Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

|  |  |
| --- | --- |
|  | Номер зачетной книжки \_\_\_\_\_2510119\_\_\_\_  Преддипломная практика зачтена с оценкой  \_\_\_\_\_\_\_ (\_\_ \_\_\_\_\_\_\_\_\_\_)  (цифрой) (прописью)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись руководителя практики от БГУИР)  \_02\_\_\_.\_02\_\_\_.2017 |

**ОТЧЕТ**

**по преддипломной практике**

Место прохождения практики: ИООО «Эпам системз», г.Минск

Сроки прохождения практики: с 02.02.2017 по 29.03.2017

|  |  |
| --- | --- |
| Руководитель практики от предприятия:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д. В. Харевич  (подпись руководителя)  М.П. | Студент группы 251002  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.Д. Навицкий (подпись студента)  Руководитель практики от БГУИР  Сурков К.А. – ассистент кафедры ПОИТ |

Минск 2017

СОДЕРЖАНИЕ

[**РЕФЕРАТ** 3](#_Toc482141743)

[**ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ, СОКРАЩЕНИЯ** 4](#_Toc482141744)

[**ВВЕДЕНИЕ** 5](#_Toc482141745)

[**1 АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ** 7](#_Toc482141746)

[**1.1.** **Анализ использованных литературных источников** 7](#_Toc482141747)

[**1.2.** **Обоснование выбора языка разработки** 7](#_Toc482141748)

[**1.3.** **Обоснование выбора баз данных** 9](#_Toc482141749)

[**1.4.** **Формирование требований к проектируемому программному средству** 10](#_Toc482141750)

[**2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ** 16](#_Toc482141751)

[**3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА** 21](#_Toc482141752)

[**4.1. libcore - основной программный модуль** 26](#_Toc482141753)

[**6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ** 29](#_Toc482141754)

[**6.2** Расчет затрат на разработку ПО 30](#_Toc482141755)

[**6.3** Оценка результата (эффекта) от использования (или продажи) ПО 34](#_Toc482141756)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 37](#_Toc482141757)

# **РЕФЕРАТ**

Объектом исследования является система управления цифровыми документами на производстве.

Цель работы – разработка программного средства для хранения, поиска, редактирования и обработки файлов и их вспомогательной информации.

Данное программное средство упростит работу с файлами для компаний, у которых есть проблемы с хранением и доступом к файлам, т.к. это программное средство предоставит единое хранилище файлов с убодным пользовательским интерфейсом. Это приложение позволит так же ассоциировать с файлами дополнительную информацию, которая необходима в процессе обработки файлов. Пользователь сможет классифицировать классы по категориям, добавлять новые атрибуты, осуществлять поиск не только по имени файла, но и по заданным атрибутам и категориям, что значительно ускорит поиск необходимых файлов. Единое хранилище позволит не только проще обмениваться файлами между людьми, но и обеспечить безопасность файлов и версионность.

Проведен анализ похожих продуктов: их производительность, степень нагрузки на систему, простоту в использовании и подготовки серверов для разворачивания приложения.

Результатом дипломного проекта является веб приложение, позволяющее управлять файлами: хранением, поиском, модификацией и удалением.

Областью применения являются компании, которые оперируют с большим колличеством файлов, и когда помимо самого файла необходимо хранить информацию, описывающую его.

# **ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ, СОКРАЩЕНИЯ**

*Программное обеспечение* (software) - программы, процедуры, прави-ла и любая соответствующая документация, относящиеся к работе вычисли-тельной системы.

*Процесс обработки данных* - система действий, реализующая определен-ную функцию в системе обработки информации и оформленная так, что управ-ляющая программа данной системы может перераспределять ресурсы этой си-стемы в целях обеспечения мультипрограммирования.

*Подпрограмма* - программа, являющаяся частью другой программы и удовлетворяющая требованиям языка программирования к структуре програм-мы.

*Программный модуль* - программа или функционально завершенный фрагмент программы, предназначенный для хранения, трансляции, объедине-ния с другими программными модулями и загрузки в оперативную память.

*Программирование* - научная и практическая деятельность по созданию программ.

*Программа* - данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определен-ного алгоритма.

*Программное обеспечение* - совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ.

*DAM* (Digital Asset Management) – Система управления цифровыми документами.

*Ассет* – экономический ресурс, представляющий собой электронный документ, изображение, аудио- и видеофайл, права на который, принадлежат организации, производившей данный продукт.

*Метаданные* – вспомогательная информация конкретного ассета.

*Поля* – атрибуты файла, составная часть метаданных, которые известны только системе и хранят дополнительную информацию о файле.

*Классификации* – название группы файлов, объединенных по определенным бизнесс правилам в одну группу и имеющую одну цель.

# **ВВЕДЕНИЕ**

Быстрое наращивание объема цифровых данных, хранимых в корпоративной сети компании, часто создает трудности в поиске необходимых данных. Согласно оценке экспертов, средний пользователь тратит десятую часть своего рабочего времени на работу с файлами. При этом возможности реляционных баз данных общего назначения ограничены. Пользователю часто требуется не только просто сохранить, найти или извлечь необходимую информацию. Кроме этого, нужно сохранять метаданные к изображениям, аудио- и видеофайлам, отслеживать их версии и права доступа к ним, конвертировать их в различные форматы. Для выполнения всех этих операций предназначены системы управления цифровыми архивами (digital asset management, DAM). Так как DAM хранит не только файлы, но и метаданные, относящиеся к каждому ассету в отдельности. Метаданные можно использовать как инструкции при обработке ассетов внешними сервисами, так и как информацию, необходимую для конечных пользователей, в зависимости от бизнесс требований. Например, ассеты попадающие под специальную категорию могут учавствовать в формировании отчета. Таким образом можно формировать разные отчеты, взависимости от категории ассета. К категориям можно привязать роли пользователей, т.о. ограничив фозможности отдельных групп пользователей.

Для примера, американская футбольная лига (National Football League, NFL) также использует на своих сайтах возможности DAM-системы, в частности, для ограничения доступа к контенту. Компания WebWare сопровождает несколько сайтов NFL. Один из сайтов предназначен для зарегистрированных спортивных журналистов. Зайдя на него, можно скачать, например, фотографии игроков и тренеров. На сайте для рядовых пользователей можно получить фрагменты игр с низким разрешением. Для получения по почте изображения с высоким разрешением в виде фотографии пользователь должен послать соответствующий заказ. В то же время существует B2B-система, в рамках которой коммерческие партнеры NFL могут скачать те же изображения с высоким разрешением.

Некоторые DAM системы поддерживают интеграцию с AgilePoint workflow. Это позволяет описать процесс, который должен пройти ассет, в схеме Visio, которая сохраняется в системе и используется для того, что бы назначать определенные задачи. За выполнение этих задач могут быть ответственны разные сотрудники или группы сотрудников, в зависимости от их обязанностей. Например, если организация занимается производством обуви, то ей нужно проводить маркетинговую компинию для привлечения покупателей, которая включает в себя производство плакатов с фирменной продукцией. Фотографии продукции должны соответствовать определенным критериям. Процесс преобразования фотографии в изображение на плакате описывается в схеме. После чего разным группам людей выдаются задания, например проверить качество фотографии. Если качество не устраивает, фотографию бракуют и процесс начинается заного. Если фотография сделана успешно, из этой фотографии нужно вырезать часть, которая будет на плакате, для этого дезайнерам создается задача - выбрать подходящий размер изображения, которое будет вырезана из фотографии. Таким образом, на каждой стадии процесса изображение преобразуется к финальному виду, дополняется необходимыми документами, и поставляется набором на фабрику для печати.

Целью данного дипломного проекта является написание программного средства управления цифровыми документами на предприятии. В разделе «Формирование требований к программному средству» представлены методы определения бизнесс требований заказчика, которые будут использованы при проектировании системы. Раздел «Разработка программного средства» содержит подробное описание этапов проектирования и реализации системы. Обоснование выбора языка программирования, операционной среды находится в разделе «Обоснование технических приемов программирования». Так же пояснительная записка содержит результаты тестирования системы, руководство пользователя, технико-экономические обоснования принятых решений, раздел охраны труда и экологической безопастности.

# **1 АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ**

## **Анализ использованных литературных источников**

1. Digital Asset Management Best Practice guide

Данная книга позволяет систематизировать знания о том, что такое система управления цифровымифайлами и для каких целей она используется. Книга включается в себя описание основных требований к системе и рассказывает, что нужно учитывать при проектировке системы.

Первые главы дают баовое представление о DAM системе и в каких случаях она применятся. В них рассказываются как нужно анализировать бизнесс требования заказчика для формирования понятия о том, какая система ему нужна.

Далее предоставляется описание жизненного цикла файла в системе: через какие стадии проходит, какие функции доступны пользователю, при работе с файлами, входные и выходные данные системы.

В разделе «Catalogs» рассказывается как правильно хранить файлы в системе для их оптимального хранения и удобного использования.

Рассказывается про метаданные файла которые должны храниться в ассете и для чего они нужны. Затрагивается тема классифицирования ассетов и их доставка пользователям. Создание резервных копий.

1. <http://digitalassetmanagement.com/dam-basics/>

По этой ссылке можно прочитать статью, которая дает базовое представление о DAM сисемах. К томуже содержит видео ролики в которых можно увидеть эти системы в действии.

1. <http://digitalassetmanagement.com/advanced-dam/>

Данная статья дает более детальное представление о DAM системе. Эта информация полезна для тех, чьей обязанностью будет поддержка системы или ее создание.

1. <http://www.cmswire.com/cms/digital-asset-management/5-good-reasons-to-avoid-dam-software-019130.php>

В данной статье освещаются недостатки DAM систем.

## **Обоснование выбора языка разработки**

Для написания данного дипломного проекта использовался язык программирования Rust. Это системный, строго типизированный, компилируемый язык программирования который предотвращает большое количество ошибок за счет своей системы типов и умного компилятора, который на этапе компиляции может выявить большое количество ошибок, которые в других языках проявляются только во время запуска программы.

Данный язык программирования является системным, это значит, что при разработке приложений с помощью этого языка, программист полностью контролирует работупрограммы. Языки программирования, такие как С# или Java используют сборщик мусора для управления памятью, следовательно у программиста практически нет контроля над тем, когда и как освобождается память. Эта неопределенность может плохо повлиять на работу некоторых систем (например, в медицинском оборудовании от этой неопределенности не должно быть, т.к. от этого оборудования зависит человеческая жизнь). Эта проблема актуальна и для высоконагруженных систем, где память должна освобождаться сразу же после того, как необходимость в ней пропадает. Это позволяет уменьшить нагрузку на систему и повысить ее производительность. В Rust нет сборщика мусора так, что программист получает полный контроль над памятью, но так же нет и ручного удаления памяти, как в С/C++. Это позволяет избежать ошибок при работе с памятью, свойственных таким языкам, как С/C++.

Rust компилируется сразу в машинный код, в отличии от тех же C# и Java, что позволяет лучше оптимизировать код для конкретной платформы (Rust поддерживает множество платформ: Windows, Linux, MacOS и д.р.). Так как код на Rust преобразуется в байт-код для LLVM, то производительность программ сравнима с производительностью кода на C/C++.

Rust имеет очень сильную систему типов и вместе с правилами владения и заимствования, соблюдение которых требуется компилятором, написание многопоточного кода перестает быть проблемой. В программах на Rust не бывает гонки данных. Это позволяет писать многопоточный код без страха, в отличии от других языков программирования в частности C/C++, в которых гонка данных может привести к проблемам в безопасности приложения.

Rust поддерживает C ABI, что позволяет вызывать код написанный на Rust из других языков (С#, Java и даже C/C++). Таким образом можно писать высокопроизводительный код на Rust, который может заменить низкопроизводительный код другого языка. Для примера, в Ruby есть метод blank? который проверяет, является ли строка пустой или нет. Эта проверка выполняется с помощью регулярного выражения, что достаточно медленно. Из-за этого, во многих Ruby on Rails приложениях страдала производительность. Когда данный метод был переписан на Rust, производительность увеличилась в 11 раз (причем размер кода на Rust составил всего одну строку, в отличии от версии Си, где потребовалось 50 строк кода).

На Rust написано множество open source библиотек, часть которых написано самими разработчиками Rust-а. В данном дипломном проекте были использованны:

1. Diesel
2. Rs-es
3. Serde
4. Rocket

Diesel – модуль для коммуникации с реляционными базами данных. Поддерживаются: PostgreSQL, MySQL, SQLite. Данный модуль предоставляет возможность трансформирования кода на Rust в SQL запросы. Diesel использует систему типов Rust что бы проверять правильность запросов на этапе компиляции, а нулевая стоймость структур в Rust позволяет оптимизировать запросы к базе так же и на стороне клиента (кода). Это позволяет выполнять запросы даже быстрее чем это делает С. Так же в комплект с данным модулем входит консольная утилита для создания и применения миграций базы данных.

Rs-es – модуль для коммуникации с elasticsearch базой данных. Данный модуль поддерживает все основные операции предоставляемые elasticsearch и имеет удобный API для формирования собственных запросов.

Serde – модуль сериализации и десериализации данных в Rust. Так как elasticsearch хранит документы в формате Json, данный модуль является очень полезным, т.к. позволяет без написания большого количества кода сделать структуры данных Rust сериализуемыми\десериализуемыми.

Rocket – веб фреимворк для Rust. Простой в использовании, безопасный и расширяемый веб фреимворк.

## **Обоснование выбора баз данных**

В данном дипломном проекте использовалось две базы данных: PostgreSQL и elasticsearch.

PostgreSQL — свободная объектно-реляционная система управления базами данных (СУБД).

Существует в реализациях для множества UNIX-подобных платформ, включая AIX, различные BSD-системы, HP-UX, IRIX, Linux, macOS, Solaris/OpenSolaris, Tru64, QNX, а также для Microsoft Windows.

PostgreSQL базируется на языке SQL и поддерживает многие из возможностей стандарта SQL:2011.

Сильными сторонами PostgreSQL считаются:

1. высокопроизводительные и надёжные механизмы транзакций и репликации;
2. расширяемая система встроенных языков программирования: в стандартной поставке поддерживаются PL/pgSQL, PL/Perl, PL/Python и PL/Tcl; дополнительно можно использовать PL/Java, PL/PHP, PL/Py, PL/R, PL/Ruby, PL/Scheme, PL/sh и PL/V8, а также имеется поддержка загрузки C-совместимых модулей;
3. наследование;
4. легкая расширяемость.

Так как PostgreSQL – реляционная база данных, она используется для хранения структуры системы, ее зависимостей и вспомогательной информации. Это позволяет перенести часть логики работы с данными на сторону базы и тем самым увеличить производительность системы в целом.

Elasticsearch – поисковой движек разработанный на базе Lucene. Он предоставляет распределенный полнотекстовый поиск с HTTP веб интерфейсом. Elasticsearch написан на языке Java и является опен сорс продуктом под лицензией Apache. Официальные клиенты написаны для множество языков, таких как Java, C#, Python и т.д. Elasticsearch одна из самых популярных поисковых движков, так же как Apache Solr, который тоже разработан на базе Lucene.

В elasticsearch базе храняться документы (вся самая необходимая информация), которые не поддерживают схемы, но поиск в этой базе значительно быстрее чем в реляционных. Ассеты будут загружаться быстро как раз потому, что основная часть данных будет храниться в elasticsearch, а остальные данные, хранящиеся в PostgreSQL будут подгружаться по мере необходимости.

## **Формирование требований к проектируемому программному средству**

Управление цифровыми документами (DAM – Digital Asset Management) на предприятии представляет собой обработку и распространение цифровых документов, таких как изображения, документы, аудио и видио файлы. Системы управления DAM позваляют организовать централизованное хранение, получение и распространение больших объемов данных. Системы управления цифровыми документами предоставляют множество полезных возможностей:

1. Единое, защищенное хранилище данных организации
2. Улучшение cсовместной работы над одними данными
3. Динамическая рассылка данных (документов) внутри комманд и внешним пользователям
4. Быстрый и удобный поиск всех необходимых документов
5. Улучшенный процесс работы с цифровыми документами по отношению к цене и времени
6. Уменшение затрат на организацию работы с документами
7. Быстрая интеграция новых служб в текущий процесс

Для того что бы правильно спроектировать такую систему, нужно определить, на каких людей повлияет внедрение этой технологии. Нужно определить, какие проблемы чаще всего появляются в процессе работы с данными и может ли эта система решить их. Для этого нужно обсудить с заинтересованными сторонами вопросы, касающиеся:

1. Получения необходимой информации о текущих процессах
2. Определения трудностей, возникающих в текущих процессах
3. Определения бизнесс требования
4. Распростронения информации о том, по каким правилам происходят текущие процессы

Есть два способа достичь слаженной работы: «сверху вниз» и «снизу вверх». В случае «сверху вниз», управление процессами будет принадлежать главным менеджерам. При «снизу вверх» подходе, то как будет происходить процесс определяется финальными пользователями, тем, кому нужно решение из проблем. Каждый подход имеет свои «за» и «против» и нужно знать, какие преимущества получат люди и есть ли бизнес причины реализовывать данную систему. Так же, необходимо узнать, как новая система повлеяет на все уровни организации.

При введении DAM системы часто сталкиваются с проблемой, что люди не понимают как с этой системой работать и что делать, если что-то работает не правильно. Для успешной реализации DAM системы необходимо начать с:

1. Изучения бизнесс процессов
2. Изучения структуры организации: как данные (документы) распространяются? Кто задействован в этих процессах? Как распределяется ответственность за процессы?
3. Какие программные средства используются и какое оборудование установлено
4. Какие проблемы возникают

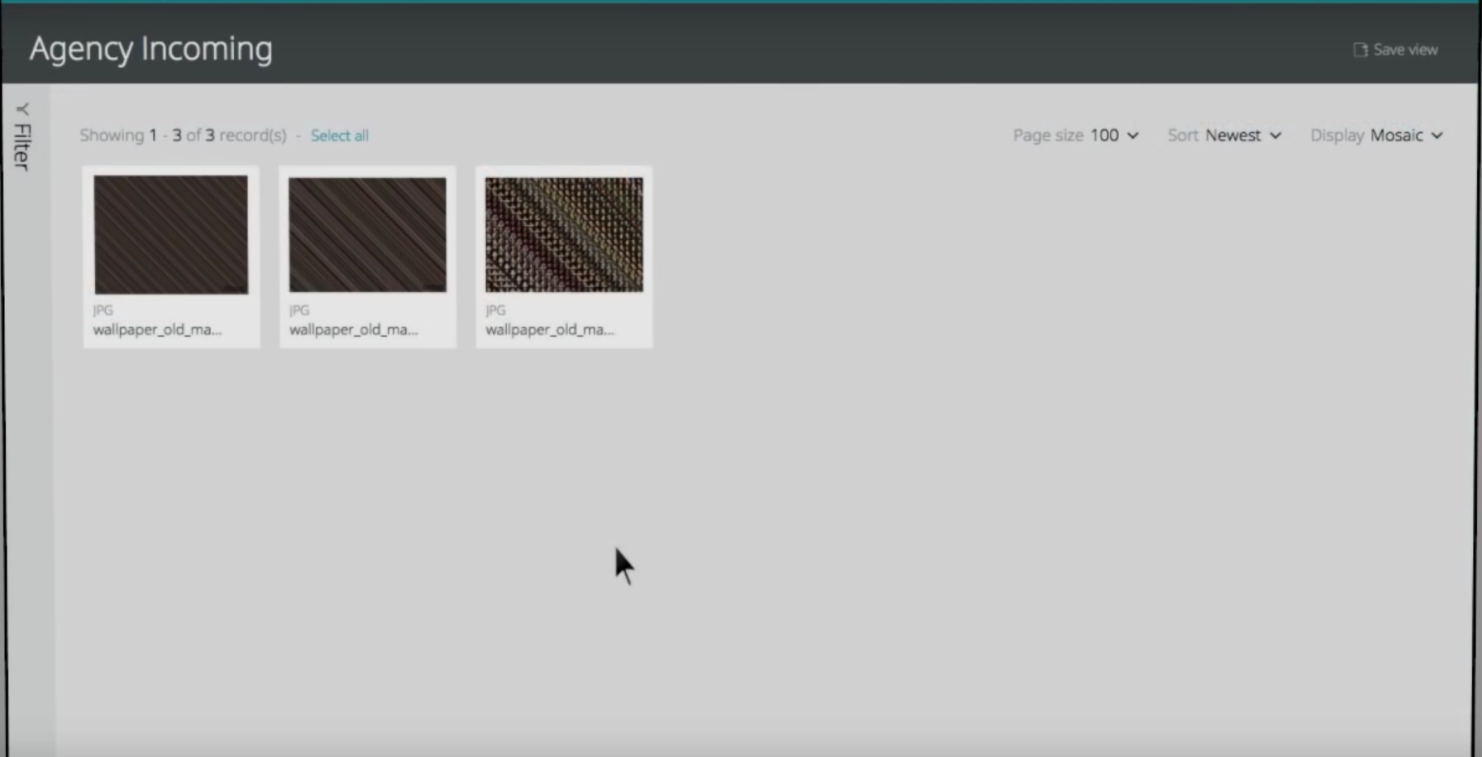
Изучение текущих процессов может помочь в понимании того, что DAM система позволит упростить или автоматизировать.

Когда исследование процессов было произведено и решение о внедрении DAM системы было одобрено, нужно начать собирать информацию о процессах и задачах, выполняемых каждыми отдельными лицами. Система должна быть достаточно гибкой и должна позволять настроить систему под конкретные бизнесс задачи.

Стандартный жизненный цикл ассета включает в себя следующие этапы:

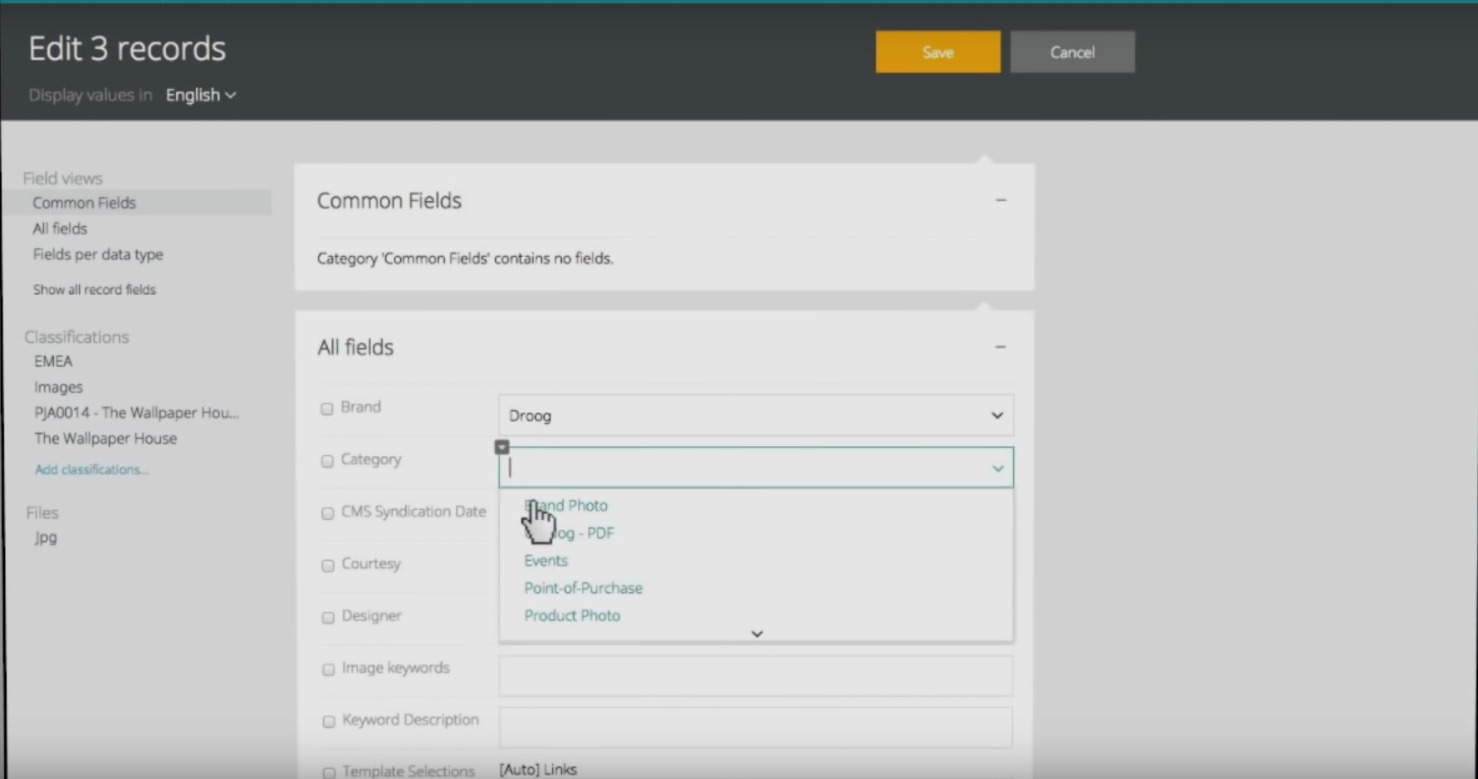
1. Получить файлы с внешнего ресурса
2. Разместить файлы во внутренней структуре каталогов
3. Добавить информацию к файлу (Добавить метаданные)
4. Переименовать, редактировать или изменить оригинал
5. Расположить файлы в соответствии с категориями
6. Предоставить доступ к файлам пользователям
7. Отправлять файлы пользователям или на внешние носители

В качестве примера такой системы можно взять ADAM Software проект. Эта система предоставляет центролизованное хранилище данных с поддержкой версионности, безопасность, удобство при работе с большим количеством файлов (ассетов), удобный поиск. Уменьшает количество ручной работы с ассетами, выявляет дубликаты. Обеспечивает удобную интеграцию с внешними сервисами, партнерами. Предоставляет «Role-based» систему безопастности для доступа к контенту. Гарантирует то, что только проверенные материаллы публикуются и доставляются трейтим лицам.



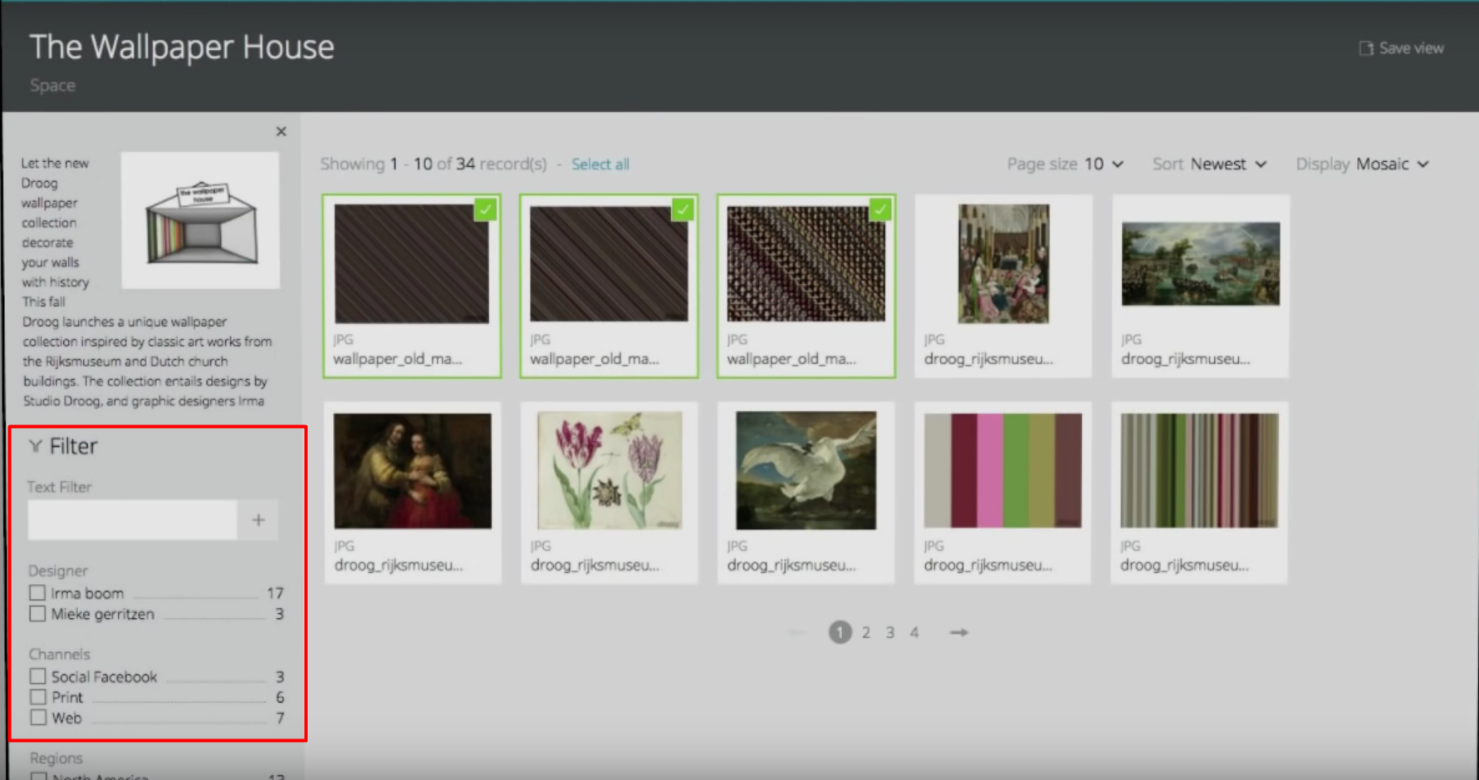
**Рис 1.1 Отображение списка ассетов в системе**

На рисунке 1.1 отображено, в каком виде пользователь видит ассеты в системе. Если нажать левой кнопкой мыши на картинку, появиться окно с превью изображения, которое храниться в ассете. Если нажать на название ассета, то осуществиться переход к подробной информации по данному ассету. Ассеты можно искать используя фильтры или использовать поисковую строку для ввода критериев поиска. Критерии поиска можно комбинировать, составляя очень сложные выражения, которые потом используются для поиска нужных ассетов. Поиск может производиться как по названию файла, классификациям, значеням полей (если поле позволяет). Т.к. классификации представляют собой иерархическую структуру, то можно осуществлять поиск ассетов, которые принадлежат классификациям-потомкам выбранной классификации.

****

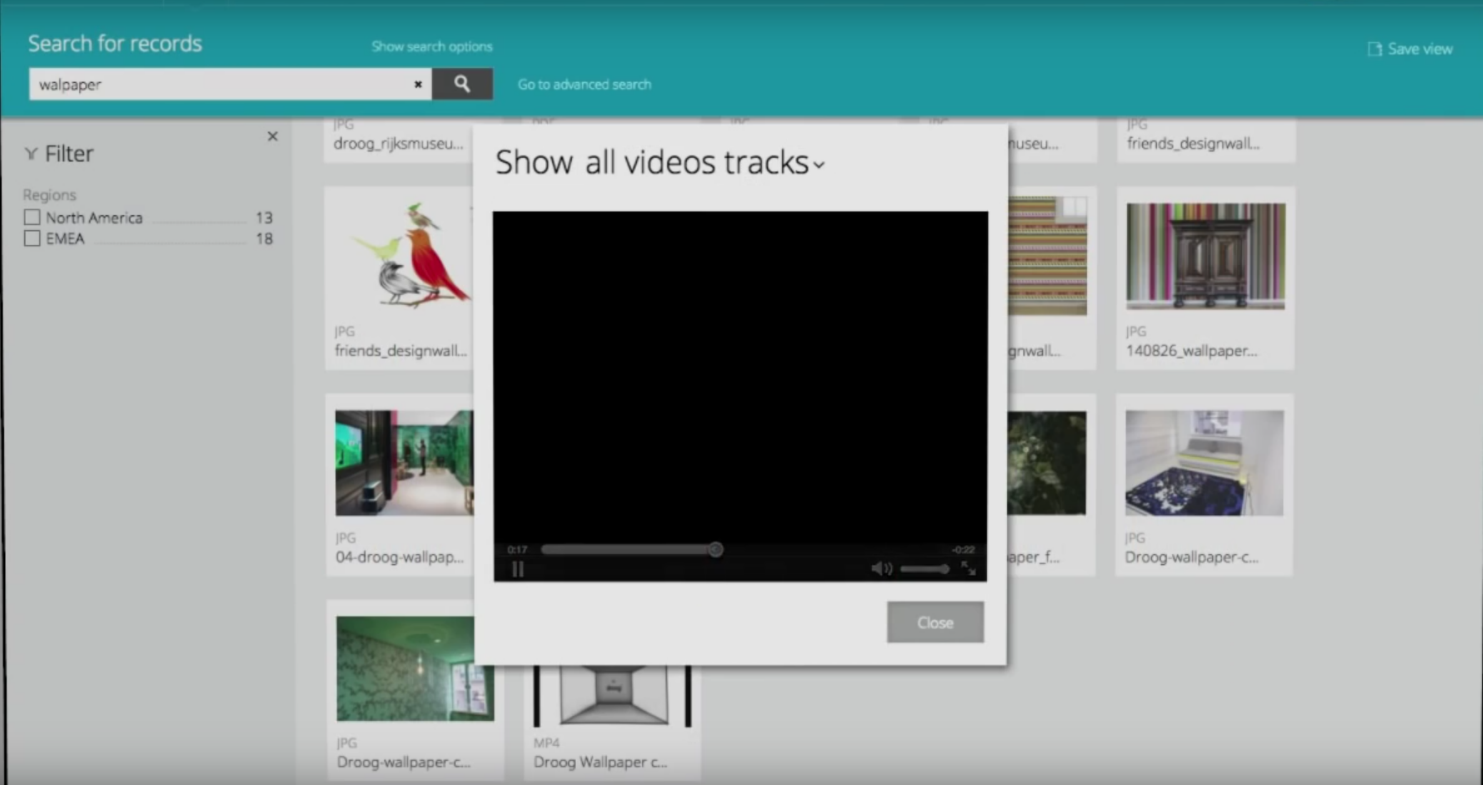
**Рис 1.2 Редактирование ассетов**

На рисунке 1.2. можно увидеть, что есть возможность редактировать сразу несколько ассетов. Так же на скриншоте видно, как выглятят поля ассета и как устанавливаются значения этих полей.



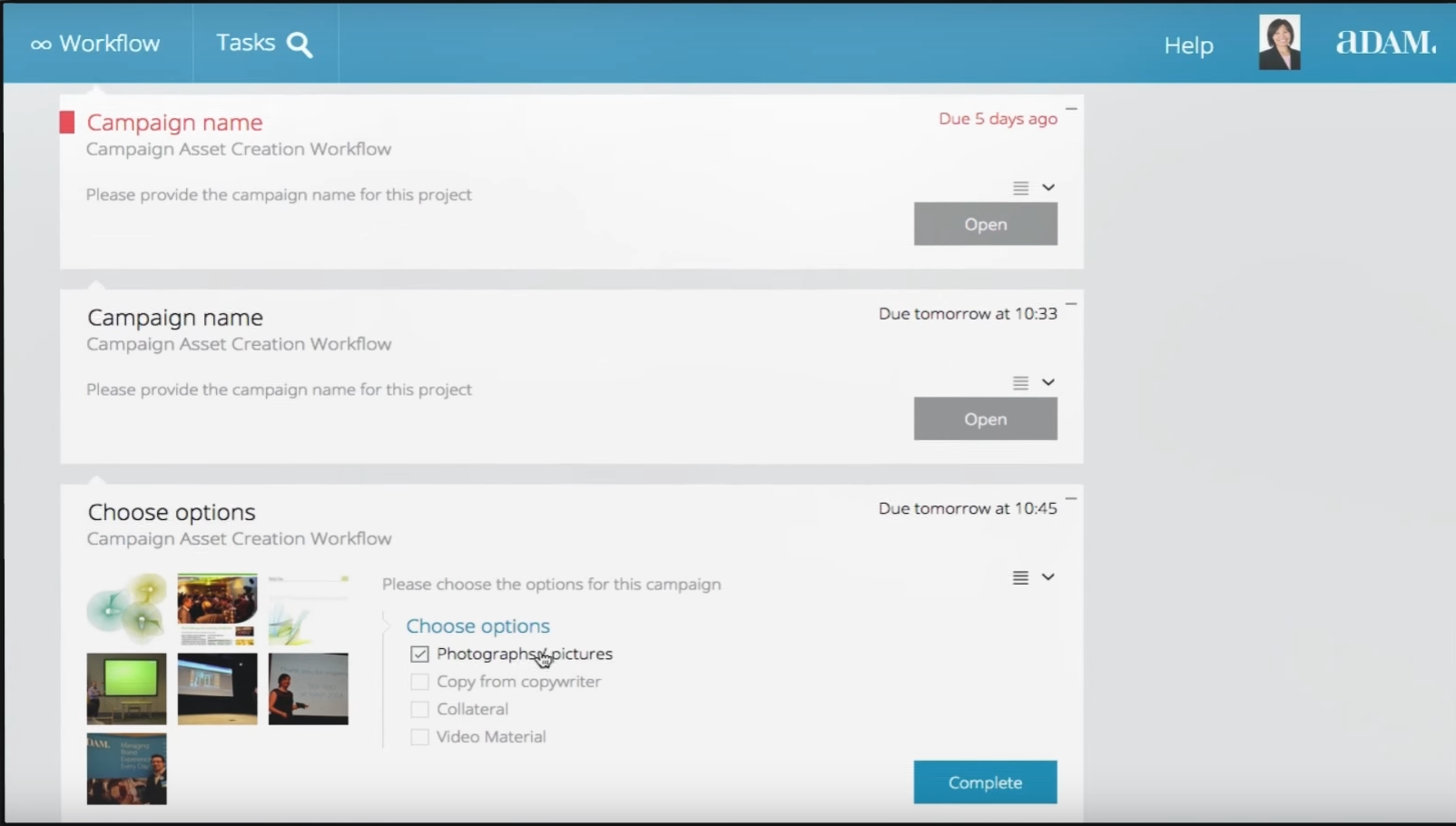
**Рис 1.3. Фильтры**

На рисунке 1.3. можно увидеть как выглядят фильтры. Фильтры представляют собой название полей и список их значений. Напротив каждого значения указывается количество ассетов, у которых данное поле имеет выбранное значение. Ассеты можно выделять, для того что бы произвести операции сразу над несколькими ассетами.

****

**Рис 1.4 Видео проигрыватель**

Пользователи могут загружать не только изображения, но и аудио, видео файлы. Для изображений и PDF докуметнов генерируется превью. В случае видео, отображение проигрыватель для просмотра видео.



**Рис 1.5 Процессы**

Пользователи могут создавать процессы, через которые должен пройти ассет для того что бы его можно было использовать в рекламе и распространении другим компаниям. Каждый процесс представляет собой набор заданий, которые должна выполнить определенная комманда (комманда бренд менеджеров, например). На рисунке 1.5 можно увидеть как выглядят задачи и как их выполнять.

Система, реализованная в рамках данного дипломного проекта, будет иметь возможность:

1. Распределять файлы по категориям (у одного файла может быть несколько категорий).
2. Поддерживать поиск ассетов по имени, категориям и значениям полей метаданных.
3. Отображение файлов будет зависить от ролей, которые есть у пользователя
4. Для доступа к файлам будет использоваться веб-страница
5. Метаданные будут заполняться в процессе создания записи (пользователем или внешней системой), содержащей нужный файл.

# **2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ**

Данный дипломный проект посвящен решению проблемы предприятий по организации работы с электронными документами большому числу людей. Программное средство должно позволять создание, просмотр, редактирование и удаление ассетов, которые будут содержать документы, необходимые сотрудникам для выполнения их задач. Программное средство представляет собой веб приложение, к которому можно обращаться без установки дополнительного программного обеспечения. Это позволит облегчить внедрение средства в процесс.

Во многих организациях часто возникает проблема обмена документами между разными отделами. Фотографы хранят фотографии у себя на компьютере или общей папке, которую видят только его коллеги. Дизайнеры хрянят свои шаблоны, Indesign files в своих папках. В отделе дезайнеров или фотографов, например, возможно организовать общий доступ к фалам и ассоциировать с ними какую либо полезную, для конкретного отдела, информацию (для фотографов резрешение, dpi; для дезайнеров – модель, SKU), но когда требуется поддерживать обмен данными между разными отделами – это может стать проблемой. Файловая система не достаточно гибкий инструмент для того, что бы удовлетворить все бизнесс требования организации. Поиск файлов в файловой системе не может учитывать метаданные файла, а поиск по имени файла не всегда полезен. Так же файловая система не может обеспечить автоматизацию (изменение метаданных файла автоматически, без вмешательства пользователя). Отсутствие возможности автоматического изменения состаяния файлов приводит к большим временным потерям, так как сотрудники должны вручную изменять данные взависимости от наступления каких либо событий, а в отсутствии удобного поиска и децентрализованного места хранения файлов – это становиться почти невозможным для больших компаний как Adidas, например. Даже добавление нового файла в процесс может повлечь за собой большие временные затраты, т.к. сотрудник должен оповестить всех заинтересованных лиц в том, что появляется новый файл, который нужно включить в текущие процессы.

Компинии часто испытывают трудности с большими заказами продукции у вендоров, так как в результате могут получить сотни, если не тысячи (в зависимости от заказа) наименований продукции в виде файлов, изображений, видио или аудиофайлов, которые нужно добавить в процессы и каким либо образом начать с ними работать. Нужно обеспечить доступ к этим файлам, заполнить метаданные. Этот большой объем работы должны выполять сотрудники, что влечет за собой ошибки.

Для решения всех этих проблем вводятся системы управления цифровыми документами. Эти системы помогают уменьшить расходы и временные затраты, уменьшить количество ошибок, так как большая часть процесса автоматизируется. Сотрудники получают удобство работы с файлами и у них появляется больше времени для решения бизнесс задач.

Компания так же может нанять команду разработчиков для того, что бы она поддерживала систему в актуальном состоянии и дополняла ее дополнительными модулями автоматизации по мере необходимости. Таким образом переноситься техническая работа на более квалифицированных сотрудников, что позволяет получить более качественный результат.

Основные функции системы:

1. Загрузка файлов
2. Единый доступ к файлам
3. Категоризация (классификация) ассетов
4. Ролевой доступ к ассетам
5. Метаданные ассета
6. Одинаковый способ работы с файлами для всех сотрудников организации
7. Расширяемость
8. Гибкость
9. Высокая производительность системы

Загрузка файлов с внешнего ресурса будет выполняться в основном пользователями, так как они должны решать какие файлы должны попадать в систему, а какие нет. В случае если необходимо интегрировать внешний источник, файлов, система должна предоставлять возможность интерфейс для разработки средств интеграции. Этот интерфейс представляет собой библиотеку, написанную на том же языке программирования, что и основная система. Эта библиотека будет использоваться для того, что бы программистам не приходилось вручную работать с сложной системой. Данная библиотека будет выполнять большую часть работы самостоятельно.

Структура каталогов с системе, которая поддерживает категоризацию ассетов, может быть достаточно простой. Единственная сложность возникает, когда файлов накапливается много и размера дискового пространства одного сервера может не хватать. В этом случае вводится понятие виртуальных каталогов, т.е. путь файла представляет собой путь внутри виртуальной файловой системы. Когда пользователь хочет скачать файл, виртуальный путь файла преобразуется в физический, по которому и осуществляется поиск файла для скачки. Виртуализация файловой системы целиком зависит от реализации системы и конечные пользователи не должны знать об этом. Файлы должны быть доступны одинаковым способом, независимо как они храняться в системе.

Если пользователю необходимо объеденить ассеты в каталог, в таком случае создается категория (классификация) к которой относятся необходимые ассеты. Это никак не влияет на то, как и где храняться файлы ассетов. Категории позволяют выделить ассеты в отдельную группу, с обственным набором метаданных и ролей пользователей, которым доступны эти ассеты. Категории позволяют обеспечить поддержку разных бизнесс процессов в системе одновременно. К примеру, компания занимается производством одежды и хранит фатографии своей продукции в системе. Различные типы продукции, как майки и рубашки можно разделить на разные категории. Это позволяет пользователям, которым необходимо работать с майками видеть только майки, тем самым упростив поиск нужной модели. Если же компания включает в себя производство продукции различных брендов, каждый бренд может являтся собственной категорией.

Категории могут иметь иерархическую структуру. Такая структура называет таксономией. Она представляет собой информацию о ассетах, включая метаданные.

Поддержка ролей для пользователей очень важна в таких системах, т.к. есть различные бизнесс причины (требования), котрые должны учитываться, когда пользователь пытается запросить доступ к ассету.

Метаданные ассетов похожи на те атрибуты, которые привязывает файловая система к каждому файлу, но метаданные ассета в DAM системе имеют больше возможностей, т.к. позволяют создавать собственные атрибуты исходя из бизнесс требований. Атрибуты являются частью категорий, и когда ассет отностится к какой либо категории, то он автоматически получает список атрибутов этой категории. Если вернуться к примеру с майками, можно представить, что пользователю необходимо хранить цвет майки. В этом случае ассет, нужно отнести к категории «Майка» тогда он получит все атрибуты маек, среди которых, будет атрибут «Цвет». Для этого атрибута можно будет указать цвет майки. И если пользователю будет необходимо получить список всех «зеленых» маек, он может использовать атрибут «Цвет» как критерий поиска. Система не может автоматически заполнить все поля метаданных, поэтому пользователю, при создании ассета нужно будет вручную прописать значения необходимых аттрибутов. В случае создания ассетов отдельным сервисом, метаданные можно получать как отдельный файл XML или Json формата в месте с файлом документа.

После того, как ассет создан и храниться в системе, к нему есть доступ только через веб приложение. Это позволяет предоставить единый вид доступа к, интересующим их ассетам, всем пользователям. Это значительно проще, чем когда файлы храняться в разных местах (на разных ftp серверах, папках файловой сисемы и т.д.).

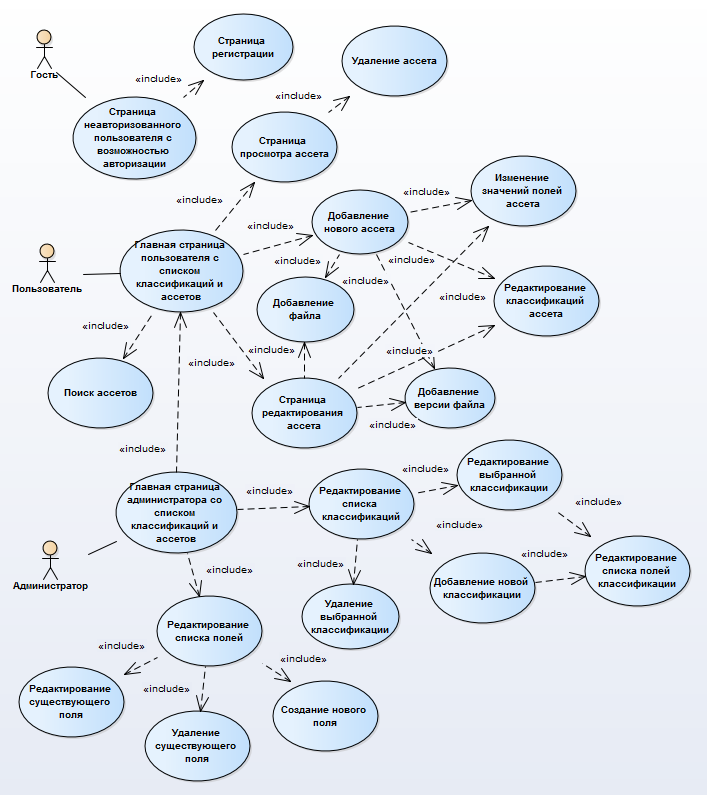
В этом веб приложении так же будет происходить редактирование и поиск существующих ассетов. В зависимости от сложности системы, редактирование может заключаться как в редактировании метаданных ассета, так и редактирование самого файла. Система, поддерживающая версионность файлов ассета позволяет обеспечить пользователей последними версиями файла, что позволяет избежать случая, когда один пользователь изменил файл, а другой не узнал об этих изменениях и использует старую версию файла.

Можно разрабатывать различные приложения (веб-приложения, сервисы, приложения командной строки), которые будут разделять одно «ядро» тем самым подстраивая систему под конкретные бизнесс требования. Данное «ядро» включается в себя базу данных, участок дискового пространства под файлы, интерфейс (библиотеку) для работы с базой и файловой системой.

Программное средство должно быть гибким для того, что бы можно было как можно точнее настроить под требования заказчика.

Огранизация, которая преобрела данное программное средство, может нанять людей для того, что бы поддерживать проложение в актуальном состоянии в течении всего срока эксплуатации. Это необходимо тем организациям, у которых постоянно изменяются требования. Таким образом, люди из поддержки могут создавать классификации, поля, пользовательские роли и т.д. для удовлетворения новых требований.

В системах DAM производительность играет большую роль. Поиск ассетов и загрузка его метаданных должен быть очень быстрый, для того что бы пользователь получал все необходимое с минимальной задержкой. Это влияет на скорость работы сотрудников компании и на скорость обработки ассетов другими модулями программы. Если, например, один из модулей генерирует отчет по новым ассетам, то загрузка ассетов этим модулем должна быть максимально возможной, так как этот модуль сам может произвдить ресурсоемкие операции и в итоге это скажется на общей производительности модуля.



**Рис 2.1. Use-case диаграмма проекта**

# **3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

Программное средство разработанное в рамках этого дипломного проекта будет состоять из нескольких основных модулей:

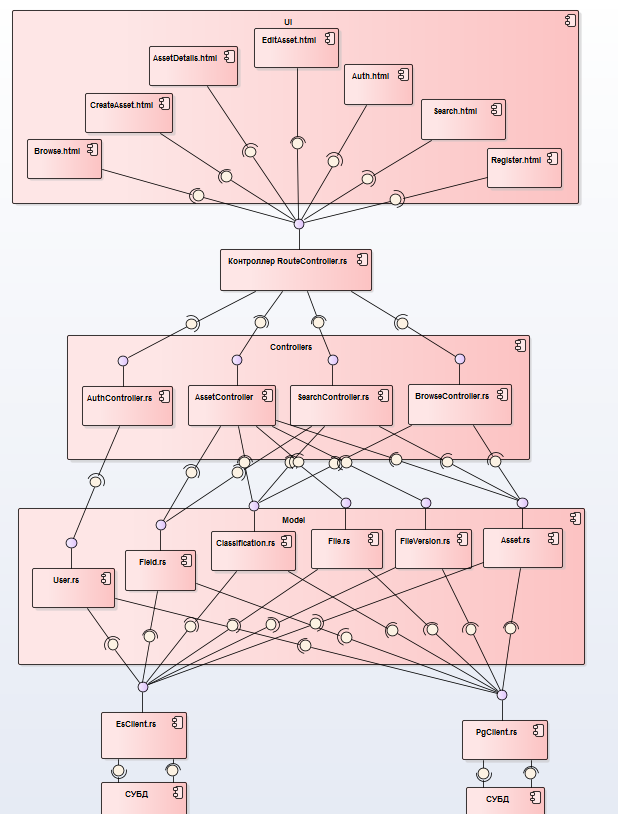
1. Веб приложение
2. Библиотека
3. Модуль работы с PostgreSQL базой данных
4. Модуль работы с elasticsearch базой данных

Веб приложение будет работать как графический интерфейс для работы с системой. Вся основная логика будет вынесена в отдельную библиотеку. Приложение предоставит возможность авторизации, поиска, создания, редакирования, удаления ассетов. Администраторы будут иметь возможность создавать новые классификации, поля, редактировать уже существующие объекты.

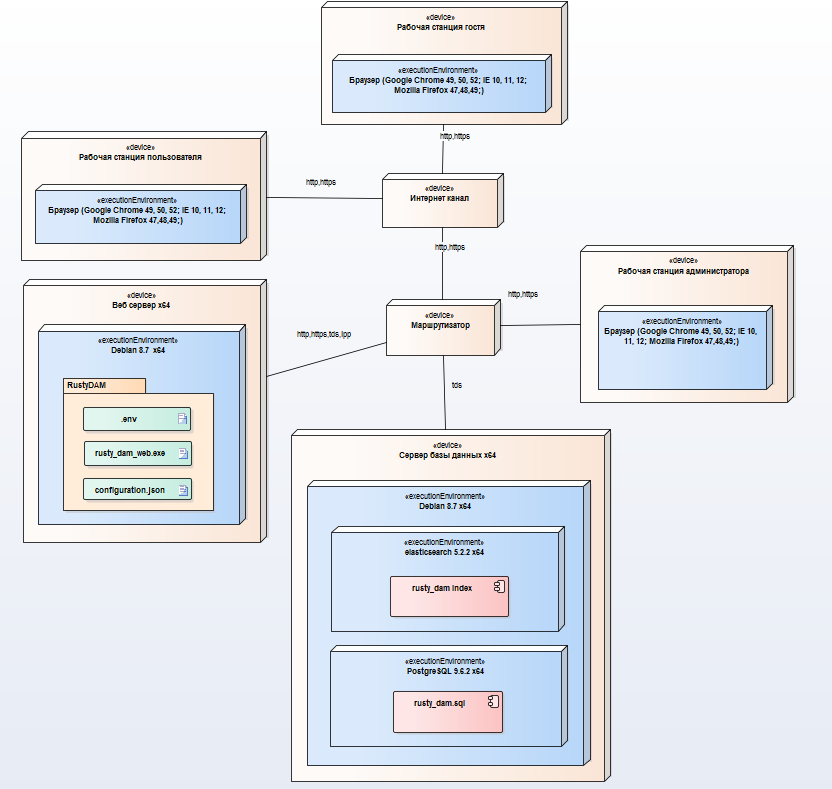
В библиотеке содержиться вся логика работы системы. Используя эту библиотеку можно будет разрабатывать различные дополнительные компоненты системы. Это позволит расширять систему как горизонтально, так и вертикально, что позволит лучше подстроить систему под требования заказчика.

Система должна быть гибкой в работе и поддерживать множество функций, для того что бы упростить ее использование, и предотвратить случаи неправильного использования. Для этих целей очень хорошо подходит реляционные базы данных, т.к. они позволяют создавать сложные связи между объектами. Это необходимо при поиске ассетов в системе, так как поисковые запросы могут быль сложными и могут зависить от других объектов. Например, нужно найти ассеты находящиеся в определенной классификации. Для этого нужно хранить перечень классификаций, в которые определен ассет.

Одно из функциональных требований к системе – производительность и скорость обработки запросов. Реляционные базы данных достаточно быстрые, но если хранить в них сложные структуры с множеством связей, это может негативно сказаться на производительности системы. В этом случает используется NoSql базы данных. Эти базы хранят данные в качестве документов, обычно это Json объекты. Связи в этих базах данных рактически не используются. Такая база нужна для того, что бы хранить всю самую необходимую информацию для ассета. В момент загрузки ассета каким либо компонентом, будет загружаться только основная информация, которая храниться в elasticsearch базе, это значительно увеличит скорость загрузки. Но все же ассет будет загружен не полностью. Оставшиеся части ассета будут загружаться из PostrgeSQL базы по мере необходимости. Таким образом система будет гибкои и удобной в использовании, и привсем этом не будет испытывать проблем с производительностью.



**Рис 3.1. Диаграмма компонентов системы**



**Рис 3.2. Диаграмма развертывания**

На диаграмме развертывания можно увидеть какие компоненты будет содержать программное средство после развертывания. На веб сервер устанавливается операционная система Debian 8.7, так как эта ОС легковесна (требует малое количество ресурсов) она отлично подходит вкачестве операционной системы для сервера. На веб сервер установливается веб интерфейс для работы с DAM системой. Для установки системы достаточно будет скопировать файлы на сервер, настроить .env файл, что бы в нем указывались все параметры, актуальны для данного сервера и сконфигурироваль само приложение с помощью configuration.json файла.

Для сервера базы данных будет использоваться таже операционная система, что и для веб сервера. Так же устанавливается две базы данны: elasticsearch 5.2.2 и PostgreSQL 9.6.2. Базы данных необходимо настроить на разные порты и запустить.

Для клиентских устройств необходимо только наличие веб браузера.

**4 СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА**

Для создания программного средства использовался язык программирования Rust. Rust не является классическим объектно-ориентированным языком программирования: в нем нет наследования, т.к. основной тип данных - структура (по аналогии с языком Си, структуры хранятся на стеке), в место нее используется агрегация и композиция. Есть возможность писать код в функциональном стиле, т.к. язык поддерживает объявление функций разных порядков. Структура проекта состоит из файла Cargo.toml и src фиректории. В Cargo.toml – конфигурационный файл проекта, в нем содержится вся информация касающаяся проекта, автора, зависимости и т.д. Директория src кодержит код проекта. При разработке библиотеки, точкой входа является lib.rs файл. Он должен лежать в корне src директории. Официальное название библиотек – крейт (crate). Для Rust написан пакетный менеджер – cargo. Именно он использует Cargo.toml файл. Cargo имеет возможность создавать воркспейсы (workspace), это аналог солюшена (solution) в С#. Данное программное средство совмещает в себя несколько проектов:

1. libcore
2. file\_storage
3. web\_app

libcore содержит в себе всю основную функциональность программного средства. Эту библиотеку использует web\_app и могут использовать другие приложения. Таким образом, если заказчику потребуется написать собственное программное средство, он может исользовать данную библиотеку что бы получить весь функционал на стороне кода. При помощи этой библиотеки появляется возможность создавать специфичные программные модули нацеленые на решение конкретных бизнесс задач. В папке с данным модулем хранятся все миграции базы данных, таким образом, при развертывании базы данных, необходимо применить все миграции, что бы подготовить схему для работы (создать все необходимые таблицы, связи, индексы, функции и т.д.).

file\_storage представляет собой веб API для работы с файлами. Это приложение будет работать на стороне сервера и осуществлять сохранение и отправку файлов клиенту. Это позволит обеспечить хранение файлов на отдельном сервере, что способствует уменьшению нагрузки на основные сервера. file\_storage будет управлять файломи таким образом, что бы сохранить их структуру и обеспечить версионность файлов, в случае если потребуется сохранить несколько файлов с одинаковым именем для одного ассета.

web\_app предоставит пользовательский интерфейс в виде веб страницы, через которую будет удобно управлять системой и пользователи смогут использовать весь функционал, предоставляемой системой используя любое устройство с установленным на нем браузером.

## **4.1. libcore - основной программный модуль**

Точкой входа данного модуля является файл с именем lib.rs. В файле перечисляются все внешние библиотеки, использующиеся в данном модуле, внутренние модули (аналогом из других языков является пространство имен) и трейты – аналог абстрактных классов в объектно-ориентированных языках, таких как C#, Java, но поддерживается реализация нескольких трейтов для одной структуры. Внутренние модули, перечисленные в этом файле, имеют имя файла или папки, которое является пространством имен того кода, который содержится в этих файлах\папках. В корневом пространстве имен данногомодуля перечислены следующие внутренние модули (пространства имен):

1. Configuration
2. Connection
3. Es
4. Pg
5. models

Configuration – модуль, в котором содержиться трейт, который необходимо реализовать пользователю. Структура, которая реализует этот трейт должна возвращать IP адрес и порт, который прослушивается elasticsearch базой и строку подключения к базе PostgreSQL. Так же указывается путь к папке, в которой система будет сохранять временные файлы, путь к файловому серверу и имя подключения (идентификатор соеденения, нужен для того, что бы можно было сконфигурировать разные среды, например: develop, staging, production).

Connection – модуль описывающий соединение с системой. В этом модуле описано две структуры: App и Connection. Connection содержит подключения к базам данных. Эти подключения должны быть в единственном числе, поэтому пользователю предоставляется структура App, которая содержит Rc<RefCell<Connection>>, таким образом при копировании экземпляра структурысоздание новых подключений происходить не будет. Rc расшифровывается как reference counted (подсчет ссылок), это означает, что при копировании внутренний объект не копируется, а создается новая ссылка на тот же объект (подсчет ссылок обеспечивает своевременное очищение памяти, когда все ссылки выйдут из области действия). App дает возможность выполнять действия от имени администратора, на случай, если у обычного пользователя не будет хватать прав на выполнение каких либо критически важных операций, которые может выполнять только администратор.

При создании экземпляра Connection структуры, пользователю необходимо залогиниться, что бы получить доступ к системе. Во время авторизации пользователя происходит создание сессии. В пространстве имен models::session содержится структура, описывающая логику создания сессии. Для структуры Session реализован трейт Drop работающий как деструктор в языке C++, таким образом реализуется удаление разлогинивание пользователя. Session дает возможность авторизоваться с поможью логина и пароля или если известен идентификатор сессии, то используется он вместе с логином.

Es – внутренний модуль, содержащий клиент elasticsearch базы данных.

Es содержит три уровня абстракции, которые позволяют уменьшить размер кода и облегчить его понимание. Первым уровнем является EsClient структура, она содержит в себе клиент, предоставляемый библиотекой rs-es, и оборачивает вызовы методов базового клиента со стандартным набором параметров. Название индекса в elasticsearch храниться так же в EsClient, а тип документа возвращает сама струкрура, которая используется в качестве парамента. Это становиться возможным после того, как структура, которую необходимо сохранить в elasticsearch базу, реализует трейт EsData. Данный трейт является маркером того, что структуру можно использовать при операциях с elasticsearch.

Следующим уровенем абстракции является EsRepository структура. Она нужна для того, что бы результат выполнения запросов в базу преобразовать в объекты данных, так называемые data transfer object, они не несут какой либо логики, поэтому они должны быть преобразованы в объекты более высокого уровня. Эти занимется следующий уровень абстракции – EsService. Данный уровень абстракции является последним и он осуществляет финальные преобразования перед тем, как вернуть результат вызывающему коду. EsService практически не содержит кода, тесно связанного с rs-es библиотекой, следовательно, если появится необходимость использовать другую библиотеку, это можно будет сделать не изменяя другой код, т.к. интерфейс EsService не измениться.

Pg – последний модуль того же уровня. В данном модуле описана лишь логика создания подключения к PostgreSQL базе. Благодаря библиотеке diesel, нет необходимости писать какие либо абстракции над стандартным функционалом, т.к. diesel сделает всю работу сам. Пользователю нужно лишь писать запросы используя структуры сгенерированные на основе схебы базы на этапе компиляции, после чего код преобразуется в SQL запрос и отправляется в базу данных на выполение.

models содержит в себе те объекты, с которыми пользователь будет непосредственно взаимодействовать. Основными из них являются:

1. classification
2. field
3. file
4. record
5. user

classification содержит 2 структуры: Classification – структура, описывающая классификацию как отдельный объект системы и RecordClassification – структура являющаяся частью ассета. Classification содержит методы для изменения имени, перемещения классификации к другому родителю и д.р. Изменения производимие над экземпляром этой структуры влечет за собой изменение всех ассетов, которые принадлежат изменяемой классификации. Вся информация о классификации берется из PostgreSQL базы. В базе, таблица классификаций является рекурсивной, т.о. используя рекурсивнеы запросы можно получить все дерево классификаций, путь к отдельной классификации от корня и т.д. Так как PostgreSQL база поддерживает множество языков программирования, помимо SQL, можно написать функции, которые будут выполнять операции на стороне базы, тем самым разгрузив систему. RecordClassification – неизменяемая структура, которая представляет собой классификацию, к которой относится ассет. Данную структуру можно либо добавить, либо удалить из ассета. RecordClassification хранит только свой идентификатор, иденттификатор родителя и путь в дереве классификаций. Это позволит уменьшить расход памяти.

Field содержит так же 2 структуры, как и classification, но описывает поведения полей ассета. Field структура содержит полную информацию относительно поля и не зависит от конкретного ассета. RecordField в отличии от Field неизменяемо, кроме своего значения. Пользователь может получить значение поля загруженного ассета и установить новое значение. Поле может принимать значение «Пусто», строковое значение (текст), логичесткое значение («Правда» или «Ложь») и целочисленное. В будущем список типов, поддерживаемых полем расширится.

User – описывает пользователя системы. Поддерживаются основные операции, такие как: добавление, удаление, редактирование и загрузка пользователя по идентификатору. Пользователя можно добавить в группу, что бы предоставить ему права, определенные для группы.

Record – основной объект системы, ассет. Единственная структура, которая храниться в elasticsearch базе. Ассет представляет собой коллекцию полей, классификаций и файлов, а так же вспомогательно информации (когда и кем был создан, когда и кем был изменен). Большая часть информации храниться в elasticsearch базе, остальное подгружается по мере необходимости.

# **6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**6.1** Краткая характеристика разрабатываемого ПО

Разрабатываемое в дипломном проекте программное средство управления электронными документами на предприятии позволит упростить управление, хранение, поиск нужных документов для всех служащих предприятия, которые будут использовать данное программное средство. Единое хранилище электронных документов позволит быстро получить доступ к самым последним версиям файла и позволит значительно упростить обмен документами между сотрудниками как внутри одного отдела, так и между ними. Программное средство так же позволит автомотизировать некоторые ручные операции, для того что бы экономить время и ресурсы. Такие операции на больших предприятиях c документооборотом могут требовать больших затрат человеко-часов. Например, для формирования отчета о новых фотографиях для рекламной компании продукции сотруднику необходимо вести постоянный учет файлов, которые появились в системе и вести подробное описание данных файлов, что бы можно было вывести статистику. Если отчеты требующие больших временных затрат нужно предоставлять ежедневно, то один сотрудник не будет успевать составлять отчеты вовремя. Если увеличить число сотрудникв, это повлечет за собой увеличение расходов. При этом сотрудник все время будет занят сбором информации вместо его основной работы, что плохо скажется на производительности предприятия. А если учитывать что отчетов может потребоваться больше одного, тогда количество труда значительно увеличится. Это касается не только отчетов, но и других ручных операций, которые могут быть заменены программным средством, которое не будет требовать человеческого вмешательства, не будет совершать ошибки и будет работать значительно быстрее человека. Данное программное средство разрабатывается для того, что бы решить ранее описанные проблемы наиболее эффективно.

Программное средство будет свободно распростроняться на рынке IT. Подробная документация позволит обучить персонал работать с системой, а документация разработчика позволит подготовить специалистов для поддержки системы и ее расширения в будущем.

Основной задачей технико-экономического обоснования программного средства является подтверждение актуальной потребности в разрабатываемой программном продукте у потенциальных пользователей и определение эконо-мической выгоды от внедрения программного средства как со стороны разра-ботчика (проявляется в виде чистой прибыли от реализации ПС), так и поль-зователя (выражается в экономии трудовых и экономических ресурсов).

Расчеты выполнены на основании методического пособия [2].

## **6.2** Расчет затрат на разработку ПО

**6.2.1** Расчет затрат на основную заработную плату разработчиков

Затраты на основную заработную плату рассчитываются по формуле:

,

где *n* – количество исполнителей, занятых разработкой конкретного ПО;

– дневная заработная плата i-го исполнителя, руб.;

– трудоемкость работ, выполняемых *i*-м исполнителем, дн.

Для реализации программного средства управления электронными документами было принято решение выбрать команду разработчиков в составе трех инженеров-программистов, одного архитектора базы данных и тестировщика. Это объясняется тем, что конечный продукт должен состоять из трех частей: веб-приложения, непосредственно с которым будут работать пользователи, основной модуль, сорержащий весь функционал приложения, веб-сервис для работы с файлами (сохранение, управление каталогами, версионность, загрузка). Поэтому целесообразно вести работу параллельно и поручить разработку пользовательского приложения и файлового хранилища веб-программистам, а третью часть отдать на выполнение специалисту в области работы с базами данных. Такое разделение позволит закончить проект вовремя с учетом рисков, связанных с разработкой, и выполнить его качественнее благодаря специализации разработчиков.

В качестве размера часовой тарифной ставки 1-го разряда для расчетов заработной платы выбирается значение, принятое в организации, которая занимается разработкой проекта, и равное 10 руб для разработчиков и архитектора базы данных, 8 руб. - для тестировщика.

Среднемесячное количество рабочих дней при пятидневной рабочей неделе в 2017 году составляет 21,1 дн.

В таблице 6.1 сведены данные о команде разработчиков, их окладе и назначенном объеме работ для каждого.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Участник команды | Выполняемые работы | Часовая ставка, тыс. руб | Трудоем-кость работ, часов. | Основная заработная плата, тыс. руб |
| 1 | Rust – разработчик | Разработка основного программного модуля (серверной логики) ПС | 0,010 | 240 | 2,4 |
| 2 | Rust – разработчик | Разработка модуля файлового хранилища и веб-сервиса для него | 0,010 | 80 | 0,8 |
| 3 | Архитектор базы данных | Проектирование и разработка базы данных для приложения | 0,010 | 80 | 0,8 |
| 4 | ReactJS - разработчик | Разработка пользовательского интерфейса ПС. | 0,010 | 80 | 0,8 |
| 5 | Тестировщик | Тестирование программного средства. | 0,008 | 56 | 0,448 |
| Премия, % | | | | | 50 | |
| Итого затраты на основную заработную плату разработчиков | | | | | 7,872 | |

Таблица 6.1 – Расчет затрат на основную заработную плату команды разработчиков

**6.2.2** Расчет затрат на дополнительную заработную плату

Затраты на дополнительную заработную плату команды разработчиков включает выплаты, предусмотренные законодательством о труде, и определяется по формуле:

,

где Зо – затраты на основную заработную плату с учетом премии (руб.);

Нд – норматив дополнительной заработной платы (10-20%).

Примем Нд=15% и получим:

**6.2.3** Расчет отчислений на социальные нужды

Обязательные отчисления на социальные нужды включают в себя выплаты в фонд социальной защиты населения и обязательное страхование и определяются в соответствии с действующими законодательными актами по формуле:

,

где Нсоц – норматив отчислений на социальные нужды (согласно действующему законодательству), %.

Учитывая, что в настоящее время норматив отчислений на социальные нужды в Республике Беларусь составляет 35%, получаем:

**6.2.4** Расходы по статье «Прочие затраты»

Прочие затраты включают затраты, связанные с разработкой конкретного программного обеспечения напрямую, а также связанные с функционированием организации-разработчика в целом. Расчет прочих затрат выполняется в процентах от затрат на основную заработную плату команды разработчиков с учетом премии по формуле:

,

где Нпз – норматив прочих затрат (100–150%).

Примем Нпз=120% и рассчитаем сумму прочих затрат:

**6.2.5** Расчет общих затрат на разработку и себестоимости ПО

Полная сумма затрат на разработку программного обеспечения находится путем суммирования всех рассчитанных статей затрат по формуле:

.

Для разрабатываемого ПС затраты на разработку составят:

Предполагается, что компания-разработчик также будет заниматься сопровождением программного средства. В связи с этим рассчитываются расходы на эти услуги, которые принимаются в размере Нс=30% от основных затрат на разработку:

Себестоимость программного средства будет определяться как сумма затрат на разработку и сопровождение:

Расчет затрат на разработку и сопровождение программного средства сведен в таблицу 6.3.

|  |  |
| --- | --- |
| Статья затрат | Сумма, тыс. руб. |
| Основная заработная плата команды разработчиков | 7,872 |
| Дополнительная заработная плата команды разработчиков | 1,18 |
| Отчисления на социальные нужды | 3,17 |
| Прочие затраты | 9,45 |
| Общая сумма затрат на разработку | 21,67 |
| Расходы на сопровождение | 6,5 |
| Себестоимость программного средства | 28,17 |

Таблица 6.2 – Затраты на разработку и сопровождение ПО

Рассчитанное значение полной себестоимости, которая составила 28,17 тыс. руб., будет использоваться в дальнейшем для определение цены ПС.

## **6.3** Оценка результата (эффекта) от использования (или продажи) ПО

В рамках данного технико-экономического обоснования рассматривается экономический эффект, который получит компания-разработчик, а также экономический и неэкономический эффект для заказчика и пользователей разрабатываемого программного средства.

**6.3.1** Экономический эффект у разработчика

Экономический эффект для организации-разработчика представляет собой прибыль, полученную от продажи разработанного программного продукта множеству потребителей. Прибыль от реализации в данном случае напрямую зависит от объемов продаж, цены и затрат на разработку данного ПО.

Таким образом, необходимо рассчитать предполагаемое количество копий проданное за год. Количество лицензий проданых за год в среднем, берется равным 36 шт.

Цену программного средства для управления цифровыми документами будем определять, исходя из затрат на разработку и реализацию ПО и запланированного уровня рентабельности. В таком случае расчет прибыли, получаемой организацией-разработчиком, осуществляется по формуле:

,

где Сп – себестоимость программного обеспечения, руб.;

Ур – запланированный норматив рентабельности, %.

N – количество лицензий ПО, которое будет куплено клиентами за год

Выбрав Ур=30%, рассчитаем ожидаемую прибыль с одной лицензии:

В настоящее время, в Беларуси налог на добавленную стоймость при реализации услуг, работ и имущественных прав равен 20%:

Налог на прибыль составляет 18%:

Отпускная цена одной лицензии будет определяться:

Ц – цена реализации одной копии (тыс. руб.),

– сумма расходов на разработку и реализацию (тыс. руб.),

N – количество лицензий Пб которое будет куплено киентами за год

П – прибыль, получаемая организацией-разработчиком от реализации одной копии программного продукта (тыс. руб.),

НДС – сумма налога на добавленную стоймость (тыс. руб.),

НП – налог на прибыль (тыс. руб.),

Таким образом, при продаже месячной лицензии программного продукта трем заказчикам по цене 1,11 тыс. руб. за лицензию разработчик получит экономическую выгоду в размере 28,17 тыс. руб.

**6.3.2** Оценка эффекта у заказчика

Судф по ценам на DAM системы за 2016 год [1], в среднем, годовая стоймость равняется 1800 руб. за лицензию [3], которая действительна месяц. Цены на лицензии так же зависят от количества пользователей и размер требуемой памяти, так что данная цена может значительно вырасти [4]. Преобретая продукт по рассчитанной выше цене, заказчик в среднем может получить выгоду от разработки в выбранной организации в размере:

руб.

Предполагается, что программное средство управления документами на предприятии уменьшит затраты и решит проблемы с управлением файлами на предприятии. Это позволит увеличить оборот документов и тем самым увеличить прибыль, так как больше времени будет уделяться более приоритетным задачами.

**6.3.3** Оценка эффекта у пользователя

Данное программное средство разрабатывается в виде веб-приложения и будет доступно всем сотрудникам предприятия, которое приобрело данный продукт. Сотрудникам не придется самим организовывать место хранения файлов. Так как для сотрудников разных отделов это может быть не тривиальной задачей. При использовании данного продукта, пользователи получат веб-приложение, доступное под любой платформой и любом устройстве. Данное приложение будет содержать весь функционал, который будет необходим для поиска, загрузки, обмена и выгрузки файлов. Это значительно удобнее, потому что для отправки файлов другому пользователю, достаточно будет отправить ему ссылку на ассет, содержащий нужный файл. Получив ссылку на ассет, пользователь сможет увидеть всю информацию об этом ассете, касающуюся только получателя файла (видимость данных зависит от пользовательской роли в системе и пользователь не будет видить информацию, которая относиться к той роли, которой у пользователя нет). Пользователь сможет выбрать определенную версию файла для скачки, так что информация будет всегда актуальна. Продукт будет обладать расширяемостью, для того, что бы соответствовать самым последним бизнесс требованиям.

Данный продукт – хороший выбор тех предприятий, у кого есть трудности с большим объемом файлов, которые нужны для большого числа сотрудников. Продукт поможет уменьшить затраты на обслуживание и управление всеми файлами. Уменьшение ручного труда и делегирование ответственности на систему позволит уменьшить количество ошибок при работе с файлами, тем самым уменьшив затраты предприятия.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Jacobsen, Jens; Schlenker, Tilman; Edwards, Lisa (2005). Implementing a Digital Asset Management System: For Animation, Computer Games, and Web Development. Focal Press
2. Krogh, Peter (2009). The DAM Book, Second Edition. O'Reilly Media
3. Krogh, Peter (2005). The DAM Book: Digital Asset Management for Photographers. O'Reilly Media
4. Austerberry, David (2006). Digital Asset Management, Second Edition. Focal Press
5. Mauthe, Andreas; Thomas, Peter (2004). Professional Content Management Systems: Handling Digital Media Assets. Wiley

[1] <http://digitalassetmanagementnews.org/vendors/dam-vendors-2016-pricing-survey-released/>

[2] <https://www.bsuir.by/m/12_100229_1_80049.pdf>

[3] <https://www.imagerelay.com/pricing>

[4] <http://www.intelligencebank.com/uk/digital-asset-management/pricing>