

АННОТАЦИЯ

В настоящей дипломной работе выполнена разработка репозитория инсталляционных пакетов корпоративной сети, основными функциями которого являются:

- централизованное структурированное хранение инсталляционных пакетов;
- учет версий инсталляционных пакетов;
- выгрузка инсталляционных пакетов;
- регистрация заявок от пользователей на добавление инсталляционных пакетов.

В рамках дипломной работы выполнены: анализ аналогов; анализ видов архитектур; выбор архитектуры для разработанного приложения; спроектированы и разработаны база данных и классы, для работы с ней; разработаны алгоритмы, реализующие требуемый функционал; разработан пользовательский интерфейс; проведено тестирование. **ВНЕДРЕНО?**

ANNOTATION

Repository of installation package of corporate network is developed in this graduation project. This application provides the following functions:

- centralized and structured storage of installation packages;
- installation packages version control;
- downloading of installation packages;
- Users request registration for add new installation packages.

In this project, the following problems are solved: analysis of similar programs; analysis of application architectures; choosing the architecture for project development; designed and developed a database and classes; developed algorithms that implement the required functionality; developed graphical user interface; performed components testing.

РЕФЕРАТ

Записка ??? с., ?? рис., ?? табл., ?? источник

РЕПОЗИТОРИЙ, ИНСТАЛЛЯЦИОННЫЙ, ПАКЕТ, ВЕРСИЯ, КАТЕГОРИЯ, СВОЙСТВО.

Объектом разработки является репозиторий инсталляционных пакетов корпоративной сети.

Цель работы – разработка репозитория инсталляционных пакетов корпоративной сети, основное назначение которого заключается в:

- централизованном и структурированном хранении инсталляционных пакетов;
- учет версий инсталляционных пакетов;
- выгрузке инсталляционных пакетов;
- регистрации заявок от пользователей на добавление инсталляционных пакетов.

В рамках выполненных работ решены следующие задачи:

- анализ аналогичного программного обеспечения;
- анализ видов архитектур распределенных систем;
- анализ предметной области;
- проектирование и разработка базы данных;
- проектирование и разработка классов;
- разработка алгоритмов, реализующих требуемый от системы функционал;
- разработка пользовательского интерфейса;
- сборка приложения;
- тестирование отдельных компонентов и всего приложения;
- внедрение?

Результатом дипломной работы является репозиторий инсталляционных пакетов корпоративной сети в виде исходных текстов программы и ее дистрибутивной версии.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ.....	6
1 Анализ аналогов, анализ видов архитектур и выбор архитектуры для разрабатываемой системы	8
1.1 Анализ аналогов и определение требований к разрабатываемой системе	8
1.2 Анализ и выбор архитектуры разрабатываемой системы.....	13
1.2.1 Архитектура «файл-сервер»	13
1.2.2 Архитектура «клиент-сервер»	16
1.2.2.1 Двухзвенная архитектура «клиент-сервер»	18
1.2.2.2 Трехзвенная архитектура «клиент-сервер»	21
1.2.3 Выбор архитектуры разрабатываемого приложения	23
1.3 Вывод.....	24
2 Разработка репозитория инсталляционных пакетов	25
2.1 Анализ предметной области и уточнение спецификации.....	25
2.1.1 Уточнение требований к системе	25
2.1.2 Выбор способа проектирования	28
2.1.3 Выбор программной платформы.....	28
2.2 Разработка структуры программы и ее компонентов.....	33
2.2.1 Разработка структуры программы	33
2.2.2 Разработка базы данных и классов.....	34
2.2.3 Разработка алгоритмов.....	50
2.2.4 Разработка пользовательского интерфейса	56
2.3 Компоновка репозитория инсталляционных пакетов	62
2.4 Выводы	63
3 Тестирование репозитория инсталляционных пакетов.....	64
3.1 Тестирование методом ручного контроля.....	66
3.1.1 Тестирование методом инспекции исходного текста.....	66
3.1.2 Тестирование методом сквозных просмотров	66
3.2 Тестирование по принципу «белого ящика»	69
3.3 Тестирование по принципу «черного ящика»	73
3.4 Оценочное тестирование.....	76
3.5 Выводы	78
4 Технико-экономическое обоснование разработки.....	79
4.1 Расчет трудоемкости проекта.....	79
4.2 Определение численности исполнителей	81
4.3 Построение сетевого графика выполнения проекта	83
4.4 Построение календарного графика выполнения проекта.....	88

4.5	Расчет затрат на разработку проекта.....	90
4.6	Вывод.....	95
ЗАКЛЮЧЕНИЕ		96
В результате выполнения дипломной работы был разработан репозиторий инсталляционных пакетов корпоративной сети		96
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ		97
Приложение А	Техническое задание	99
Приложение Б	Графические материалы.....	100
Приложение В	Алгоритм добавления нового инсталляционного пакета	101

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ИС	—	информационная система
СУБД	—	система управления базами данных
БД	—	база данных
ПО	—	программное обеспечение
ТЗ	—	техническое задание
ОС	—	операционная система
ЯП	—	язык программирования
РФ	—	Российская Федерация
ПЭВМ	—	персональная электронно-вычислительная машина
ВДТ	—	видеодисплейный терминал
ЭКГ	—	электрокардиограмма
ПДК	—	предельно допустимая концентрация
КЕО	—	коэффициент естественной освещённости
ЛБ	—	
КЛЛ	—	компактная люминесцентная лампа
ЛПО	—	
ЛДЦ	—	
ПДУ	—	предельно допустимый уровень
ЭМП	—	электромагнитное поле
ВДУ	—	

ВВЕДЕНИЕ

На АО "ВПК "НПО машиностроения" используется большое количество разнообразного программного обеспечения. В настоящее время для хранения инсталляционных пакетов работники АО "ВПК "НПО машиностроения" используют свои рабочие компьютеры. Такой способ хранения приводит к тому, что одни и те же файлы хранятся на разных компьютерах, занимают место на диске. К тому же, на одном компьютере файлы могут дублироваться. Для поиска необходимого инсталляционного пакета и передачи его другому работнику может потребоваться много времени. Для того, чтобы решить данную проблему возникла необходимость в разработке репозитория инсталляционных пакетов корпоративной сети (далее, репозиторий инсталляционных пакетов), который предназначен для централизованного хранения и упорядочивания инсталляционных пакетов.

1 Анализ аналогов, анализ видов архитектур и выбор архитектуры для разрабатываемой системы

1.1 Анализ аналогов и определение требований к разрабатываемой системе

Перед тем, как приступить к проектированию, необходимо проанализировать возможности аналогичных систем и определить требования для разрабатываемой системы.

В интернете есть большое количество разнообразных программных продуктов способных в той или иной степени реализовать часть возможностей, которые необходимы от разрабатываемой системы. Наиболее известными, среди таких программ, являются Simple File Manager, DocMGR, Xincos DMS и Alfresco. Они были рассмотрены как ближайшие аналоги разрабатываемой системы.

Simple File Manager – простейший файловый менеджер. Это приложение позволяет:

- загружать файлы на сервер;
- выгружать файлы с сервера;
- удалять файлы на сервере;
- создавать категории для группировки файлов.

Пользовательский интерфейс Simple File Manager представлен на рисунке 1.1.

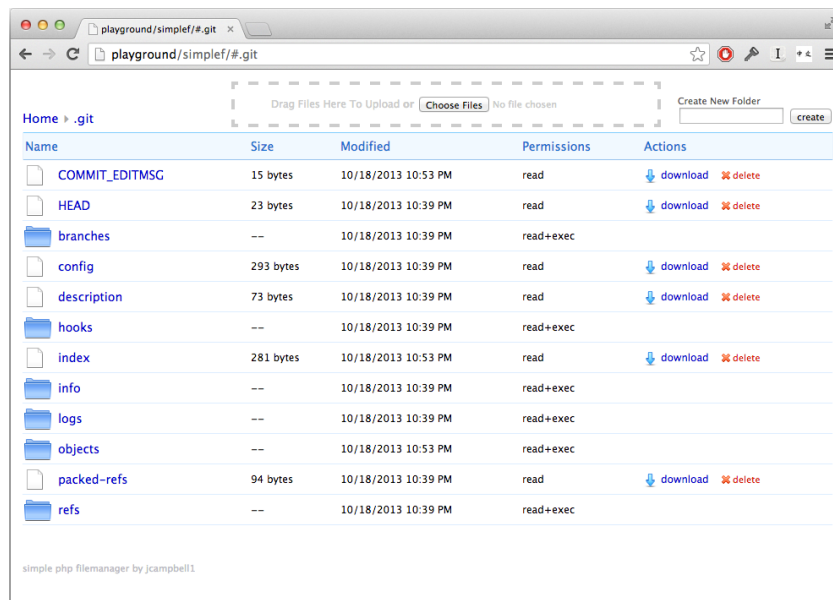


Рисунок 1.1 – Пользовательский интерфейс приложения Simple File Manager

Из достоинств данного приложения можно выделить следующие:

- программный продукт находится в свободном доступе и является проектом с открытым исходным кодом;
- простой пользовательский интерфейс;
- маленький размер исходных файлов.

Среди недостатков можно отметить отсутствие аутентификации и разграничения доступа, нету учета версий файлов, отсутствует перевод пользовательского интерфейса на русский язык.

Система управления файлами DocMGR позволяет хранить любые типы файлов. Поддерживается аутентификация с помощью LDAP, создание и удаление категорий для группировки файлов, ведется учет версий файлов. Пользовательский интерфейс DocMGR представлен на рисунке 1.2.

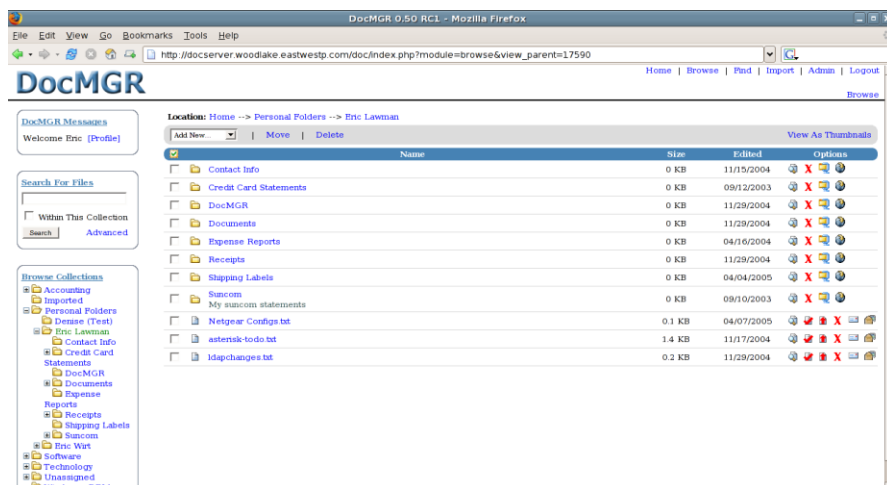


Рисунок 1.2 – Пользовательский интерфейс DocMGR

Достоинства программы:

- поддержка аутентификация с помощью LDAP;
- имеется предварительный просмотр файлов;
- поддержка полнотекстового поиска для файлов текстового формата.

Недостатки приложения: неудобная настройка параметров, нет перевода пользовательского интерфейса на русский язык.

Система управления документами Xincso DMS для работы с файлами, веб-адресами и контактами. В ней ведется учет версий файлов, есть поддержка нескольких языков. Пользовательский интерфейс Xincso DMS представлен на рисунке 1.3.

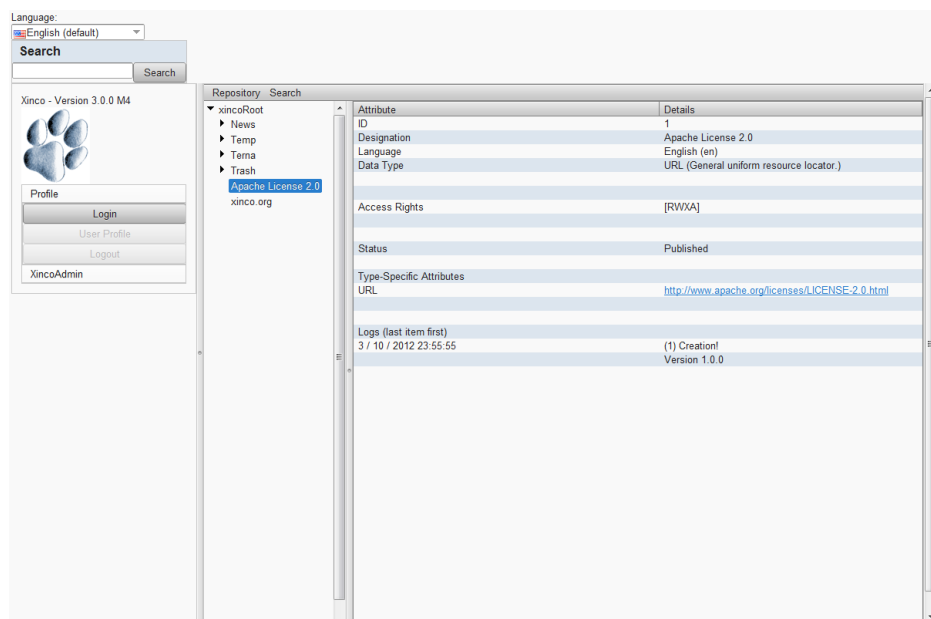


Рисунок 1.3 – Пользовательский интерфейс Xincos DMS

Из достоинства Xincos DMS можно отметить следующие:

- простой интерфейс;
- поддержка полнотекстового поиска;
- поддержка работы с веб-адресами и контактами.

К недостаткам данной системы относятся отсутствие поддержки протокола LDAP, нет разграничения прав для пользователей.

Alfresco – это система управления контентом с открытым исходным кодом. Она предоставляет широкие возможности: позволяет создавать категории для группировки файлов, искать по содержимому документов, ведет учет версий файлов, предоставляет возможность предварительного просмотра файлов. Пользовательский интерфейс Alfresco представлен на рисунке 1.4.

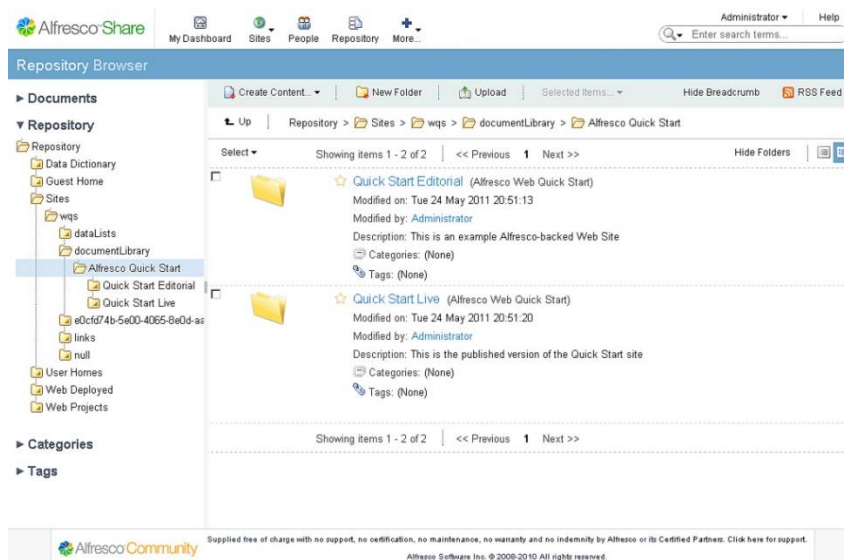


Рисунок 1.4 – Пользовательский интерфейс Alfresco

К достоинствам Alfresco можно отнести:

- предоставляет возможность организовать аутентификацию по протоколу LDAP;
- позволяет конвертировать одни типы документов в другие с помощью встроенного Open Office;
- есть коммерческая версия, которая включает в себя техническую поддержку.

Из недостатков можно отметить то, что некоторые функции недоступны до покупки коммерческой версии.

В результате анализа приведенных выше продуктов видно, что все они имеют недостатки: некоторые из систем больше ориентированы на работу с документами, чем с другими типами файлов, некоторые не поддерживают протокол LDAP. Так же, все рассмотренные системы предоставляют ограниченный набор свойств для хранящихся файлов. Анализ аналогов позволил выделить следующие требования к разрабатываемой системе:

- для пользователей системы необходимо предоставить набор стандартных свойств файлов, а также, необходимо организовать возможность добавления и заполнения новых свойств;

- для удобной работы пользователей с существующими категориями необходимо отображать список категорий в виде дерева.

1.2 Анализ и выбор архитектуры разрабатываемой системы

Архитектура информационной системы – концепция, определяющая модель, структуру, выполняемые функции и взаимосвязь компонентов.

Так как была поставлена задача создания централизованного хранилища инсталляционных пакетов, то разрабатываемая программная система будет содержать файлы, к которым необходимо обеспечить совместный доступ пользователей.

По степени распределенности информационные системы делятся на:

- Настольные (локальные), в которых все компоненты работают на одном компьютере;
- Распределенные, в которых компоненты распределены по нескольким компьютерам.

Для организации совместного доступа пользователей к данным подходят распределенные информационные системы. Они, в свою очередь, делятся на:

- ИС с архитектурой «файл-сервер»;
- ИС с архитектурой «клиент-сервер».

Необходимо рассмотреть перечисленные способы построения информационных систем в качестве архитектуры для разрабатываемого приложения.

1.2.1 Архитектура «файл-сервер»

Данная архитектура представляет собой наиболее простой случай архитектуры с совместным использованием файлов. Она состоит из двух компонентов:

- Файловый сервер;
- Клиентское программное обеспечение: персональный компьютер с клиентской частью приложения и СУБД.

Приложения, основанные на такой архитектуре схожи с локальными приложениями и используют сетевой ресурс для хранения данных. Функции сервера ограничиваются только хранением данных, он является пассивным источником. Вся работа по получению, обработке, а также, по поддержанию целостности и актуальности данных происходит в клиентском приложении, которое запущено на рабочих станциях. Это клиентское приложение совмещает в себе компонент представления, прикладной компонент и СУБД. Структура архитектуры «файл-сервер» представлена на рисунке 1.5.

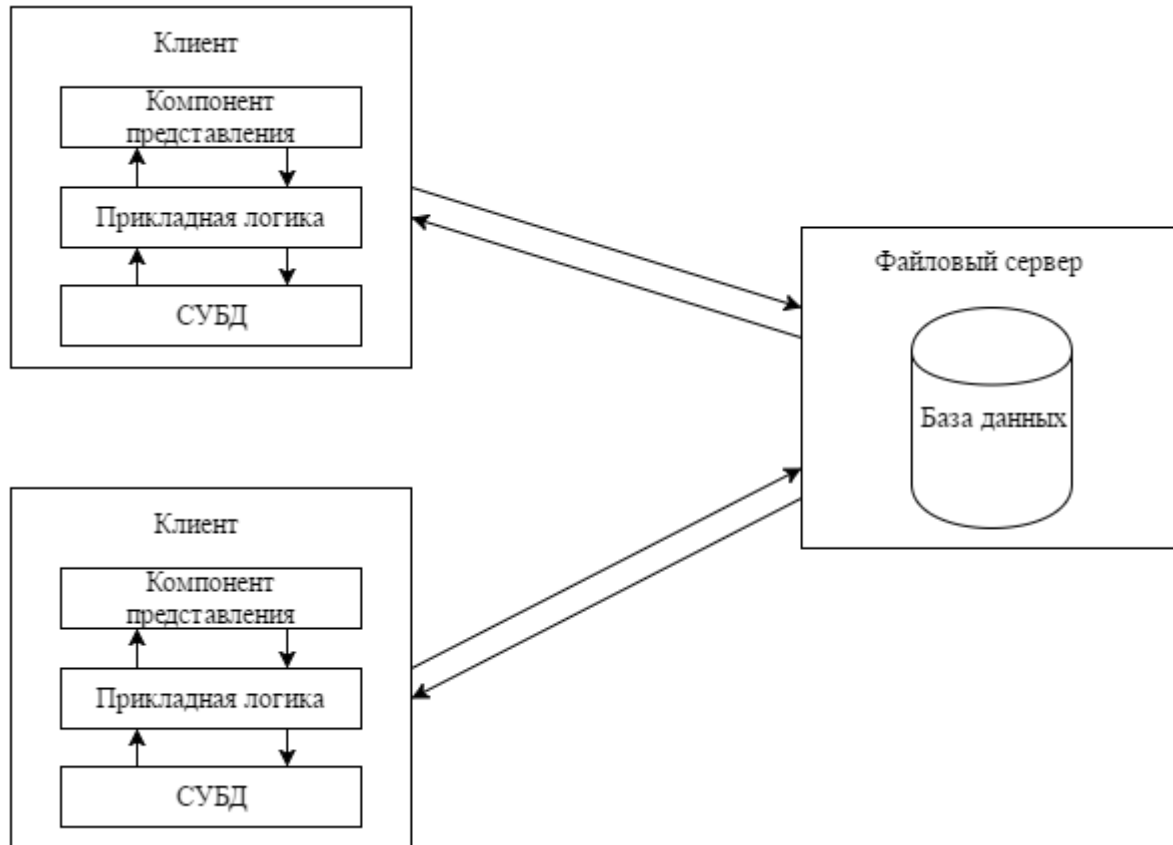


Рисунок 1.5 – Структура архитектуры «файл-сервер»

Во время работы приложения, для каждого клиента создается локальная копия базы данных. При каждом запросе клиента данных в его локальной копии базы данных полностью обновляются из базы данных сервера. Даже если запрос относится всего к одной записи, обновляются все записи базы данных. Следствием такого принципа работы является то, что по сети передается большое количество избыточной информации, что приводит к большим нагрузкам на сеть, снижению ее производительности и снижению производительности информационной системы в целом. Это приводит к тому, что приложения с таким подходом подходят только для работы с небольшим объемом данных и с небольшим количеством пользователей. Ограничение на количество пользователей так же накладывает то, что несколько пользователей не могут одновременно иметь доступ на запись к одному файлу. Однако они могут работать с такими файлами, если они обращаются к ним исключительно в режиме чтения.

Так как все данные обрабатываются на компьютере пользователя, то это влечет повышенные требования к пользовательскому аппаратному обеспечению. Соответственно, чем больше у данной системы пользователей, тем дороже будет оснащение компьютеров.

Еще одним моментом, который нужно учитывать при разработке приложений с файл-серверной архитектурой это то, что вся ответственность за сохранность и целостность базы данных лежит на клиентском приложении.

Применительно к поставленной задаче можно использовать данный подход следующим образом: сервер будет использоваться в качестве хранилища файлов базы данных и хранилища файлов, которые будут загружаться пользователями. В клиентскую часть выносятся работа с базой данных, вся логика приложения по обработке данных и по их представлению пользователю.

Главным преимуществом использования данного подхода при создании необходимой информационной системы является высокая скорость и низкая стоимость разработки.

Однако, этот подход обладает большим количеством недостатков. Рассмотренная архитектура не предполагает большого количества пользователей. Использование такой модели большим количеством человек приведет к огромной нагрузке на сеть, повышенным требованиям к пропускной способности, в результате чего, работа с системой будет практически невозможна.

К другим недостаткам такой системы можно отнести трудность обеспечения непротиворечивости и целостности данных и трудность обеспечения безопасности, так как для предоставления возможности работы в системе каждому пользователю необходимо дать полный доступ, а данные для доступа к серверу будут храниться на стороне пользователя.

1.2.2 Архитектура «клиент-сервер»

Ключевым отличием архитектуры клиент-сервер от архитектуры файл-сервер является абстрагирование от внутреннего представления данных. Основная идея данной архитектуры заключается в распределении прикладной программы по двум компонентам: клиенту и серверу.

Клиент – приложение, которое посылает запрос на обслуживание сервером. Задачей клиента является инициирование связи с сервером, отправка запроса серверу на обслуживание, получение от сервера результата обслуживания и предоставление результатов пользователю. Клиентская часть приложения функционирует на рабочем месте пользователя.

Сервер – процесс, который обслуживает запросы клиентов, предоставляет доступ к определенным ресурсам или услугам. Задачей сервера является получение запросов от клиентов, выполнение каких-либо функций по

запросу клиента и отправка результатов выполнения клиенту. Так же, сервер должен обеспечивать разграничение доступа к ресурсам.

Взаимодействие клиентской и серверной частей осуществляется через сеть и стандартизируется, так что сервер может обслуживать клиентов, реализованных различным образом. Для этого выбирается протокол запросов клиента и ответов сервера.

Компоненты такой клиент-серверной архитектуры по выполняемым функциям можно разделить на три слоя:

- Слой доступа к данным;
- Слой прикладной логики;
- Слой представления (визуализации) данных.

Слой доступа к данным отвечает за хранение, выборку, модификацию и удаление данных. Для большинства приложений логика данного слоя сосредоточена в коде СУБД.

Слой прикладной логики описывает основные функции приложения. Например, к таким функциям можно отнести вычисления на основе вводимых и хранимых данных, обработку команд, поступающих от слоя представления, передача информации слою доступа к данным.

Слой представления (визуализации) данных отвечает за отображение данных пользователю и за организацию взаимодействия с пользователем. Слой представления всегда выполняется на рабочем месте пользователя. Он не должен иметь прямых связей со слоем доступа к данным по требованиям масштабируемости и безопасности.

Критерием, который позволяет отнести систему к клиент-серверной архитектуре является то что, хотя бы один из слоев выполняется на другом компьютере и взаимодействие между компонентами системы осуществляется через сеть, посредством передачи запросов.

Клиент-серверные информационные системы разделяют на двухзвенные и трехзвенные.

1.2.2.1 Двухзвенная архитектура «клиент-сервер»

Двухзвенной данная архитектура называется так из-за необходимости распределения трех слоев между двумя узлами: клиентом и сервером. На сторону клиента выносятся логика представления данных пользователю, а на сторону сервера – логика доступа к данным, СУБД и сама база данных. Такой сервер называют сервером баз данных, и он представляет собой многопользовательскую СУБД, работающую, как правило, на мощном компьютере. Он принимает на себя функции обработки запросов пользователей.

Сервер баз данных дает возможность отказаться от пересылки по сети файлов данных целиком и передавать только ту выборку из БД, которая удовлетворяет запросу пользователя. Таким образом, увеличивается общая производительность информационной системы в результате объединения вычислительных ресурсов сервера и клиентской рабочей станции.

Оставшиеся компоненты прикладной логики приложения могут быть размещены как на стороне клиента, так и на стороне сервера. Реализация прикладной логики на одной из сторон приводит к тому, что мы имеем две конфигурации двухзвенной архитектуры:

- Конфигурация «толстый клиент»;
- Конфигурация «тонкий клиент».

Конфигурация «толстый клиент» более распространена, потому что суммарная вычислительная мощность клиентов предполагается большей, чем мощность единственного сервера. Такая модель подразумевает объединение логики работы с данными и логики представления на клиенте, а серверу оставляется только поддержание базы данных. Структура двухзвенной

архитектуры «клиент-сервер» в конфигурации «толстый клиент» представлена на рисунке 1.6.

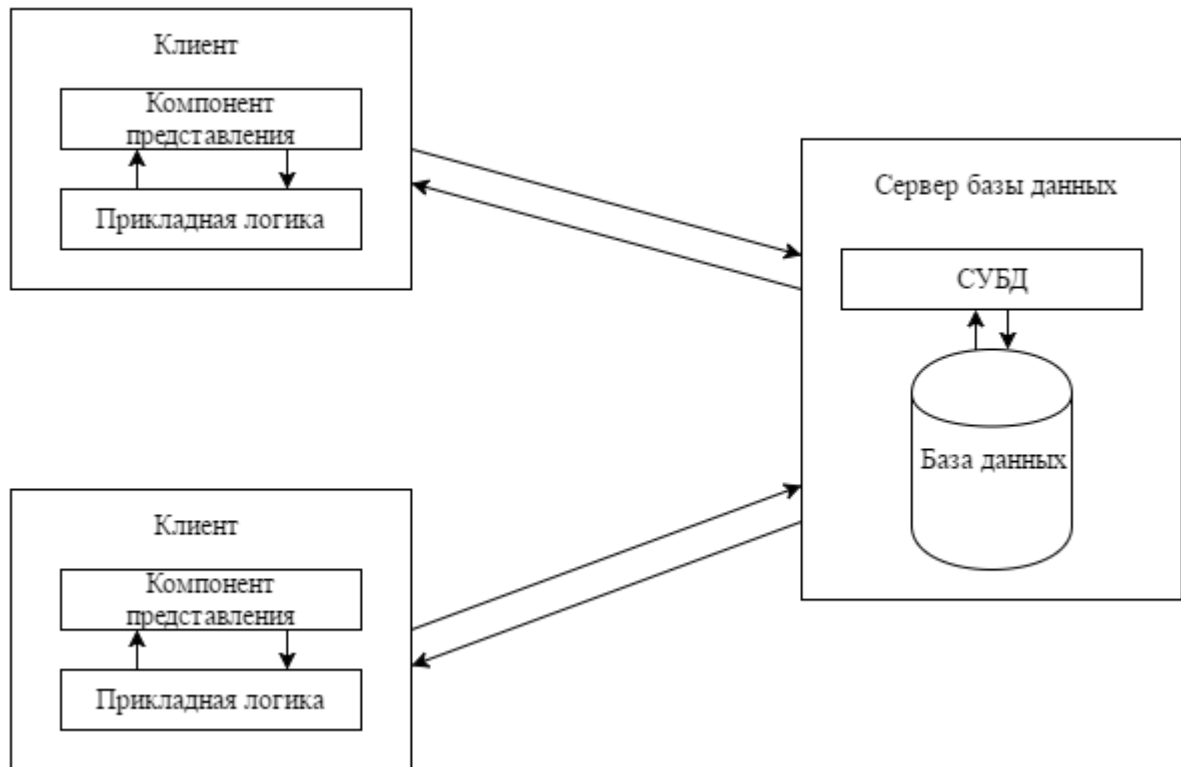


Рисунок 1.6 – Структура конфигурации «толстый клиент» двухзвенной архитектуры «клиент-сервер»

К преимуществам данной модели относятся высокое быстродействие, которое зависит от аппаратных средств клиента и возможность работы даже при обрывах связи с сервером.

Основные недостатки «толстого клиента»:

- Сложность обновления программного обеспечения, поскольку его замену нужно проводить одновременно во всей системе;
- Слабая защита данных, поскольку сложно правильно распределить полномочия;
- Большой размер клиентского приложения.

Конфигурация «тонкий клиент» заключается в минимизации прикладной логики на клиентском приложении за счет использования на сервере хранимых процедур, которые реализовывали часть логики приложения. Это позволяет сократить объем информации, которая передается по сети и увеличить безопасность. Структура такой конфигурации представлена на рисунке 1.7.

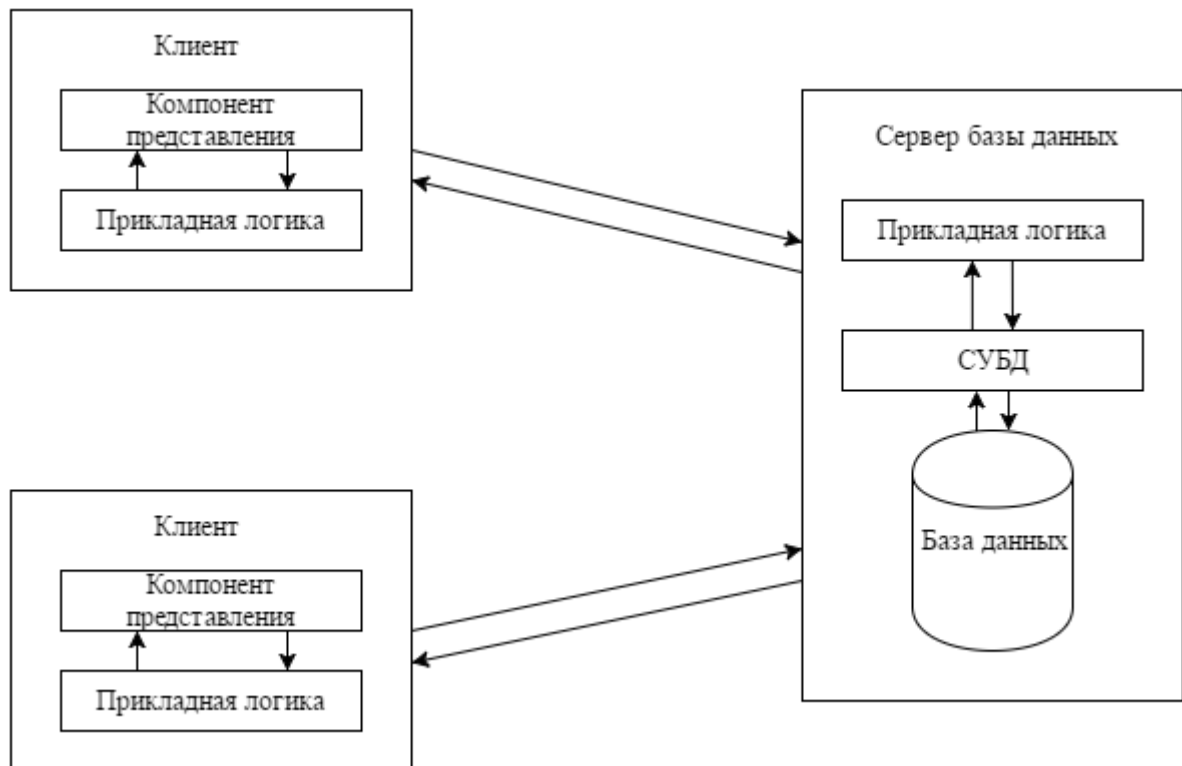


Рисунок 1.7 – Структура конфигурации «тонкий клиент» двухзвенной архитектуры «клиент-сервер»

При использовании такой модели невозможно перенести всю прикладную логику приложения на сервер, поэтому часть прикладной логики по-прежнему остается в клиентском приложении. Поэтому, такую архитектуру называют 2.5-звенной архитектурой.

Преимуществом такого подхода является то, что его легче поддерживать, так изменения нужно вносить только в одном месте – на

сервере. Так же, для таких приложений не требуются высокоскоростные каналы связи между клиентом и сервером. Улучшается защита информации - пользователям даются права на доступ к функциям системы, а не на доступ к ее данным.

В качестве недостатков такой архитектуры можно отметить то, что программы для серверной части системы пишутся на встроенных в СУБД языках описания хранимых процедур, предназначенных для проверки данных и построения несложных отчетов, а вовсе не для написания информационных систем масштаба предприятия. Это приводит к снижению быстродействия системы, повышению трудоемкости создания и модификации информационной системы. Еще одним недостатком является невозможность работы в системе без доступа к серверу.

Для решения основных проблем двухзвенной архитектуры была предложена трехзвенная архитектура.

1.2.2.2 Трехзвенная архитектура «клиент-сервер»

Трехзвенная архитектура представляет собой дальнейшее совершенствование технологии «клиент-сервер». Основным отличием от двухзвенной модели является то, между клиентом и сервером баз данных появляется третий, промежуточный уровень, являющийся для пользователя сервером, а для сервера баз данных – клиентом. Этот промежуточный уровень называется сервером приложений, и он реализует логику работы приложения, которая является наиболее часто изменяемым компонентом. Структура трехзвенной архитектуры «клиент-сервер» представлена на рисунке 1.8.

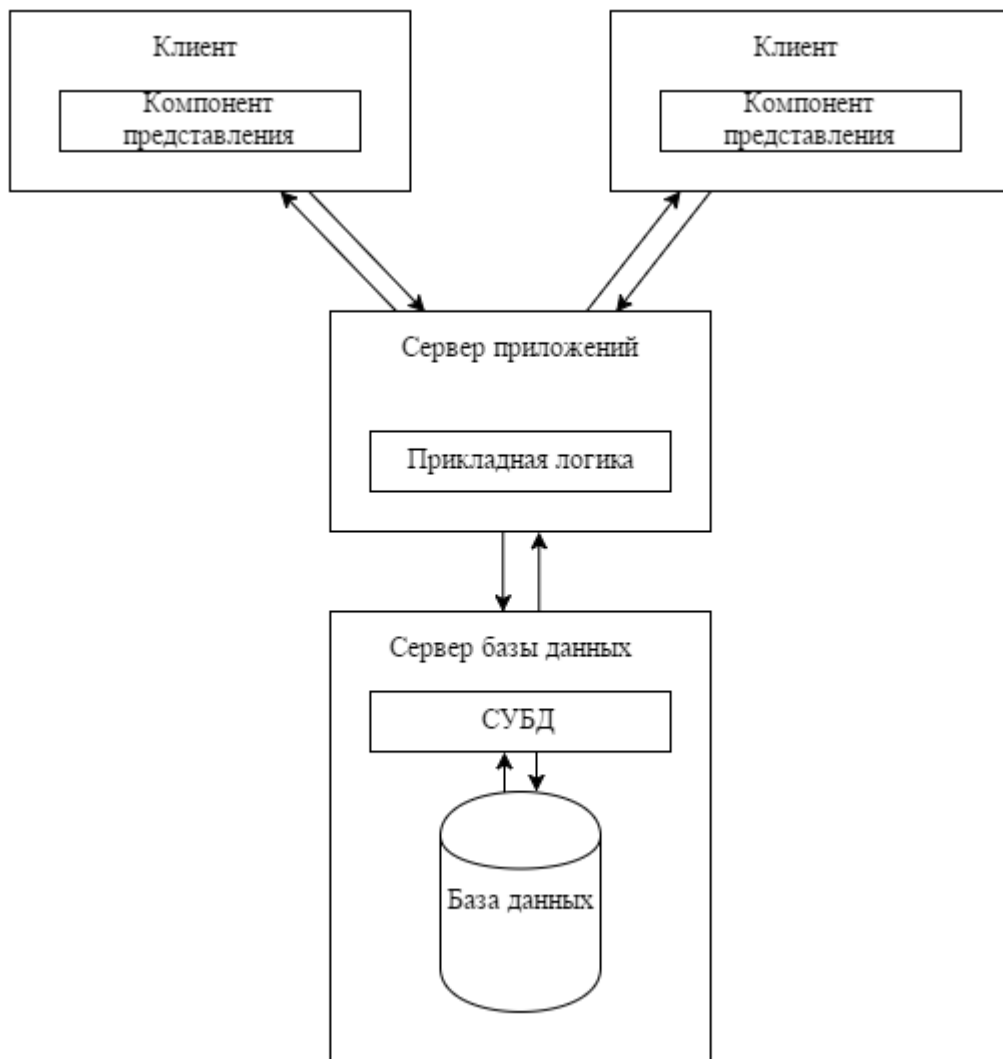


Рисунок 1.8 – Структура трехзвенной архитектуры «клиент-сервер»

Такая архитектура позволяет более гибко распределять функции системы и снизить нагрузку на сетевое и вычислительное оборудование. Клиентское приложение становится «тонким клиентом», так как в нем остается только логика представления данных пользователю. Это приводит к тому, что снижаются требования к аппаратным ресурсам на рабочих местах. На сервере баз данных остается только логика доступа к данным, СУБД и сама база данных, а весь оставшийся функционал переносится на сервер приложений.

К достоинствам такой архитектуры можно отнести простоту внесения изменений в алгоритмы прикладной логики, так как при ее изменении нет необходимости обновлять клиентское программное обеспечение. Так же,

между клиентом и сервером приложений по сети передается минимальный набор данных, снижается нагрузка на сервер баз данных, увеличивается безопасность.

Из недостатков можно отметить сложность разработки, внедрения, более высокие расходы на администрирование и обслуживание серверной части.

1.2.3 Выбор архитектуры разрабатываемого приложения

Для выбора архитектуры разрабатываемого приложения необходимо определить требования и ограничение к архитектуре приложения, учитывая специфику функционирования предприятия.

На предприятии АО «ВПК «НПО машиностроения» большой штат сотрудников, поэтому «файл-серверная» архитектура не подходит для создания нужной информационной системы, так как у этой архитектуры сильно ограничено количество пользователей. Еще одним недостатком является сложность обеспечения целостности и актуальности данных.

Если рассматривать двухзвенную «клиент-серверную» архитектуру, то у нее, в обеих ее конфигурациях, в клиентском приложении располагается прикладная логика. Большое количество работников станет проблемой в случае необходимости обновления логики работы системы, так как нужно будет обновлять клиент на каждом рабочем месте.

Следовательно, было решено использовать трехзвенную архитектуру «клиент сервер». В этом случае, приложение будет разбито на три уровня:

- Сервер базы данных и файлов для хранения данных и файлов, загружаемых пользователями;
- Сервер приложения для реализации логики работы системы;
- Клиентское приложение для представления данных и обеспечения взаимодействия с пользователем.

Репозиторий инсталляционных пакетов будет реализован как веб-приложение. Такой подход позволит работать с системой без установки и настройки клиентского приложения на рабочие места, так как для работы с веб-приложением необходим только браузер. Так же, обновление такой системы будет происходить автоматически.

1.3 Вывод

В разделе был проведён анализ возможностей аналогичных систем, после чего были определены некоторые требования к разрабатываемой системе. Также, были проанализированы основные виды архитектур информационных систем с организацией совместного доступа пользователей к данным. На основе основных достоинств и недостатков рассмотренных видов архитектур, для разработки репозитория инсталляционных пакетов было решено использовать трехзвенную архитектуру «клиент-сервер».

2 Разработка репозитория инсталляционных пакетов

2.1 Анализ предметной области и уточнение спецификации

2.1.1 Уточнение требований к системе

Разработка программного обеспечения начинается с анализа требований к функциональности, которые указаны в техническом задании. В соответствии с техническим заданием у разрабатываемой системы должно быть три типа пользователей – любой аутентифицированный пользователь, пользователь с правами модератора и пользователь с правами администратора. Аутентифицированный пользователь должен иметь возможность создавать заявки на добавление ПО, просматривать свои заявки на добавление ПО, просматривать каталог файлов и выгружать с сервера файлы. Модератор должен иметь возможность просматривать каталог ПО; добавлять новые файлы в каталог ПО; выгружать файлы с сервера; заполнять, редактировать и просматривать информацию о файлах; создавать и удалять категории; привязывать файлы к категориям; просматривать файлы, информация о которых еще не заполнена; просматривать информацию о пользователях. Администратор разрабатываемой системы, помимо всех возможностей модератора, должен иметь возможность изменять настройки приложения и выполнять скрипты.

Для уточнения требований к функциональности системы необходимо рассмотреть варианты ее использования. Вариант использования представляет собой характерную процедуру применения разрабатываемой системы конкретным действующим лицом [8]. Выявленные варианты использования разрабатываемой системы представлены на диаграмме вариантов использования (рисунок 2.1).

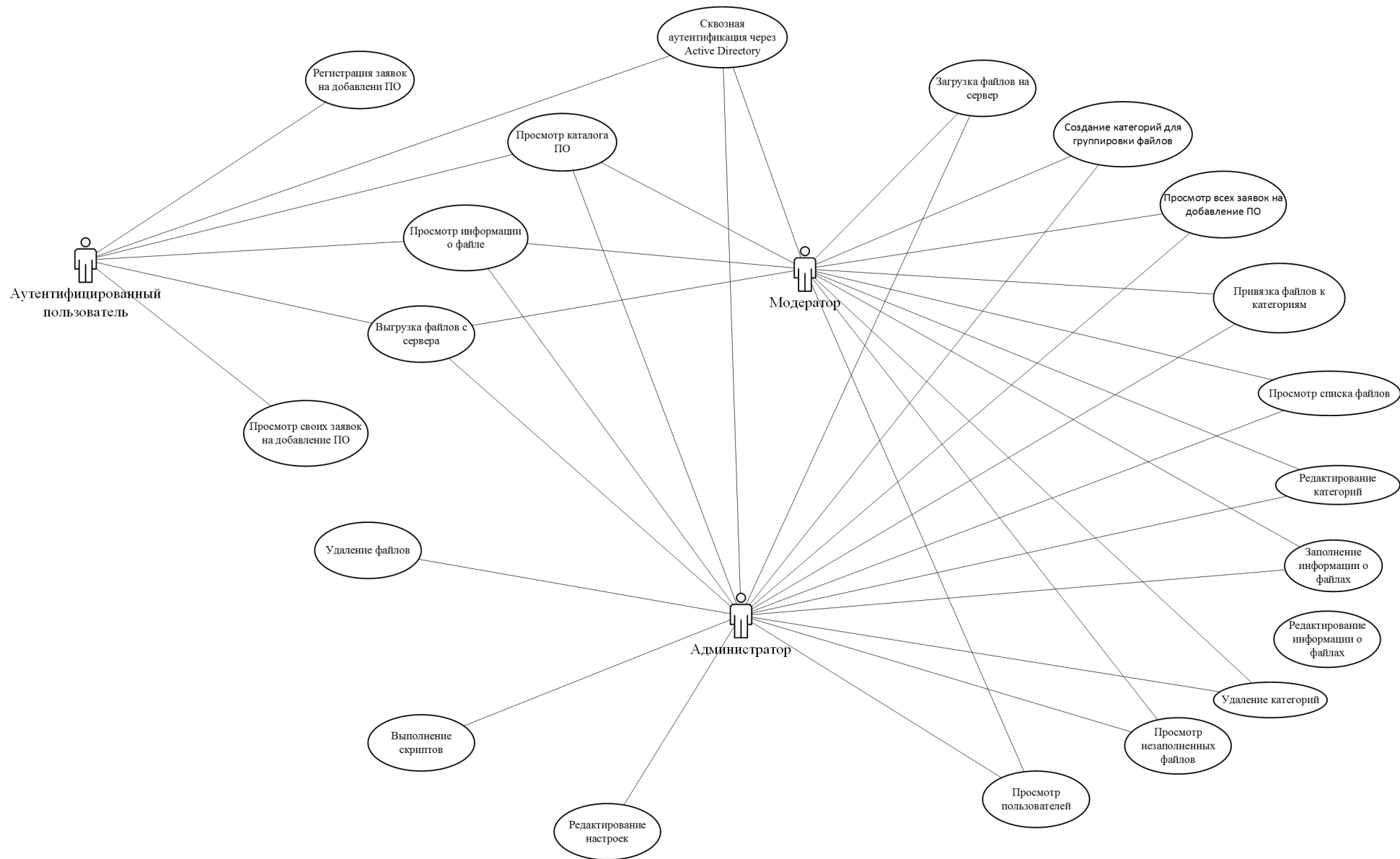


Рисунок 2.1 – Диаграмма вариантов использования разрабатываемой системы

Согласно ТЗ, необходимо уточнить тип информации, которую могут заполнять модераторы и администраторы у файлов. В соответствии с ТЗ, разрабатываемая системы должна автоматически извлекать из файлов формата Portable Executable следующую информацию:

- описание файла;
- версию файла;
- версию продукта;
- название производителя;
- название продукта.

У тех файлов, у которых не заполнены данных свойства, или у файлов других форматов, данные свойства должны заполняться модератором вручную. В пункте 1.1 было решено предоставить пользователям набор стандартных свойств файлов. Следовательно, в качестве стандартных свойств будут использоваться описание файла, версия файла, версия продукта, название производителя и название продукта. Также, в пункте 1.1 было решено предоставить пользователям возможность добавлять новые свойства для файлов. Для того, чтобы свойства можно было добавлять к файлам, необходимо, также, реализовать привязку свойств к файлам и их заполнение.

Также, согласно ТЗ, необходимо уточнить тип информации о пользователях, которую модераторы и администраторы смогут просматривать. В качестве такой информации было решено использовать:

- ФИО;
- рабочий телефон;
- рабочий email;
- название отдела;
- номер отдела;
- адрес отдела.

2.1.2 Выбор способа проектирования

В качестве способа проектирования был выбран объектно-ориентированный подход, который основан на объектной декомпозиции. При такой декомпозиции каждый объект обладает своим собственным поведением и каждый из них моделирует некоторый объект реального мира. Программа, полученная при использовании такого подхода, описывает взаимодействие объектов. Объектная декомпозиция имеет ряд преимуществ:

- Объектная декомпозиция уменьшает размер программных систем за счет повторного использования общих механизмов, что приводит к существенной экономии выразительных средств. [7]
- Объектно-ориентированные системы более гибки и проще эволюционируют со временем, потому что их схемы базируются на устойчивых промежуточных формах. Действительно, объектная декомпозиция существенно снижает риск при создании сложной программной системы, так как она развивается из меньших систем, в которых мы уже уверены. [7]
- Объектная декомпозиция помогает нам разобраться в сложной программной системе, предлагая нам разумные решения относительно выбора подпространства большого пространства состояний. [7]

При выполнении объектной декомпозиции в результате анализа предметной области были выделены следующие сущности: пользователь, файл, версия файла, категория, свойство, свойство файла, свойство версии файла, категория файла, заявка и файл заявки. Для описания этих сущностей будут использованы классы.

2.1.3 Выбор программной платформы

Согласно техническому заданию, при разработке репозитория инсталляционных пакетов должен использоваться язык программирования Java. Java - это объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Приложения, написанные на Java, транслируются в специальный байт-код, что позволяет им работать на любой виртуальной Java-машине вне зависимости от компьютерной архитектуры. К основным преимуществам данного языка программирования можно отнести мощные стандартные библиотеки, большое количество разнообразных инструментов сборки проектов, автоматическая сборка мусора. Язык программирования Java - это язык со строгой статической типизацией, что позволяет выявлять большинство ошибок на стадии компиляции. Так же, скорость выполнения статически типизированных языков практически всегда выше, чем скорость выполнения динамически типизированных языков.

В пункте 1.2 был произведен анализ основных видов архитектур, в результате чего, для разработки репозитория инсталляционных пакетов была выбрана трехзвенная архитектура «клиент-сервер». Данная архитектура в качестве промежуточного слоя между клиентом и сервером использует сервер приложений, который должен реализовывать логику работы разрабатываемой системы. Так как в качестве языка программирования используется Java, то необходимо выбрать сервер приложений, который совместим с Java приложениями. Сервер приложений – это программа промежуточного слоя между клиентом и сервером, которая предоставляет среду для развертывания, управления и выполнения Java приложений. При выборе сервера приложений были рассмотрены бесплатные сервера приложений с открытым исходным кодом, которые имеют полную поддержку последней версии Java Enterprise Edition. Java Enterprise Edition (Java EE) – это платформа, содержащая набор взаимосвязанных технологий для создания многоуровневых серверных приложений. Актуальная версия Java EE на сегодняшний момент имеет номер

7.0. В качестве серверов приложений, которые удовлетворяют описанным выше требованиям, можно выделить GlassFish и WildFly.

GlassFish является сервером приложений с открытым исходным кодом, реализующим спецификации Java EE, принадлежит корпорации Oracle. С 2013 года Oracle объявили о прекращении оказания услуг коммерческой техподдержки. Пользователям коммерческих версий было рекомендовано перейти на закрытый сервер приложений WebLogic Server.

WildFly - сервер приложений с открытым исходным кодом. На его основе создаётся сертифицированный для Java EE коммерческий продукт JBoss Enterprise Application Platform. Коммерческая версия JBoss предоставляет такие возможности, как пятилетний жизненный цикл поддержки, системы кэширования и кластеризации, устойчивость к веб-атакам и собственную среду разработки.

В качестве сервера приложений был выбран WildFly. К данному серверу приложений чаще выходят обновления, и он имеет режим Server Suspend Mode. При включении данного режима сервер приложений перестает принимать новые запросы, обрабатывает оставшиеся в очереди запросы и выключается, для проведения необходимых работ по обновлению или администрированию программного обеспечения.

Для того, чтобы ускорить процесс разработки было решено использовать фреймворк. Фреймворк (от англ. framework — каркас, структура) - программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта. Кроме того, фреймворк определяет архитектуру, взаимосвязь между компонентами разрабатываемого приложения. При выборе фреймворка были рассмотрены три наиболее популярных Java-фреймворка: Apache Struts, Java Server Faces и Spring Framework. Большинство фреймворков в качестве архитектурного шаблона

используют шаблон MVC (Model-View-Controller). Model-view-controller (MVC, «модель-представление-контроллер», «модель-вид-контроллер») — схема использования нескольких шаблонов проектирования, с помощью которых модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные.

Apache Struts это фреймворк с открытым исходным кодом для разработки веб-приложений на Java. В качестве архитектурного шаблона данный фреймворк использует шаблон "Модель-представление-контроллер" (MVC). Основными достоинствами фреймворка являются легкость в изучении, помощь в валидации форм и удобная работа с событиями на стороне сервера. К недостаткам можно отнести то, что фреймворк имеет плохую документацию и то, что фреймворк морально устарел.

Java Server Faces (JSF) – это фреймворк для веб-приложений, написанный на Java. Он служит для того, чтобы облегчать разработку пользовательских интерфейсов для Java-приложений. В отличие от прочих фреймворков, которые управляются запросами, подход JSF основывается на использовании компонентов. Фреймворк предоставляет набор стандартных элементов для построения пользовательских интерфейсов. Среди достоинств данного фреймворка можно отметить то, что он содержит большое количество элементов для создания пользовательских элементов и позволяет экономить время на верстке страниц, так как html и javascript код генерируются фреймворком автоматически. К недостаткам относится сложность реализации функциональности, которая не предусмотрена авторами и проблемы с совместимостью версий.

Spring Framework – универсальный фреймворк с открытым исходным кодом для Java платформы. Он имеет довольно широкую функциональность и активно используется при разработке сложных приложений. Spring Framework

может быть рассмотрен как коллекция меньших модулей. Большинство этих модулей может работать независимо друг от друга. В качестве архитектурного шаблона данный фреймворк предлагает использовать шаблон "Модель-представление-контроллер" (MVC). К достоинствам Spring Framework относится большое сообщество, хорошая документация и гибкость – так как он состоит из множества модулей, то можно подключать только необходимые разрабатываемому проекту модули. Из недостатков можно отметить то, что он сложнее других рассмотренных фреймворков в изучении и не предоставляет никаких инструментов для создания пользовательских инструментов.

В качестве фреймворка для разработки репозитория инсталляционных пакетов был выбран Spring Framework, так как он состоит из компонентов, которые не зависят друг от друга. Этот подход позволяет выбрать только нужные компоненты и не подключать к нашему приложению лишние библиотеки, которые не будут использоваться.

Последним компонентом разрабатываемой системы является СУБД. Система управления базами данных (СУБД) – это совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями. Основные функции СУБД заключаются в:

- управлении данными во внешней памяти;
- управлении данными в оперативной памяти;
- журнализация изменений, резервное копирование и восстановление базы данных после сбоев;
- поддержка языков БД (язык определения данных, язык манипулирования данными).

В состав современной СУБД входят:

- ядро, которое отвечает за управление данными во внешней и оперативной памяти и журнализацию;

- процессор языка базы данных, который обеспечивает оптимизацию запросов на извлечение и изменение данных и создание, как правило, машинно-независимого исполняемого внутреннего кода;
- подсистему поддержки времени исполнения, которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД;
- сервисные программы, обеспечивающие ряд дополнительных возможностей по обслуживанию информационной системы.

В соответствии с техническим заданием при разработке репозитория инсталляционных пакетов должна использоваться СУБД PostgreSQL. PostgreSQL – это свободная объектно-реляционная СУБД. Она имеет большое количество реализаций для множества UNIX-подобных платформ и реализацию для Microsoft Windows. Разработка PostgreSQL является общественным проектом и не управляется какой-либо компанией.

2.2 Разработка структуры программы и ее компонентов

2.2.1 Разработка структуры программы

Детальный разбор поставленной задачи и вариантов использования позволил разработать концептуальную модель предметной области. Концептуальная модель представляет собой контекстную диаграмму классов. Эта диаграмма показывает связи между основными понятиями предметной области, которые были выделены при объектной декомпозиции в пункте 2.1.2. Контекстная диаграмма классов репозитория инсталляционных пакетов представлена на рисунке 2.2.

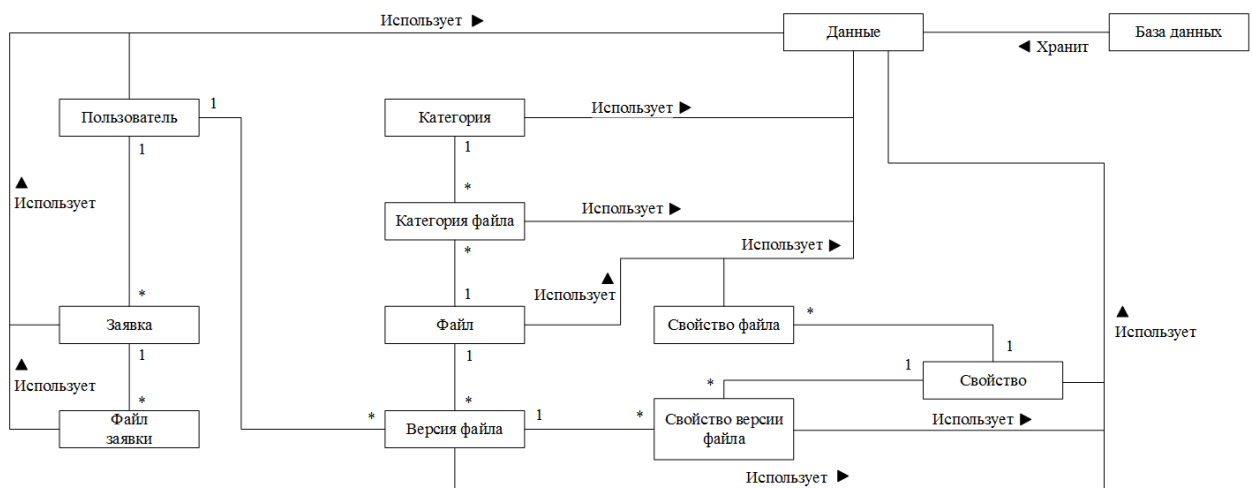


Рисунок 2.2 – Контекстная диаграмма классов предметной области

2.2.2 Разработка базы данных и классов

В соответствии с техническим заданием, вся информация в системе должна храниться в СУБД PostgreSQL. В результате объектной декомпозиции в разрабатываемой системе были выделены сущности, каждая из которых будет отображена соответствующей реляционной таблицей. Кроме того, для каждой сущности необходимо разработать классы. Разрабатываемые классы будут содержать работу с базой данных: чтение записей из базы данных, добавление и обновление записей в базу данных и удаление записей из базы данных. Получается, что в классах будут однотипные методы, но у них будет разная реализация. Поэтому, целесообразно вынести такие методы в базовый класс `ModelInterface`, который будет являться интерфейсом. В нем содержаться методы:

- `update()` для обновления записи в базе данных;
- `add()` для сохранения новой записи в базу данных;
- `delete()` для удаления записи из базы данных;
- `validate()` для проверки правильности заполнения полей.

Структура интерфейса ModelInterface представлена на рисунке 2.3. Для каждой таблицы базы данных будет разработан класс, который будет с ней работать. В качестве полей класса будут выступать названия полей из соответствующей таблицы, а в качестве методов – различные необходимые функции, например, функции поиска в базе данных по различным параметрам, функции для получения связанных моделей и другие. Все такие классы будут реализовывать интерфейс ModelInterface. Каждой записи в таблице будет соответствовать экземпляр соответствующего класса, значений полей объекта будут соответствовать значениям полей записи в таблице базы данных.

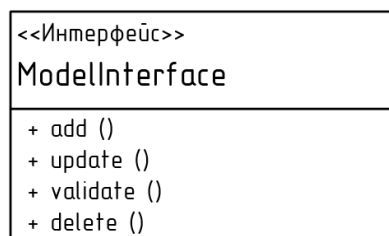


Рисунок 2.3 – Диаграмма интерфейса ModelInterface

Для хранения основных данных о пользователях в системе была разработана таблица user. Данная таблица содержит: поле displayName для хранения ФИО пользователя; поле phone для хранения телефона; поле email для хранения электронного адреса; поля department и departmentNumber для хранения названия отдела, в котором работает пользователь и номер отдела соответственно; поле address для хранения адреса отдела. Для работы с таблицей пользователей user был разработан класс UserModel. Диаграмма данного класса представлена на рисунке 2.4.

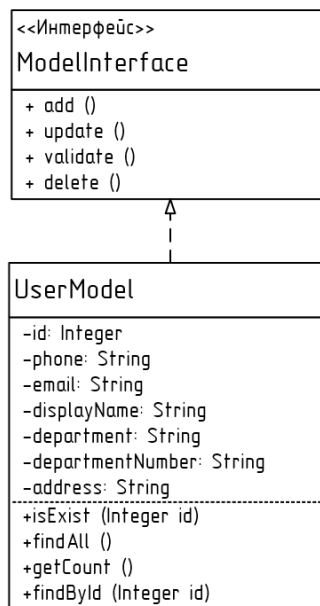


Рисунок 2.4 – Диаграмма класса UserModel

Помимо реализации методов интерфейса ModelInterface разработанный класс содержит следующие функции:

- `findAll ()` – возвращает массив экземпляров UserModel для каждой записи таблицы;
- `findById (Integer id)` – возвращает экземпляр UserModel, которая соответствует записи, найденной по ее уникальному идентификатору или null;
- `getCount ()` – возвращает количество записей в таблице.

Согласно поставленной задаче, модераторы и администраторы могут добавлять файлы. Для работы с этими файлами была выделена сущность «версия файла» и разработана таблица fileVersion. Эта таблица предназначена для хранения информации по конкретной версии файла программного обеспечения. Поле `userId` указывает на `id` пользователя из таблицы `user`, который загрузил данный файл, а поле `fileId` – на сущность «файл», которая позволяет группировать разные версии файлов. Поле `version` предназначено для хранения версии загруженного файла, поле `fileName` – для сохранения имени файла. Для того, чтобы избежать дублирования версий в системе были добавлены 2 поля: `hash`, которое хранит хэш-сумму файла и `fileSize`, которое

хранит размер файла. В поле дата хранится дата загрузки файла в систему, поле `isFilled` является флагом который показывает, заполнены ли у версии обязательные поля, а поле `isDisabled` – флагом, который показывает включен или отключен ли файл для вывода в каталог программного обеспечения.

Класс `FileVersionModel` необходим для работы с таблицей `fileVersion`. При добавлении новых файлов в систему они считаются сущностью «версия файла». В соответствии с техническим заданием, из файлов с расширением «Portable Executable» (ОС Windows) система автоматически извлекается список заполненных у них свойств, таких как название продукта, версия продукта, автор, описание файла и другие. В качестве двух основных свойств, заполнение которых обязательно для каждого загружаемого файла были определены название и версия. Название необходимо для группировки версий в одну сущность «файл», а версия – для возможности вывода списка версий. Если у загружаемого файла данные поля заполнены, то происходит поиск сущности «файл» с таким же названием. В случае, если такая сущность уже существует, загруженный файл добавляется как новая версия в данной сущности. Если в таблице `file` не было найдено записей с таким названием, то такая запись создается и к ней добавляется версия файла. В случае, если у загружаемого файла не заполнены название или версия, он помечается как незаполненный и нуждается в дальнейшем заполнении вручную. Диаграмма класса `FileVersionModel` представлена на рисунке 2.5.

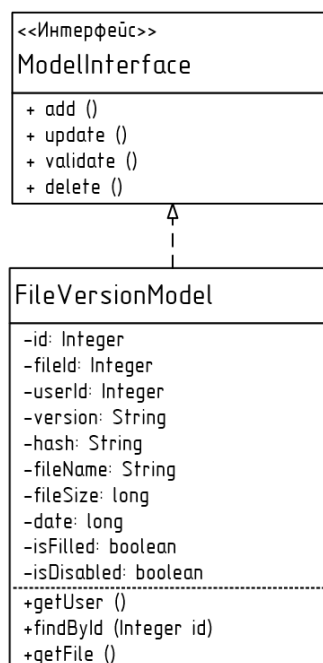


Рисунок 2.5 – Диаграмма класса FileVersionModel

Так как при добавлении нового файла должна осуществляться проверка на дублирование, то для этого был добавлен метод `isExist (String hash, long fileSize)`. В качестве параметров он принимает хэш-сумму файла, который хотят добавить в систему и его размер, и ищет запись в таблице с такими же значениями. Метод возвращает `true`, если такой файл уже существует, или `false`, если такого файла еще нет. Метод `getById (Integer id)` возвращает экземпляр класса `FileVersionModel` для записи с нужным `id`, или `null`, если такой записи в таблице нет. Для получения экземпляра пользователя, который добавил файл, был добавлен метод `getUser()`, для получения экземпляра `FileModel`, к которому прикреплен версия – метод `getFile()`.

Сущность «файл» предназначена для того, чтобы группировать версии файлов по общим признакам, таким как название. Таким образом, получается, что у одного файла может быть несколько версий. Для данной сущности была разработана таблица `file`. В данной таблице храниться информация, которая является общей для всех объединенных в один файл версий. В качестве такой информации было выделено только название, для которого предназначено

поле title. Для работы с таблицей file был разработан класс FileModel. Диаграмма класса представлена на рисунке 2.6.

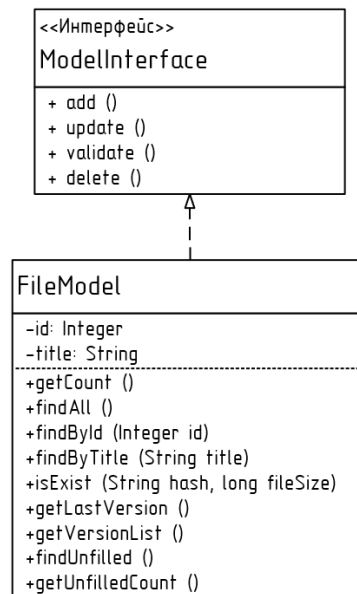


Рисунок 2.6 – Диаграмма класса FileModel

Класс содержит следующие методы:

- `findAll ()` – возвращает массив экземпляров класса `FileModel`;
- `findById (Integer id)` – возвращает экземпляр класса `FileModel` для записи с необходимым уникальным идентификатором;
- `findByTitle (String title)` - возвращает экземпляр класса `FileModel` для записи с необходимым названием;
- `getCount ()` – возвращает количество записей в таблице;
- `isLastVersion ()` – возвращает экземпляр `FileVersionModel`, который соответствует последней версии файла;
- `getVersinoList ()` – возвращает список всех версий файла в виде массива экземпляров `FileVersionModel`;
- `findUnfilled ()` – возвращает массив незаполненных файлов в системе (файлы, которые не привязаны ни к одной категории и версии файлов, у которых не заполнены обязательные поля).

Так как в соответствии с техническим заданием у файлов должны быть свойства, была выделена сущность «свойство». Для хранения свойств предназначена таблица property. Название свойства храниться в поле title. Для работы со свойствами был разработан класс PropertyModel, диаграмма которого представлена на рисунке 2.7.

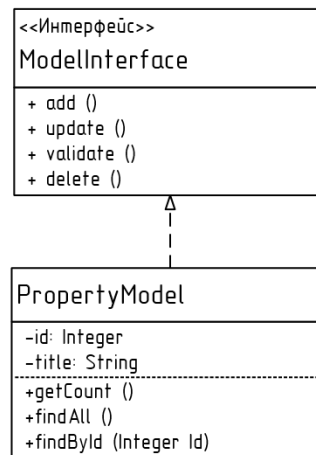


Рисунок 2.7 – Диаграмма класса PropertyModel

Метод findAll() предназначен для получения массива всех свойств в системе, метод getCount() – для получения количества свойств в системе. В соответствии с техническим заданием, свойства могут быть привязаны как к сущности «файл», так и к сущности «версия файла». Для этого были выделены две сущности – свойства файла и свойства версии соответственно.

Для работы со свойствами файла предназначена таблица fileProperty и класс FilePropertyModel. Данная таблица реализует связь «многие ко многим». Для связи с таблицей file предназначено поле fileId, а для связи со свойством – поле propertyId. Для хранения значения свойство предназначено поле value. Диаграмма класса FilePropertyModel представлена на рисунке 2.8.

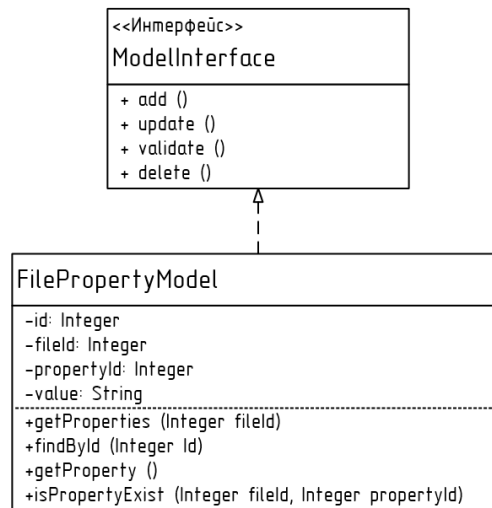


Рисунок 2.8 – Диаграмма класса FilePropertyModel

Для получения экземпляра свойства предназначен метод `getProperty()`. Метод `getProperties()` предназначен для получения массива всех свойств в системе, метод `findById (Integer id)` – для получения экземпляра свойства файла `FilePropertyModel`. Для того, чтобы избежать дублирования связи файла со свойством, был добавлен метод `isPropertyExist (Integer fileId, Integer propertyId)`, который проверяет, существует ли в таблице базы данных запись для заданного файла и свойства.

Для свойств версии файла были разработаны таблица `fileVersionProperty` и класс для работы с ней – `FileVersionPropertyModel`. Таблица `fileVersionProperty`, аналогично таблице `fileProperty`, реализует связь «многие ко многим». Для связи с таблицей `fileVersion` нужно поле `fileVersionId`, для связи с таблицей `property` – поле `propertyId`, а для хранения значения свойства – поле `value`. Класс `FileVersionPropertyModel`, в свою очередь, аналогичен классу `FilePropertyModel` – он содержит такие же методы. Диаграмма данного класса представлена на рисунке 2.9.

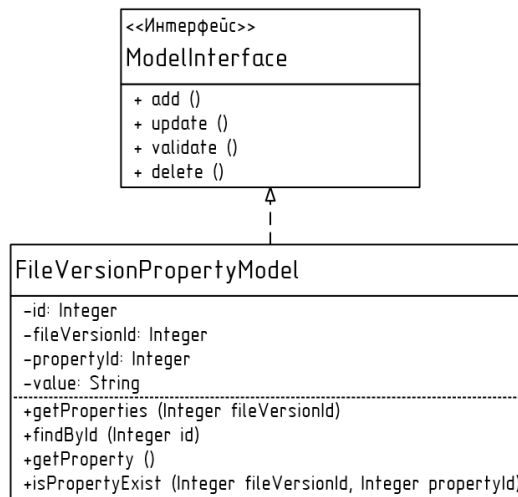


Рисунок 2.9 – Диаграмма класса FileVersionPropertyModel

Согласно техническому заданию, в системе должны быть категории, которые предназначены для группировки файлов. Для хранения категорий в базе данных была разработана таблица `category`, а для работы с ней – класс `CategoryModel`. Для того чтобы категории можно было вкладывать друг в друга, образуя тем самым древовидную структуру, используется поле `parent`, в котором храниться `id` категории родителя или `0`, если родителя нет. Для предоставления модератору или администратору возможности изменения порядка категорий предназначено поле `position`, которое отражает номер позиции для категории. Название категории хранится в поле `title`. Диаграмма класса `CategoryModel` представлена на рисунке 2.10.

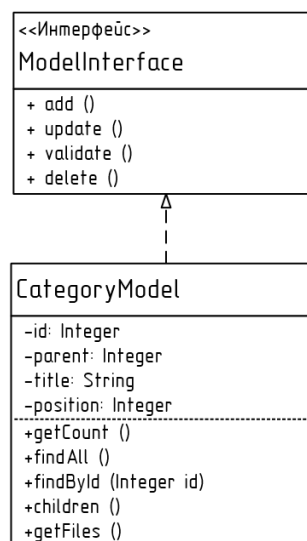


Рисунок 2.10 – Диаграмма класса CategoryModel

Класс CategoryModel включает в себя следующие методы:

- getCount () – получение количества категорий в системе;
- findAll () – получения массива всех категорий;
- findById (Integer id) – получение экземпляра CategoryModel для категории с необходимым уникальным идентификатором;
- children () – получение массива категорий, которые привязаны к текущей категории;
- getFiles () – получение списка файлов, которые привязаны к категории.

Для обеспечения возможности связи файлов с категориями была выделена сущность «категория файла», после чего разработана таблица fileCategory и класс FileCategoryModel. Таблица fileCategory реализует связь «многие ко многим». Она связана с таблицей category полем categoryId и с таблицей file полем fileId. Диаграмма класса FileCategoryModel представлена на рисунке 2.11.

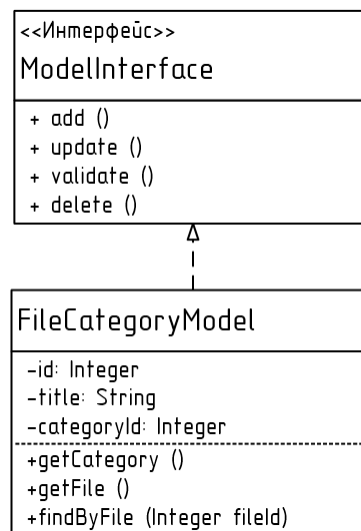


Рисунок 2.11 – Диаграмма класса FileCategoryModel

Данный класс имеет связи с классами FileModel и CategoryModel. Для получения экземпляров классов по этим связям служат методы getFile() и

getCategory(). Для получения всех связей с категориями для необходимого файла был добавлен метод findByFile (Integer fileId).

Согласно техническому заданию, пользователи должны иметь возможность оставлять заявки на добавление программного обеспечения. Заявки должны включать в себя набор файлов, которые пользователя хотят добавить в систему, и текст заявки с небольшим описанием файлов. Так как у одной заявки может быть сразу несколько прикрепленных файлов, то было выделено две сущности: заявка и файл заявки. Для сущности заявка были разработаны таблица в базе данных request и класс RequestModel. Таблица request содержит поля: userId – для связи с пользователем, который оставил заявку; text – для хранения текста заявки; date – для хранения даты регистрации заявки; status – для хранения статуса заявки; comment – для комментария модератора. Диаграмма класса RequestModel представлена на рисунке 2.12.

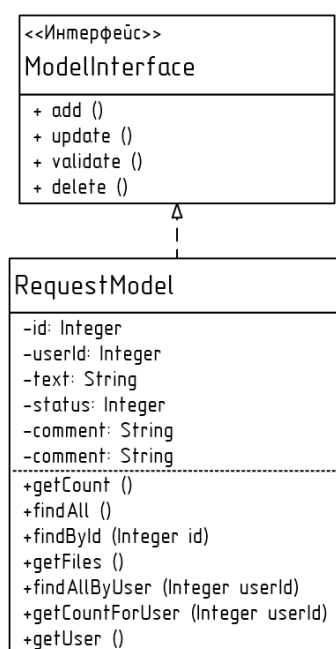


Рисунок 2.12 – Диаграмма класса RequestModel

Когда пользователь регистрирует новую заявку, ей присваивается статус «новая» и она ожидает проверки модератором. Модератор может отклонить заявку, тогда статус заявки измениться на «отклонена», или принять заявку,

тогда ее статус измениться на «принята», а файлы будут добавлены в систему. Так же, при принятии или отклонении заявки модератор может оставить комментарий. Для получения массива всех заявок в класс был добавлен метод `findAll()`, для получения количества заявок в системе – метод `getCount()`. Метод `getById(Integer id)` позволяет получить заявку по ее уникальному идентификатору. Метод `getAllByUser(Integer userId)` позволяет получить список всех заявок от конкретного пользователя, метод `getCountForUser(Integer userId)` – получить количество заявок от пользователя. Для связи с пользователем был добавлен метод `getUser()`, а для получения списка файлов в заявке – метод `getFiles()`.

Для того, чтобы к одной заявке можно было прикрепить несколько файлов, файлы заявки были выделены в отдельную сущность и для нее были разработаны таблица `requestFile` и класс `RequestFileModel`. Поле `requestId` необходимо для связи файлов с заявкой, поле `fileName` – для хранения имени файла. В целях обеспечения безопасности загружаемые в заявку файлы кодируются в формат `base64` и сохраняются на диске как текстовые файлы. Для того, чтобы в дальнейшем была возможность восстановить файл в его исходное состояние было добавлено поле `extension`, для сохранения расширения файла. Для проверки файлов на дублирование, так же как у сущности «версия файла», были добавлены два поля: `hash` – для хранения хэш-суммы файла и `fileSize` – для хранения размера файла. В момент создания заявки, файлы проверяются на дублирование по этим двум полям. Диаграмма класса `RequestFileModel` представлена на рисунке 2.13.

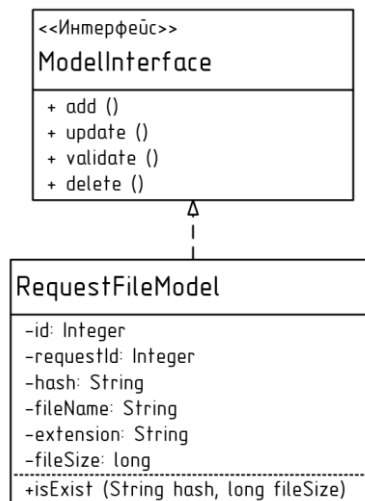


Рисунок 2.13 – Диаграмма класса RequestFileModel

Для проверки файлов на дублирование в класс был добавлен метод `isExist(String hash, long fileSize)`. Аналогично методу в классе `FileVersionModel`, данный метод в качестве параметров принимает хэш-сумму файла, который хотят добавить в систему и его размер, после чего ищет запись в таблице `requestFile` с такими же значениями. Метод `isExist()` возвращает `true`, если такой файл уже существует, или `false`, если такого файла еще нет.

Согласно техническому заданию, в разрабатываемой системе необходимо предусмотреть возможность выполнения скриптов. Данная возможность в дальнейшем будет использована для экспорта программ в `System Center Configuration Manager`, который используется на предприятии АО "ВПК "НПО машиностроения". Для того, чтобы не вводить одни и те же команды для различных файлов, были разработаны таблица в базе данных `exportTemplate` и класс `ExportTemplateModel`. Они позволяют сохранять наборы необходимых команд в шаблон, а затем использовать этот шаблон для экспорта нескольких файлов. Поле `title` предназначено для имени шаблона, поле `parameters` – для списка параметров в шаблоне, поле `finalCommands` – для команд шаблона и поле `finalCommandsInterpreter` – для хранения типа интерпретатора, с помощью которого будут выполняться команды. Диаграмма класса `ExportTemplateModel` представлена на рисунке 2.14.

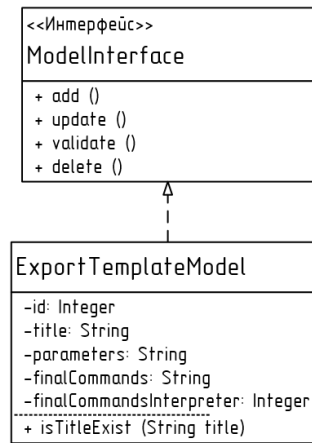


Рисунок 2.14 – Диаграмма класса ExportTemplateModel

Метод `isTitleExist (String title)` предназначен для проверки существования шаблона с таким названием.

Полученные в результате проектирования логическая схема базы данных и диаграмма классов представлены на рисунке 2.15 и 2.16 соответственно.

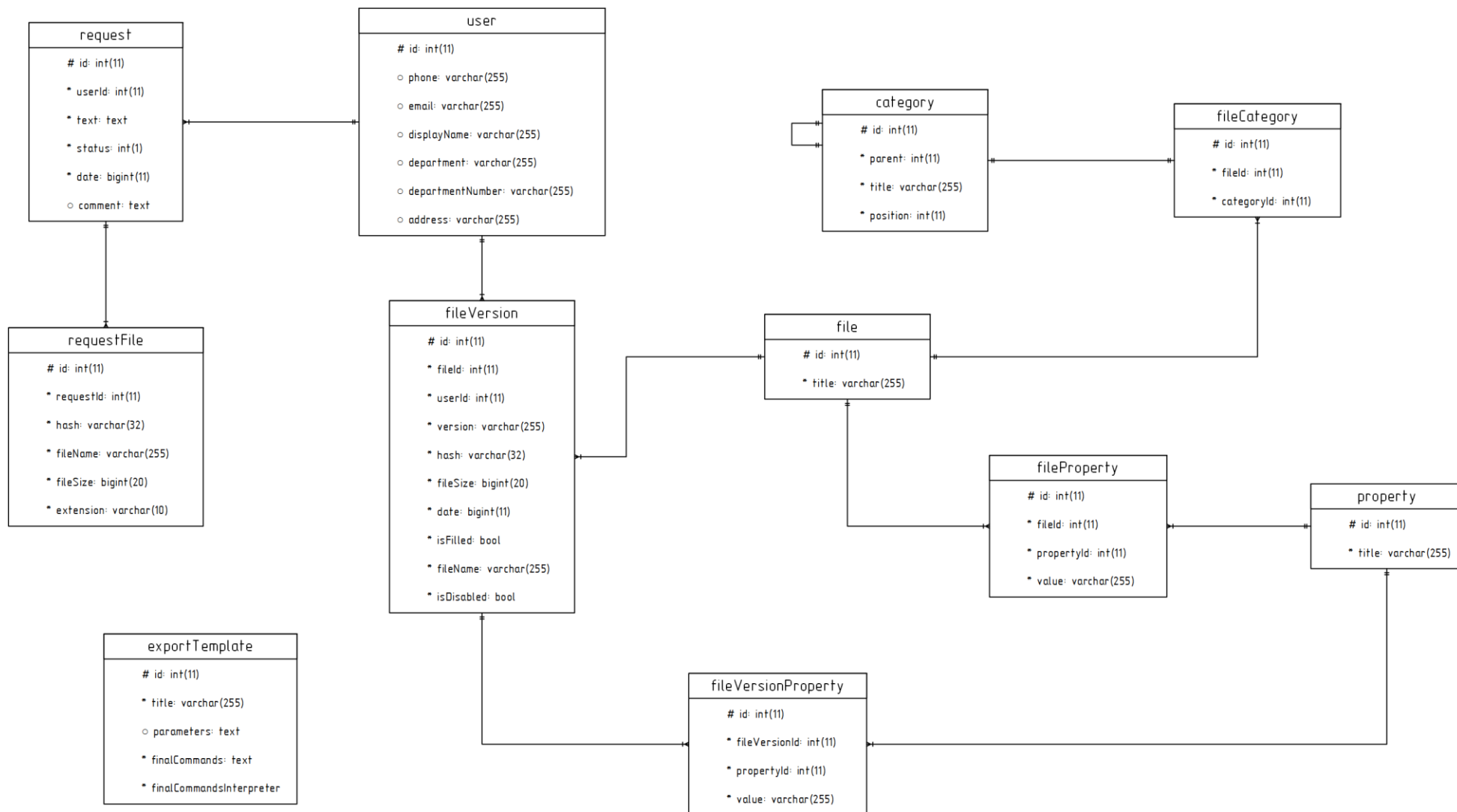


Рисунок 2.15 – Логическая схема базы данных

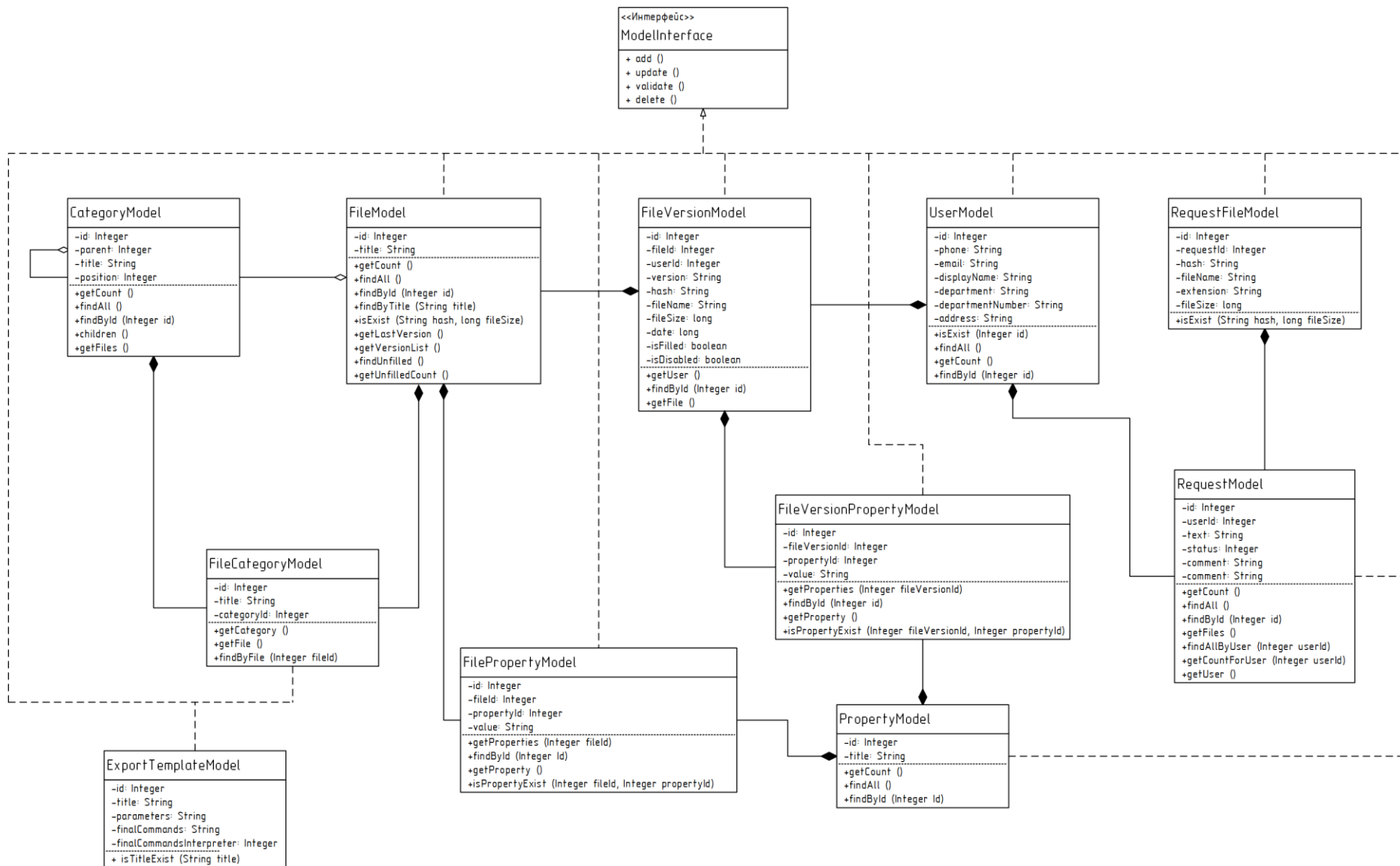


Рисунок 2.16 – Диаграмма классов

2.2.3 Разработка алгоритмов

После того, как были разработаны база данных и классы, необходимо реализовать алгоритмы работы приложения, за счет которых разрабатываемая система будет выполнять требуемые от нее функции, которые описаны в техническом задании.

В соответствии с техническим заданием для работы с приложением все пользователи должны пройти процедуру аутентификации через Active Directory. Диаграмма последовательности процедуры аутентификации представлена на рисунке 2.17.

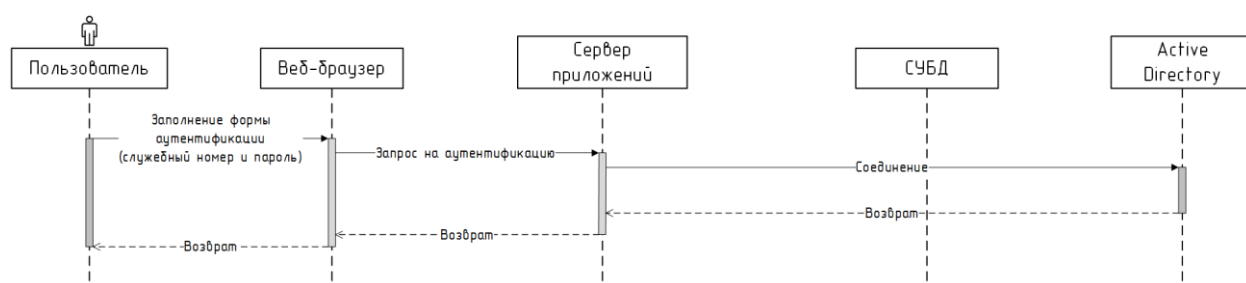


Рисунок 2.17 – Диаграмма последовательности процедуры аутентификации

Пользователь заполняет форму, вводя свой служебный номер и пароль и отправляет запрос на аутентификацию к серверу приложений. Диаграмма активности процедуры аутентификации представлена на рисунке 2.18.

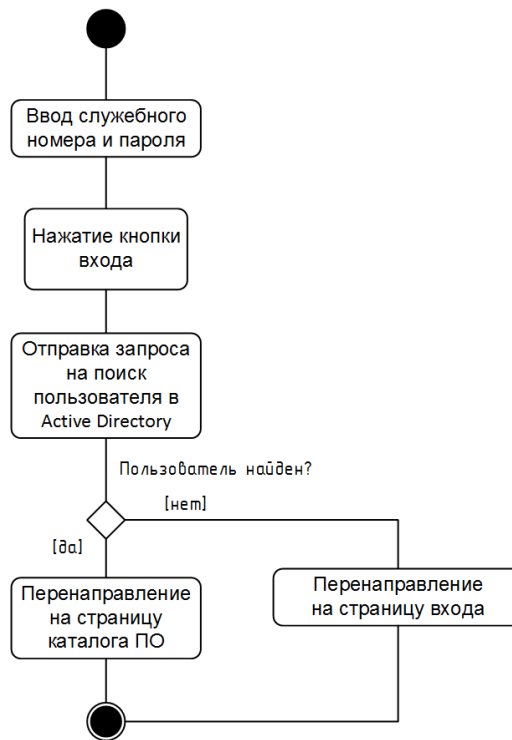


Рисунок 2.18 – Диаграмма активности процедуры аутентификации

Одной из основных функций репозитория инсталляционных является просмотр каталога программного обеспечения. Данная возможность должна быть доступна всем пользователям, которые прошли процедуру аутентификации, поэтому проверка роли пользователя не требуется. Диаграмма последовательности рассматриваемой процедуры представлена на рисунке 2.19.

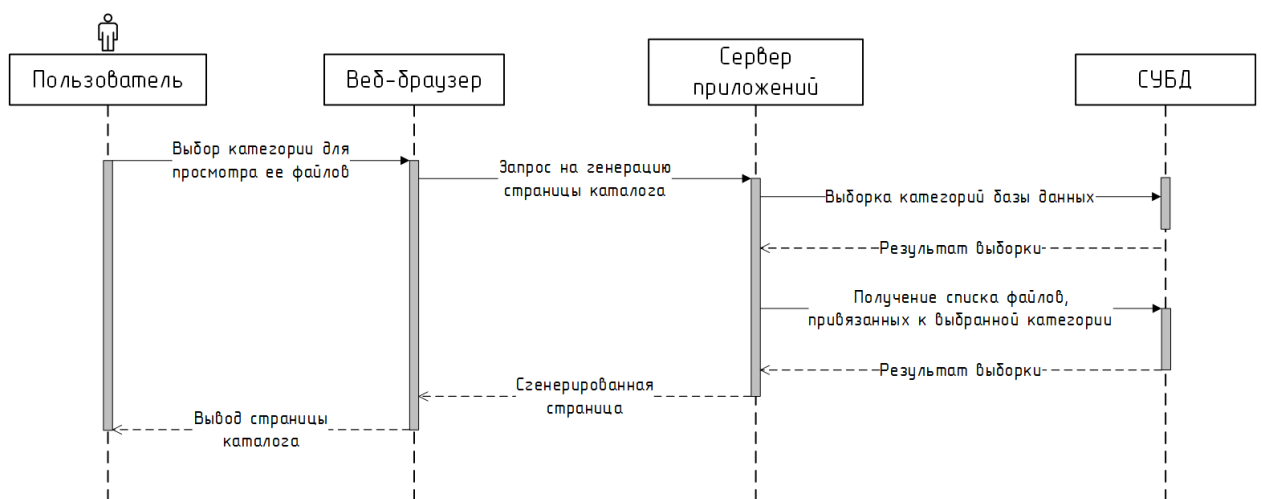


Рисунок 2.19 – Диаграмма последовательности процедуры просмотра каталога

Пользователь может выбрать одну категорию из списка всех категорий, для просмотра файлов, которые привязаны к этой категории. Если же пользователь не выбрал ни одной категории, что считается, что он находится в корневой категории, список файлов у которой всегда пустой. На сервере производится выборка дерева категорий из базы данных, для вывода списка категорий пользователю и, если пользователем была выбранная категория, получение информации о этой категории, а также, получение списка файлов, которые привязаны к этой категории. Диаграмма активности процедуры просмотра каталога представлена на рисунке 2.20.

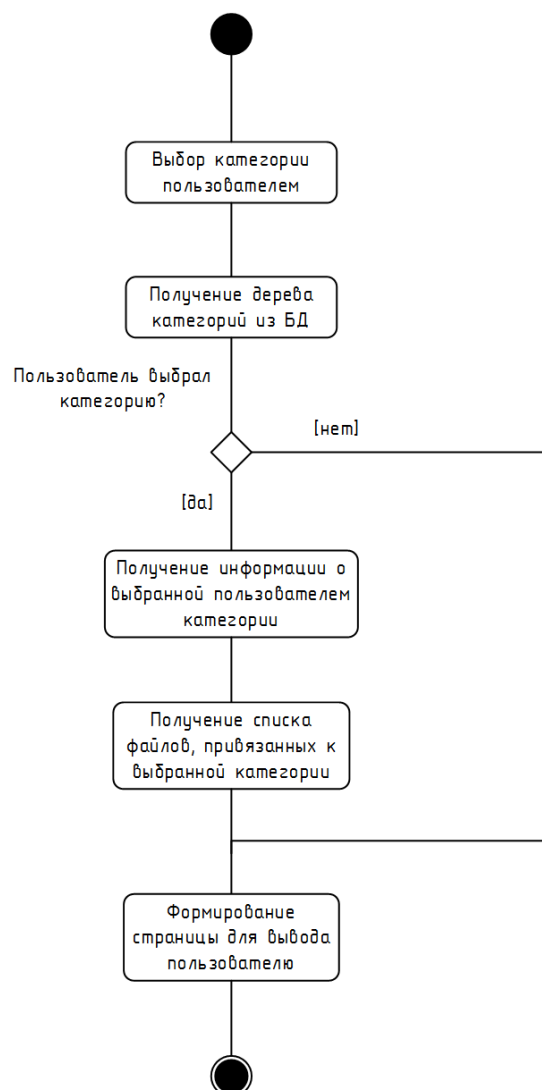


Рисунок 2.20 – Диаграмма активности процедуры просмотра каталога

Для того, чтобы модераторы и администраторы имели возможность наполнять каталог программного обеспечения файлами, необходима процедура добавления файлов в каталог программного обеспечения. Диаграмма последовательности данной процедуры представлена на рисунке 2.21.

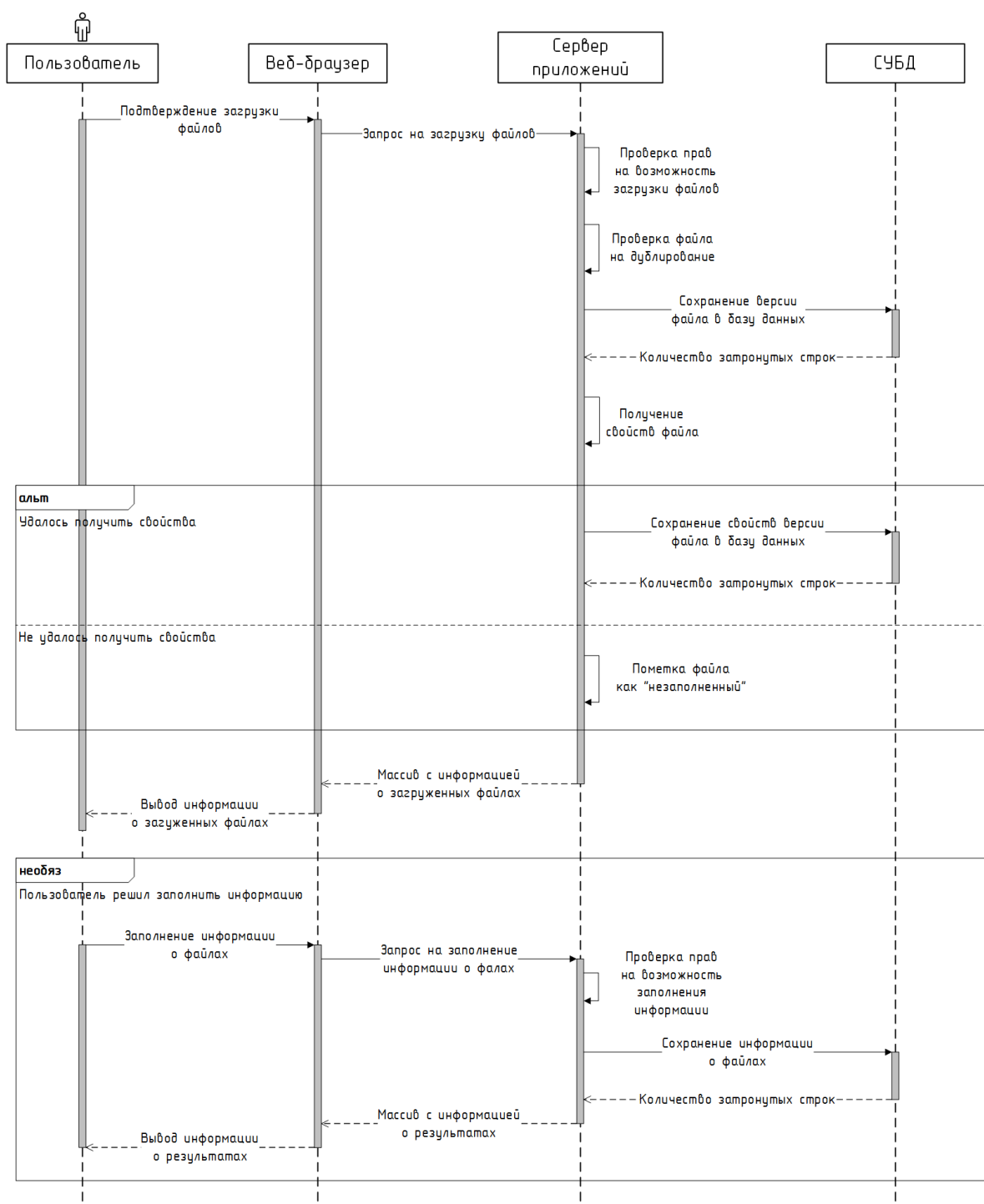


Рисунок 2.21 – Диаграмма последовательности процедуры добавления файлов

Пользователь выбирает в браузере файлы и отправляет их на сервер. Перед сохранением необходимо проверить наличие прав у пользователя на добавление файлов. Если у пользователя нет таких прав, то ему выводится сообщение об ошибке. Следующим этапом идет проверка каждого файла на дублирование. Для каждого файла, который не прошел проверку на дублирование, пользователю выводится сообщение о том, что такой файл уже существует. На следующем этапе, файлы, которых еще нет в системе, добавляются в нее. Затем, для каждого файла, у которого удалось извлечь свойства, свойства сохраняются в базу данных, и он помечается как «заполненный». Если у файла не удалось извлечь свойства, то он помечается как «незаполненный» файл, то есть информацию о нем необходимо заполнить вручную. Далее, пользователю выводится информация о каждом сохраненном файле и предоставляется возможность заполнить вручную информацию о файлах, помеченных как «незаполненные». Если пользователь решил ввести информацию о файлах, то алгоритм, в случае наличия прав модератора или администратора, сохраняет введенную информацию в базу данных и выводит пользователю сообщение о результате сохранения.

Диаграмма активности процедуры добавления файлов в каталог программного обеспечения представлена на рисунке 2.22.

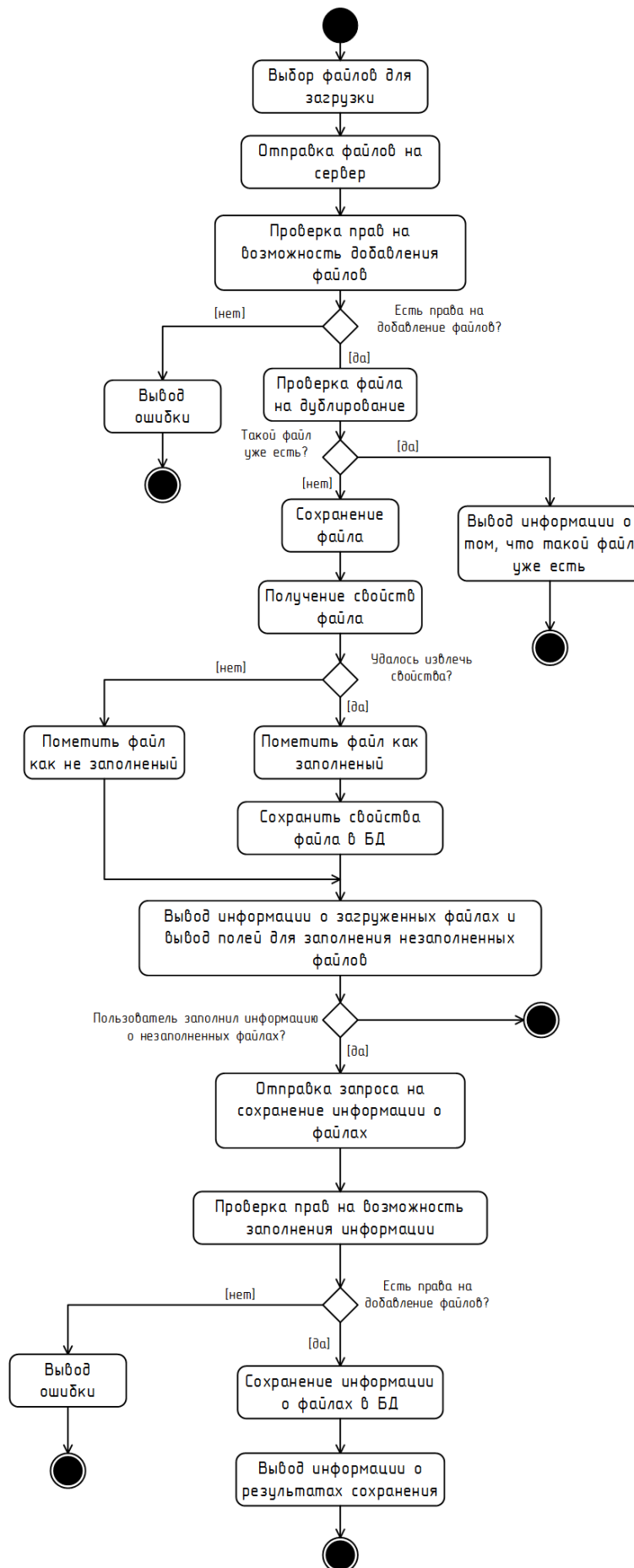


Рисунок 2.22 – Диаграмма активности процедуры добавления файлов

2.2.4 Разработка пользовательского интерфейса

Пользовательский интерфейс представляет собой совокупность программных и аппаратных средств, обеспечивающих взаимодействие пользователя с системой [8]. Основу такого взаимодействия составляют диалоги.

В соответствии с техническим заданием, для доступа к системе пользователи должны пройти сквозную аутентификацию через Active Directory. Если пользователь еще не прошел аутентификацию, то при запросе любой страницы он будет перенаправлен на страницу аутентификации. Так как на АО "ВПК "НПО машиностроения" уже есть системы, которые требуют аутентификацию пользователей, то было принято решение в разрабатываемой системе в качестве параметров аутентификации использовать те же параметры, что и в существующих системах, а именно служебный номер и пароль. Страница аутентификации является одинаковой для всех пользователей и располагается по адресу «/login».

Так как функции, которые предоставляются обычным пользователям и модераторам с администраторами, сильно различаются было принято решение сделать два вида меню: меню для обычных пользователей и меню для модератора или администратора. Остальные части страниц выглядят одинаково для любого пользователя. Меню для модератора или администратора располагается слева и содержит все необходимые ссылки. Внешний вид данного меню представлен на рисунке 2.23.

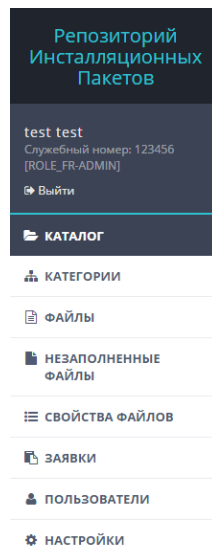


Рисунок 2.23 – Вид меню модератора и администратора

Меню для обычных пользователей содержит не так много ссылок, поэтому оно было вынесено вверх страницы, чтобы предоставить больше места для рабочей области страницы. Внешний вид такого меню представлен на рисунке 2.24.



Рисунок 2.24 – Вид меню обычного пользователя

При успешном завершении аутентификации пользователи попадают на страницу с каталогом программного обеспечения, которое хранится в системе. Данная страница является главной и располагается по адресу «/». Внешний вид страницы представлен на рисунке 2.25.

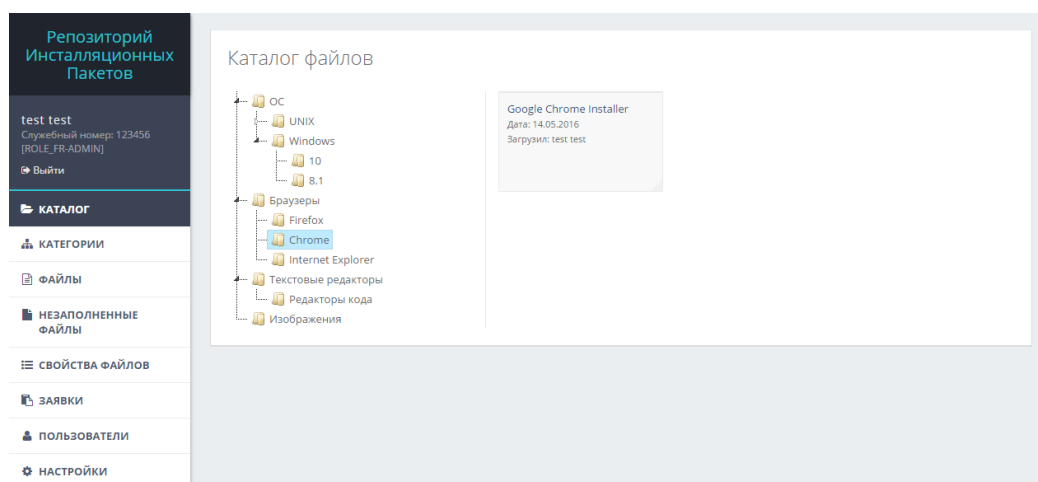


Рисунок 2.25 – Вид страницы каталога программного обеспечения

Каталог программного обеспечения представляет собой страницу, где слева расположен список категорий в системе в виде дерева, а справа – файлы, которые привязаны к выбранной категории. Выбор нужной категории осуществляется путем щелчка мышкой на нужную категорию в дереве категорий. Из каталога программного обеспечения, путем щелчка мыши на один из файлов, можно перейти на страницу просмотра информации о файле. На рисунке 2.26 представлен вид данной страницы для обычных пользователей, на рисунке 2.27 – для модераторов и администраторов.

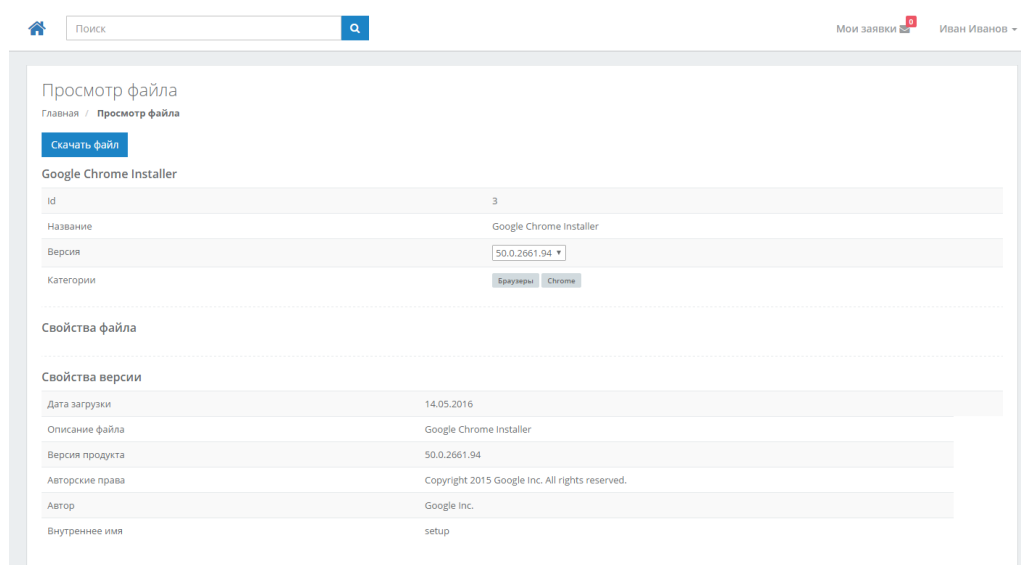


Рисунок 2.26 – Вид страницы просмотра информации о файле для обычных пользователей

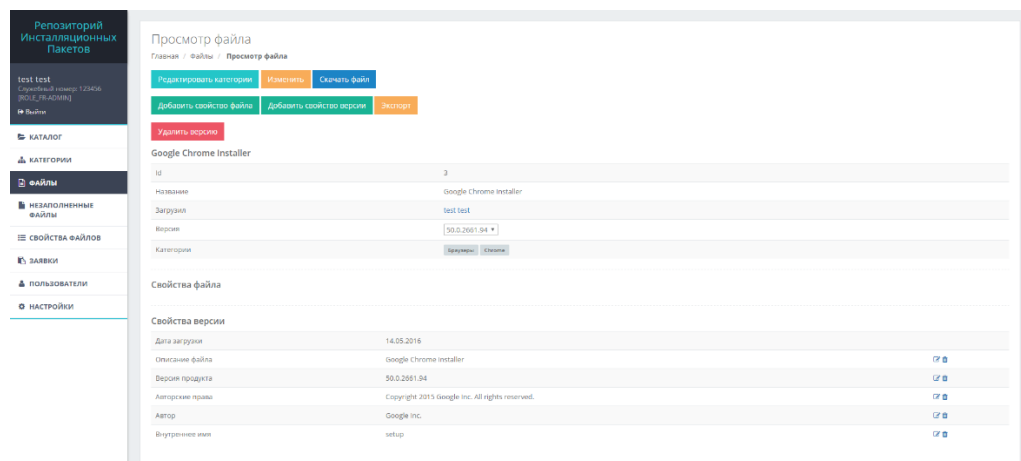


Рисунок 2.27 – Вид страницы просмотра информации о файле для администраторов и модераторов

На этой странице можно выбрать для просмотра другую версию файла, выгрузить файл с сервера. Администраторы и модераторы имеют дополнительные элементы управления для изменения информации о файле, удаления файла, добавление и удаления категорий, добавления и удаления свойства, выполнения команд для экспорта.

При нажатии кнопки «редактировать категории» на странице просмотра файла открывается страница редактирования списка категорий, к которым прикреплен файл. Эта страница расположена по адресу «/file-categories», ее внешний вид приведен на рисунке 2.28.

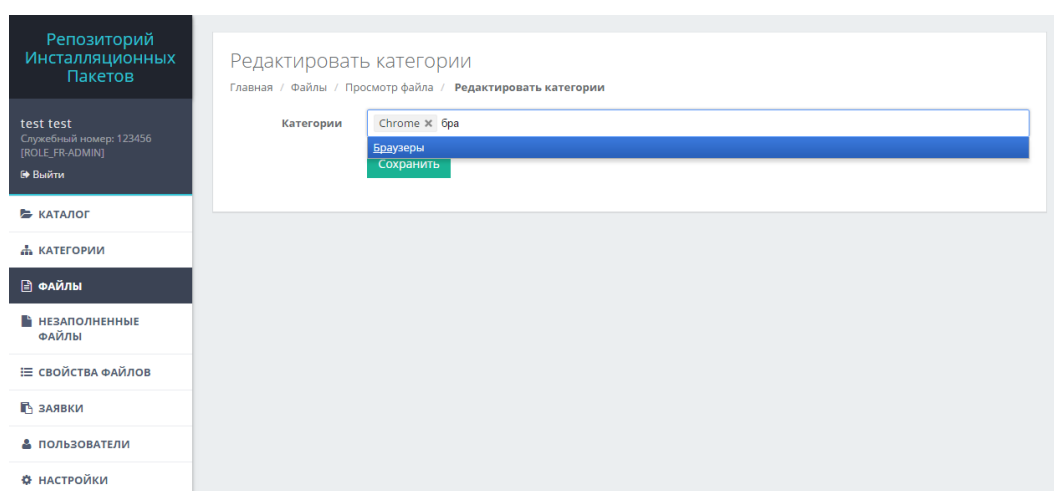


Рисунок 2.28 – Вид страницы редактирования списка категорий для файла

Редактировать список категорий могут только администраторы и модераторы. Пользователь может выбрать необходимую для добавления категорию из выпадающего списка, или начать вводить название категории, после чего в списке останутся только те категории, в названии которых присутствует введенный текст. После нажатия кнопки «сохранить» список категорий у файла будет обновлен.

При нажатии кнопки «изменить» на странице просмотра файла открывается страница заполнения главных свойств файла: названия и номера версии. Эта страница расположена по адресу «/file-filling». Доступ к странице имеют только модераторы и администраторы. Для изменения информации о файле необходимо заполнить оба текстовых поля и нажать кнопку «Сохранить».

При нажатии «добавить свойство файла» или «добавить свойство версии» на странице просмотра файла откроются страницы «/file-property-add» и «/file-version-property-add» соответственно, где можно добавить новое свойство ко всем версиям файла сразу, или к конкретной версии.

Для добавления свойства пользователю необходимо выбрать нужное свойство из выпадающего списка и ввести его значение, после чего нажать кнопку «сохранить». Данная возможность доступна только модераторам и администраторам.

Для просмотра и редактирования списка категорий необходимо нажать на кнопку «категории» в левом меню, после чего откроется страница «/categories», внешний вид которой представлен на рисунке 2.29.

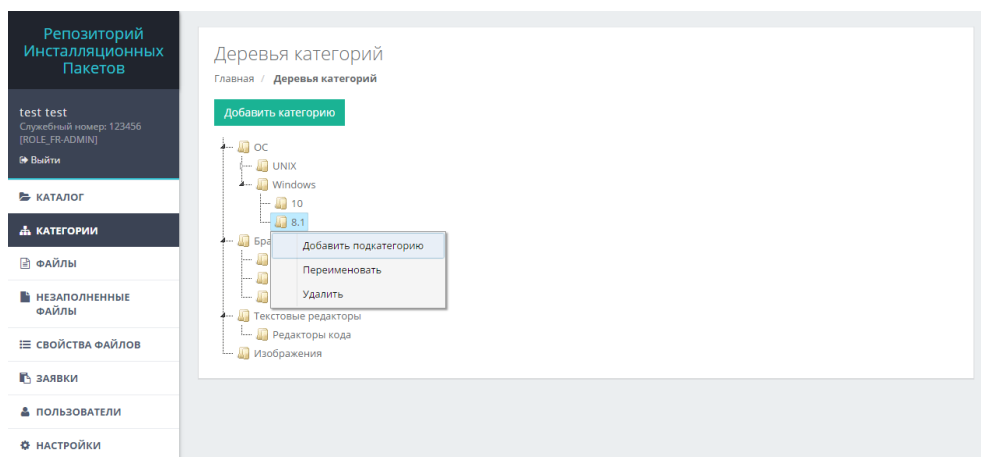
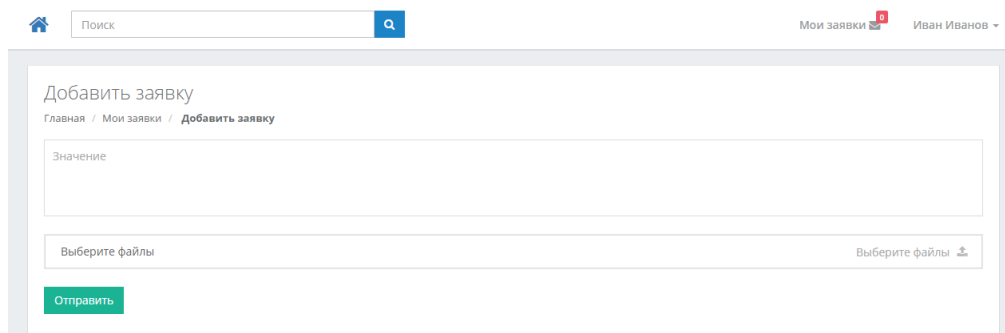


Рисунок 2.29 – Вид страницы списка категорий

На странице выводится дерево категорий, которое можно изменять с помощью операции «drag-and-drop». Пользователь может изменить порядок категорий, вкладывать категорию в категорию, переименовать категорию и удалить категорию. Для добавления новой категории необходимо нажать кнопку «добавить категорию», после чего, в появившемся поле, ввести название новой категории и нажать кнопку «добавить». Функции редактирования категорий доступны администраторам и модераторам.

Модераторы и администраторы могут редактировать список свойств. Для просмотра списка свойств необходимо нажать на кнопку «свойства файлов» в левом меню. Страница со списком существующих свойств расположена по адресу «/properties». Здесь можно добавить новое свойство, изменить существующие свойство и просмотреть подробную информацию о свойстве.

Обычные пользователи могут оставлять заявки на добавление ПО в каталог. Для этого необходимо перейти на страницу заявок, нажав на кнопку «мои заявки» в верхнем меню и, затем, нажать кнопку «добавить заявку». Вид страницы представлен на рисунке 2.30.



Добавить заявку

Главная / Мои заявки / Добавить заявку

Значение

Выберите файлы

Выберите файлы

Отправить

Рисунок 2.30 – Вид страницы добавления заявки

Пользователь должен выбрать список файлов, которые хочет прикрепить к заявке и указать комментарий, после чего, нажать кнопку «отправить». Добавленная заявка попадает на рассмотрение модератором.

Модераторы и администраторы могут просматривать список пользователей в системе. Для этого необходимо нажать кнопку

«пользователи» в левом меню. При нажатии на иконку просмотра детальной информации о пользователе открывается страница, расположенная по адресу «/user-view». Так же, на эту страницу можно попасть со страницы просмотра файла, нажав на имя пользователя, который добавил файл.

Для редактирования настроек, необходимо нажать на кнопку «настройки» в левом меню, после чего откроется страница «/settings». На этой странице находятся настройки соединения с БД, настройки соединения с Active Record и настройки путей сохранения файлов.

Выполнение скриптов происходит в четыре этапа. На первом этапе необходимо выбрать один из существующий шаблонов экспорта или ввести название для нового шаблона. Далее, на втором этапе, пользователю предлагается ввести список параметров, которые можно будет использовать на последнем этапе. При этом параметры могут быть представлены как название и значение, а также, как название, набор команд и регулярное выражение. Если параметр представлен как набор команд и регулярное выражение, то данный набор команд, при переходе на второй этап, будет выполнен, а на результат ее выполнения будет применено регулярное выражение. После применения регулярного выражения будут получены возможные варианты значения параметра. На третьем этапе, для таких параметров необходимо выбрать одно значение из всех возможных. На последнем этапе можно сформировать скрипт, при этом в тексте скрипта можно использовать заполненные ранее параметры, написав название параметра в фигурных скобках, например, {directory}. Так же, на последнем этапе доступны некоторые стандартные параметры, такие как, название файла, версия файла и другие.

2.3 Компоновка репозитория инсталляционных пакетов

После разработки базы данных, классов для работы с базой данных и пользовательского интерфейса был осуществлен процесс компоновки разработанных компонентов в исполняемый файл. Так как система

разрабатывалась на ЯП Java, то компоновка была осуществлена в файл формата «war» (Web Application Archive). Данный файл содержит внутри себя все необходимое, для запуска вер-приложения на веб-сервере. Диаграмма компоновки представлена на рисунке 2.31.

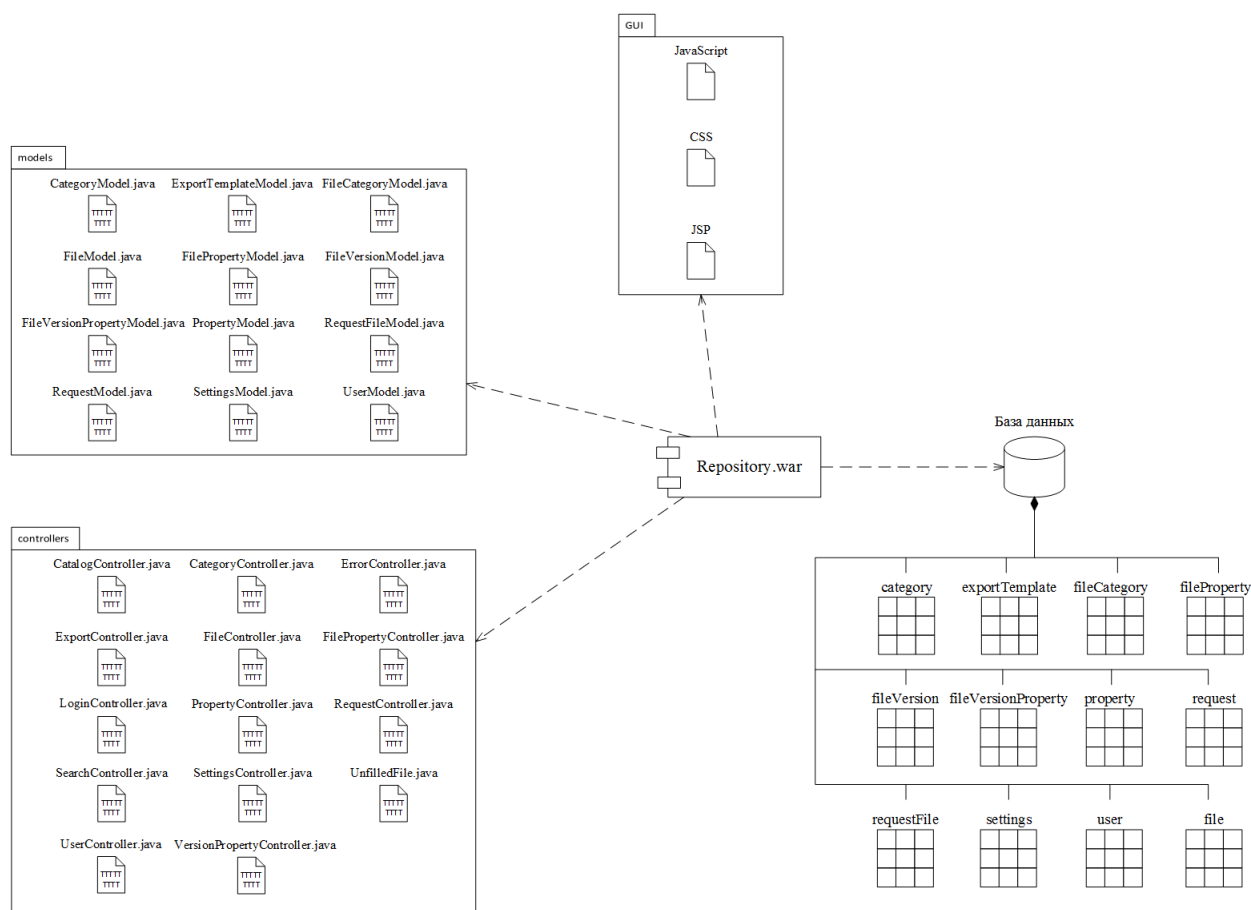


Рисунок 2.31 – Диаграмма компоновки компонентов системы

2.4 Выводы

В результате выполнения конструкторской части работы была разработана база данных. Для работы с базой данных были разработаны классы для сущностей, который были выделены в результате объектной декомпозиции. Для обеспечения взаимодействия с пользователем был разработан пользовательский интерфейс. Из разработанных компонентов был собран файл формата «war», который содержит внутри себя все необходимое, для запуска приложения на веб-сервере.

3 Тестирование репозитория инсталляционных пакетов

Тестирование – это процесс исполнения программы с целью обнаружения ошибок. Никакое тестирование не может доказать отсутствие ошибок в хоть сколько-нибудь сложном программном обеспечении. Для такого программного обеспечения выполнение полного тестирования, т. е. задания всех возможных комбинаций исходных данных, становится невозможным, а, следовательно, всегда имеется вероятность того, что в программном обеспечении остались не выявленные ошибки. Однако соблюдение основных правил тестирования и научно обоснованный подбор тестов может уменьшить их количество.

Согласно [8], процесс разработки программного обеспечения предполагает три стадии тестирования:

- автономное тестирование компонентов программного обеспечения;
- комплексное тестирование разрабатываемого программного обеспечения;
- системное или оценочное тестирование на соответствие основным критериям качества.

В качестве методов автономного тестирования компонентов системы используются различные подходы:

- Метод ручного контроля;
- Тестирование по принципу «белого ящика»;
- Тестирование по принципу «черного ящика».

Согласно [8], существуют следующие подходы для комплексного тестирования компонентов ПО:

- Восходящее тестирование;
- Нисходящее тестирование.

Восходящее тестирование предполагает, что каждый компонент тестируют отдельно на соответствие имеющимся спецификациям на него, затем собирают оттестированные компоненты в компоненты более высокой степени интеграции и тестируют их. При этом проверяют интерфейсы между компонентами, используемые для подключения компонентов более низкого уровня иерархии. И так далее, пока не будет собран весь программный продукт.

Нисходящее тестирование заключается в том, что, когда проектирование какого-либо компонента заканчивается, его кодируют и передают на тестирование. При его тестировании все вызываемые им компоненты заменяют компонентами, которые в той или иной степени имитируют поведение вызываемых компонентов. Такие компоненты принято называть «заглушками». В отличие от тестирующих программ заглушки очень просты, например, они могут просто фиксировать, что им передано управление. Часто заглушки просто возвращают какие-либо фиксированные данные. Как только тестирование основного компонента завершено, к нему подключают компоненты, непосредственно им вызываемые, и необходимые заглушки, а затем проводят их совместное тестирование. Далее последовательно подключают следующие компоненты, пока не будет собрана вся система.

При разработке репозитория инсталляционных пакетов первым делом разрабатывались классы моделей сущностей, которые были выделены в результате объектной декомпозиции, затем производилась разработка контроллеров, которые описывают бизнес-процессы системы. Параллельно с разработкой моделей и контроллеров велась разработка пользовательского интерфейса. При этом, контроллеры, которые должны были вызываться элементами пользовательского интерфейса были заменены на «заглушки». Таким образом, тестирование моделей и контроллеров происходило с использованием восходящего подхода, а тестирование пользовательского интерфейса – с использованием нисходящего подхода.

3.1 Тестирование методом ручного контроля

Методы ручного контроля предназначены для периода разработки, когда программа закодирована, но тестирование на машине еще не началось.

3.1.1 Тестирование методом инспекции исходного текста

Данный метод тестирования представляет собой один из методов ручного контроля. Согласно [8], инспекция исходного текста представляет собой набор процедур и приемов обнаружения ошибок при изучении текста группой специалистов. В эту группу входят: автор программы, проектировщик, специалист по тестированию и координатор - компетентный программист, но не автор программы. Общая процедура инспекции предполагает следующие операции:

- участникам группы заранее выдается листинг программы и спецификация на нее;
- программист рассказывает о логике работы программы и отвечает на вопросы инспекторов;
- программа анализируется по списку вопросов для выявления исторически сложившихся общих ошибок программирования.

Рассмотрим тестирование данным способом на примере проверки файлов на дублирование. Изначально, для определения дублирования файла, от нового файла бралась хэш-сумма, и производился поиск в базе данных файла с такой же хеш-суммой. В ходе ответов на вопросы инспектора было обнаружено, что одной хеш-суммы для проверки на дублирование недостаточно, так как для разных файлов возможны коллизии. Для исправления данной ошибки было принято решение осуществлять проверку на дублирование по двум параметрам: по хэш-сумме и по размеру файла.

3.1.2 Тестирование методом сквозных просмотров

Данный метод, согласно [8], представляет собой набор способов обнаружения ошибок, осуществляемых группой лиц, просматривающих текст программы, и состоит из следующих этапов:

1. Участникам группы заранее выдается листинг программы, блок схема и спецификация на нее;
2. Участникам заседания предлагается несколько тестов, написанных на бумаге, и тестовые данные подвергаются обработке в соответствии с логикой программы (каждый тест мысленно выполняется);
3. Программисту задаются вопросы о логике проектирования и принятых допущениях;
4. Состояние программы (значения переменных) отслеживается на бумаге или доске.

Рассмотрим тестирование данным методом на примере процедуры добавления нового файла в каталог программного обеспечения. Алгоритм данной процедуры представлен на рисунке 3.1, листинг приведен в приложении В. Спецификация алгоритма:

Пользователь отправляет на сервер несколько файлов. Процедура должна проверить наличие прав у пользователя на загрузку файлов. Если у пользователя нет прав на загрузку файлов, то необходимо вывести сообщение об ошибке. Если у пользователя есть права – то для каждого файла необходимо убедиться, что такого файла в системе еще нет. Если файл уже существует, то необходимо вывести об этом сообщение пользователю. Если такого файла нет, то файл нужно сохранить на диск и добавить запись о файле в базу данных. Затем, необходимо прочесть свойства файла и, если они заполнены, сохранить их в базу данных. В конце, необходимо вывести пользователю информацию о файлах, которые были добавлены в систему и предложить пользователю заполнить информацию о файлах, свойства которых получить не удалось.

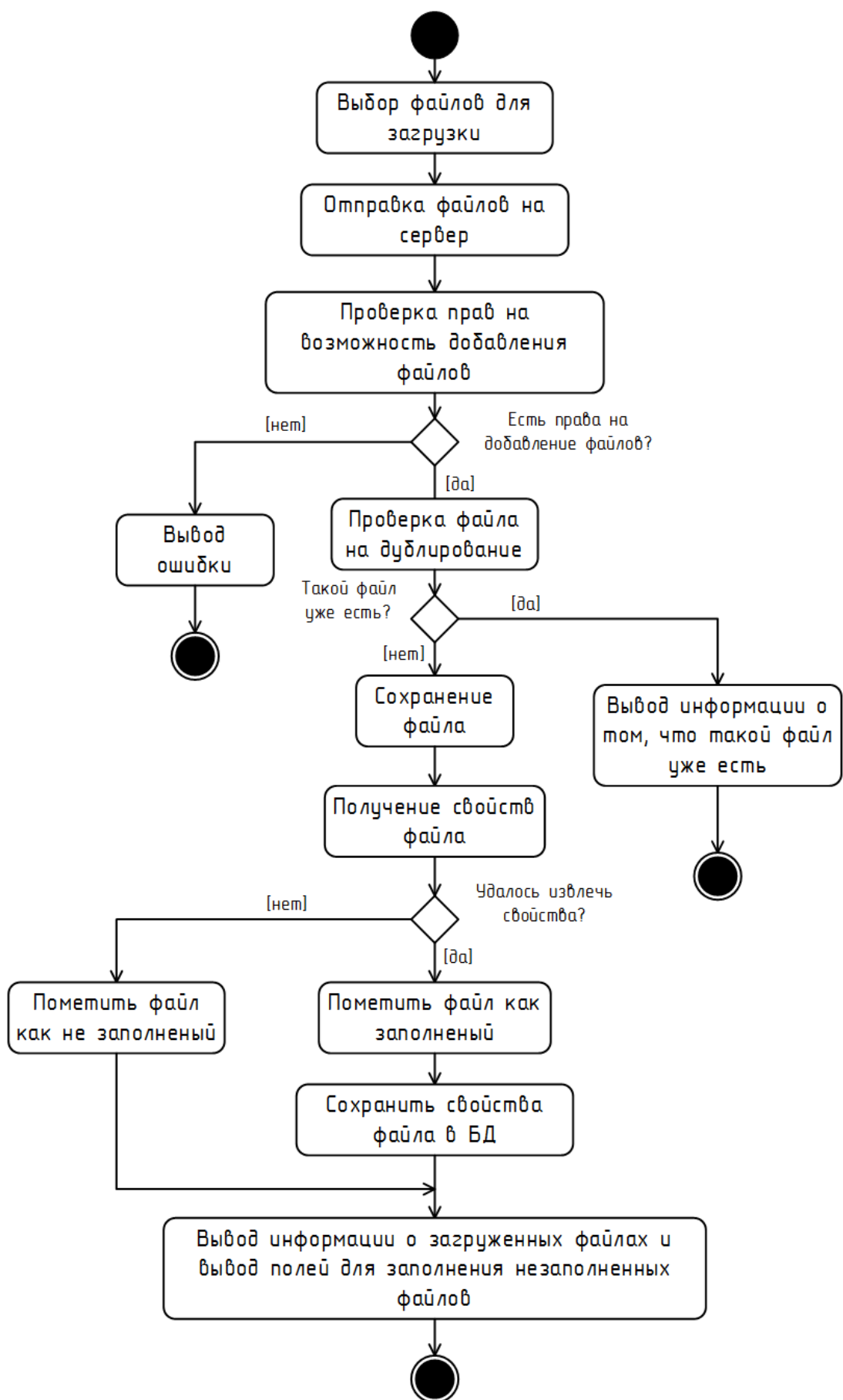


Рисунок 3.1 – Схема процедуры добавления файла в каталог

В результате анализа схемы и листинга программы была обнаружена ошибка. При отправке пользователем файлов для добавления в каталог программного обеспечения они помещаются во временную папку. Затем, каждый файл открывается для копирования содержимого файла в новый файл рабочей директории приложения. Ошибка заключалась в том, что после завершения копирования содержимого, файл не закрывался, что приводило к невозможности удаления временных файлов.

3.2 Тестирование по принципу «белого ящика»

Согласно [8], Стратегия тестирования по принципу «белого ящика», или стратегия тестирования, управляемая логикой программы (с учетом алгоритма), позволяет проверить внутреннюю структуру программы. В этом случае тестирующий получает тестовые данные путем анализа логики программы.

Стратегия «белого ящика» включает в себя следующие методы тестирования:

- Покрытие операторов
- Покрытие решений
- Покрытие условий
- Покрытие решений/условий
- Комбинаторное покрытие условий

При тестировании по принципу «белого ящика» использовался метод комбинаторного покрытия условий, так как данный метод охватывает решений и условий. Данный метод требует создания такого множества тестов, чтобы все возможные комбинации результатов условий в каждом решении и все операторы выполнялись, по крайней мере, один раз.

Рассмотрим тестирование методом комбинаторного покрытия условий на примере процедуры добавления нового файла в каталог программного обеспечения, алгоритм которого представлен на рисунке 33. В данном возможны следующие комбинации:

1. У пользователя нет прав на добавление файлов;
2. У пользователя есть права на добавление файлов;
3. Файл уже существует;
4. Файл еще не существует;
5. У файла заполнены свойства;
6. У файла не заполнены свойства.

Для каждой комбинации необходимо провести тестирование, то есть необходимо провести 6 тестов.

В результате проведенных тестов, было обнаружено, что в случае загрузки файлов форматов, отличных от Portable Executable (OC Windows), возникала ошибка, приводящая к неожиданному завершению работы. Для исправления данной ошибки в функцию получения свойств файла была добавлена проверка файла на принадлежность формату Portable Executable. Алгоритмы работы функции получения свойств файла до и после исправления ошибки представлены на рисунках 3.2 и 3.3 соответственно.

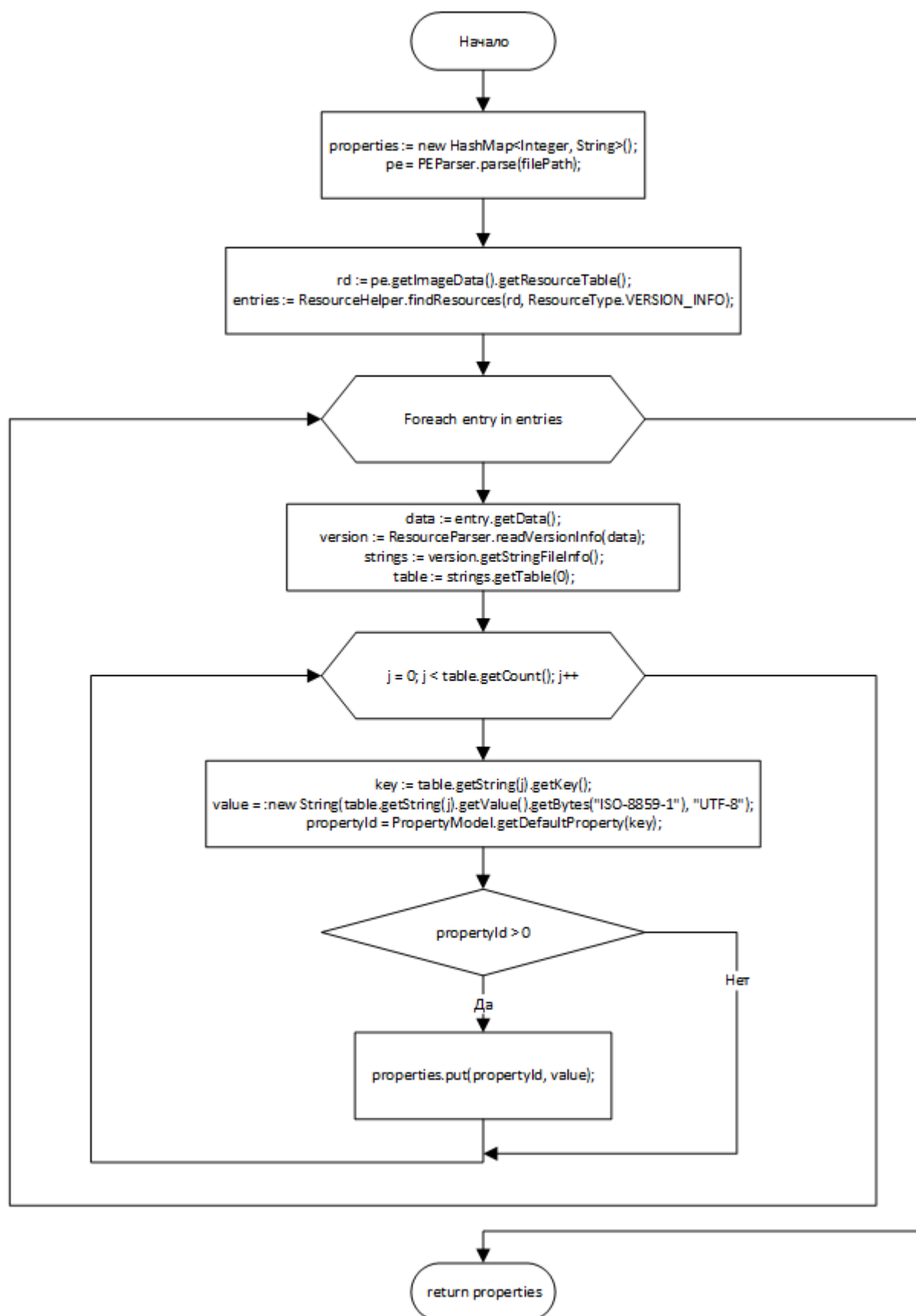


Рисунок 3.2 – Блок схема алгоритма получения свойств файла до исправления ошибки

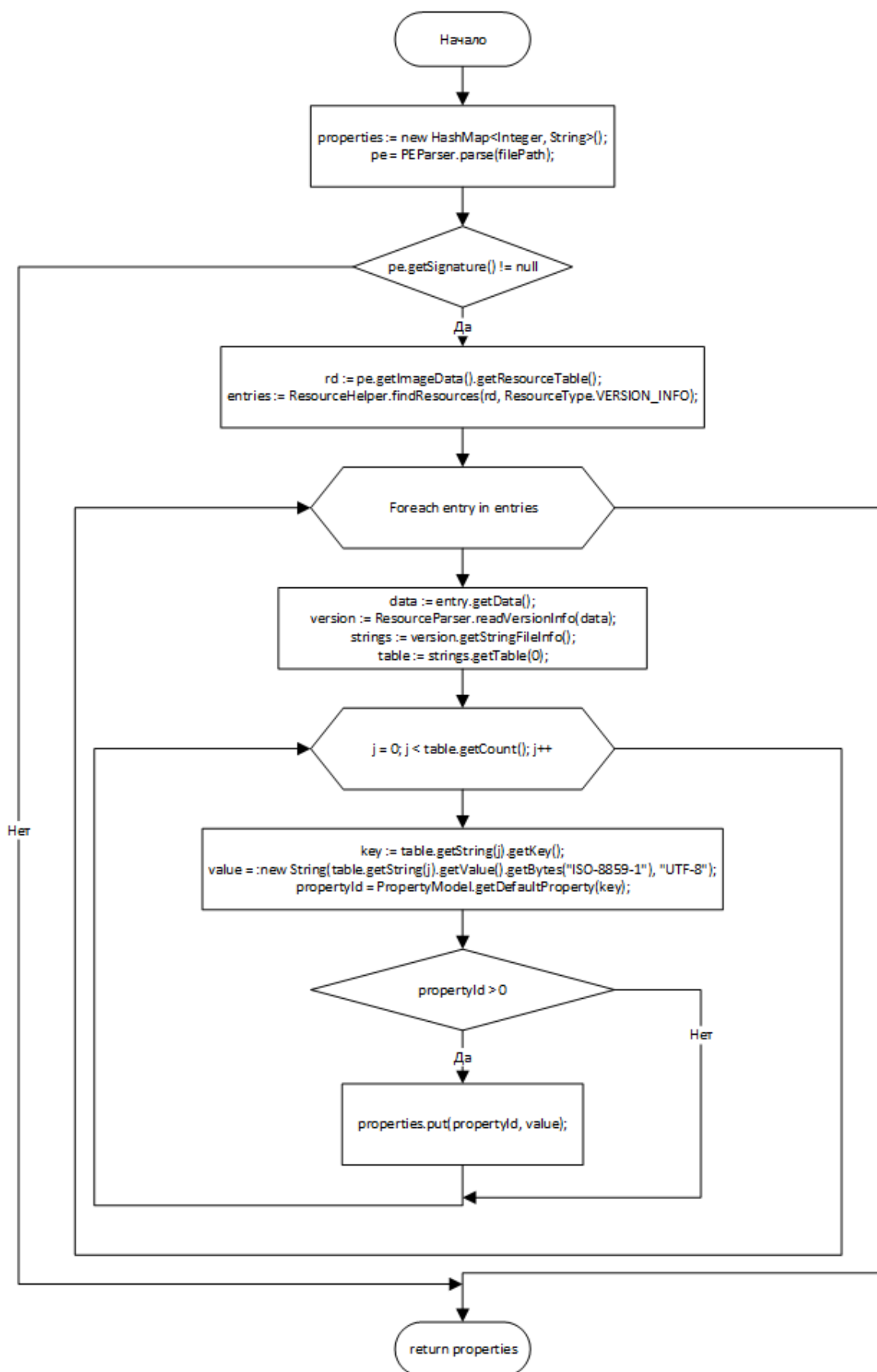


Рисунок 3.3 – Блок схема алгоритма получения свойств файла после исправления ошибки

3.3 Тестирование по принципу «черного ящика»

При тестировании данным методом программа рассматривается как «черный ящик», и целью тестирования является выяснение обстоятельств, в которых поведение программы не соответствует спецификации. Для обнаружения всех ошибок в программе, необходимо выполнить исчерпывающее тестирование, т. е. тестирование на всех возможных наборах данных. Для программ, в которых исполнение команды зависит от предшествующих ей событий, необходимо проверить и все возможные последовательности. Очевидно, что проведение исчерпывающего тестирования для подавляющего большинства случаев невозможно. Поэтому обычно выполняют «разумное» или «приемлемое» тестирование, которое ограничивается прогонами программы на небольшом подмножестве всех возможных входных данных. Этот вариант не дает гарантии отсутствия отклонений от спецификаций.

Согласно [8], различают следующие методы формирования тестовых наборов:

- эквивалентное разбиение;
- анализ граничных значений;
- анализ причинно-следственных связей;
- предположение об ошибке.

Метод эквивалентного разбиения заключается в следующем. Исходные данные разбиваются на конечное число групп – классов эквивалентности. Наборы данных такого класса объединяют по принципу обнаружения одних и тех же ошибок: если набор какого-либо класса обнаруживает некоторую ошибку, то предполагается, что все другие тесты этого класса эквивалентности тоже обнаружат эту ошибку и наоборот. Так же, каждый тест должен включать по возможности максимальное количество различных входных условий, что позволяет минимизировать общее число необходимых тестов.

Анализ граничных значений. Граничные условия - это ситуации, возникающие на, выше или ниже границ входных классов эквивалентности. Анализ граничных значений отличается от эквивалентного разбиения следующим:

- выбор любого элемента в классе эквивалентности в качестве представительного при анализе граничных условий осуществляется таким образом, чтобы проверить тестом каждую границу этого класса;
- при разработке тестов рассматриваются не только входные условия (пространство входов), но и пространство результатов.

Анализ причинно-следственных связей. Метод анализа причинно-следственных связей помогает системно выбирать высоко результативные тесты. Он дает полезный побочный эффект, позволяя обнаруживать неполноту и неоднозначность исходных спецификаций. Спецификация разбивается на «рабочие» участки. В спецификации определяются множество причин и множество следствий. Причина – это отдельное входное условие или класс эквивалентности входных условий. Следствие – это выходное условие или преобразование системы. Каждым причине и следствию приписывается отдельный номер. На основе анализа семантического содержания спецификации строится таблица истинности, в которой последовательно перебираются все возможные комбинации причин и определяются следствия каждой комбинации причин.

Предположение об ошибке. Основная идея метода состоит в том, чтобы перечислить в некотором списке возможные ошибки или ситуации, в которых они могут появиться, а затем на основе этого списка составить тесты.

Рассмотрим тестирование на примере привязки свойства к версии файла. Исходными данными для связи версии файла со свойством являются:

- ID версии файла;

- ID свойства;
- Значение свойства, которое мы хотим привязать.

При использовании метода эквивалентных разбиений, классами эквивалентности будут передаваемые параметры:

- ID версии файла. Правильный класс эквивалентности – это такое ID, которое существует в таблице базы данных fileVersion. Неправильный класс эквивалентности – остальные числа, которые не существуют в таблице fileVersion.
- ID свойства. Аналогично ID версии, правильный класс эквивалентности – это такое ID, которое существует в таблице базы данных property. Неправильный класс эквивалентности – остальные числа, которые не существуют в таблице property.
- Значение свойства. Значение свойства может быть любой текстовой строкой, длина которой не должна превышать 255 символов.

При проведении тестирования было установлено, что отсутствует проверка длины строки у входного параметра «значение свойства», что приводило к ошибке при сохранении информации в базу данных.

По методу анализа граничных условий, граничными условиями для исходных данных будут:

- У параметра ID версии файла граничные условия отсутствуют.
- У параметра ID свойства граничные условия отсутствуют.
- Значение свойства. Граничным условием является длина строки, равная 255 символам.

При проведении тестирования методом граничных условий было обнаружено, что строка, длина которой равна 255 символов, считается не допустимой, и пользователю выдавалась ошибка.

По методу анализа причинно-следственных связей можно выделить две причинно-следственные связи: одна – вывод пользователю сообщения об ошибке правильности исходных данных, вторая – вывод пользователю сообщения об успешном добавлении свойства к версии файла. Таблица истинности процедуры привязки свойства к файлу представлена в таблице 3.1.

Таблица 3.1 – Таблица истинности привязки свойства к файлу

ID версии существует	ID свойства существует	Значение меньше или равно 255 символам	Результат
0	0	X	Сообщение об ошибке
1	0	X	Сообщение об ошибке
0	1	X	Сообщение об ошибке
1	1	0	Сообщение об ошибке
1	1	1	Сообщение о успешном завершении

При тестировании, для обеспечения полной проверки исходных данных, были проверены все 5 приведенных выше варианта.

3.4 Оценочное тестирование

После завершения комплексного тестирования приступают к оценочному тестированию. Целью данного тестирования является тестирование программы на соответствие основным требованиям. В рамках данного проекта были проведены следующие виды тестирования:

- тестирование удобства использования;
- тестирование на предельных объемах;
- тестирование защиты;
- тестирование удобства установки;
- тестирование совместимости;
- тестирование документации.

При проведении тестирования на удобство использования было установлено, что у пользователей, не имеющих прав модератора или администратора, в левом меню, которое занимает достаточно много места, слишком мало элементов. В результате, было принято решение для обычных пользователей сделать другой вид меню, который располагается сверху, во всю ширину страницы. Такое меню стало занимать намного меньше места, при этом, не потеряв в функциональности.

При проведении тестирования на предельных объемах было одновременно, одним пользователем, добавлено в каталог большое количество файлов. В результате, была обнаружена ошибка, которая заключалась в том, что после работы с базой данных, соединение с ней не закрывалось. Эта ошибка приводила к тому, что база данных выдала ошибку, сообщая о том, что достигнуто максимальное количество соединений.

После тестирования защиты было установлено, что любой пользователь может перейти на страницу просмотра категорий, которая должна быть доступна только модераторам и администраторам. Аналогичная ошибка была обнаружена на странице экспорта файла. Данная страница должна быть доступна только администраторам, но модераторы так же имели к ней доступ.

После проведения тестирования удобства установки были выявлено, что редактирование настроек соединения с базой данных и с Active Directory необходимо менять после каждой перезагрузки сервера. Это происходило потому, что файлы настроек лежали внутри архива приложения. Для решения

этой проблемы было решено при запуске приложения проверять наличие файлов настроек в домашней директории пользователя и, если этих файлов нет, создавать их и предлагать пользователю настроить параметры соединения.

В рамках тестирования совместимости производился запуск разработанного приложения на различных операционных системах, а именно, на Windows 10, Mac OS X и Ubuntu 14.04. В результате данного тестирования было установлено, что на всех приведенных операционных системах приложение работает в нормальном режиме, и никаких ошибок выявлено не было.

3.5 Выводы

В данном разделе было произведено тестирования разработанного приложения различными методами: методом инспекции исходного текста, методом сквозных просмотров, методом белого ящика и методом черного ящика. Затем было произведено оценочное тестирование разработанной системы.

4 Техничко-экономическое обоснование разработки

В данной главе дипломного проекта выполняется технико-экономическое обоснование разработки репозитория инсталляционных пакетов с целью определения этапов работ, связанных с процессом разработки программной системы. Основными задачами, которые решаются на данном этапе, являются:

- Определение этапов работ по создания программного обеспечения;
- Расчет трудоемкости проекта;
- Определение численности исполнителей;
- Построение сетевого графика выполнения проекта;
- Построение календарного графика выполнения работ;
- Расчет затрат на разработку проекта.

4.1 Расчет трудоемкости проекта

Расчет трудоемкости проекта необходим для определения затрат на реализацию репозитория инсталляционных пакетов. Через трудоемкость в конечном итоге оценивается один из основных затратных показателей – совокупные затраты на оплату труда исполнителей проекта.

Полный перечень этапов работ выполнения проекта, а также трудоемкость каждого этапа представлен в таблице 4.1.

Таблица 4.1 – Перечень этапов выполнения проекта

Этап	№ работы	Содержание работы	Трудоемкость t	
			чел. час	чел. дни
1 Подготовительный этап	1	Анализ требований ТЗ	4,2	0,525
2 Проектирование программного обеспечения	1	Уточнение требований	3,8	0,475
	2	Выбор архитектуры	12	1,5
	3	Выбор программной платформы	9,6	1,2
3 Реализация компонентов системы	1	Разработка базы данных	3,8	0,475
	2	Разработка классов	4,8	0,6
	3	Написание кодов программы на выбранном языке программирования	170	21,25
4 Тестирование программного обеспечения, завершение работы	1	Тестирование	7	0,875
	2	Внесение поправок	44	5,5
5 Подготовка документации	1	Подготовка сопроводительной документации	19	2,375

В таблице 4.2 приведена экспертная оценка продолжительностей работ разработки репозитория инсталляционных пакетов.

Таблица 4.2 – Экспертная оценка продолжительностей работ

№ этапа	№ работы	T_{\min} (чел. час)	T_{\max} (чел. час)	$t_{ож}$ (чел. час)
1	1	3	6	4,2
2	1	3	5	3,8
	2	10	15	12
	3	8	12	9,6
3	1	3	5	3,8
	2	4	6	4,8
	3	150	200	170
4	1	5	10	7
	2	40	50	44
5	1	15	25	19

Общая трудоемкость разработки проекта Q определяется как совокупные затраты труда на выполнение каждого из его этапов:

$$Q = \sum t_i, \quad (4.1)$$

где t_i - затраты труда на выполнение i -го этапа проекта. После подстановки в формулу (4.1) продолжительностей работ каждого этапа, была получена общая трудоемкость $Q = 278.2$ чел. час. = 34.775 чел. дней.

4.2 Определение численности исполнителей

Для оценки возможности выполнения проекта, в соответствии с имеющимся в распоряжении разработчика штатным составом исполнителей, рассчитывается их средняя численность. Она определяется соотношением:

$$N = \frac{Q}{F}, \quad (4.2)$$

где Q – затраты труда на выполнение проекта, чел/мес.;

F – фонд рабочего времени, мес.

Величина фонда рабочего времени определяется соотношением:

$$F = T \cdot F_M, \quad (4.3)$$

где T - время выполнения проекта, мес. (как правило, устанавливается заказчиком проекта);

F_M - фонд времени в текущем месяце, ч. (мес.), который рассчитывается из учета общего числа дней в году, числа выходных и праздничных дней:

$$F_M = \frac{t_p \cdot (D_K - D_B - D_{\Pi})}{12}, \quad (4.4)$$

где t_p - продолжительность рабочего дня, ч.;

D_K - общее число дней в году;

D_B - число выходных дней в году;

D_{Π} - число праздничных дней в году.

Согласно (4.4), фонд времени в текущем месяце

$$F_M = \frac{8 \cdot (366 - 101 - 18)}{12} = 165. \text{ Из формулы (4.3), фонд рабочего времени } F = 2 \cdot$$

165 = 330 часов. Согласно (4.2), средняя численность исполнителей $N =$

$$\frac{278.2}{330} = 0.843. \text{ Следовательно, для выполнения данной работы достаточно 1}$$

специалиста.

4.3 Построение сетевого графика выполнения проекта

Сетевой график – это графическое представление логической последовательности работ в целях выполнения проекта. Сетевой график устанавливает взаимосвязь между всеми работами проекта и позволяет определить продолжительность как отдельных этапов, так и всего проекта в целом.

Построение сетевого графика предполагает использование метода сетевого планирования, на базе которого разрабатывается информационно-динамическая модель процесса выполнения проекта. Построение сетевой модели включает оценку степени детализации комплекса работ, определения логической связи между отдельными работами и временные характеристики выполнения этапов проекта. В сетевой модели выделяют события и работы. В качестве событий, например, принимают факты начала проекта, окончания разработки отдельных модулей, интерфейсов, выполнения отладки и т.п. Все события нумеруются по порядку от исходного к завершающему.

В процессе достижения каждого из событий реализуется определенная последовательность работ. Конечным событием является выполнение всего проекта по разработке программного обеспечения. Каждой работе присваивается «код работы», состоящий из номера наступившего события и номера того события, которое достигается в результате выполнения данной работы, например, если 0 - начало проекта, а 1 - событие "разработка структуры данных завершена", то 0-1 - определяет работу по разработке структуры данных. В качестве работы может выступать и "фиктивная работа",

которая определяет ожидание окончания связанных работ и длительность которой равна 0 чел.-дней. Кодовые номера работ каждого этапа указываются в соответствующем блоке строк, относящегося к этому этапу.

Основные события и работы по разработке репозитория инсталляционных пакетов представлены в таблице 4.3.

Таблица 4.3 – Основные события и работы проекта

№	Событие	Код работы	Содержание работ	Трудоемкость, чел/часы.	Трудоемкость, чел/дни.
0	Начало работ	0-1	Анализ требований ТЗ	4,2	0,525
1	Проанализированы требования ТЗ	1-2	Уточнение требований	3,8	0,475
		1-3	Выбор архитектуры	12	1,5
2	Уточнены требования	2-3	Фиктивная работа. Ожидание завершения работы 1-3	0	0
3	Анализ завершен	3-4	Выбор программной платформы	9,6	1,2
4	Выбрана программная платформа	4-5	Разработка базы данных	3,8	0,475
		4-6	Разработка классов	4,8	0,6
5	Завершена разработка базы данных	5-6	Фиктивная работа. Ожидание завершения работы 4-6	0	0
6	Завершена разработка БД и классов	6-7	Написание кодов программы на выбранном языке программирования	170	21,25

Продолжение таблицы 4.3

1	2	3	4	5	6
7	Завершено написание кодов программы	7-8	Тестирование	7	0,875
8	Завершено тестирование	8-9	Внесение поправок	44	5,5
9	Внесены поправки	9-10	Подготовка сопроводительной документации	19	2,375

Графическое отображение сетевой модели (сетевой график) содержит окружности, которые отображают основные события проекта, и векторы, соединяющие эти окружности, которые определяют необходимость выполнения соответствующих работ. Реальные работы изображаются сплошной линией, фиктивные – штриховой, а работы, лежащие на критическом пути – линией двойной толщины.

Окружности разделены на четыре сектора, в каждом из которых показаны номер данного события (в нижнем секторе), значение раннего срока наступления текущего события (в левом секторе), значение резерва времени текущего события (в верхнем секторе) и значение позднего срока наступления события (в правом секторе), как показано на рисунке 4.1.

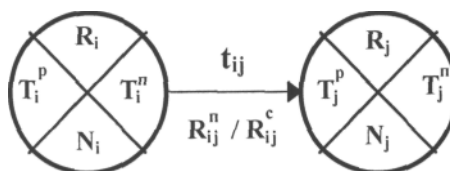


Рисунок 4.1 – Обозначение основных элементов сетевого графика

Обозначение основных элементов сетевого графика:

- N_i, N_j – номер события;

- T_i^P – ранний срок наступления события i ;
- $T_i^П$ – поздний срок наступления события i ;
- R_i – резерв времени события i ;
- t_{ij} - продолжительность работы $i-j$.

В соответствии с содержанием таблицы основных событий и работ проекта строится графическая модель сетевого графика.

Ранний срок совершения события определяет минимальное время, необходимое для выполнения всех работ, предшествующих данному событию и равен продолжительности наибольшего из путей, ведущих от исходного события к рассматриваемому, и рассчитывается при движении по графу слева направо (от начала проекта к его завершению) по формуле:

$$T_j^P = \max(T_i^P + t_{i-j})$$

Таким образом, двигаясь по графу слева направо, необходимо для каждого из предшествующих событий выбрать то, для которого сумма раннего срока и длительности работы, соединяющей его с данным событием, максимальна. Результаты расчета приведены ниже на сетевом графе.

Поздний срок совершения события – это максимально допустимое время наступления данного события, при котором сохраняется возможность соблюдения ранних сроков наступления последующих событий. Поздние сроки вычисляются, начиная с последнего события – завершения проекта, по критическому пути (т.е. справа налево по графику). Они равны разности между поздним сроком совершения j -го события и продолжительностью ij работы. Поздний срок совершения события определяется по формуле:

$$T_i^П = \min(T_j^П - t_{i-j}).$$

Резерв времени события определяется по формуле:

$$R_i = T_i^{\Pi} - T_i^P.$$

Полный резерв времени работы – это максимальное время, на которое можно отложить начало работы или увеличить ее длительность, чтобы, тем не менее, уложиться в сроки проекта. Полный резерв времени работы определяется по формуле:

$$R_{i-j}^{\Pi} = T_j^{\Pi} - T_i^P - t_{i-j} \quad (4.5)$$

Из формулы (4.5) видно, что превышение опоздания в процессе выполнения работы $i-j$ (или до ее начала) значения (4.5) приведет к «сползанию» дальнейших работ на разность между этим опозданием и полным резервом (4.5), что в конечном итоге означает запаздывание и последнего события, обозначающего полное завершение проекта.

Свободный резерв времени определяется согласно формуле (4.6) и обозначает время, на которое можно отложить начало работы или увеличить ее продолжительность, чтобы, тем не менее, успеть завершить проект в срок, при условии, что все работы будут выполняться по наиболее оптимистичному графику (в свои ранние сроки):

$$R_{i-j}^C = T_j^P - T_i^P - t_{i-j} \quad (4.6)$$

Формула (4.6) показывает, что даже в случае запаздывания, превышающего значение полного резерва (4.5), возможно уложиться в срок разработки проекта, «наверстав» упущенное время за счет оптимального выполнения последующих работ.

Таким образом:

- если отставание от графика не превышает полного резерва времени, у руководителя проекта нет причин для беспокойства;
- если отставание превышает полный резерв времени, то существует возможность догнать график за счет оптимального выполнения всех оставшихся работ;
- если отставание превышает свободный резерв времени, то догнать график в рамках текущего плана невозможно. Требуются дополнительные ресурсы.

Путь сетевого графика – это непрерывная цепочка работ и событий. Полный путь – сумма всех работ от исходного до завершающего события. Критический путь – это самый продолжительный из полных путей. Критический путь определяет продолжительность всего проекта.

Сетевой график работ по разработке репозитория инсталляционных пакетов представлен на рисунке 4.2. На данном сетевом графике критическим является путь 0-1-3-4-6-7-8-9-10. Его продолжительность равна 33,825 чел. дням.

4.4 Построение календарного графика выполнения проекта

Для иллюстрации последовательности проводимых работ проекта применяют ленточный график (календарно-сетевой график или диаграмму Гантта). На диаграмме Гантта на оси X показывают календарные месяцы от начала проекта до его завершения. По оси Y - выполняемые этапы работ.

Если отдельные этапы проекта могут выполняться параллельно различными исполнителями, то они отображаются в виде нескольких нумерованных отрезков (или прямоугольников), размещенных на временных интервалах.

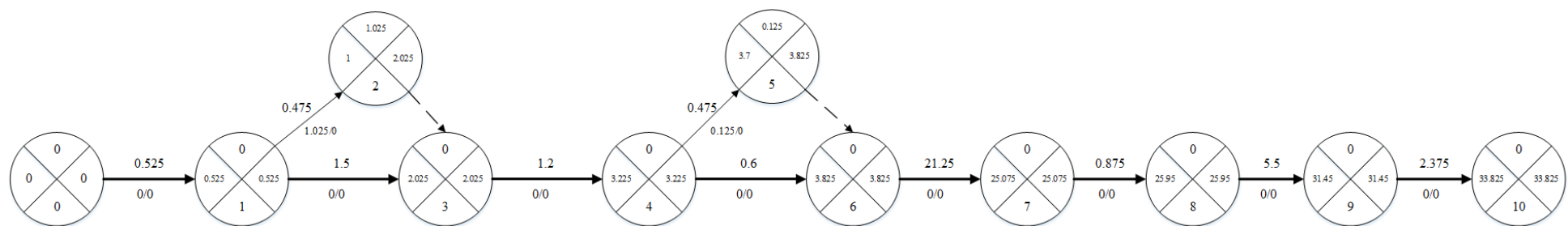


Рисунок 4.2 – Сетевой график выполнения работ

Диаграмма Гантта позволяет наглядно представить процесс выполнения работ во времени, показать ответственных за каждую работу. В тоже время, взаимосвязи между отдельными работами в явном виде на диаграмме не отображаются, что создает трудности при необходимости корректировки общего графика работ из-за изменения сроков выполнения какой-либо работы. В таком случае, необходимо вернуться назад и пересмотреть структуру декомпозиции работ и календарные планы для того, чтобы найти место и время вновь появившимся дополнительным видам деятельности

Диаграмма Гантта для работ по разработке репозитория инсталляционных пакетов представлена на рисунке 4.3.

4.5 Расчет затрат на разработку проекта

Затраты на разработку проекта выражаются формулой (4.7) складываются из затрат на заработную плату исполнителям, затрат на покупку или аренду оборудования, затрат на организацию рабочих мест и затрат на накладные расходы.

$$K_p = C_{\text{ЗАРП}} + C_{\text{ОБ}} + C_{\text{НАКЛ}}, \quad (4.7)$$

где $C_{\text{ЗАРП}}$ – заработная плата исполнителей;

$C_{\text{ОБ}}$ – затраты на покупку или аренду оборудования;

$C_{\text{НАКЛ}}$ – затраты на накладные расходы.

Затраты на выплаты исполнителям заработной платы вычисляются следующим образом:

$$C_{\text{ЗАРП}} = C_{\text{З.ОСН}} + C_{\text{З.ДОП}} + C_{\text{З.ОТЧ}}, \quad (4.8)$$

где $C_{\text{З.ОСН}}$ – основная заработная плата;

$C_{\text{З.ДОП}}$ – дополнительная заработная плата;

$C_{\text{З.ОТЧ}}$ – отчисления с заработной платы.

Расчет основной заработной платы при дневной оплате труда проводится на основе данных по окладам и графику занятости исполнителей:

$$C_{\text{З.ОСН}} = T_{\text{ЗАН}} * O_{\text{ДН}}, \quad (4.9)$$

где $T_{\text{ЗАН}}$ – число дней, отработанных исполнителем проекта;

$O_{\text{ДН}}$ – дневной оклад исполнителя.

При восьмичасовом рабочем дне оклад рассчитывается по следующей формуле:

$$O_{\text{ДН}} = \frac{O_{\text{МЕС}} * 8}{F_{\text{М}}}, \quad (4.10)$$

где $O_{\text{МЕС}}$ – месячный оклад;

$F_{\text{МЕС}}$ – месячный фонд рабочего времени.

С учетом налога на доходы физических лиц, размер месячного оклада увеличивается, что показано в формуле:

$$O_{\text{мес}} = O * (1 - \frac{H_{\text{дфл}}}{100}), \quad (4.11)$$

где O – оклад, на который может претендовать исполнитель с учетом требуемой квалификации и сложности проекта (по данным кадровых агентств);

$H_{\text{дфл}}$ – налог на доходы с физических лиц (13 %).

Согласно [9], оклад инженера-разработчика, полагаем равным $O = 57000$ рублей.

Согласно формуле (4.11), месячный оклад $O_{\text{мес}} = 57000 * (1 - \frac{13}{100}) = 49590$ рублей. При месячном фонде рабочего времени $F_{\text{м}} = 168$ часов, из формулы (4.10), дневной оклад составляет $O_{\text{дн}} = 2361,43$ рублей.

За время выполнения проекта $T_{\text{зан}} = 34$ дня суммарная основная заработная плата инженера-программиста, по формуле (4.9), составит $C_{\text{з.осн}} = 34 * 2361,43 = 80288,62$ рублей.

Дополнительная заработная плата – это выплаты исполнителю за время, не потраченное непосредственно на работу, но предусмотренное трудовым кодексом: оплата отпусков и компенсация за неиспользованный отпуск. Величина дополнительной заработной платы рассчитывается как 20% от размера основной заработной платы:

$$C_{\text{з.доп}} = 0,2 * C_{\text{з.осн}}, \quad (4.12)$$

Из формулы (4.12) получаем, что $C_{\text{з.доп}} = 16057,72$ рублей.

Отчисления с заработной платы заключаются в уплате единого социального налога. Согласно налоговому кодексу РФ применяются ставки налога для отчисления в пенсионный фонд РФ, фонд социального

страхования, фонды обязательного медицинского страхования (федеральный и территориальный фонды). Отчисления с заработной платы вычисляются по формуле:

$$C_{3.0тч} = (C_{3.0сн} + C_{3.доп}) * H_{соц}, \quad (4.13)$$

где $H_{соц}$ – отчисления с заработной платы в виде единого социального налога, равные 35,6%. Из них. 28% - в федеральный бюджет; 4% - в фонд социального страхования; 3,4% - в территориальный фонд обязательного медицинского страхования и 0,2% - в федеральный фонд обязательного медицинского страхования.

Из формулы (4.13), что отчисления составят $C_{3.0тч} = 34299,3$ рублей. По формуле (4.8) суммарные затраты на выплату заработной платы составят $C_{зарп} = 80288,62 + 16057,72 + 34299,3 = 130645,64$ рублей.

Необходимое для выполнения работ оборудование уже имеется в отделе и предоставляется безвозмездно, следовательно, затраты на оборудование $C_{об} = 0$.

Накладные расходы вычисляются в расчете 234,5 % [9] от расходов на основную заработную плату:

$$C_{накл} = 2,345 * C_{3.0сн} \quad (4.14)$$

Исходя из (4.14), накладные расходы составляют:

$$C_{накл} = 2,345 * 80288,62 = 188276,81 \text{ руб.}$$

Таким образом, по формуле (4.7), суммарные затраты на разработку репозитория инсталляционных пакетов составляют:

$$K_p = 130645,64 + 0 + 188276,81 = 318922,45 \text{ руб.}$$

Для расчета нормы прибыли необходимо суммарные затраты умножить на 20% [9]:

$$P = K_p * 0,2 = 63784,49 \text{ руб.}$$

4.6 Вывод

В данном разделе была определена структура работ по созданию репозитория инсталляционных пакетов, был произведен расчет трудоемкости проекта, в результате которого было получено, что общая трудоемкость $Q = 278.2 \text{ чел. час.} = 34.775 \text{ чел. дней}$. Весь проект может быть выполнен одним специалистом в 10 этапов общей продолжительностью 33,825 человеко-дней. Так же, были построены сетевой график и календарный график выполнения проекта. При расчете затрат на разработке проекта было определено что, они составят 318922 рублей 45 копеек.

ЗАКЛЮЧЕНИЕ

В результате выполнения дипломной работы был разработан репозиторий инсталляционных пакетов корпоративной сети

Разработанная система **ВНЕДРЕНА?** И используется

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Кузнецов С. Архитектуры ИС [Электронный ресурс] // Reksoft. URL: <http://www.reksoft.com/pdf/lectures/01-ArhitekturyIS.pdf> (дата обращения: 13.02.2016).
2. Архитектура ИС. [Электронный ресурс] // Научно-образовательный кластер CLAIM. URL: http://it-claim.ru/Education/Course/ISDevelopment/Lecture_3.pdf (дата обращения: 13.02.2016).
3. Кононов А., Кузнецов Е. Онтология распределенных прикладных систем [Электронный ресурс] // Открытые системы. URL: <http://www.osp.ru/os/2002/11/182086/> (дата обращения: 13.02.2016).
4. Белостоцкого А.И. Варианты построения информационных приложений и этапы компьютеризации управления в промышленности [Электронный ресурс] // "CREDO": Life long Learning & ONTOGENEZ - персональный WEB-сайт доцента прикладной математики, канд.техн.наук. А.И.Белостоцкого. URL: <http://belani.narod.ru/1/Lklser1.htm> (дата обращения: 12.02.2016).
5. Анатольев А.Г. Компоненты сетевого приложения. Клиент-серверное взаимодействие и роли серверов [Электронный ресурс] // Учебно-методические материалы для студентов кафедры АСОИУ. URL: <http://www.4stud.info/networking/lecture5.html> (дата обращения: 13.02.2016).
6. Крис Шефер [Chris Schaefer], Кларенс Хо [Clarence Ho], Роб Харроп [Rob Harrop] Spring 4 для профессионалов: пер. с англ. Изд. 4-е, М.: ООО "И.Д. Вильямс", 2015.
7. Буч Г. [Booch G.] Объектно-ориентированный анализ и проектирование: пер. с англ. Изд. 2-е / под ред. И. Романовского и Ф. Андреева, М.: ООО "И.Д. Вильямс", 2008.
8. Иванова Г.С. Технология программирования: Учебник для вузов. М: Изд-во МГТУ им. Н.Э. Баумана, 2002.

9. Приказ

10. Мощелкова В.Ю. Анализ ключевых организационно-экономических показателей наукоемкого инженерно-технического проекта. Электронное учебное издание – МГТУ им. Н.Э. Баумана, 2011.
11. СанПиН 2.2.2/2.4.1340-03 Гигиенические требования к персональным электронно-вычислительным машинам и организации работы. Минздрав России, Москва, 2003.
12. СанПиН 2.2.4.548-96 Гигиенические требования к микроклимату производственных помещений. Минздрав России, Москва, 1997.
13. СанПиН 2.2.4.1294-03 Гигиенические требования к аэроионному составу воздуха производственных и общественных помещений. Минздрав России, Москва, 2003.
14. ГН 2.1.6.1338-03 Предельно допустимые концентрации (ПДК) загрязняющих веществ в атмосферном воздухе населенных мест. Минздрав России, Москва, 2003.
15. СП 52.13330.2011 Свод правил естественное и искусственное освещение. Актуализированная редакция СНиП 23-05-95*, 2011.
16. СН 2.2.4/2.1.8.566-96. Производственная вибрация, вибрация в помещениях жилых и общественных зданий. Санитарные нормы. Минздрав России, Москва, 1997.
17. ГОСТ 12.1.038-82 Предельно допустимые значения напряжений прикосновения и токов. 2001.
18. СП 12.13130.2009 Определение категорий помещений, зданий и наружных установок по взрывопожарной и пожарной опасности. Министерство Российской Федерации по делам гражданской обороны, чрезвычайным ситуациям и ликвидации последствий стихийных бедствий, 2009.

Приложение А

Техническое задание

Приложение Б

Графические материалы

Приложение В

Алгоритм добавления нового инсталляционного пакета

```
CustomUserDetails activeUser = (CustomUserDetails) ((Authentication) principal).getPrincipal();
if (!UserHelper.isModerator(activeUser)) {
    logger.warn("Попытка загрузки файлов (/file-add-handler) без прав модератора; служебный номер - "+activeUser.getEmployeeId());
    throw new ForbiddenException("Доступ запрещен");
}

JSONObject result = new JSONObject();
JSONArray errors = new JSONArray();
JSONArray success = new JSONArray();

String uploadRootDir = Settings.getUploadPath();

int fileCounter = 0;
for (MultipartFile file : files) {
    if (!file.isEmpty()) {
        try {
            String uploadedFileName = new String(file.getOriginalFilename().getBytes("ISO-8859-1"),
"UTF-8");

            InputStream inputStream = file.getInputStream();
            // формирование пути до файла
            String hash = FileHelper.getHash(inputStream);
            inputStream.close();

            StringBuilder newFileName = new StringBuilder();
            newFileName
                .append(uploadRootDir)
                .append(File.separator)
                .append(FileHelper.getHashPath(hash));

            // проверяем дублирование файла
            if (FileVersionModel.isExist(hash, file.getSize())) {
                JSONObject err = new JSONObject();
```

```

err.put("number", fileCounter);
err.put("msg", "Такой файл уже есть");
errors.add(err);
} else {
    // проверка существования пути до файла
    File uploadDir = new File(String.valueOf(newFileName));
    if (!uploadDir.exists()) {
        uploadDir.mkdirs();
    }

    // добавление конечного имени файла
    // к исходному имени добавляется хэш файла
    StringBuilder fileName = new StringBuilder()
        .append(FilenameUtils.getBaseName(uploadedFileName))
        .append("_")
        .append(hash);
    String extension = FilenameUtils.getExtension(uploadedFileName);
    if (!extension.equals("")) {
        fileName.append(".");
        fileName.append(extension);
    }
    newFileName.append(File.separator).append(fileName);

    // сохранение файла на диск
    inputStream = file.getInputStream();
    FileOutputStream stream = new FileOutputStream(new File(newFileName.toString()));
    IOUtils.copy(inputStream, stream);
    stream.close();
    inputStream.close();
    file.getInputStream().close();

    // получение свойств файла
    Map<Integer, String> properties = null;
    try {
        properties = PEProperties.parse(newFileName.toString());
    }

```

```

    } catch (Exception e) {
        e.printStackTrace();
    }
    boolean isFilled = false;
    String fileTitle = null;
    String versionValue = null;
    if (properties != null) {
        fileTitle = properties.get(PropertyModel.PRODUCT_NAME);
        versionValue = properties.get(PropertyModel.FILE_VERSION);
        if (fileTitle != null && !fileTitle.trim().equals("") && versionValue != null &&
!versionValue.trim().equals("")) {
            isFilled = true;
        }
    }
}

// сохранение файла в бд, если свойства заполнены
FileModel fileModel = null;
if (isFilled) {
    fileModel = FileModel.findByTitle(fileTitle);
    if (fileModel == null) {
        fileModel = new FileModel();
        fileModel.setTitle(fileTitle);
        fileModel.add();
    }
}

// добавлением новой версии файла
FileVersionModel fileVersion = new FileVersionModel();
if (fileModel == null) {
    fileVersion.setFileId(0);
} else {
    fileVersion.setFileId(fileModel.getId());
}
fileVersion.setFileName(fileName.toString());
fileVersion.setHash(hash);

```

```

fileVersion.setIsFilled(isFilled);
fileVersion.setUserId(activeUser.getEmployeeId());
long time = new Date().getTime();
fileVersion.setDate(time);
if (versionValue != null && !versionValue.trim().equals("")) {
    fileVersion.setVersion(versionValue);
}
fileVersion.setFileSize(file.getSize());
if (fileVersion.add()) {
    logger.info("Загружена новая версия, id=" + fileVersion.getId()+"; служебный номер - "
+ activeUser.getEmployeeId());
} else {
    logger.error("Ошибка при загрузке новой версии; служебный номер - " +
activeUser.getEmployeeId());
}

// добавление остальных свойств версии в БД
if (properties != null) {
    for (Map.Entry entry : properties.entrySet()) {
        int propertyId = (Integer) entry.getKey();
        if (propertyId != PropertyModel.FILE_VERSION && propertyId !=
PropertyModel.PRODUCT_NAME) {
            FileVersionPropertyModel fileProperty = new FileVersionPropertyModel();
            fileProperty.setFileVersionId(fileVersion.getId());
            fileProperty.setPropertyId(propertyId);
            fileProperty.setValue(String.valueOf(entry.getValue()));
            if (fileProperty.add()) {
                logger.info("Версии id=" + fileVersion.getId()+" добавлено новое свойство
id="+propertyId+", значение - "+fileProperty.getValue()+"; служебный номер - " +
activeUser.getEmployeeId());
            } else {
                logger.error("Ошибка при добавлении свойства id"+propertyId+" версии
id="+fileVersion.getId()+"; служебный номер - " + activeUser.getEmployeeId());
            }
        }
    }
}

```



```

        }
    }
    JSONObject succ = new JSONObject();
    succ.put("fileVersionId", fileVersion.getId());

    succ.put("fileVersionName", uploadedFileName);
    succ.put("isFilled", isFilled);
    success.add(succ);
}
} catch (Exception e) {
    JSONObject err = new JSONObject();
    err.put("number", fileCounter);
    err.put("msg", e.getMessage());
    errors.add(err);
}
} else {
    JSONObject err = new JSONObject();
    err.put("number", fileCounter);
    err.put("msg", "Проблема с обработкой файла");
    errors.add(err);
}
fileCounter++;
}
result.put("errors", errors);
result.put("success", success);
return result.toJSONString();

```