**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.

**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.

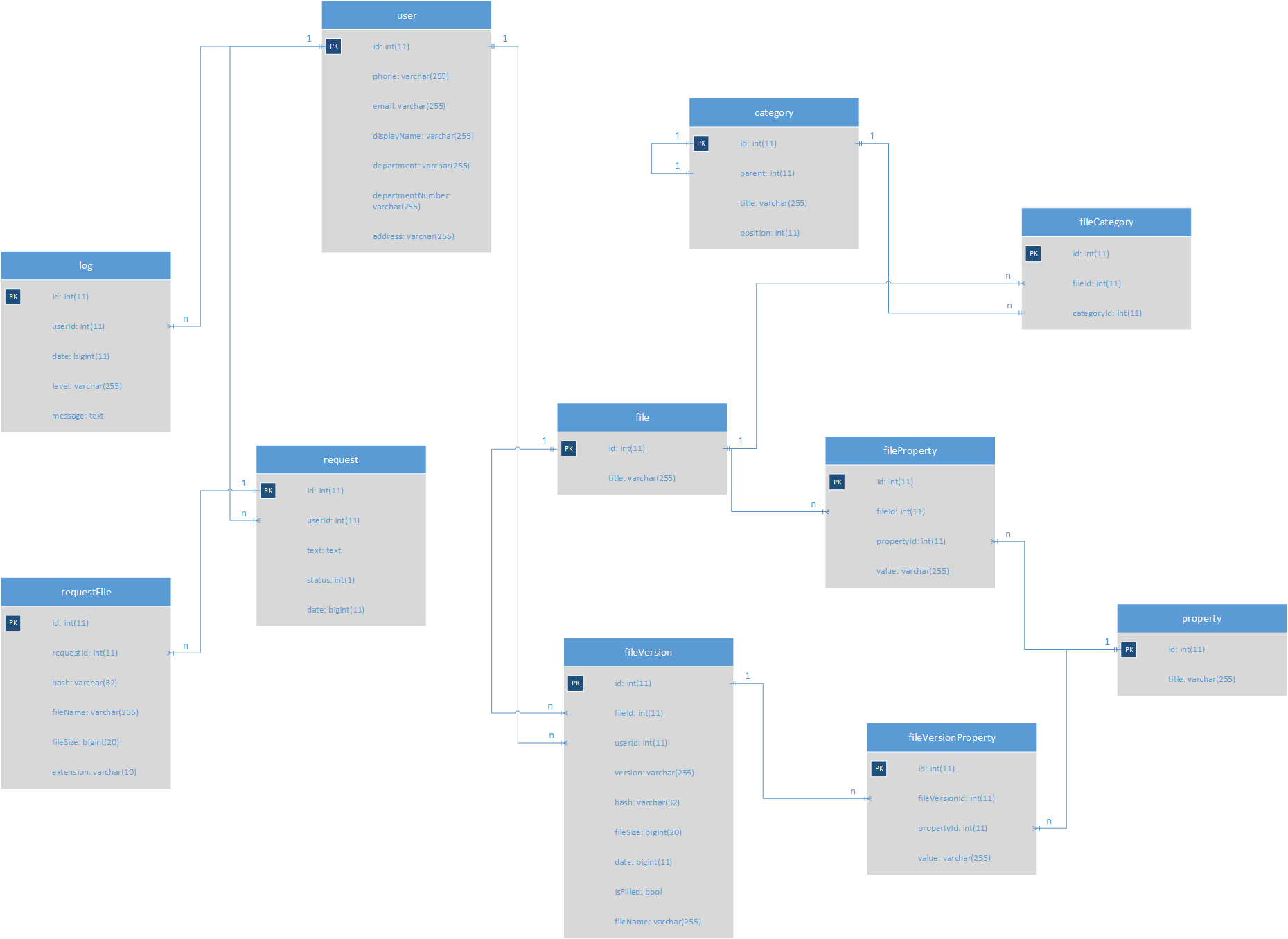


Рисунок 22 – Логическая схема базы данных.

**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.

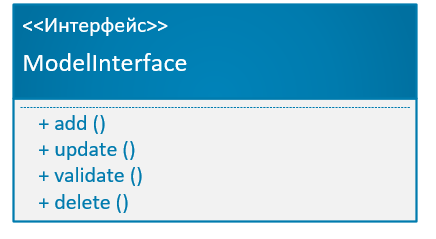


Рисунок 21 – Диаграмма класса ModelInterface.

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

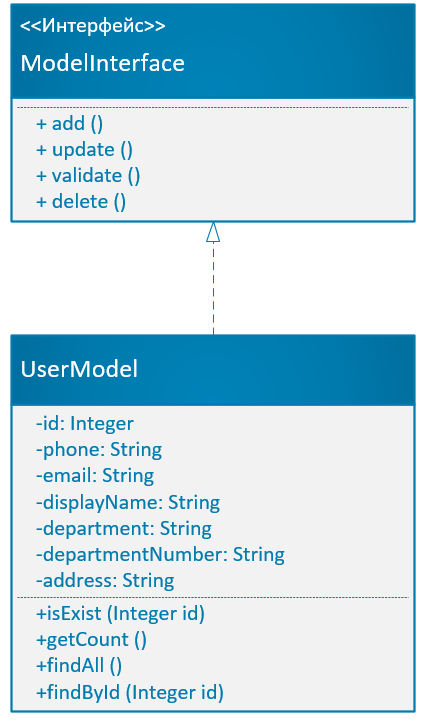


Рисунок 22 – Диаграмма класса UserModel.

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.

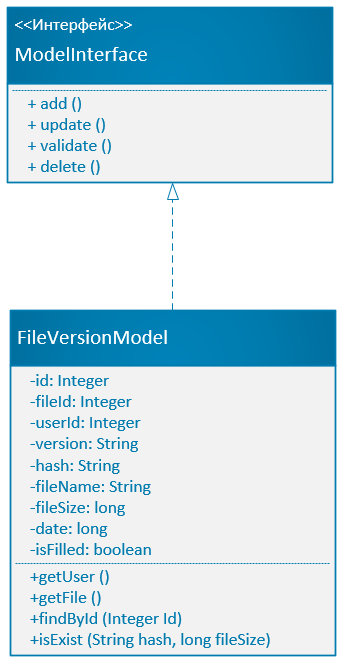


Рисунок 23 – Диаграмма класса FileVersionModel.

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.

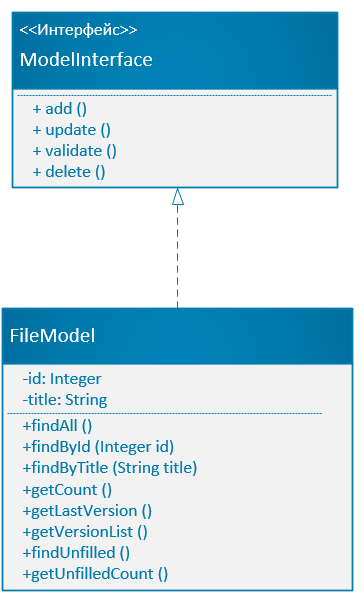


Рисунок 23 – Диаграмма класса FileModel.

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.

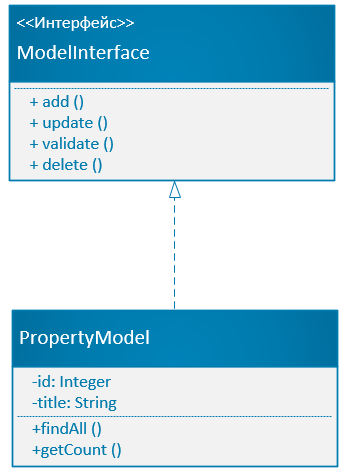


Рисунок 25 – Диаграмма класса PropertyModel.

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.

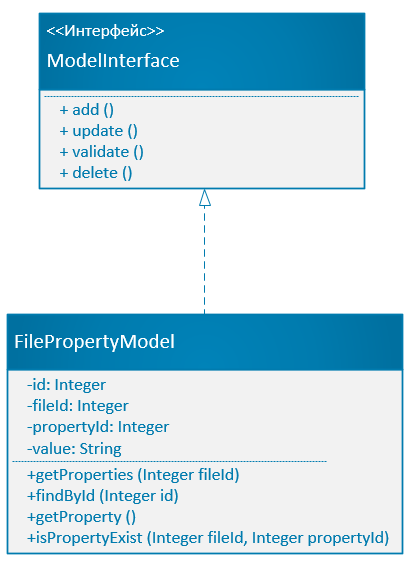


Рисунок 26 – Диаграмма класса FilePropertyModel.

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.

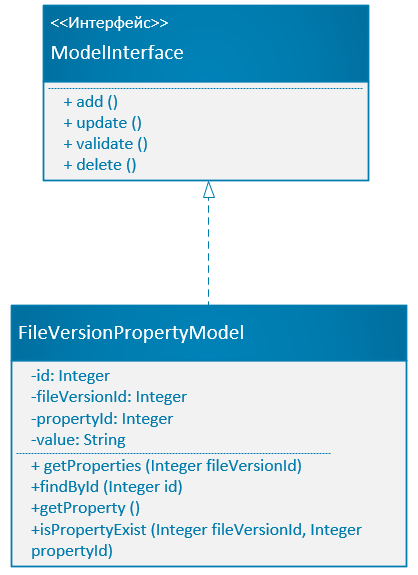


Рисунок 27 – Диаграмма класса FileVersionPropertyModel.

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.

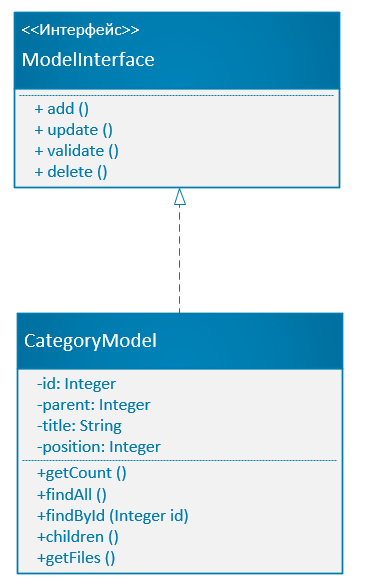


Рисунок 28 – Диаграмма класса CategoryModel.

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.

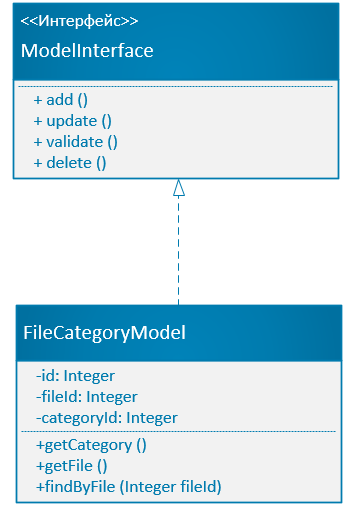


Рисунок 29 – Диаграмма класса FileCategoryModel.

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).