электронный научно-технический журнал
ИНЖЕНЕРНЫЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51036. ISSN 2307-0595

Архитектура и программная реализация системы для оценки качества Парето-аппроксимации в задаче многокритериальной оптимизации

 # 12, декабрь 2014

Грошев С. В., Сабитов Д. Р.

УДК: 519.6

Россия, МГТУ им. Н.Э. Баумана
sgro@newmail.ru

Введение

Задача многокритериальной оптимизации возникает в различных областях науки и техники, и предложено несколько различных способов ее решения. Один из способов основан на построении конечномерной аппроксимации Парето-оптимального множества решений.

Существуют различные методы построения Парето-аппроксимации [1], а также способы ее оценки [2, 3]. Реализации этих методов можно найти в различных программных комплексах, которые не предоставляют удобный интерфейс для расчетов и анализа [4]. С целью преодоления недостатков существующих систем, предложена веб-ориентированная программная система для оценки качества Парето-аппроксимации. Эта система призвана решать проблемы консолидации алгоритмов и индикаторов качества аппроксимации, а также предоставляет понятный интерфейс для работы пользователя. Общие принципы работы такой системы рассмотрены в [5]. В данной работе рассматривается архитектура и программная реализация системы.

1. Постановка задачи на разработку программной системы

Целью является разработка масштабируемого веб-приложения для создания и анализа Парето-аппроксимаций, которое должно организовывать одновременную работу многих пользователей, с возможностью одновременного проведения расчетов.

Приложение должно иметь системы аутентификации и регистрации пользователей. Также необходимо обеспечить создание расчета Парето-аппроксимации, расчет индикаторов качества и статистического анализа результатов расчетов. Приложение должно интегрировать различные расчетные модули максимально независимо от языка и применяемых

программных платформ. Кроме того, приложение должно предоставлять пользователям возможность хранить результаты своих расчетов и анализов. Для управления данными в приложении необходимо реализовать панель администратора.

Поскольку разрабатываемое приложение является веб-приложением, логично использовать клиент-серверную архитектуру. При этом клиентом выступает браузер пользователя, а сервером — веб-сервер [6]. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети.

Поскольку разрабатываемая система должна быть многопользовательской и связанной с большим количеством расчетов, можно сделать предположение, что архитектура такой системы должна соответствовать крупным веб-системам. Проведенный анализ применения веб-приложений показывает, что крупнейшие веб-сервисы строят свою архитектуру на основе открытого программного обеспечения [7]. С ростом этих веб-сервисов появились передовые методики управления и развития их архитектуры.

На начальном уровне создание и управление масштабируемым веб-сервисом или приложением это просто соединение пользователей с удаленными ресурсами через Интернет. А ресурсы или доступ к этим ресурсам, которые рассредоточены на множестве серверов и являются звеном, обеспечивающим масштабируемость сервиса. Понимание методик и компромиссов, применяемых на крупных проектах, может принести плоды в виде грамотных решений при создании меньших веб-сервисах.

При рассмотрении архитектуры системы необходимо определить, какие компоненты стоит использовать, как они совмещаются друг с другом и на какие компромиссы можно пойти. При проектировании масштабируемой системы, бывает полезным разделить функциональность и подумать о каждой части системы как об отдельной службе с четко определенным интерфейсом. Считается, что системы разработанные таким образом имеют *Service-Oriented Architecture* (*SOA*) [8]. Для этих типов систем у каждой службы существует свой собственный отличный функциональный контекст, и взаимодействие с чем-либо за пределами этого контекста происходит через абстрактный интерфейс, обычно общедоступный *API* другой службы.

Разбиение системы на ряд независимых сервисов изолирует работу одних частей от других. Эта абстракция помогает устанавливать четкие отношения между службой, ее базовой средой и потребителями службы. Создание четкой схемы может помочь локализовать проблемы, но также и позволяет каждой части масштабироваться независимо друг от друга. Этот вид сервис-ориентированного проектирования систем для обслуживания широкого круга запросов аналогичен объектно-ориентированному подходу в программировании.

2. Проектирование серверной части веб-приложения

Сервер веб-приложения должен осуществлять следующие функции:

- обрабатывать *HTTP* запросы клиентов и отдавать *html*-документы в ответ;

- хранить данные;
- производить вычисления и анализ Парето аппроксимаций.

По этим функциональным требованиям удобно разделить серверную часть на несколько подсистем: *MVC*-приложение, система управления базами данных (СУБД) и сервер обработки расчетов (рис 1).

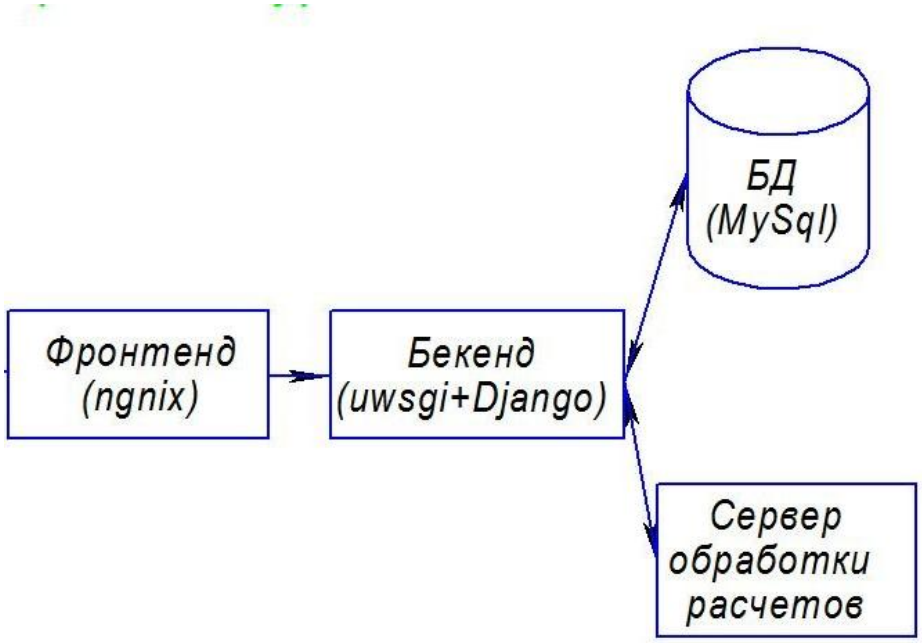

Рис. 1. Архитектура серверной части приложения

Шаблон проектирования «модель – представление – контроллер» (MVC: *Model – View – Controller*) используется длительное время [9]. Концепция MVC позволяет разделить данные, представление и обработку действий пользователя на три отдельных компонента, что представлено на Рис. 2.

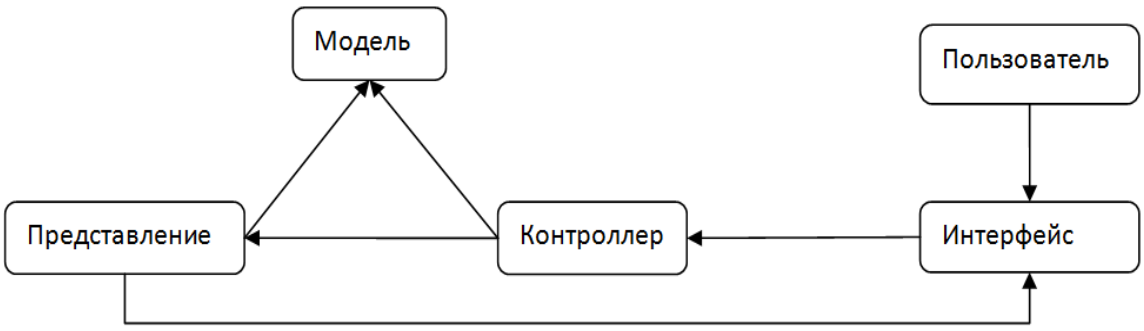

Рис. 2. Структура MVC

Модель предоставляет знания: данные и методы работы с этими данными, реагирует на запросы, изменяя своё состояние. При этом информации о том, как эти знания можно визуализировать в модели не содержится.

Представление, отвечает за отображение информации (визуализацию). Часто в качестве представления выступает форма (окно) с графическими элементами.

Контроллер обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции.

За счет такого разделения повышается возможность повторного использования. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах или с различных точек зрения.

Пользователь, работая с интерфейсом, управляет контроллером, который перехватывает действия пользователя. Далее контроллер уведомляет модель о действиях пользователя, тем самым изменяя состояние модели. Контроллер также уведомляет представление. Представление, используя текущее состояние модели, строит пользовательский интерфейс.

На данный момент паттерн *MVC* реализован в том или ином виде для большинства языков программирования используемых для разработки веб-приложений. Самое большое количество реализаций имеет *PHP*, но и для *Java, Perl, Python, Ruby* есть свои варианты.

MVC-приложение работает под управлением веб-сервера. Существует множество веб-серверов, которые можно разделить по двум моделям работы: многопоточная модель и асинхронная модель.

Многопоточная модель имеет широчайшее распространение. Приложение создает некоторое количество потоков (пул), передавая каждому из них задачу и данные для обработки. Задачи выполняются параллельно. Если потоки не имеют общих данных, то не будет издержек на синхронизацию, что делает работу достаточно быстрой. После завершения работы поток не завершает работу, а лежит в пуле, ожидая следующей задачи. Это убирает накладные расходы на создание и удаление потоков. Каждый скрипт работает в своем потоке. Один поток обрабатывает один запрос. Потоков достаточно большое количество, медленные запросы забирают поток надолго, быстрые — обрабатываются почти мгновенно, освобождая поток для важной другой работы. Это не позволяет очень медленным запросам забирать все доступное процессорное время, заставляя ждать прерываний быстрые запросы.

Асинхронная модель построена на очереди событий (*event-loop*). При возникновении некоторого события (пришел запрос, выполнилось считывание файла, пришел ответ от БД) оно помещается в конец очереди. Поток, который обрабатывает эту очередь, берет событие из начала очереди и выполняет связанный с этим событием код. Пока очередь не пуста процессор будет занят работой. У нас имеется единственный поток, обрабатывающий очередь событий. Почти все операции неблокирующие.

Каждый запрос может выполняться несколько медленнее в целом, но в единицу времени мы можем обработать гораздо больше запросов. Общая производительность будет выше. Процессор всегда будет занят полезной работой. При этом на обработку очереди и переходе от события к событию тратится гораздо меньше времени, чем на переключение между потоками в многопоточной системе. Поэтому асинхронные системы с неблокирующими операциями должны иметь не больше потоков, чем количество ядер в системе. Многопоточная система с блокирующими операциями имеет большое время простоя. Чрезмерное количество потоков может создать много накладных расходов, недостаточное же количество может привести к замедлению работы при большом количестве медленных запросов. Асинхронное приложение с неблокирующими операциями использует процессорное время эффективнее, но более сложно при проектировании.

В разрабатываемом веб-приложении предполагается использовать как веб-сервер с многопоточной моделью, так и с асинхронной. Первым запрос от клиента будет получать веб-сервер с асинхронной моделью обработки, также называемый фронтендом (эталонным примером такого веб-сервера является *nginx*) [10]. Если при формировании ответа требуется обращения в СУБД или какие-либо вычисления, то запрос передается на второй веб-сервер (бекенд сервер), где запущенно *MVC*-приложение. Описанная трехзвенная архитектура представлена на Рис 3.

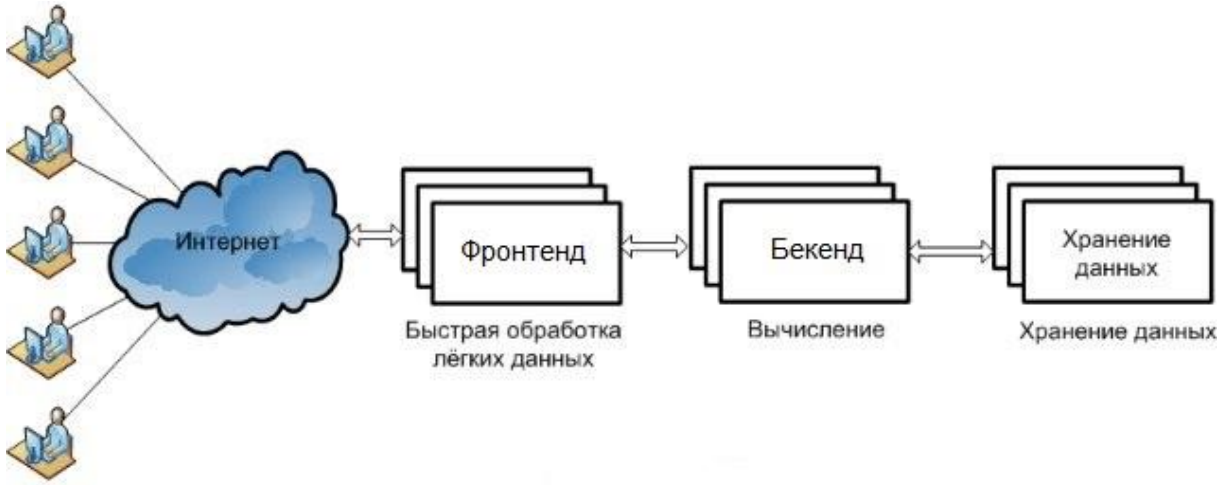

Рис. 3. Трёхзвенная архитектура

Таким образом, фронтенд сервер решает проблемы отдачи статического контента, буферизации запросов, масштабирования бекендов, а также обслуживания медленных клиентов.

Проектируемой системе необходимо сохранять множество данных: информацию о пользователях, результаты расчетов и анализов. Для этого на сервере понадобиться СУБД [11].

MVC-приложение оперирует объектами, а СУБД обычно оперирует таблицами. Отсюда возникает задача в преобразовании таких объектов в форму, в которой они могут быть сохранены в файлах или базах данных, и которые легко могут быть извлечены в последующем, с сохранением свойств объектов и отношений между ними. Эти объекты называют «хранимыми» (*persistent*).

Использование реляционной базы данных для хранения объектно-ориентированных данных приводит к семантическому разрыву, заставляя программистов писать программное обеспечение, которое должно обрабатывать данные в объектно-ориентированном виде, а также сохранять эти данные в реляционной форме. Эта постоянная необходимость в преобразовании между двумя разными формами данных не только сильно снижает производительность, но и создает трудности для программистов, так как обе формы данных накладывают ограничения друг на друга.

Во избежание подобных проблем, используется технология программирования *object-relational mapping* (*ORM*) [12]. Суть технологии заключается в создании виртуальной объектной базы данных. Разработано множество *ORM* решений, устраняющих необходимость в преобразовании объектов для хранения в реляционных базах данных. Некоторые пакеты решают эту проблему, предоставляя библиотеки классов, способных выполнять такие преобразования автоматически. Имея список таблиц в базе данных и объектов в программе, они автоматически преобразуют запросы из одного вида в другой.

ORM избавляет программиста от написания большого количества кода, часто однообразного и подверженного ошибкам, тем самым значительно повышая скорость разработки. Кроме того, большинство современных реализаций *ORM* позволяют программисту при необходимости самому жёстко задать код *SQL*-запросов, который будет использоваться при тех или иных действиях (сохранение в базу данных, загрузка, поиск) с постоянным объектом.

Сервер обработки расчетов выделен как отдельный функциональный элемент архитектуры приложения. Он предназначен для обработки задач по расчету и анализу Парето аппроксимаций. Сами расчеты производятся программами, которые могут быть написаны на различных языках программирования. Сервер обработки расчетов должен осуществлять взаимодействие с исполняемыми файлами расчетных модулей и файлами с данными.

Расчеты могут выполняться длительное время и создавать серьёзную нагрузку на сервер, а это значит производить расчеты на том же сервере, где работает *MVC*-приложение нельзя. Взаимодействие сервера обработки расчетов и *MVC*- приложения происходит при помощи *API.*

Обработка расчетов происходит следующим образом. *MVC*-приложение посылает запрос, в котором передается тип расчета и данные. Затем сервер обработки проверяет необходимые параметры, и если проверка проходит успешно, сервер расчетов отдает ответ *HTTP* 200 и отложено выполняет расчет. Когда расчет завешен, сервер отдает результаты в *MVC*-приложение. При такой схеме работы клиент не ждет окончания расчетов, так как они могут занимать много времени.

Сервер обработки расчетов должен состоять из веб-сервера, модуля работы с *API*, модуля работы с файлами. Такая структура позволит обеспечить гибкость взаимодействовать с программами расчета и анализа. Расчетный сервер является независимым элементом, это важное условие позволит легко решить задачу горизонтального масштабирования. При высокой загрузки одного сервера расчетов необходимо установить дополнительные копии приложения.

3. Проектирование клиентской части веб-приложения

Традиционным клиентом веб-приложения является браузер. Основное предназначение браузера – отображать веб-ресурсы. Для этого на сервер отправляется запрос, а результат выводится в окне браузера. Способы обработки и отображения *HTML*-файлов определены спецификациями *HTML* и *CSS*. Они разрабатываются Консорциумом *W3C*, который внедряет стандарты для Интернета.

Каскадные таблицы стилей (*CSS*) – формальный язык описания внешнего вида документа, написанного с использованием языка разметки. *CSS* используется создателями веб-страниц для задания цветов, шрифтов, расположения отдельных блоков и других аспектов представления внешнего вида этих веб-страниц. Основной целью разработки *CSS* являлось разделение описания логической структуры веб-страницы, которое производится с помощью *HTML* или других языков разметки, от описания внешнего вида этой веб-страницы, которое теперь производится с помощью формального языка *CSS*. Такое разделение может увеличить доступность документа, предоставить большую гибкость и возможность управления его представлением, а также уменьшить сложность и повторяемость в структурном содержимом. Кроме того, *CSS* позволяет представлять один и тот же документ в различных стилях или методах вывода, таких как экранное представление, печатное представление.

Вычисления, которые необходимо выполнить на стороне клиента, в разрабатываемой системе выполняются при помощи *JavaScript*. Этот язык обычно используется как встраиваемый для программного доступа к объектам приложений [13]. Ядро *JavaScript* содержит набор основных объектов и основной набор элементов языка. Ядро *JavaScript* может быть расширено для различных целей путём дополнения его новыми объектами

JavaScript позволяет делать *AJAX* запросы. *AJAX*-подход к построению интерактивных пользовательских интерфейсов веб-приложений заключается в фоновом обмене данными браузера с веб-сервером [14]. В результате, при обновлении данных веб-страница не перезагружается полностью. На рис. 4 представлена модель работы с применением *AJAX*.

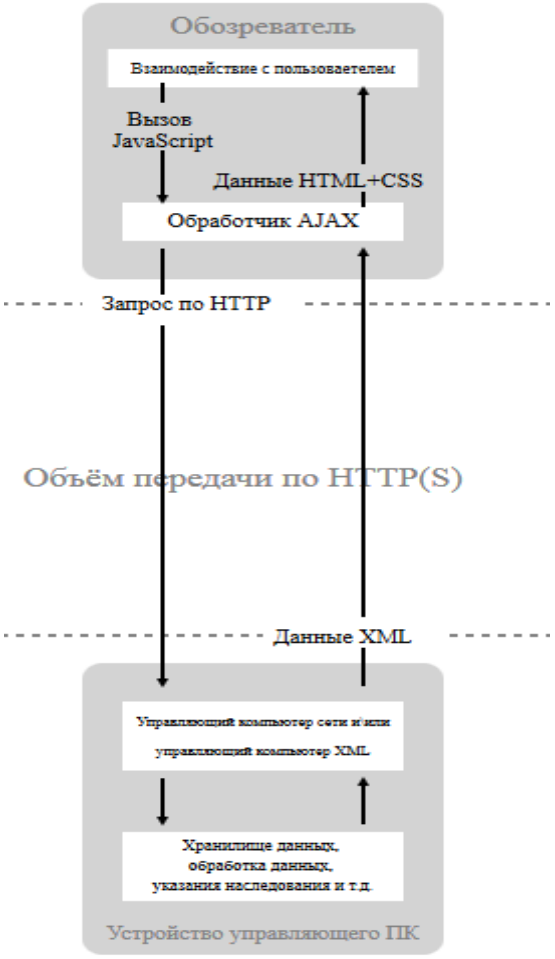

Рис. 4. Запрос с применением Ajax

AJAX — не самостоятельная технология, а концепция использования нескольких смежных технологий. Заключается в использовании технологии динамического обращения к серверу «на лету», без перезагрузки всей страницы полностью.

4. Программная реализация системы Pareto-Q

Клиентом разрабатываемого приложения является браузер. Чтобы сделать универсальное клиентское приложение, которое будет правильно отображаться и работать в различных браузерах, проектируемое веб-приложение будет использовать фреймворки, которые учитывают особенности браузеров различных производителей.

Для *CSS* верстки использовался *Bootstrap* 2.3.2, а для *JavaScript* применяется *jQuery* 1.11.1. Использование в проекте этих фреймворков гарантирует корректную работу в популярных браузерах, при этом существенно экономится время на разработку клиентской части.

В проекте PARETO-Q *Bootstrap* используется при верстке всех основных страниц. *HTML* документ разделен на несколько частей: заголовок, основная часть и нижнее меню.

В заголовке присутствует название и небольшое меню пользователя, что продемонстрировано на Рис. 5.

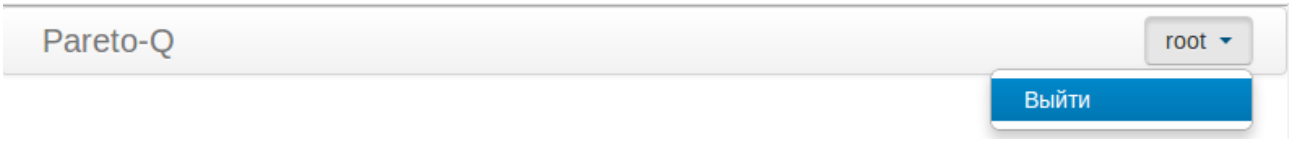

Рис. 5. Заголовок и меню проекта

Чтобы не отрывать пользователя от контекста, в разрабатываемой системе используется поддержка модальных окон.

Основными сущностями проекта являются: анализ, расчет и оценка. Они имеют много различных параметров и при их выводе целесообразно использовать таблицы. На Рис. 6 представлен пример таблицы с расчетом.

Название поля	Значение поля
Название	sdfsefse
Дата	18-05-2014 19:12:22
Время вычислений	0
Алгоритм	ibea
Тестовая задача	zdt2
Фронт	
Индикаторы	
Вывод	

Рис. 6. Пример таблицы с расчетом

В клиентской части приложения, пользователь должен как можно реже отвлекаться от контекста – как можно реже должна перезагружаться страница, при этом работу с сервером осуществляется в фоновом режиме при помощи *AJAX*. При работе с *AJAX* удобно использовать различные библиотеки, такие как *jQuery*.

Расчеты в системе могут выполняться длительное время, и получить результаты сразу после создания проекта невозможно. Если бы не было фоновых запросов к серверу, пользователю приходилось бы в ожидании результата каждый раз перезагружать страницу. Но средствами *jQuery* мы можем вывести результат, как только расчетный сервер пришлет ответ. Сразу после создания пользователь увидит сообщение о том, что расчет выполняется. Затем, как выполниться запрос, выведется сообщение об успешном окончании, и автоматически загрузятся результаты расчетов.

Все это будет выполнено без перезагрузки страницы, и соответственно без участия пользователя. Данная ситуация продемонстрирована на Рис. 7, Рис. 8.

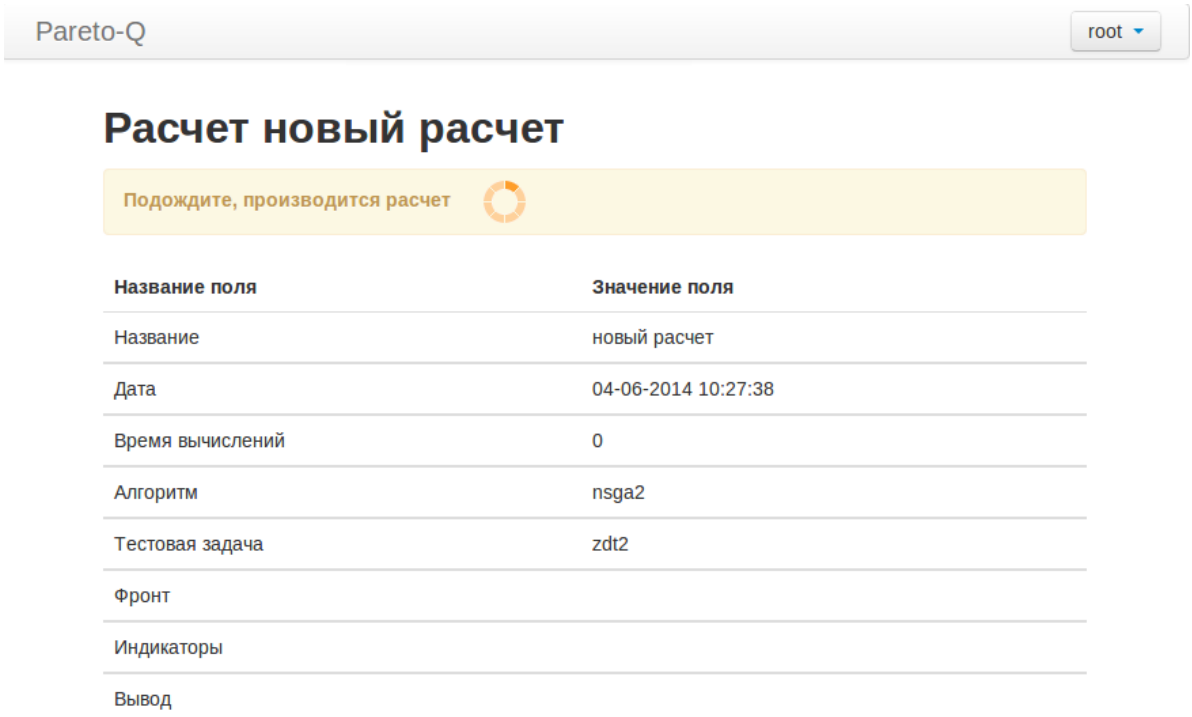

Рис. 7. Выполнение расчета

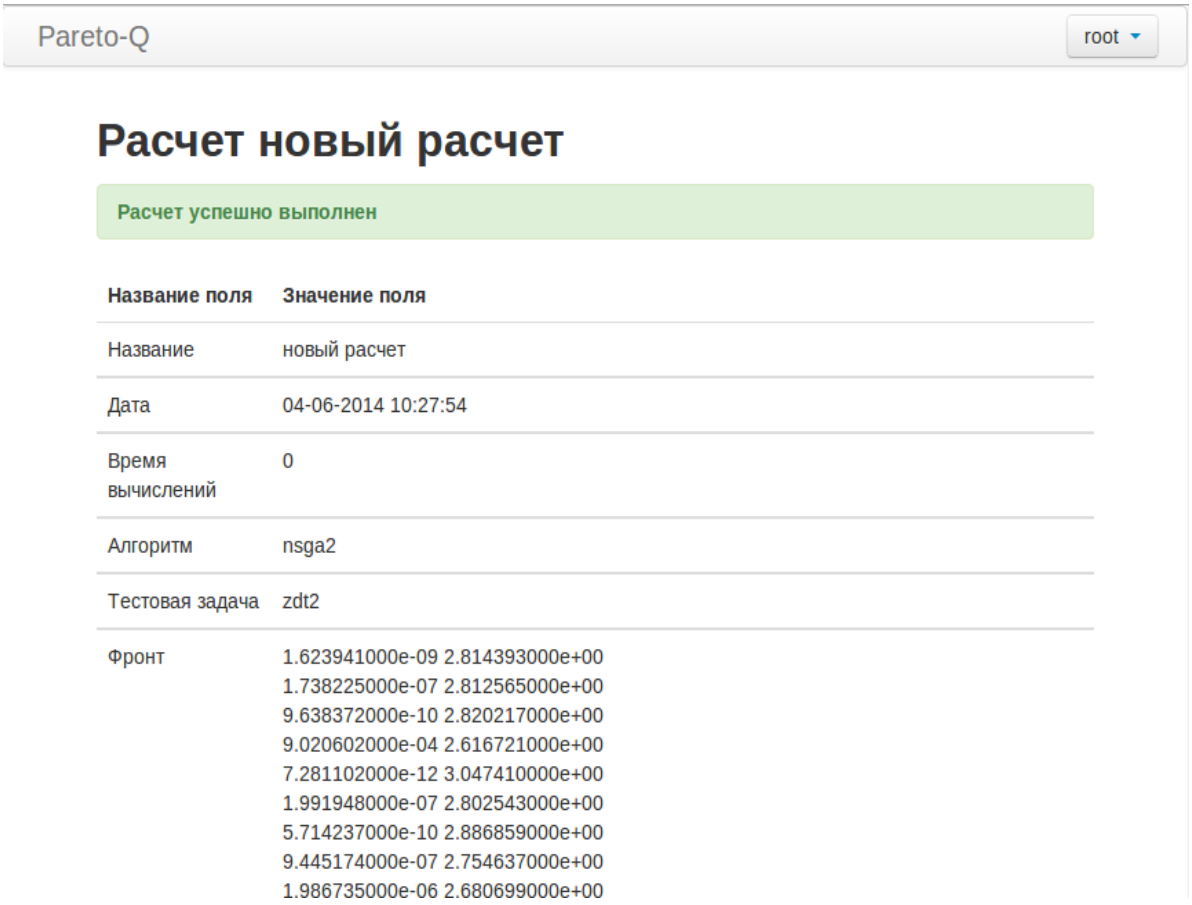

Рис. 8. Результаты расчета

Серверная часть разрабатываемого приложения состоит из нескольких компонентов: *MVC*-приложения и сервера расчетов. *MVC*-приложение должно обрабатывать запросы пользователя, работать с *СУБД* и отдавать ответы в виде *HTML*-документов.

Обычно сервер пишут с использованием языков программирования типа *Perl, Python, PHP* или *Java*. Каждый из этих языков имеет свои преимущества, но, в общем, они используются из-за скорости разработки, так как существует множество готовых библиотек и фреймворков для этих языков. Для разрабатываемом *MVC*-приложения нет необходимости в максимальной производительности. Ожидается, что одновременно с *MVC*-приложением будут работать несколько сот пользователей. Такую производительность может обеспечить один сервер, написанный на интерпретируемом языке с динамической типизацией. В качестве такого языка был выбран *Python*.

На *Python* написан один из самых популярных *MVC* фреймворков *Django*. Контроллер классической модели *MVC* примерно соответствует уровню, который в *Django* называется Представление (*View*), а презентационная логика представления реализуется в *Django* уровнем Шаблонов (*Template*). Для работы с базой данных *Django* использует собственный *ORM*, в котором модель данных описывается классами *Python*, и по ней генерируется схема базы данных.

В разрабатываемом приложении есть несколько основных модулей: *urls.py, views.py, models.py* и *settings.py*. В модуле *urls.py* определяется, какой контролер будет обрабатывать определённый запрос, идентифицируемый по *URL* (*Uniform Resource Locator*) с помощь регулярного выражения. Модуль *views.py* содержит все контролеры необходимые в проекте. Внутри контролеров реализуется логика предметной области, с которой мы работаем. Сущности, которыми оперирует приложение, представлены в модуле *models.py*. Они описываются с помощью классов с соответствующими полями и методами. Настройки необходимые приложению хранятся в виде словаря в модуле *settings.py*.

Когда приходит запрос от пользователя, по *URL* вызывается нужный контролер. Контролер проверяет данные, сохраняет или извлекает данные из СУБД. Затем он вызывает шаблон, где генерируется *HTML*-документ, который в конечном итоге отдается пользователю. Контроллер в *Django* не работает с чистым языком запросов *SQL*, а использует *ORM*, который определяется модулем *models.py*. Работа с *SQL* прямо из контролера часто приводит к такой уязвимости веб-приложения, как *SQL*-инъекция.

Схема базы данных генерируется по классам, представленным в модуле *models.py*. Такой подход не привязывает нас к конкретной СУБД и позволяет при необходимости ее сменить. Также *Django* поддерживает миграции версий СУБД: при изменении какого-либо класса в *models.py, Django* автоматически приведет СУБД в соответствие с новой схемой. Полученная схема базы данных продемонстрирована на Рис. 9.

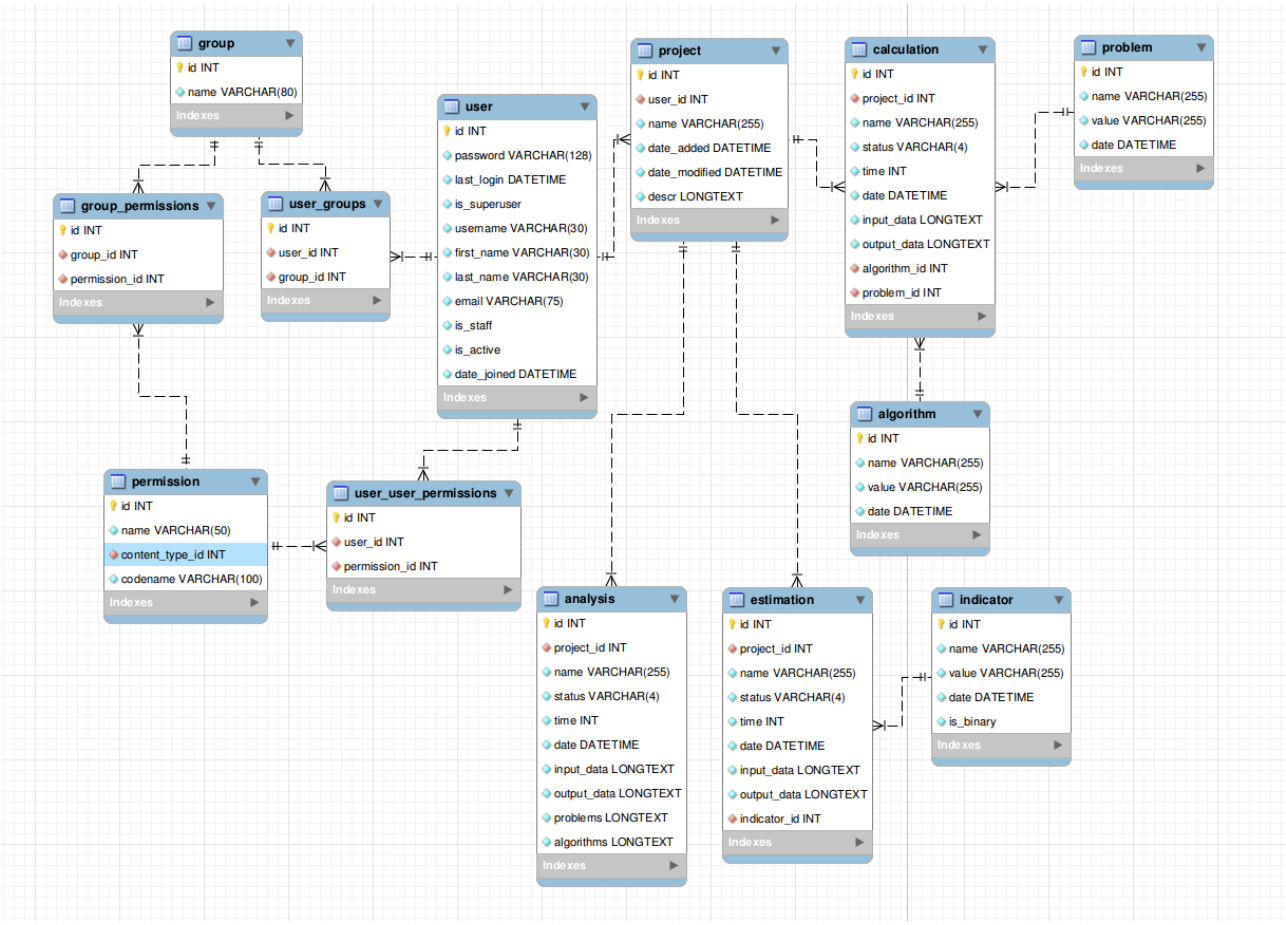

Рис. 9. База данных приложения

В проекте требуется производить различные расчеты фронтов Парето и проводить их анализ. Делать это внутри *MVC*-приложения нецелесообразно, расчеты должны производиться на отдельном сервере, который не влияет на работу веб-приложения. Расчетные сервера должны хорошо масштабироваться, так как с ростом числа пользователей будет серьезно возрастать нагрузка на вычислительную подсистему.

Сервер обработки расчетов представляет собой веб-сервер *Nginx*, который может запускать код на интерпретируемом языке программирования *Lua*. В *Lua* реализовано множество полезных функций: регулярные выражения, ассоциативные массивы и удобная работа с файлами. При всех широких возможностях, *Lua* легковесный язык и поэтому прекрасно подходит для встраивания в различные приложения. Многие веб-сервера, хранилища данных, текстовые редакторы и даже аудио проигрыватели позволяют писать расширения на *Lua*. В сервере обработки расчетов *Lua* используется для реализации логики обработки данных, которые приходят от *MVC*-приложения, и запуска расчетных программ.

В *MVC*-приложении пользователь создает анализ, расчет или оценку, вводя необходимые данные в форму создания. Затем следует запрос к серверу обработки расчетов, где

проверяются входные данные. Если все данные введены верно, то сервер обработки расчетов ставит задачу на расчет в очередь, и отдает ответ об успешной постановки в очередь. В случае если данные не прошли проверку, возвращается ответ с указанием какой из параметров ошибочный.

В *Nginx* есть механизм таймеров, который используется как очередь. Задача расчета выполняется в теле таймера отложено. Единовременно может выполняться один или несколько таймеров, этот параметр определяется в конфигурационном файле *Nginx*. Сервер обработки расчетов конструирует команду для расчета и записывает необходимые данные в файлы. Общение по *HTTP*, проверки входных параметров, чтение и запись файлов реализовано при помощи *Lua*. Когда расчет выполнен, сервер обработки расчетов посылает результаты в *MVC*-приложение при помощи *API*.

API для взаимодействия веб-приложения и сервера обработки расчетов представляет собой обмен *JSON* сообщениями по протоколу *HTTP*. В сообщение всегда присутствуют такие данные как статус операции (*state*), тип сообщения (*type*), идентификатор обрабатываемого расчета, анализа или оценки (*id*). Также передаются дополнительные данные, разные для каждой из сущностей: полученные фронты, вычисленные индикаторы и результаты анализа. Такой подход обеспечивает масштабируемость системы и позволяет ей эволюционировать с новыми требованиями.

Разработка проекта ведется с использованием распределенной системы контроля версий *git*. *Git* позволяет работать нескольким разработчика одновременно и хранить код в удаленном репозитории. Репозиторий проекта расположен по адресу *https://github.com/ziontab/pareto*. Помимо кода проекта в репозитории содержатся настройки и файлы конфигураций.

5. Обзор интерфейса и сценариев работы

Работа с сервисом начинается с процедуры аутентификации. Пользователь входит в проект, вводя логин и пароль. В случае отсутствия учетной записи пользователь проходит регистрацию. Неавторизованным пользователям доступна только общая информация о системе и контакты разработчиков.

Для регистрации необходимо ввести следующие данные (Рис.10):

- имя пользователя;
- *E-mail* пользователя;
- пароль;
- подтверждение пароля.

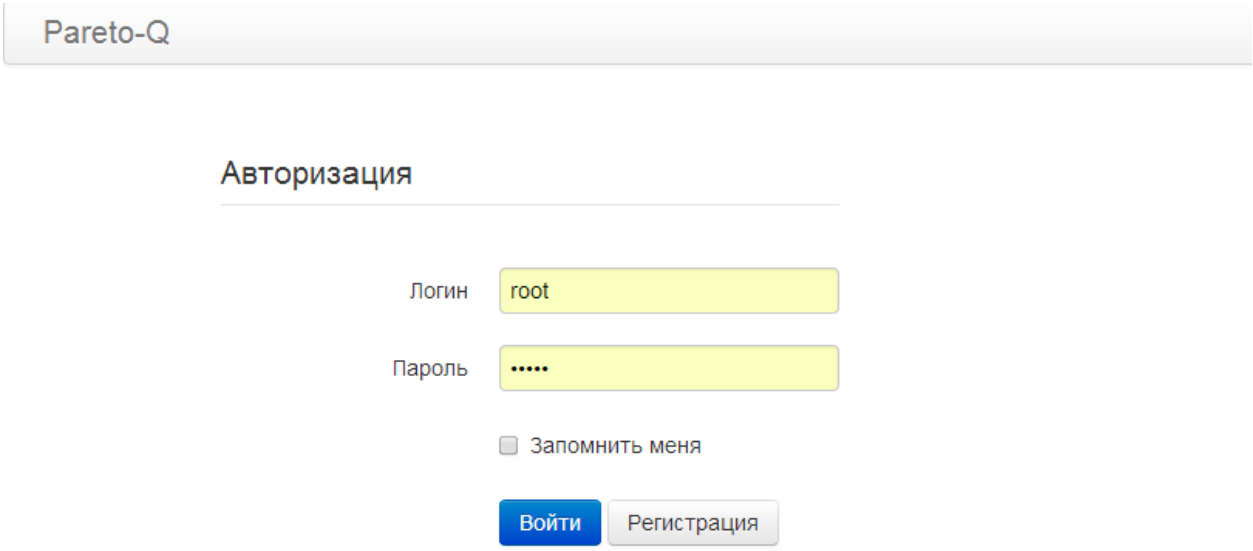

Рис. 10. Страница аутентификации

Система регистрации обрабатывает следующие ошибки: попытка введения уже существующего имени в базе данных пользователей, отсутствие имени вообще в форме ввода, отсутствие пароля в форме ввода, различие введенных паролей. В случае если пользователь введет некорректные данные, система выведет подсказки. При успешной аутентификации пользователь сможет продолжить работу с сервисом и будет перенаправлен на страницу проектов, где приведен список всех проектов, которые создавал текущий пользователь (Рис 11).

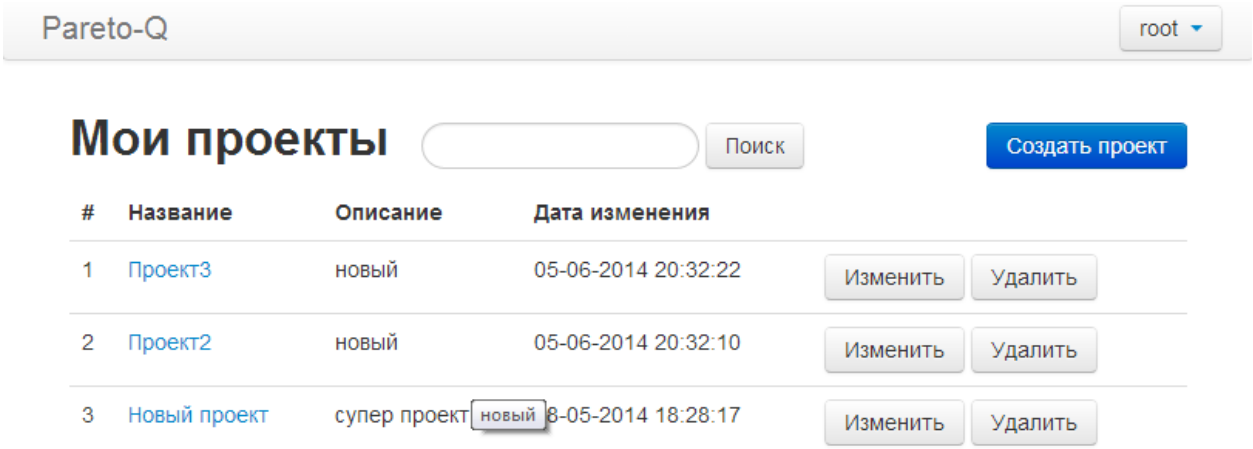

Рис. 11. Страница проектов

На этой странице реализованные следующие возможности.

- Создание нового проекта - для этого достаточно кликнуть на кнопку "создать проект" и ввести имя нового проекта во всплывающем окне.
- Редактирование существующего проекта происходит на отдельной странице. Чтобы на неё попасть, необходимо кликнуть на кнопку "изменить" напротив соответствующего проекта. Возможно изменение имени и описания проекта.
- Удаление существующего проекта по клику на кнопку "удалить" напротив соответствующего проекта.
- Поиск по имена проектов в окне "Найти проект по имени".
- Переключение по страницам проектов. Необходимо для пользователей, у которых создано большое количество проектов.

После того как пользователь нашел необходимый проект и нажал на имя проекта, он попадает на страницу, где имеется три закладки: "расчеты", "анализ" и "оценка". Кликая по этим закладкам, пользователь может увидеть какие оценки, расчеты и анализы проводились в рамках выбранного проекта. Чтобы перейти к подробному просмотру пользователь должен кликнуть на соответствующее имя сущности. Для того чтобы создать новый расчет, анализ или оценку необходимо кликнуть на кнопку "Создать расчет" или "Создать оценку" и пользователь перейдет на соответствующую страницу.

Оценка качества Парето аппроксимации может быть создана различными способами в зависимости от того какие входные представления данных мы имеем. Для создания оценки необходимо ввести следующие данные.

- Имя оценки.
- Выбрать тип оценки бинарный или унарный.
- Выбрать индикатор, в зависимости от выбранного типа оценки в выпадающем списке будут представлены либо унарные, либо бинарные критерии.
- Загрузить файлы фронтов. Если был выбран бинарный критерий, то нужно загружать два фронта. Для унарных критериев необходим один фронт.

Файлы фронтов представляют собой обычные текстовые файлы, где координаты точек перечислены через пробел, а каждая новая точка соответствует новой строчки в текстовом файле. Если оценка была успешно произведена, то пользователь будет автоматически перенаправлен на страницу оценки, иначе будет получено диагностическое сообщение. Страница создания оценки представлена на Рис 12.

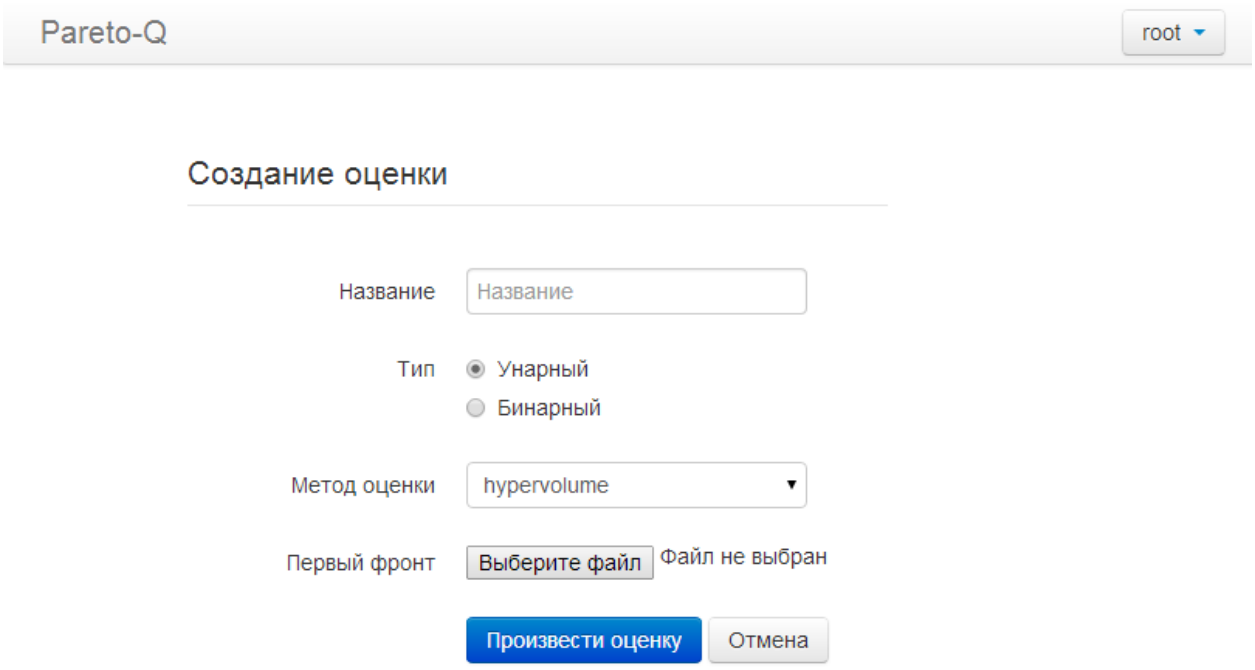

Рис. 12. Страница создания оценки

Создание расчета и анализа происходит аналогично созданию оценки. Различия заключаются в полях формы, которые необходимо заполнить для успешного создания. Вместо типов индикаторов и загрузки фронтов, представлен список тестовых задач и алгоритмов, с помощью которых будет проводиться расчет и анализ фронтов Парето.

В проекте есть панель администратора, необходимая для настройки проекта. Панель администратора доступна по следующему адресу *http://{domain}/admin/*, где *{domain}* – адрес хоста. Панель администратора включает следующие возможности:

- добавление новых пользователей в систему и редактирование настроек уже существующих с широким набором прав разграничения доступа к ресурсам сайта;
- добавление и редактирование групп пользователей;
- просмотр записей из всех таблиц базы данных с возможностью создания новых и редактирования уже существующих;
- на каждой из страниц доступна возможность поиска и фильтрации соответствующих сущностей;
- на основной странице доступен журнал последних изменений, внесенных из панели администратора.

6. Функциональное тестирование системы

Существует множество различных видов тестирования программного обеспечения [15, 16]. Классификация по объекту тестирования имеет следующий вид:

- функциональное тестирование;
- тестирование производительности;
- тестирование эргономичности;

- тестирование безопасности;
- тестирование локализации;
- тестирование совместимости.

В разрабатываемой системе проводится функциональное тестирование. Функциональное тестирование рассматривает заранее указанное поведение и основывается на анализе спецификаций функциональности компонента или системы в целом. Функциональные тесты основываются на функциях, выполняемых системой, и могут проводиться на всех уровнях тестирования (компонентном, интеграционном, системном, приемочном). Преимуществом функционального тестирования является имитирование фактическое использование системы.

Для проекта написаны автоматизированные функциональные тесты с использованием *Selenium WebDriver* - программной библиотеки для управления браузерами [17]. Фактически она представляет собой целое семейство драйверов для различных браузеров, а также набор клиентских библиотек на разных языках, позволяющих работать с этими драйверами, что представлено на Рис 13.

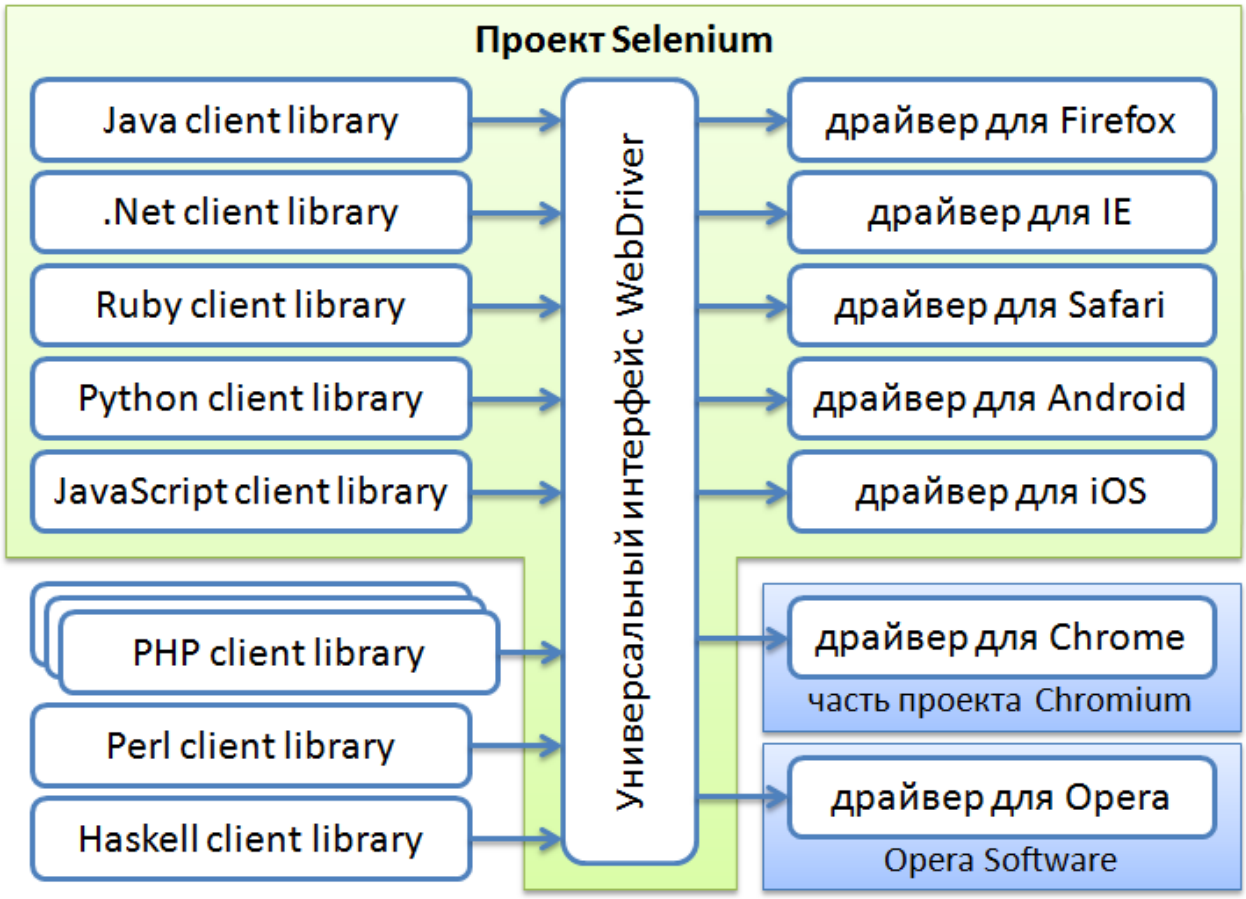

Рис. 13. Структура *Selenium WebDriver*

Автоматизированные тесты проверяют результаты следующих действий пользователя:

- регистрация;
- аутентификация;
- задание проекта;
- создание оценки;
- создание анализа;
- создание расчета.

При этом используется следующие методы *WebDriver*:

- "driver.getPage" - загрузка небходимой страницы по URL.
- "driver.findElement" - поиск элемента на страницы.
- "element.sendKeys" - ввод данных в элемент.
- "element.clear" - очистка данных в теге input.
- "element.getAttribute" - получение атрибуда тега.
- "element.submit" - отправка формы.

Заключение

В работе сформулирована задача на разработку масштабируемой веб-архитектуры для системы оценки качества Парето-аппроксимации. Подробно разобраны способы взаимодействия клиента и сервера. Приведены подходы к проектированию масштабируемых клиент-серверных веб-приложений с использованием трехзвенной архитектуры и очередей для отложенной обработки. Описана схема работы веб-серверов с асинхронной обработкой запросов.

Разработана архитектура программной системы. Описана структура и схема работы всех основных компонент серверной части приложения, включая *MVC*-приложение, СУБД и сервер обработки расчетов. Исходя из функциональных возможностей системы, сервер обработки расчетов был выделен как отдельный архитектурный элемент, необходимый для интеграции расчетных модулей.

Описана программная реализация разрабатываемого проекта. Приведена реализация клиентской части веб-приложения с использованием клиентских фреймворков. Описан процесс разработки серверной части *MVC*-приложения и сервера обработки расчетов. Рассмотрен вопрос взаимодействия между серверами.

Показаны основные элементы интерфейса приложения. Приведен обзор сценариев работы пользователя с проектом.

Показана классификация тестирования веб-приложения по объекту тестирования. Разработаны автоматизированные функциональные тесты с использованием *Selenium WebDriver*.

Список литературы

1. Карпенко А.П., Митина Е.В., Семенихин А.С. Популяционные методы аппроксимации множества Парето в задаче многокритериальной оптимизации // Наука и образо-

вание: электронное научно- техническое издание, 2012, №4, (http://www.technomag.edu.ru/doc/363023.html).
2. Zitzler E., Deb K., Thiele L. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results // Evolutionary Computation, 2000, Vol. 8(2), pp. 173-195.
3. Белоус В.В., Грабик А.В., Грошев С.В., Шибитов И.А. Качество Парето-аппроксимации в задаче многокритериальной оптимизации // XVIII Байкальская Всероссийская конференция «Информационные и математические технологии в науке и управлении» Часть 1.- Иркутск: ИСЭМ СО РАН, 2013.- с. 6-12.
4. Белоус В.В., Грошев С.В., Карпенко А.П., Шибитов И.А. Программные системы для оценки качества Парето-аппроксимации в задаче многокритериальной оптимизации. Обзор. Наука и образование: электронное научно- техническое издание, 2014, № 4, DOI: 10.7463/0414.0709198 (http://technomag.bmstu.ru/doc/709198.html).
5. Грошев С.В., Карпенко А.П., Сабитов Д.Р., Шибитов И.А. Программная система PARETO RATING для оценки качества Парето-аппроксимации в задаче многокритериальной оптимизации**.** Наука и образование: электронное научно- техническое издание, 2014, № 7, DOI:10.7463/0714.0720253 (http://technomag.bmstu.ru/doc/720253.html).
6. Коржов В.А. Многоуровневые системы клиент-сервер // Издательство Открытые системы, 1997, с. 32-54.
7. K. Matsudaira. Scalable Web Architecture and Distributed Systems. Электронный ресурс. http://sysmagazine.com/posts/185636/
8. Service-Oriented Architecture (SOA) Definition. Электронный ресурс. http://www.service-architecture.com/articles/web-services/service-oriented_architecture_soa_definition.html
9. N. Rupp. Part One: History of the MVC Pattern // Beyond MVC: A new look at the Servlet Infrastructure, 2003, pp. 10-15.
10. Бунин О.Е., Лапшин М.В. Разработка высоконагруженных систем // Highload, 2013, Москва, с. 5-56.
11. Коннолли Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика, 2003, с. 520-582.
12. ORM. Электронный ресурс https://ru.wikipedia.org/wiki/ORM
13. J.J. Durillo, A. J. Nebro. jMetal. A Java Framework for Multi-Objective Optimization // Advances in Engineering Software 2011, 42, pp. 760-771.
14. AJAX. Электронный ресурс https://ru.wikipedia.org/wiki/AJAX
15. Котляров В., Коликова Т. Основы тестирования программного обеспечения. Бином. 2009, 288с
16. Джеф Рэшка, Элфид Дастин, Джон Пол Тестирование программного обеспечения. Лори. 2012, 568 с.
17. Selenium WebDriver. Электронный ресурс. http://docs.seleniumhq.org/projects/webdriver/