ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,   
МЕХАНИКИ И ОПТИКИ

Факультет  Информационных Технологий и Программирования

Направление (специальность)  Прикладная математика и информатика

Квалификация (степень)  Бакалавр прикладной математики и информатики

Кафедра  Компьютерных технологий Группа  4539

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Определение сайтов-зеркал на основе анализа графа ссылок и дублирования контента

Автор квалификационной работы Ларионов О. С. (подпись)

Руководитель Романенко А. А. (подпись)

**Консультанты:**

а) По экономике и организации (подпись)

производства

б) По безопасности жизнедея- (подпись)

тельности и экологии

в) (подпись)

**К защите допустить**

Зав. кафедрой Васильев В.Н. (подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2013 г.

Санкт-Петербург, 2013 г.

Оглавление

[Введение 4](#_Toc359261709)

[Глава 1. Основные понятия и обзор методов 5](#_Toc359261710)

[1.1. Основные определения 5](#_Toc359261711)

[1.2. Причины создания зеркал и примеры использования информации о зеркалах 5](#_Toc359261712)

[1.3. Проблемы поиска зеркал 6](#_Toc359261713)

[1.4. Существующие алгоритмы 7](#_Toc359261714)

[Глава 2. Разработка и реализация системы 10](#_Toc359261715)

[2.1. Проектирование системы 10](#_Toc359261718)

[2.1.1. Постановка задачи 10](#_Toc359261719)

[2.1.2. Обзор используемых технологий 10](#_Toc359261720)

[2.1.3. Архитектура системы 12](#_Toc359261721)

[2.1.4. Подбор пар 14](#_Toc359261722)

[2.1.5. Проверка пар 14](#_Toc359261723)

[2.2. Реализация системы 15](#_Toc359261724)

[2.2.1. Алгоритмы подбора пар 15](#_Toc359261725)

[2.2.2. Проверка совпадения страниц 17](#_Toc359261726)

[2.2.3. База зеркал 18](#_Toc359261727)

[2.2.4. Диаграмма классов системы 19](#_Toc359261728)

[Глава 3. Использование и тестирование 21](#_Toc359261729)

[3.1. Использование системы 21](#_Toc359261730)

[3.2. Тестирование на сайтах-зеркалах 24](#_Toc359261732)

[3.3. Тестирование на большом количестве сайтов со сходной структурой путей 26](#_Toc359261733)

[3.4. Рекомендации по улучшению 27](#_Toc359261734)

[Выводы 29](#_Toc359261735)

[Список используемых источников 30](#_Toc359261736)

[Приложение 1. Диаграмма классов системы 31](#_Toc359261737)

[Приложение 2. Примеры запросов к системе через REST интерфейс 32](#_Toc359261738)

# Введение

В последнее время наблюдается значительный рост интернета. Уже сейчас он хранит в себе огромную коллекцию информации. Например, об этом свидетельствует объем «Машины времени интернета» [1]: более 10 петабайт. Количество сайтов в интернете, по данным аналитических компаний, превысило 620 млн. Однако, по данным исследователей, значительная их часть (30-45%) [2] являются дубликатами. Казалось бы, зачем создавать сайты с одним и тем же содержимым? Но у дубликатов есть как положительные, так и отрицательные стороны. С одной стороны, это поможет в случае, если доступ к одному хосту будет невозможен, например, по причине поломки оборудования. С другой стороны, дубликаты представляют проблему для поисковых систем: если интернет на треть заполнен идентичными сайтами, то индекс систем как минимум на треть содержит избыточную информацию.

Таким образом, выявление таких сайтов является актуальной задачей. В настоящей работе разработана и реализована система, позволяющая обнаружить сайты, которые являются зеркалами, и проверить это.

В главе 1 вводятся основные определения, производится постановка задачи, а также выполняется краткий обзор техник решения.

В главе 2 выполняется проектировка системы для решения поставленной задачи: описываются используемые технологии и компоненты, разрабатывается общий концепт, приводятся используемые алгоритмы, описываются детали реализации системы.

В главе 3 описывается пример использования системы, приводятся результаты тестирования системы, и производится их анализ.

# Глава 1. Основные понятия и обзор методов

## Основные определения

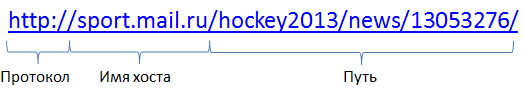
Хотя понятие «Сайты-зеркала» доступно на интуитивном уровне, его весьма трудно формализовать. Вот несколько плохих примеров определения.

Считать сайты зеркалами, если они совпадают до последнего бита, некорректно. Например, даже результаты нескольких запросов одной и той же страницы могут различаться. Это происходит из-за динамических объектов (рекламы, календарей, картинок), временных изменений, различных пользователей и так далее.

Считать сайты зеркалами, если на них есть множество похожих страниц, также неверно. Например, новостные сайты часто получают информацию из одного источника, либо ссылаются друг на друга. Текст новостей зачастую совпадает. Таким образом, в таком определении большинство новостных сайтов являются зеркалами, что на самом деле неверно.

Поэтому предлагается использовать следующее определение сайтов-дубликатов, данное в работе [5]. Зеркалами будут считаться такие сайты, в которых:

* Множества путей на этих сайтах мало отличаются (рис. 1).
* Страницы, находящиеся по общим путям, сильно совпадают. “Сильно совпадают” значит, что они отличаются деталями, не влияющими на смысл и основное содержимое, например, рекламой или версткой.



**Рис. 1. Схема URL адреса**

## Причины создания зеркал и примеры использования информации о зеркалах

Существует много причин зеркалирования сайтов:

* Технические причины. Зеркала создаются, чтобы распределить нагрузку между несколькими машинами, функционировать в случае, если один из серверов сломался, создать несколько копий в разных странах, чтобы снизить время доступа к сайту.
* Политические причины. В этом случае зеркала создаются, если по каким-либо причинам закрыт доступ к основному сайту. В качестве примеров зеркал, созданных по таким причинам, можно привести сайты *www.elgoog.im*, который заменяет сайт известной поисковой системы *www.google.com* в Китае, и *www.wikileaks.nl*, который дублирует сайт *www.wikileaks.org*, закрытый во многих странах по соображениям государственной безопасности.
* Экономические причины. Для интернет-провайдеров траффик из автономной системы (например, другой страны) может стоить дороже, чем траффик внутри своей сети. Поэтому провайдеры могут создавать внутрисетевые зеркала используемых сайтов. По таким соображениям часто создают зеркала сайтов, которые являются репозиториями различных дистрибутивов ОС *Linux.*

Для поисковых систем знание о том, какие сайты являются зеркалами, может быть весьма полезно:

* Когда основной сайт недоступен, можно выдавать ссылку на зеркало.
* Можно фильтровать поисковую выдачу, чтобы выдавать только основной сайт, а зеркала скрывать во избежание дублирования.
* Можно не индексировать страницы, находящиеся на зеркалах, тем самым освободив часть дискового пространства на серверах.
* Иногда сайты-зеркала создают, чтобы поднять рейтинг сайта в поисковых системах и обмануть алгоритмы ссылочного ранжирования, например, *PageRank* [6]. В таких случаях зеркала не несут никакой полезной нагрузки. Информация о зеркалах помогает усовершенствовать алгоритмы ранжирования, чтобы они не учитывали такие сайты.

## Проблемы поиска зеркал

При поиске дубликатов мы сталкиваемся со следующими проблемами:

* Для анализа доступен только список URL-адресов страниц, полученный веб-пауком во время обхода интернета, либо созданный прокси-сервером. Кроме этого, может быть доступна миниатюра страницы (*fingerprint*), либо её текст на какой-либо момент времени.
* Список может быть устаревшим, неполным. Часть страниц из него может быть удалена, недоступна, иметь статус *301 Moved Permanently* и так далее.
* Невозможно скачать и сохранить все страницы в памяти, так как их слишком много. Кроме этого трудность составляет загрузить в память уже существующую информацию о зеркалах.
* У некоторых сайтов-зеркал одинаковые страницы могут находиться по несовпадающим путям. Хотя эта ситуация не попадает под принятое определение “сайта-зеркала”, тем не менее такие зеркала существуют.
* Как уже говорилось, даже у сайтов-зеркал на совпадающих страницах может быть различное содержимое.

Если же рассмотреть проблемы создания системы, то можно выделить следующие сложности:

* Архитектура системы должна быть расширяемой. То есть, необходимо, чтобы в действующую систему можно было легко добавить дополнительную функциональность, улучшить старые компоненты, либо же добавить новые, реализовать другие алгоритмы определения зеркал.
* Система должна быть проста в использовании. Также необходимо, чтобы её можно было использовать вместе с существующей поисковой системой.
* Система должна быть не только вертикально, но и горизонтально масштабируемой. Преимущества горизонтального масштабирования в том, что установить несколько серверов для выполнения какой-либо задачи проще, чем создать более мощный сервер. С другой стороны, проектирование распределенных программ сложнее, чем проектирование параллельных.

Это сильно усложняет задачу. Часть из этих проблем в этой работе удастся решить, часть еще предстоит.

## Существующие алгоритмы

Если бы количество сайтов в интернете не было таким большим, то обнаружить идентичные сайты было бы достаточно просто. В таком случае можно было бы для каждого сайта создать документ, в котором сохранить все пути и страницы сайта. После этого несложно среди этих документов найти совпадающие. Однако, интернет слишком велик.

Существующие сейчас реализации для поиска зеркал являются закрытыми и/или запатентованными. Значительная часть описанных в работах [2,3,5] алгоритмов являются вероятностными и действуют по такому принципу:

* Получают на вход список страниц (URL адресов) с различных сайтов.
* Прореживают этот список по какому-либо принципу. Например, оставляют не более пятидесяти сайтов, только сайты с количеством страниц, не меньшим ста и так далее.
* Составляют пары сайтов — кандидатов на зеркала.
* Запрашивают об этих сайтах различную информацию:
  + IP-адрес хоста;
  + Миниатюру страницы;
  + Список ссылок со страницы;
  + Другую вспомогательную информацию.
* Производят анализ пар на основе существующей информации.

По используемой информации алгоритмы можно разделить на несколько категорий:

* Алгоритмы, основанные на IP адресе. Такие алгоритмы группируют входной список по IP-адресам и считают, что зеркалами являются те сайты, у которых IP-адрес одинаков.
* Алгоритмы, основанные на URL адресе. Такие алгоритмы для оценки сайтов используют различные части адреса (название хоста, схему пути до файла).
* Алгоритмы, основанные на переходах. Этот класс алгоритмов оценивает сайты, основываясь на переходах между страницами сайта, либо на ссылках на внешние ресурсы.
* Алгоритмы, основанные на деревьях сайтов. Основываясь на URL адресах страниц, эти алгоритмы строят деревья сайтов и сравнивают их.

Все эти алгоритмы несовершенны. Например, сейчас хостинг может выкладывать на одном и том же IP адресе много сайтов, благодаря технологии *NAT* (*Network Address Translation*). Алгоритмы, основанные на переходах, могут обмануть новостные сайты, так как часто они ссылаются на один источник или друг на друга (например, для повышения собственного веса в системе *PageRank*). Таким образом, процент положительных срабатываний каждого из этих классов по отдельности невелик. Однако, эти алгоритмы достаточно просты в реализации и удобны в использовании. Поэтому на поверхности лежит идея группировки результатов этих алгоритмов. В работе [3] был проведен анализ алгоритмов из первых трех классов. В результате получилось, что лучшие результаты работы получаются в случае, когда используются алгоритмы из первого и третьего классов в совокупности. При проектировке системы будут учитываться эти результаты.

# Глава 2. Разработка и реализация системы



## Проектирование системы

## Постановка задачи

Задачу поиска зеркал можно разбить на две: выделить из входного списка URL адресов пары сайтов, которые с большой вероятностью являются дубликатами, и проверить для каждой пары, действительно ли это сайты-зеркала.

Таким образом, необходимо спроектировать и реализовать систему поиска зеркал, обладающую следующими свойствами:

* Высокая скорость работы. Система должна быстро работать как с небольшим количеством больших сайтов, так и с большим количеством маленьких сайтов.
* Удобство интеграции в уже существующую систему поиска. Система должна быть реализована, как отдельный компонент, которой можно относительно просто встроить в некоторую поисковую систему, например, состоящую из веб-паука, индексатора и компонента, осуществляющего ответы на поисковые запросы пользователя.
* Стабильность работы при высокой нагрузке. Система не должна зависать в случае большого количества запросов.
* Достаточный интерфейс взаимодействия. Система должна отвечать как на запросы поиска зеркал среди переданных сайтов, так и на запросы проверки по уже известным сайтам.
* Высокое качество поиска. Должен быть высоким процент найденных сайтов-зеркал, а также количество ложных срабатываний должно быть мало.

## Обзор используемых технологий

Языком программирования системы был выбран объектно-ориентированный язык *Java* [7]. Этот язык широко распространен при реализации сложных систем со сложной архитектурой, взаимодействием нескольких компонентов, а также при разработке Enterprise систем [8].

Достоинствами *Java* являются:

* Программы, написанные на *Java*, компилируются в байт-код, который исполняется на виртуальной машине *Java* (*JVM*)*.* Реализации виртуальной машины существуют под большинство популярных операционных систем, например, *Microsoft Windows* или *Linux*. Байт-код независим от операционной системы и конфигурации оборудования. Таким образом, язык *Java* является кроссплатформенным.
* Богатая библиотека. Для *Java* существует большое количество уже реализованных алгоритмов, программ, фреймворков. Всё это помогает в разработке приложений.
* Строгая полиморфная, статическая типизация облегчает разработку и поиск ошибок.
* Удобство работы с памятью. Память для объектов выделяется в так называемой «куче». Также в *JVM* существует сборщик мусора (*garbage collector*), который освобождает память, удаляя неиспользуемые объекты, что упрощает работу с памятью при написании программ.

К сожалению, у языка программирования *Java* есть и свои недостатки:

* Ресурсоёмкость и медлительность.
* При большом количестве вспомогательных библиотек управление ими может стать затруднительным. Такая ситуация известна, как JAR hell.
* При большом количестве классов затрудняется их создание и управление зависимостями между ними.

Последние две проблемы удается решить фреймворками *Maven* [9] и *Spring* [10].

*Apache Maven* — фреймворк для автоматизации сборки приложений, специфицированных на XML-языке *POM* (*Project Object Model*). *Maven* упрощает управление подключаемыми библиотеками и сборку программы в исполняемый файл. При использовании *Maven* от разработчика требуется в специальном файле *pom.xml* указать, какие подключаемые библиотеки используются в приложении, расширение итогового приложения, дополнительную информацию по сборке и так далее. После этого *Maven* самостоятельно установит дополнительные зависимости, скомпилирует приложение и соберет исполняемый файл.

*Spring Framework* — многоцелевой фреймворк для разработки приложений. Он состоит из нескольких компонентов:

* *Inversion of Control* (*IoC*)-контейнер;
* Фреймворк аспектно-ориентированного программирования;
* Фреймворк удалённого доступа;
* Фреймворк доступа к данным;
* И другие.

Основными используемыми компонентами в данной работе являются контейнер внедрения зависимостей и фреймворк удаленного доступа.

Одной из реализаций *IoC* является внедрение зависимостей (*dependency injection*). Условно, если объекту нужен для работы какой-либо сервис, в простом случае объект либо сам создает сервис, либо сам получает ссылку на него. Используя внедрение зависимостей, объект сообщает контейнеру, что ему требуется какая-либо зависимость, и контейнер сам передает объекту сервис. Внедрение зависимостей более гибко, потому что становится легче создавать альтернативные реализации данного типа сервиса, а зависимости указывать в конфигурационном файле, без изменения в самих объектах.

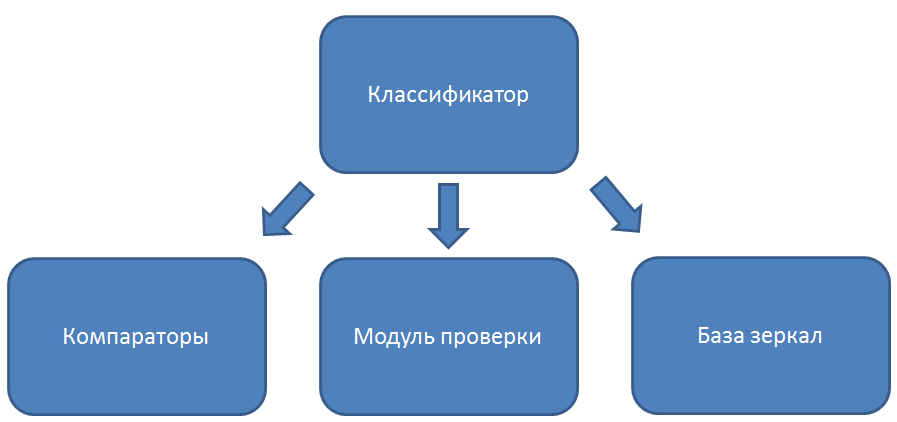
Фреймворк удаленного доступа обеспечивает представление приложения в виде веб-сервиса, обеспечивая взаимодействие с ним других приложений, передачу *Java*-объектов через сеть в стиле удаленного вызова процедур (*Remote Procedure Call, RPC*), что предоставляет пользователю удобство взаимодействия с системой.

Таким образом, применение данных технологий в системе обосновано и разумно.

## Архитектура системы

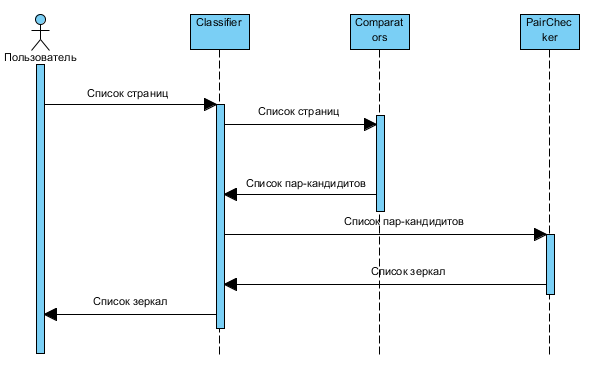
Система поиска состоит из четырех модулей:

1. модуль, отвечающий за подбор пар-кандидатов;
2. модуль, проверяющий пары на зеркала;
3. модуль взаимодействия с базой известных зеркал;
4. основной модуль системы, с которым происходит общение пользователя.



**Рис. 2. Компоненты системы**

Основной модуль (классификатор) получает на вход список сайтов с адресами страниц, затем с помощью компараторов (первого модуля) составляет список пар кандидатов. При этом может использоваться база зеркал. Затем этот список обрабатывается с помощью модуля проверки. После этого основной модуль выдает пользователю список с вероятными зеркалами.



**Рис. 3. Взаимодействие компонентов системы и пользователя**

## Подбор пар

При подборе пар-кандидатов используется несколько компараторов. Компаратор это сущность, которая умеет создавать из большого списка адресов список пар по определенному признаку:

* По IP адресу сервера, на котором находится сайт
* По URL адресам страниц
* По связности сайтов
* По уже известным зеркалам
* …

Классификатор содержит в себе несколько компараторов. Каждому из них он передает список страниц, и от каждого компаратора получает в ответ свой список пар. Каждой паре в списке компаратор назначает вес. Чем больше этот вес, тем больше вероятность, что эти сайты являются зеркалами. Также у компараторов есть своя степень влияния на итоговый список. Она берется из предположения о том, что у каждого компаратора своя ценность. Например, информация от компаратора, который анализирует деревья сайтов может быть более важна, чем информация о том, что сайты находятся (или не находятся) на одном сервере.

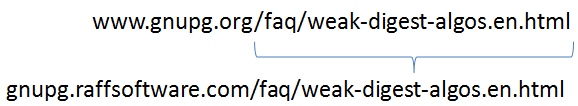
Основываясь на всех списках от компараторов, а также влиянии компараторов, классификатор создает итоговый список взвешенных пар сайтов. Для этого он для каждой пары сайтов производит следующую операцию:

1. Находит ее во всех списках.
2. Вычисляет суммарный рейтинг пары на основе весов в каждом из списков и влиянии признаков.
3. Составляет итоговый список взвешенных пар.

После этого список фильтруется, чтобы убрать пары с малым весом.

## Проверка пар

После того, как классификатор обработал данные компараторов и составил итоговый список пар для проверки, этот список передается системе проверки. Ее задача подтвердить, либо опровергнуть, что пара сайтов это действительно зеркала. Для каждой пары сайтов система выберет несколько страниц с этих сайтов и проверит их на сходство. Для выбора пар страниц система может использовать любую стратегию. В общем случае, когда ничего, кроме адресов страниц, о сайтах неизвестно, можно выбирать произвольный адрес страницы на одном из сайтов и добавлять путь к файлу на первом сайте к имени хоста другого сайта.



**Рис. 4. Преобразование ссылок**

Создав несколько пар страниц, система получает их содержимое и измеряет их сходство.

## Реализация системы

## Алгоритмы подбора пар

В текущей системе реализованы следующие компараторы для подбора пар:

1. Компаратор на основе IP адресов. Этот компаратор работает по следующему алгоритму. Для каждого сайта во входном списке вычисляется IP адрес, на котором находится сайт (с помощью запроса к DNS серверу). После этого для каждого из полученных IP адресов составляется список находящихся на нем сайтов. Далее алгоритм составляет пары кандидатов из сайтов, находящихся на одном IP-адресе. Чтобы уменьшить вероятность ложных срабатываний, считается, что если количество сайтов на одном IP адресе больше порогового значения, то этот IP адрес игнорируется. Этой оптимизацией можно отбросить сервера хостингов. В качестве примера таких сайтов можно привести сайты http://debian.com и http://debian.org.

В качестве веса пары берется число, обратное количеству сайтов на этом IP адресе.

1. Компаратор на основе URL адресов. Для определения кандидатов на основе URL адресов страниц сайта используется модификация алгоритма шинглов [4]. Анализ происходит следующим образом. Каждый URL адрес из списка разделяется на части по символу “/”. Например, адрес

www.google.com/calendar/events/Saint-Petersburg-2012/

разбивается следующим образом:

www.google.com calendar events Saint-Petersburg-2012

После этого все числа заменяются символом “\*” для удобства анализа.

www.google.com calendar events Saint-Petersburg-\*

Во многих случаях числа используются как идентификатор чего-либо в коллекции. Эта оптимизация поможет выделить значимую часть из таких адресов как, например, http://news.mail.ru/politics/13238342/ или http://news.mail.ru/politics/13236325/. Обе эти ссылки приведутся к виду http://news.mail.ru/politics/\*/.

Далее несколько подряд идущих частей группируется в терм, и к нему добавляется номер первой части. Например, если группировать в терм по 2 части, то из указанных выше частей получатся следующие термы:

* www.google.com\_calendar\_1
* calendar\_events\_2
* events\_Saint-Petersburg-\*\_3

Позиции помогут обнаружить адреса, сходные по структуре, а не только по частям адреса.

После преобразования всех адресов в термы, для каждого сайта собирается множество самых частых из полученных термов. Далее высчитывается оценку похожести хостов . Для каждого терма, который встречается на нескольких сайтах, вычисляется его вес в оценке по следующей формуле:

,

где — количество хостов, на которых встретился этот терм. Этот вес добавляется в оценку. После обработки всех термов общая оценка нормализуется по формуле:

где и — количество самых частых термов host1 и host2 соответственно. Нормализация делается, так как у сайтов с большим числом адресов получается много термов, и, соответственно, оценка сходства получается очень большой. Знаменатель дроби помогает сделать ее пропорциональной размеру сайта. Нормализованная оценка похожести выступает в качестве веса пары.

1. Компаратор на основе известных зеркал. Сперва заметим, что отношение «зеркало» обладает транзитивностью. Действительно, если сайт A является зеркалом сайта B, а сайт B — зеркалом сайта C, то сайт A является зеркалом. В качестве примера можно рассмотреть сайты www.php.net, jp1.php.net и fi1.php.net. Если уже известно, что jp1.php.net и fi1.php.net являются дубликатами www.php.net, то они также являются зеркалами и между собой. Таким образом, если хранить уже известные сайты зеркала, то при составлении пар кандидатов можно для каждой пары проверять, существуют ли такой промежуточный сайт B. Данный компаратор как раз проверяет, существует ли путь в графе зеркал от одного сайта к другому.

## Проверка совпадения страниц

Как уже было сказано, для проверки двух страниц на совпадение можно использовать несколько стратегий.

1. Можно сравнивать на точное совпадение. Для этого, например, можно вычислять хэш-функцию. Однако, эта проверка слишком строга, так как в реальных ситуациях сайты часто могут отличаться только версткой или картинками, оставаясь сходными по содержанию.
2. Другая идея состоит в том, чтобы брать только текста страницы, без различных тегов, ссылок, картинок, рекламы и так далее. Это позволит убрать со страницы большинство динамического содержимого, которое не относится к сути страницы и может мешать работе системы. В данной работе для такой проверки страниц используется алгоритм шинглов [4]. Он работает следующим образом. Несколько подряд идущих слов в тексте объединяются в кортежи, называемые чешуйками (shingles). При этом выборка происходит внахлест. Например, для текста «а роза упала на лапу азора» можно получить три 4-шингла (4 — это количество слов в кортеже):

* (а, роза, упала, на)
* (роза, упала, на, лапу)
* (упала, на, лапу, азора)

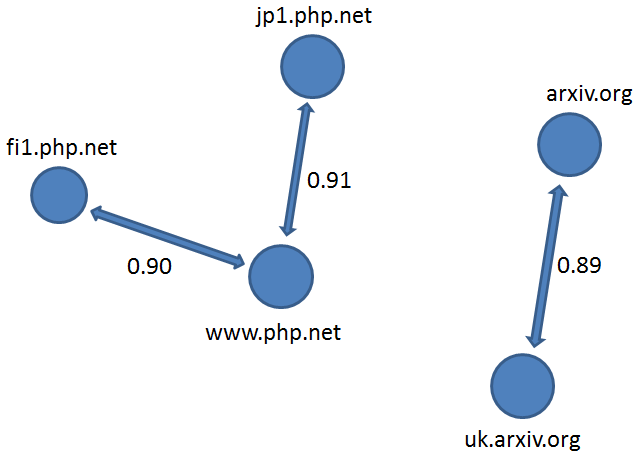
Для каждой страницы page1 и page2 строятся множества шинглов и , которые встретились в тексте. Для оценки сходства используется коэффициент Жаккарда [11]:

Нетрудно заметить, что эта оценка лежит в интервале , при этом 1 соответствует идентичности страниц, а 0 полному различию.

Проверив предлагаемые пары страниц на сходство, можно сделать итоговое предположение о том, действительно ли сайты являются зеркалами. Если более 25 процентов страниц доступны, а среднее сходство страниц составляет не менее 0.5, то сайты признаются зеркалами.

## База зеркал

Как уже было сказано выше, известную информацию о сайтах зеркалах можно использовать благодаря транзитивности отношения «зеркало». Для этого достаточно сохранять уже найденные зеркала, либо использовать данные о зеркалах, которые сообщит пользователь. Найденные зеркала можно представить в виде графа, в котором вершинами будут сайты, а неориентированными ребрами отражается отношение дублирования (рис. 5).



**Рис. 5. Пример графа зеркал**

Проблема заключается в том, что, при добавлении в него новых зеркал, этот граф может стать слишком большим и перестать вмещаться в память. Таким образом, его хранение и обработка может быть затруднена.

В качестве решения этой проблемы в данной работе была использован специальный вид баз данных: графо-ориентированная база данных *OrientDB* [12]. Особенности этой базы:

* *OrientDB* специально разработана, чтобы хранить графы.
* Удобные алгоритмы обхода графа и сборки информации.
* Встроенные функции поиска пути/кратчайшего пути между двумя вершинами.
* Удобный *Java API*, реализованные системы запросов на языках *SQL* и *Gremlin*.
* Реализованная поддержка репликации и масштабирования.
* На вершинах и ребрах графа можно хранить произвольную информацию.

Таким образом, данная база идеально подходит для задачи хранения графа зеркал.

В системе поиска *OrientDB* используется следующим образом. Для каждого сайта в ней хранится вершина, у которой в качестве свойства *host* указано название сайта. Между сайтами, которые являются зеркалами, в базе данных хранятся ребра. У каждого ребра параметром *weight* указана степень похожести между сайтами, которые ребро соединяет.

В системе реализована функция *checkMirrors*, параметрами которой являются адреса сайтов, а результатом — существование пути в графе зеркал между этими сайтами. По этим адресам система сначала находит в базе вершины, свойствами которых являются эти адреса. Если оба адреса известны, то производится поиск пути между вершинами. Если путь найден, то в качестве значения сходства сайтов используется значение минимального по весу ребра в таком пути.

При поиске пути не нужно загружать в память целиком весь граф. Также данную базу можно распределить в случае, если она не помещается на жесткий диск.

База зеркал используется в качестве одного из компараторов при составлении списка пар кандидатов, а также как отдельный модуль, который может использоваться пользователем для проверки сайтов.

## Диаграмма классов системы

Диаграмму классов системы можно увидеть в Приложении 1. Для проверки список с URL-адресами передается в Classifier, который делегирует составление списков классам, реализующим интерфейс IComparator. После этого он соединяет полученные списки пар в один и передает каждую его пару на обработку классу PageChecker. Получив итоговые результаты, Classifier выбирает из них пары сайтов с большой вероятностью зеркалирования и помещает их в базу известных зеркал. После этого выдает итоговый список пар пользователю.

Как можно видеть из диаграммы классов, в большинстве случаев в программе используются интерфейсы, а не конкретные реализации классов. Благодаря этому систему достаточно просто расширять, добавлять новые компараторы, использовать другие алгоритмы подбора ссылок и проверки текста, переделывать принцип построения итогового списка. Также в такой системе несложно покрывать unit-тестами каждый компонент программы в отдельности и использовать *Spring IoC*. В каждом из компонентов не хранится текущее состояние (stateless-компоненты), таким образом, система доступна для многопоточного использования.

Все компоненты:

* Классификатор;
* Набор компараторов;
* База зеркал;

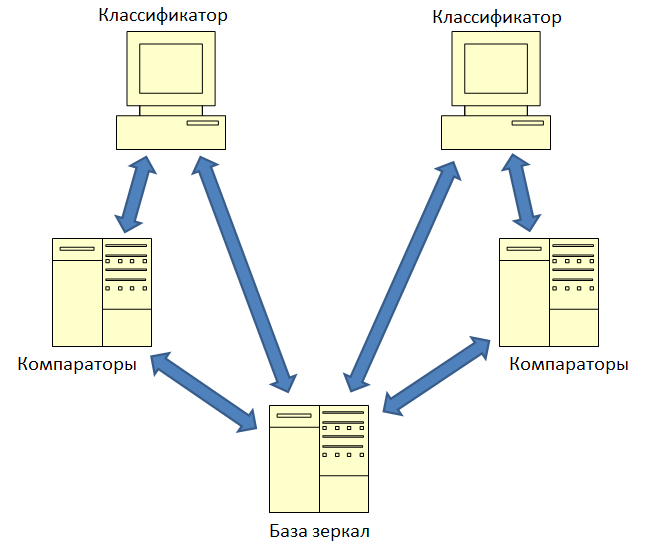
реализованы в виде отдельных модулей. Общение между модулями происходит по *HTTP* протоколу. Сериализация передаваемых объектов происходит в *Json* нотацию. Это обеспечивает:

* Простоту взаимодействия между модулями. Для каждого модуля реализован свой клиент на языке *Java*, с помощью которого осуществляется связь с модулем.
* Так как при взаимодействии используется протокол *HTTP*, то язык программ, которые будут использовать классификатор, может быть любым. Для этого достаточно иметь поддержку протокола *HTTP*.
* Модули можно запускать на разных машинах. Таким образом, поддерживается горизонтальное масштабирование системы.

# Глава 3. Использование и тестирование

## Использование системы

Система выполнена в качестве отдельного приложения, работает, как REST веб-сервис [13, 14]. То есть, система получает запросы от удаленного пользователя и синхронно отсылает ему ответ. Система состоит из трех модулей, каждый модуль собирается в файл с расширением *war.* Эти модули можно запустить на любом контейнере приложений, например, *Apache Tomcat* [15]. При этом модулю в файле настройки необходимо только указать сетевой адрес требуемых ему компонентов.

**Рис. 6. Возможная конфигурация системы**

На рис. 6 представлена одна из возможных конфигураций системы. В ней на запросы пользователя отвечают два классификатора. Для распределения нагрузки каждый из них использует свой сервер компараторов, и вместе они обращаются к одной базе зеркал.

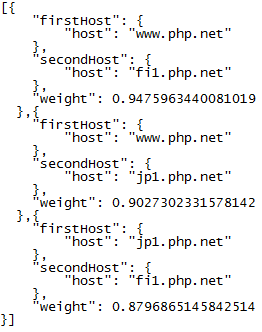
Взаимодействие с системой происходит через HTTP интерфейс. Интерфейс состоит из двух точек доступа: *classifierEndpoint* и *mirrorsBaseEndpoint.* Сводную таблицу примеров запросов можно увидеть в Приложении 2.

Через точку *classifierEndpoint* происходит взаимодействие пользователя с классификатором. Для этого необходимо совершить POST запрос по адресу *http://{host}/classifier/classify*. В качестве содержимого запроса необходимо указать список адресов с сайтов (рис. 7).



**Рис. 7. Пример входных данных запроса классификации**

В случае успешного выполнения поиска зеркал пользователю отсылается ответ со статусом *200 OK*. В качестве содержимого выступает массив из объектов *WeightedPair*, то есть два сайта и степень сходства между ними. Пользователь получает этот массив в виде JSON объекта (рис. 8). В случае какой-либо ошибки сервера пользователю возвращается ответ со статусом *400 Bad request* и текстом об ошибке.



**Рис. 8. Пример выходных данных запроса классификации**

Точка *mirrorsBaseEndpoint* используется для взаимодействия напрямую с базой зеркал. Через неё можно проверить, есть ли путь в базе между двумя зеркалами, и добавить существующие зеркала напрямую.

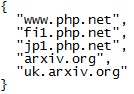
Чтобы добавить зеркала, необходимо совершить *PUT* запрос по адресу *http://{host}/mirrorsBase/pair*. Содержимым должен быть *Json* объект *WeightedPair* с полями *firstHost* и *secondHost,* в которых должны быть записаны адреса сайтов, и полем *weight* со значением сходства этих сайтов (рис. 9). Возвращаемым значением является статус *200 OK* в случае успеха, и статус *400 Bad request* с сообщением об ошибке в случае неудачи.



**Рис. 9. Пример входных данных запроса добавления зеркала**

Чтобы проверить сайты по базе, необходимо сделать *POST* запрос по адресу *http://{host}/mirrorsBase/pair*. В запросе необходимо передать *Json* с полями *first* и *second*, в которых необходимо указать сайты для проверки. В случае успешного поиска пользователь получит ответ *200 OK* с объектом *WeightedPair,* сериализованным в *Json*. Если какого-то из переданных сайтов нет в базе, значение *weight* будет равно -1. В случае ошибки пользователь получит ответ *400 Bad request* и информацию об ошибке.

Также можно передать базе список из хостов сайтов. База проведет поиск между каждой парой хостов в этом списке. В ответ пользователь получит список пар сайтов, между которыми база нашла путь. Для этого необходимо совершить *POST* запрос по адресу *http://{host}/mirrorsBase/list.* Содержимое запроса должно состоять из массива строк — хостов.



**Рис. 10. Пример входных данных запроса проверки списка**

В случае успешной проверки результатом будет статус *200 OK* список объектов *WeightedPair.* В случае ошибки сервера пользователь получит сообщение со статусом *400 Bad request* и текст ошибки.



## Тестирование на сайтах-зеркалах

Сначала система была испытана на списке из 51 тысяч адресов, состоящем только из сайтов, являющихся зеркалами:

* www.php.net, fi1.php.net, jp1.php.net.
* cran.r-project.org, cran.gis-lab.info.
* arxiv.org, uk.arxiv.org.
* www.wikileaks.org, www.wikileaks.no.
* www.gnupg.org, gnupg.raffsoftware.com, gnupg.parentinginformed.com.
* www.lenta.ru
* www.sports.ru
* habrahabr.ru

Изначально база известных зеркал была пуста. Для проверки каждой пары сайтов составлялось 40 пар страниц. После этого каждая пара страниц загружалась из интернета.

Все пары зеркал были найдены и добавлены в базу.

Таблица 1. Результаты первого запуска.

|  |  |
| --- | --- |
| Количество ссылок в списке | 51200 |
| Количество сайтов в списке | 15 |
| Количество сайтов после фильтрации | 15 |
| Известных пар зеркал | 9 |
| Предложенных пар кандидатов | 9 |
| Найденных пар зеркал | 9 |
| Из них найденных верно | 9 |

Проблемы возникли только с парой *www.wikileaks.org* и *www.wikileaks.no*:

* Во-первых, формат путей не всегда совпадал. Часть адресов страниц с сайта *www.wikileaks.no* оканчивалась на */index.html*:

http://www.wikileaks.no/wiki/US\_Marine\_Corps\_Intelligence\_Exploitation\_of\_Enemy\_Material%2C\_2006/index.html

В свою очередь, адреса страниц сайта *www.wikileaks.org* были без этого окончания:

http://www.wikileaks.org/wiki/US\_Marine\_Corps\_Intelligence\_Exploitation\_of\_Enemy\_Material%2C\_2006/

* Во-вторых, на сайте *www.wikileaks.org* в некоторых случаях страница содержала только информацию о том, что исходная страница перемещена на другой адрес, но при этом автопереход не совершался.

http://www.wikileaks.org/wiki/Minton\_report\_secret\_injunction\_gagging\_The\_Guardian\_on\_Trafigura,\_11\_Sep\_2009/

http://www.wikileaks.org/wiki/Minton\_Report\_secret\_injunction\_gagging\_The\_Guardian\_on\_Trafigura,\_11\_Sep\_2009

Несмотря на это пара зеркал была обнаружена, хотя с не слишком большим весом: всего 0.646. Результаты проверки остальных пар разделились на две категории:

1. Удалось получить все пары страниц. Обычно это означает, что сайты являются полными зеркалами. К примеру, при проверке пары *fi1.php.net* и *jp1.php.net* удалось получить все пары страниц, и среднее сходство составило 0.94.
2. Часть пар удалось получить для проверки, а из другой части доступна была только одна из страниц в паре. Это означает, что либо проверяемые сайты не являются зеркалами, либо один из сайтов является неполным зеркалом другого. Это произошло при проверке пары *www.gnupg.org* и *gnupg.raffsoftware.com*. Из 40 пар страниц для проверки оказались доступны только 26. Это связано с тем, что на момент проверки на сайте *gnupg.raffsoftware.com* отсутствовала часть страниц по руководству пользователя.

На построение списка пар-кандидатов ушло 2 секунды, на проверку всех пар, то есть на загрузку страниц и их сравнение ушла 21 секунда. При профилировании видно, что наибольшее время выполняется загрузка страниц с сайта и построение дерева разбора html-содержимого.

Из тестирования на зеркалах видно, что система действительно работает, как от нее требуется, при этом на малом количестве сайтов работает достаточно быстро (если не учитывать скачивание страниц).

## Тестирование на большом количестве сайтов со сходной структурой путей

Система была протестирована на большом количестве блогов с сайта *blogspot.com*. Адреса страниц были взяты с ресурса *commoncrawl.org*. В списке адресов страниц, использованном для тестирования, содержится 254 тысячи адресов с 44500 блогов. После фильтрации, в списке остались лишь 568 сайтов с количеством страниц, большим 50. Адреса страниц на этом сайте, независимо от блога, обладают схожей схемой:

http://{blogName}.blogspot.com/{year}/{month}/{postNumber}-{postName}.html

Все блоги находятся по одному IP адресу, поэтому IP компаратор каждой паре назначает отрицательный вес. База зеркал не содержит таких сайтов, поэтому компаратор на основе базы также выдает пустой список. А вот компаратор на основе URL адресов считает, что у всех сайтов похожие схемы адресов. Популярными являются термы:

* (\*, \*, 1), что соответствует году и месяцу в URL адресе. Однако, такой терм встречается на каждом сайте, поэтому вес его небольшой.
* Термы, содержащие название поста, в котором используются популярные слова, либо которые посвящены какому-либо событию. Например, (\*, \*-christmas-\*.html, 2), (\*, \*-hello.html, 2), (\*, \*-trip-\*.html, 2), (\*, \*-new\_year.html, 2), (\*, \*-coming\_soon.html, 2). Такие совпадающие термы встречаются нечасто, поэтому их вес больше, чем у терма (\*, \*, 1).

Однако не было найдено ни одной пары блогов, чтобы у них совпадало большое количество значимых термов. Поэтому ни одной паре не удалось превысить тот отрицательный вес, который был выдан IP компаратором. Таким образом, на этом входном файле список пар кандидатов оказался пустым.

Таблица 2. Результаты второго запуска

|  |  |
| --- | --- |
| Количество ссылок в списке | 254800 |
| Количество сайтов в списке | 44722 |
| Количество сайтов после фильтрации | 568 |
| Известных пар зеркал | Предположительно 0 |
| Предложенных пар кандидатов | 61 |
| Найденных пар зеркал | 0 |
| Из них найденных верно | 0 |

На обработку таких входных данных потребовалось 7 секунд. Это связано с тем, что:

* Не требовалось скачивать страницы для проверки, так как не было пар кандидатов.
* База не содержала этих сайтов, таким образом, не требовалось искать пути между ними.

Если же запретить IP компаратору выставлять отрицательный вес, то список пар кандидатов становится непустым. Но после построения списка пар страниц для проверки, большинство пар оказываются невалидными, и ни одна пара кандидатов не проходит проверку.

Из этого тестирования можно сделать вывод, что система может обрабатывать и большие массивы данных.

## Рекомендации по улучшению

По итогам тестирований можно сделать вывод о том, что система пригодна для промышленного использования. Также было обнаружены некоторые недостатки:

* Всё выполнение одного запроса происходит в одном потоке. Это уменьшает скорость выполнения. Кроме этого вся работа происходит на одном сервере.
* Алгоритм генерации пар страниц для проверки не является идеальным. Это можно наблюдать на примере сайтов *www.wikileaks.org* и *www.wikileaks.no*, в которых незначительные изменения в путях мешали в проверке.
* Также некоторые зеркала могут быть не найдены по причине недостаточности существующих компараторов для анализа. Вполне возможен вариант, при котором одинаковые страницы на сайтах-зеркала находятся по разным путям, например, на первом сайте используется следующий формат адресов:

http://{host}/archive/show?pageid={pageid}

А на зеркале следующий формат:

http://{host}/archive/{pageid}/display

Сайт с такими адресами потеряется при построении пар кандидатов.

Отсюда следует, что систему можно улучшить следующим образом:

1. Реализовать взаимодействие между компонентами на основе очереди сообщений (message queue). Использование очереди позволить независимо увеличивать количество классификаторов, как производителей заданий, и компараторов, как потребителей.
2. Реализовать новые компараторы, например, на основе правил нормализации URL адресов. С их помощью можно решить проблему с немного отличающимися адресами страниц. Подробнее про правила нормализации можно прочитать в работе [16].

# Выводы

В данной работе была разработана система для обнаружения сайтов зеркал. Данная система для анализа использует URL адреса страниц сайтов, их содержимое, а также информацию об уже найденных зеркалах. Для этого в программе используются различные компараторы, модуль сравнения текстов и база, хранящая граф зеркал.

Как показали испытания, разработанная система помогает обнаружить зеркала по небольшой информации, известной о сайтах. Система может применяться вместе с веб-пауком. Для этого достаточно лог обхода сайтов предоставить в качестве входных данных системы. Также база зеркал может эффективно быть в алгоритмах ранжирования ссылок. Для этого достаточно запросить у базы информацию о зеркалах среди ссылок в поисковой выдаче.

Все поставленные задачи были выполнены. Таким образом, цель работы была достигнута.

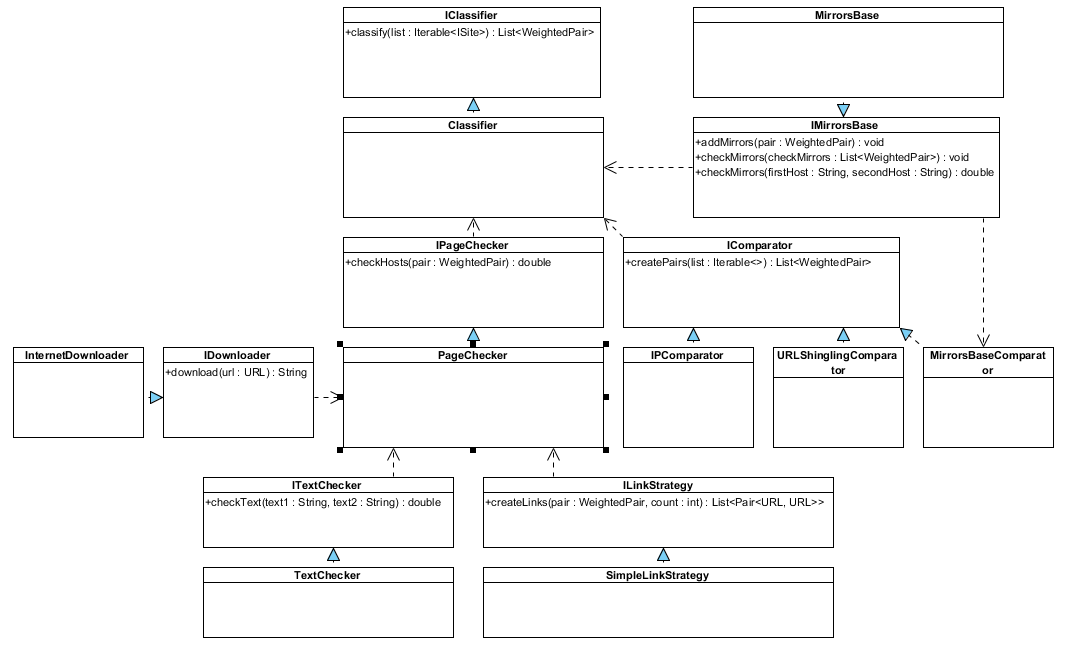
# Список используемых источников

1. *Wayback Machine: Internet Archive*.[http://web.archive.org/‎](http://web.archive.org)
2. *N. Shivakumar, H. Garcia-Molina.* Finding near replicas of documents on the web. Proceedings of Workshop on Web Databases (WebDB’98), 1998.
3. *K. Bharat, A. Broder, J. Dean, M. Henzinger.* A Comparison of Techniques to Find Mirrored Hosts on the WWW. Fourth ACM Conference on Digital Libraries, 1999.
4. *A. Broder, S. Glassman, M. Manasse, G. Zweig.* Syntactic Clustering of the Web. Sixth international conference on World Wide Web, 1997.
5. *K. Bharat, A.Broder.* Mirror, mirror on the Web: a study of host pairs with replicated content. Eighth International World Wide Web Conference, 1999.
6. *S. Brin, L. Page*.The Anatomy of a Large-Scale Hypertextual Web Search Engine, Seventh International World-Wide Web Conference (WWW 1998), 1998.
7. Язык программирования Java*.* [*http://www.java.com/ru/*](http://www.java.com/ru/)
8. Java Platform Enterprise Edition.

<http://www.oracle.com/technetwork/java/javaee/overview/index.html>

1. Apache Maven. <http://maven.apache.org/>
2. Spring Framework. <http://www.springsource.org/>
3. Jaccard similarity coefficient. <http://en.wikipedia.org/wiki/Jaccard_index>
4. NoSQL graph-based DBMS OrientDB. <http://www.orientdb.org/>
5. *R. Fielding.* Architectural styles and the design of network-based software architectures. 2000.
6. *C. Pautasso, O. Zimmermann, F. Leymann*. RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision. 17th International World Wide Web Conference (WWW2008), 2008.
7. Apache Tomcat. <http://tomcat.apache.org/>
8. *T. Lei, R. Cai, J.-M. Yang, Y. Ke, X. Fan, L. Zhang*. A pattern tree-based approach to learning URL normalization rules. 19th international conference on World Wide Web, 2010.

# Приложение 1. Диаграмма классов системы



# Приложение 2. Примеры запросов к системе через REST интерфейс

|  |  |  |  |
| --- | --- | --- | --- |
| Описание функции | Описание запроса | Формат содержимого запроса | Формат ответа |
| Поиск зеркал среди предоставленных сайтов | POST запрос по адресу http://{host}/classifier/classify | Текстовый список URL адресов страниц. Разделителем адресов служит перенос строки. | Список объектов класса WeightedPair, сериализованный в Json. |
| Добавление зеркала в базу | PUT запрос по адресу http://{host}/mirrorsBase/pair | Объект класса WeightedPair, сериализованный в Json, с названиями сайтов и степенью сходства. | Статус 200 OK в случае успеха, без содержимого. Статус 400 Bad request в случае неудачи, содержимое – сообщение об ошибке. |
| Проверка сайтов на зеркала по базе | POST запрос по адресу http://{host}/mirrorsBase/pair | Пара из двух адресов сайтов в формате Json:  {  “first”:”firstSite”,  “second”:”secondSite”  } | В случае успешного выполнения ответ со статусом 200 OK и объектом класса WeightedPair в формате Json. В случае неудачи ответ со статусом 400 Bad request. |
| Проверка списка сайтов на зеркала.  Выполняется поиск пути между каждой парой сайтов в списке. | POST запрос по адресу http://{host}/mirrorsBase/list | Список из названий хостов в формате Json:  [  “www.google.com”,  “www.yandex.ru”,  “www.mail.ru”  ] | В случае успешного выполнения ответ со статусом 200 OK и список из объектов классов WeightedPair в формате Json |