СОДЕРЖАНИЕ

[**ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ, СОКРАЩЕНИЯ** 6](#_Toc482546032)

[**ВВЕДЕНИЕ** 7](#_Toc482546033)

[**1 АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ** 9](#_Toc482546034)

[**1.1** Формирование требований к проектируемому программному средству 9](#_Toc482546035)

[**1.2** Исследование методов и подходов в предметной области 9](#_Toc482546036)

[**1.3** Обзор существующих аналогов 14](#_Toc482546037)

[**1.4** Обоснование выбора языка разработки и баз данных 18](#_Toc482546038)

[**1.5** Требования к программному средству 21](#_Toc482546039)

[**2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ** 22](#_Toc482546040)

[**2.1** Диаграмма прецедентов 22](#_Toc482546041)

[**2.2** Общие требования к программному средству 25](#_Toc482546042)

[**2.3** Требования к составу клиентской части приложения 25](#_Toc482546043)

[**2.4** Спецификация функциональных требований 26](#_Toc482546044)

[**3** **ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА** 27](#_Toc482546045)

[**3.1** Структура программного средства 27](#_Toc482546046)

[**4.1. Основной программный модуль** 31](#_Toc482546047)

[**5 РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ** 36](#_Toc482546048)

[**6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ** 39](#_Toc482546049)

[**6.2** Расчет затрат на разработку ПО 40](#_Toc482546050)

[**6.3** Оценка результата (эффекта) от использования (или продажи) ПО 43](#_Toc482546051)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 47](#_Toc482546052)

# **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.

ПС – программное средство.

# **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**

## Формирование требований к проектируемому программному средству

* + 1. Назначение разработки и состав выполняемых функций

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 cовместной 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.

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 ассетам будет использоваться веб-страница;
5. Заполнять и редактировать метаданные ассета.

## Исследование методов и подходов в предметной области

* + 1. Клиент-серверная архитектура

Архитектура «клиент-сервер» определяет общие принципы организации взаимодействия в сети, где имеются серверы, узлы-поставщики некоторых специфичных функций (сервисов) и клиенты, потребители этих функций.

Практические реализации такой архитектуры называются клиент-серверными технологиями. Каждая технология определяет собственные или использует имеющиеся правила взаимодейстия между клиентом и сервером, которые называются протоколом обмена (протоколом взаимодействия).

В любой сети (даже одноранговой), построенной на современных сетевых технологиях, присутствуют элементы клиент-серверного взаимодействия, чаще всего на основе двухзвенной архитектуры. Двухзвенной (two-tier, 2-tier) она называется из-за необходимости распределения трех базовых компонентов между двумя узлами (клиентом и сервером).

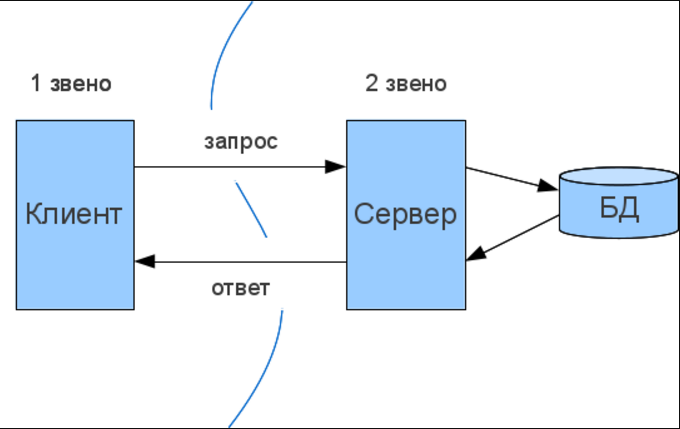


Рисунок 1.1 - Двухзвенная клиент-серверная архитектура

Двухзвенная архитектура используется в клиент-серверных системах, где сервер отвечает на клиентские запросы напрямую и в полном объеме, при этом используя только собственные ресурсы. Т.е. сервер не вызывает сторонние сетевые приложения и не обращается к сторонним ресурсам для выполнения какой-либо части запроса.

Для работы с файлами реализуется модель файлового сервера, представлявшего доступ файловым ресурсам, в т.ч и к удаленной базе данных. В этом случае выделенный узел сети является файловым сервером, на котором размещены файлы базы данных. На клиентах выполняются приложения, в которых совмещены компонент представления и прикладной компонент (СУБД и прикладная программма), использующие подключенную удаленную базу как локальный файл. Протоколы обмена при этом представляют набор низкоуровневых вызовов операций файловой системы.

Еще одна тенденция в клиент-серверных технологиях связана со все большим использованием распределенных вычислений. Они реализуются на основе модели сервера приложений, где сетевое приложение разделено на две и более частей, каждая из которых может выполняться на отдельном компьютере. Выделенные части приложения взаимодействуют друг с другом, обмениваясь сообщениями в заранее согласованном формате. В этом случае двухзвенная клиент-серверная архитектура становится трехзвенной (three-tier, 3-tier).

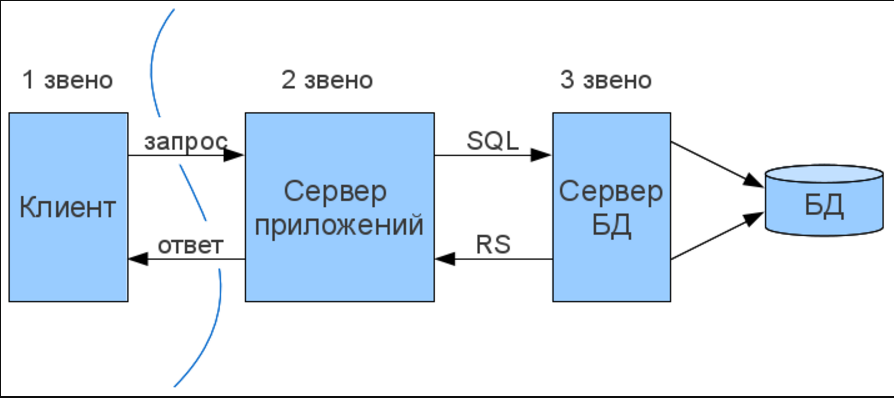


Рисунок 1.2 - Трехзвенная клиент-серверная архитектура

Трехзвенная архитектура может быть расширена до многозвенной (N-tier, Multi-tier) путем выделения дополнительных серверов, каждый из которых будет представлять собственные сервисы и пользоваться услугами прочих серверов разного уровня.

Данный дипломный проект использует двухзвенную клиент-серверную архитектуру.

* + 1. RAII идиома программирования

Для уменьшения трудозатрат на разработку сложного программного обеспечения, целесообразно использовать готовые унифицированные решения, разработанные для облегчения коммуникации между разработчиками и снижения количества ошибок.

Идиома программирования — устойчивый способ выражения некоторой составной конструкции в одном или нескольких языках программирования. Идиома является шаблоном решения задачи, записи алгоритма или структуры данных путём комбинирования встроенных элементов языка.

Идиому можно считать самым низкоуровневым шаблоном проектирования, применяемым на стыке проектирования и кодирования на языке программирования. Идиома предписывает конкретный способ реализации определённых деталей и отношений между ними средствами конкретного языка. Набор идиом зависит от языка программирования, и одна и та же идиома может выглядеть в двух разных языках по-разному, либо в ней может не быть надобности в одном из них.

Получение ресурса есть инициализация (англ. Resource Acquisition Is Initialization (RAII)) — программная идиома объектно-ориентированного программирования, смысл которой заключается в том, что с помощью тех или иных программных механизмов получение некоторого ресурса неразрывно совмещается с инициализацией, а освобождение — с уничтожением объекта.

Типичным (хотя и не единственным) способом реализации является организация получения доступа к ресурсу в конструкторе, а освобождения — в деструкторе соответствующего класса. Поскольку деструктор автоматической переменной вызывается при выходе её из области видимости, то ресурс гарантированно освобождается при уничтожении переменной. Это справедливо и в ситуациях, в которых возникают исключения. Это делает RAII ключевой концепцией для написания безопасного при исключениях кода в языках программирования, где конструкторы и деструкторы автоматических объектов вызываются автоматически, прежде всего — в C++.

В языке программирования Rust, деструктором является метод drop трейта Drop. Если тип работает с ресурсами, которые требуют освобождения, этот тип должен реализовать трейт Drop, тем самым описав логику освобождения ресурса. Таким образом реализуется RAII идиома в языке программирования Rust.

* + 1. newtype паттерн программирования

Данный паттерн характерен для таких языков программирования как Haskell и Rust. Суть этого паттерна заключается в том, что бы создать новый тип, который является оберткой другого типа, но содердит расширенный функционал, по сравнению с внутренним типом. В других языках данный шаблон проектирования называется «Оберткой». В отличии от других языков, Rust и Haskell имеют поддержку со стороны языка. В Haskell есть ключевое своло newtype для обявления типа, а в Rust есть синтаксис для создания типа-записи (tuple struct).

Основным стимулом использовать данный паттерн является создание абстракций. Данный паттерн позволяет разделять имплементацию между несколькими типами, полностью контролируя интерфейст взаимодействия. Использование newtype вместо расширения уже существующего типа позволяет сохранить обратную совместимость. Использование данного паттерна в языке программирования Rust имеет как положительные, так и отрецательные стороны.

Положительные:

1. Внутренний и внешний тип не совместимы (разные типы), в отличие от псевдонимов типа;
2. «Обертка» является структурой нулевой стоймости (оптимизатор удалит ее исходнего кода, при компиляции);
3. Внутренний тип является приватным (повышается инкапсуляция кода).

Минусом данного паттерна является отсутствие возможности указать метод внутреннего типа, доступным снаружи, это увеличивает размер кода. Однако отсутствие данной возможности не позволит нарушить инкопсуляцию.

* + 1. Архитектурный стиль REST

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами.

В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.

В сети Интернет вызов удалённой процедуры может представлять собой обычный HTTP-запрос (обычно «GET» или «POST»; такой запрос называют «REST-запрос»), а необходимые данные передаются в качестве параметров запроса. В отличие от веб-сервисов (веб-служб) на основе SOAP, не существует «официального» стандарта для RESTful веб-API. Дело в том, что REST является архитектурным стилем, в то время как SOAP является протоколом. Несмотря на то, что REST не является стандартом сам по себе, большинство RESTful-реализаций используют стандарты, такие как HTTP, URL, JSON и XML.

Рой Филдинг — один из главных авторов спецификации протокола HTTP, описывает влияние архитектуры REST на масштабируемость следующим образом:

1. Простота унифицированного интерфейса;
2. Открытость компонентов к возможным изменениям для удовлетворения изменяющихся потребностей (даже при работающем приложении);
3. Прозрачность связей между компонентами системы для сервисных служб;
4. Переносимость компонентов системы путем перемещения программного кода вместе с данными;
5. Надежность является устойчивостью к отказам на уровне системы при наличии отказов отдельных компонентов, соединений, или данных.

Существует шесть обязательных ограничений для построения распределённых REST-приложений по Филдингу:

1. Модель клиент-сервер;
2. Отсутствие состояния;
3. Кеширование;
4. Единообразие интерфейса;
5. Слои.

Выполнение этих ограничительных требований обязательно для REST-систем. Накладываемые ограничения определяют работу сервера в том, как он может обрабатывать и отвечать на запросы клиентов.

Действуя в рамках этих ограничений, система приобретает такие желательные свойства как производительность, масштабируемость, простота, способность к изменениям, переносимость, отслеживаемость и надежность.

Преимущества REST:

1. Надёжность (за счёт отсутствия необходимости сохранять информацию о состоянии клиента, которая может быть утеряна);
2. Производительность (за счёт использования кэша);
3. Масштабируемость;
4. Прозрачность системы взаимодействия (особенно необходимая для приложений обслуживания сети);
5. Простота интерфейсов;
6. Портативность компонентов;
7. Лёгкость внесения изменений;
8. Способность эволюционировать, приспосабливаясь к новым требованиям (на примере Всемирной паутины).

## Обзор существующих аналогов

Прежде чем приступать к работе следует рассмотреть аналогичные сайты и ресурсы, для того, чтобы понять все ошибки и недостатки, а также достоинства создателей этих сайтов. Это следует сделать для того, чтобы продукт был более оптимизированным и уникальным по сравнению с остальными.

* + 1. Анализ программного средства ADAM Software

Программное средство ADAM Software 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.

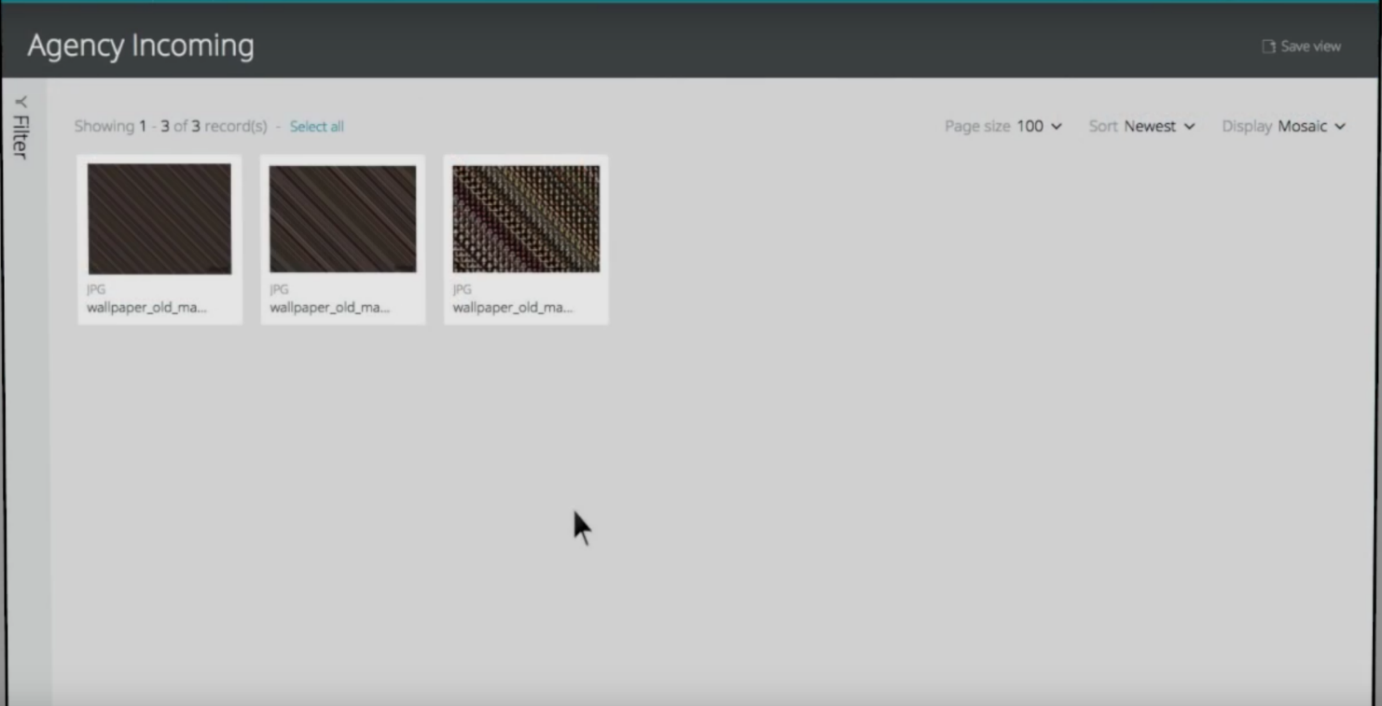


Рисунок 1.3 - Отображение списка ассетов в системе

На рисунке 1.3 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.

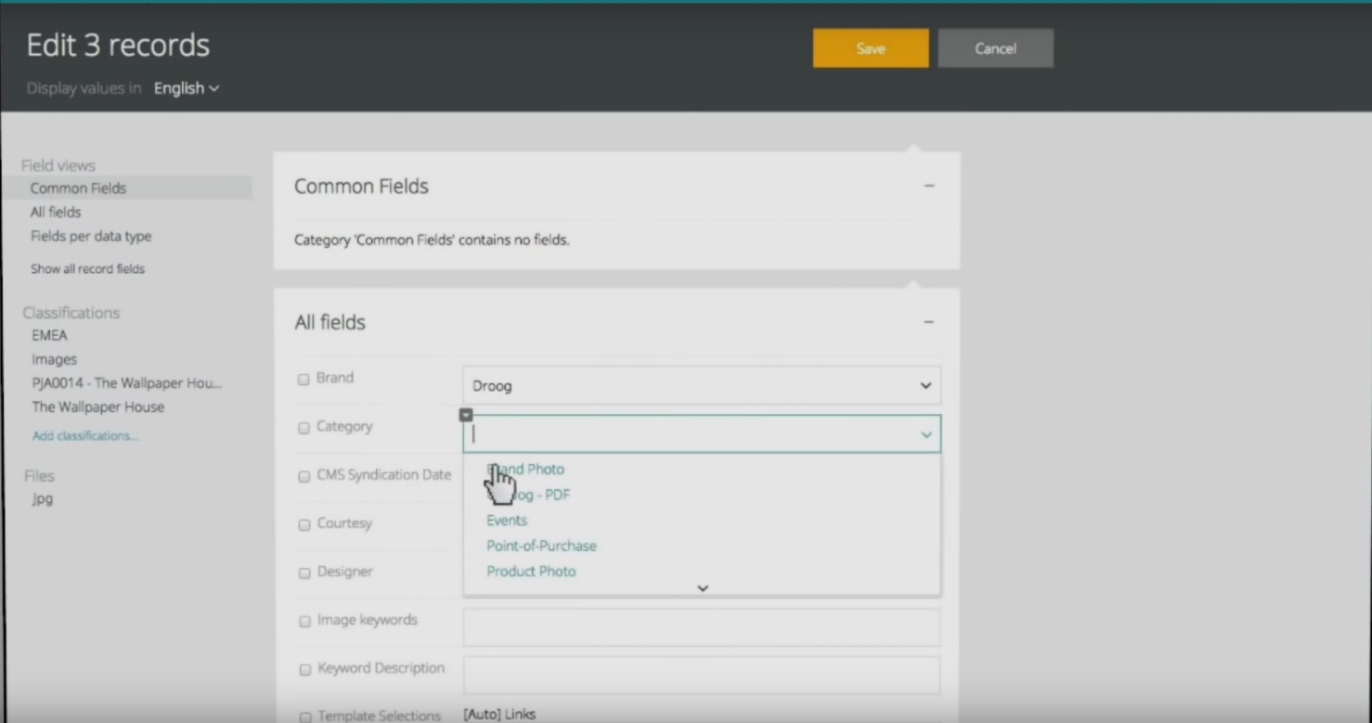
****

Рисунок 1.4 - Редактирование ассетов

На рисунке 1.4 можно увидеть, что есть возможность редактировать сразу несколько ассетов. Так же на скриншоте видно, как выглятят поля ассета и как устанавливаются значения этих полей.

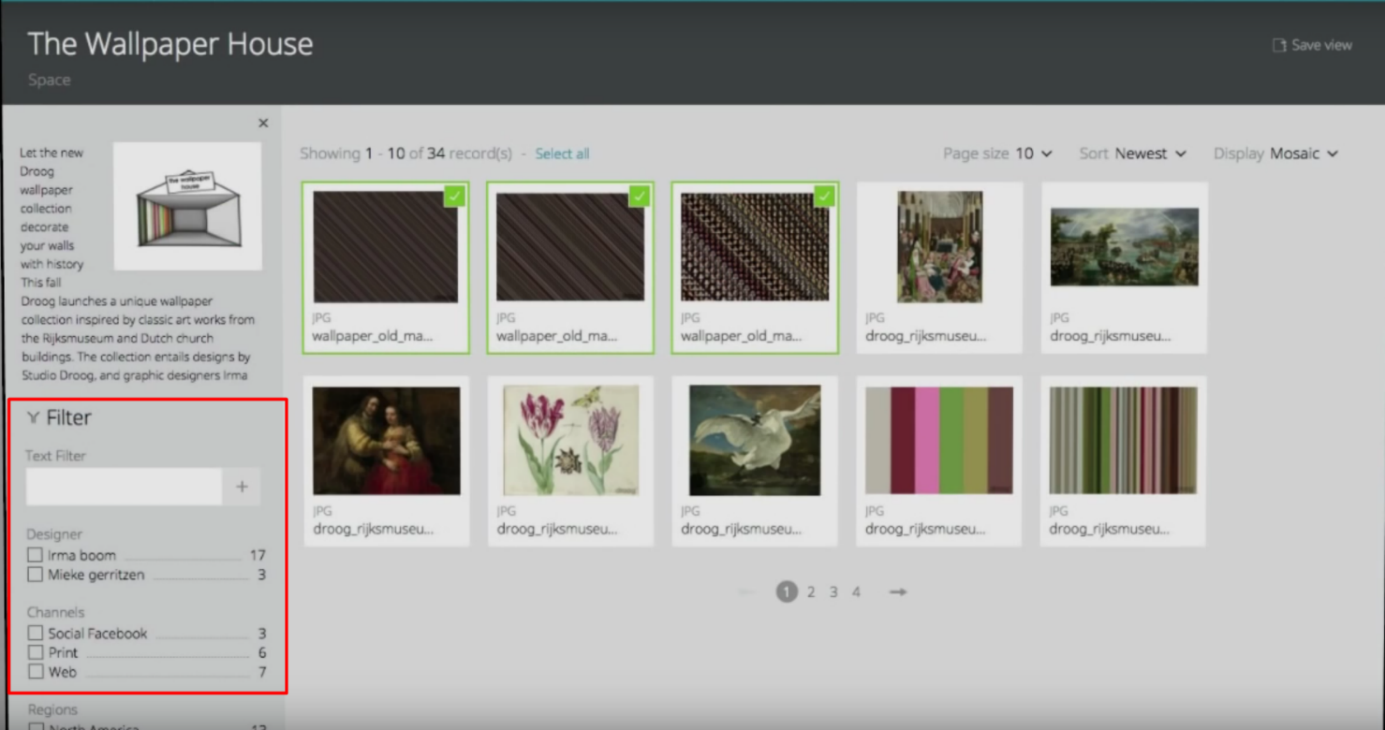


Рисунок 1.5 - Фильтры

На рисунке 1.5 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.

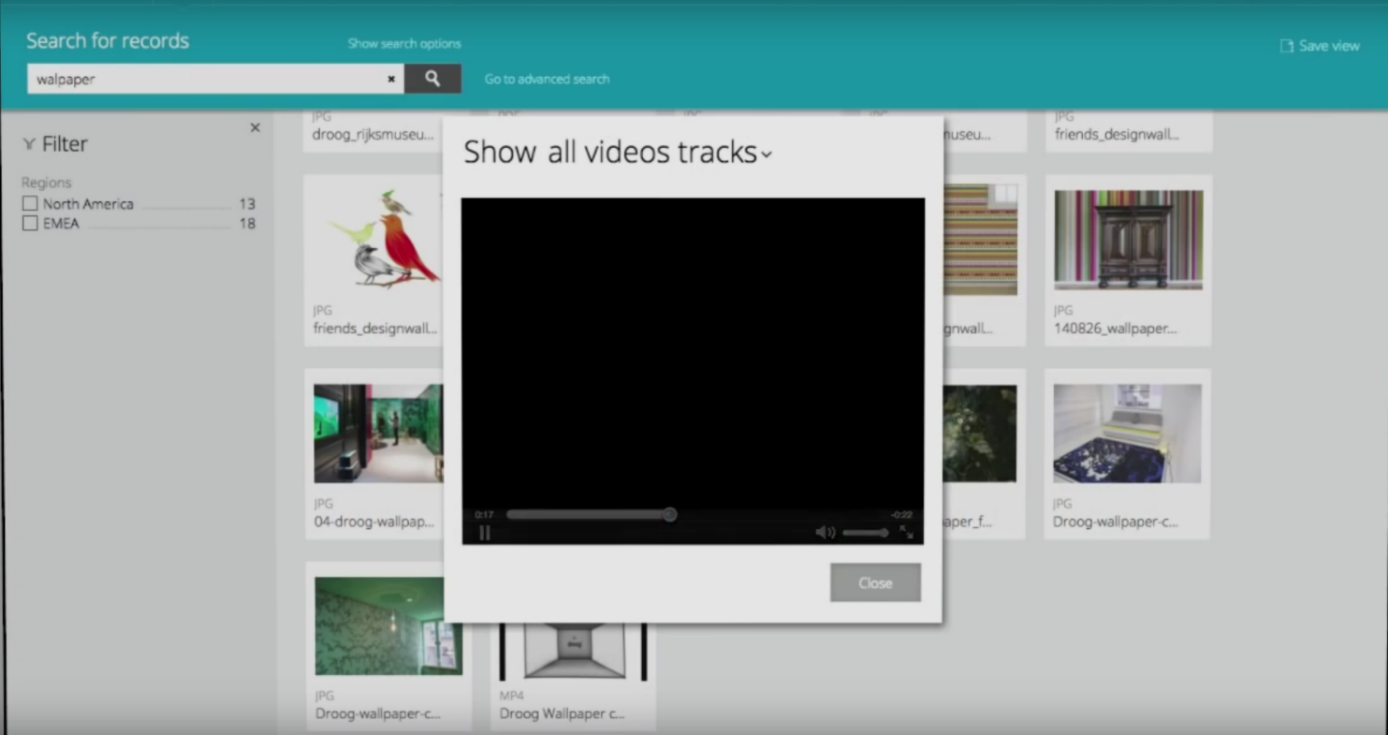
****

Рисунок 1.6 - 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.

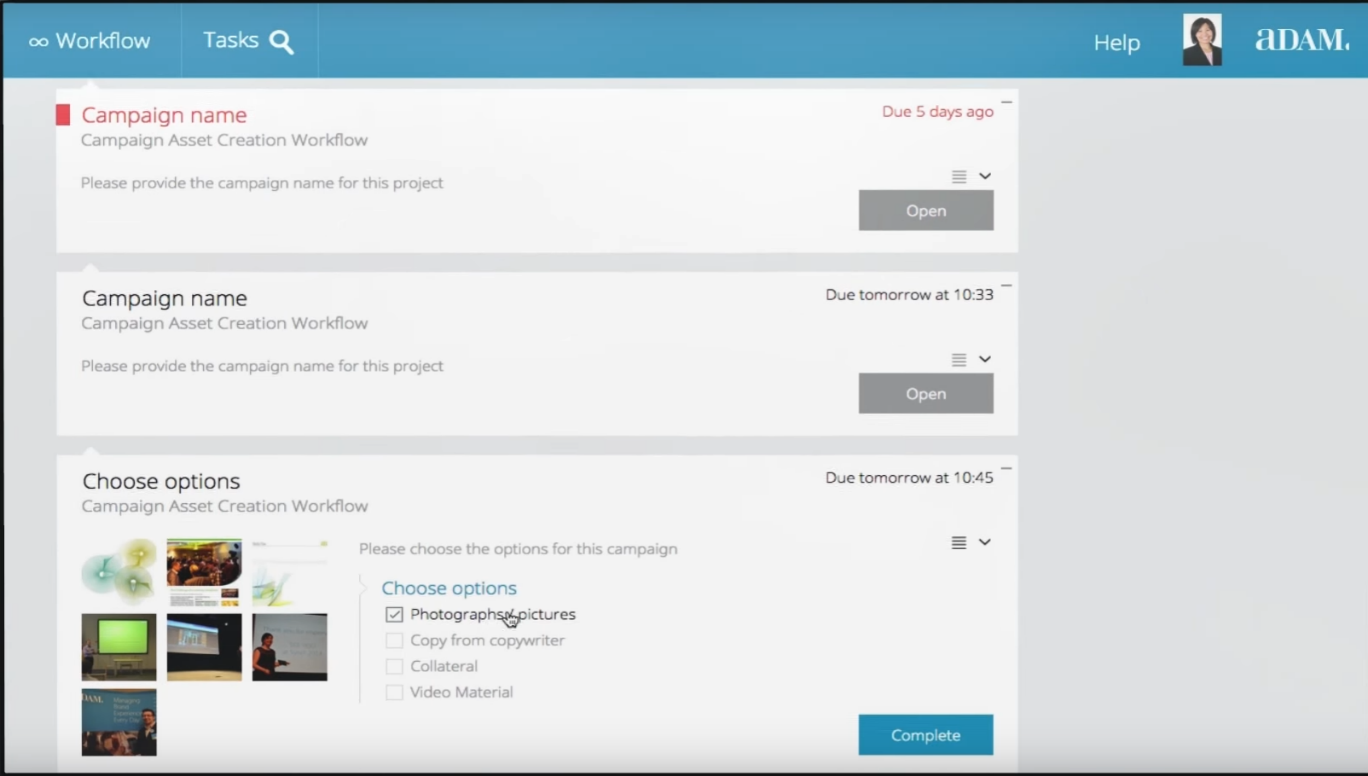


Рисунок 1.7 - Процессы

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 1.7 можно увидеть как выглядят задачи и как их выполнять.

Приложение ADAM Software имеет следующие достоинства:

1. Огромный функционал, способный покрыть базовые бизнесс требования любого заказчика;
2. Удобный пользовательский интерфейс;
3. Гибкая архитектура, позволяющая администраторам легко настраивать систему под требования заказчика;
4. Встроенная интеграция с AgilePoint и ConceptShare;

## Обоснование выбора языка разработки и баз данных

* + 1. Язык программирования Rust

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.

* + 1. База данных PostgreSQL

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.

* + 1. База данных elasticsearch

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. Скорость поиска позволяет получать результаты в момент написания запроса, почти мгновенно. Elasticsearch является распределенной базой данных, это означает, что она может быть разделена на несколько кластеров. Каждый узел сети, может содержать один или несколько кластеров, которые могут делегировать запросы другим кластерам. Балансировка нагрузки выполняется автоматически. Данные, хранящиеся в одном индексе могут быть разделены на разные кластеры. Когда индекс создан, количество кластеров индекса не может быть изменено.

Elasticsearch использует Lucene, что позволяет использовать его функционал с помощью Json и Java API.

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.

## Требования к программному средству

* + 1. Назначение разработки

Функциональным назначением программного средства является предоставление единого места хранения файлов, доступ к которому можно получить с любого устройства, имеющего браузер.

Программное средство может использоваться для создания ассетов, их поиска, получения все информации об нужно ассете, просмотр файлов, хранящихся в ассете.

Конечными пользователями программного средства являются сотрудники компании, которая преобрела данное программное средство и имеют любое устройство позволяющее выходить в сеть интернет.

* + 1. Состав выполняемых функций

Программное средство должно обеспечивать возможность выполнения перечисленных ниже функций:

1. Функция регистрации и аутентификации пользователя;
2. Функция просмотра информации о каждом ассете;
3. Функция создания ассета;
4. Функция удаления ассета;
5. Функция редактирования ассета;
6. Функция добавления новой версии файла в существующий ассет;
7. Функция поиска ассета по имени файла, по значениям его полей или категориям;
8. Функция создания новой категории;
9. Функция создания нового поля метаданных ассета;
10. Функция создания групп пользователей;
11. Функция создания групп полей метаданных;
12. Функция добавления группы полей метаданных категории.
    * 1. Требования к составу и параметрам технических и программных средств

Серверная часть программного средства должна функционировать на ЭВМ со следующими минимальными характеристиками:

* 1. процессор с тактовой частотой 3 ГГц и более;
  2. жесткий диск объемом от 3 Тб;
  3. оперативная память 32 Гб и более;
  4. сетевая карта Ethernet 1 Гбит.

Клиентская часть программного средства должна функционировать на всех устойствах со следующими минимальными характеристиками:

1. процессор с тактовой частотой 1.2 ГГц и более;
2. оперативная память 1 Гб и более.

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

## Диаграмма прецедентов

Дле представления функциональной модели была выбрана диаграмма прецедентов. Диаграмма прецедентов (диаграмма вариантов использования) в UML — диаграмма, отражающая отношения между актёрами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Прецедент — возможность моделируемой системы (часть её функциональности), благодаря которой пользователь может получить конкретный, измеримый и нужный ему результат. Прецедент соответствует отдельному сервису системы, определяет один из вариантов её использования и описывает типичный способ взаимодействия пользователя с системой. Варианты использования обычно применяются для спецификации внешних требований к системе.

Актёр (англ. actor) — стилизованный человечек, обозначающий набор ролей пользователя (понимается в широком смысле: человек, внешняя сущность, класс, другая система), взаимодействующего с некоторой сущностью (системой, подсистемой, классом). Актёры не могут быть связаны друг с другом (за исключением отношений обобщения/наследования).

На рисунке 2.1 представлены варианты использования разрабатываемого программного средства. На диаграмме видно, что в системе будет существовать 3 роли: гость, пользователь и администратор.

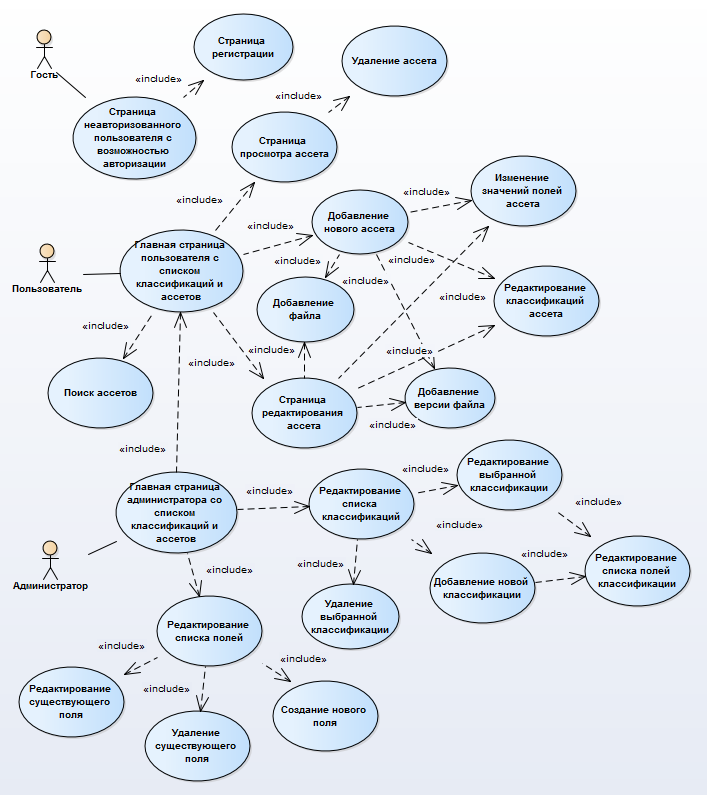


Рисунок 2.1 – Диаграмма прецедентов

* + 1. Гость

Незарегистрированные пользователи имеют только две возможности: авторизоваться или заригистрироваться. Так как в системе храняться файлы предприятия, которые являются интелектуальный собственностью предприятия, доступ к ним должен быть только у тех лиц, которые имеют право на работу с файлами системы.

Регистрация пользователя происходит с помощью специально формы, содержащей имя пользователя, пароль, электронный адрес. После чего заполненная форма отправляется на сервер. Пользователь после регистрации имеет самый низкий уровень доступа (принадлежит группе пользователей «Гость»). Пользователи, находящиеся в данной группе не могут просматривать ассеты, получить список классификаций или полей. Единственное, что доступно таким пользователям, это страница с запросом на добавление роли. Для этого, необходимо ввести имя пользователя, который имеет достаточно прав, для присваения ролей другим пользователям и указать причину добавления новой роли. Данный запрос отправляется адресату и после одобрения, пользователю добавляется новая роль.

Авторизация пользователей происходит путем указания имени и пароля, указынные при регистрации в форму авторизации, которая отправляется на сервер.

* + 1. Пользователь

Пользователям назначаются разные группы (роли) в зависимоти от которых, пользователю доступны или не доступны те или иные операции. Пользователи имеют следующие возможности:

* 1. Просмотр дерева классификаций (некоторые группы могут ограничивать видимость определенных классификаций);
  2. Просмотр списка ассетов, принадлежащих конкретной классификций (при просмотре коренной классификации будут отображаться все ассеты, тех классификаций, которые доступны пользователю);
  3. Просмотр информации о конкретном ассете: будут отображаться поля ассета тех классификаций, которые «видны» пользователю, файлы и системная информация (когда создан, кем создан);
  4. Редактирование ассета: изменение списка классификаций (добавление и удаление), редактирование значений полей, добавление новых файлов или новых версий файлов (если у пользователя есть право надобавление или удаление нужных классификаций);
  5. Удаление ассета (если у пользователя есть право на удаление всех классификаций ассета).
     1. Администратор

Администратор создается при разветывании приложения. После чего, сотрудник, который зашел в систему с помощью имени и пароля администратора получает доступ к всему функционалу приложения. Помимо тех возможностей, которые предоставляются пользователю с самым высоким уровнем доступа (пользователь состоит в группе, которая имеет доступ ко всем классификациям с полным перечнем операций над классификациями) администратор так же имеет следующие возможности:

1. Добавлять нового пользователя;
2. Добавлять пользователя в необходимые группы для расширения прав пользователя;
3. Создавать пользователя с правами администратора;
4. Создавать, удалять и редактировать классификации;
5. Создавать, удалять и редактировать поля;
6. Добавлять поля в группы и привязывать их к классификациям;
7. Создавать, удалять и редактировать группы полей;
8. Создавать удалять и редактировать группы пользователей;
9. Редактировать права групп пользователей;
10. Редактировать набор доступных операций над классификациями ассета;
11. Создавать удалять и редактировать настройки, которые могут быть использованны любой частью системы;
12. Просмотр логов приложения.

## Общие требования к программному средству

Общие требования к разрабатываемому программному средству включают:

1. ПС должно работать на любом устройстве с возможностью выхода в интернет используя браузер;
2. ПС должно позволять сохранять вайлы пользователей в едином месте в виде ассета, к которому могут получить доступ другие пользователи имеющие нужные права;
3. ПС должно позволять осуществолять поиск ассетов по его имени, идентификатору, полям и классификациям;
4. ПС должно позволять изменять поля ассетов и набор классификаций;
5. ПС должно позволять добавлять новые файлы в ассет или новые версии уже существующих файлов;
6. ПС должно позволять удаялять ассеты из системы.

## Требования к составу клиентской части приложения

Клиентская часть программного средства должна содержать следующие представления:

1. Экран авторизации;
2. Экран регистрации;
3. Экран просмотра списка ассетов и дерева классификаций;
4. Экран просмотра информации о ассете;
5. Экран редактирования ассета;
6. Экран настроек (только для администратора);
7. Экран запроса новой группы для пользователя (только для администратора);
8. Экран просмотра логов приложения (только для администратора);
9. Экран добавления классификации (только для администратора);
10. Экран редактирования классификации (только для администратора);
11. Экран добавления поля (только для администратора);
12. Экран добавления группы полей (только для администратора);
13. Экран редактирования поля (только для администратора);
14. Экран редактирования группы полей (только для администратора).

## Спецификация функциональных требований

Разрабатываемое программное средство должно представлять собой веб-приложение управления электронными документами на предприятии. Администратор имеет наибольший перечень возможностей посравнению с обычными пользователями и он обладает возможностью:

1. Создание, просмотр, редактирование и удаление ассетов;
2. Создание, просмотр, редактирование и удаление классификаций;
3. Создание, просмотр, редактирование и удаление полей;
4. Создание, просмотр, редактирование и удаление пользователей;
5. Создание, просмотр, редактирование и удаление настроек системы;
6. Просмотр логов приложения.

# **ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

# Структура программного средства

Программное средство разработанное в рамках данного дипломного проекта будет состоять из модулей. Под модулями понимаются логически независимые части приложения. Подобный подход к построению программного средства имеет ряд преимуществ:

1. Каждый модуль представляется в виде черного ящика со строго определенным интерфейсом, а внутренняя реализация не доступна снаружи. Это позволяет изменять код модуля, сохраняя интерфейс и не нарушить работу других модулей;
2. Модули имеют четко определенную структуру и состав, что позволяет увеличить понимания принципа работы модуля;
3. Модули позволяют держать всю с бизнес-логику какой-то части предметной области в одном месте.

Проектируемое приложение будет состоять из следующих модулей:

1. Веб приложение;
2. Ядро;
3. Модуль работы с PostgreSQL базой данных;
4. Модуль работы с elasticsearch базой данных;
5. Файловое хранилище.

Модуль «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 модуле «ядро» 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.

Модуль работы с PostgreSQL включает в себя реализацию формирования запросов в базу данных PostgreSQL. Данный модуль будет содержать все типы данных, которые сгенерируются с помощью макроса infer\_schema! из библиотеки diesel исходя из схеммы базы данных. Это позволит производить валидацию запростов на этапе компиляции и тем самым увеличить общую производительность приложения.

Модуль работы с базой данных elasticsearch будет содержать все необходимые методы работы с базой. Данный модуль позволит абстрагироваться от функциональности библиотеки rs-es и сделать код понятнее и более гибким. Обертки, которые содержаться в данном модуле не только упрощают работу с elasticsearch но и имеют возможность быть оптимизированными на этапе компиляции в более простое представление, что положительным образом скажется на производительности приложения.

Модуль файлового хранилища позволяет перенести нагрузку по управлению файлами на отдельный сервер. Основной задачей данного модуля будет поддержание файловой структуры таким образом, что бы можно было с помощью REST запроса получить файл по идентификатору ассета, которому этот файл принадлежит и идентификатору файла и его версии. Таким образом, файлы будут храниться в отдельном месте и загружаться только при необходимости. Файловое хранилище будет сохранять путь к файлу в соответствующей колонке таблицы файлов в базе данных, при создании ассета. Таким образом создание ассета и созранение файла в нужной дериктории файлового хранилища можно производить в разных потоках, что позволит увеличить скорость сохранения ассета и уменьшить нагрузку на базу данных, так как файлы будут храниться не в ней.

На рисунке 3.1 представлена схема компонентов модуля «веб-приложения».

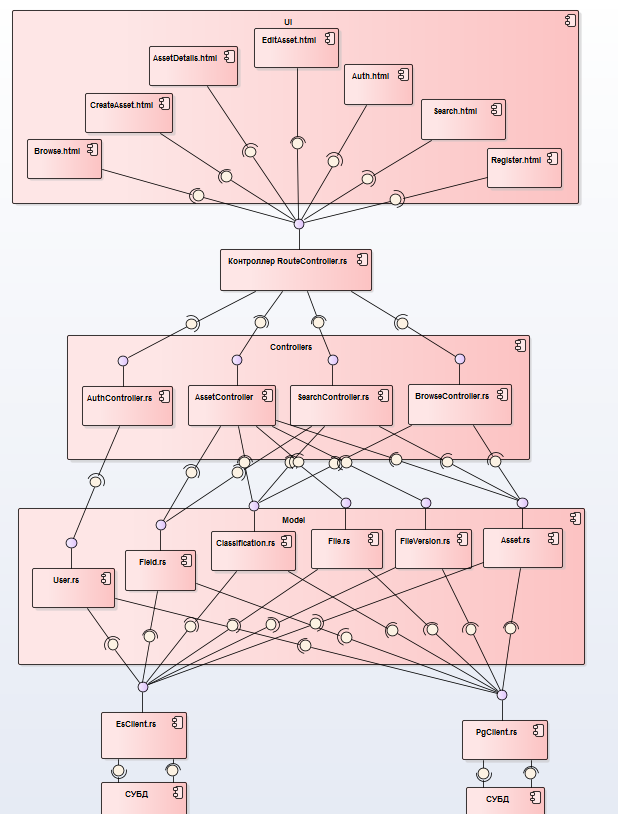


Рисунок 3.1 - Диаграмма компонентов системы

## Развертывание программного средства

Для получения полной картины о функционировании программного средства необходимо рассмотреть, каким образом оно будет развернуто.

Приложение представляет собой несколько артефактов развертывания: рабочая станция пользователя, рабочая станция администратора, веб-сервер, сервер базы данных.

На сервера устанавливается операционная система Debian 8.7. На веб сервер устанавливается веб-интерфейс для работы с приложением. 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 Web.toml файла.

На сервер базы устанавливается две базы: 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.

На рисунке 3.2 можно увидеть схему развертывания приложения.

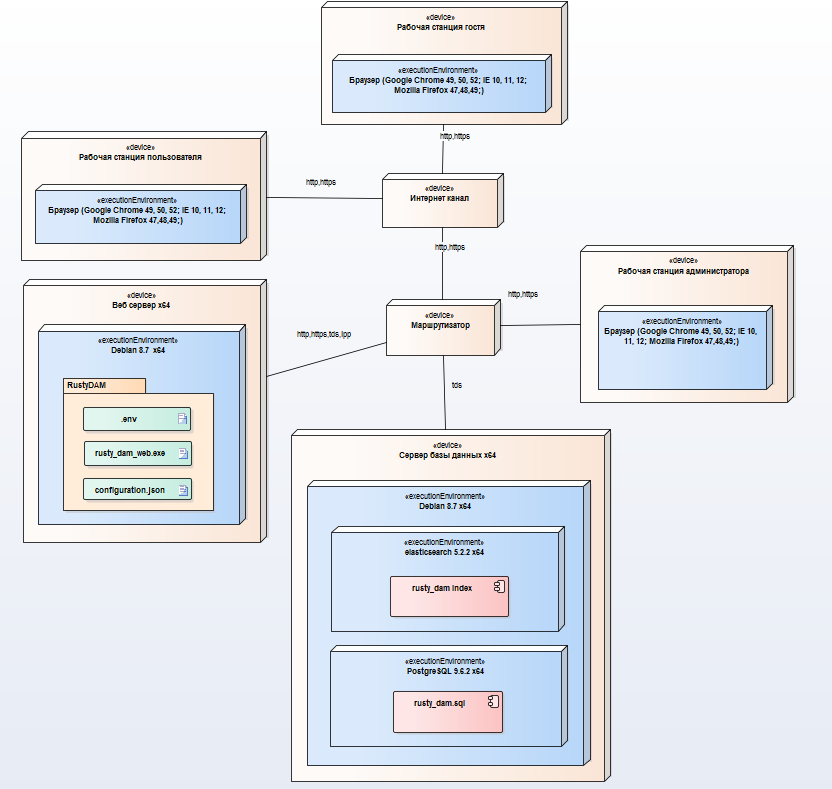


Рисунок 3.2 - Диаграмма развертывания

* 1. Проектирование взаимодействия модулей системы

Данная модель выполена по методологии IDEF0. IDEF0 — методология функционального моделирования (англ. function modeling) и графическая нотация, предназначенная для формализации и описания бизнес-процессов. Отличительной особенностью IDEF0 является её акцент на соподчинённость объектов.

На рисунке 3.3 изображениа декомпозиция контекстной диаграммы и основные процессы, такие как аутентификация, авторизация и регистрация, создание ассета, заполнения его информацией, загрузка файлов.

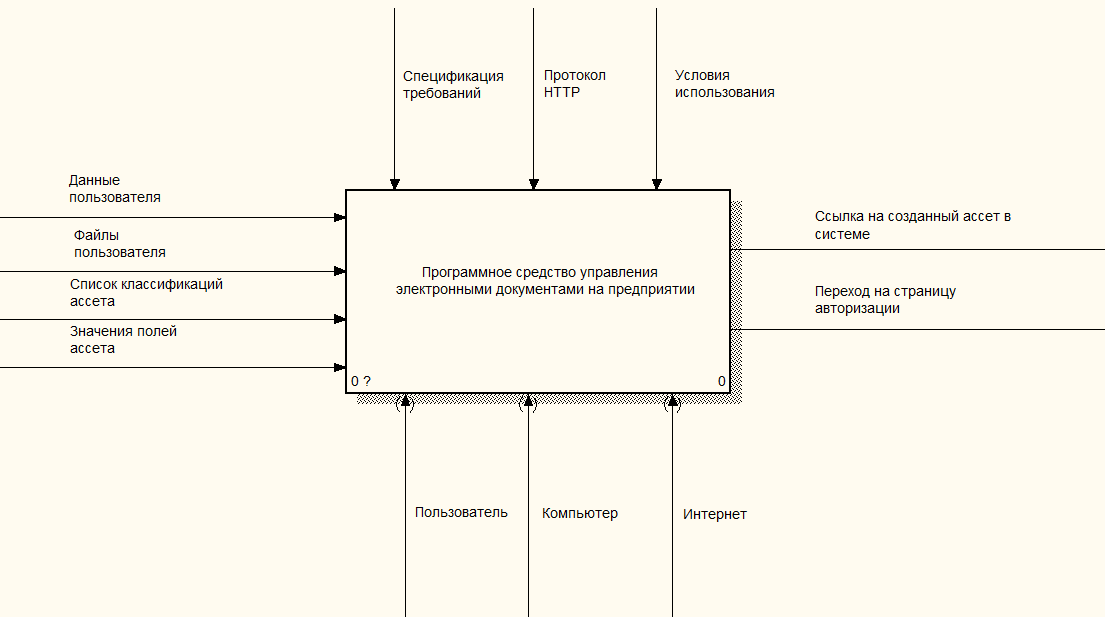


Рисунок 3.3 – Модель программного средства

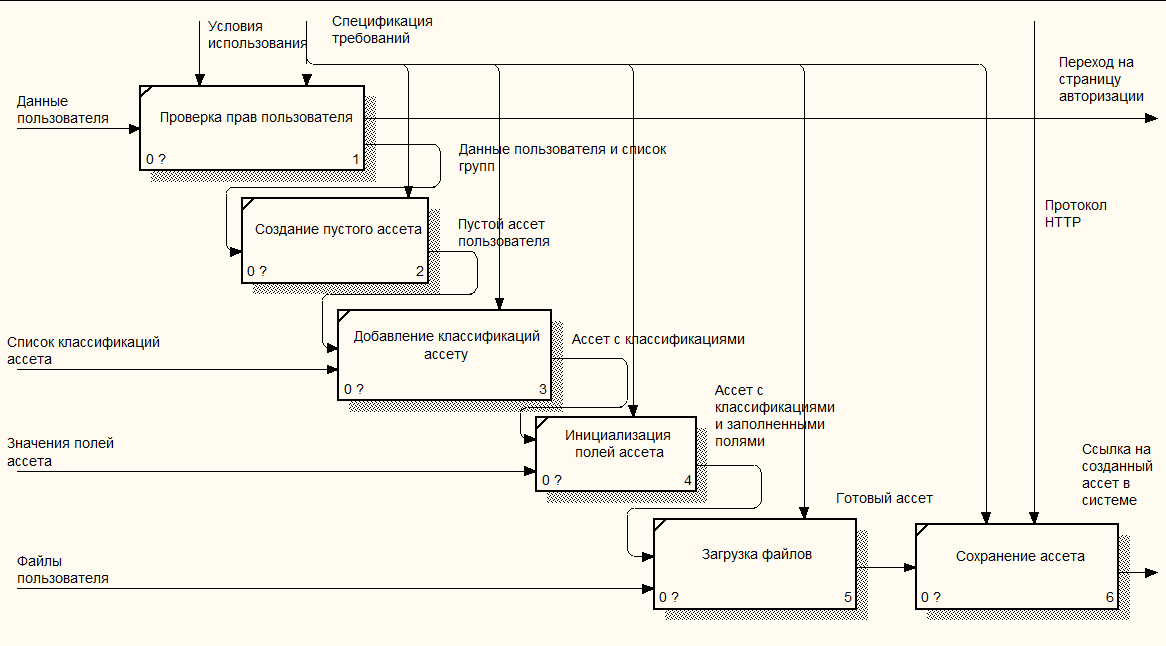


Рисунок 3.4 – Декомпозиция контекстной диаграммы

* 1. Разработка модели базы данных

Основной частью программного продукта является база данных. В данном программном средстве используется две базы данных: PostgreSQL и elasticsearch. Так как elasticsearch это NoSQL база данных, она не имеет схемы, т.е. данные храняться в виде не связанных документов с различным набором полей. На рисунке 3.5 приведена схема базы данных PostgreSQL. Данная база является реляционной и используется таблицы и связи между ними для хранения схемы.

Разработанная схема базы данных проектируемого программного средства соответствует первой, второй и трейтей нормальной форме. На схеме можно увидеть таблицу пользователей, сессий, слассификаций, полей, групп полей, групп пользователей.

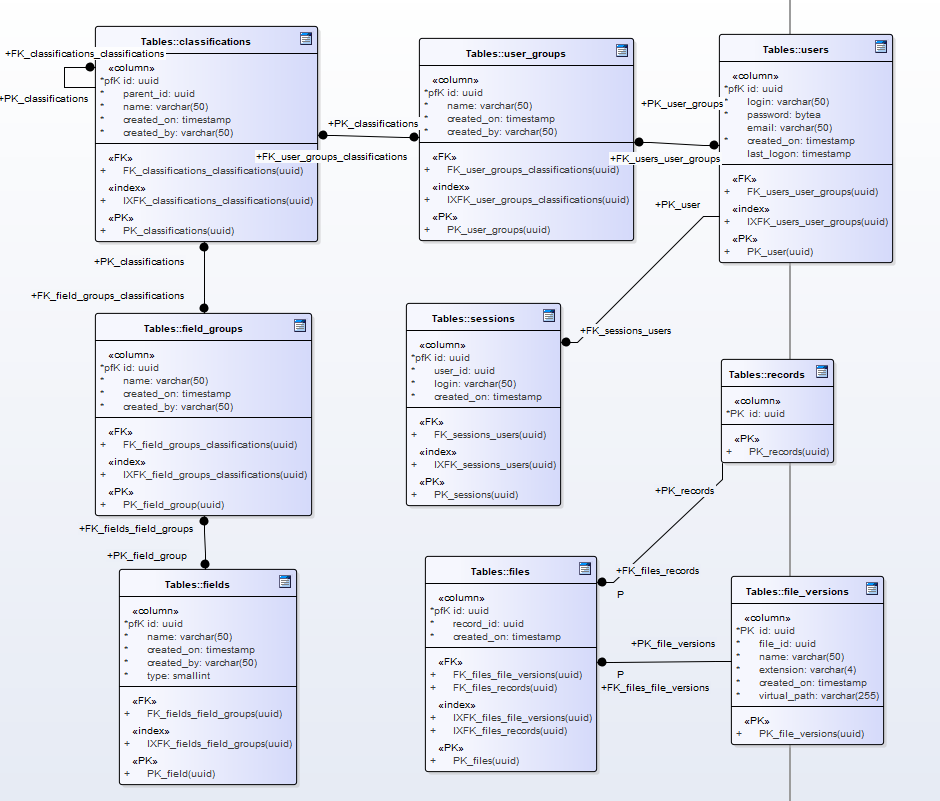


Рисунок 3.5 – Схема базы данных ПС

Таблица classification содержит классификации, к которым пользователь может отнести свои ассеты. Данная таблица соеденена с таблицей field\_groups с помощью связи многие ко многим. Так как данная таблица хранит дерево классификаций, используя рекурсивные запросы можно получить поречень наследников или отца классификации, получить полный путь классификации состоящий из имен всех классификаций, которые расположены на пути к нужно классификации в дереве. При использовании языка PL/pgSQL можно буз труда написать необходимые рекурсивные функции, которые будут выполняться на стороне базы данных, что позволит оптимизировать работу программного средства. Таблица состоит из полей:

1. Id (обязательное поле) - первичный ключ записи;
2. Parent\_id (обязательное поле) – ссылка на родительский узел;
3. Name (обязательное поле) – имя классификации
4. Created\_on (обязательное поле) – дата и время, создания классификации;
5. Created\_by (обязательное поле) – пользователь, создавный классификацию.

Таблица users содердит информацию о зарегистрированных пользователях системы и состоит из полей:

1. Id (обязательное поле) – первичный ключ записи (уникальный идентификатор пользователя);
2. Login (обязательное поле) – имя пользователя;
3. Password (обязательное поле) – sha3 хеш код пароля пользователя (тип данных используется bytea в качестве массива байт хеша);
4. Email (опциональное поле) – e-mail адрес пользователя (поле может быть пустым в случае если пользователь является администратором);
5. Created\_on (обязательное поле) – дата и время регистрации пользователя;
6. Last\_logon (опциональное поле) – дата и время последней авторизации пользователя.

Таблица files содержит информацию о файле ассета. Ассет может хранить несколько файлов, у каждого файла может быть несколько версий. Таблица состоит из следующих полей:

1. Id (обязательное поле) – первичный ключ записи (уникальный идентификатор файла, по которому файловое хранилище будет искать файл в своей структуре папок);
2. Record\_id (обязательное поле) – ссылка на ассет, который содержит данный файл;
3. Created\_on (обязательное поле) – дата и время создания файла.

Таблица file\_versions содержит информацию версии конкретного файла. Имеет следующие поля:

1. Id (обязательное поле) – первичный ключ версии (будет использоваться файловым хранилищем для поиска нужной версии файла в папке самого файла);
2. File\_id (обязательное поле) – ссылка на файл ассета;
3. Name (обязательное поле) – имя версии файла с расширением;
4. Extension (обязательное поле) – поле, хранящее расширение файла (может использоваться для фильтрации или поиска ассетов с конкретным расширением файла);
5. Created\_on (обязательное поле) – время загрузки новой версии файла;
6. Virtual\_path (обязательное поле) – полный путь к файлу внутри файлового хранилища (данный путь будеть преобразовываться в локальный путь к файлу в системе пользователя, в момент загрузки файла).

Таблица fields содержит информацию о полях, которые могут быть применены на ассет. Таблица состоит из следующих полей:

1. Id (обязательное поле) – первичный ключ записи;
2. Name (обязательное поле) – название поля;
3. Created\_on (обязательное поле) – дата и время создания поля;
4. Created\_by (обязательное поле) – пользователь, создавший поле;
5. Type (обязательное поле) – содержит индекс типа данных, которые могут храниться в поле.

Таблица field\_groups позволяет объединять поля в группы, это позволяет ассоциировать большое количество полей для классификации. Таблица состоит из полей:

1. Id (обязательное поле) – первичный ключ записи;
2. Name (обязательное поле) – название группы полей (используется при поиске группы);
3. Created\_on (обязательное поле) – дата и время создания группы полей;
4. Created\_by (обязательное поле) – пользователь, создавший данную группу полей.

Таблица sessions хранить список активных подключений к программному средству. Используется в веб-приложении для получения соединения без авторизации логином и паролем при получении каждого запроса. Так как во время запроса пользователь не авторизуется заного, скорость обработки запроса возрастает. Идентификатор сессии храниться в куках пользователя вместо с логином, что уменьшает вероятность авторизоваться, подобрав идентификатор сессии. Таблица состоит из полей:

1. id (обязательное поле) – уникальный идентификатор сессии (первичный ключ);
2. user\_id (обязательное поле) – идентификатор пользователя (используется для ускорения поиска пользователя, при необходимости);
3. login (обязательное поле) – используется для увеличения безопастности работы с сессиями;
4. created\_on (обязательное поле) – дата и время создания сессии (позволяет отслеживать количество времени, которое пользователь подключен к приложению).

Таблица records хранит только идентификатор ассета, так как вся необходимая информация храниться в elasticsearch. В базе PostgreSQL данная таблица нужна для построения сложных запросов, в которых необходимо знать перечень всех ассетов в базе.

Таблица user\_groups соединяется с таблицей классификаций связью многие ко многим. Промежуточная таблица, используемая для этой связи, будет так же содержать права, группы на классификацию (чтение, запись, ассоциирование ассета с классификацией). Таблица user\_groups содержит поля:

1. Id (обязательное поле) – уникальный идентификатор группы (первичный ключ);
2. Name (обязательное поле) – имя группы пользователей (используется при поиске группы для добавления в нее пользователей);
3. Created\_on (обязательное поле) – дата и время создания группы;
4. Created\_by (обязательное поле) – пользователь, который создал группу.

При развертывании приложения создается группа «Everyone» в нее добавляются все новые пользователи. Данная группа обладает самым низким уровнем доступа к приложению.

1. **ПРОЕКТИРОВАНИЕ 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 компонентов:

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. Основной программный модуль**

Основным программным модулем является libcore. 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. При сохранении ассета вызывается index метод у клиента, который формирует POST запрос в теле которого, содержится сериализованный в формате Json, ассет. Так как документы в elasticsearch не изменяемые, то при обновлении ассета нужно выполнить сначала удаление старого документа, а потом индексировать новый документ (в будущем возможно изменение способа обновления ассета). У ассета есть метод classify\_as, он позваляет классифицировать ассет в одну из существующих классификаций. Аргументом фунции может быть любой тип, который реализует трейт IntoEntity, что позволяет использовать разные способы указания классификации, к которой нужно отнести ассет. Когда ассет ассоциируют с классификацией, он получает набор полей, которые имеет классификация. После чего, каждому полю можно установить значение. Значение полей храняться в elasticsearch базе, и могут участвовать в поиске ассета по значению полей. Это позволяет упростить поиск нужного ассета в системе.

Для создания веб-приложений использовался фреймворк Rocket. Для языка программирования Rust написано несколько веб-фреймворков, самыми известными являются: iron-rs, nickel-rs и rocket-rs. Rocket был выбран по причине его простоты, надежности и гибкости. Это позволит уменьшить расходы на разработку и уменьшить время тестирования, что позволит уменьшить стоймость разработки проекта.

В составе с Rocket идет две библиотеки для формирования html шаблонов: handlebars и tera. Разница между ними не существенна, поэтому был выбран tera.

Шаблоны страниц распологаются в папке templates и именуются с помощью двух расширений: первое расширение является .html, второе - .tera. Таким образом Rocket получает информацию о том, что использовать handlebars или tera для формировании страницы. Модель, используемая для генерации страницы валидируется на этапе компиляции, это позволяет убедиться, что используется правильная модели и что в шаблоне не используются поля, которых у модели нет.

Стили страниц описаны в .css файлах, которые распологаются в styles папке. В качестве основных стилей использовался bootstrap. JavaScript файлы, содержащие логику работы клиентской части приложения, хранятся в папке scripts. Весь клиентский код базируется на использовании JQuery версии 3.2.1. Это последняя версия JQuery на момент написания проекта.

Авторизация пользователя происходит путем установления куки в браузер пользователя, они содержат идентификатор сессий и логин пользователя. При авторизации на странице Login, с пользователем ассоциируется сессия. Каждый раз, когда пользователь заходит на страницу, происходит валидация идентификатора сессии и выполняется проверка того, что сессия ассоциирована с текущим пользователем используя его логин. При посещении любой страницы, кроме авторизации и регистрации выполняется формарование APIKey структуры, которая содержит в себе App структуру, в случае успеха. Если у пользователя не верный идентификатор сессии или логин не совпадает с сессией, в этом случае пользователь перенаправляется на страницу авторизации. В случае успеха, на вход обработчику запроса приходит APIKey c App структурой, которую можно использовать для для загрузки ассетов, классификаций, полей и т.д. Установка соединения происходит вызовом метода connect\_to\_session, которая принимает идентификатор сесии и логин. Таким образом осуществляется защита данных от несанкционированного доступа.

В процессе разработки была использована технология Docker. Docker позволяет разделить среду в которой разрабатывается приложение и средой, в которой установлены базы данных и файловые сервера, при этом они находятся на одном компьютере. Благодаря функциям виртуализации Hyper-V, есть возможность создавать «контейнеры». Контейнер является мини операционной системой, которая сконфигурирована под определенные задачи. Контейнеры могут быть скомпилированны в пакет, который можно разместить в docker.hub репозитории и при необходимости скачивать и использовать без каких либо дополнительных настроек.

При разработке проекта было сконфигурированно три контейнера: один содержал развернутую базу данных PostgreSQL (образ контейнера на docker.hub называется postgres, это официальный контейнер базы данных PostgreSQL), второй контейнер содержит elasticsearch базу (название образа на docker.hub – elasticsearch; содержит в себе версию Java, подходящую по версии к установленному elasticsearch), последний контейнер содержал файловый сервис, написанный в рамках дипломного проекта. Файловый сервис размещается в отдельном контейнере для тестирования скорости работы с файлами (загрузка и выгрузка).

# **ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

Проведено тестирования программного средства. Целью данного испытания была проверка его работоспособности.

Таблица 5.1 – Набор тест-кейсов модуля «Авторизации» и «Регистрации»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № тест-кейса | Тестируемая функциональность | Последовательность действий | Ожидаемый результат | Полученный результат |
| 1 | Авторизация. Валидация пользовательского ввода | * + - 1. Перейти на сайт приложения.       2. Нажать кнопку «Login» | Сообщение под полем ввода логина «Enter user name» | Тест успешно пройден |
| 2 | Авторизация. Валидация пользовательского ввода | 1. Перейти на сайт приложения. 2. Ввести имя пользователя 3. Нажать кнопку «Login» | 1. Сообщение под полем ввода пароля «Enter user password» | Тест успешно пройден |
| 3 | Регистрация. Валидация пользовательского ввода | 1. Перейти на сайт приложения. 2. Ввести имя пользователя 3. Ввести не более 5 символов в поле ввода пароля 4. Нажать кнопку «Register» | 1. Сообщение под полем ввода пароля   «Password too short. Minimum password length must be 5 characters» | Тест успешно пройден |
| 4 | Регистрация. Валидация пользовательского ввода | 1. Перейти на сайт приложения. 2. Ввести имя пользователя 3. Ввести не более 5 символов в поле ввода пароля   Нажать кнопку «Register» |  |  |

# **РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ**

Программное средство состоит серверной части и веб-приложения. Для работы веб-прилежения достаточно установить на устройство браузер и установить интернет соединение. Серверная часть программного средства состоит из трех компонентов, база данных, веб-приложение и файловое хранилище. Данные компоненты могут быть установлены на разные компьютеры, виртуальные машины или размещаться на одном сервере. Для того, что бы понять, нужно ли разделять компоненты по разным машинам, необходимо знать, с каким объемом данных будет работать приложение. Если объем данных будет большим, в этом случае целесообразно будет разделить компоненты.

В самом начале резвертывания приложения необходимо установить компилятор языка Rust, для этого нужно перейти на оффициальную страницу [?] проекта rustup. Данный проект является менеджером версий компилятора Rust и позволяет устанавливать разные версии компиляторов, для разных платформ и многое другое. Для конкретной операционной среды будет предложен инсталятор. После установки rustup необходимо установить подходящую версию компилятора. Для этого нужно выполнить команду

rustup default stable

Данная команда выполнит установку стабильной версии компилятора, которая содержит только протестированные изменения новых версий. Основной модуль программного средства написан используя стабильную версию компилятора, это означает что при выходе новой версии компилятора, код, компилирующийся на старой версии, будет так же успешно компилироваться и в новой версии. Далее нужно установить nightly версию компилятора. Это необходимо для сервера с веб-приложением и файловым хранилищем, так как они используют возможности компилятора, которые на текущий момент не достаточно протестированны, однако вероятность вызникновения ошибок всвязи с этим мала. Установка производиться командой

rustup toolchain install nightly

Для Windows операционной системы, необходимо выбрать какую сборочную систему должен использовать компилятор. Для Windows доступно две версии: GNU и MSVC. Для того, что бы использовать GNU версию компилятора, необходимо перейти на сайт [?] и скачать последнюю версию. После установки нужно указать в переменной окружение PATH путь к папке, в которую производилась установка, для того, что бы компилятор мог найти необходимые библиотеки и линкер. MSVC версия используется в качестве стандартной версии для Windows, но она требует либо установленной Visual Studio 2015, либо Visual Studio 2015 Build Tools (содержит только библиотеки и необходимые программы для создания бинарного файла на этапе компиляции). Build Tools имеет значительно меньший размер, по сравнению с Visual Studio 2015. Для установки Build Tools необходимо перейти на страницу [?] и нажать на кнопку Download Visual C++ Build Tools 2015. На момент написания дипломного проекта доступна так же 2017 версия Build Tools, но она не поддерживается Rust компилятором. После установки Build Tools 2015 нужно проверить что все установлено правильно, для этого нужно выполнить несколько комманд:

rustc –version

cargo –version

cargo new –bin console

cd .\console\

cargo build

Данные команды выводят версии компилятора и проектного менеджера (rustc, Cargo), после чего создают консольное приложение под названием «console» и выполняется его сборка. Если все команды выполнились без ошибок – установка прошла успешно.

Следующий шаг – установка PostgreSQL базы данных. Для начала, нужно выбрать сервер, на котором производить установку. Далее на официальной странице PostgreSQL [?] необходимо скачать версию 9.6.3. Используя данную версию базы производилась разработка продукта, более поздние версии не тестировались. После установки базы данных нужно установить драйвер для соединения с базой на все сервера, на которых будет установлен программный продукт. Драйвер находится на официальной странице PostgreSQL ODBC driver [?]. После установки драйвера, необходимо установить переменную среды LIB\_PQ\_DIR, значение которой будет путь к папке с драйвером (libpq.dll в случае Windows). Такой же путь нужно добавить в переменную окружения PATH. Строку подключения необходимо указать в настройках веб-приложения и файлового хранилища. Для веб-приложения файл настроек называется Web.toml, а для файлового хранилища – FileStore.toml. В данных конфигурационных файлах в секции databases нужно указать строку с ключем postgres таким образом:  
 ...

[databases]

postgres = “…”

В PostgreSQL базе данных нужно создать и настроить пользователя, после чего его имя и пароль указать в строке подключения.

После того, как PostgreSQL база установлена, необходимо установиль elasticsearch. Для того, что бы установить elasticsearch необходимо перейти на официальный сайт [?] и скачать версию 5.2. Почле чего, нужно скачать и установить Java JDK последней версии [?]. Путь к папке, в которую был установлен Java JDK нужно установить переменной окружения JAVA\_HOME. Далее необходимо установить размер кучи для JVM размером в 4 гигабайта. В случае с Windows необходимо перейти в панель управления, нажать на иконку Java, после чего, в появившемся окне перейти на вкладку Java и нажать на кнопку View …. В появившемся окне, в столбце Runtime Parameters нужно написать:

-Xms4g -Xmx4g

В случае UNIX операционных систем, необходимо установить такое же значение для переменной среды \_JAVA\_OPTIONS. Это увеличит размер доступной памяти для виртуальной машины Java, что позволит ускорить ее работу и, как следствие, ускорить поиск документов. После того, как Java установлена и настроена, нужно настроить запуск сервиса elasticsearch. Для Windows это происходит путем запуска команды service.bat из папки, в которую был распакован архив elasticsearch, с параметром install:

service.bat install

В случае с UNIX необходимо перейти в папку elasticsearch и выполнить команды:

sudo update-rc.d elasticsearch defaults 95 10

sudo /etc/init.d/elasticsearch start

Эти команды установят демона elasticsearch в сисему.

После установки elasticsearch и PostgreSQL нужно открыть порты, которые прослушиваются данными базами для подключения извне, в случае, если базы находятся на отдельных серверах.

# **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>

[?] <https://www.rustup.rs/>

[?] <http://www.mingw.org/>

[?] <http://landinghub.visualstudio.com/visual-cpp-build-tools>

[?] <https://www.postgresql.org/>

[?] <https://odbc.postgresql.org/>

[?] <https://www.elastic.co/>

[?] <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>