**Введение**

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. Предлагается разработать репозиторий инсталляционных пакетов, в котором будут храниться такие файлы.

1. **Анализ аналогов, анализ видов архитектур и выбор архитектуры для разрабатываемой системы.**
   1. **Анализ аналогов и определение требований к разрабатываемой системе**

Перед тем, как приступать 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, Xinco 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.

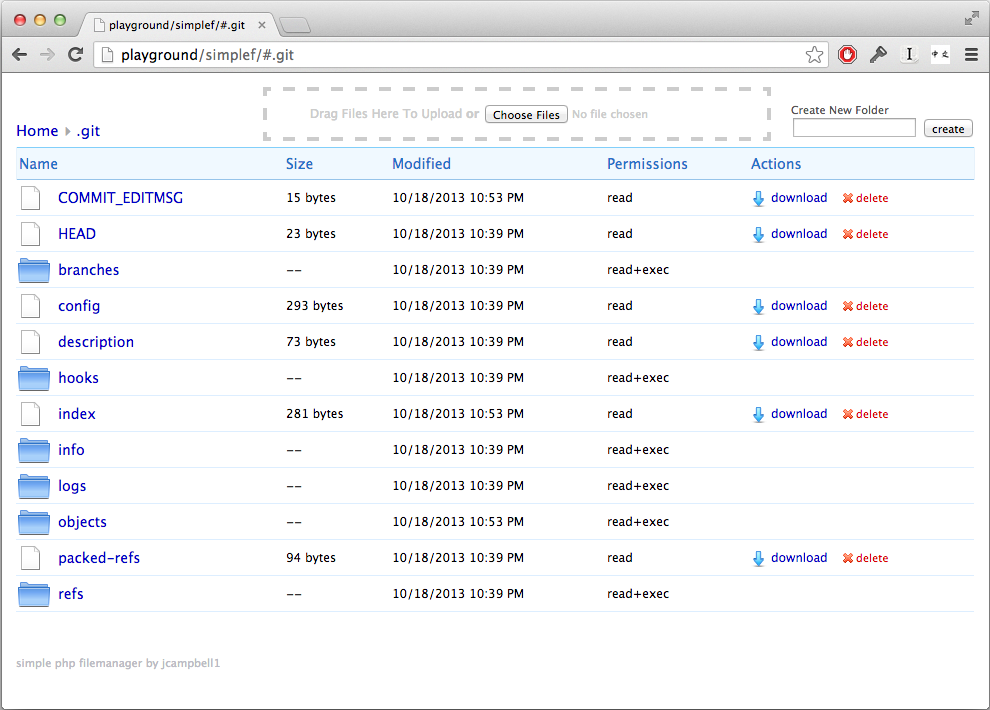


Рисунок 1.1 – Пользовательский интерфейс приложения Simple File Manager.

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.

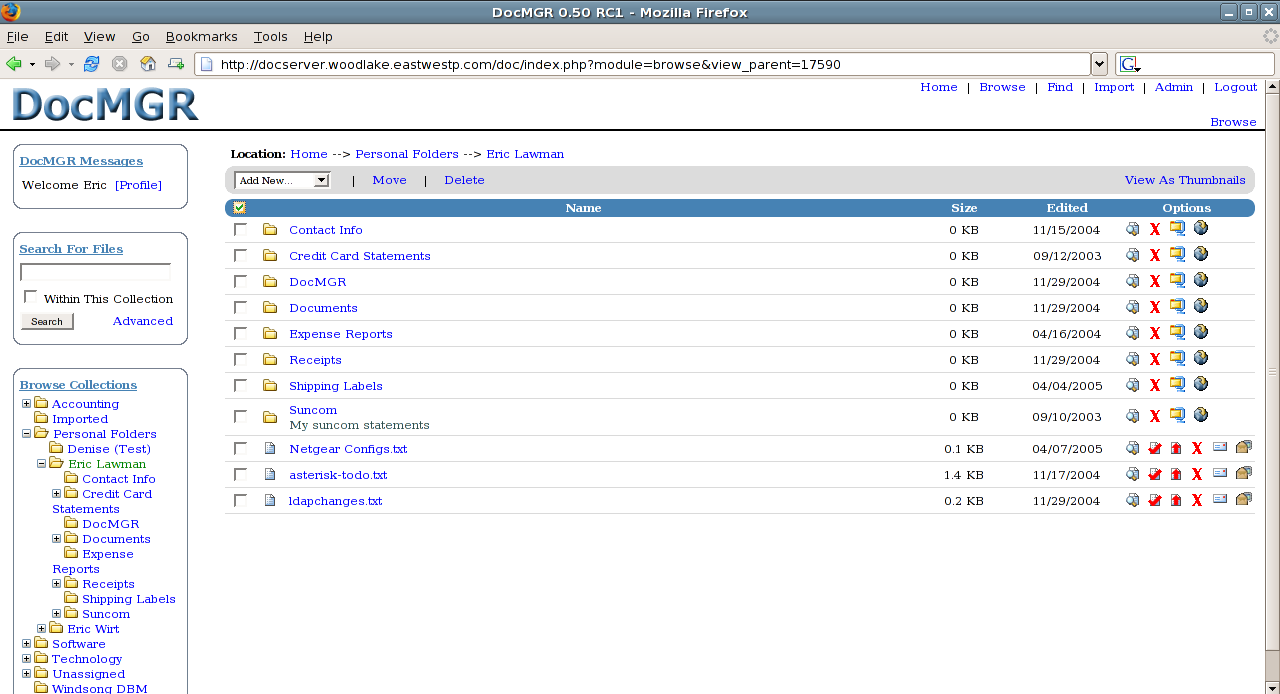


Рисунок 1.2 – Пользовательский интерфейс DocMGR.

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

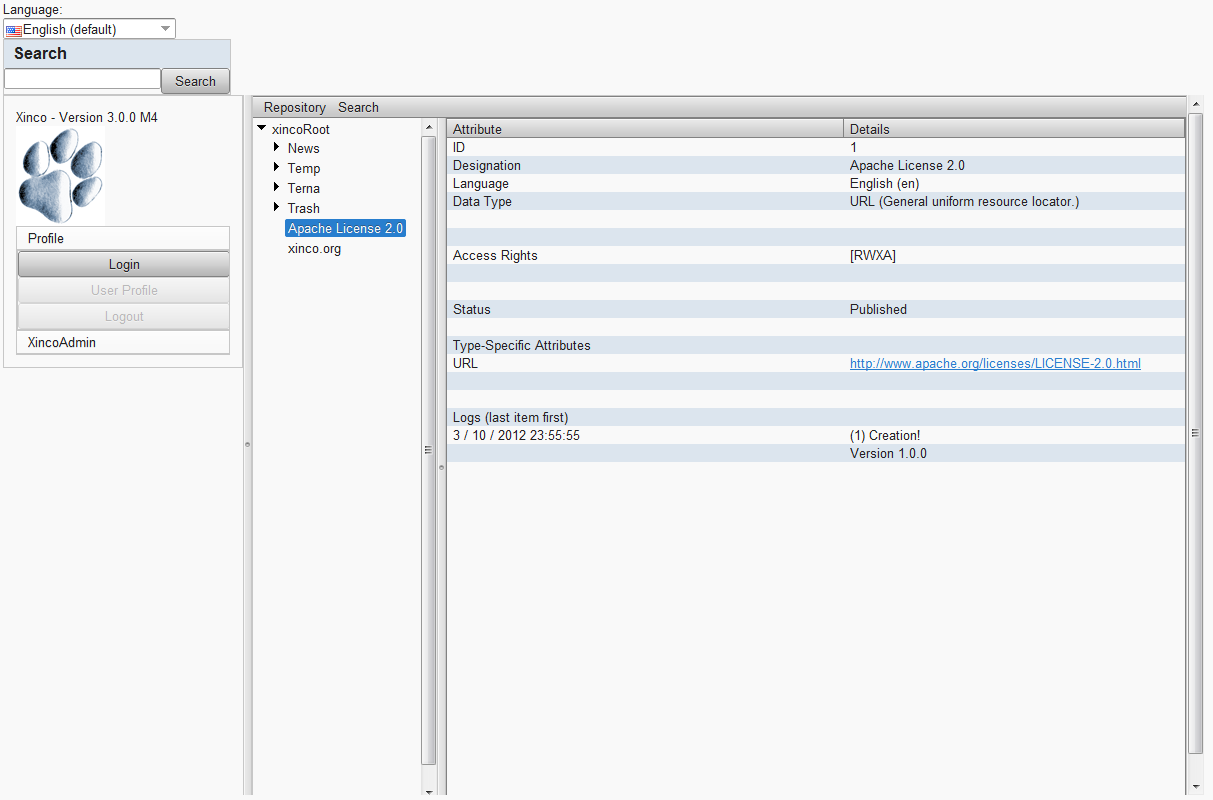


Рисунок 1.3 – Пользовательский интерфейс Xinco DMS.

Из достоинства Xinco 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.

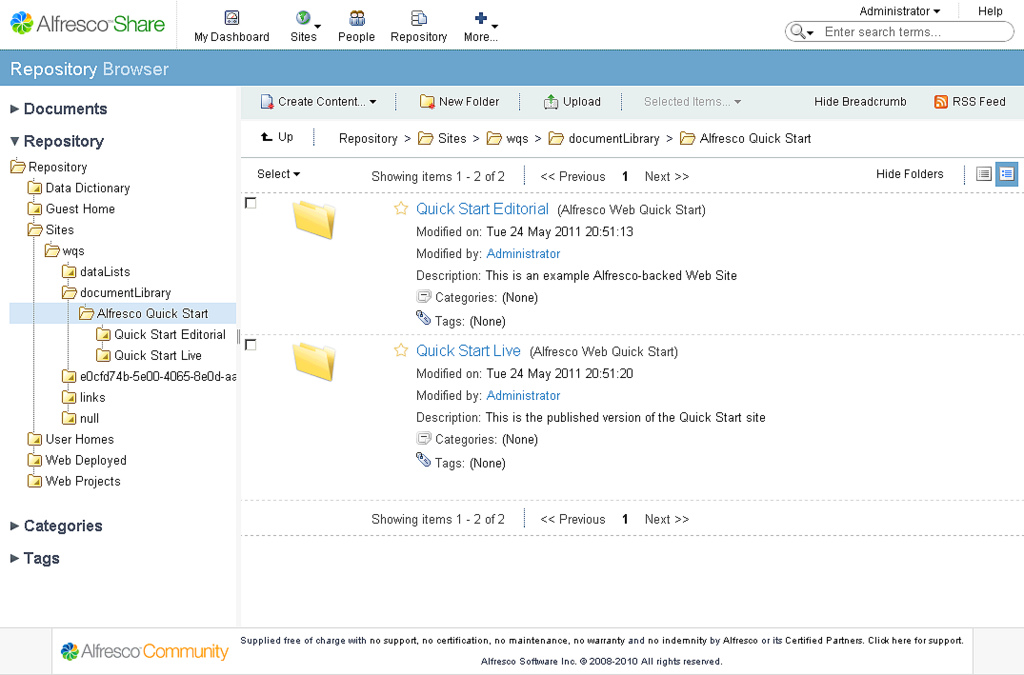


Рисунок 1.4 – Пользовательский интерфейс Alfresco.

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. **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. **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.

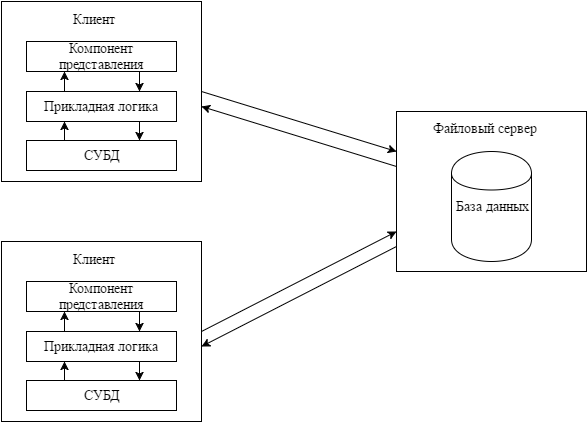


Рисунок 1.5 - Структура архитектуры «файл-сервер».

Во время работы приложения, для каждого клиента создается локальная копию 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 снижение 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 могу одновременно иметь доступ на запись к одному файлу. Однако они могу работать с такими файлами, если они обращаются к ним исключительно в режиме чтения.

Так как все данных 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 выносится работа с базой данных, вся логика приложения по обработке данных и по их представлению пользователю.

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

Однако, этот подход обладает большим количеством недостатков. Рассмотренная архитектура не предполагает большого количества пользователей, а штат сотрудников на предприятии АО "ВПК "НПО машиностроения" достаточно велик. 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 другими недостатками такой системы можно отнести трудность обеспечения непротиворечивости и целостности данных и трудность обеспечение безопасности, так как для предоставления возможности работы в системе каждому пользователю необходимо дать полный доступ, а данные для доступа к серверу будут хранятся на стороне пользователя.

* + 1. **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 функционируем 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 исполняется 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. **Двухзвенная архитектура «клиент-сервер»**

Двухзвенной данная архитектура называется так из-за необходимости распределения трех слоев между двумя узлами: клиентом и сервером. На сторону клиента выносится 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.

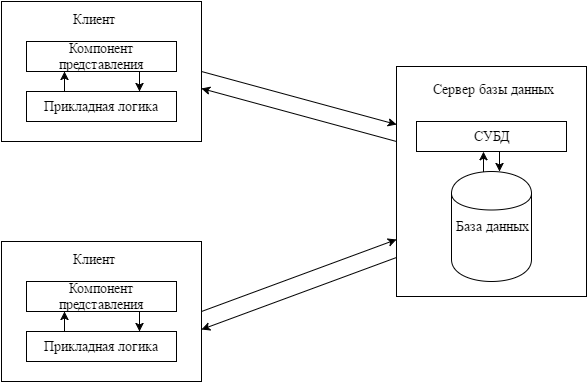


Рисунок 1.6 - Структура конфигурации «толстый клиент» двухзвенной архитектуры «клиент-сервер».

К преимуществам данной модели относиться 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.

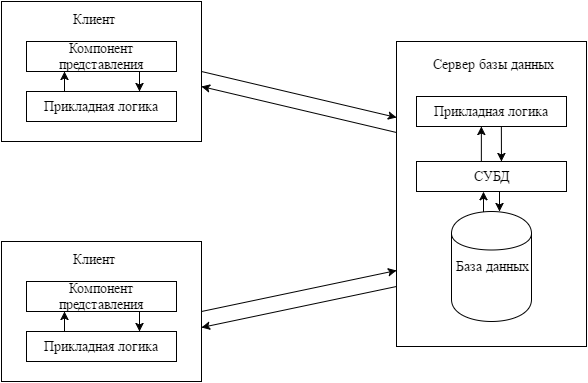


Рисунок 1.7 - Структура конфигурации «тонкий клиент» двухзвенной архитектуры «клиент-сервер».

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 быстродействие 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. **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 архитектура «клиент-сервер» представлена на рисунке 1.8.

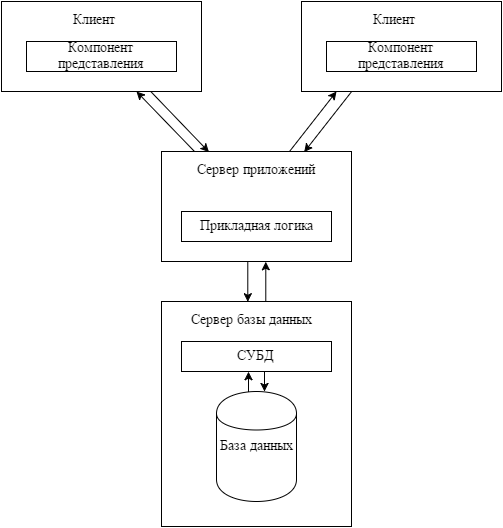


Рисунок 1.8 - Структура трехзвенной архитектуры «клиент-сервер».

Такая архитектура позволяет более гибко распределять функции системы и снизить нагрузку на сетевое и вычислительное оборудование. Клиентское приложение становиться «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 переноситься 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. **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 обоих 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 будут 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. **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».

1. **Разработка репозитория инсталляционных пакетов.**
   1. **Анализ предметной области и уточнение спецификации.**
      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 новый 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 команды операционной системы.

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

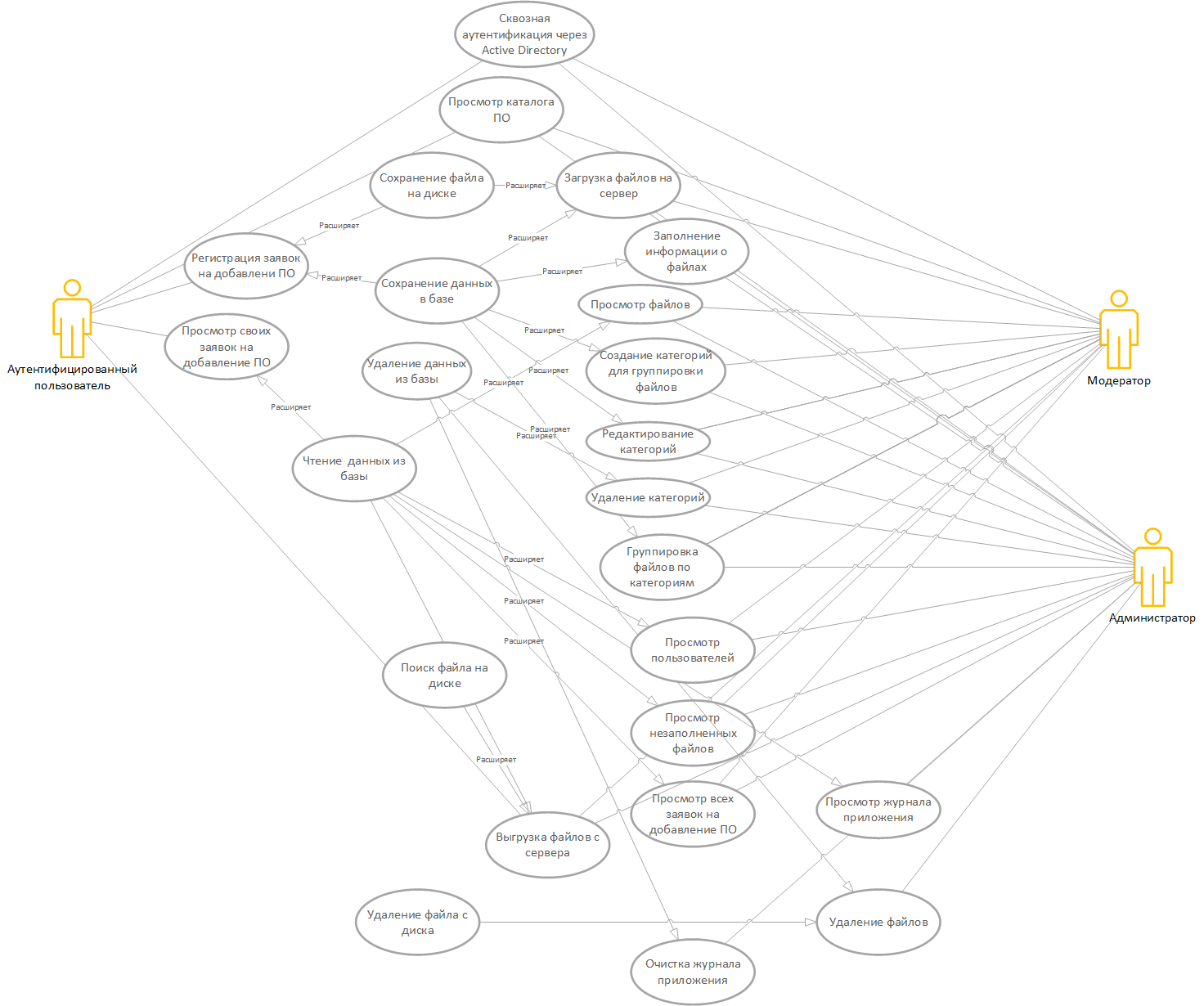


Рисунок 2.1 – Диаграмма вариантов использования разрабатываемой системы.

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 смогу просматривать. В качестве такой информации было решено использовать:

* ФИО;
* рабочий телефон;
* рабочий email;
* название отдела;
* номер отдела;
* адрес отдела.
  + 1. **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. [Гради Буч]
* Объектно-ориентированные системы более гибки и проще эволюционируют со временем, потому что их схемы базируется 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. [Гради Буч]
* Объектная декомпозиция помогает нам разобраться в сложной программной системе, предлагая нам разумные решения относительно выбора подпространства большого пространства состояний. [Гради Буч]

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.

* + 1. **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 работы 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. К преимуществам данной СУБД относится:

* большое количество стандартных типов данных;
* высокопроизводительные и надёжные механизмы транзакций и репликации;
* хорошая документация;
* присутствует операция сборки мусора VACUUM.

В качестве недостатков PostgreSQL можно отметить отсутствие кэша запросов, который бы позволил увеличить скорость обработки повторяющихся запросов. Главным достоинством PostgreSQL является команда сборки мусора VACUUM. В PostgreSQL операции, модифицирующие записи UPDATE и DELETE, не удаляют на самом деле кортеж, а помечают его удаленным (в случае обновления – старая версия кортежа помечается как удаленная, а новая версия вставляется). Такой подход позволяет избежать падения производительности при удалении элементов и ускорить обновление, но это приводит к тому, что в ходе модификации таблицы в ней появляется неиспользуемое пространство (старые удаленные кортежи). Это пространство не может быть повторно использовано без предварительной подготовки. Для этого была сделана команда сбора мусора VACUUM.

* 1. **Разработка структуры программы и ее компонентов.**
     1. **Разработка структуры программы.**
     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 является 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.

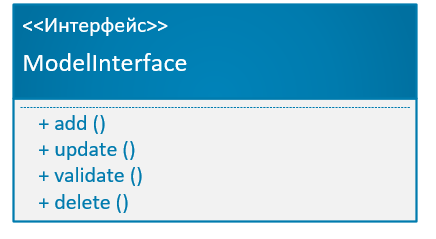


Рисунок 21 – Диаграмма интерфейса ModelInterface.

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 22.