**СОДЕРЖАНИЕ**

[Введение 5](#_Toc502054765)

[1Анализ требований к проектируемому программному средству 6](#_Toc502054766)

[1.1Обзор аналогов программных средств 6](#_Toc502054767)

[1.2Требования к проектируемому программному средству 8](#_Toc502054768)

[2 Моделирование предметной области и разработка функциональных требований 11](#_Toc502054769)

[2.1 Описание функциональности ПС 11](#_Toc502054770)

[2.2 Спецификация функциональных требований 12](#_Toc502054771)

[3 Проектирование программного средства 14](#_Toc502054772)

[3.1 Разработка программной архитектуры 14](#_Toc502054773)

[3.2 Разработка модели базы данных 14](#_Toc502054774)

[3.3 Разработка алгоритма выполнения загрузки 16](#_Toc502054775)

[4 Создание программного средства 18](#_Toc502054776)

[4.1 Обоснование выбора средств разработки ПС 18](#_Toc502054777)

[4.2 Используемые модули и фреймворки 20](#_Toc502054778)

[4.3 Описание классов и методов 21](#_Toc502054779)

[5 Тестирование программного средства 23](#_Toc502054780)

[6 Руководство по установке и использованию 26](#_Toc502054781)

[6.1 Установка программного средства 26](#_Toc502054782)

[6.2 Использование программного средства 26](#_Toc502054783)

[Заключение 30](#_Toc502054784)

[Список использованных источников 31](#_Toc502054785)

[Приложение А 32](#_Toc502054786)

# **ВВЕДЕНИЕ**

Информационные технологи широко применяются во всех сферах деятельности, в том числе при сборе и обработке информации человеком. Благодаря развитию технологий вычислительных сетей, степень фрагментации информации увеличилась.

Необходимая человеку информация может находиться в разных узлах сети, к которым необходимо иметь доступ. Для того чтобы собрать информацию в одном месте для ее обработки и анализа, необходимо совершить множество механических действий для установки сеанса с удаленным узлом сети и загрузки информации с него.

Эти процедуры требуют от пользователя серьезной подготовки, а также затрат времени на неосновную деятельность.

Все это создает хорошие условия для создания программного обеспечения, которое бы автоматизировало этот процесс, освобождая тем самым человеческие ресурсы.

Целью данного курсового проекта является создание программного средства, которое позволит автоматизировать процесс сбора информации с разных узлов сети. Позволит автоматизировать процесс загрузки, тем самым оптимизируя процесс сбора и анализа информации.

Программное средство не должно быть ограничено типом загружаемой информации и должно предоставлять некоторую универсальность использования, чтобы покрывать широкий спектр потребностей пользователей.

Таким образом необходимо создать менеджер загрузок файлов с удаленных узлов в сети с поддержкой большого количества различных протоколов и высокой степенью автоматизации процесса.

# **1 АНАЛИЗ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ**

**1.1 Обзор аналогов программных средств**

Для создания нового программного средства, которое будет представлять собой решение для управления загрузками, необходимо изучить аналоги данного программного средства и выделить их основные недостатки и преимущества, определить ведущие тенденции в данном направлении.

В качестве исследуемых аналогов были выбраны программные продукты, связанные с управлением и осуществлением загрузок файлов с удаленных узлов сети. Основным критерием для выбора служила актуальность данных ПС, частота их использования. Источником информации послужили электронные базы в сети Интернет.

Все найденные программные средства представлены в таблице ниже.

Таблица 1.1 – Результаты поиска аналогов программных средств

|  |  |
| --- | --- |
| Наименование ресурса и источник | Признаки выявленных аналогов |
| 1 | 2 |
| 1.FlashGet (http://www.flashget.com) | Бесплатная компьютерная программа, менеджер закачек с закрытым исходным кодом для ОС Microsoft Windows.  Имеет следующие функции:   * позволяет скачивать несколько файлов одновременно; * позволяет скачивать файл одновременно в несколько потоков; * позволяет установить скорость закачки; * позволяет скачивать файл одновременно из нескольких источников; * содержит обозреватель сайта, чтобы просматривать содержимое каталогов сайта; * позволяет докачивать файлы после разрыва соединения. |

Продолжение таблицы 1.1

|  |  |
| --- | --- |
| 1 | 2 |
| 2.DownloadMaster (<https://westbyte.com/dm/>) | Менеджер загрузок с закрытым исходным кодом, работающий под Windows.  Имеет следующие функции:   * многопоточная закачка, способная намного повысить скорость закачки файлов; * встроенный FTP-клиент; * регулируемый режим для динамического управления скоростью закачки, возможность установки приоритетов для закачек; * встроенный поиск файлов, программ, игр и музыки. |
| uGet (http://ugetdm.com/) | Свободный, кроссплатформенный менеджер загрузок с открытым исходным кодом.  Имеет следующие функции:   * загрузка файлов по протоколам HTTP, HTTPS, FTP, и BitTorrent; * авторизация по FTP; * пакетная закачка; * многопоточная закачка; * установка ограничения скорости; * закачка одного файла с нескольких источников; * возможность докачивать файлы после разрыва соединения; * возможность создавать шаблоны для загружаемых файлов. |

В ходе поиска аналогов было найдено три программных средства, обладающих различными необходимыми функциональными особенностями. Анализ особенностей этих средств показывает, что создаваемое программное средство должно расширять функциональные возможности средств подобного рода. Функция организации расписания загрузок является таким расширением и оправдывает разработку программного средства.

**1.2 Требования к проектируемому программному средству**

1.2.1 Назначение разработки

Функциональное назначение программного средства – управление загрузкой файлов с удаленных узлов сети, то есть учет и управление данными загрузок, контроль возможности и качества соединения, эффективное использование канальных и вычислительных ресурсов для ускорения загрузки файлов.

Эксплуатационное назначение программного средства – предоставление пользователям персональных компьютеров возможностей по управлению загрузками файлов, настройке параметров загрузок, организации очереди загрузок в соответствии с существующими требованиями.

1.2.2 Состав выполняемых функций

Программное средство должно обеспечивать возможность выполнения перечисленных ниже функций:

– управление списком выполняемых загрузок;

– управление временем совершения загрузок;

– прохождение авторизации при загрузке;

– эффективное использование вычислительных ресурсов компьютера при выполнении загрузок;

– эффективное использование сетевого канала при загрузке.

1.2.3 Требования к организации входных данных

Поскольку возможность осуществления загрузки зависит от корректности введенных пользователем параметров, то программное средство должно обладать интерфейсом, который будет минимизировать возможность ввода некорректных данных, осуществлять форматно-логическую проверку введенных данных и информировать пользователя о совершенных при вводе ошибках.

Также программное средство должно обеспечивать целостность данных при долгосрочном хранении. Повреждение или внесение ошибок в данные при их вводе в программу и хранении должно быть исключено.

Программное средство должно информировать пользователя об ошибках в данных, обнаруженных в процессе из обработки.

1.2.4 Требования к организации выходных данных

Выходными данными при эксплуатации программного средства будут файлы, загруженные из удаленных узлов сети.

Недопустимо повреждение и внесение ошибок в файлы при загрузке и сохранении на компьютере пользователя.

Также, необходимо организовать хорошо воспринимаемый пользователем способ сохранения файлов, при котором дисковое пространство компьютера будет расходоваться эффективно.

1.2.5 Требования к временным характеристикам

Требования к временным характеристикам должны зависеть от объема обрабатываемых данных и доступных для использования вычислительных и сетевых ресурсов.

Таким образом, графический интерфейс программы должен иметь минимальное время отклика, зависимое от конфигурации компьютера, в то время как загрузка файла полностью зависит от возможности канала и сервера.

1.2.6 Требования к составу и параметрам технических и программных средств

Программное средство должно функционировать на ЭВМ со следующими минимальными характеристиками:

− процессор семейства Intel или AMD с тактовой частотой 600 МГц и более;

− жесткий диск объемом в 20 Гб;

− оперативная память 512Мб и более;

− сетевая карта Ethernet 100 Мбит;

− операционная система с графическим интерфейсом пользователя.

1.2.8 Требования к информационной и программной совместимости

Исходный код программы должен быть реализован на широко распространенном языке общего назначения, который позволяет реализовать следующие функции программного средства:

− создание графического интерфейса пользователя;

− работа с базами данных;

− работа с сетевыми ресурсами.

Язык реализации программного средства должен иметь большой набор готовых эффективных решений, библиотек или фреймворков для эффективной реализации функций программного средства.

Для долговременного хранения данных программа должна использовать популярное решение для управления базами данных, поддерживающее язык запросов SQL. Используемое решение должно обеспечивать надежность хранения данных, минимизировать риск их повреждения, гарантировать их целостность.

Графический интерфейс программы должен быть достаточно информативен, должен содержать понятные пользователю определения и наименования, состоять из компонентов, широко используемых в других программных продуктах для выполнения схожих функций. Цветовая гамма интерфейса должна быть нейтральна, не должна содержать резких контрастов.

# **2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

## **2.1 Описание функциональности ПС**

Для представления функциональной модели была выбрана диаграм-ма вариантов использования UML, которая отражает отношения между актерами и прецедентами и позволяет описать систему на концептуальном уровне. Прецедент соответствует отдельному сервису системы, определяет один из вариантов её использования и описывает типичный способ взаи-модействия пользователя с системой. UML предназначен для определения, визуализации, проектирования и документирования программных систем.

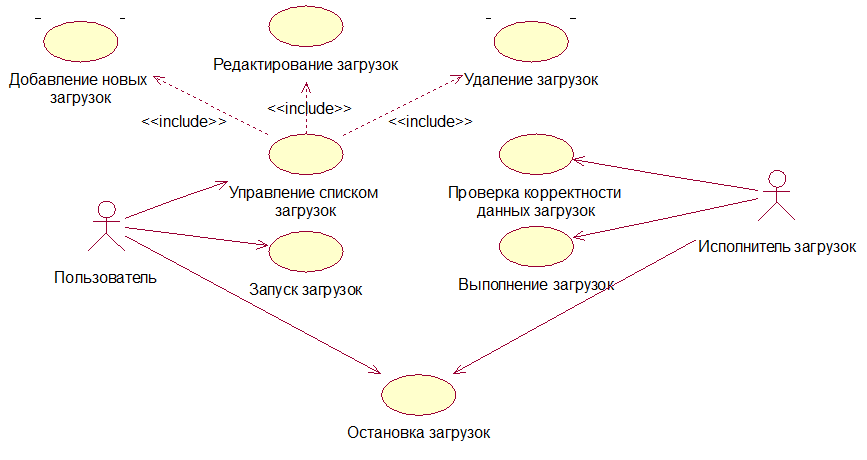


Рисунок 2.1 – Диаграмма вариантов использования

Диаграмма вариантов использования разрабатываемого программ-ного средства представлена на рисунке 2.1. На диаграмме можно выделить два основных составляющих элемента – актер и прецедент. Актер – стили-зованный человек, обозначающий набор ролей пользователя, взаимодей-ствующего с некоторой сущностью. Прецедент – эллипс с надписью, обо-значающий выполняемые системой действия, приводящие к наблюдаемым актером результатам.

На основании представленной диаграммы вариантов использования можно сделать вывод, что в системе будет существовать два основных актера: пользователь и исполнитель загрузок.

Рассмотрим каждый из прецедентов более подробно для каждого ак-тера.

2.1.1 Пользователь

Пользователю предоставляются следующие возможности:

− управление списком загрузок, что подразумевает добавление новых загрузок в список, удаление загрузок из него, а также редактирование данных загрузок;

− запуск загрузок: пользователь принимает решение о том, когда необходимо отправить загрузку на исполнение в Исполнитель загрузок;

− остановка загрузок: пользователь может прекратить исполнение загрузки для того, чтобы освободить вычислительные ресурсы или отложить на время ее исполнение.

Возможности пользователя ограничены для того, чтобы эффективнее автоматизировать процесс загрузки файлов. Пользователь выступает в роли администратора, в то время как исполнитель загрузок берет на себя роль исполнителя.

2.1.2 Исполнитель загрузок

Исполнитель загрузок – это фоновый процесс программного средства, который отвечает за проверку и исполнение загрузок. Пользователь не имеет доступа к возможностям исполнителя, однако может влиять на ход его работы через управление списком загрузок. Исполнитель загрузок выполняет следующие функции:

− проверка корректности данных загрузок: исполнитель загрузок проверяет возможность осуществления загрузки и сообщает пользователю о об ошибках в данных загрузки, если они есть;

− исполнение загрузок: после проверки корректности данных, исполнитель выполняет загрузку и информирует пользователя о ее завершении

− остановка загрузок: в случае обнаружения ошибки или завершения, исполнитель исключает загрузку из очереди на исполнение.

Исполнитель загрузок является отдельным компонентом программного средства, что позволит достичь высокой степени автоматизации при выполнении загрузки файлов.

## **2.2 Спецификация функциональных требований**

На основании анализа исходных данных для проектируемого программного средства можно сделать вывод, что основной целью является создание качественного программного продукта, позволяющего с минимальными трудозатратами решать существующие проблемы пользователей, такие как:

− эффективное управление расписанием загрузок;

− автоматизации процесса загрузки файлов;

− эффективное использование вычислительных и сетевых ресурсов.

В ходе разработки будут реализованы следующие возможности:

− редактирование списка планируемых загрузок;

− редактирование данных загрузок;

− проверка возможности осуществления загрузки;

− формирование очереди исполнения загрузок;

− исполнение загрузок;

− прохождение аутентификации и авторизации, если это необходимо;

− сохранение истории совершенных загрузок;

− сбор данных о количестве принятых данных.

Программное средство будет представлять собой прикладную программу для настольных компьютеров, которая работает в пользовательской среде операционной системы. Программное средство будет доступно всем пользователям компьютера.

# **3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

## **3.1 Разработка программной архитектуры**

Прежде чем приступить к разработке программного средства, необходимо определить основные программные модули, которые будут отвечать за функциональные возможности приложения.

Исходя из функциональных требований, можно описать модули, которые будут реализованы в программном средстве:

− база данных: база данных необходима для долговременного хранения данных загрузок, включая данные о ходе их выполнения исполнителем загрузок;

− графический интерфейс пользователя, который необходим для эффективного взаимодействия пользователя с программным средством и позволит уме в полной мере использовать все функции программного средства

− сетевой модуль, который будет осуществлять взаимодействие с удаленными узлами сети;

− модуль бизнес логики, который будет отвечать за обработку команд пользователя, а также управлять работой сетевого модуля.

Поскольку разрабатываемое программное средство является программой для персонального использования, то нет никаких особых требований к его процессу его развертывания.

## **3.2 Разработка модели базы данных**

Неотъемлемой частью конечного программного средства является база данных, используемая системой в процессе работы. Информационную модель предметной области можно представить на языке IDEF1X.

В модели использованы три типа связей: не идентифицирующая ноль-или-один-ко-многим, не идентифицирующая многие-ко-многим и идентифицирующая один-ко-многим. Первая обозначена штриховой линией с ромбом со стороны родительской сущности и кружком со стороны дочерней сущности, вторая – штриховой линией с ромбом со стороны родительской сущности и кружком со стороны дочерней сущности, третья – линией с кружком со стороны дочерней сущности. При наличии связи «один ко многим» одна запись в одной таблице связана с множеством записей в другой таблице. Связь между таблицами организуется на основе общего поля. На стороне «один» должно выступать ключевое поле, содержащее уникальные значения – такое поле называют внешним ключом. Значения на стороне «многие» могут повторяться.

Организация связей между таблицами обеспечивает целостность данных. Система не допустит, чтобы одноименные поля в разных таблицах имели разные значения. Ввод данных автоматически контролируется. Рассмотрим сущности по-отдельности.

3.2.1 Сущность «Загрузка»

Представляет данные о загрузке, планируемой к выполнению, выполняющейся или уже выполненной. Имеет следующие поля:

− «Идентификатор загрузки» – первичный ключ загрузки в базе данных;

− «Имя» – название, данное загрузке пользователем для того, чтобы ему было легче найти загрузку в списке;

− «Папка сохранения» – папке, куда будут сохранятся все загруженные файлы;

− «Приоритет» – приоритет загрузки при исполнении;

− «Статус» – отображает статус загрузки в текущий момент;

− «Дата создания» – дата внесения пользователем загрузки в список загрузок;

− «Сообщение» – описание ошибки выявленной при попытке выполнения загрузки.

Сущность имеет связь «один ко многим» с сущностью «Файл».

3.2.2 Сущность «Файл»

Представляет данные о конкретном файле который будет скачан в ходе выполнения загрузки. Имеет следующие поля:

– «Идентификатор файла» –первичный ключ файла в базе данных;

– «Идентификатор загрузки» – внешний ключ на сущность «Загрузка»;

– «Ссылка» – ссылка по которой можно загрузить файл;

– «Логин» – логин пользователя для прохождения авторизации на сервере при выполнении загрузки файла;

– «Пароль» – пароль пользователя для прохождения авторизации на сервере при выполнении загрузки файла;

– «Статус» – текущий статус файла;

– «Имя» – имя файла на жестком диске компьютера;

– «Сообщение» – описание ошибки, выявленной при попытке загрузить файл;

«Дата создания» – дата внесения файла в список загрузок.

Сущность имеет связь «Многие к одному» с сущностью «Загрузка».

## **3.3 Разработка алгоритма выполнения загрузки**

Выполнение загрузки состоит из проверки загрузки и последовательной загрузки всех указанных файлов. Проверка загрузки состоит из проверки того, что папка назначения существует, что программа может записывать файлы в нее, и проверки каждого файла перед скачиванием.

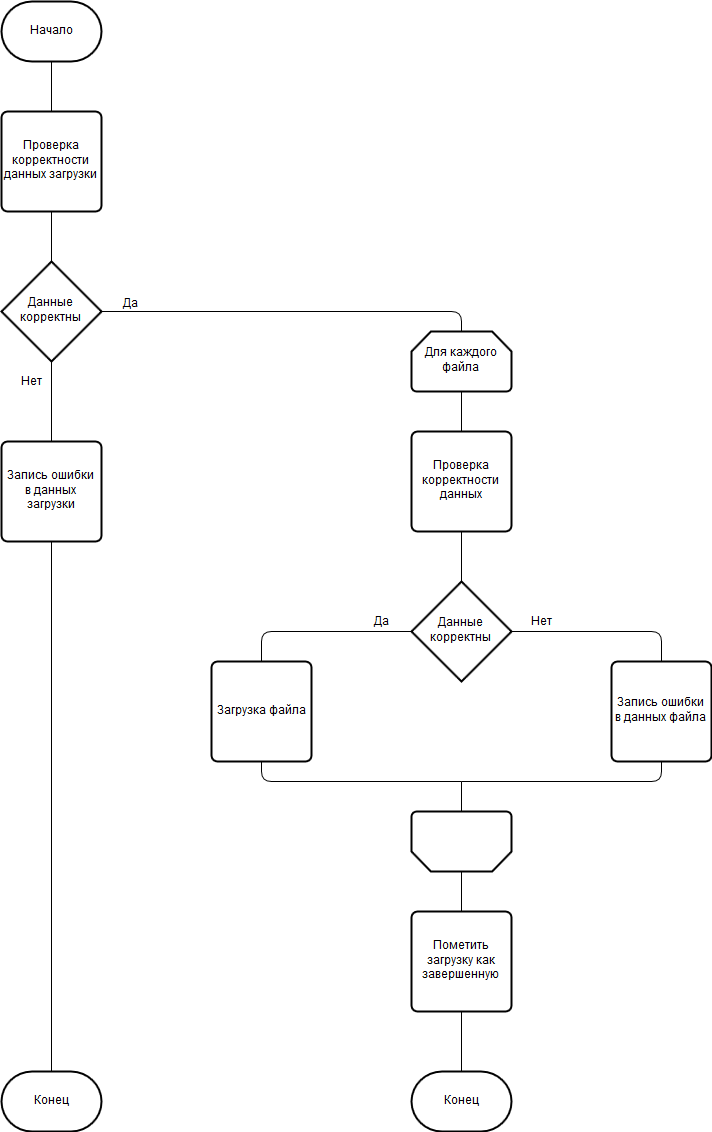


Рисунок 3.1 – Алгоритм выполнения загрузки.

Обнаружение ошибок при проверке загрузки ведет к тому, что загрузка будет удалена из очереди на выполнение. Обнаружение ошибки при скачивании файла приведет к тому, что файл будет помечен как имеющий ошибки, а исполнитель загрузок перейдет к скачиванию следующего файла.

Проверка файла представляет собой проверку доступа к серверу с файлом, наличия файла на удаленном сервере, разрешения на загрузку этого файла, а также того, что файл поместится на жестком диске компьютера.

По завершению загрузки файла, файл будет помечен как загруженный, а после загрузки всех файлов, загрузка будет отмечена как завершенная. Если в ходе выполнения загрузки, были обнаружены ошибки при скачивании файлов, то загрузка будет помечена как имеющая ошибки.

# **4 СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА**

Спецификация функциональных требований и спроектированная архитектура программного средства служат фундаментом, на котором основывается выбор наиболее подходящих технологий для разработки программного средства. Успешное и обоснованное завершение данных этапов позволит создать расширяемое, надёжное и функциональное приложение, призванное решать поставленные задачи.

## **4.1 Обоснование выбора средств разработки ПС**

Для разработки программного продукта были выбрана технология MVVM, протокол FTP и языки Java и SQL , которые позволяют создавать настольные приложения с большой эффективностью и в сжатые сроки.

4.1.1 Технология MVVM

Шаблон Model-View-ViewModel (MVVM) — применяется при проектировании архитектуры приложения. Первоначально был представлен сообществу Джоном Госсманом (John Gossman) в 2005 году как модификация шаблона Presentation Model. MVVM ориентирован на современные платформы разработки.

Шаблон MVVM делится на три части:

– Модель (англ. Model), так же, как в классической MVC, Модель представляет собой бизнес логику и фундаментальные данные, необходимые для работы приложения;

– Представление (англ. View) — это графический интерфейс, то есть окно, кнопки и т. п. Представление является подписчиком на событие изменения значений свойств или команд, предоставляемых Моделью Представления. В случае, если в Модели Представления изменилось какое-либо свойство, то она оповещает всех подписчиков об этом, и Представление, в свою очередь, запрашивает обновленное значение свойства из Модели Представления. В случае, если пользователь воздействует на какой-либо элемент интерфейса, Представление вызывает соответствующую команду, предоставленную Моделью Представления;

– Модель Представления (англ. ViewModel) является, с одной стороны, абстракцией Представления, а с другой, предоставляет обёртку данных из Модели, которые подлежат связыванию. То есть, она содержит Модель, которая преобразована к Представлению, а также содержит в себе команды, которыми может пользоваться Представление, чтобы влиять на Модель.

MVVM используется для разделения модели и её представления, что необходимо для изменения их отдельно друг от друга. Например, разработчик задает логику работы с данными, а дизайнер соответственно работает с пользовательским интерфейсом.

4.1.2 Протокол FTP

FTP (англ. File Transfer Protocol — протокол передачи файлов) — стандартный протокол, предназначенный для передачи файлов по TCP-сетям (например, Интернет). Использует 21-й порт. FTP часто используется для загрузки сетевых страниц и других документов с частного устройства разработки на открытые сервера хостинга.

Протокол построен на архитектуре «клиент-сервер» и использует разные сетевые соединения для передачи команд и данных между клиентом и сервером. Пользователи FTP могут пройти аутентификацию, передавая логин и пароль открытым текстом, или же, если это разрешено на сервере, они могут подключиться анонимно. Можно использовать протокол SSH для безопасной передачи, скрывающей (шифрующей) логин и пароль, а также шифрующей содержимое.

4.1.3 База данных SQLite

SQLite — компактная встраиваемая реляционная база данных.

Слово «встраиваемый» (embedded) означает, что SQLite не использует парадигму клиент-сервер, то есть движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а представляет собой библиотеку, с которой программа компонуется, и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций (API) библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа. Простота реализации достигается за счёт того, что перед началом исполнения транзакции записи весь файл, хранящий базу данных, блокируется; ACID-функции достигаются в том числе за счёт создания файла журнала.

4.1.4 Язык программирования Java

Java — сильно типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, с помощью виртуальной Java-машины.

Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание.

Основные возможности:

– автоматическое управление памятью;

– расширенные возможности обработки исключительных ситуаций;

– богатый набор средств фильтрации ввода-вывода;

– набор стандартных коллекций: массив, список, стек и т. п.;

– наличие простых средств создания сетевых приложений (в том числе с использованием протокола RMI);

– наличие классов, позволяющих выполнять HTTP-запросы и обрабатывать ответы;

– встроенные в язык средства создания многопоточных приложений, которые потом были портированы на многие языки (например, python);

– унифицированный доступ к базам данных: на уровне отдельных SQL-запросов — на основе JDBC, SQLJ, и на уровне концепции объектов, обладающих способностью к хранению в базе данных — на основе Java Data Objects и Java Persistence API;

– поддержка обобщений (начиная с версии 1.5);

– поддержка лямбд, замыканий, встроенные возможности функционального программирования (с 1.8);

– множество вариантов реализации многопоточных программ.

Язык программирования Java удовлетворяет всем требованиям к языку программирования, предъявляемым программным средством.

## **4.2 Используемые модули и фреймворки**

Для создания программного средства с большей эффективностью были выбран фреймворк Spring.

4.2.1 Фреймворк Spring

Spring Framework (или коротко Spring) — универсальный фреймворк с открытым исходным кодом для Java-платформы.

Spring обеспечивает решения многих задач, с которыми сталкиваются Java-разработчики и организации, которые хотят создать информационную систему, основанную на платформе Java. Из-за широкой функциональности трудно определить наиболее значимые структурные элементы, из которых он состоит.

Центральной частью Spring является контейнер Inversion of Control, который предоставляет средства конфигурирования и управления объектами Java с помощью рефлексии. Контейнер отвечает за управление жизненным циклом объекта: создание объектов, вызов методов инициализации и конфигурирование объектов путём связывания их между собой.

Spring предоставляет свой слой доступа к базам данных и поддерживает все популярные СУБД. Для каждой СУБД фреймворк предоставляет следующие возможности:

– управление ресурсами — автоматическое получение и освобождение ресурсов базы данных;

– обработка исключений — перевод исключений при доступе к данным в исключения Spring;

– транзакционность — прозрачные транзакции в операциях с данными;

– распаковка ресурсов — получение объектов базы данных из пула соединений;

– абстракция для обработки BLOB и CLOB.

Таким образом, фреймворк Spring позволяет эффективно реализовать все необходимые функции программного рседства.

## **4.3 Описание классов и методов**

В системе используется шаблон проектирования MVVM, благодаря которому код понятен по своей структуре, а логика и представление разделены между собой. Рассмотрим основные классы приложения.

4.3.1 Класс DonwloadsService

Класс занимается обработкой данных загрузок, включая добавление и удаление загрузок из списка, отправкой загрузок на исполнении и приостановку их исполнения. Класс содержит следующие методы:

– createDownload – добавляет новую загрузку в список;

– startDownloads – добавляет указанные загрузки в список на ипсолнение;

– stopDownloads – удаляет указанные загрузки из списка на исполнении;

– deleteDownloads – удаляет указанные загрузки из сипска;

– updateDownload – сохраняет загрузку с измененными данными.

Класс реализует функции управления загрузками, доступные пользователю программы.

4.3.2 Класс DownloadsExecutor

Класс представляет собой внутреннюю службу, которая занимается исполнением загрузок. Работа службы в основном автоматизирована, а пользователи взаимодействуют с ней только в том случае, если хотят остановить выполнение загрузки.

Имеет следующий доступный метод: stopDownloadExecution – используется для немедленного завершения выполнения загрузки.

4.3.3 Класс DownloadsViewModel

Класс используется как механизм передачи сообщений об изменениях даных загрузок в графический интерфейс приложения. Основные методы:

* subscribeOnDownloadsListChange – позволяет клиенту подписаться на обновление данных загрузок;
* emitDownloadsListChange – сообщает всем подписавшимся клиентам о том, что данные загрузок изменились.

# **5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

Проведено тестирование программного средства. Целью данного ис-пытания было ознакомление с программным средством и проверка его работоспособности.

Установка и тестирование программного средства производилась на персональном компьютере с установленной операционной системой Windows 7, версия виртуальной машины языка Java – 1.8.0-151.

Таблица 5.1 – набор тест-кейсов модуля управления загрузками

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № тест-кейса | Тестируемая функциональность | Порядок действий | Ожидаемый результат | Итог |
| 1 | 2 | 3 | 4 | 5 |
| 1 | Создание загрузки. Валидация ошибок | 1. Нажать на кнопку «Добавить».  2. Нажать на кнопку «Сохранить». | 1. Сообщение об ошибке: «Не указаны ссылки на файлы». | Тест успешно пройден |
| 2 | Создание загрузки. Валидация ошибок | 1. Нажать на кнопку «Добавить».  2. Очистить поле «Имя»  3.Нажать на кнопку «Сохранить». | 1. Сообщение об ошибке «Не задано имя загрузки». | Тест успешно пройден |
| 3 | Создание загрузки. Валидация ошибок. | 1. Нажать на кнопку добавить.  2. Ввести в поле «Имя» имя уже существующей загрузки.  3. Нажать на кнопку «Сохранить». | 1. Сообщение об ошибке «Загрузка с таким именем уже существует» | Тест успешно пройдем |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 4 | Создание загрузки | 1. Нажать на кнопку «Добавить»  2. Заполнить все пустые поля  3. Нажать кнопку сохранить | 1. Новая загрузка появится в списке | Тест успешно пройден |
| 5 | Удаление загрузки | 1.Выбрать загрузку из списка.  2.Нажать кнопку «Удалить» | 1. Загрузка исчезнет из списка. | Тест успешно пройден |
| 6 | Запуск загрузки | 1.Выбрать загрузку из списка.  2.Нажать кнопку «Запустить» | 1. Статус загрузки изменится на «Загружается» | Тест успешно пройден |
| 7 | Исполнение загрузки | 1.Выбрать загрузку из списка.  2.Нажать кнопку «Запустить»  3.Дождать изменения статуса на «Загружено» | 1. Загруженные файлы в папке для сохранения загрузки | Тест успешно пройден |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 8 | Остановка загрузки | 1.Выбрать загрузку из списка.  2.Нажать кнопку «Запустить»  3. Нажать кнопку «Остановить» | 1.Статус загрузки сменится «Остановлено» | Тест успешно пройден |

Все созданные тест-кейсы выполнены успешно, из чего можно сделать вывод, что программное средство удовлетворяет предъявляемым к нему требованиям.

# **6 РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ**

## **6.1 Установка программного средства**

Для полноценной работы программного средства необходим компьютер следующей конфигурации:

− процессор семейства Intel или AMD с тактовой частотой 600 МГц и более;

− жесткий диска объемом в 20 гб;

− оперативная память 512МБ и более;

− сетевая карта Ethernet 100 Мбит;

− операционная система с поддержкой виртуальной машины Java;

− операционная система с графическим интерфейсом пользователя.

Также для работы программного средства необходимо установить виртуальную машину языка Java, версии 1.8 и выше.

После того, как виртуальная машина будет установлена, необходимо скачать исполняемый jar файл из репозитория программы и запустить его командой :

java –jar dmanager.jar

Данная команда запустит программу, после чего пользователь увидит ее основное окно.

## **6.2 Использование программного средства**

На рисунке 6.1 представлено основное окно программы.

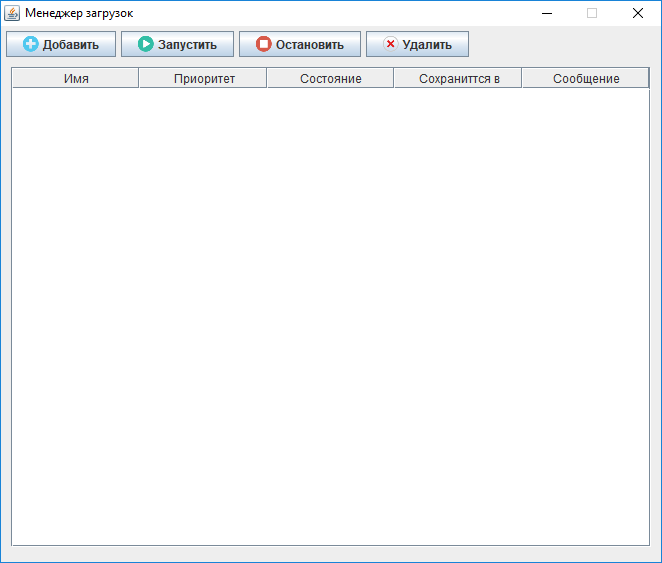


Рисунок 6.1 – основное окно программы.

Основное окно программы содержит кнопки управления загрузками и список управляемых загрузок.

В списке загрузок демонстрируется информация о состоянии каждой загрузки. Информация представлена в виде таблицы со следующими столбцами:

− «Имя» – имя загрузки для быстрого ее поиска;

− «Приоритет» – место загрузки в очереди на исполнение;

− «Состояние» – текущий статус загрузки на этапе ее выполнения программой;

− «Сохранится в» − папка, куда будут записаны файлы загрузки;

− «Сообщение» − сообщение об ошибке при выполнении загрузки.

Чтобы добавить новую загрузку в список, необходимо нажать на конку добавить на панели управления, и заполнить появившуюся форму информацией новой загрузки.

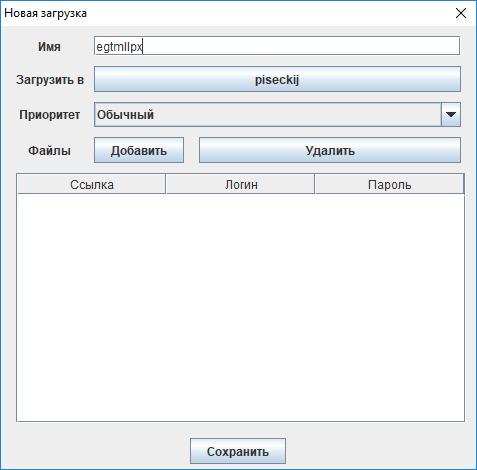


Рисунок 6.2 – окно создания новой загрузки.

Поле «Имя» содержит имя загрузки, которое может быть использовано пользователем для быстрого поиска загрузки в списке. Поле так же имеет значение по умолчанию. Для каждой новой загрузки программа генерирует уникальное имя.

В поле «Загрузить в» надо указать папку. В которую программа будет сохранять файлы при загрузке. По умолчанию значение поле указывает на домашнюю папку пользователя операционной системы.

В поле приоритет пользователь может выбрать приоритет загрузки из выпадающего списка. По умолчанию загрузки имеют приоритет «Обычный».

Таблица «Файлы» содержит список файлов загрузки. Для того чтобы добавить в нее новый файл, пользователя необходимо нажать на кнопку «Добавить», а затем заполнить поля новой строки таблицы.

Обязательным для заполнения полем строки является поле «Ссылка», которое содержит ссылку на файл на удаленном сервере.

Поля «Логин» и «Пароль» не обязательны для заполнения, так как содержат информацию и данных авторизации при загрузке.

Программа выполняет проверку введенных пользователем данных, и сообщает об ошибках, если такие имеются.

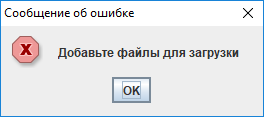


Рисунок 6.3 – сообщение об ошибке

Для того чтобы добавить загрузку в список, пользователя необходимо нажать кнопку сохранить после того, как он заполнит все необходимые поля.

Для того чтобы запустить выполнение загрузки, пользователю необходимо выбрать загрузку в списке и нажать кнопку «Запустить», после чего состояние выбранных загрузок изменится на «Загружается».

По окончанию выполнения загрузки программа изменит состояние выполненной загрузки на «Загружено», если загрузка прошла успешно, или на «Ошибка» − если во время выполнения загрузки были обнаружены ошибки.

Загруженный файлы пользователь может найти в папке назначения, где будет находится папка с именем загрузки.

Исправить ошибки в данных загрузки пользователь может в специальной форме, аналогичной форме создания загрузки, которую он может открыть, кликнув два раза на элементе списка загрузок.

# **ЗАКЛЮЧЕНИЕ**

В ходе работы над курсовым проектом проанализированы программные средства, задача которых помогать пользователям эффективно управлять загрузкой файлов с удаленных узлов сети. Исследованы разные направления и подходы к созданию таких программ.

Проведен анализ предметной области, исследованы существующие аналоги. Результатом этого анализа явилось обобщение достоинств и не-достатков существующих решений, которые учтены при разработке функ-циональных требований к разработанному программному средству.

На основе функциональных требований было произведено проекти-рование программного средства. В нем представлены разработка архитек-туры ПС, разработка модели базы данных, разработка алгоритма про-граммного средства и алгоритмов отдельных модулей. Детально рассмотрен алгоритм выполнения загрузки.

Согласно требованиям были сформированы тестовые наборы, кото-рые успешно пройдены в ходе тестовых испытаний программного сред-ства. Успешность прохождения тестов показывает корректность работы программы с реальными данными, соответствие функциональным требо-ваниям.

На завершающем этапе подробно описана методика использования программного средства, которая позволяет за достаточно быстрые сроки освоить работу с программой.

Главной целью при разработке программного средства было по-ставлено устранение основных недостатков существующих аналогов, а также разработка и реализация всего основного функционала. В ходе ра-боты над курсовым проектом эта цель была успешно достигнута: разра-ботан минималистический, удобный, интуитивно понятный пользователь-ский интерфейс; программное средство имеет повышенную надежность хранения данных и отказоустойчивость благодаря использованию совре-менных технологий защиты данных и построению грамотной архитектуры системы; разработанное ПС позволяет автоматизировать деятельность пользователя по загрузке файлов из удаленных узлов сети.

Т.к. программа является настольным приложением написанным на языке программирования Java, то может работать на большом наборе разных конфигураций компьютеров.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Спецификация протокола FTP [Электронный ресурс]. – Минск,2017 – Режим доступа: https://tools.ietf.org/html/rfc959. – Дата доступа: 01.12.2017.
2. Спецификация протокола SFTP [Электронный ресурс]. – Минск,2017 – Режим доступа: https://tools.ietf.org/html/draft-ietf-secsh-filexfer-13. – Дата доступа: 01.12.2017.
3. Спецификация протокола SSH [Электронный ресурс] – Минск,2017 – Режим доступа: https://tools.ietf.org/html/rfc4251. – Дата доступа: 01.12.2017.
4. Документация базового модуля фреймворка Spring [Электронный ресурс] – Минск,2017 – Режим доступа: https://docs.spring.io/spring/docs/5.0.2.RELEASE/spring-framework-reference/core.html. – Дата доступа: 01.12.2017.
5. Документация модуля доступа к данным фреймворка Spring [Электронный ресурс] – Минск,2017 – Режим доступа: https://docs.spring.io/spring/docs/5.0.2.RELEASE/spring-framework-reference/data-access.html. – Дата доступа: 01.12.2017.
6. Руководство по созданию диалоговых окон средствами библиотеки Swing [Электронный рксурс] – Минск,2017 – Режим доступа: https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html. – Дата доступа: 01.12.2017.
7. Руководство по использованию таблиц для отображения данных средствами библиотеки Swing [Электронный ресурс] – Минск,2017 – Режим доступа: https://docs.oracle.com/javase/tutorial/uiswing/components/table.html. – Дата доступа: 01.12.2017.
8. Визуальный гид по менеджерам компоновки компонентов библиотеки Swing [Электронный ресурс] – Минск,2017 – Режим доступа: https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html. – Дата доступа: 01.12.2017.
9. Руководство по использования менеджера компоновки «GridLayout» [Электронный ресурс] – Минск,2017 – Режим доступа: https://docs.oracle.com/javase/tutorial/uiswing/layout/grid.html. – Дата доступа: 01.12.2017.
10. Особенности синтаксиса SQL: [Электронный ресурс] : официальный сайт программного продукта SQLite, - Минск 2017 – Режим доступа: https://sqlite.org/lang.html. – Дата доступа: 01.12.2017.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРИЛОЖЕНИЯ**

**package** by.bsuir.ksis.dmanager.domain;  
  
**import** lombok.Builder;  
**import** lombok.Data;  
  
**import** java.time.LocalDateTime;  
  
*/\*\*  
 \** ***@author*** *Vladislav Piseckij  
 \*/*@Data  
@Builder  
**public class** Download {  
   
 **private** Integer **id**;  
 **private** String **name**  
 **private** String **destination**;  
 **private** Priority **priority**;  
 **private** Status **status**;  
 **private** LocalDateTime **created**;  
 **private** String **username**;  
 **private** String **password**;  
 **private long allFilesCount**;  
 **private long totalBytes**;  
 **private long loadedBytes**;  
  
 **public enum** Priority {  
 ***HIGH***(**"Высокий"**, 1),  
 ***NORMAL***(**"Обычный"**, 2),  
 ***LOW***(**"Низкий"**, 3);  
  
 **private** String **value**;  
 **private int position**;  
  
 Priority(String value, **int** position) {  
 **this**.**value** = value;  
 **this**.**position** = position;  
 }  
  
 **public int** position() {  
 **return position**;  
 }  
  
 @Override  
 **public** String toString() {  
 **return value**;  
 }  
 }  
  
 **public enum** Status {  
 ***WAIT***(**"Остановлена"**),  
 ***ERROR***(**"Ошибка"**),  
 ***RUN***(**"Загружается"**),  
 ***END***(**"Завершена"**);  
  
 **private** String **value**;  
  
 Status(String value) {  
 **this**.**value** = value;  
 }  
  
  
 @Override  
 **public** String toString() {  
 **return value**;  
 }  
 }  
}

**package** by.bsuir.ksis.dmanager.domain;  
  
**import** lombok.Builder;  
**import** lombok.Data;  
  
*/\*\*  
 \** ***@author*** *Vladislav Piseckij  
 \*/*@Data  
@Builder  
**public class** Item {  
   
 **private** Integer **id**;  
 **private** Integer **downloadId**;  
 **private** String **link**;  
 **private** String **destination**;  
 **private** Status **status**;  
 **private** String **name**;  
 **private long totalBytes**;  
 **private long loadedBytes**;  
  
 **public enum** Status {  
 ***WAIT***(**"Готов к загрузке"**),  
 ***ERROR***(**"Не удается загрузить"**),  
 ***RUN***(**"Загружается"**),  
 ***END***(**"Загружен"**);  
  
 **private** String **value**;  
  
 Status(String value) {  
 **this**.**value** = value;  
 }  
   
 @Override  
 **public** String toString() {  
 **return value**;  
 }  
 }  
}

**package** by.bsuir.ksis.dmanager.domain;  
  
**import** lombok.Builder;  
**import** lombok.Data;  
  
@Data  
@Builder  
**public class** ItemsInfo {  
  
 **private** Integer **downloadId**;  
 **private long count**;  
 **private long loadedBytes**;  
 **private long totalBytes**;  
}

**package** by.bsuir.ksis.dmanager.logic;  
  
**import** by.bsuir.ksis.dmanager.domain.Download;  
**import** by.bsuir.ksis.dmanager.domain.Item;  
**import** by.bsuir.ksis.dmanager.persistence.DownloadDAO;  
**import** by.bsuir.ksis.dmanager.persistence.ItemDAO;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Service;  
**import** org.springframework.transaction.annotation.Transactional;  
  
**import** java.io.File;  
**import** java.time.LocalDateTime;  
**import** java.util.List;  
**import** java.util.Objects;  
**import** java.util.stream.Collectors;  
  
**import static** by.bsuir.ksis.dmanager.ui.Util.*getRandomString*;  
  
*/\*\*  
 \** ***@author*** *Vladislav Piseckij  
 \*/*@Service  
@Transactional  
**public class** DownloadsService {  
   
 **private final** DownloadDAO **downloadDAO**;  
 **private final** ItemDAO **itemDAO**;  
 **private final** DownloadsViewModel **downloadsViewModel**;  
  
 @Autowired  
 **public** DownloadsService(DownloadDAO downloadDAO, ItemDAO itemDAO, DownloadsViewModel downloadsViewModel) {  
 **this**.**downloadDAO** = downloadDAO;  
 **this**.**itemDAO** = itemDAO;  
 **this**.**downloadsViewModel** = downloadsViewModel;  
 }  
   
 **public** Result create(NewDownload newDownload) {  
 Download download = Download.*builder*()  
 .name(newDownload.getName())  
 .priority(newDownload.getPriority())  
 .username(Objects.*toString*(newDownload.getUsername(), **"ANONYMOUS"**))  
 .password(newDownload.getPassword())  
 .status(newDownload.isStart() ? Download.Status.***WAIT*** : Download.Status.***RUN***)  
 .created(LocalDateTime.*now*())  
 .build()  
 ;  
 **if** (!checkDownloadNameUnique(download)) {  
 **return** Result.*fail*(**"Закачка с таким именем уже существует"**);  
 }  
  
 String itemsDestination = newDownload.getDestination() + File.***pathSeparator*** + download.getName();  
  
  
 List<Item> files = newDownload.getLinks().stream()  
 .map(link -> Item.*builder*()  
 .link(link)  
 .destination(itemsDestination)  
 .name(*getRandomString*(10))  
 .status(Item.Status.***WAIT***)  
 .build()  
 )  
 .collect(Collectors.*toList*());  
 download = **downloadDAO**.create(download);  
 **final** Integer downloadId = download.getId();  
 files.stream().peek(file -> file.setDownloadId(downloadId)).forEach(**itemDAO**::create);  
  
 **downloadsViewModel**.emitDownloadsListChange();  
   
 **return** Result.*success*();  
 }  
  
 **public** Result startDownloads(List<Download> downloads) {  
 **long** updatedCount = downloads.stream()  
 .filter(d -> Download.Status.***RUN*** != d.getStatus())  
 .peek(d -> d.setStatus(Download.Status.***RUN***))  
 .peek(**downloadDAO**::update)  
 .count();  
  
 **if** (updatedCount > 0) {  
 **downloadsViewModel**.emitDownloadsListChange();  
 }  
  
 **return** Result.*success*();  
 }  
  
 **public** Result stopDownloads(List<Download> downloads) {  
 *//****todo: need to stop running downlods before delete* long** stoppedCount = downloads.stream()  
 .filter(d -> Download.Status.***RUN*** == d.getStatus())  
 .peek(d -> d.setStatus(Download.Status.***WAIT***))  
 .peek(**downloadDAO**::update)  
 .count();  
  
 **if** (stoppedCount > 0) {  
 **downloadsViewModel**.emitDownloadsListChange();  
 }  
  
 **return** Result.*success*();  
 }  
  
 **public** Result deleteDownloads(List<Download> downloads) {  
 *//****todo: need to stop running downlods before delete* long** deletedCount = downloads.stream()  
 .peek(d -> **itemDAO**.deleteByDownload(d.getId()))  
 .peek(d -> **downloadDAO**.delete(d.getId()))  
 .count();  
  
 **if** (deletedCount > 0) {  
 **downloadsViewModel**.emitDownloadsListChange();  
 }  
  
 **return** Result.*success*();  
 }  
  
 **private boolean** checkDownloadNameUnique(Download download) {  
 **return downloadDAO**.findByName(download.getName()) == **null**;  
 }  
}

**package** by.bsuir.ksis.dmanager.logic;  
  
**import** by.bsuir.ksis.dmanager.domain.Download;  
**import** by.bsuir.ksis.dmanager.domain.ItemsInfo;  
**import** by.bsuir.ksis.dmanager.persistence.DownloadDAO;  
**import** by.bsuir.ksis.dmanager.persistence.ItemDAO;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Service;  
  
**import** java.util.\*;  
**import** java.util.concurrent.ExecutorService;  
**import** java.util.concurrent.Executors;  
**import** java.util.function.Consumer;  
  
@Service  
**public class** DownloadsViewModel {  
  
 **private final** ExecutorService **executor** = Executors.*newSingleThreadExecutor*();  
 **private** DownloadDAO **downloadDAO**;  
 **private** ItemDAO **itemDAO**;  
  
 @Autowired  
 **public** DownloadsViewModel(DownloadDAO downloadDAO, ItemDAO itemDAO) {  
 **this**.**downloadDAO** = downloadDAO;  
 **this**.**itemDAO** = itemDAO;  
 }  
  
 **private final** Set<Consumer<List<Download>>> **downloadsListListeners** = **new** HashSet<>();  
  
 **public void** subscribeOnDownloadsListChange(Consumer<List<Download>> listener) {  
 **if** (listener != **null**) {  
 **downloadsListListeners**.add(listener);  
 }  
 }  
  
 **public void** emitDownloadsListChange() {  
 **executor**.submit(() -> {  
 List<Download> downloads = Collections.*unmodifiableList*(**downloadDAO**.list());  
 Map<Integer, ItemsInfo> downloadsItemsInfo = **itemDAO**.itemsInfo();  
 downloads.forEach(d -> {  
 ItemsInfo i = downloadsItemsInfo.get(d.getId());  
 **if** (i == **null**) **return**;  
 d.setAllFilesCount(i.getCount());  
 d.setLoadedBytes(i.getLoadedBytes());  
 d.setTotalBytes(i.getTotalBytes());  
 });  
 **downloadsListListeners**.forEach(listener -> listener.accept(downloads));  
 });  
 }  
}

**package** by.bsuir.ksis.dmanager.logic;  
  
**import** by.bsuir.ksis.dmanager.domain.Download;  
**import** lombok.Builder;  
**import** lombok.Data;  
  
**import** java.util.List;  
  
*/\*\*  
 \** ***@author*** *Vladislav Piseckij  
 \*/*@Data  
@Builder  
**public class** NewDownload {  
   
 **private final** List<String> **links**;  
 **private final** String **destination**;  
 **private final** String **name**;  
 **private final** Download.Priority **priority**;  
 **private** String **username**;  
 **private** String **password**;  
 **private boolean start**;  
}

**package** by.bsuir.ksis.dmanager.logic;  
  
**import** lombok.Data;  
  
*/\*\*  
 \** ***@author*** *Vladislav Piseckij  
 \*/*@Data  
**public class** Result {  
   
 **private static final** Result ***SUCCESS*** = **new** Result(Status.***SUCCESS***, **""**);  
  
 **public static** Result success() {  
 **return *SUCCESS***;  
 }  
  
 **public static** Result fail(String message) {  
 **return new** Result(Status.***FAIL***, message);  
 }  
   
 **private final** Status **status**;  
   
 **private final** String **message**;  
  
 **public boolean** isSuccess() {  
 **return status** == Status.***SUCCESS***;  
 }  
  
 **public enum** Status {  
 ***SUCCESS***,  
 ***FAIL*** }  
   
}

**package** by.bsuir.ksis.dmanager.persistence;  
  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.jdbc.core.JdbcTemplate;  
**import** org.springframework.jdbc.core.PreparedStatementCreator;  
**import** org.springframework.jdbc.support.GeneratedKeyHolder;  
**import** org.springframework.jdbc.support.KeyHolder;  
  
**import** javax.sql.DataSource;  
  
*/\*\*  
 \** ***@author*** *Vladislav Piseckij  
 \*/***class** DAO {  
   
 JdbcTemplate **jdbcTemplate**;  
   
 @Autowired  
 **public void** setDataSource(DataSource dataSource) {  
 **this**.**jdbcTemplate** = **new** JdbcTemplate(dataSource);  
 }  
  
 Number insert(PreparedStatementCreator psc) {  
 KeyHolder keyHolder = **new** GeneratedKeyHolder();  
 **jdbcTemplate**.update(psc, keyHolder);  
 **return** keyHolder.getKey();  
 }  
}

**package** by.bsuir.ksis.dmanager.persistence;  
  
**import** by.bsuir.ksis.dmanager.domain.Download;  
**import** org.springframework.jdbc.core.RowMapper;  
**import** org.springframework.stereotype.Repository;  
**import** org.springframework.transaction.annotation.Transactional;  
  
**import** java.sql.PreparedStatement;  
**import** java.sql.Statement;  
**import** java.sql.Timestamp;  
**import** java.util.List;  
  
*/\*\*  
 \** ***@author*** *Vladislav Piseckij  
 \*/*@Repository  
@Transactional  
**public class** DownloadDAO **extends** DAO {  
   
 **private static final** String ***SELECT*** = **""** +  
 **"select\n"** +  
 **" id, name, priority, status, created, username, password\n"** +  
 **"from\n"** +  
 **" download"**;  
   
 **private static final** String ***SELECT\_BY\_ID*** = **""** +  
 ***SELECT*** + **"\n"** +  
 **"where\n"** +  
 **" id = ?"**;  
   
 **private static final** String ***CREATE*** = **""** +  
 **"insert into\n"** +  
 **" download(name, priority, status, created, username, password)\n"** +  
 **"values\n"** +  
 **" (?, ?, ?, ?, ?, ?)"**;  
  
 **public** Download create(Download download) {  
 Integer id = (Integer)insert(  
 con -> {  
 PreparedStatement statement = con.prepareStatement(***CREATE***, Statement.***RETURN\_GENERATED\_KEYS***);  
 statement.setString(1, download.getName());  
 statement.setString(2, download.getPriority().name());  
 statement.setString(3, download.getStatus().name());  
 statement.setTimestamp(4, Timestamp.*valueOf*(download.getCreated()));  
 statement.setString(5, download.getUsername());  
 statement.setString(6, download.getPassword());  
  
 **return** statement;  
 }  
 );  
 download.setId(id);  
  
 **return** download;  
 }  
  
 **private static final** String ***SELECT\_BY\_NAME*** = **""** +  
 ***SELECT*** + **"\n"** +  
 **"where\n"** +  
 **" name = ?"**;  
  
 @Transactional(readOnly = **true**)  
 **public** Download findByName(String name) {  
 List<Download> downloads = **jdbcTemplate**.query(***SELECT\_BY\_NAME***, **new** Object[]{name}, ***DOWNLOAD\_ROW\_MAPPER***);  
  
 **return** downloads.isEmpty() ? **null** : downloads.get(0);  
 }  
  
 @Transactional(readOnly = **true**)  
 **public** List<Download> list() {  
 **return jdbcTemplate**.query(***SELECT***, ***DOWNLOAD\_ROW\_MAPPER***);  
 }  
  
 **private static final** String ***UPDATE*** = **""** +  
 **"update\n"** +  
 **" download\n"** +  
 **"set\n"** +  
 **" name = ?\n"** +  
 **" ,priority = ?\n"** +  
 **" ,status = ?\n"** +  
 **" ,created = ?\n"** +  
 **" ,username = ?\n"** +  
 **" ,password = ?\n"** +  
 **"where\n"** +  
 **" id = ?"**;  
  
 **public void** update(Download download) {  
 **jdbcTemplate**.update(  
 ***UPDATE***,  
 download.getName(),  
 download.getPriority().name(),  
 download.getStatus().name(),  
 Timestamp.*valueOf*(download.getCreated()),  
 download.getUsername(),  
 download.getPassword(),  
 download.getId()  
 );  
 }  
  
 **private static final** String ***DELETE*** = **""** +  
 **"delete from\n"** +  
 **" download\n"** +  
 **"where\n"** +  
 **" id = ?"**;  
  
 **public void** delete(Integer id) {  
 **jdbcTemplate**.update(***DELETE***, id);  
 }  
  
 **private static final** RowMapper<Download> ***DOWNLOAD\_ROW\_MAPPER*** = (rs, rowNum) -> Download.*builder*()  
 .id(rs.getInt(**"id"**))  
 .name(rs.getString(**"name"**))  
 .priority(Download.Priority.valueOf(rs.getString(**"priority"**)))  
 .status(Download.Status.valueOf(rs.getString(**"status"**)))  
 .created(rs.getTimestamp(**"created"**).toLocalDateTime())  
 .username(rs.getString(**"username"**))  
 .password(rs.getString(**"password"**))  
 .build();  
}

**package** by.bsuir.ksis.dmanager.persistence;  
  
**import** by.bsuir.ksis.dmanager.domain.Item;  
**import** by.bsuir.ksis.dmanager.domain.ItemsInfo;  
**import** org.springframework.stereotype.Repository;  
**import** org.springframework.transaction.annotation.Transactional;  
  
**import** java.sql.PreparedStatement;  
**import** java.sql.Statement;  
**import** java.util.HashMap;  
**import** java.util.Map;  
  
*/\*\*  
 \** ***@author*** *Vladislav Piseckij  
 \*/*@Repository  
@Transactional  
**public class** ItemDAO **extends** DAO {  
   
 **private static final** String ***create*** = **""** +  
 **"insert into\n"** +  
 **" item(id\_download, link, destination, status, name)\n"** +  
 **"values\n"** +  
 **" (?, ?, ?, ?, ?)"**;  
   
 **public** Item create(Item item) {  
 Integer id = (Integer)insert(  
 con -> {  
 PreparedStatement statement = con.prepareStatement(***create***, Statement.***RETURN\_GENERATED\_KEYS***);  
 statement.setInt(1, item.getDownloadId());  
 statement.setString(2, item.getLink());  
 statement.setString(3, item.getDestination());  
 statement.setString(4, item.getStatus().name());  
 statement.setString(5, item.getName());  
  
 **return** statement;  
 }  
 );  
 item.setId(id);  
   
 **return** item;  
 }  
  
 **private static final** String ***info*** = **""** +  
 **"select\n"** +  
 **" id\_download,** *count***(id) as items\_count,** *sum***(totalBytes) as totalBytes,** *sum***(loadedBytes) as loadedBytes\n"** +  
 **"from\n"** +  
 **" item\n"** +  
 **"group by \n"** +  
 **" id\_download"**;  
  
 **public** Map<Integer, ItemsInfo> itemsInfo() {  
 Map<Integer, ItemsInfo> result = **new** HashMap<>();  
 **jdbcTemplate**.query(***info***, rs -> {  
 ItemsInfo info = ItemsInfo.*builder*()  
 .downloadId(rs.getInt(**"id\_download"**))  
 .count(rs.getLong(**"items\_count"**))  
 .totalBytes(rs.getLong(**"totalBytes"**))  
 .loadedBytes(rs.getLong(**"loadedBytes"**))  
 .build();  
 result.put(info.getDownloadId(), info);  
 });  
  
 **return** result;  
 }  
  
 **private static final** String ***DELETE\_BY\_DOWNLOAD*** = **""** +  
 **"delete from\n"** +  
 **" item\n"** +  
 **"where\n"** +  
 **" id\_download = ?"**;  
  
 **public void** deleteByDownload(Integer downloadId) {  
 **jdbcTemplate**.update(***DELETE\_BY\_DOWNLOAD***, downloadId);  
 }  
  
}

**package** by.bsuir.ksis.dmanager.ui;  
  
**import** by.bsuir.ksis.dmanager.domain.Download;  
**import** by.bsuir.ksis.dmanager.logic.DownloadsService;  
**import** by.bsuir.ksis.dmanager.logic.DownloadsViewModel;  
**import** by.bsuir.ksis.dmanager.logic.NewDownload;  
**import** by.bsuir.ksis.dmanager.logic.Result;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.util.Comparator;  
**import** java.util.stream.Collectors;  
  
**import static** by.bsuir.ksis.dmanager.ui.Util.*openWindow*;  
**import static** by.bsuir.ksis.dmanager.ui.Util.*showError*;  
  
**public class** MainWindow **extends** JFrame {  
  
 **private final** DownloadsService **service**;  
 **private final** DownloadsViewModel **viewModel**;  
  
 **private** ControlPanel **controlPanel** = **new** ControlPanel();  
 **private** DownloadsPanel **downloadsPanel** = **new** DownloadsPanel();  
 **private** DownloadDialog **newDownloadDialog**;  
  
 **public** MainWindow(DownloadsService service, DownloadsViewModel viewModel) **throws** HeadlessException {  
 **super**(**"Менеджер загрузок"**);  
  
 **this**.**service** = service;  
 **this**.**viewModel** = viewModel;  
  
 add(**controlPanel**, BorderLayout.***NORTH***);  
 add(**downloadsPanel**, BorderLayout.***CENTER***);  
 defineControlActions();  
 pack();  
 setDefaultCloseOperation(WindowConstants.***EXIT\_ON\_CLOSE***);  
 setLocationRelativeTo(**null**);  
 viewModel.emitDownloadsListChange();  
 }  
  
 **private void** defineControlActions() {  
 **controlPanel**.addOnAddButtonClick(e -> {  
 **this**.**newDownloadDialog** = **new** DownloadDialog(**this**, **this**::createNewDownload);  
 *openWindow*(**this**.**newDownloadDialog**);  
 });  
  
 **controlPanel**.addOnStartButtonClick(e -> startDownloads(**downloadsPanel**.getSelectedDownloads()));  
  
 **controlPanel**.addOnStopButtonClick(e -> stopDownloads(**downloadsPanel**.getSelectedDownloads()));  
  
 **controlPanel**.addOnDeleteButtonClick(e -> deleteDownloads(**downloadsPanel**.getSelectedDownloads()));  
  
 **viewModel**.subscribeOnDownloadsListChange(downloads -> {  
 downloads = downloads.stream()  
 .sorted(  
 DownloadByPriorityComparator.***INSTANCE*** .thenComparing(Comparator.*comparing*(Download::getCreated).reversed())  
 )  
 .collect(Collectors.*toList*());  
 **downloadsPanel**.showDownloads(downloads);  
 });  
 }  
  
 **private void** createNewDownload(**final** NewDownload download) {  
 SwingUtilities.*invokeLater*(() -> {  
 **try** {  
 Result result = **service**.create(download);  
 **if** (result.isSuccess()) {  
 **this**.**newDownloadDialog**.dispose();  
 } **else** {  
 *showError*(result.getMessage());  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 });  
 }  
  
 **private void** startDownloads(java.util.List<Download> downloads) {  
 SwingUtilities.*invokeLater*(() -> {  
 **try** {  
 Result result = **service**.startDownloads(downloads);  
 **if** (!result.isSuccess()) {  
 *showError*(result.getMessage());  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 });  
 }  
  
 **private void** stopDownloads(java.util.List<Download> downloads) {  
 SwingUtilities.*invokeLater*(() -> {  
 **try** {  
 Result result = **service**.stopDownloads(downloads);  
 **if** (!result.isSuccess()) {  
 *showError*(result.getMessage());  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 });  
 }  
  
 **private void** deleteDownloads(java.util.List<Download> downloads) {  
 SwingUtilities.*invokeLater*(() -> {  
 **try** {  
 Result result = **service**.deleteDownloads(downloads);  
 **if** (!result.isSuccess()) {  
 *showError*(result.getMessage());  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 });  
 }  
  
}