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.

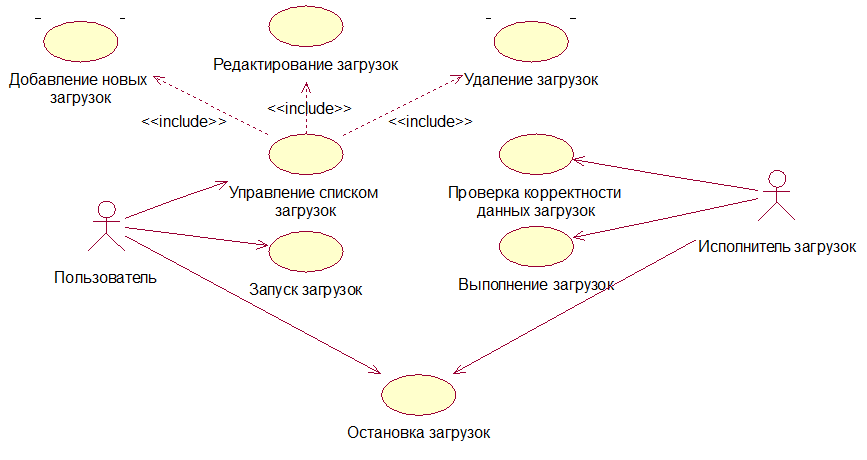


Рисунок 2.1 – Диаграмма вариантов использования

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:

1. управление списком загрузок, что подразумевает добавление новых загрузок в список, удаление загрузок из него, а также редактирование данных загрузок;
2. запуск загрузок: пользователь принимает решение о том, когда необходимо отправить загрузку на исполнение в Исполнитель загрузок;
3. остановка загрузок: пользователь может прекратить исполнение загрузки для того, чтобы освободить вычислительные ресурсы или отложить на время ее исполнение.

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 влить на ход его работы через управление списком загрузок. Исполнитель загрузок выполняет следующие функции:

1. проверка корректности данных загрузок: исполнитель загрузок проверяет возможность осуществления загрузки, и сообщает пользователю о об ошибках в данных загрузки, если они есть;
2. исполнение загрузок: после проверки корректности данных, исполнитель выполняет загрузку и информирует пользователя о ее завершении
3. остановка загрузок: в случае обнаружения ошибки или завершения, исполнитель исключает загрузку из очереди на исполнение.

2.2 Спецификация функциональных требований

На основании анализа исходных данных для проектируемого про-граммного средства можем выделить, что основной целью является созда-ние качественного программного продукта, позволяющего решить суще-ствующие проблемы, такие как:

* эффективное управление расписанием загрузок;
* эффективное использование вычислительных ресурсов при выполнении загрузок.

В ходе разработки будут реализованы следующие возможности:

1. редактирование списка планируемых загрузок;
2. редактирование данных загрузок;
3. проверка возможности осуществления загрузки;
4. формирование очереди исполнения загрузок;
5. исполнении загрузок;
6. прохождение аутентификации и авторизации, если это необходимо для осуществления загрузки;
7. сохранения истории совершенных загрузок;
8. сбор данных о количестве принятых данных.

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:

1. база данных: база данных необходима для долговременного хранения данных загрузок, включая данные о ходе их выполнения исполнителем загрузок;
2. графический интерфейс пользователя, который необходим для эффективного взаимодействия пользователя с программным средством и позволит уме в полной мере использовать все функции программного средства
3. сетевой модуль, который будет осуществлять взаимодействие с удаленными узлами сети;
4. 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 выполнения, выполняющейся или уже выполненной. Имеет следующие поля

* Идентификатор загрузки – идентификатор загрузки(первичный ключ);
* Название – название данные загрузке пользователем, для того, чтобы пользователю было легче найти загрузку в списке;
* Папка сохранения – путь к 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;
* Логин – логин для авторизации на удаленном сервере, если это потребуется при выполнении загрузки;
* Пароль – пароль для авторизации на удаленном сервере, если это потребуется для выполнения загрузки;
* Ошибка – 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 – идентификатор файла(первичный ключ);
* Идентификатор загрузки – внешний ключ на сущность «Загрузка»;
* Ссылка – ссылка по которой можно загрузить файл;
* Статус – текущий статус файла;
* Размер – размер файла на удаленном сервере;
* Количество загруженных байт – количество информации, загруженной из файла на удаленно компьютере на данный момент;
* Ошибка – описание ошибки выявленной при попытке загрузить файл.

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 что папка назначения существует, что программа может записывать файлы в нее, и проверки каждого файла перед скачиванием.

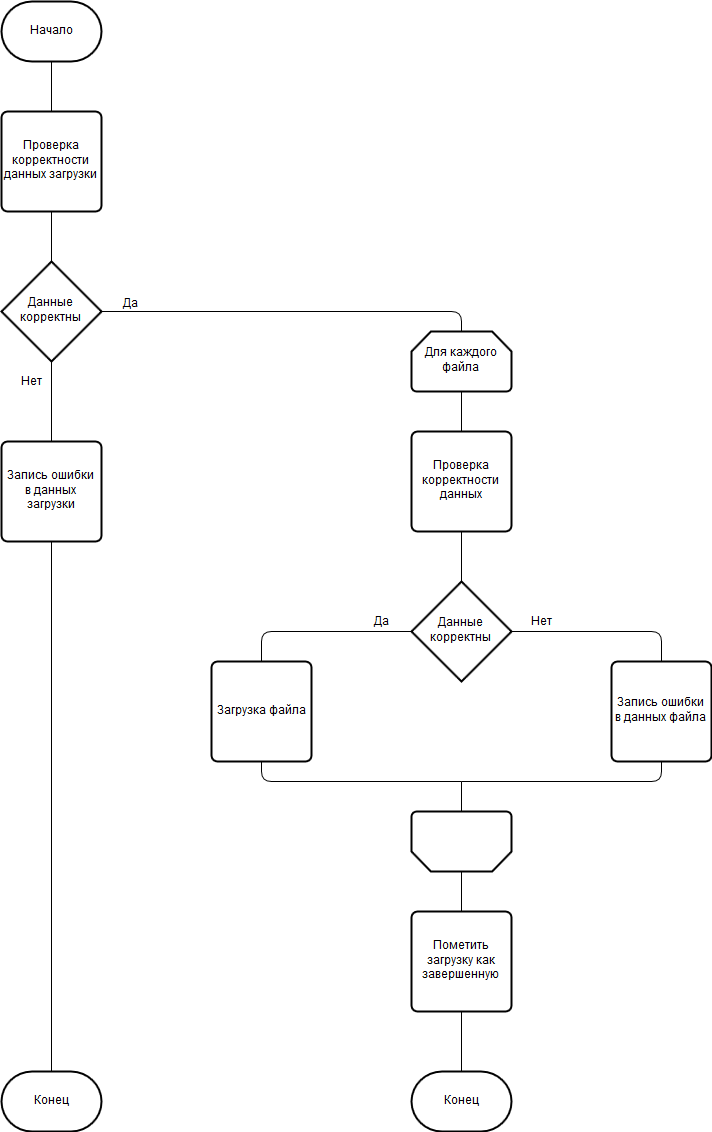


Рисунок 3.1 – Алгоритм выполнения загрузки.

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

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 всех из них он предоставляет следующие возможности:

* Управление ресурсами — автоматическое получение и освобождение ресурсов базы данных;
* Обработка исключений — перевод исключений при доступе к данным в исключения Spring-а;
* Транзакционность — прозрачные транзакции в операциях с данными;
* Распаковка ресурсов — получение объектов базы данных из пула соединений;
* Абстракция для обработки BLOB и CLOB.

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 – удалят указанные загрузки из списка на исполнении;
* deleteDownloads – удаляет указанные загрузки из сипска;
* updateDownload – сохраняет загрузку с измененными данными.

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: stopDwonloadExecution – 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:

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