**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

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 21).

Рисунок 21 – Диаграмма вариантов использования разрабатываемой системы.

**Выбор способа проектирования**

Существует два способа проектирования программных систем:

* Структурное проектирование;
* Объектно-ориентированное проектирование.

Структурное проектирование основано на алгоритмической декомпозиции, то есть разбиении системы на элементарные функции. Система разбивается на функциональные подсистемы, которые в свою очередь делятся на подфункции, подразделяемые на задачи и так далее. Процесс разбиения продолжается вплоть до конкретных процедур. Каждый модуль системы выполняет один из этапов общего процесса. При этом создаваемая система сохраняет целостное представление, все составляющие компоненты в ней взаимосвязаны.

Объектно-ориентированное проектирование основано 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.

Эти способы противоположны друг другу, поэтому нельзя сконструировать сложную систему использую одновременно оба способа. Объектная декомпозиция имеет ряд преимуществ над алгоритмической декомпозицией:

* Объектная декомпозиция уменьшает размер программных систем за счет повторного использования общих механизмов, что приводит к существенной экономии выразительных средств.
* 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.
* Объектная декомпозиция помогает нам разобраться в сложной программной системе, предлагая нам разумные решения относительно выбора подпространства большого пространства состояний. [Груди Буч]

На основе приведенных преимуществ был выбран объектно-ориентированный подход. 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.

**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 результате выбора архитектуры для разработки репозитория инсталляционных пакетов была выбрана трехзвенная клиент-серверная архитектура. 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, развиваемый сообществом. 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.

**Разработка базы данных**

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.

Таблица user необходима для хранения информации о пользователях в системе. В ней будут храниться ФИО, телефон, электронный адрес, номер и название отдела и адрес. Таблица file предназначена для хранения общей информации о файлах и для объединения версий файлов. Для хранения версий файлов предназначена таблица fileVersion, в которой храниться информация по конкретной версии файла: размер, номер версии, хэш-сумма, дата загрузки. Таблица category хранит информацию о категориях, которые предназначены для группировки файлов. Для того чтобы категории можно было выстраивать в виде дерева в таблице присутствует поле parent, которое указывает на id родительской категории. Так же, в таблице храниться название категории и ее порядковый номер. Для хранения связей файлов с категориями предназначена таблица fileCategory. В соответствии с техническим заданием, у файлов должны быть свойства. Для их хранения служит таблица property. Чтобы свойства можно было привязывать как к файлу, так и версии файла, необходимы две таблицы fileProperty и fileVersionProperty. Для сохранения заявок на добавление ПО предназначена таблица request. Так как к одной заявке может быт прикреплено сразу несколько файлов, то файлы заявки были вынесены в отдельную таблицу requestFile. Таблица log предназначена для ведения журнала приложения.

Полученная в результате проектирования логическая схема базы данных представлена на рисунке 22.

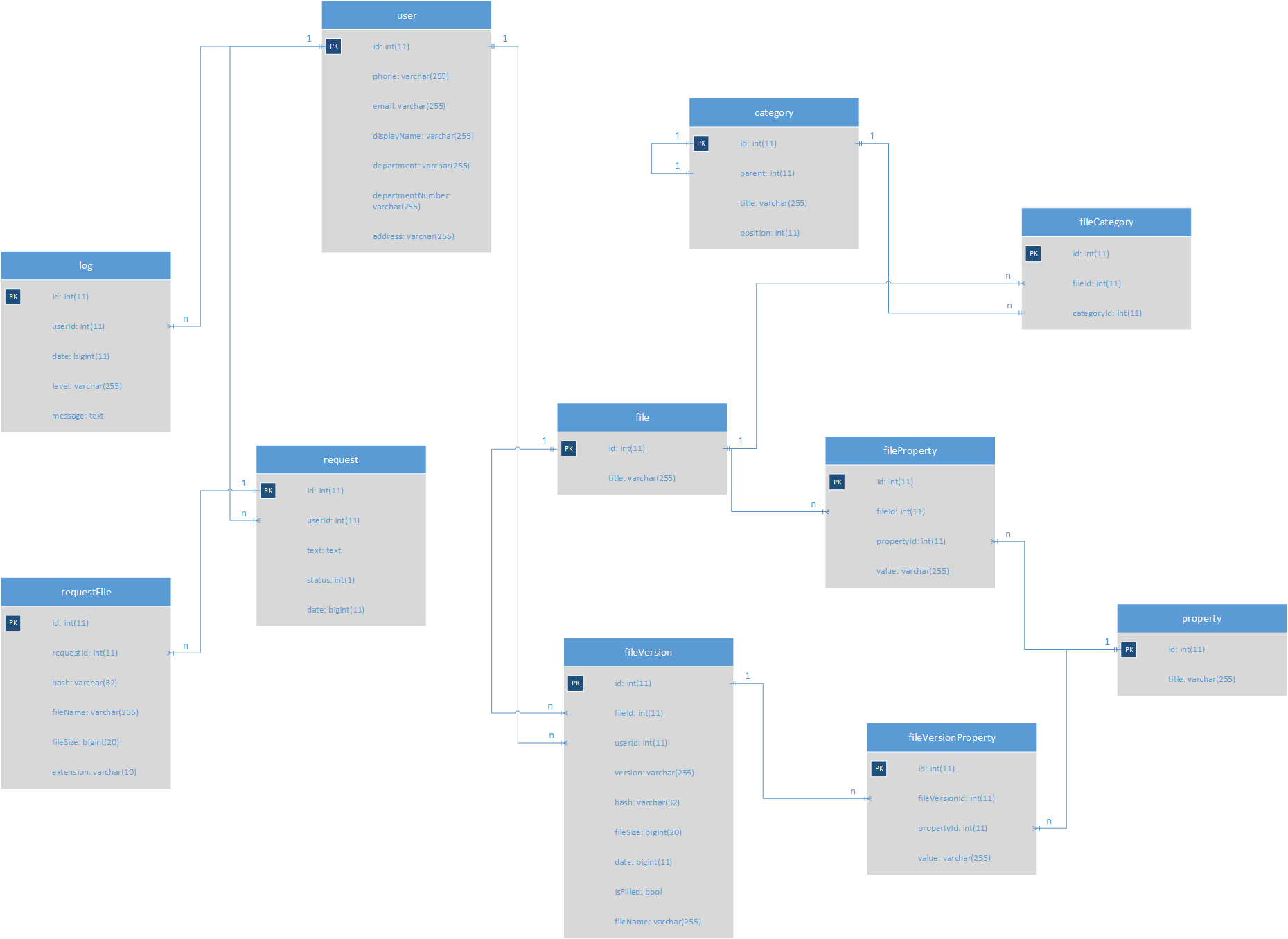


Рисунок 22 – Логическая схема базы данных.

**Разработка классов**

Для каждой сущности, выделенной в результате объектной декомпозиции, 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 21. 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 дргуие. 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.

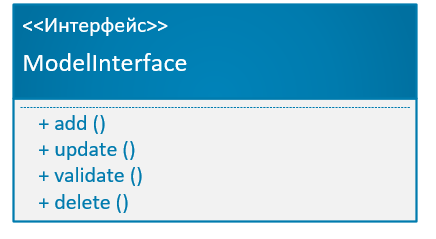


Рисунок 21 – Диаграмма класса ModelInterface.

Для работы с таблицей пользователей user был разработан класс UserModel. Диаграмма данного класса представлена на рисунке 22.

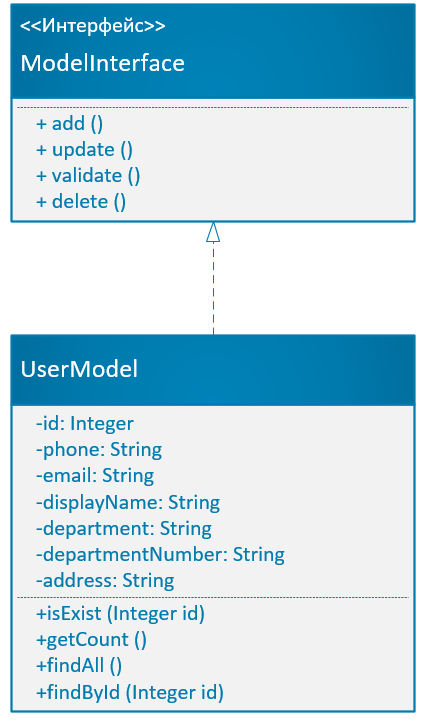


Рисунок 22 – Диаграмма класса UserModel.

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. 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» система автоматически получает 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 заполнением вручную. Диаграмма класса FileVersionModel представлена на рисунке 23.

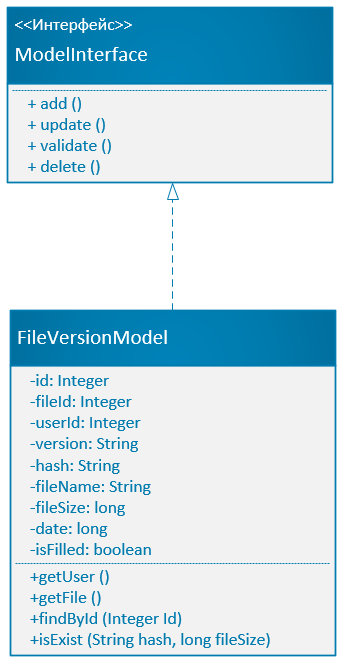


Рисунок 23 – Диаграмма класса FileVersionModel.

Поле userId указывает на id пользователя из таблицы user, который загрузил данных файл. Для того, чтобы избежать дублирования версий в системе были добавлены 2 поля: hash, которое хранит хэш-сумму файла и fileSize, которое хранит размер файла. При добавлении нового файла должна осуществляться проверка на дублирование. 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 прикреплена 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 работы с таблицей file был разработан класс FileModel. Диаграмма класса представлена на рисунке 23.

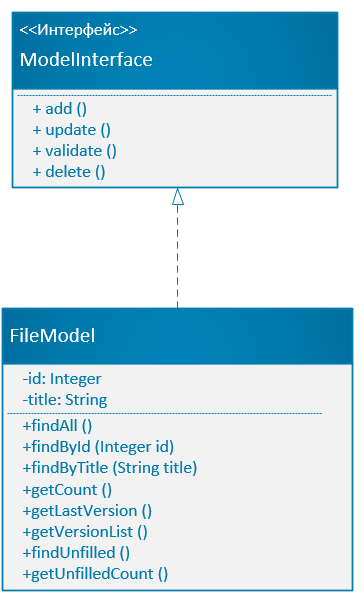


Рисунок 23 – Диаграмма класса FileModel.

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).

Для работы со свойствами был разработан класс PropertyModel, диаграмма которого представлена на рисунке 25.

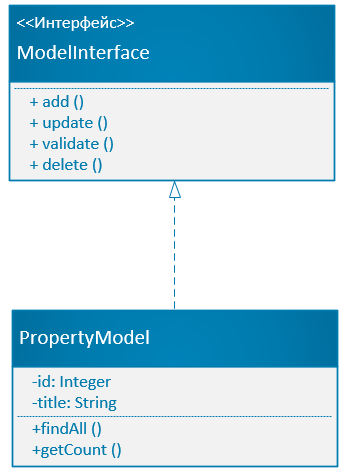


Рисунок 25 – Диаграмма класса PropertyModel.

Сущность «свойство» содержит только название свойства. 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. Диаграмма данного класса представлена на рисунке 26.

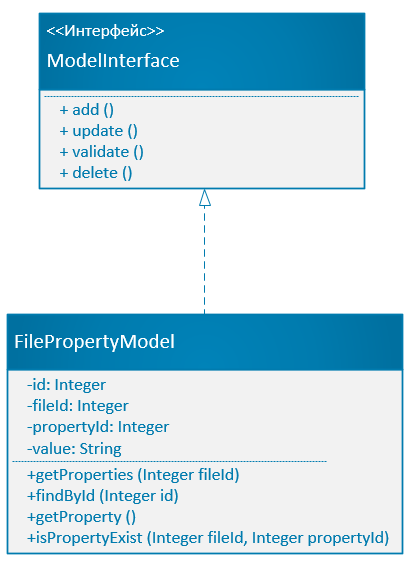


Рисунок 26 – Диаграмма класса FilePropertyModel.

Данная сущность связывает между собой сущности «файл» и «свойство», а также хранит значения свойства. 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 свойстом, был добавлен метод isPropertyExist (Integer fileId, Integer propertyId), который проверяет, существует ли в таблице базы данных запись для заданного файла и свойства. Аналогичный класс был разработан для свойств версии файла - FileVersionPropertyModel. Он содержит такие же поля и методы, как и класс FilePropertyModel. Диаграмма класса FileVersionPropertyModel представлена на рисунке 27.

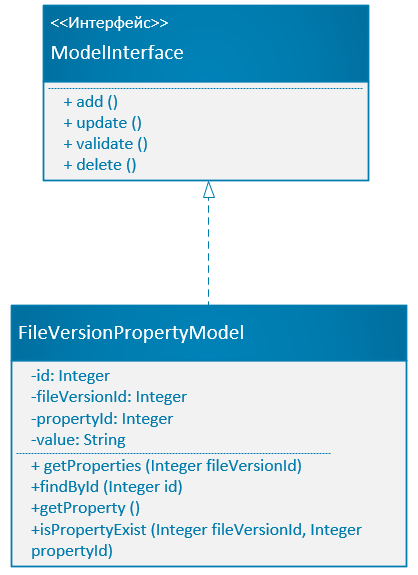


Рисунок 27 – Диаграмма класса FileVersionPropertyModel.

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. Его диаграмма представлена на рисунке 28. 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.

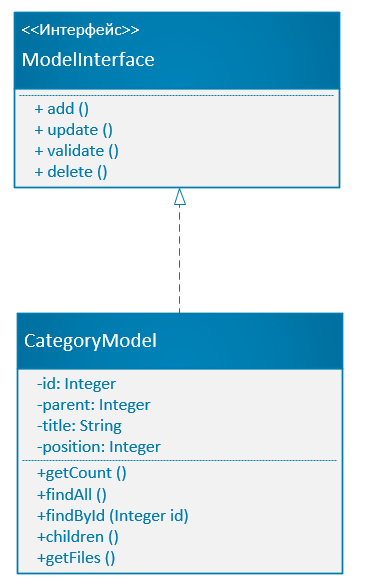


Рисунок 28 – Диаграмма класса CategoryModel.

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 уникальный 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, диаграмма которого представлена на рисунке 29.

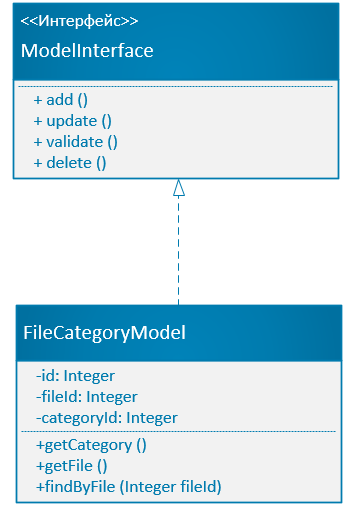


Рисунок 29 – Диаграмма класса FileCategoryModel.

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. Диаграмма класса RequestModel представлена на рисунке 30.

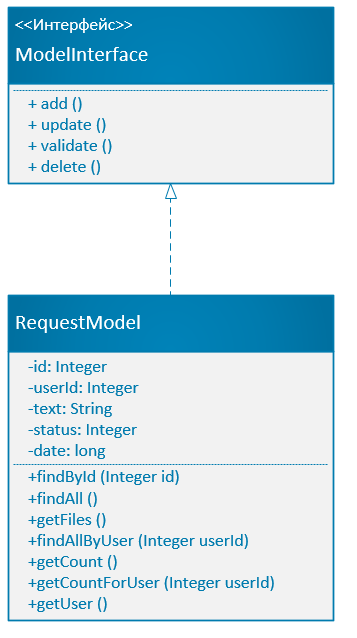


Рисунок 30 – Диаграмма класса RequestModel.

Заявка включает в себя текст заявки, связь с пользователем, который оставил эту заявку, дату регистрации заявки, текущий статус заявки и комментарий модератора. 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 в базе данных requestFile и класс RequestFileModel. Диаграмма данного класса представлена на рисунке 31.

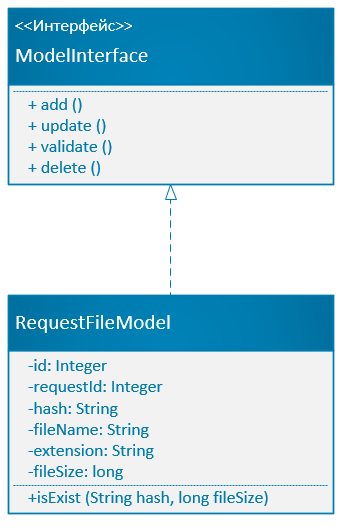


Рисунок 31 – Диаграмма класса RequestFileModel.

В целях обеспечения безопасности загружаемые в заявке 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. Поле requestId необходимо для связи файлов с заявкой, поле filename – для хранения имени 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. Для этого в класс был добавлен метод isExist(String hash, long fileSize).

LogModel

**Разработка интерфейса**

Пользовательский интерфейс представляет собой совокупность программных и аппаратных средств, обеспечивающих взаимодействие пользователя с системой [Иванова]. 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 требует 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». Внешний вид страницы аутентификации представлен на рисунке 34.

Рисунок 34 – Страница аутентификации.

Так как функции, которые предоставляются обычным пользователям и модераторам с администраторами, сильно различаются было принято решение сделать два вида меню: меня 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 35.

Рисунок 35 – Меню модератора или 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 36.

Рисунок 36 – 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 35.

Рисунок 37 – Страница 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. Вид данной страницы представлен на рисунке 38.