МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**
Кафедра **«Прикладная математика и информатика»**

01.03.02 ПРИКЛАДНАЯ МАТЕМАТИКА И ИНФОРМАТИКА

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

# БАКАЛАВРСКАЯ РАБОТА

на тему Разработка алгоритма анализа научных статей

Студент(ка) Е.С. Розанов _____ _____

Руководитель В.С. Климов _____ _____

**Допустить к защите**

Заведующий кафедрой, к.т.н, доцент, А.В. Очеповский _____

«_____»_____20_____г.

Тольятти 2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ

Зав.кафедрой «Прикладная математика и информатика»

_____А. В. Очеповский

«____»_____2016 г.

**ЗАДАНИЕ**

**на выполнение бакалаврской работы**

Студент Розанов Евгений Сергеевич

1. Тема Разработка алгоритма анализа научных статей

2. Срок сдачи студентом законченной выпускной квалификационной работы 24.06.2016.

3. Исходные данные к выпускной квалификационной работе скриптовый язык общего назначения PHP, прототипно-ориентированный сценарный язык программирования JavaScript, фреймворк Yii2.

4. Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов, разделов): анализ состояния вопроса; разработка алгоритма анализа научных статей; программная реализация предложенных решений; апробация предложенных решений на реальных данных; выводы по работе

5. Ориентировочный перечень графического и иллюстративного материала презентация, включающая блок-схемы работы приложения, графики, диаграммы, экранные формы, демонстрирующие работоспособность программного продукта.

6. Дата выдачи задания « 11 » января 2016 г.

| | | |
|---|---|---|
| Руководитель выпускной квалификационной работы | (подпись) | В. С. Климов |
| Задание принял к исполнению | (подпись) | Е. С. Розанов |

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав.кафедрой«Прикладная
математика и информатика»
_____А.В.Очеповский

«____»_____2016 г.

**КАЛЕНДАРНЫЙ ПЛАН**
**выполнения бакалаврской работы**

Студента Розанов Евгений Сергеевич
по теме Разработка алгоритма анализа научных статей

| Наименование раздела работы | Плановый срок выполнения раздела | Фактический срок выполнения раздела | Отметка о выполнении | Подпись руководителя |
|---|---|---|---|---|
| Анализ состояния вопроса | 1.03.2016 | 1.03.2016 | выполнено | |
| Разработка обеспечения | 11.03.2016 | 11.03.2016 | выполнено | |
| Программная реализация предложенных решений | 25.03.2016 | 25.03.2016 | выполнено | |
| Апробация предложенных решений на реальных данных | 3.04.2016 | 3.04.2016 | выполнено | |
| Оформление пояснительной записки | 6.04.2016 | 6.04.2016 | выполнено | |

| | | | | |
|---|---|---|---|---|
| Создание презентационного материала | 21.04.2016 | 21.04.2016 | выполнено | |
| Проверка на наличие заимствований (плагиата) в системе «Антиплагиат ВУЗ» | 30.05.2016 | 30.05.2016 | выполнено | |
| Предварительная защита | 6.06.2016-11.06.2016 | 31.05.2016 | выполнено | |
| Сдача на кафедру отзыва научного руководителя и ознакомление с ним | 20.06.2016 | 20.06.2016 | выполнено | |
| Сдача на кафедру комплекта документов для защиты | 24.06.2016 | 24.06.2016 | выполнено | |
| Защита ВКР | 27-30.06.2016 | 29.06.2016 | выполнено | |

Руководитель выпускной квалификационной работы _____ (подпись) В. С. Климов

Задание принял к исполнению _____ (подпись) Е. С. Розанов

# АННОТАЦИЯ

## к бакалаврской работе на тему «Разработка алгоритма анализа научных статей»

Работа выполнена студентом Тольяттинского государственного университета Розановым Евгением Сергеевичем.

Цель работы – разработка эффективного алгоритма анализа научных статей с записью полученного результата в базу данных.

Объектом исследования является технологии анализа неструктурированной информации.

Предмет исследования – процесс автоматизированного выделения атрибутов научной статьи

Для достижения поставленной цели в работе решаются следующие задачи:

1. Провести анализ алгоритмов информационного поиска.
2. Разработать алгоритм анализа файлов научных статей.
3. Спроектировать, разработать и протестировать программную реализацию предложенных решений.

Выпускная квалификационная работа состоит из введения, трех глав, заключения.

В главе 1 рассматриваются различные методы анализа данных.

В главе 2 проводится анализ средств, необходимых для выделения из текста статьи таких ее элементов, как: автор статьи, тематика, название статьи и т.д. Применяются такие технологии как индексация, регулярные выражения.

В главе 3 описывается процесс разработки приложения. Описан разработанный программный продукт. Приведены примеры его использования.

В заключении подводятся итоги исследования, формируются окончательные выводы по рассматриваемой теме.

**Оглавление**

# Введение

В наше время существует неисчислимое количество алгоритмов, которые подразделяются на множество групп, таких как комбинаторные, алгоритмы поиска, алгоритмы вычисления, сортировки, слияния, алгоритмы работы с данными. И это лишь малая часть алгоритмов известных на сегодняшний день. По своей сути, алгоритм это набор инструкций, описывающих порядок действий исполнителя для достижения некоторого результата. В частности можно выделить алгоритм анализа данных. Задача данного вида алгоритмов, это структурировать данные, для получения эффективного доступа к ним.

Получив достаточные сведения об алгоритмах информационного поиска, мы адаптируем их для анализа файлов научных статей. Адаптированный алгоритм должен находить в файлах научных статей фамилии и инициалы авторов, и тематику статьи.

Для удобства, реализуем web-интерфейс, для эффективного администрирования.

Исходя из цели данной работы, нами была предпринята попытка решить следующие задачи:

1. Провести анализ алгоритмов информационного поиска.
2. Разработать алгоритм анализа файлов научных статей.
3. Спроектировать, разработать и протестировать программную реализацию предложенных решений.

Выпускная квалификационная работа состоит из введения, трех глав, заключения.

Сначала рассматриваются различные методы анализа данных. Затем проводится анализ средств, необходимых для выделения из текста статьи таких ее элементов, как: автор статьи, тематика, название статьи и т.д. Применяются такие технологии как индексация, регулярные выражения. Затем описывается процесс разработки приложения. Также приводятся примеры использования разработанного приложения.

# 1 Алгоритмы анализа файлов.

## 1.1 Описание предметной области

Из-за стремительного развития сети интернет на просторах всемирной сети находится огромное количество текстовой информации. Многие исследователи и аналитики изъявляют желание получить толк из всей этой информации. Текстовая информация, находящаяся на просторах интернета, по большому счету неструктурированна и трудна для анализа. Создается много программ, сервисов, алгоритмов способствующих структуризации информации. Структурированная информация приносит намного больше пользы, ведь она поддается эффективному анализу. Структурированная информация отличается от информации без структуры тем, что она хранится в формате, который сохраняет не только сами данные, но и разнообразные свойства этих данных и сведения об их структуре.

Существует термин информационный поиск, который заключает в себе огромное количество понятий и может применяться в широком спектре задач. Тем не менее, информационный поиск как научная дисциплина – это процесс поиска в большой коллекции некого неструктурированного материала, удовлетворяющего информационные потребности [1-5].

Если рассматривать информационный поиск, таким образом, то можно отметить, что раньше подобной деятельностью занимались лишь отдельные специалисты. Подход к данному виду деятельности изменился, и теперь сотни миллионов людей, ежедневно пользуются поисковыми системами или другими сервисами, в той или иной степени использующими информационный поиск. Данный вид поиска быстро стал основной формой доступа к информации, вытесняя традиционный поиск по ключу.

К информационному поиску также можно отнести некоторые задачи, которые не включены в базовое определение. При использовании словосочетания «неструктурированные данные», мы имеем ввиду данные, которые не имеют явной, семантически очевидной и легко реализуемой на

компьютере структуры. В реальности же абсолютно неструктурированных данных практически не существует. Для примера, обычные текстовые данные имеют скрытую структуру, характерную для естественных языков. Очевидно, что большинство текстовых документов имеют некую структуру, в виде абзацев, заголовков, сносок, которые обычно представлены в тексте в виде явной разметки. Поэтому методы информационного поиска используются также для «полуструктурированного» поиска, например для нахождения документа, в заголовке которого содержится некое слово[6-10].

Системы информационного поиска можно разделить по классификациям относительно масштаба их действия. Стоит выделить три класса. Первый тип веб-поиск, его использование означает, что система должна осуществить поиск среди миллиардов документов. Особенностью данного типа поиска в том, что присутствует необходимость сбора документов для индексации. Другой тип поиска, кардиально отличается от веб-поиска – персональный информационный поиск. Данный тип поиска ориентирован на меньшие размеры данных для анализа. Например, персональный информационный поиск используется для распределения писем по папкам. Для подобных систем критично обрабатывать всё многообразие форматов документов на обычном персональном компьютере. Третий класс – системы корпоративного, ведомственного и ориентированного на предметную область поиска. Данные системы, в основном, работают с документами, хранящимися в централизированных файловых системах [11-15].

## 1.2 Схематизация документа и декодирование последовательности символов

### 1.2.1 Выделение последовательности символов в документе

Цифровые документы, являющиеся входной информацией для процесса индексирования, как правило, представляют собой набор байтов в файле или веб-сервере. Первым этапом обработки является преобразование последовательности в линейную последовательность символов. Для

английского текста, набранного в системе ASCII, такая задача является банальной. Однако, бывают более сложные случаи. Последовательности символов могут быть закодированы в одной из многих однобайтовых или многобайтовых кодировок, например UTF-8, в том числе национальные стандарты. Для начала работы, нужно определить кодировку. Данная проблема относится к задаче классификации на основе машинного обучения, но на практике она часто решается с помощью эвристических методов или метаданных о документе. После выявления кодировки, последовательность байтов преобразуется в последовательность символов. Выявленную кодировку стоит зафиксировать, ведь это дает некоторое представление о языке, на котором написан документ.

Есть вероятность, что символы нужно декодировать из двоичного представления, например из doc-файла или архивных файлов. Очевидно, что нужно сначала определить формат документа, а потом выбрать соответствующий декодер. Даже для простых текстовых документов может потребоваться дополнительная декодировка.

### 1.2.2 Выбор структурной единицы документа

Следующий этап заключается в выборе структурной единицы документа для индексирования. Не во всех ситуациях документы представляют собой фиксированные единицы, предназначенные для индексирования. Например, в системе Unix последовательность электронных сообщений хранится в одном файле, что может затруднить информационный поиск по одному сообщению.

При работе с большими документами возникает проблема детализации индексирования. Например, при работе с коллекцией книг, было бы логично индексировать всю книгу, как отдельный документ. Но в этом случае при поиске «hello world», есть вероятность, что в результате поиска будет выведена книга, где слово «hello» встречается в первой главе, а «world» в последней, что было бы неудовлетворительным результатом. Следовательно, более релевантно было бы разделить книгу на несколько индексируемых документов, размером с

главу, или даже абзац. Но не следует разбивать документ на слишком маленькие части, ведь искомая информация может позиционироваться на большом промежутке.

## 1.3 Определение лексикона терминов

### 1.3.1 Разделение текста на лексемы

После идентификации последовательности символов и выделения структурных единиц документа, текст разделяется на лексемы. Также, из текста убираются некоторые символы, такие как знаки пунктуации.

| | | | | |
|---|---|---|---|---|
| Ввод: | Hello, my dear friend! | | | |
| Вывод: | Hello | my | dear | friend |

Рисунок 1.1 - Выявление лексем.

Лексема – это экземпляр последовательности символов в определенном документе, объединенные в семантическую единицу для обработки. Термин – это тип, включенный в словарь системы информационного поиска [2]. Множество терминов индекса может абсолютно отличаться от лексем, которые, например, могут быть семантическими идентификаторами в иерархии, но на практике в большинстве систем информационного поиска они напрямую связанны с лексемами в документе. Основным вопросом является верное разделение текста на лексемы. Очевидно, что можно разбить текст по пробелам и убрать знаки пунктуации. Но существуют такие знаки пунктуации как апостроф, который часто используется в текстах на английском языке. Для того, чтобы поиск был корректным, обработка запросов осуществляется с помощью одного и того же лексического анализатора. Данный подход дает гарантию того, что последовательность символов в тексте всегда совпадет с такой же последовательностью, набранной в запросе.

Эти проблемы напрямую зависят от языка. Следовательно, язык должен быть известен до начала манипуляций с текстовыми данными. Определение

языка, основанное на классификаторах, использующих в качестве признаков короткие подпоследовательности символов.

Большая часть языков и конкретных предметных областей имею специфические лексемы, которые следует распознавать. В данный список входят различные названия, вошедшие в культурный контекст. Так же компьютерные технологии ввели множество специальных лексем, таких как адреса электронной почты, веб-адреса, числовые IP-адреса и многие другие. В теории можно было бы отказаться от таких лексем, но это бы значительно повлияло на эффективность индексирования.

Иногда, разделение текста по пробелам приводит к разрыву слов, которые следовало бы рассматривать как цельные лексемы. Это часто происходит с названиями (Нижний Новгород, Набережные Челны), сложными словами, которые могут записываться как с пробелами, так и слитно. К словам с внутренними пробелами, которые следовало бы рассматривать как отдельную лексему, относятся телефонные номера.

### 1.3.2 Игнорирование распространенных терминов

Иногда, некоторые самые распространенные слова в тексте вообще исключаются из лексикона. Данный тип слов называется стоп-слова. Обычно, список стоп-слов создается относительно частоте включения в коллекцию, то есть наиболее часто встречающиеся слова, проходят ручную фильтрацию с учетом их семантической связи с предметной областью и создается список стоп-слов, которые исключаются из индексации.

| | | | | | |
|---|---|---|---|---|---|
| и | в | или | за | на | под |
| из | для | как | что | бы | ли |
| с | да | к | | | |

Рисунок 1.2 - Пример списка стоп-слов.

В большинстве случаев игнорирование стоп-слов не вызывает проблем. Однако при поиске фраз бывают исключения. Запрос на поиск фразы «самолёт до Москвы», скорее всего, потеряет смысл, при игнорировании стоп-слова

«до». Подобное игнорирование в разной степени влияет на разнообразные запросы. Именно из за этой проблемы большинство поисковых систем отказались от использования игнорирования стоп-слов.

### 1.3.3 Классификация терминов по классам эквивалентности

Предположим, что документ и запрос уже разделены на лексемы. В самом простом случае лексемы в запросе точно совпадают с лексемами в документе. Однако присутствуют случаи, когда две последовательности не совпадают в точности, но считаются эквивалентными. Например, если мы ищем слово «США», то возможно, мы бы хотели также видеть документы, содержащие аббревиатуру «С.Ш.А.».

Нормализация лексем – это процесс приведения лексем к канонической форме, чтобы устранить несущественные различия между последовательностями символов. Самый часто используемый способ нормализации заключается в неявном создании классов эквивалентности.

Преимущество использования правил отображения, которые, допустим, удаляют символы, такие как дефис, проявляются в том, что классы эквивалентности возникают неявно, а не вычисляются заранее. Термины, которые в результате применения этих правил становятся идентичными, относятся к одному и тому же классу эквивалентности. Несмотря на то, что удалить символы из лексем относительно легко, обратная процедура является довольно сложной, ведь из-за того, что классы эквивалентности возникают неявно, совершенно неясно, куда следует вставить недостающие символы.

Альтернативный метод создания классов эквивалентности основан на связях между лексемами. Данный метод может использовать списки синонимов, составленные вручную, например слова «правильно» и «верно» могут быть записаны в этот список. Подобные зависимости можно создать двумя способами. В большинстве случаев, сначала выполняется индексирование ненормализованных лексем, а затем для конкретного термина из запроса создается список расширенных запросов, состоящий из нескольких

вариантов соответствующего термина, включенных в лексикон. В этом случае термин запроса является результатом логического сложения нескольких инвертированных списков. Вторым способом является расширение в ходе построения индекса. Использование данных методов является менее эффективным, чем создание классов эквивалентности, поскольку в этих случаях приходится хранить и объединять большое количество позиций. Первый способ предусматривает создание дополнительного словаря для расширения запросов и увеличивает время их обработки, в то время как второй метод требует большие пространства для хранения позиций. Обычно, увеличение требований к объему памяти, необходимой для хранения дополнительной информации считались серьезным недостатком, но если учесть снижение стоимости устройств хранения данных, и появление других преимуществ, то, объективно можно принять все минусы.

### 1.3.4 Стемминг и лемматизация

По понятным причинам, в документах встречаются разные форма одних и тех же слов, например «программировать» и «программирование». Во многих ситуациях кажется полезным по одному из них находить документы, содержащие другие слова из этого семейства.

Цель стемминга и лемматизации – привести словоформы и производные формы к общей, основной форме.

| | |
|---|---|
| Входная строка | у моего ноутбука не работает несколько портов. |
| Выходная строка | у мой ноутбук не работать несколько порт |

Рисунок 1.3 - Пример стемминга и лемматизации.

Понятия стемминга и лемматизации похожи, но имеют некоторые отличия. Стеммингом обычно называется приближенный эвристический процесс, в ходе которого от слов отбрасываются окончания в расчете на то, что в большинстве случаев это действие будет оправданно. Лемматизация – это точный процесс с использованием лексикона и морфологического анализа слов, в результате которого удаляются только флективные окончания, и

возвращается основная форма слова, называемая леммой. Например, лексема «руки» в ходе стемминга может превратиться в «ру», в то время как лемматизация вернет «рука».

### 1.4 Поиск по регулярным выражениям

Регулярные выражения, это система обработки текста, основанная на системе записи образцов для поиска. Правила поиска задают шаблоном (маской). Сейчас данных алгоритм поиска используется многими утилитами, связанными с текстовыми данными, например текстовые редакторы или утилиты поиска. При составлении масок поиска, используется специальный синтаксис, который, обычно, поддерживает операции перечисления, группировки, квантификации. Операции перечисления схожи с операцией ИЛИ, которая используется для перечисления допустимых вариантов. Группировка используется для определения области действия и приоритетности других операторов, например «hello | hallo» и «h(e|a)llo» являются разными образцами, но они оба описывают множество, которое содержит hello и hallo. Квантификация определяет количество включений выражения. Существует несколько типов квантификаторов:

- {*n*,*m*} – от *n* до *m* повторений включительно
- {*n*,} – *n* и более повторений
- {,*m*} – не более *m* повторений
- {n} – ровно *n* повторений
- ? – то же самое, что и {0,1} ноль или один раз
- – ноль, один или любое количество повторений
- + – то же самое, что и {1,} хотя бы один раз.

Таким образом, можно сделать следующие выводы:

1) Выявление тематики научной статьи может быть осуществлено с помощью индексации.

2) Выявление авторов статьи может быть осуществлено с помощью поиска по регулярным выражениям.

# 2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

## 2.1 Описание предметной области

Первый этап проектирования приложения, в нашем случае web-сайта, нужно выполнить анализ предметной области, то есть осветить используемые объекты и провести связи между ними.

Чтобы провести анализ, нами был выбран функциональный подход, который заключается в том, чтобы отталкиваясь от необходимых функциональных требований, определить набор необходимых объектов предметной области.

Несмотря на выбор функционального подхода, для анализа предметной области, в начале проектирования, лучше использовать предметный подход, в котором объекты предметной области выбираются с расчетом, чтобы использовать их при решении множества задач.

## 2.2 Разметка страниц

Наиболее известным и распространенным языком, используемым при разработке web-сайтов, является HTML (Hypertext Markup Language), который представляет собой язык, разработанный исключительно для создания web-документов. Его синтаксис и размещение специальных инструкций, указывают браузеру, как нужно выводить на экран (или не выводить) представленную информацию.

Web-разработчики «общаются» с браузером, с помощью дескрипторов, которые также называют тегами. Количество тегов увеличивается с каждой новой версией HTML. Но в любой версии есть обязательные теги, такие как <html>,<head>,<body>. Обязательные теги отвечают за разделение документа на такие части, как голова и тело. «Голова» содержит информацию для браузеров и поисковых систем. В свою очередь, контейнер body хранит в себе, всё содержание web-страницы, которое отображается в браузере.

## 2.3 Обработка информации

### 2.3.1 Выбор языка программирования

Так как наш проект заключается в применении анализа в отношении научных статей, нам нужно выбрать метод обработки информации и проведения подсчетов. PHP (Hypertext Preprocessor) – наиболее популярный скриптовый язык, используемый для разработки web-сайтов. Также PHP является одним из простейших языков программирования. Но при всей своей простоте, может использоваться как для создания простейших сайтов, так и для крупных проектов. PHP, является серверным языком программирования, то есть когда посетитель посылает запрос на сервер, сервер подключает php-интерпретатор, который в свою очередь обрабатывает найденный php-код и отправляет web-страницу серверу, а тот по назначению, в браузер клиенту. После проведенных манипуляций на сервере, браузер выводит полученную страницу на экран.

Для организации более сложной и защищенной системы можно воспользоваться фреймворком.

### 2.3.2 Выбор фреймворка

Фреймворк (от англ. framework – каркас, структура) – это некий скелет, содержащий в себе множество библиотек, облегчающий разработку продукта. Обычно фреймворк уже включает в себя такие шаблонные этапы разработки, как настройка подключения к базе данных, аутентификацию пользователей, создание каркаса дизайна. В отличие от понятия библиотека, фреймворк строго задает правила организации структуры приложения.

Рассмотрим существующие PHP-фреймворки и проанализируем, какой из них максимально облегчит разработку конкретно нашего проекта. Исходя из огромного количества претендентов, было принято решение провести сравнительный анализ только лишь самых популярных фреймворков. Для получения информации о популярности наших претендентов, мы обратимся к результатам уже проведенного исследования. *Sitepoint.com* – сайт

специализирующийся на программировании. И в прошлом году были опубликованы результаты исследования популярности фреймворков.

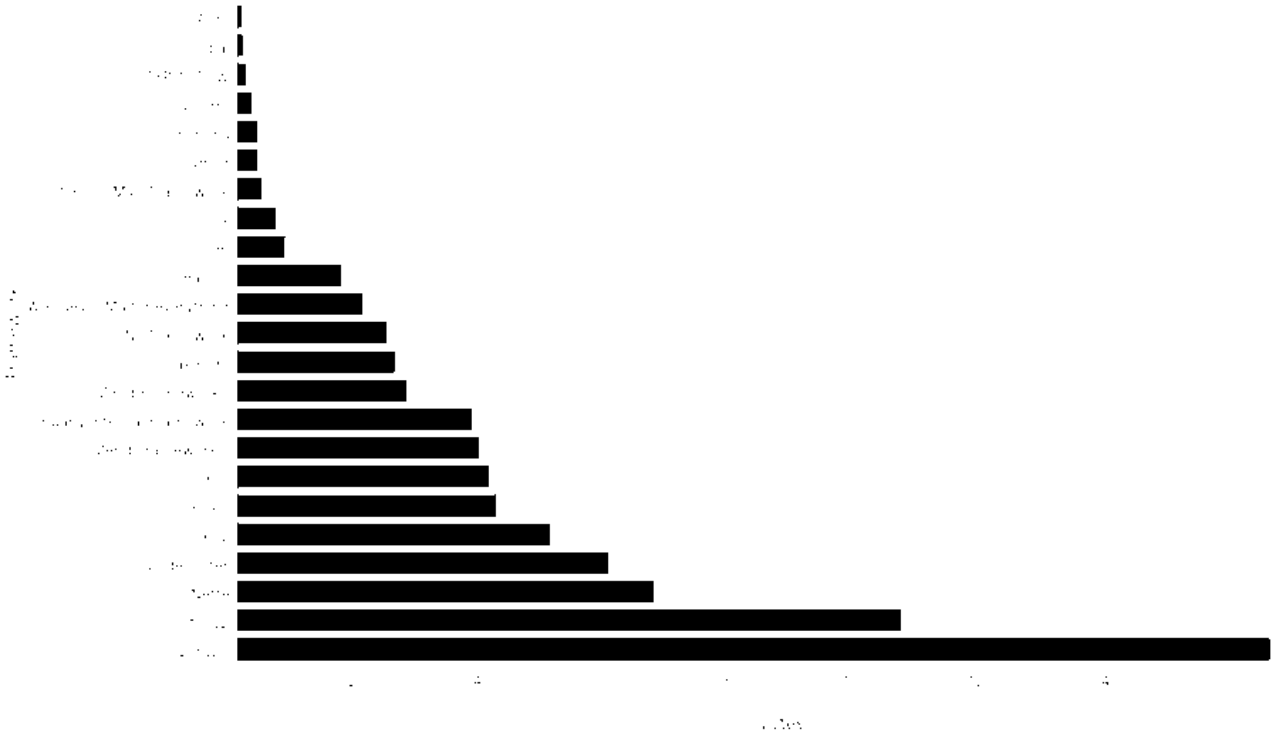

Рисунок 2.1 - Популярность фреймворков в рабочих проектах.

Как можно заметить, Laravel имеет огромное преимущество по отношению к другим фреймворкам. Так же в пятерку самых популярных попали Symfony 2, Nette, Codelgniter и Yii 2. Но это, что касается рабочих проектов, над которыми работают множество людей и имеется возможность вложения дополнительных средств в разработку. В нашем случае лучше опираться на статистику собранную среди персональных проектов.

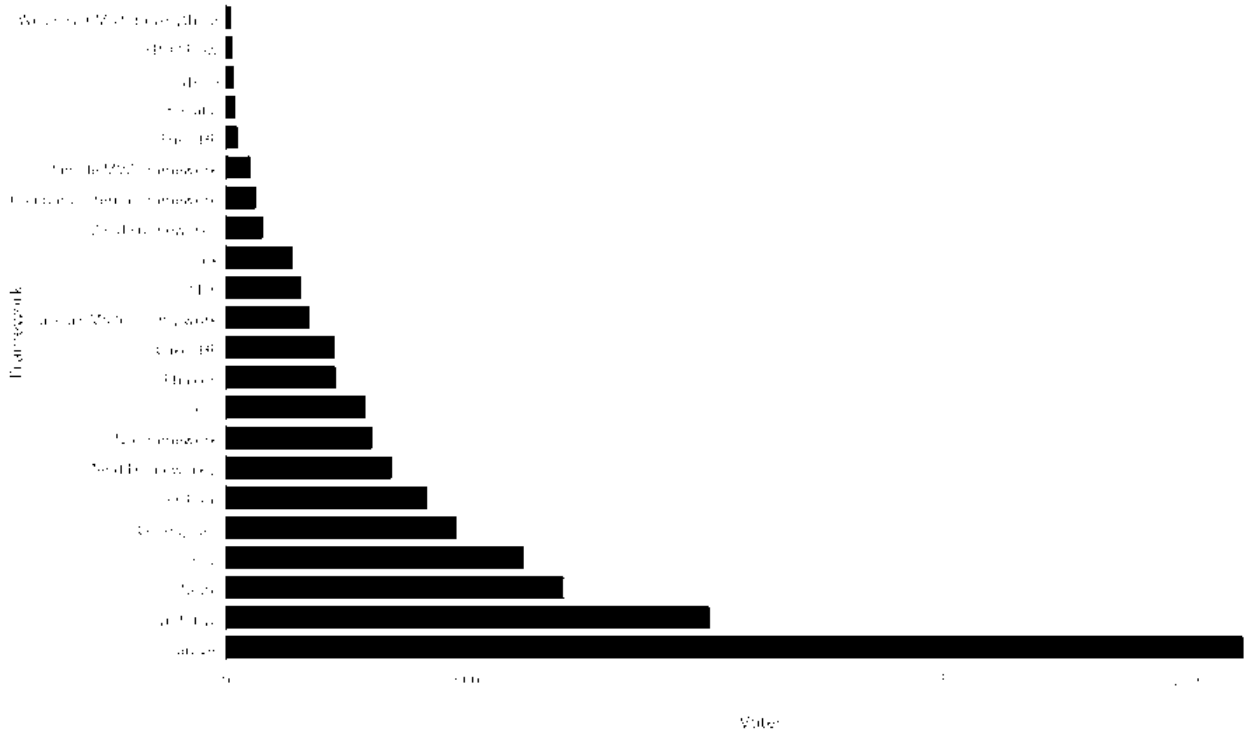

Рисунок 2.2 - Популярность фреймворков в персональных проектах.

Посмотрев на данный график, можно заметить, что Yii 2 используется чаще, чем Codelgniter в персональных проектах. Разберем самые популярные фреймворки и выявим их достоинства и недостатки.

Laravel – это фреймворк для web-приложений с богатым и четким синтаксисом. Данный фреймворк дает возможность облегчить решение многих задач, таких как аутентификация, маршрутизация, сессии и кэширование. Laravel – это сборная версия достоинств из разных фреймворков, а также Ruby on Rails, ASP.NET и MVC.

YII 2 – высокопроизводительный PHP фреймворк для быстрой и качественной разработки ходовых web-приложений. Он обладает широким спектром возможностей, которые позволяют реализовывать проекты типа форумов, порталов, CMS и других крупных проектов. Как и у большей части фрейворков, в основу Yii лежит MVC паттерн(Model – View – Controller). Данный фреймворк обладает одной из самых лучших производительностей среди своего класса. Одним из главных преимуществ, пожалуй, является

возможность расширения. Вы можете дополнить или переписать любой фрагмент изначального кода.

Symfony – свободный фреймворк, написанный на php5, который как и многие использует паттерн MVC. После проведенного анализа и поиска информации было выяснено, что данный фрейворк требует очень высокого уровня понимания и большого опыта работы с ним.

Nette Framework представляет собой каркас с открытым исходным кодом для создания web-приложений в PHP 5 и PHP 7. Он поддерживает AJAX, а также строится на основе MVC. Nette включает в себя множество различных дополнений, многие из них уникальны. Такие как Latte - уникальный компонент являющимся движком для шаблонов.

После проведенного сравнительного анализа популярных фреймворков, выделить самый подходящий не удалось. Было принято решение рассмотреть результаты анализа популярности фреймворков относительно нашей страны.

Таблица 1 - популярность фреймворков в представленных странах.

| Страна | Количество голосов | Рабочие проекты | Голоса | Персональные проекты | Голоса |
|---|---|---|---|---|---|
| США | 819 | Laravel | 219 | Laravel | 293 |
| Чехия | 770 | Nette | 611 | Nette | 639 |
| Украина | 263 | PHPixie | 66 | PHPixie | 67 |
| Российская Федерация | 235 | Yii 2 | 53 | Yii 2 | 72 |

Как мы можем увидеть, Yii 2 является самым популярным фреймворком в нашей стране. Опираясь на информацию полученную из исследования популярных фреймворков, было принято решение остановиться на Yii 2 framework.

После того как выбран язык программирования и фреймворк, для лучшей организации приложения следует перейти к проектированию базы данных.

## 2.4 Проектирование базы данных

Процесс проектировки базы данных можно разделить на два подраздела: физическое проектирование и логическое моделирование. Результат логического моделирования это логическая модель данных, которая обычно выражается диаграммой «сущность-связь». Результатом физического проектирования является готовая база данных, по этому правильнее будет начать с логического моделирования.

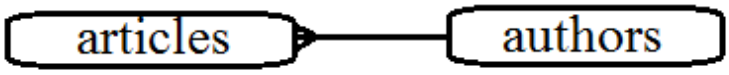


Рисунок 2.3 - Концептуальная модель данных.

### 2.4.1 Логическое моделирование

Логическое моделирование подразумевает под собой разработку будущей базы данных. Чтобы создать логическую модель базы данных, нужно описать объекты, которые позже будут внесены в базу. Логическая модель состоит из сущностей, их атрибутов и связей между ними. Если провести аналогию с физической моделью, то сущностью будет являться таблица в базе данных, атрибут – поле таблица. В свою очередь сущность состоит из экземпляров этой сущности, который с физической точки зрения являются записями в таблице. Соответственно правилам заполнения реляционной СУБД, записи в таблице должны быть уникальными, проведя аналогию на логическую модель, экземпляры сущности должны быть уникальными, то есть полный набор значений их атрибутов не должен повторяться. Аналогично полям таблицы, атрибуты могут быть ключевыми и не ключевыми. На этапе логического моделирования базы данных, каждому атрибуту должен быть присвоен примерный тип данных. Точный тип данных будет присвоен уже на этапе физического моделирования, когда все нюансы будут определены.

Одним из главных аспектов информационного обеспечения является информационная база, представляющая собой объединение данных,

помогающих при решении разнообразных задач. Большая часть процесса разработки базы данных состоит в моделировании данных, целью которой является обеспечение разработчика информационной системы концептуальной схемой БД в форме одной или нескольких локальных моделей, которые могут быть отображены в других системах баз данных. Диаграммы «сущность-связь» являются наиболее распространенным средством для создания моделей данных.

Сущность – многочисленные экземпляры объектов, включающие в себя общие атрибуты или характеристики. Абсолютно каждый объект системы может и должен быть представлен единой сущностью, которая должна иметь уникальный идентификатор.

Атрибут – любая характеристика сущности, учитываемая при рассмотрении конкретной области, и предназначенная для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности. Атрибут представляет собой разновидность характеристик, связанных с объектами.

Связь – проименованная ассоциация между сущностями. Притом, каждый экземпляр сущности может быть связан с любым количеством экземпляров другой сущности.

Существует некоторое количество разновидностей связей между экземплярами:

Один-к-одному – данный тип связей подразумевает под собой связь между двумя экземплярами. Часто, данный вид связи указывает на возможность объединения сущностей.

Один-ко-многим – самый популярный тип связи, который означает, что экземпляр первой сущности связан со многими экземплярами второй сущности. При использовании данного типа связи, сущность, использующая в данной связи один экземпляр, называется родительской, а другая сущность зовется дочерней.

Много-ко-многим – временный тип связи, использующийся только лишь на начальных этапах разработки. Данный вид связи работает по следующему

принципу: каждый экземпляр первой сущности может быть связан с каждым экземпляром второй сущности и наоборот.

В ходе моделирования логической модели данных были выделены следующие сущности и атрибуты:

- Статьи (идентификатор, название, автор, тема исследования, раздел исследования, дата создания, дата редактирования, соавторы).
- Авторы (идентификатор, фамилия, имя, отчество, страна, город, степень, организация, ORID).

Решения, принятые на данном этапе разработки, будут подвергнуты изменениям при физическом проектировании и дополнены при дальнейшей разработке функциональной части приложения.

Следующим этапом будет описание и выбор архитектуры проектируемой базы данных.

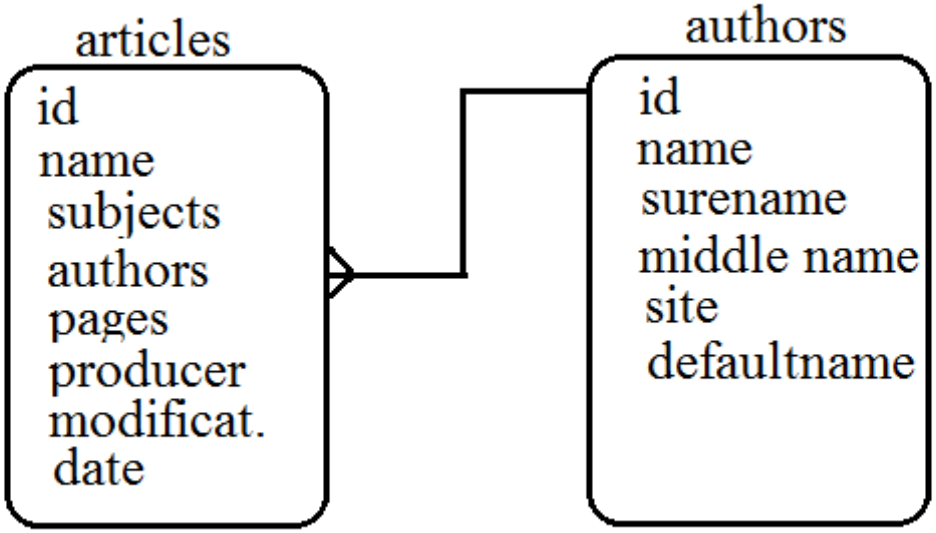


Рисунок 2.4 - Логическая модель данных.

### 2.4.2 Выбор архитектуры базы данных

Существует несколько различных моделей представления данных. Основными считаются иерархическая, сетевая, реляционная модели. Для

создания базы данных больше всего подойдет реляционная. Архитектура базы данных будет строиться по принципу клиент-сервер.

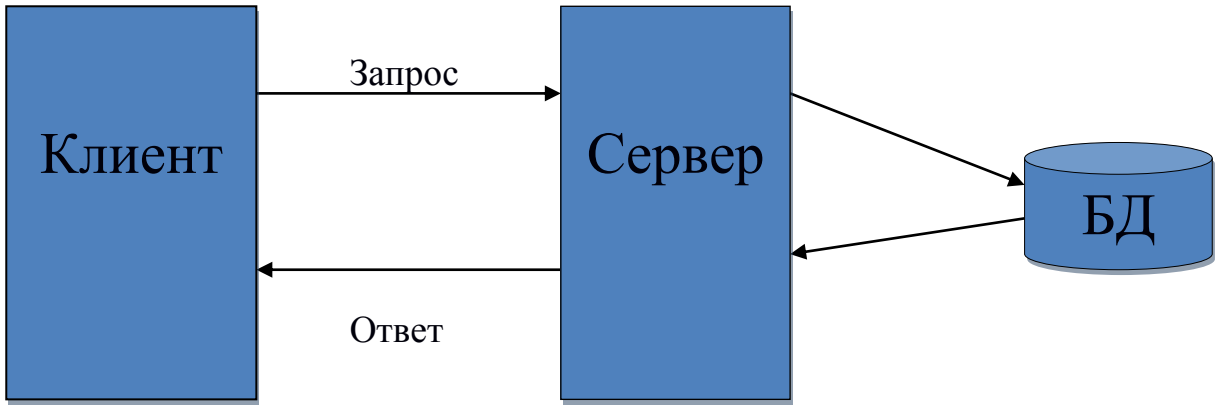


Рисунок 2.5 - Двухзвенная клиент-серверная архитектура.

Двухзвенная архитектура используется при условии, что в клиент-серверной системе сервер отвечает на запросы клиента, используя только лишь свои ресурсы. То есть сервер не прибегает к помощи сторонних приложений или других ресурсов, для выполнения какого-либо запроса от клиента.

Расположение компонентов определяет основные модели взаимодействия между клиентом и сервером при использовании двухзвенной архитектуры.

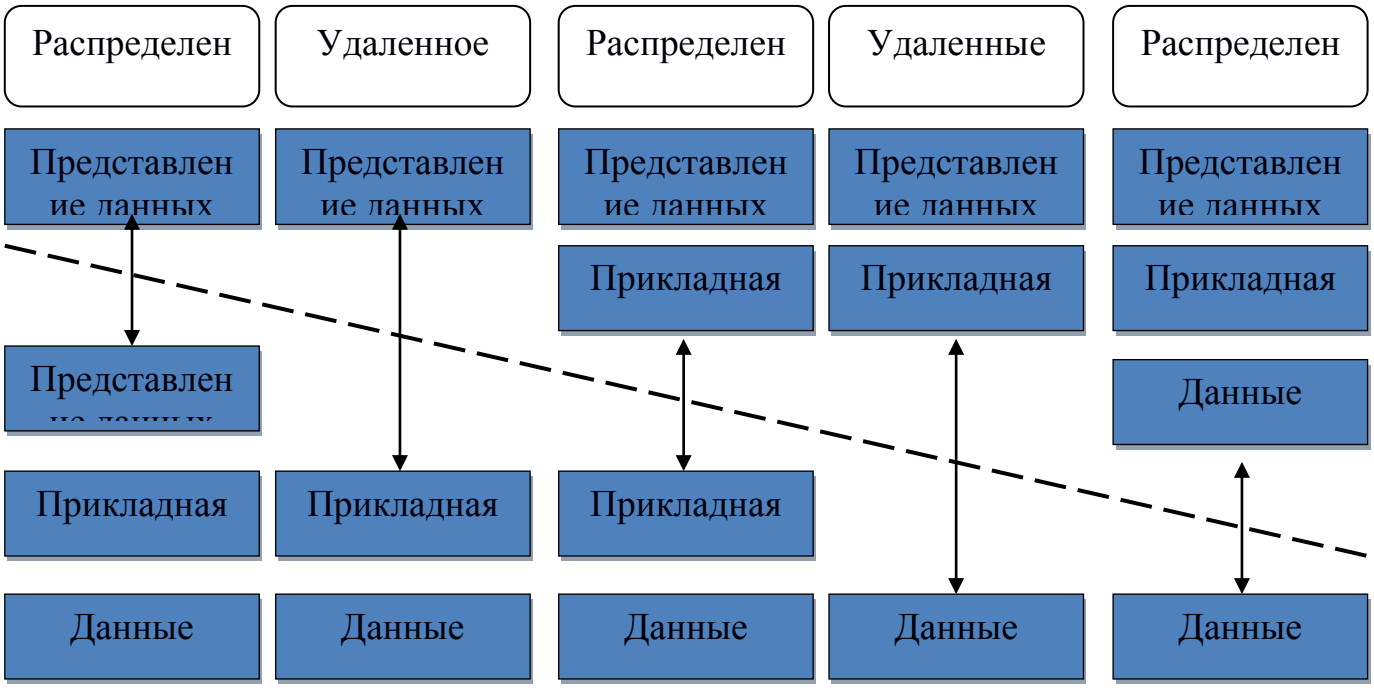


Рисунок 2.6 - Модели клиент-серверного взаимодействия.

Серверная часть базы данных, по своей сути, это программа, запускающаяся на серверной машине и создающая возможность клиентам иметь доступ к базе данных. Так же стоит заметить, что один сервер может обслуживать сразу несколько клиентов.

Такая архитектура подразумевает выполнение большинства действий с данными сервером базы данных. SQL запросы, исходящие от пользователя или приложения поступают к серверу базы данных, который в свою очередь выполняет полученные запросы и передает данные обратно клиенту. Архитектура типа клиент-сервер имеет большое количество положительных свойств. Например, в большинстве случаев, функции вычислительной системы можно распределить между независимыми компьютерами в сети, что позволяет максимально упростить модернизацию или при необходимости ремонт сервера, не создавая неудобства для клиента. Удобство хранения основной информации на сервере, довольно очевидное преимущество, ведь сервер, обычно, имеет гораздо лучшую защиту. Плюс на сервере намного выгоднее организовывать

контроль полномочий, чтобы давать правовое превосходство, только клиентам, имеющим соответствующие полномочия. К сожалению, как и у любой системы, у клиент-серверной архитектуры есть свои недостатки. Поломка сервера может вызвать цепную реакцию и перевести всю систему в неработоспособное положение. По этому, обычно обеспечением работоспособности серверной части занимается отдельный класс специалистов – системные администраторы.

Архитектура клиент-сервер позволит получать данные из базы данных любому пользователю, в каком бы месте он не находился. При этом основные данные будут в полной сохранности на сервере.

### 2.4.3 Выбор системы управления базой данных

Выбор системы управления базой данных – один из самых главных этапов при разработке приложения. Выбранная система управления должна удовлетворять все текущие и будущие потребности приложения. При этом стоит учитывать большое количество нюансов, таких как финансовые затраты, возможности оборудования и многие другие.

Система управления базами данных должна соответствовать основным функциональным требованиям. Мобильность – это возможность работы системы не зависимо от среды, в которой она используется. Масштабируемость – это способность системы управления соответствовать росту информационной системы. Рост может заключатся как в увеличении количества пользователей, так и в увеличении количества обрабатываемых данных. Также при анализе различных СУБД стоит обратить внимание на возможности средств разработки, представляемыми разработчиками СУБД. Некоторые системы включают в себя средства автоматического проектирования баз данных, что может оказаться полезным дополнением. Мультиязычность, на первый взгляд не несет в себе никаких функциональных преимуществ, но присутствие родного языка в интерфейсе СУБД может значительно ускорить разработку приложения. Широкая известность той или иной системы управления базами данных может

послужить приятным бонусом, ведь при возникновении каких-либо затруднений при разработке, вы сможете найти большое количество документации и различных инструкций по многим ситуациям.

Понятие надежности системы может нести множество различных смысловых аспектов. Это и стабильность работы, невозможность несанкционированного доступа третьих лиц и отсутствие непредвиденных сбоев. Эффективность механизма восстановления после сбоев напрямую влияет на жизнеспособность системы. Также в результате одного из сбоев может возникнуть проблема с целостностью данных либо данные могут быть безвозвратно уничтожены. На этот случай в большинстве систем управления базами данных предусмотрено резервное копирование, что является одним из главных защитников от ошибочных удалений таблицы и устранения последствий несанкционированно доступа. Так как большинство действий производимых внутри баз данных являются транзакциями, учитывая главное правило транзакций – либо транзакция выполняется полностью, либо не выполняется вообще. Это значит, что в случае сбоя в ходе выполнения транзакции она должна быть анулирована. Использование базы данных подразумевает под собой хранение некой информации, которая должна быть скрыта от всеобщего взора. Для предотвращения несанкционированного доступа к системе, используется служба идентификации пользователей.

Для более глубокого анализа были выбраны две системы управления базами данных: MySQL и PostgreSQL. Проведем сравнительный анализ указанных выше систем на основе основных требований к проектируемой базе данных.

Таблица 2 – Сравнительный анализ СУБД

| Критерии оценки | PostgreSQL | MySQL |
|---|---|---|
| Размер базы данных не более 1 Гб | + | + |
| Поддержка доступа большого количества пользователей | + | + |
| Защита сервера | + | + |
| Защита данных | + | + |
| Мощность языка SQL | + | - |
| Низкие технические требования | + | + |
| Простота настройки | + | + |
| Поддержка дальнейшего развития базы данных | - | + |
| Платформа использования | (Unix/Linux only) | (Windows+Linux) |

На основе проведенного сравнительного анализа была выбрана MySQL. Данная СУБД является относительно небольшой и быстрой реляционной системой управления базами данных. Плюсом является стоимость использования данной СУБД. Данный выбор является оптимальным с точки зрения будущего развития проектируемой базы данных. В нашем проекте, предположительно, большое количество пользователей должно будет иметь доступ к информации и быстро получать ответы на разнообразные запросы.

### 2.4.4 Физическое моделирование

Физическая модель базы данных определяет способ размещения данных на носителях, а также алгоритм результативного доступа к ним. Учитывая то, что

система управления базами данных работает под началом операционной системы, которая соответственно оказывает большое влияния на принципы хранения и управления данными. Процесс физического моделирования базы данных включает в себя следующие пункты: выбор метода организации БД, разработку спецификации внутренней схемы и описание отображения концептуальной схемы во внутреннюю. В наше время многие СУБД не предоставляют обширного выбора при проектировании физической модели. Остается лишь несколько настроек, которые мы сможем адаптировать под свою систему, такие как: выбор схемы размещения данных (разделить по файлам или RAID-массив) и определение числа, типа индексов.

Такой атрибут как выбор способа хранения базы даны, обычно, определяется в автоматическом режиме, основываясь на спецификации базы данных. Так как физическая модель данных строится на основе логической модели данных, каждый объект логической модели связан с объектом физической модели. Сущность логической модели соответствует таблице в физическом представлении, экземпляр сущности – строке в таблице, а атрибут – колонке таблицы. Тип всех остальные объекты, таких как индексы, представления, триггеры, процедуры, обычно зависит от СУБД.

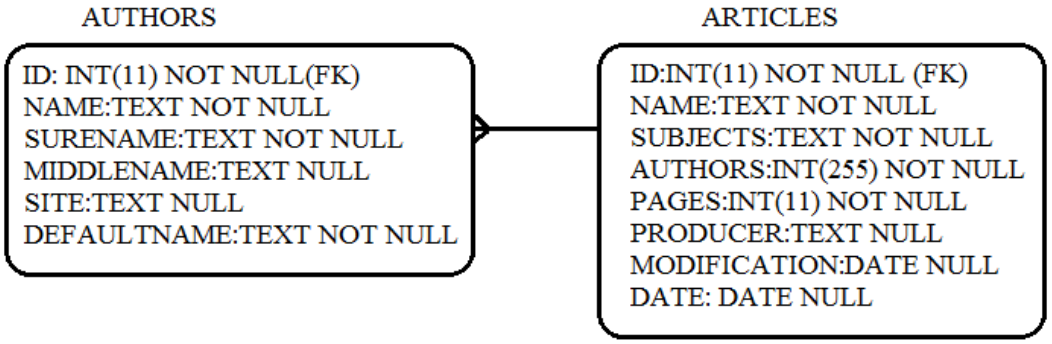


Рисунок 2.7 – Физическая модель данных.

## 2.5 Анализ файла

### 2.5.1 Формат файла статьи

На вход нашему приложению будет подаваться файл научной статьи в формате PDF.

Формат PDF отличается от других привычных форматов документов, ведь он был задуман, как формат документов, который будет абсолютно статичен. Файлы данного формата будут отображаться всегда одинаково, не зависимо от устройства. В отличие от других типов человеко-читаемых документов, PDF не предназначен для редактирования и чаще всего создается из других форматов путем машинной обработки.

PDF – формат, который поддерживает несколько основных типов данных, некоторые из них понадобятся нам для поиска информации – это строки, потоки, массивы и объекты. Строки были добавлены в данный формат из PostScript, как следствие этого, в PDF строка является последовательностью 8-битных символов, заключенных в круглые скобки. Эта информация будет ключевой при поиске текстовых данных внутри документа. Потоки, по своей сути являются последовательностью данных и характеризуются, как минимум, своей длиной (/Length Z). Массивы, в которых также могут находиться текстовые данные, заключаются в квадратные скобки. Например: [(Hello),12(o)]. Осталось вплотную рассмотреть объекты внутри PDF документов. Этот тип данных может содержать в себе любой другой тип данных, показателями начала и конца объекта являются слова «obj» и «endobj».

### 2.5.2 Получение текста из файла

Получение текста из PDF файла будет осуществлено поэтапно, первым этапом является чтение данных из полученного файла в строку. Учитываем, что полученная строка содержит не только текстовые данные, а так же разнообразные hex вставки, шрифты и многое другое. Следующим шагом мы разделяем строку на объекты по ключевым словам, обозначающим начало и конец объекта. Далее начинается цикл, который перебирает каждый найденный

объект. В каждом объекте нужно проверить наличие потока данных, поиск осуществляется по признаку ключевого слова окончания потока данных. Далее, для ускорения процесса, осуществляются минимальные отсечения не интересующих нас данных, например опций объекта. Полученные данные будут подвергнуты декодингу с учетом символьных трансформаций. Под словом трансформация подразумевается перевод символа в hex-представлении в другой или даже некую последовательность. Данные между beginbgchar и endbfchar преобразоывают один hex-код в другой (или последовательность кодов) по отдельности. Между beginbfrange и endbfrange организуется преобразование над последовательностями данных, что сокращает количество определений. Вначале обрабатываются отдельные символы. Перед каждым списком данных, есть число обозначающее количество строк, которые нужно прочитать, его нужно учитывать. Далее переходим к последовательностям, по документации последовательности бывают двух видов, которые преобразовываются различными способами. Переводим данные в десятеричную систему счисления, так проще будет пройти все данные циклом. Добавляем все элементы последовательности в массив трансформаций. После получения всех данных трансформаций отдельных символов, можно перейти к следующему этапу – получение текста из «грязных» данных. В PDF «грязные» текстовые строки могут выглядеть как отображено на рисунке 2.8.

(Hello)12(World)

Рисунок 2.8 – Пример «грязных» данных.

В данном примере, данных, заключенные в круглые скобки, являются текстовыми, а «12» являет собой величину пробела.

Следующим этапом, нам нужно разобрать грязные данные, параллельно сохраняя текстовые данные. Существует два представления текста, когда текст

находится в hex-представлении и в plain-представлении. В первом варианте данные заключены в угловые скобки, а во втором варианте, данные представлены в круглых скобках. Посимвольно сканируем текст и определяем, что делать с выбранным символом. В зависимости от выбранного символа, мы определяем его в hex-представление или в plain-представление. Есть и третий вариант – это если выбранный символ является символом экранирования, следовательно, следующий за ним символ нужно записать в соответствии с его значением. После проделанных манипуляций с текстом, на выходе мы получим неструктурированный текст.

### 2.5.3 Регулярные выражения

Для того чтобы структурировать полученные текстовые данные, воспользуемся информационным поиском. Информационной потребностью в данном примере будет являться информация об авторе статьи. Поиск по текстовым данным будет осуществляться с помощью регулярных выражений. Регулярны выражения это формальный язык поиска, основанный на использования метасимволов. Для поиска используется шаблон (англ. Pattern), состоящий из символов и метасимволов, образующих правило для поиска. Данный алгоритм поиска был выбран из-за того, что авторы статей, обычно, указываются, используя одну и ту же маску.

### 2.5.4 Индексирование

После получения данных об авторах, следует преступить к извлечению из текста тематики статьи. Поиск с использованием регулярных выражений не поможет достичь нужного результата, ведь мы не сможем задать маску ключевым словам, которые нам не известны заранее. Для получения нужной информации, следует провести индексацию слов, используемых в статье. Для этого мы будем перебирать все слова, параллельно добавляя новые в таблицу. И в результате мы используем полученную статистику для определения тематике статьи. Логично предположить, что стоит исключить предлоги, частицы, местоимения из результатов собранной статистики. Так как с точки

зрения программирования, слово – это набор символов, и мы не разделяем его на части, в полученную статистику могут быть добавлены несколько склонений одного и того же слова, что пагубно повлияет на результат. Для более точного подсчета количества повторений, слова должны сравниться по некоторой части слова. Для сравнения возьмем первые семьдесят процентов слова, округляя в меньшую сторону. В тематику статьи запишем самые часто встречающиеся слова.

### 2.5.5 Атрибуты файла

Так же, некоторые данные о статье можно получить, просмотрев свойства файла. Такие свойства как дата создания и количество страниц указываются в стандартных свойствах файла.

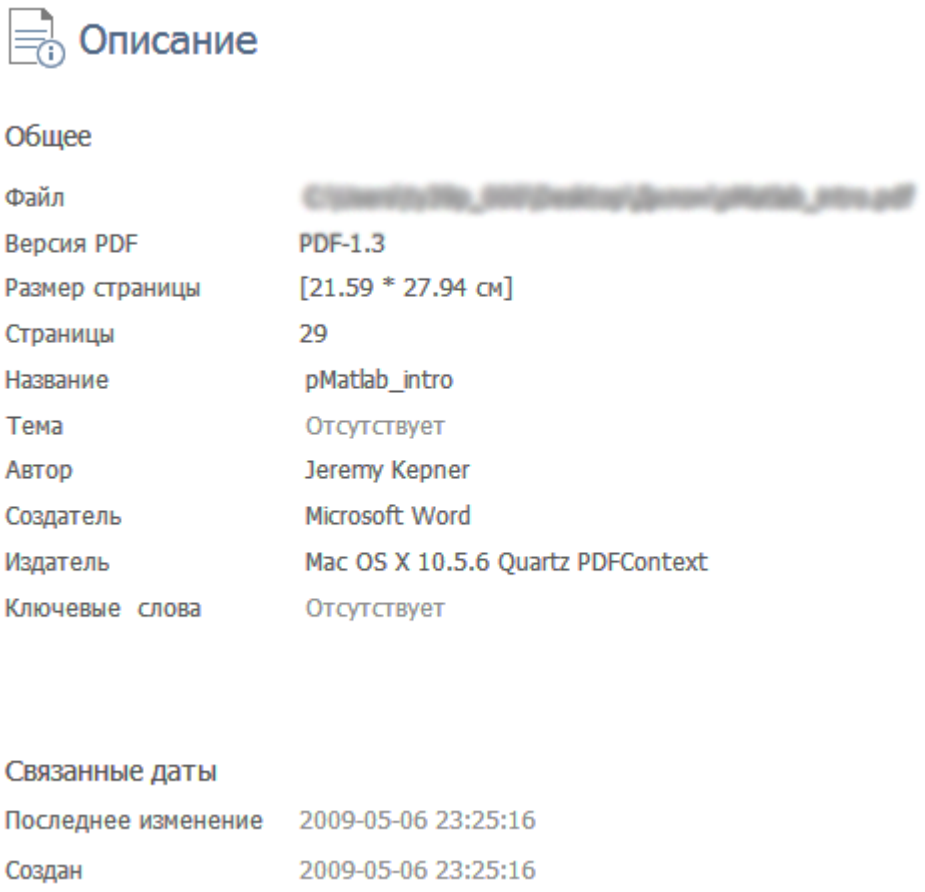


Рисунок 2.9 – Атрибуты PDF файла.

С помощью получения информации из атрибутов файла, мы можем получить дополнительные данные о статье и занести их в базу данных.

Таким образом, можно сделать следующие выводы:

1) Данные будут храниться в базе.
2) Оптимальная СУБД для данного проекта – MySQL.
3) Проект будет реализован используя Yii 2 framework.

# 3 ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

Принимая в расчет, что наше приложение должно быть, реализовано на web сервере, появилась необходимость установить web сервер на локальную машину. Так как большинство серверов схожи по своим функциональным возможностям, выбор пал на OpenServer, в который включено большое количество дополнительных сервисов, упрощающих разработку приложения. Архитектура расположения каталогов построена таким образом, что динамические и статические данные разделены по разным папкам. Данная структура создана для упрощения синхронизации данных между различными копиями комплекса и экономии места при резервном копировании.

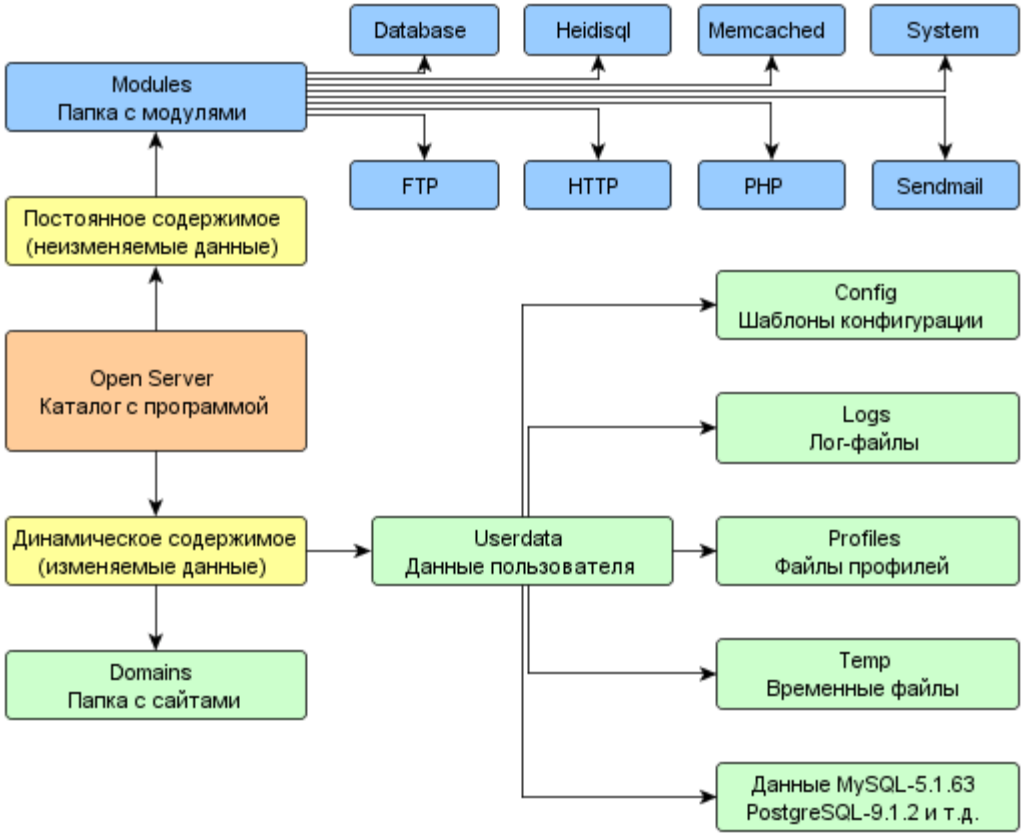


Рисунок 3.1 - Архитектура OpenServer.

## 3.1 Установка сервера

Установку нашего сервера на локальную машину начинаем с прочтения мануалов, представленных на официальном сайте разработчика. Для начала обратимся к системным требованиям и убедимся в том, наша локальная машина удовлетворяем им.

Системные требования:

- Необходимый минимум системных ресурсов: 200 Мб RAM и 1 Гб на HDD;
- Windows (32-bit или 64-bit): Windows 10 / Windows 8 / Windows 7 / Windows Server 2008 / Windows Vista / Windows XP SP3;
- Установленный набор библиотек Microsoft Visual C++ 2005-2008-2010 Redistributable Package x86;

Используемый компьютер полностью удовлетворяет данным системным требованиям. В процессе обновлений сервиса, выбранное нами программное обеспечение стало портативным комплексом, не требующим установки. Достаточно лишь разместить дистрибутив на локальном компьютере и начать процесс настройки.

После запуска исполняемого файла Open Server x64.exe от имени администратора, программа автоматически сворачивается в трей, отображая статус сервера цветом иконки. Изначально наш сервер находится в выключенном состоянии.

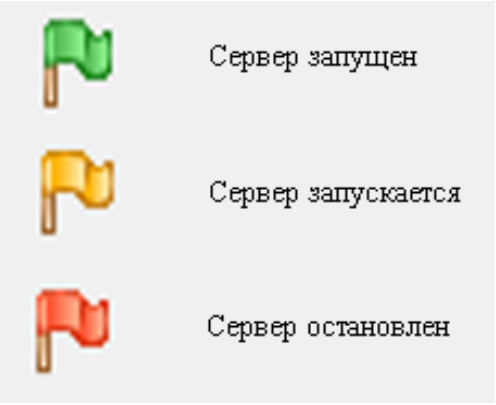


Рисунок 3.2 – Состояние сервера.

Перед запуском нужно перейти в раздел настроек, чтобы убедиться, что стандартные настройки нас устраивают. В окне настроек находится большое количество подразделов таких как: основные, сервер, модули, меню, кодировки, FTP сервер, почта, закладки, автозагрузка, разное, планировщик заданий, алиасы, домены. Для начала в разделе «основные» выберем язык «Russian» языком интерфейса приложения, если он не был выбрал изначально. Перейдя в следующую вкладку настроек, мы будем должны настроить непосредственно наш сервер. Выбрать удовлетворяющий нас IP-адрес сервера, так же настроить порты во избежание ошибок при запуске. Следующая ступень настроек нашего сервера является самой важной с функциональной стороны, это настройка модулей. В данном разделе нам нужно выбрать какие версии представленных модулей нужно использовать на данном сервере. Выбираем во вкладке «HTTP» «Apache-2.2» и ставим галочку «вести лог запросов». Во вкладке PHP мы должны указать версию удовлетворяющую, используемый нами, Yii 2 фреймворк. На официальном сайте разработчиков Yii 2, указанно, что данный фреймворк работает на версии PHP от 5.4, следовательно, выбираем версию выше 5.4 или 5.4 . Далее нам предложено выбрать СУБД MySQL / MariaDB иои PostgreSQL. Так как мы уже провели сравнительный анализ мы выбираем MySQL, версию можно выбрать 5.5 , как самую стабильную на данное время. Следующие вкладки в меню настроек не так важны и не несут никакой функциональной значимости для нашего проекта. После настройки сервера мы запустим его и убедимся, что мы всё верно настроили и не будет проблем с запуском сервера. Увидев зеленый флажок в трее, мы убедились, что сервер успешно запущен. Следующим шагом реализации нашего приложения будет установка и настройка Yii 2.

### 3.2 Установка фреймворка

Сборка Yii 2 поставляется в двух вариантах, basic и advanced. Судя по названию basic – это минимальная сборка, а advanced продвинутая сборка, включающая в себя уже готовые решения некоторых задач. Еще одно

преимущество продвинутой сборки, это разделение пользовательской части (frontend) и модуля управления (backend), именно это различие сборок сподвигло нас установить именно продвинутый вариант фреймворка.

После установки продвинутого шаблона мы получили некоторые возможности, которые помогут нам при разработке приложения:

- Форма регистрации и входа пользователей;
- разделенные области пользователей и администраторов;
- встроенная интеграция Twitter Bootstrap;
- автоматическая генерация моделей, контроллеров и отображений;
- стандартная функция восстановления пароля.

После установки фреймворка и запуска сервера, мы можем посмотреть, что же у нас получилось. Для этого нужно зайти по адресу, выбранному в настройках сервера, к тому же необходимо указать порт. Зайдя по нужному адресу, мы  увидим соответствующую страницу.

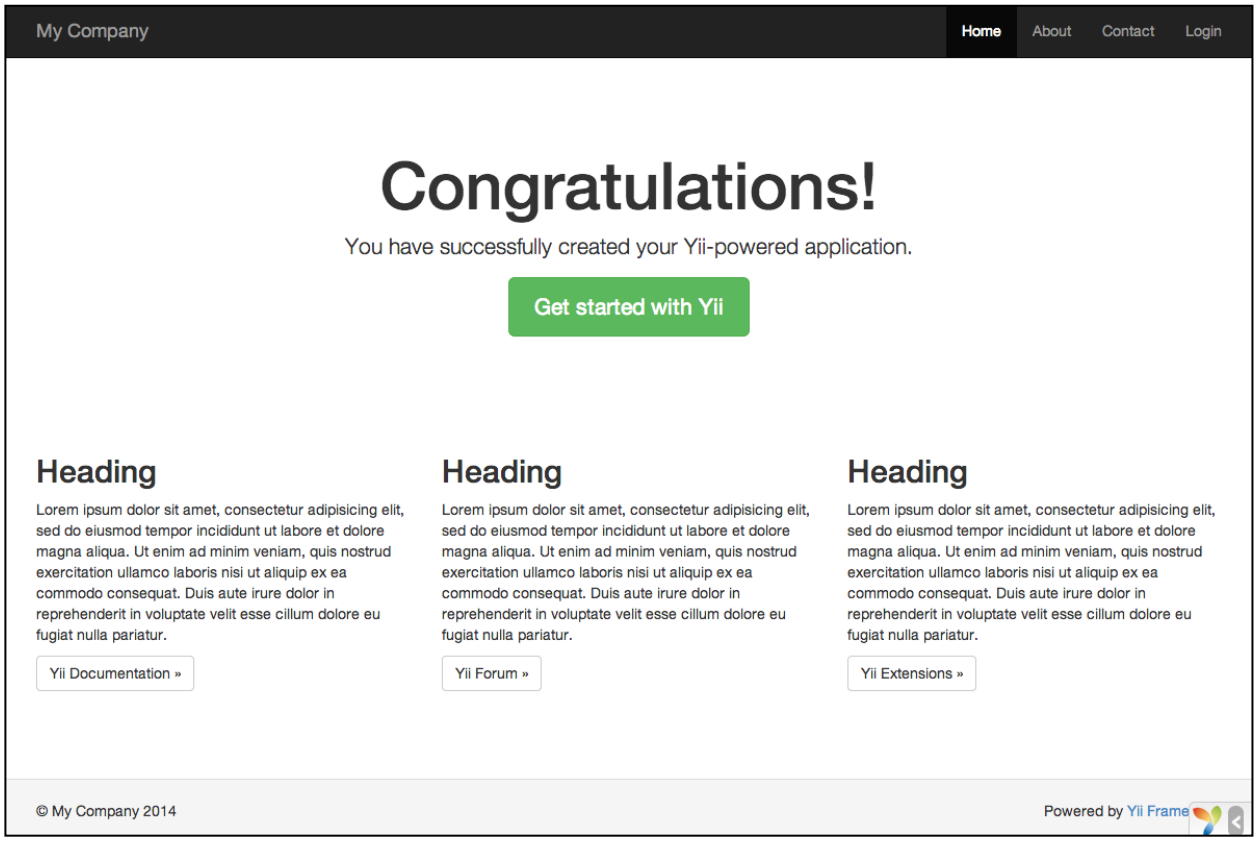


Рисунок 3.3 – Вступительная страница.

Как мы знаем, Yii framework использует MCV систему. То есть состоит из моделей, контроллеров и отображений. Для создания модели используем встроенный в Yii сервис, Gii. Данный сервис является web-ориентированным генератором кода для приложений на Yii. Его можно использовать для быстрого создания моделей, форм, модулей, CRUD и других. Но для создания моделей, нужно подключить наш фреймворк к базе данных. Для этого нужно в папке config, в файле bd.php ввести данные подключения к нашей базе. После успешного подключения фреймворка к базе данных, нужно создать базу данных через СУБД. После выбора активной базы данных внутри нашего фреймворка можем приступить к созданию модели, через Gii генератор.

### 3.3 Реализация приложения

После входа в сервис генерирования, нам нужно выбрать генератор моделей. И ввести необходимую информацию, касающуюся создаваемой модели. В поле «Table Name» нужно ввести название таблицы в базе данных, в которой будет храниться вся информация выбранной модели, название класса модели создастся автоматически исходя из названия таблицы. После введения всех данных и создания таблицы в СУБД, нам будет предложен предпросмотр сгенерированной модели, после которого мы сможем подтвердить генерацию модели. Далее, чтобы производить операции создания, чтения, удаления и обновления, нам нужно воспользоваться CRUD генератором, включенным в фреймворк. После входа в генератор мы вводим пути к создаваемым моделям и контроллерам.

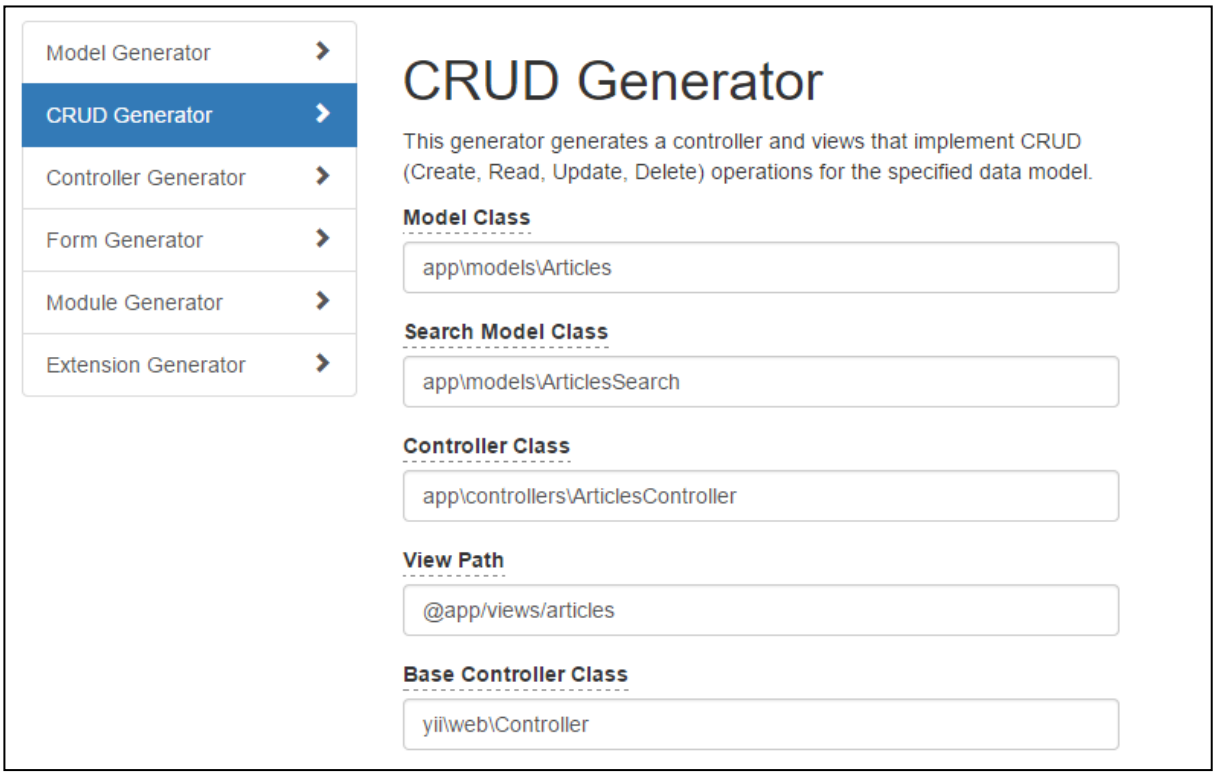


Рисунок 3.4 – CRUD генератор.

После ввода необходимой информации, нам будет предложена возможность предпросмотра генерируемых файлов, так же выбор, какие файлы следует сгенерировать, а какие не нужны на данном этапе разработки, или же будут написаны вручную.

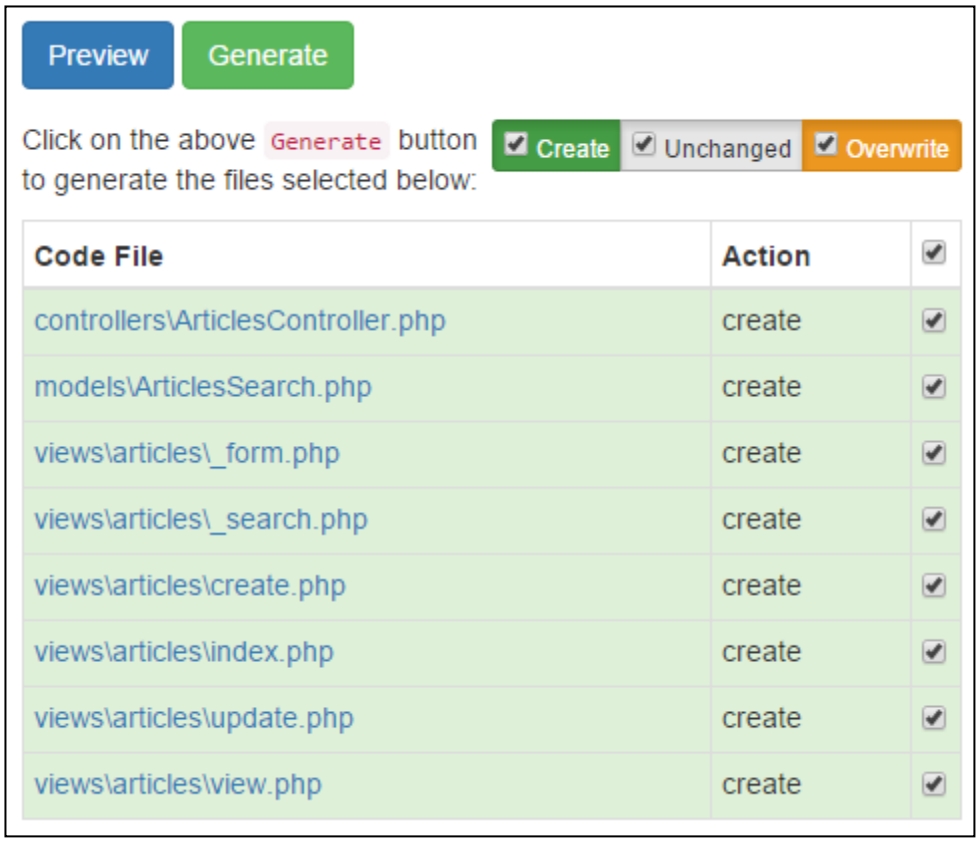


Рисунок 3.5 – Сгенерированные файлы.

На данном этапе были сгенерированы файлы всех трех типов архитектуры MVC: модель, контроллер и представления. Как мы можем убедиться на рисунке 14, все созданные файлы были созданы и распределены по своим местам.

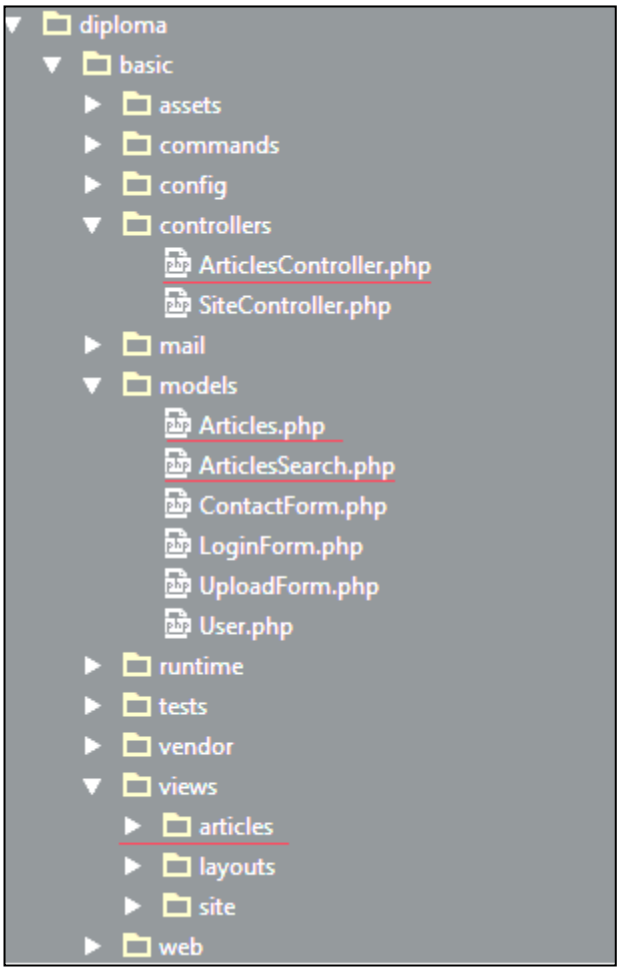


Рисунок 3.6 – Сгенерированные файлы.

Далее следует перейти к организации доступа, для этого добавим ссылку на созданное представление в меню. Для этого нужно открыть файл отображения нашего меню, которое находится в файле app/views/layouts/main.php и добавить в пункт навигационного виджета нашу ссылку. В результате мы получим ссылку на созданную страницу, перейдя на которую мы увидим таблицу со статьями.

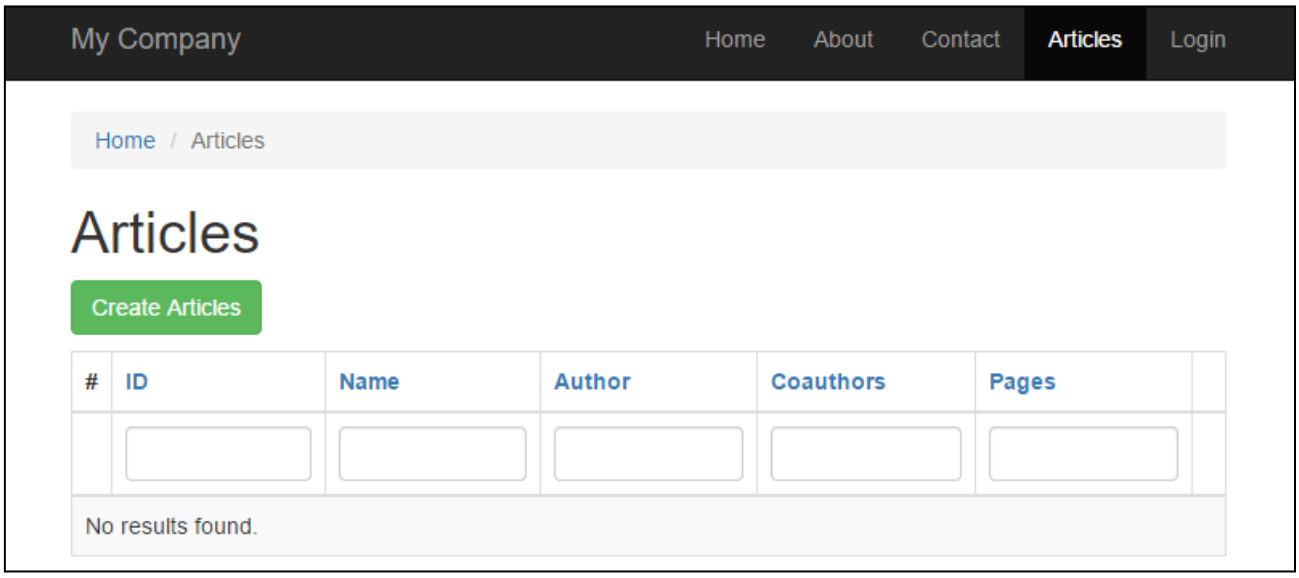


Рисунок 3.7 – Страница управления статьями.

Аналогичным методом генерируем таблицу с авторами.

### 3.4 Реализация алгоритма

Разрабатываемый алгоритм заключается в парсинге данных из файла формата PDF. Основной задачей является выявление автора статьи, названия и дату создания. За основу будет взят уже готовый плагин анализа PDF файла, выявляющий количество страниц, дату создания и программу, использовавшуюся при создании документа. Данный плагин будет интегрирован в наш проект по средствам пакетного менеджера composer. Алгоритм разбора файла на части заключается в грамотном применении функций работы с PDF, предусмотренных в PHP, начиная с версии 4.0.5.

Формат PDF отличается от других привычных форматов документов, ведь он был задуман, как формат документов, который будет абсолютно статичен. Файлы данного формата будут отображаться всегда одинаково, не зависимо от устройства. В отличие от других типов человеко-читаемых документов, PDF не предназначен для редактирования и чаще всего создается из других форматов путем машинной обработки.

### 3.4.1 Реализация алгоритма получения текста из файла

Реализация алгоритма получения текстовых данных из файла будет состоять из нескольких этапов. Первым этапом будет чтение содержимого файла в строку, используя функцию file_get_contents. После получения данных из файла, нам нужно разделить полученные данные на объекты, используя метод поиска по текстовым данным -0 preg_match_all("#obj(.*)endobj#ismU", $infile, $objects). После получения массива объектов, нужно обойти все объекты и найти потоки внутри объектов, используя ту же функцию preg_match_all, только используя другой атрибут поиска. Внутри объектов могут находиться разнообразные данные, которые не будут использоваться в нашей программе, такие как шрифты. Исходя из этого, мы уберем все ненужные данные используя следующие методы: preg_match(“#<<(.*)>>#ismU”, $object, $options); для поиска этих данных и preg_replace, для того чтобы убрать их. Далее расшифровываем полученный текст. Если у текущего потока есть свойство Filter, то он точно сжат или зашифрован. Иначе, просто возвращаем содержимое потока обратно. Если же поток оказывается зашифрованным, нужно определить тип кодировки. Для этого нужно исследовать ключи методом, показанным на рисунке 3.8.

```
foreach ($options as $key => $value) {
            if ($key == "ASCIIHexDecode")
                $_stream = decodeAsciiHex($_stream);
            if ($key == "ASCII85Decode")
                $_stream = decodeAscii85($_stream);
            if ($key == "FlateDecode")
                $_stream = decodeFlate($_stream);
        }
```

Рисунок 3.8 – Метод определения типа кодировки текста.

По результатам определения кодировки текста, мы выбираем подходящий алгоритм декодировки текста и получаем «чистый» текст.

### 3.4.2 Реализация алгоритма получения информации об авторе из текстовых данных статьи

Чтобы получить данные об авторе, мы будем использовать поиск с использованием регулярных выражений. Как показал анализ большого количества научных статей на русском языке, большинство авторов указанны с использованием одного и того же шаблона. И поиск будет осуществлен с использованием алгоритма, указанного на рисунке 3.9.

```
$pattern = "/[А-Я]\s??\.+[А-Я]+\s??\.+\s??\s??[А-Яа-я]+/";
preg_match_all($pattern, $stpage, $matches);
```

Рисунок 3.9 – Алгоритм выявления авторов статьи.

### 3.4.3 Поиск дополнительной информации о статье

Дополнительную информацию о статье мы будем получать из свойств файла. Свойства файла могут быть получены с помощью стандартных методов PHP. Данные полученные из свойств файла, будут записаны в таблицу научных статей.

После реализации алгоритма найденная статья будет записана в таблицу статей. Так же новый автор, появившийся в системе, будет занесен в таблицу авторов. Как показано на рисунках 3.10 и 3.11, в поле автора статьи будет записан идентификационный номер автора, а выведены будут его имя и фамилия. Данная система связей используется для более качественной привязки статей к одному автору.

| id | name | author | coauthors | pages | producer | modification | date |
|---|---|---|---|---|---|---|---|
| 1 | The Code Wiki | 2 | *NULL* | 79 | Microsoft® Office Word 2007 | *NULL* | 2009-08-02 |

Рисунок 3.10 – Отображение информации о статье в базе данных.


Рисунок 3.11 – Отображение информации о статье на сервисе.

### 3.4.3 Поиск тематики статьи

Поиск тематики статьи, будет осуществляться с использованием индексации. Каждое слово будет проиндексировано и найдено количество вхождений каждого слова, учитывая разные формы представления. После исключения стоп-слов, самые часто используемые слова будут записаны в базу данных в поле тематики статьи.

| id | name | Theme | author | coauthors | pages | producer | modification | date |
|---|---|---|---|---|---|---|---|---|
| 1 | The Code Wiki | program, object, orient, unit, test | 2 | *NULL* | 79 | Microsoft® Office Word 2007 | *NULL* | 2009-08-02 |

Рисунок 3.12 – Отображение информации о статье в базе данных.

Таким образом, можно сделать следующие выводы:

1) Реализованный алгоритм осуществляет анализ данных файлов научных статей в формате PDF.
2) Алгоритм автоматически добавляет информацию о статье в базу данных.
3) Проект реализован используя Yii 2 framework.
4) Данные о статьях и авторах выведены на web-сайт, для удобства администрирования.

# Заключение

В результате выпускной квалификационной работы было создано приложение, реализующее алгоритм анализа научных статей. Также реализована система записи полученных данных в таблицы базы данных. Созданное приложение реализует анализ файлов научных статей в формате PDF. Выделяя из полученного файла информацию о тематике статьи, авторах, ключевых словах, программе создания, дате создания, количестве страниц. Приложение реализовано с использованием адаптивной верстки, следовательно, будет корректно отображаться на любом экране.

Предполагаемыми улучшениями полученного приложения являются:

- улучшение алгоритма поиска информации в файле;
- реализация автоматического индексирования статей, находящихся в общем доступе;
- создание личного кабинета для авторов для возможности подтверждения авторства.

## Список используемой литературы

1. Интернет как инструмент библиографического поиска / И. С. Галеева. - Санкт-Петербург : Профессия, 2007. - 247 с. : ил. - (Библиотека). - Библиогр.: с. 182-201. - Прил.: с. 202-246.

2. Потопахин В. В. Искусство алгоритмизации: учеб. пособие / В. В. Васильев, Н. В. Сороколетова. – Гриф УМО. – М.: ФОРУМ, 2009. – 335 с.

3. С. В. Мельников. Perl для профессиональных программистов. Регулярные выражения : учеб. пособие / С. В. Мельников. - Москва : Интернет-Ун-т Информ. Технологий : БИНОМ. Лаб. знаний, 2007. - 190 с.

4. Schwartz, B. Hight Perfomance MySQL: Optimization, Backups and Replication / B. Schwartz, P. Zaitsev. – O'Reilly Media, 2012. – 826 p.

5. Manning, C.D. Introduction to Information Retrieval / C.D. Manning. – Cambridge University Press, 2008. – 506 p.

6. DuBois, P. MySQL – Developer's Library / P. DuBois. – Addison-Wesley Professional, 2013. – 1176 p.

7. Baeza-Yates, R. Modern Information Retrieval: The Concepts and Technology behind Search / R. Baeza-Yates, B. Ribeiro-Neto. – Addison-Wesley Professional, 2012. – 944 p.

8. Wu S. Data Fusion in Information Retrieval / Shengli Wu. – Springer Berlin Heidelberg, 2012. – 212 p.

9. Yekhanin, S. Locally Decodable Codes and Private Information Retrieval Schemes / Sergey Yekhanin. – Springer Berlin Heidelberg, 2010. – 82 p.

10. Kowalski, G. Information Retrieval Architecture and Algorithms / Gerald Kowalski. – Springer US, 2011. – 280 p.

11. Melucci, M. Advanced Topics in Information Retrieval / Massimo Melucci, Ricardo Baeza-Yates. – Springer Berlin Heidelberg. 2011. – 274 p.

12. Melucci, M. Introduction to Information Retrieval and Quantum Mechanics / Massimo Melucci. – Springer Berlin Heidelberg, 2015. – 232 p.

13. Kowalski, G.J. Information Storage and Retrieval Systems / G.J. Kowalski,

M.T. Maybury. – Springer US, 2002. – 316 p.

14. Metzler, D. A Feature-Centric View of Information Retrieval / Donald Metzler. – Springer Berlin Heidelberg, 2011. – 166 p.

15. Zhang, J. Visualization for Information Retrieval / Jin Zhang. – Springer Berlin Heidelberg, 2008. – 293 p.

16. Колисниченко, Д.Н. PHP и MySQL. Разработка веб-приложений / Д.Н. Колисниченко. – СПб.: Изд-во «БХВ-Петербург», 2015. – 592 с.

17. Kroenke, David M. Database Processing: Fundamentals, Design, and Implementation / David M. Kroenke, David J. Auer. – 14th edition. – Pearson Education Limited, 2015. – 640 p.

18. Kromann, Frank M. PHP and MySQL Recipes: A Problem-Solution Approach / Frank M. Kromann. – 2nd edition. – Appress, 2015. – 700 p.

19. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript / Р. Никсон; пер. с анг. Н. Вильчинский. – 2-е изд. – СПб.: Изд-во «Питер», 2013. – 560 с.

20. Прохоренок, Н.А. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А. Прохоренок, В.А. Дронов. – 4-е изд. – СПб.: Изд-во «БХВ-Петербург», 2015. – 766 с.

# Приложение А
# Получение данных из файла

```
//Декодирование
function decodeAsciiHex($sinput) {
   $output = "";
   $isOdd = true;
   $isComment = false;
   for($i = 0, $codeHigh = -1; $i < strlen($input) && $input[$i] != '>'; $i++) {
      $c = $input[$i];
      if($isComment) {
         if ($c == '\r' || $c == '\n')
            $isComment = false;
         continue;
      }
      switch($c) {
         case '\0': case '\t': case '\r': case '\f': case '\n': case ' ': break;
         case '%':
            $isComment = true;
            break;
         default:
            $code = hexdec($c);
            if($code === 0 && $c != '0')
               return "";
            if($isOdd)
               $codeHigh = $code;
            else
               $output .= chr($codeHigh * 16 + $code);
            $isOdd = !$isOdd;
            break;
      }
   }
   if($input[$i] != '>')
      return "";
   if($isOdd)
      $output .= chr($codeHigh * 16);
   return $output;
}
function decodeAscii85($input) {
   $output = "";
   $isComment = false;
   $ords = array();
```

```
    for($i = 0, $state = 0; $i < strlen($input) && $input[$i] != '~'; $i++) {
        $c = $input[$i];
        if($isComment) {
            if ($c == '\r' || $c == '\n')
                $isComment = false;
            continue;
        }
        if ($c == '\0' || $c == '\t' || $c == '\r' || $c == '\f' || $c == '\n' || $c == ' ')
            continue;
        if ($c == '%') {
            $isComment = true;
            continue;
        }
        if ($c == 'z' && $state === 0) {
            $output .= str_repeat(chr(0), 4);
            continue;
        }
        if ($c < '!' || $c > 'u')
            return "";
        $code = ord($input[$i]) & 0xff;
        $ords[$state++] = $code - ord('!');
        if ($state == 5) {
            $state = 0;
            for ($sum = 0, $j = 0; $j < 5; $j++)
                $sum = $sum * 85 + $ords[$j];
            for ($j = 3; $j >= 0; $j--)
                $output .= chr($sum >> ($j * 8));
        }
    }
    if ($state === 1)
        return "";
    elseif ($state > 1) {
        for ($i = 0, $sum = 0; $i < $state; $i++)
            $sum += ($ords[$i] + ($i == $state - 1)) * pow(85, 4 - $i);
        for ($i = 0; $i < $state - 1; $i++)
            $output .= chr($sum >> ((3 - $i) * 8));
    }
    return $output;
}
//Декодирование
function decodeFlate($input) {
    return @gzuncompress($input);}
```

```
//Получение параметров объекта
function getObjectOptions($object) {
   $options = array();
   if (preg_match("#<<(.*)>>#ismU", $object, $options)) {
      $options = explode("/", $options[1]);
      @array_shift($options);
      $o = array();
      for ($j = 0; $j < @count($options); $j++) {
         $options[$j] = preg_replace("#\s+#", " ", trim($options[$j]));
         if (strpos($options[$j], " ") !== false) {
            $parts = explode(" ", $options[$j], 2);
            $o[$parts[0]] = $parts[1];
         } else
            $o[$options[$j]] = true;
      }
      $options = $o;
      unset($o);
   }
   return $options;
}
//Расшифровывание потока
function getDecodedStream($stream, $options) {
   $data = "";
   if (empty($options["Filter"]))
      $data = $stream;
   else {
      $length = !empty($options["Length"]) && strpos($options["Length"], " ") ===
false ? $options["Length"] : strlen($stream);
      $_stream = substr($stream, 0, $length);.
      foreach ($options as $key => $value) {
         if ($key == "ASCIIHexDecode")
            $_stream = decodeAsciiHex($_stream);
         if ($key == "ASCII85Decode")
            $_stream = decodeAscii85($_stream);
         if ($key == "FlateDecode")
            $_stream = decodeFlate($_stream);
      }
      $data = $_stream;
   }
   return $data;
}
```

```
//Получение грязного текста
function getDirtyTexts(&$texts, $textContainers) {
   for ($j = 0; $j < count($textContainers); $j++) {
      if (preg_match_all("#\[(.*)\]\s*TJ#ismU", $textContainers[$j], $parts))
         $texts = array_merge($texts, @$parts[1]);
      elseif(preg_match_all("#Td\s*(\(.*\))\s*Tj#ismU", $textContainers[$j], $parts))
         $texts = array_merge($texts, @$parts[1]);
   }
}
//Получение трансформаций отдельных символов
function getCharTransformations(&$transformations, $stream) {
   preg_match_all("#([0-9]+)\s+beginbfchar(.*)endbfchar#ismU", $stream, $chars,
PREG_SET_ORDER);
   preg_match_all("#([0-9]+)\s+beginbfrange(.*)endbfrange#ismU", $stream,
$ranges, PREG_SET_ORDER);
         if (preg_match("#<([0-9a-f]{2,4})>\s+<([0-9a-f]{4,512})>#is",
trim($current[$k]), $map))
            $transformations[str_pad($map[1], 4, "0")] = $map[2];
      }
   }
      for ($k = 0; $k < $count && $k < count($current); $k++) {
         if (preg_match("#<([0-9a-f]{4})>\s+<([0-9a-f]{4})>\s+<([0-9a-f]{4})>#is",
trim($current[$k]), $map)) {
            $from = hexdec($map[1]);
            $to = hexdec($map[2]);
            $_from = hexdec($map[3]);
            for ($m = $from, $n = 0; $m <= $to; $m++, $n++)
               $transformations[sprintf("%04X", $m)] = sprintf("%04X", $_from + $n);

         } elseif (preg_match("#<([0-9a-f]{4})>\s+<([0-9a-f]{4})>\s+\[(.*)\]#ismU",
trim($current[$k]), $map)) {
            $from = hexdec($map[1]);
            $to = hexdec($map[2]);
            $parts = preg_split("#\s+#", trim($map[3]));

            for ($m = $from, $n = 0; $m <= $to && $n < count($parts); $m++, $n++)
               $transformations[sprintf("%04X", $m)] = sprintf("%04X",
hexdec($parts[$n]));
         }
      }
   }
}
```

```
//Выявление трансформаций текста
function getTextUsingTransformations($texts, $transformations) {
   $document = "";
   for ($i = 0; $i < count($texts); $i++) {
      $isHex = false;
      $isPlain = false;
      $hex = "";
      $plain = "";
      for ($j = 0; $j < strlen($texts[$i]); $j++) {
         $c = $texts[$i][$j];
         switch($c) {
            case "<":
               $hex = "";
               $isHex = true;
               break;
            case ">":
               $hexs = str_split($hex, 4);
               for ($k = 0; $k < count($hexs); $k++) {
                  $chex = str_pad($hexs[$k], 4, "0");
                  if (isset($transformations[$chex]))
                     $chex = $transformations[$chex];
                  $document .= html_entity_decode("&#x".$chex.";");
               }
               $isHex = false;
               break;
            case "(":
               $plain = "";
               $isPlain = true;
               break;
            case ")":
               $document .= $plain;
               $isPlain = false;
               break;
            case "\\":
               $c2 = $texts[$i][$j + 1];
               if (in_array($c2, array("\\", "(", ")"))) $plain .= $c2;
               elseif ($c2 == "n") $plain .= '\n';
               elseif ($c2 == "r") $plain .= '\r';
               elseif ($c2 == "t") $plain .= '\t';
               elseif ($c2 == "b") $plain .= '\b';
               elseif ($c2 == "f") $plain .= '\f';
               elseif ($c2 >= '0' && $c2 <= '9') {
                  $oct = preg_replace("#[^0-9]#", "", substr($texts[$i], $j + 1, 3));
```

```
                $j += strlen($oct) - 1;
                $plain .= html_entity_decode("&#".octdec($oct).";");
            }
            $j++;
            break;
        default:
            if ($isHex)
                $hex .= $c;
            if ($isPlain)
                $plain .= $c;
            break;
        }
    }
    $document .= "\n";
  }
  return $document;
}
//Выделение текста из файла PDF
function pdf2text($filename) {
  $infile = @file_get_contents($filename, FILE_BINARY);
  if (empty($infile))
    return "";
  $transformations = array();
  $texts = array();
  preg_match_all("#obj(.*)endobj#ismU", $infile, $objects);
  $objects = @$objects[1];
  for ($i = 0; $i < count($objects); $i++) {
    $currentObject = $objects[$i];
    if (preg_match("#stream(.*)endstream#ismU", $currentObject, $stream)) {
      $stream = ltrim($stream[1]);
      $options = getObjectOptions($currentObject);
      if (!(empty($options["Length1"]) && empty($options["Type"]) &&
empty($options["Subtype"])))
        continue;
      $data = getDecodedStream($stream, $options);
      if (strlen($data)) {
        if (preg_match_all("#BT(.*)ET#ismU", $data, $textContainers)) {
          $textContainers = @$textContainers[1];
          getDirtyTexts($texts, $textContainers);
        } else
          getCharTransformations($transformations, $data);
      }
```

```
        }
    }
    return getTextUsingTransformations($texts, $transformations);
}
```