karkasų charakteristikų analizę, t.y., skirtingų funkcijų vykdymą skirtinguose karkasuose ir rezultatų gavimą priklausomai nuo pasirinktos charakteristikos.

APRIBOJIMAI

- Šis komponentas turi sąveikauti su karkasų tyrimo tinklalapio komponentu.
- Šis komponentas turi būti prieinamas visą parą, kadangi sutrikus jo veiklai sutriktų ir pagrindinio komponento veikla.

STRUKTŪRA

Tiriamo funkcionalumo komponentas yra sudarytas iš kelių paketų (žr. 3.4.3.1 pav.). Kiekvienas paketas reiškia skirtingą *PHP* programavimo kalbos karkasą. Taip pat yra vienas paketas, skirtas tirti funkcijų vykdymo charakteristikas nenaudojant jokio karkaso. Visų šių paketų veikimo principas yra toks pats (kiekviename pakete yra realizuotos tos pačios funkcijos, pavyzdžiui, duomenų gavimo iš duomenų bazės, funkcijų rezultatai tokiu pačiu būdu yra skaičiuojami visuose paketuose. Skiriasi tik funkcijų realizavimas priklausomai nuo karkaso turimų priemonių), todėl toliau nagrinėsime vieną iš jų.



3.4.3.1 pav. Tiriamo funkcionalumo komponento struktūra

Karkaso komponentas

Klasifikacija

Paketas

Apibrėžimas

Bet kurio karkaso komponentas yra tiriamo funkcionalumo komponento dalis, kurios tikslas yra realizuoti apibrėžtas funkcijas konkrečiame karkase.

Atsakomybės

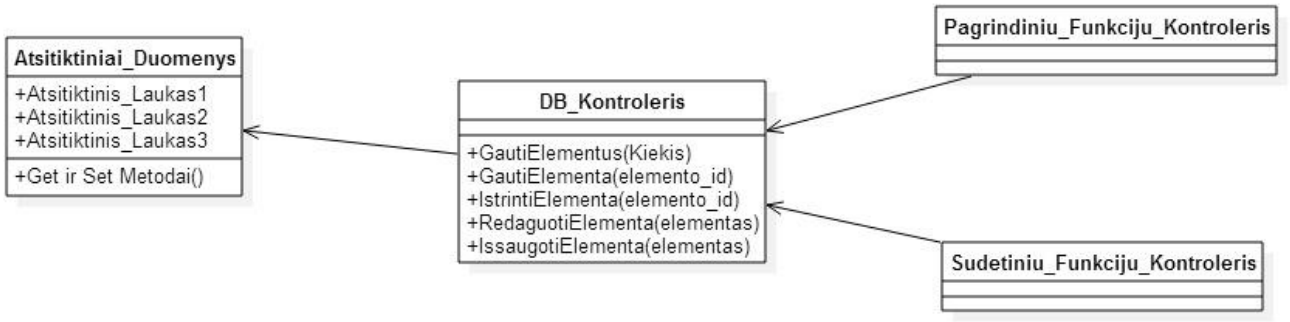
Šis komponentas yra atsakingas už pateiktų užklausų vykdymą ir rezultatų grąžinimą.

Apribojimai

Komponentas turi realizuoti tokias pačias funkcijas kaip ir kiti to paties lygio komponentai, kaip įmanoma labiau užtikrinant objektyvius rezultatus.

Struktūra

Karkaso komponento struktūra yra pavaizduota klasių diagramoje (žr. 3.4.3.2 pav.), kuri yra sudaryta iš keturių klasių. Pirmoji (*Atsitiktiniai_Duomenys*) skirta apibrėžti duomenų tipui, kuris bus naudojamas šiame pakete. Tai bus atsitiktiniai duomenys, t.y. jų tipas neturės didelės įtakos bendram sistemos veikimui. Pagrindinė klasė (*DB_Kontroleris*) šiame pakete yra atsakinga už veiksmus su duomenų baze, t.y. duomenų manipuliavimu. Likusios klasės yra atsakingos už pagrindinių ir sudėtinių funkcijų vykdymą. Pagrindinės funkcijos galėtų būti naujo elemento sukūrimas ir jo pateikimas. Sudėtinės funkcijos: duomenų atvaizdavimas, jų išrikiavimas ir atsitiktinės reikšmės ištrynimas.



3.4.3.2 pav. Karkaso komponento klasių diagrama

Karkasų komponento klasės

3.4.3.1 lentelė. Duomenų bazės kontrolerio klasės specifikacija

Komponento pavadinimas	*DB_Kontroleris*
Klasifikacija	Klasė
Apibrėžimas	Klasė, kurios tikslas yra atlikti veiksmus su duomenų baze.
Atsakomybės	Atsakingas už duomenų, esančių duomenų bazėje, manipuliaciją (duomenų gavimas, trynimas, redagavimas).
Apribojimai	Turi gebėti apdoroti didelės apimties duomenis.
Sąveikavimas	Šis komponentas sąveikauja su *Atsitiktiniu_Duomenu*, *Pagrindiu_Funkciju_Kontrolerio* ir *Sudetiniu_Funkciju_Kontrolerio* komponentais.
Resursai	Komponentas naudoja duomenų bazės resursus.
Skaičiavimai	Šiame komponente skaičiavimai nėra atliekami, jis atlieka tarpininko tarp duomenų bazės ir funkcijų vaidmenį.
Sąsaja	Šis komponentas yra sužadinamas tiesiogiai karkaso komponento.

3.4.3.2 lentelė. Pagrindinių funkcijų kontrolerio klasės specifikacija

Komponento pavadinimas	*Pagrindiniu_Funkciju_Kontroleris*
Klasifikacija	Klasė
Apibrėžimas	Klasė, kurios tikslas yra atlikti veiksmus, susijusius su pagrindinėmis funkcijomis.
Atsakomybės	Atsakingas už pagrindinių funkcijų iškvietimą ir tų funkcijų charakteristikų apskaičiavimą.
Apribojimai	-
Struktūra	-
Sąveikavimas	Šis komponentas sąveikauja su *DB_kontrolerio* komponentu.
Resursai	Komponentas naudoja duomenų bazės resursus.
Skaičiavimai	Šiame komponente yra atliekami skaičiavimai, susiję su funkcijos greičio, sąnaudų ir sudėtingumo skaičiavimais.
Sąsaja	Šis komponentas yra sužadinamas tiesiogiai karkaso komponento.

Sudėtinių_Funkciju_Kontrolerio komponentas yra labai panašus į *Pagrindinių_Funkciju_Kontrolerio* komponentą, skiriasi tik tuo, jog jame yra vykdomos sudėtinės funkcijos ir yra atliekamas jų charakteristikų skaičiavimas. Paskutinis komponentas *Atsistiktiniai_Duomenys* yra tik struktūra, skirta apibrėžti duomenims, kuriuos naudos karkaso komponentas. Iš esmės nėra svarbu, kiek ir kokių atributų turės šis komponentas, kadangi tyrimas koncentruojasi į charakteristikų apskaičiavimą naudojant įvairiausius duomenis.

Sąveikavimas

Karkaso komponentas sąveikauja tik su tėviniu savo komponentu (tiriamo funkcionalumo komponentas), kuris valdo visus jame esančius karkasų komponentus.

Resursai

Šis komponentas naudoja duomenų bazės resursus.

Skaičiavimai

Šiame komponente yra atliekami skaičiavimai, susiję su jo greičio, sąnaudų ir sudėtingumo charakteristikų skaičiavimu.

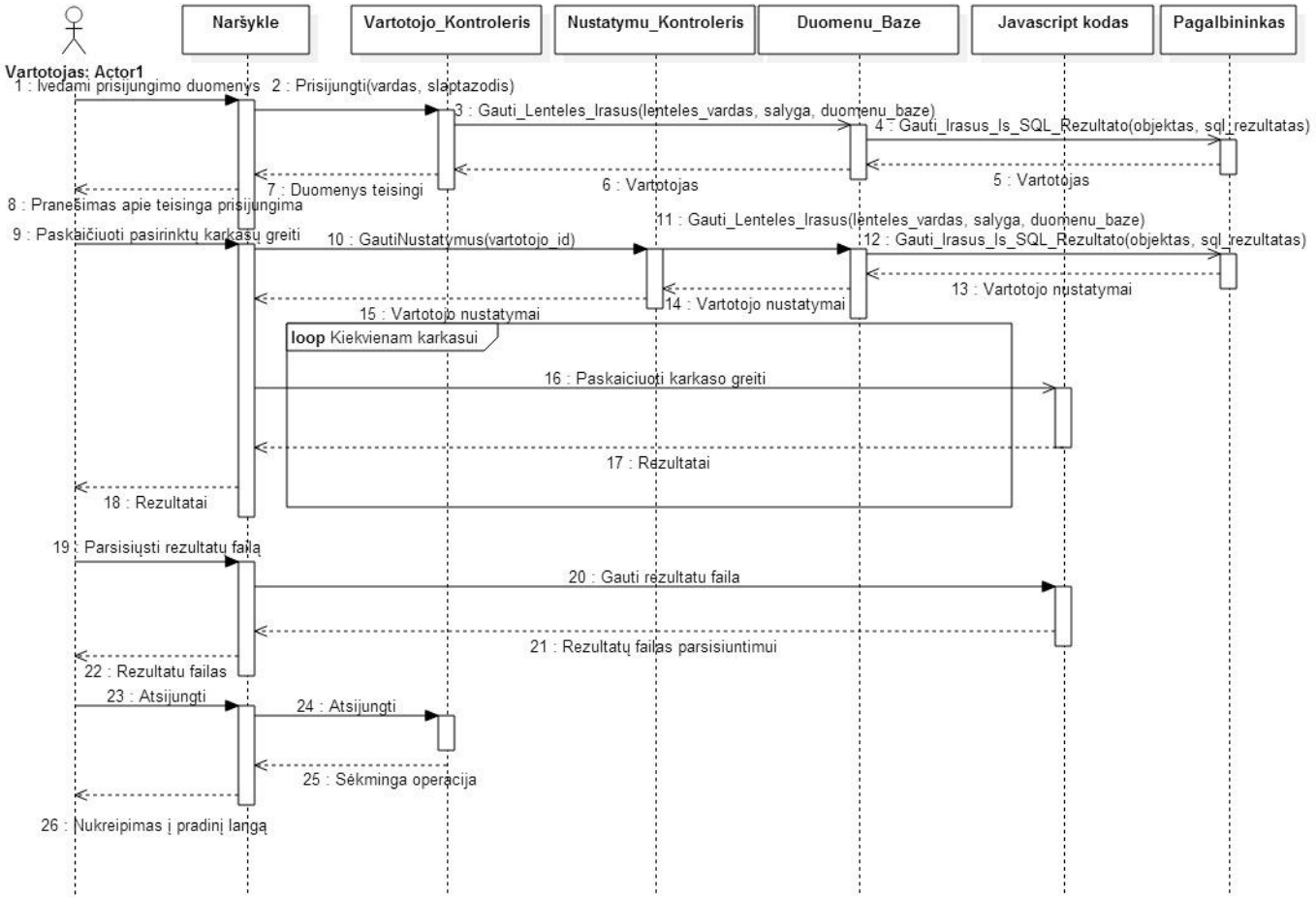
Sąsaja

Tiriamo funkcionalumo komponentas perduoda informaciją (kokia charakteristika bus tiriama, kokia duomenų apimtis, kokią funkciją naudoti ir pan.) karkaso komponentui, kuris jau apdoroja užklausą bei pateikia rezultatus. Šiame komponente nėra jokios vartotojo sąsajos, kadangi vartotojas tiesiogiai komponento pasiekti negali.

3.5. Sistemos dinaminis vaizdas

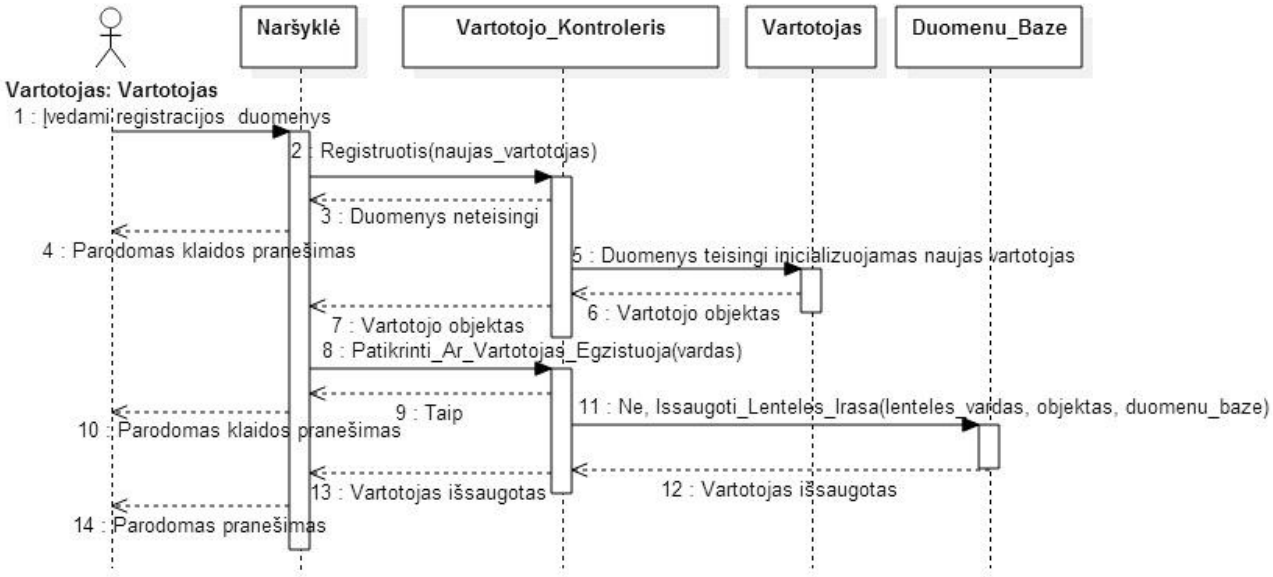
3.5.1. Sekų diagramos

Toliau esančiame paveikslėlyje yra pavaizduota sekų diagrama (žr. 3.5.1.1. pav.), kurioje yra realizuota standartinė šios sistemos vartotojo veiksmų seka. Pirmiausiai suvesdamas svetainės adresą vartotojas yra nukreipiamas į pradinį langą, kuris prašo vartotoją prisijungti arba prisiregistruoti. Vartotojui pateikus duomenis ir paspaudus ,,prisijungti" (1 žingsnis) yra iškviečiamas vartotojo kontrolerio metodas (2 žingsnis), kuris naudodamas duomenų bazės objektą iškviečia duomenų gavimo metodą, kuriame nurodo vartotojų lentelę ir vartotojo vardą kaip sąlygą (3 žingsnis). Šis, gavęs teigiamą atsakymą iš duomenų bazės, kreipiasi į pagalbinę klasę, kuri apdoroja atsakymą (4 žingsnis) ir grąžina vartotoją duomenų bazės klasei (5 žingsnis), o ši vartotojo kontroleriui (6 žingsnis). Tada vartotojo kontroleris patikrina slaptažodžius ir grąžina atsakymą vartotojui (7 žingsnis). Apie teisingą/klaidingą prisijungimą vartotojas yra informuojamas pranešimu (8 žingsnis). Kitame žingsnyje vartotojas gali pasirinkti, kokius karkasus lygins bei pagal kokią charakteristiką. Šiuo atveju vartotojas atlieka karkasų greičio matavimą (9 žingsnis). Pirmiausia yra gaunami vartotojo nustatymai iškviečiant nustatymų kontrolerį (10 žingsnis), šis kreipiasi į duomenų bazės klasę pateikdamas vartotojų nustatymų lentelės pavadinimą bei vartotojo numerį (11 žingsnis), pastaroji gavusi atsakymą kreipiasi į pagalbinę klasę, kuri apdoroja atsakymą bei grąžina vartotojo nustatymus (12, 13 žingsniai), vėliau jie yra perduodami nustatymų kontroleriui, galiausiai naršyklei (14, 15 žingsniai). Gavus vartotojo nustatymus kliento naršyklėje *JavaScript* kodas atlieka visus reikalingus skaičiavimus (16 žingsnis) bei grąžina vartotojui gautus rezultatus (17, 18 žingsniai).Vartotojas paspaudžia ,,parsisiųsti" rezultatų failą (19 žingsnis), tuo metu yra šaukiamas *JavaScript* kodas, kuris iškviečia spausdinimo langą (20 žingsnis), kuriame vartotojas parsisiunčia failą (21, 22 žingsniai). Galiausiai vartotojas atsijungia (23 žingsnis) iš sistemos naudodamas vartotojo kontrolerį (24, 25 žingsniai) ir yra nukreipiamas į pradinį puslapį (26 žingsnis).



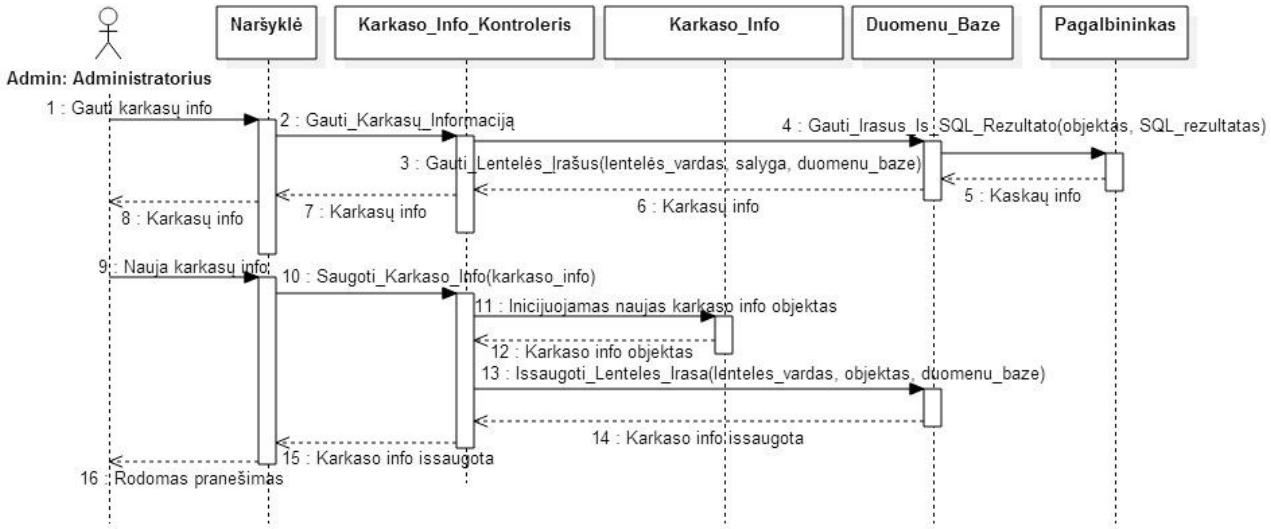
3.5.1.1 pav. *PHP* karkasų charakteristikų palyginimo sekų diagrama

Panaudojimo atvejo registruotis sekų diagrama yra pavaizduota paveikslėlyje (žr. 3.5.1.2 pav.). Pirmiausiai vartotojas norėdamas prisiregistruoti prie sistemos turi pateikti registracijai reikalingus duomenis (elektroninį paštą, slaptažodį, vardą, pavardę ir pan.) (1 žingsnis). Duomenys keliauja į vartotojo kontrolerį, kur yra patikrinami (2 žingsnis). Jeigu duomenys yra klaidingi, pavyzdžiui, vartotojo vardas per trumpas, tokiu atveju vartotojui parodomas klaidos pranešimas (3, 4 žingsniai). Jeigu duomenys teisingi, naujo vartotojo objektas yra sukuriamas ir grąžinamas vartotojo kontroleriui (5, 6, 7 žingsniai). Tuomet yra tikrinama, ar nurodytu vartotojo vardu nėra sukurtas kitas vartotojas (8 žingsnis), jeigu yra, vartotojui yra parodomas pranešimas (9, 10 žingsniai). Jeigu tokio vartotojo nėra, kontroleris kreipiasi į duomenų bazės klasę paduodamas naujo vartotojo objektą (11 žingsnis). Išsaugojus vartotoją duomenų bazėje apie sėkmingai atliktą operaciją vartotojas yra informuojamas pranešimu (12, 13, 14 žingsniai).



3.5.1.2 pav. Sekų diagrama panaudojimo atvejui: Registruotis

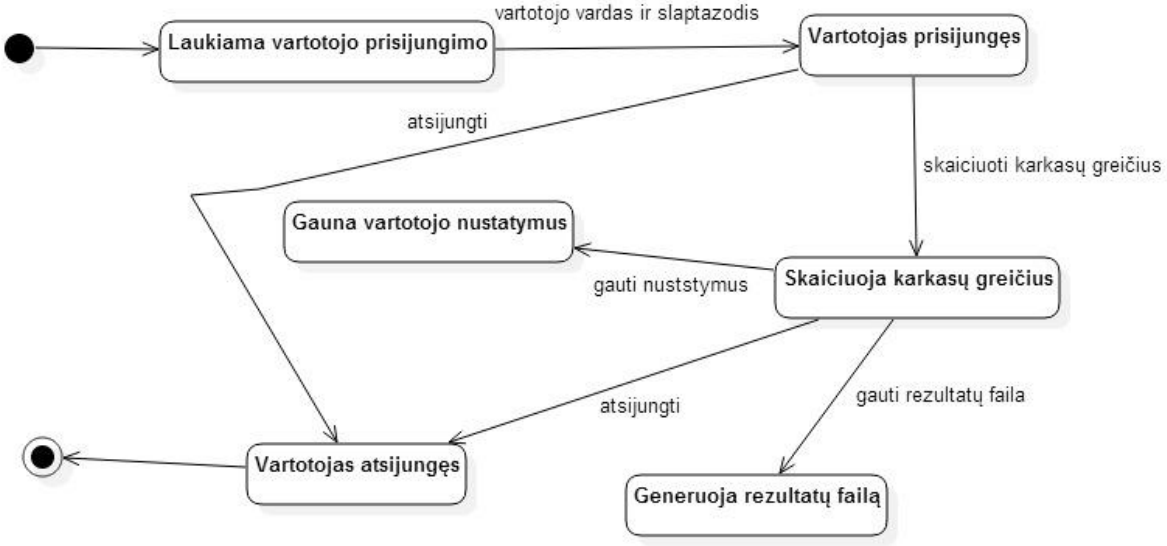
Karkaso informacijos redagavimo sekų diagrama (žr. 3.5.1.3 pav.) rodo, jog šią funkciją atlikti gali tik sistemos administratorius. Pirmiausia atsidarant karkasų informacijos puslapį jam yra parodomas visų karkaso savybių sąrašas (1, 2, 7, 8), šiam sąrašui gauti karkaso informacijos kontroleris kreipiasi į duomenų bazės klasę (3), kuri gauna duomenis ir kreipiasi į pagalbinę klasę, kad juos apdorotų (4). Apdoroti duomenys yra grąžinami karkaso informacijos kontroleriui (5, 6). Administratorius paspaudžia „pridėti naują savybę" bei suveda jos pavadinimą ir reikšmes (9), duomenys yra siunčiami kontroleriui (10), kuris kreipiasi į duomenų bazės klasę (11, 12, 13), kuri išsaugo duomenis ir grąžina informaciją apie sėkmingai atliktą operaciją (14, 15), tada vartotojui yra parodomas pranešimas apie sėkmingai atliktą operaciją (16).



3.5.1.3 pav. Sekų diagrama panaudojimo atvejui: Redaguoti karkasų informaciją

3.5.2. Būsenų diagrama

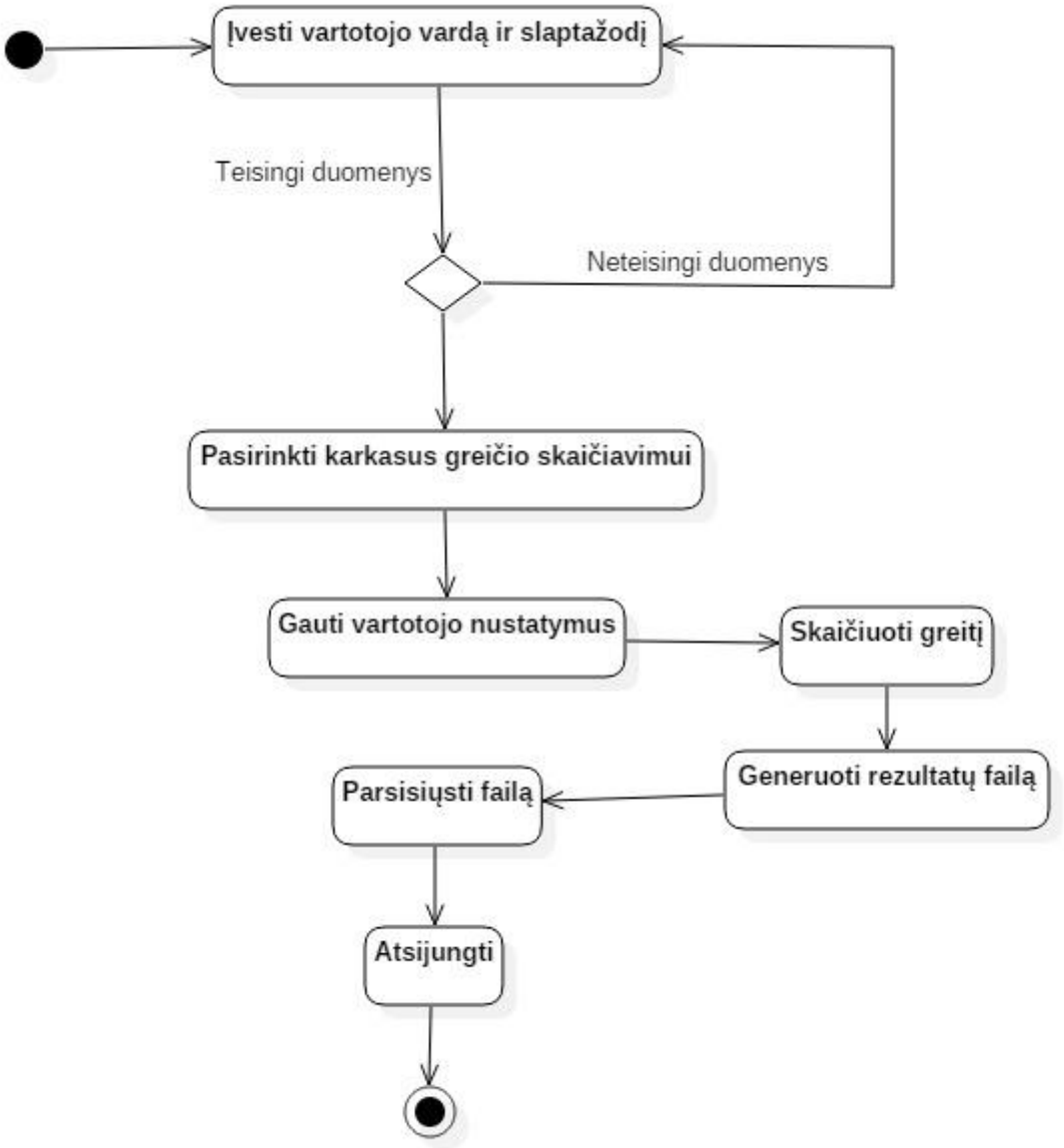
Tam pačiam scenarijui, koks buvo realizuotas sekų diagramoje (žr. 3.5.1.1 pav.), yra pavaizduota būsenų diagrama (žr. 3.5.2.1 pav.). Šioje diagramoje yra pavaizduotos būsenos, kuriose yra sistema duotajame scenarijuje. Pirmiausia sistema laukia, kol vartotojas prisijungs, kadangi neprisijungęs vartotojas sistema naudotis negali. Kai vartotojas pateikia savo prisijungimo duomenis (vartotojo vardą ir slaptažodį), būsena pasikeičia į prisijungusio vartotojo. Toliau vartotojui pareikalavus atlikti pasirinktų karkasų skaičiavimą sistemos būsena pasikeičia į karkaso greičio skaičiavimą. Atlikus skaičiavimus vartotojas paprašo rezultatų failo ir būseną pasikeičia į failo generavimo. Paskutinė sistemos būsena yra vartotojo atsijungimas, kurį jis gali atlikti tik prisijungęs.



3.5.2.1 pav. *PHP* karkasų charakteristikų palyginimo būsenų diagrama

3.5.3. Veiklos diagrama

Naudojantis ta pačia vartotojo veiksmų seka (kaip ir praeitose diagramose) buvo sugeneruota būsenų diagrama. Pirmoji veikla yra vartotojo prisijungimo duomenų įvedimas, suvedus duomenis, patikrinama, ar jie teisingi. Jeigu duomenys teisingi, vartotojas gali atlikti kitus veiksmus, priešingu atveju jis yra nukreipiamas į tą pačią veiklą, kad galėtų pataisyti blogai nurodytus duomenis. Sėkmingai prisijungęs vartotojas pasirenka norimus karkasus charakteristikų tyrimui. Toliau sistema gauna vartotojo nustatymus ir atlieka vartotojo pasirinktą karkasų tyrimo būdą (šiuo atveju greičio skaičiavimą). Atlikus skaičiavimus rezultatai parodomi vartotojui, šis pasirenka generuoti rezultatų failą, kai jis yra sugeneruojamas, vartotojas gali jį parsisiųsti. Galiausiai atlikus visus norimus veiksmus vartotojas atsijungia iš sistemos.

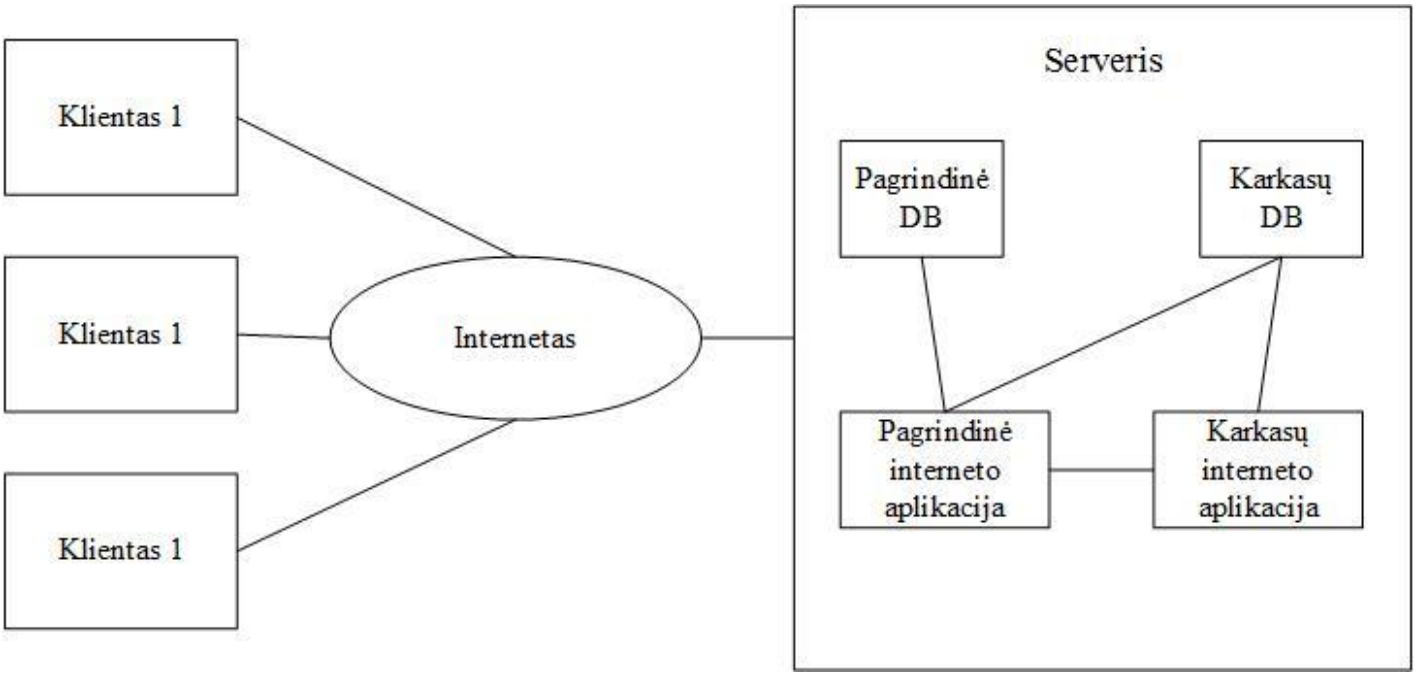


3.5.3.1 pav. *PHP* karkasų charakteristikų palyginimo veiklos diagrama

3.6. Išdėstymo vaizdas

Toliau pateiktame paveikslėlyje yra pavaizduotas sistemos išdėstymo vaizdas (žr. 3.6.1 pav.), kuriame matyti, jog vartotojas turėdamas interneto ryšį, naršyklės pagalba gali pasiekti serveryje patalpintą *PHP* programavimo kalbos charakteristikų tyrimo sistemą. Serveryje yra patalpintos dvi duomenų bazės, kurios skirtos skirtingoms programoms. Pagrindinė duomenų bazė yra orientuota teikti duomenis pagrindiniam sistemos puslapiui, kuriuo ir naudosis vartotojas. Kita duomenų bazė yra skirta saugoti duomenis, kuriuos naudos karkasų programą atlikdama manipuliaciją su tais duomenimis. Duomenų tipas ir struktūra šioje duomenų bazėje nėra svarbi. Duomenų bazės atskirtos siekiant padidinti duomenų saugumą, kadangi žmogus, įsilaužęs į vieną duomenų bazę, negalėtų gauti kitos bazės duomenų.

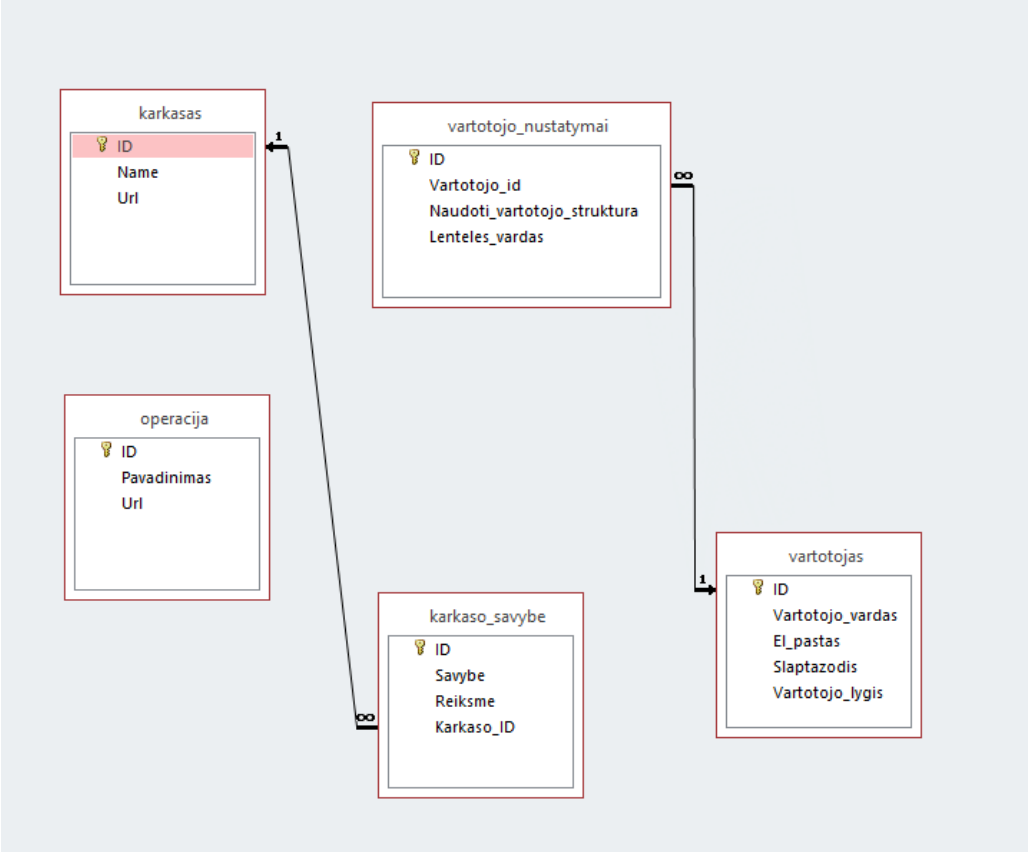
Serveryje taip pat yra patalpintos dvi programos. Pagrindinė programa yra interneto puslapis, kurį pamato vartotojas naršyklėje suvedęs svetainės adresą. Ši programa yra prieinama visiems vartotojams, tačiau šios programos funkcijos yra prieinamos priklausomai nuo vartotojo rolės. Tai reiškia, jog neprisijungusiam vartotojui galimybės yra stipriai apribotos. Kita programa yra skirta karkasų funkcijoms realizuoti. Šią programą gali iškviesti ir naudoti tiktai pagrindinė programa, tai reiškia, jog vartotojas karkasų programos tiesiogiai pasiekti negali.



3.6.1 pav. Sistemos išdėstymo vaizdas

3.7. Duomenų vaizdas

Toliau esančiame paveikslėlyje yra pateiktas kuriamos sistemos duomenų modelis (žr. 3.7.1 pav.), kuris reikalingas duomenims saugoti, kurie bus panaudoti sistemos kūrimui.



3.7.1 pav. Duomenų bazės modelis

3.8. Kokybė

Sistemos kokybei įvertinti gali būti naudojami šie parametrai: suprantamumas, išbaigtumas, glaustumas, pernešamumas, nuoseklumas, patikimumas, testuojamumas, pataisomumas, panaudojamumas, efektyvumas ir saugumas.

Architektūros įtaka sistemos kokybei:

- ✓ Užtikrina sistemos saugumą, kadangi yra naudojamos dvi duomenų bazės, taip pat prieiga prie sistemos yra apsaugota vartotojo vardu ir slaptažodžiu. Be to, duomenų bazės prieiga taip pat yra saugoma slaptažodžiu. Galiausiai vartotojo slaptažodis duomenų bazėje yra saugomas – užkoduotas.
- ✓ Sistema yra nesunkiai pernešama į kitą aplinką (šiuo atveju serverį). Tereikia programos failus ir duomenų bazės failą nukopijuoti iš vieno serverio ir perkelti į kitą. Duomenų bazės failą reikia užkrauti serveryje esančioje duomenų bazių valdymo sistemoje.
- ✓ Sistemos architektūra yra suprojektuota taip, kad padidintų sistemos išplečiamumą, tai yra, naujo karkaso į sistemą įliejimą. Architektūra leidžia nesunkiai, laikantis standartinių funkcijų išlaikymo kiekviename karkase sukurti naują karkasą, jį sukūrus įkrauti į serverį bei pasinaudojus programa užregistruoti (tik administratoriaus rolei).
- ✓ Ši sistema taip pat turi būti patikima, t.y. veikti 24 valandas per parą bei 7 dienas per savaitę. Atsitikus problemai ji turėtų būti išspręsta per 2 darbo dienas, šiam tikslui sistemą prižiūri administratorius, kuris ir yra atsakingas už teisingą sistemos veikimą.
- ✓ Sistemos architektūra leidžia nesunkiai atlikti jos testavimą, kadangi sistemos dalis, kuri naudoja karkasus, yra pritaikyta *unit* testams (*PHP* karkasai palaiko šį testavimo būdą).

4. TYRIMO IR EKSPERIMENTINĖ DALIS

4.1. Tyrimo tikslas

Pagrindinis šio tyrimo tikslas yra palyginti populiariausius *PHP* programavimo kalbos karkasus pagal pasirinktas charakteristikas bei nustatyti jų silpnąsias bei stipriąsias puses. Taip pat išsiaiškinti privalumus ir trūkumus naudojant *PHP* programavimo kalbos karkasą programuojant sistemą bei jos kūrimą nenaudojant jokio karkaso, tai yra *PHP* programavimo kalbos kodą su jo standartinėmis bibliotekomis.

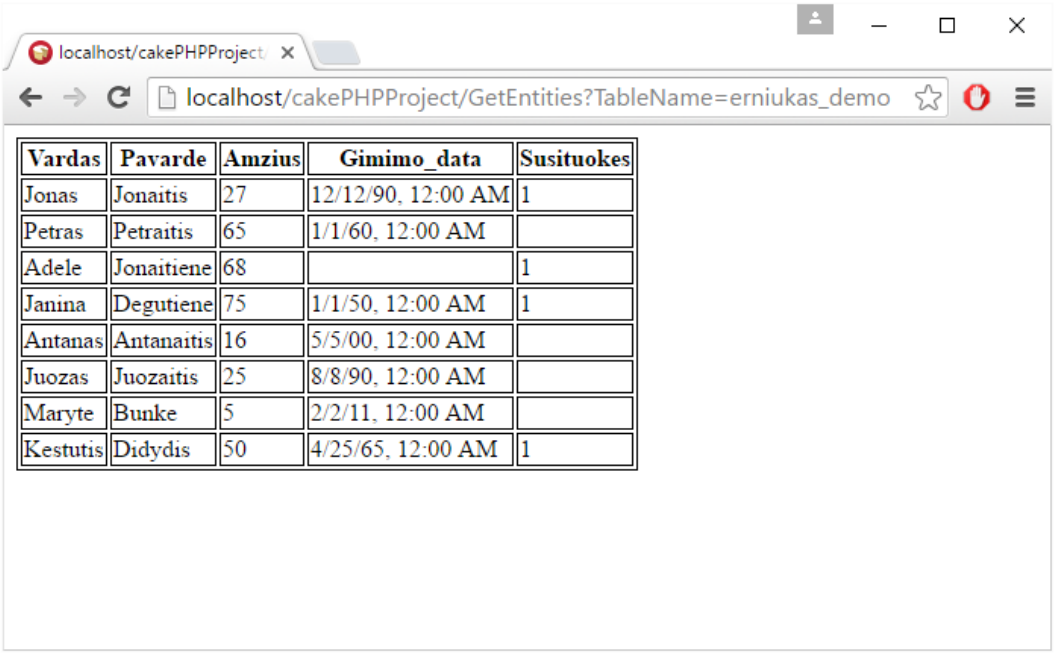
4.2. Eksperimento metodologija

PHP programavimų kalbos karkasų palyginimui buvo sukurtas internetinis puslapis. Šis puslapis buvo sukurtas naudojantis *PHP* programavimo kalba. Siekiant išvengti neobjektyvių rezultatų, puslapis buvo programuojamas nenaudojant jokio karkaso. Sukurtos sistemos pagrindinė vartotojo sąsaja yra pateikta toliau esančiame paveikslėlyje (žr. 4.2.1 pav.). Norėdamas atlikti karkasų palyginimą, vartotojas turi būti prisijungęs prie sistemos. Karkasų palyginimo langas yra suskirstytas į 3 stulpelius. Pirmajame vartotojas gali pasirinkti karkasus, kurie bus lyginami. Kitame stulpelyje yra pasirenkamos operacijos, kurios bus įvykdomos palyginimo metu. Operacijos yra tam tikras sistemos funkcionalumas (duomenų gavimas ir jų redagavimas, vartotojo prisijungimas, sesijos valdymas ir t.t), dažniausiai funkcija. Paskutiniame stulpelyje yra pasirenkamos charakteristikos, pagal kurias bus atliktas palyginimas. Vartotojas taip pat gali įvesti ciklų skaičių, kuris nusako, kiek kartų bus vykdoma ta pati operacija. Įvedęs visus reikalingus parametrus ir paspaudęs skaičiavimo mygtuką vartotojas gauna rezultatus, pateiktus grafine forma (žr. 4.2.2). Vartotojui taip pat yra pateikiama bendra informacija (vykdymo laikas, data, vartotojas bei bendros statistikos) apie tyrimą.

4.2.1 pav. Karkasų palyginimo lango grafinė vartotojo sąsaja

4.2.2 pav. Rezultatų palyginimo lango grafinė vartotojo sąsaja

Siekiant palyginti *PHP* programavimo kalbos karkasus su kiekvienu iš jų (įskaitant ir puslapį realizuotą be karkaso) buvo realizuotas interneto puslapis. Tyrime buvo analizuojamas puslapis, kuris pateikia duomenis iš duomenų bazės lentelės (žr. 4.2.3 pav.). Pagrindinis puslapis, kuris skirtas karkasų palyginimui, pateikdavo užklausas puslapiui, realizuotam konkrečiu *PHP* programavimo kalbos karkasu, ir rinkdavo charakteristikų reikšmes. Užklausos buvo atliekamos naudojantis *Ajax* technologiją. Užklausų į tą patį puslapį buvo atliekama pagal vartotojo nurodytą ciklų skaičių, gavus rezultatus buvo gaunamas charakteristikos vidurkis pasirinktam karkasui.



Vardas	Pavarde	Amzius	Gimimo_data	Susituokes
Jonas	Jonaitis	27	12/12/90, 12:00 AM	1
Petras	Petraitis	65	1/1/60, 12:00 AM	
Adele	Jonaitiene	68		1
Janina	Degutiene	75	1/1/50, 12:00 AM	1
Antanas	Antanaitis	16	5/5/00, 12:00 AM	
Juozas	Juozaitis	25	8/8/90, 12:00 AM	
Maryte	Bunke	5	2/2/11, 12:00 AM	
Kestutis	Didydis	50	4/25/65, 12:00 AM	1

4.2.3 pav. *PHP* programavimo kalbos karkasu realizuotas puslapis

Eksperimento metu kiekvienas *PHP* programavimo kalbos karkasas buvo lyginamas pagal 4 charakteristikas. Pirmoji iš jų – greitis. Ši metrika buvo pasirinkta dėl labai didelės puslapių užkrovimo spartos reikšmės šiais laikais. Kadangi puslapis, kuris kraunasi ilgiau nei 2 sekundes, trikdo vartotojų darbą ir juos erzina, visų sistemų vienas iš esminių parametrų yra puslapio užkrovimo greitis. Taigi ši charakteristika nusako laiką nuo *Ajax* užklausos pateikimo iki atsakymo gavimo, kitaip tariant, puslapio užkrovimo. Dažniausiai greičiui nustatyti yra naudojamas didesnis ciklų skaičius (šiame tyrime ciklų skaičius buvo 10). Kadangi užkraunant puslapį vieną kartą egzistuoja didesnis tam tikro rezultato atsitiktinumas, didinant ciklų skaičių rezultatai tampa tikslesni. Taigi gavus visas 10 greičio reikšmių jos buvo sudedamos ir paskaičiuojamas vidurkis taip gaunant greičio charakteristikos reikšmę.

Kita charakteristika yra *PHP* programavimo kalbos karkaso naudojamų klasių skaičius. Ši metrika nusako, kiek komponentų naudoja tiriamas karkasas. Didesnis naudojamų klasių kiekis turi įtakos puslapio užkrovimo greičiui, kadangi kiekvienos klasės užkrovimas reikalauja laiko. Ši charakteristika taip pat gali apibūdinti *PHP* programavimo kalbos karkaso funkcionalumą. Kitaip tariant, didesnis naudojamų klasių kiekis gali reikšti daugiau galimų funkcijų ir komponentų karkase. Verta paminėti, jog nenaudojant jokio karkaso sistemoje vis tiek yra naudojamas standartinės *PHP* programavimo kalbos klasės, kurios yra skirtos pagrindiniam *PHP* programavimo kalbos funkcionalumui: klaidų valdymui (angl. *exception*), masyvų valdymui, laiko zonai, kalendoriui, pranešimų formatavimui ir t.t. Šios charakteristikos pasirinkimą lėmė tikslas išsiaiškinti, kokio sudėtingumo yra karkasas bei kiek funkcijų atlieka.

Bene didžiausią laiką sistemos kūrimo metu užima programavimo darbai, taigi programuotojui yra labai svarbu, kiek kodo eilučių prireiks parašyti norint atlikti tam tikrą operaciją. Mažesnis kodo eilučių skaičius gali sumažinti ne tik sistemos kūrimo laiką, bet ir palengvinti jos tobulinimo arba keitimo darbus. Būtent dėl šių priežasčių ši charakteristika ir buvo įtrauktą į *PHP* programavimo kalbos karkasų palyginimą. Šiame tyrime kodo eilučių kiekis buvo apskaičiuojamas karkasų realizuojamų programų viduje. Gauta reikšmė buvo įrašoma užkrautame puslapyje kaip paslėptas elementas. Kodo eilučių kiekis buvo skaičiuojamas konkrečiam metodui, pavyzdžiui, duomenų gavimui iš duomenų bazės.

Paskutinė tirta charakteristika yra ciklomatinis sudėtingumas. Ši metrika nusako kodo sudėtingumą arba, kitaip tariant, visus galimus kodo įvykdymo kelius. Ciklomatinis sudėtingumas yra labai svarbi metrika, kadangi turi įtakos kitiems projekto veiksniams. Pavyzdžiui, kuo didesnis funkcijos ciklomatinis sudėtingumas, tuo sunkiau ją testuoti ir perskaityti. Taigi, darosi sudėtingesnės sistemos tobulinimo galimybės bei pasunkėja testavimas, o tai įtakoja sudėtingesnį kokybiškos sistemos užtikrinimą. Ši charakteristika gaunama panašiu būdu kaip ir kodo eilučių skaičius bei užkrautų klasių kiekis. Visos šios metrikos yra išsaugojamos užkrautame puslapyje kaip paslėpti elementai.

4.3. Eksperimento aplinka

Eksperimento aplinkos bei programinės įrangos parametrai:

- Pagrindinė programa: ~3000 kodo eilučių.
- *PHP* programavimo kalba realizuoti puslapiai: ~ 2000 kodo eilučių.
- *MySql* duomenų bazė su 300000 duomenų įrašų.
- Duomenų bazė ir interneto puslapiai patalpinti lokaliame *XAMP* serveryje.
- Operacinė sistema: *Windows 10 education* 64 bitų.
- Operatyvioji atmintis: *16gb DD3*.
- Procesorius: *Intel Core i7 4710HQ*.
- Grafinis procesorius: *NVIDIA GeForce GTX 860M*.

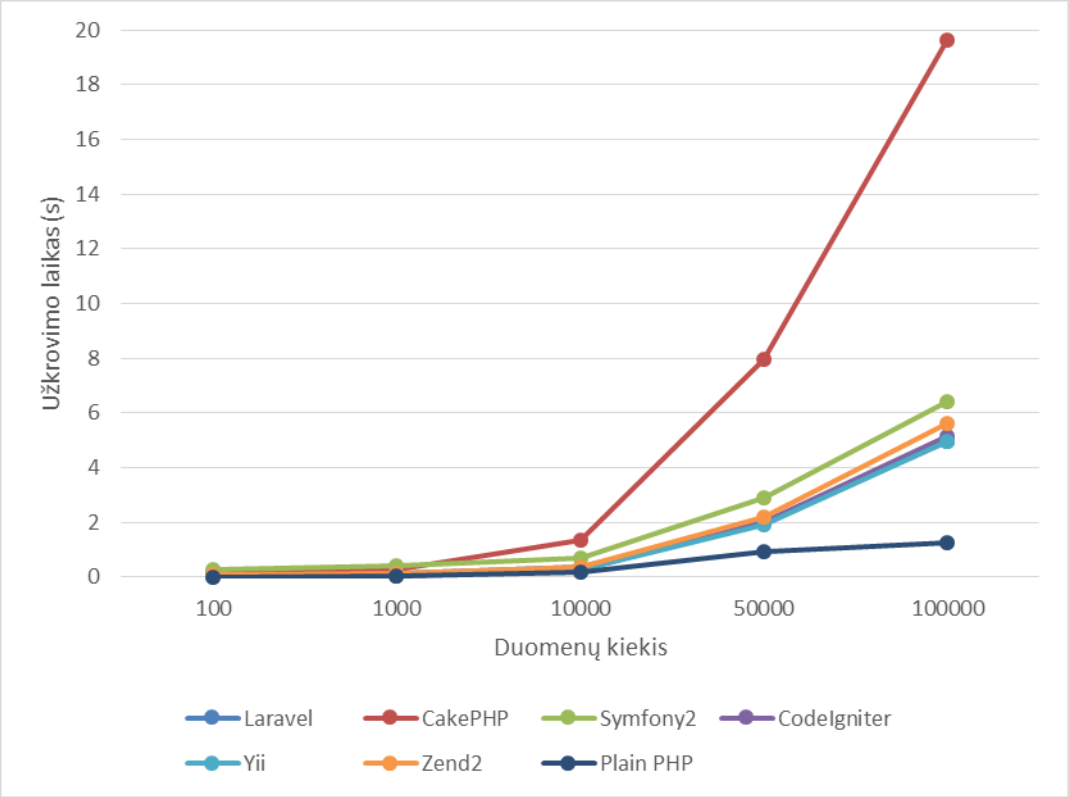
4.4. Eksperimento rezultatų analizė

Greičio charakteristikos rezultatai yra pateikti toliau esančioje lentelėje (žr. 4.4.1 lent.). Šie rezultatai rodo puslapio užkrovimo laiką su tam tikru duomenų skaičiumi, tai yra įrašų skaičiumi, kuris yra pateiktas kaip lentelė puslapyje. Sprendžiant iš rezultatų, užklausos yra greičiausiai apdorojamos programoje, kuri nenaudoja jokio karkaso. Apskaičiavus vidutinį užklausos apdorojimą naudojant *PHP* programavimo kalbos karkasą ir nenaudojant jo, galima teigti, jog programa, kuri nenaudoja jokio karkaso, veikia 5 kartus greičiau. Lyginant karkasus tarpusavyje galima pastebėti, jog greičiausiai veikia *Laravel* ir *Yii* karkasai, kurie pateikdami 100000 įrašų lentelėje vidutiniškai užtrukdavo netoli 5 sekundžių. Toliau rikiuojasi *CodeIgniter* ir *Zend2* karkasai, kurie minėtus įrašus apdoroja virš 5 sekundžių. *Symfony2 PHP* programavimo karkasas 100000 duomenų kiekiui apdoroti užtrunka apie 6,4 sekundės, kas yra 23% lėčiau negu šios metrikos lyderio *Yii* karkaso. Galiausiai blogiausią rezultatą parodė *CakePHP* karkasas, kuris didelį duomenų kiekį apdorojo labai prastu greičiu, siekiančiu 19,626 sekundės per užklausą.

4.4.1 lentelė. Greičio metrika kiekvienai programai išreikšta sekundėmis

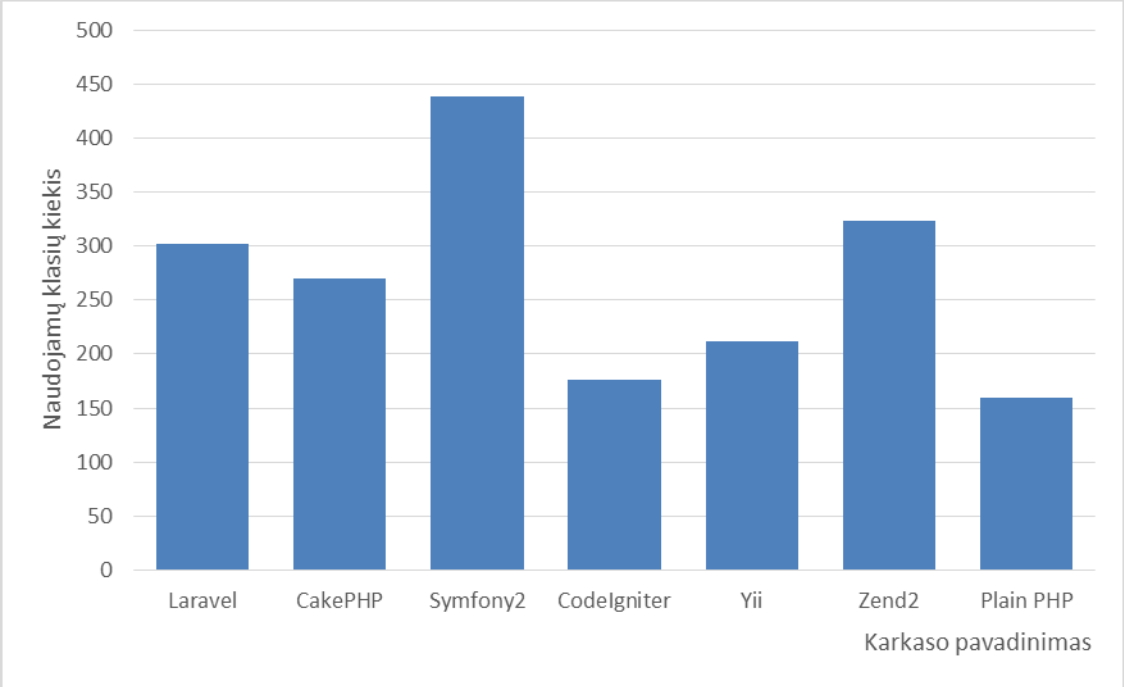
Pavadinimas	**Duomenų kiekis**				
	100	**1000**	**10000**	**50000**	**100000**
Laravel	0.136	0.138	0.353	1.988	4.982
CakePHP	0.196	0.281	1.374	7.94	19.626
Symfony2	0.289	0.394	0.711	2.889	6.435
CodeIgniter	0.036	0.054	0.272	1.986	5.138
Yii	0.062	0.072	0.264	1.913	4.967
Zend2	0.106	0.121	0.364	2.204	5.603
Plain PHP	0.017	0.031	0.186	0.949	1.282

PHP programavimo kalbos karkasų greičio priklausomybė nuo duomenų kiekio yra pavaizduota tolimesniame paveikslėlyje (žr. 4.4.1 pav.). Kaip matyti iš diagramos, jokio karkaso nenaudojimas duoda tolygiausius rezultatus. Tai yra, kylant duomenų skaičiui greitis drastiškai nemažėja. Analizuojant *PHP* programavimo kalbos karkasų greičio rezultatus matyti, jog esant iki 10000 duomenų kiekiui jų rezultatų apdorojimo greičiai yra gana tolygūs. Tačiau nuo 10000 kiekio duomenų pradeda gana stipriai kilti. Šioje vietoje į dėmesį krenta *CakePHP* karkasas, kurio apdorojimo greitis stipriai sumažėja pasiekus 10000 duomenų kiekio žymą. Taigi sprendžiant pagal gautus rezultatus šis karkasas nėra naudingas, kai interneto svetainėje yra apdorojami dideli duomenų kiekiai.



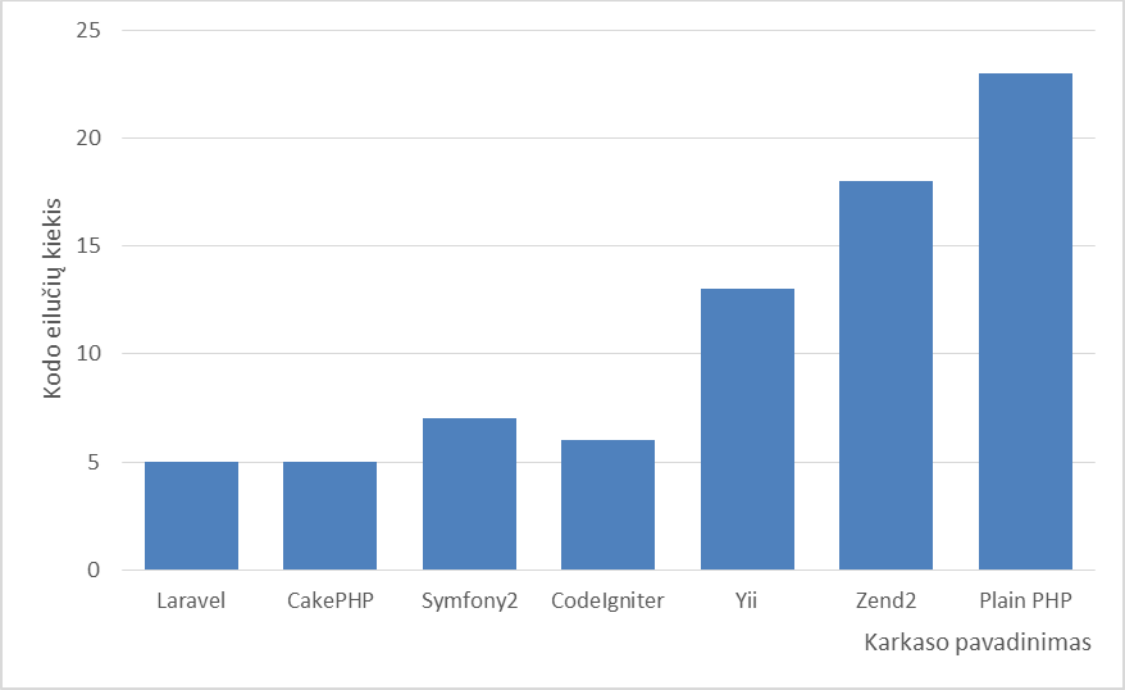
4.4.1 pav. Greičio priklausomybė nuo duomenų kiekio

Visų programų, realizuotų skirtingais karkasais, naudojamas klasių kiekis yra pavaizduotas toliau esančiame paveikslėlyje (žr. 4.4.2 pav.). Iš grafiko matyti, jog *Symfony2 PHP* programavimo kalbos karkasas naudoja daugiausiai klasių. Teorinėje tiriamų karkasų analizėje minėtas karkasas turėjo daugiausiai funkcionalumo, kuris gali būti panaudotas kuriant internetinius puslapius. Taigi šiuo atveju teorinė analizė atitinka eksperimento rezultatus, tai yra, kuo daugiau karkasas naudoja klasių, tuo daugiau funkcijų palaiko. Kaip ir buvo galima tikėtis, nenaudojant jokio karkaso klasių užkrovimas yra mažiausias, kadangi yra naudojamos tik standartinės *PHP* programavimo kalbos klasės. Galima pastebėti, jog Symfony2 karkasas naudoja net 279 savo klases (visų klasių skaičius atėmus standartines *PHP* klases). Toliau pagal užkraunamų klasių skaičių minėtini *Zend2* ir *Laravel* karkasai, kurie naudoja kiek daugiau nei 300 klasių. *CakePHP* ir *Yii* karkasai naudoja daugiau negu 200 klasių savo funkcionalumui realizuoti. Tuo tarpu *CodeIgniter PHP* programavimo kalbos karkasas naudoja mažiausiai klasių, kurių skaičius nesiekia 200, tai reiškia, jog šis karkasas tenaudoja 17 savo klasių norimam funkcionalumui įvykdyti, kas optimizuoja jo spartą, kadangi šis karkasas parodė geriausius rezultatus ir greičio metrikoje. Verta pažymėti, kad Laravel karkasas šiek tiek išsiskiria šiuo aspektu, nes jis buvo vienas greičiausių karkasų (2 rezultatas), tačiau naudojamų klasių kiekis buvo gana didelis (3 rezultatas). Apibendrinant galima teigti, kad *Laravel* karkasas yra pakankamai greitas ir turi daug funkcionalumo, o *CodeIgniter* yra truputį greitesnis, tačiau funkcijų skaičiumi atsilieka nuo pastarojo.



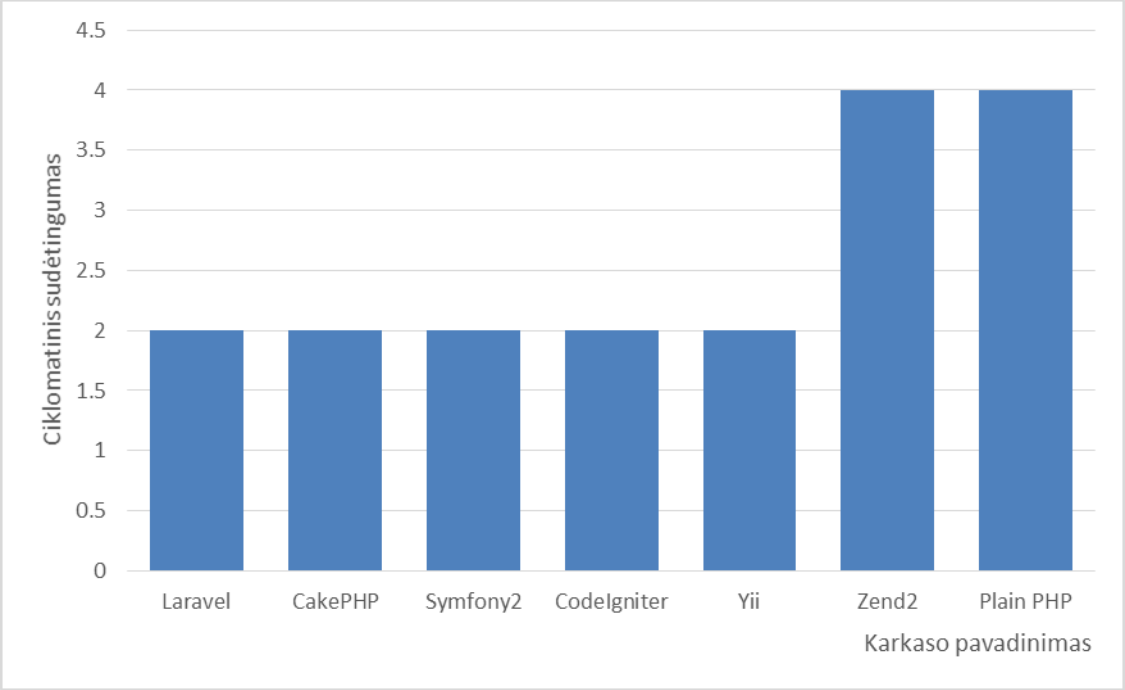
4.4.2 pav. Naudojamų klasių kiekis kiekvienai programai

Kita metrika, pagal kurią buvo atliekamas eksperimentas, yra kodo eilučių kiekis, reikalingas norimai operacijai atlikti. Ši metrika yra labai svarbi programuotojams, nes padeda taupyti laiką bei lengvina kodo skaitomumą. Kodo eilučių skaičius kiekvienai sukurtai programai yra pateiktas toliau esančioje diagramoje (žr. 4.4.3 pav.). Daugiausiai kodo eilučių pareikalavo operacija, kuri buvo sukurta nenaudojant jokio karkaso. Taip nutiko dėl to, kad daugumą standartinių funkcijų, tokių kaip bendravimas su duomenų baze, programuotojui reikia parašyti pačiam, o su karkasu tokios funkcijos dažniausiai būna realizuotos paties karkaso. Dėl šios priežasties ir išauga kodo eilučių skaičius bei klaidos tikimybė. Lyginant *PHP* programavimo kalbos karkasus tarpusavyje, aiškūs lyderiai yra *Laravel* ir *CakePHP* karkasai, kuriems prireikė 5 kodo eilučių duomenų gavimo operacijai atlikti. Toliau būtų *Symfony2* ir *CodeIgniter* karkasai, kuriems prireikė atitinkamai 7 ir 6 kodo eilučių. Likę du karkasai gana ženkliai atsilieka nuo pastarųjų, tai yra *Yii* ir *Zend2* karkasai. Šie karkasai naudoja atitinkamai 13 ir 18 kodo eilučių duomenų gavimo operacijai atlikti. Sprendžiant iš rezultatų programuotojui, kuriančiam su pastaraisiais karkasais, reikia parašyti apie 3 kartus daugiau kodo eilučių, o tai užima apie 3 kartus daugiau laiko. Šis veiksnys gali būti labai svarbus parenkant reikiamą karkasą projektui, kadangi resursai visada yra riboti ir užsakovas nori gauti produktą kiek įmanoma greičiau. Dar vienas labai svarbu dalykas yra klaidos, kurios gali atsirasti kuriant programą. Pagal gautus rezultatus, naudojant *Yii* arba *Zend2* karkasus galima padaryti apie 3 kartus daugiau klaidų nei naudojant *Laravel* arba *CakePHP* karkasus. Kuo mažesnė yra klaidų tikimybė, tuo didesnė galutinio produkto kokybė.



4.4.3 pav. Kodo eilučių kiekis kiekvienai programai

Paskutinė tirta metrika yra ciklomatinis sudėtingumas. Šioje charakteristikoje dauguma karkasų parodė vienodus rezultatus (ciklomatinio sudėtingumo reikšmė 2) (žr. 4.4.4 pav.). Šie rezultatai rodo, kad kodo sudėtingumas nėra didelis ir ši programa gali būti lengvai testuojama bei klaidų tikimybė yra ganėtinai maža. Vienintelis *Zend2* karkasas buvo sudėtingesnis (ciklomatinio sudėtingumo reikšmė 4), ši reikšmė atitinka ciklomatinį sudėtingumą programos, kuri buvo realizuota nenaudojant jokio karkaso. Tokie rezultatai gana nebūdingi, kadangi karkaso naudojimas turėtų ženkliai sumažinti ciklomatinį sudėtingumą. Verta paminėti, jog egzistuoja ir žmogiškasis faktorius, kadangi visas programas realizavo tas pats žmogus, todėl yra tikimybė, kad naudojantis *Zend2* karkasu nepavyko rasti funkcijų palengvinančių jo naudojimą. Tačiau šis faktas taip pat rodo šio karkaso dokumentacijos trūkumą arba sunkų jos įsisavinimą pačioje programoje. Apibendrinant pastebima, kad *Zend2* karkaso realizuota programa neparodė jokio pranašumo lyginant su programa be karkaso.



4.4.4 pav. Ciklomatinis sudėtingumas kiekvienai programai

5. IŠVADOS

1. Įvertinus tyrimo rezultatus pagal įvairias charakteristikas (greitį, naudojamų klasių, kodo eilučių kiekį ir ciklomatinį sudėtingumą) *Laravel* karkasas daugumoje jų pirmauja arba yra tarp lyderių.
2. Atlikus karkasų palyginimą pagal greitį, galima teigti jog *Laravel* ir *CodeIgniter* karkasai yra greičiausi ir likusius karkasus lenkia vidutiniškai 0.76 sekundės, neįtraukiant *CakePHP* karkaso, kurio rezultatai stipriai išsiskiria.
3. Kaip ir tikėtasi kuriant programinę įrangą nenaudojant jokio karkaso turi savo privalumų, tokių kaip didesnis apdorojimo greitis (vidutiniškai 5 kartus greičiau nei naudojant karkasą), tačiau kuriant programas tokiu principu yra reikalingi geresni programavimo įgūdžiai, taip pat tai reikalauja daugiau pastangų bei supratimo apie apsaugą.
4. Atlikus teorinį karkasų palyginimą pagal populiarumą ir funkcijų gausą *Laravel PHP* programavimo kalbos karkasas yra aiškus lyderis. Remiantis *Google Trends* įrankiu (2016 balandis) *Laravel* karkasas paieškų skaičiumi dvigubai lenkia antroje vietoje esantį *CodeIgniter* karkasą.
5. Lyginant teorinį karkasų populiarumą ir gautus eksperimento rezultatus galima aptikti koreliaciją tarp jų. Kadangi populiariausias karkasas (*Laravel*) tarp vartotojų rodė geriausius rezultatus. O prasčiausią rezultatą turintis (*Zend2*) buvo prasčiausias ir lyginant jo metrikas. Kiti karkasai išsirikiavo panašiai lyginant populiarumą ir gautus metrikų rezultatus.

6. LITERATŪRA

[1] V.Chaychi, „How PHP Works,“ 06 Balandis 2013. [Tinkle]. Available: http://www.runanonlinebusiness.com/how-php-works/. [Kreiptasi 11 Lapkritis 2014].

[2] L. L.Welling, „PHP and MySQL Web Develpment,“ 2003. [Tinkle]. Available: http://www.google.lt/books?hl=lt&lr=&id=fGzifMim4qYC&oi=fnd&pg=PA1&dq=php+architecture&ots=mD9H-U1lF4&sig=54__bRfunb_xtBs0A0_qUiNlb_E&redir_esc=y#v=onepage&q=php%20architecture&f=false. [Kreiptasi 11 Lapkritis 2014].

[3] P. R. K.Tatroe, „Programming PHP,“ 05 Vasaris 2013. [Tinkle]. Available: http://www.google.lt/books?hl=lt&lr=&id=w69fcU5dHgAC&oi=fnd&pg=PR5&dq=php+programming&ots=5oOXBQOo4A&sig=hZAW7T2Z7BDB9OImdieSmdsGcgY&redir_esc=y#v=onepage&q=php%20programming&f=false. [Kreiptasi 10 Lapkritis 2014].

[4] C.Hopkins, „The MVC Pattern and PHP,“ 4 Kovas 2013. [Tinkle]. Available: http://www.sitepoint.com/the-mvc-pattern-and-php-1/. [Kreiptasi 11 Lapkritis 2014].

[5] „Historical trends in the usage of server-side programming languages for websites,“ 11 Lapkritis 2014. [Tinkle]. Available: http://w3techs.com/technologies/history_overview/programming_language. [Kreiptasi 11 Lapkritis 2014].

[6] „CakePHP Cookbook Documentation,“ Lapkritis 2014. [Tinkle]. Available: http://book.cakephp.org/2.0/_downloads/en/CakePHPCookbook.pdf. [Kreiptasi 12 Lapkritis 2014].

[7] „Architecture of Laravel Applications,“ [Tinkle]. Available: http://laravelbook.com/laravel-architecture/. [Kreiptasi 12 11 2014].

[8] „Symfony,“ [Tinkle]. Available: http://symfony.com/pdf/Symfony_book_2.5.pdf?v=4. [Kreiptasi 13 11 2014].

[9] „About Yii,“ [Tinkle]. Available: http://www.yiiframework.com/about/. [Kreiptasi 13 11 2014].

[10] „Features of Yii,“ [Tinkle]. Available: http://www.yiiframework.com/features/. [Kreiptasi 13 11 2014].

[11] „Model-View-Controller (MVC),“ [Tinkle]. Available: http://www.yiiframework.com/doc/guide/1.1/en/basics.mvc. [Kreiptasi 13 11 2014].

[12] „CodeIgniter Features,“ [Tinkle]. Available: https://ellislab.com/codeigniter/user-guide/overview/features.html. [Kreiptasi 13 11 2014].

[13] „Application Flow Chart,“ [Tinkle]. Available: https://ellislab.com/codeigniter/user-guide/overview/appflow.html. [Kreiptasi 13 11 2014].

[14] „About,“ [Tinkle]. Available: amework.zend.com/about/. [Kreiptasi 13 11 2014].

[15] „Why Zend Framework Plugins Save You Time,“ Liepa 2012. [Tinkle]. Available: http://www.masterzendframework.com/zend-framework/why-zend-framework-plugins-save-you-time. [Kreiptasi 13 Lapkritis 2014].

[16] „Google trends,“ [Tinkle]. Available: https://www.google.lt/trends/explore#q=%2Fm%2F0cdvjh%2C%20laravel%2C%20%2Fm%2F09t3sp%2C%20codeigniter%2C%20symfony&cmpt=q. [Kreiptasi 15 11 2014].

[17] B.Skvorc, „Best PHP Frameworks for 2014,“ 28 Gruodis 2013. [Tinkle]. Available: http://www.sitepoint.com/best-php-frameworks-2014/. [Kreiptasi 15 11 2014].

[18] „Compare PHP frameworks,“ [Tinkle]. Available: http://www.bestwebframeworks.com/compare-web-frameworks/php/. [Kreiptasi 15 11 2014].

[19] „PHP frameworks comparison,“ 20 Rugsėjis 2014. [Tinkle]. Available: http://socialcompare.com/en/comparison/php-frameworks-comparison. [Kreiptasi 15 Lapkritis 2014].

[20] S. L.Lancor, „Analyzing PHP frameworks for use in a project-based software engineering course,“ 2013. [Tinkle]. Available: http://dl.acm.org/citation.cfm?id=2445350. [Kreiptasi 26 Kovas 2015].

[21] H.Nylen, „PHP Framework Performance for Web,“ 08 2012. [Tinkle]. Available: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5369976&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5369976. [Kreiptasi 28 Kovas 2015].

7. TERMINŲ IR SANTRUMPŲ ŽODYNAS

PHP	atviro kodo, serverio pusės programavimo kalba, skirta kurti dinaminiams interneto puslapiams.
Cookies	mažas tekstinis failas, kuriame yra žinutė gauta iš interneto serverio. Šis failas saugomas naršyklėje.
Domenas	interneto adresas (pvz. internetosvetaine.lt).
OOP	objektiškai orientuotas programavimas, tai programavimo tipas, kuriame yra apibrėžiama ne tik duomenų struktūros tipas, bet ir operacijos naudojamos šiai struktūrai. Duomenų struktūra objektiniame programavime yra laikoma objektu, kuris turi duomenis ir funkcijas.
Serveris	tinkle esantis kompiuteris ar kitoks įtaisas, kuris valdo tinklo resursus.
HTTP	protokolas, kuris apibūdina žinučių formavimą ir perdavimą bei kokius veiksmus turi prisiimti serveris ir interneto naršyklė, atliekant įvairiausias komandas.
Sesija	vartotojo veikla, kurią atlieka būdamas interneto puslapyje per tam tikrą laiką.
Serverio pusė	visi veiksmai, kurie yra atliekami serveryje (pvz. duomenų bazė, *PHP*).
Kliento pusė	visi veiksmai, kurie yra atliekami naršyklėje (pvz. *HTML, JavaScript*).
Asp	technologija, kuri leidžia kodus esančius interneto puslapyje vykyti interneto serveryje.
Python	objektiškai orientuota atviro kodo programavimo kalba.
Java	aukšto lygio objektiškai orientuota programavimo kalba.
C	aukšto lygio programavimo kalba, kuri yra labai plačiai naudojama, nuo verslo iki inžinerinių programų sprendimų.
Operacinė sist.	tai pagrindinė kompiuterio programinė įranga naudojama įrašyti kitas programas, bei atlieka pagrindines kompiuteriui naudoti skirtas funkcijas.
Linux	atviro kodo operacinė sistema.
Derinimas	būdas, surasti klaidas kode ir jas ištaisyti.
CSS	kalba skirta aprašyti dokumento išvaizdą.
JavaScript	programavimo kalba, kuri leidžia kurti interaktyvius interneto puslapius.
DOM	(angl. *document object model*) nusako interneto puslapio objektų (tekstas, paveikslai, nuorodos ir t.t) yra vaizdavimą.
Duomenų bazė	informacijos kolekcija paruošta taip, kad kompiuterio programa galėtų greitai pasiekti norimus duomenis.
MySQL	atviro kodo reliacinės duomenų bazės valdymo sistema.
Reliacinė DB	tokia duomenų bazė, kuri saugo duomenis lentelėse.
Nereliacinė DB	duomenų bazė neturinti griežtai apibrėžtos duomenų saugojimo struktūros. Šioje duomenų bazėje duomenys dažniausiai saugomi rakto ir reikšmės principu.
Transakcija	tai duomenų bazėje atliekamų veiksmų seka.
JQuerry	*JavaScript* programavimo kalbos biblioteka.
Ajax	tai kartu sujungtos kelios technologijos (*JavaScript* ir *XML*), kurios leidžia atlikti puslapio atnaujinimas be jo perkrovimo.
NetBeans	programų kūrimo aplinka skirta *Java* bei kitoms programavimo kalboms.

8. PRIEDAI

Toliau yra pateikimas straipsnis, kuris buvo pristatytas IVUS 2016 konferencijoje.

Comparison of popular PHP frameworks

Ernestas Burokas
Kaunas University of Technology, Faculty of Informatics
Kaunas, Lithuania
ernestas.burokas@ktu.edu

Dominykas Barisas
Kaunas University of Technology, Faculty of Informatics
Kaunas, Lithuania
dominykas.barisas@ktu.lt

PHP frameworks are meant to facilitate software development and are becoming more and more popular in web based projects. Before starting the project it is extremely important to choose framework that goes along well with the system and customer requirements. The decision is impeded by a wide variety of frameworks available in the market. A considerable effort is required for the user to get acquainted with all the popular frameworks, configure environments, prepare test cases and evaluate framework effectiveness and other characteristics.

In order to facilitate the evaluation process, a tool for PHP framework comparison was developed. Research based on this tool was conducted using six of the most popular frameworks (Symfony2, CakePHP, Zend2, Laravel, CodeIgniter and Yii). Results were compared in four characteristics: speed, class usage, code usage and cyclomatic complexity.

Keywords—PHP; frameworks; Laravel; Symfony2; CakePHP; Zend; CodeIgniter; Yii; comparison

I. Introduction

Decision to analyze PHP programming language frameworks was made because of lack of literature about most popular PHP frameworks and due to the framework popularity. Main purpose of this research is to pick out the best framework based on main characteristics. To achieve this goal a web tool was created. This tool allows comparing frameworks using different characteristics with different amount of data and structure in database. To make research more extensive, a plain PHP (no framework) comparison was included. It is beneficial to evaluate the cases when PHP frameworks might not be necessary.

II. Popularity of PHP Frameworks

It is fair to say that PHP is leading programming language in web site development in these days. According to statistics [1], 81.8% of all sites are written in PHP, in the second place with 15.9% is .NET. These two languages are taking 97.7% of all available websites.

Looking at all programming languages frameworks popularity among the websites (Fig. 1) it is obvious that PHP frameworks play a major role. Diagram shows that about 40 percent of websites are using PHP programming language frameworks. Next after PHP is .Net which have 21.84% of frameworks usage. Finally, other less known languages are taking small part of website usage [2].

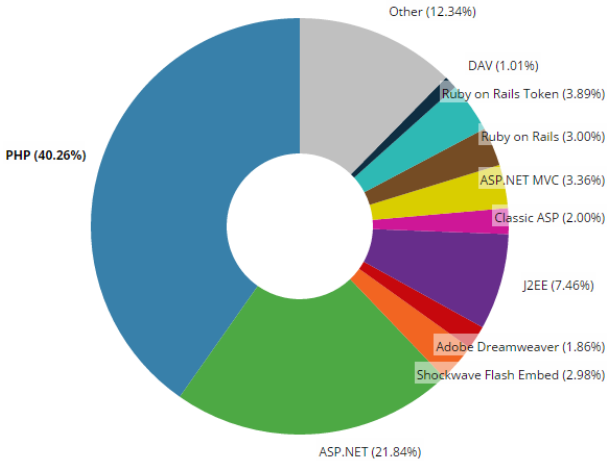


Fig. 1. Frameworks usage in websites

III. Architecture of PHP Frameworks

Programming language framework is collection of libraries used to increase system maintainability and reduce time of coding. PHP programming language frameworks have a lot of built-in functions such as: access to database, session management, security etc. These functions help to reduce coding and testing effort for developer, because it eliminates the need of manual implementation of various features. Most of the frameworks also maintain simple and clear folder structure. It keeps different framework parts apart from each other and facilitates file manipulation [3] [2].

All frameworks that are addressed in this research are using MVC model. The model isolates user interface, business logic and data model from each other and defines relationships between them. Well-designed MVC project enables independent work between front-end and back-end developers with no interaction.

MVC pattern consist of three major parts: model, view and controller. Data and relationship between different parts are shown in Fig. 2. First of all, user sends request to the controller, where the request is analyzed. Further, the controller calls model responsible for the logic and connection with database to get data that is needed to perform request. When model finishes job all data is returned back to the controller. Finally, all data is returned to view and page is rendered [4].

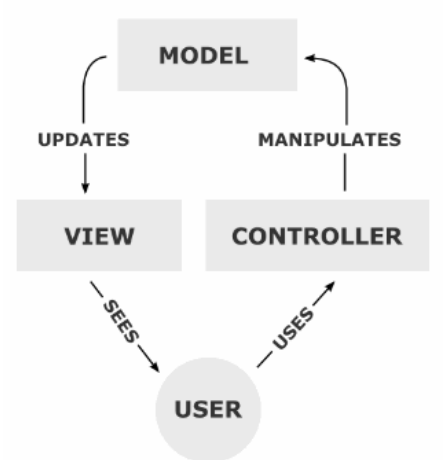


Fig. 1. MVC pattern

Model is like connection between controller and view. Essentially model is sort of "blind", because it is not capable knowing what happens with all data, when the data is passed to another component. The main purpose of model is to prepare data and pass it to the controller. However, model differs from database. Model works like data guard, which accepts all requests and processes them. In most cases this part of MVC pattern is the most difficult compared to other parts [4] [5].

View is the result of rendered data, which comes from the model. Usually, view is a part of HTML. For example, button in a webpage is generated by view. When user clicks the button controller action is triggered. Most common mistake done by developers is the absence of connection between view and model. In that case all data comes from the controller which breaks all principals of the MVC pattern [4].

Last part of MVC pattern is controller. Main task for this part is to process data which comes from the user and accordingly update model. Controller is the only part, which interacts with the user. Without user actions controller would not make sense. Controller is like data assembler, which pass data to the model. Controller can have only logic which appoints to data collection. Besides, controller is connected to one view and model only to ensure one directional data passing.

I. Related Work

In 2013 research to compare two PHP frameworks has been conducted. During the research three different versions of the same system was created. First version was in plain PHP (with no framework), other two was developed in CakePHP and CodeIgniter. Those systems was created using Eclipse PDT IDE. All versions were published in local WAMP server. Applications were compared in three different characteristics: code lines number, speed and security [2].

Total lines of code were determined using CLOC (Count Lines of Code) software. Looking at the results (table I), authors claim that using plain PHP code is not reusable and all needed libraries should be coded by the developer. So it means that without using any framework it requires to write much more code. Frameworks are using a lot of libraries with built-in functions, which makes code cleaner. Comparing CakePHP with CodeIgniter it is seen that CakePHP requires less coding, because it has built-in system, which connects database to the model. Writing less code means faster system development, better maintainability and improved security [2].

TABLE I. Total code lines of each application

	PHP	CSS	PHP files	CSS files
Plain PHP	3002	650	42	13
CodeIgniter	2321	621	36	13
CakePHP	1408	680	16	13

To test applications speed XDebug was used. This tool provides functionality like function trace, memory distribution and time to perform tasks. Speed results are introduced in table II. From the results it can be stated that plain PHP has huge advantage compared to frameworks. For example to complete login/logout operation for plain PHP code it takes only 96 milliseconds, while using CodeIgniter 2291 milliseconds. Authors claim that CodeIgniter is faster than CakePHP because most of the operations are executing faster with CodeIgniter. Frameworks are working slower because they need to load a lot of libraries while executing operations and it takes time.

TABLE II. Speed comparison of each application

Functionality	Plain PHP (ms)	CodeIgniter (ms)	CakePHP (ms)
Login/Log-Out	96	2291	1559
Session management	91	739	1309
Database	143	762	1492
Forms usage	487	1338	2262
Forms validation	321	966	2411

To compare applications security, Nessus tool was selected. According to results CakePHP has better security system than CodeIgniter and plain PHP. CakePHP framework has built-in security components, which grants protection from main web vulnerabilities. While CodeIgniter framework has similar protection system (CSRF and XSL protection), but during the tests it does not protect from SQL injection attack.

Similar research was arranged in Bleking Technology Institute, Sweden in 2012 by student. Same frameworks (CakePHP, CodeIgniter) were compared [6]. Main purpose of this analysis was to compare speed of PHP frameworks. After all, six tests was released, three of them were using database. There were 20000 requests completed to Apache server. During the tests (using database) frameworks selection (in each request) was done randomly. Results are shown in table III. According to results CakePHP is a little bit faster (by 5.34 milliseconds) than CodeIgniter [6].

TABLE III. Tests using database

TABLE I. TESTS USING DATABASE

	Response times (ms)	
	CodeIgniter	CakePHP
1 test	227	216
2 test	215	211
3 test	209	206
Average	217	211.6

Other three tests were executed without database (table IV). Results show that response time using frameworks is almost the same. CakePHP was slightly faster (1.3 millisecond) than CodeIgniter. So in both tests results are very similar, CakePHP is a little bit faster than CodeIgniter framework.

TABLE II. TESTS WITHOUT DATABASE

	Response times (ms)	
	CodeIgniter	CakePHP
1 test	233	233
2 test	230	229
3 test	229	226
Average	230.66	229.33

Comparing all six frameworks on their available functionality, table V with results are presented. Results reveal that ORM is available on all frameworks except Zend. Authentication is not implemented only in CodeIgniter framework. Popular functions like cache and validation is available in all compared frameworks. All frameworks supports PHPUnit testing tool, but Yii framework also use Selenium for testing. Looking at the other features, Symfony2 have advantage, because it has layers and CRUD generation components. Furthermore, together with Laravel, these frameworks support logging functionality. Talking about security, only Zend and Yii frameworks does not have protection against web vulnerabilities. In general Symfony2 and Laravel frameworks have most of the features. In the other hand Zend framework is in the last place [7].

One way to compare PHP programming language frameworks is their popularity. For that purpose Google Trends tool was used [8]. This tool can show how many requests by keyword was in search engine. According the results oldest framework is Zend, because in 2004 first requests with this keyword appeared. Besides that Zend framework had reached biggest popularity among all frameworks in the end of 2009. But from that time this framework popularity was going down, so now it is the last framework by this characteristic (16% of best framework search). Fifth place is taken by CakePHP framework, which searching numbers starting to grow in middle of 2005 and in 2010 reaches maximum, now there is 18% of popularity among frameworks. Yii framework is in fourth place with 21% of popularity. This framework is quite new, had an increasing popularity since 2009 and maximum was reached in 2014. Third place between PHP frameworks goes to by Symfony2 which has 30% of popularity. Symfony2 framework started to grow in 2006 and in 2010 reached maximum, from that day it has stable popularity among users. CodeIgniter with 50% is in second place. The most popular framework of the search engine is Laravel. This framework is strong leader of this characteristic. Besides that it is newest framework (started to grow in 2012).

TABLE III. FRAMEWORK FEATURES

	Cake PHP	**Yii**	**Zend 2**	**Symf. 2**	**Larav.**	**CodeI.**
Latest version	3.2	2.0	2.5.2	3.0	5.2	3.0.4
ORM	+	+	-	+	+	+
Authorization	+	+	+	+	+	-
Cache storage	+	+	+	+	+	+
Dependency injection	-	-	+	+	+	-
Code generation	+	+	-	+	+	-
Testing library	+	+	+	+	+	+
Logging management	-	-	-	+	+	-
XSS	+	-	-	+	+	+
XSRF	+	-	-	+	+	+
SQL injection	+	-	-	-	+	+
Template system	+	+	-	+	+	+
Services	+	+	-	-	+	+
CRUD Generation	-	-	-	+	-	-

I. METHODOLOGY OF THE RESEARCH

To compare PHP language frameworks web-based tool was created. It was developed using PHP programming language with no framework. In general, this web application make request to pages that are created using PHP framework and collects the results. All results are calculated (taking average) and compared in one graph for each characteristic. The comparison tool has a feature to add custom data table with custom data. This data can also be generated automatically. After user has completed the customization and preparation of his own data, PHP framework comparison tool starts sending subsequent requests and collecting the result data. During this research custom data table was created. Data table consist of five columns: first name (string), last name (string), age (integer), birthday (date), married (bool). To maintain realistic situation about 300000 records was inserted into table.

Another seven applications were created for PHP frameworks (one for plain PHP). It is simple one page applications which render page with special hidden inputs with data needed for comparing those frameworks. Those inputs have page characteristic data like loaded classes, code lines and cyclomatic complexity. Each framework has the same functionality. It gets data from database and renders that data as table in page. Of course, each PHP framework code is different,

functionality. It gets data from database and renders that data as table in page. Of course, each PHP framework code is different, because of a functions available for a particular framework. For example, some of the frameworks supports query builder for SQL, others don't. All applications are hosted on the local XAMPP server. Local environment has following characteristics: Windows 10 education OS, 16 gigabytes of RAM, 2.5 gigahertz quad core CPU. To complete all functionality two databases were used. One for main application (compares PHP frameworks), which store user information also PHP frameworks information used for analysis. Other database is for remaining applications, it is storing demo user data. All frameworks are compared by four characteristics.

First one is speed. This characteristic was selected because of importance of fast loading pages these days. Performance of page loading is essential characteristic, because users don't want to wait too long. Speed characteristic is used to define how fast page is rendered using one of PHP programming language framework. To get this information main application is initiating JavaScript Ajax call to a page. Time intervals between request and response are measured. Usually, to make good comparison of PHP frameworks a significant amount of measurements (in this case, that amount was 10) need to be collected to avoid accidental results. All response times are added and average is taken.

Next characteristic is an amount of loaded classes needed to render the page. This characteristic purpose is to define how many classes from PHP framework libraries are used to complete various kinds of actions. This characteristic is highly related with speed, because using more classes for the page rendering, means longer time to load all files. Moreover, number of classes could give some insights to a scale of functionality available to the user.

Other characteristic is an amount of code lines per page. It's very useful characteristic especially for developers, because it's good to know how many code lines is required to perform some functionality. The purpose of this characteristic is to define total amount of PHP code lines written to perform operation, in this case to render the page. Code lines also show a code complexity, because larger amount of lines mean higher complexity and more complex maintainability.

Last characteristic is a cyclomatic complexity. This characteristic is related to all different paths that can be executed within a function. This characteristic is very handy to define complexity of the code. Besides, other conclusions or recommendations can be derived from this metric. For example, greater cyclomatic complexity means harder testing of application.

I. Results

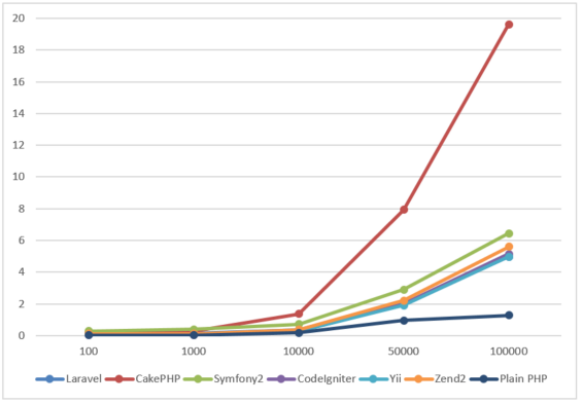
Speed characteristic measured for a custom applications based on all frameworks is shown in table VI. Results are provided in five different data record sizes, where one record size corresponds the amount loaded in one page and inserted into table. Results indicate that the fastest method to render data can be achieved without a framework. Despite this, the best results in terms of speed were achieved by Laravel and Yii frameworks. While CakePHP took the longest period in time to perform the same tasks. Other frameworks had similar results.

TABLE I. SPEED PER REQUEST ON EACH APPLICATION MEASURED IN MILLISECONDS

Name	Data amount				
	100	***1000***	***10000***	***50000***	***100000***
Laravel	0.136	0.138	0.353	1.988	4.982
CakePHP	0.196	0.281	1.374	7.94	19.626
Symfony2	0.289	0.394	0.711	2.889	6.435
CodeIgniter	0.036	0.054	0.272	1.986	5.138
Yii	0.062	0.072	0.264	1.913	4.967
Zend2	0.106	0.121	0.364	2.204	5.603
Plain PHP	0.017	0.031	0.186	0.949	1.282

Judging from the results at 100, 1000 and 10000 data records all frameworks has quite similar results. But when data is growing to 100000 records CakePHP starts working really slow (fig. 3). So in general, most stable results were presented by Laravel and Yii frameworks and of course plain PHP has best speed characteristics.

Fig. 1. Dependency between speed characteristic (in milliseconds) and



amount of data

Comparison of other three characteristics of PHP frameworks are shown in table VII. According to the results, the most lines of code are used by plain PHP and Zend2 frameworks. Leaders in this characteristic with five lines of code are Laravel and CakePHP. In case of the measurement of loaded classes, it is clear that the largest amount of classes needed for Symfony2 framework. Plain PHP is loading default PHP classes only and therefore requires a small amount of classes to be loaded.

TABLE II. PHP FRAMEWORK CHARACTERISTIC COMPARISON

TABLE I. PHP FRAMEWORK CHARACTERISTIC COMPARISON

Name	Code lines	Loaded classes	Cyclomatic complexity
Laravel	5	302	2
CakePHP	5	270	2
Symfony2	7	438	2
CodeIgniter	6	176	2
Yii	13	212	2
Zend2	18	323	4
Plain PHP	23	159	4

Most of the frameworks have similar value for cyclomatic complexity that means code simplicity in that applications. This is good thing when it comes to maintainability and testing. Zend2 framework and plain PHP have a bit higher cyclomatic complexity than other frameworks.

CONCLUSIONS

After analyzing the results collected from the local server, the following conclusions can be drawn:

1. In terms of popularity and the richness of features Laravel can be distinguished as a leading framework.
2. Experimental evaluation (speed, loaded classes, lines of code and cyclomatic complexity) gave different values to various features of a distinct frameworks. An in-depth study of these results need to be performed in order to choose the most suitable framework for a particular situation. In general, Laravel outperformed other frameworks and appeared to a leader in a majority of the characteristics.
3. Theoretical comparison of the frameworks is very similar to practical research results. So it is fair to say that most popular framework by people search on Google is also best in other characteristics.
4. As expected, without using any framework has advantages such as faster request processing, but on the downside making application without framework requires better programming skills, effort and security understanding since there is no built-in security functions.

There are several additional features to be added to the framework comparison tool in the future:

1. Creating code-blocks for each framework with a similar functionality and the same naming convention while trying to find the most effective solution for each framework.
2. Introducing additional metrics and stress testing with a large number of concurrent connections for each application created with different framework.

REFERENCES

[1] World Wide Web Technology Surveys, online access: http://w3techs.com/

[2] L. Lancor and S. Katha, "Analyzing PHP frameworks for use in a project-based software engineering cours cte", Proceeding of the 44th ACM technical symposium on Computer science education, 2013.

[3] W. Cui, L. Huang, L. J. Liang, J. Li, "The Research of PHP Development Framework Based on MVC Pattern", Conference on Computer Sciences and Convergence Information Technology, IEEE Computer Society, 2009.

[4] C. Hopkins, "The MVC pattern and PHP", 2013.

[5] C. Supaartagorn, "PHP Framework for database management based on MVC pattern", Department of Mathematics Statistics and Computer, 2011.

[6] H.Nylen, „PHP Framework Performance for Web", Computer Sciences and Convergence Information Technology, 2012.

[7] „PHP frameworks comparison",online access: http://socialcompare.com/en/comparison/php-frameworks-comparison

[8] Google Trends, online access: https://www.google.com/trends/