

Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

СЕТЕВОЕ ПРОГРАММНОЕ СРЕДСТВО
«ЭЛЕКТРОННАЯ БИБЛИОТЕКА»

БГУИР КР I-40 01 01 628 ПЗ

Студент: гр.351006 Шульга Е.С.

Руководитель: Трус В.В.

Минск 2015

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1 ПОСТАНОВКА ЗАДАЧИ	6
2 ОБЗОР АНАЛОГОВ	7
2.1 MyHomeLib	7
2.2 MyRuLib	8
3 ОПИСАНИЕ ФОРМАТОВ ДАННЫХ	9
3.1 Хранение файлов книг	9
3.2 Метаданные коллекций книг	11
3.3 Список жанров	14
4 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА	15
4.1 Проектирование архитектуры	15
4.2 Проектирование серверного программного средства	15
4.3 Проектирование клиентского программного средства	16
5 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА	18
5.1 Разработка структур данных	18
5.2 Разработка серверного программного средства	27
5.3 Разработка клиентского программного средства	29
6 РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ	32
6.1 Администрирование сервера	32
6.2 Руководство по использованию клиента	37
ЗАКЛЮЧЕНИЕ	43
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	44
ПРИЛОЖЕНИЕ А (ОБЯЗАТЕЛЬНОЕ) ИСХОДНЫЕ КОДЫ МЕТОДОВ	45

ВВЕДЕНИЕ

Долгое время единственным источником распространения и сохранения информации являлась обычная печатная книга. Однако в связи с бурным развитием технологий, в настоящее время возникло такое явление как информационный взрыв¹⁾. Нарастающее число публикаций поставило перед библиотеками вопрос хранения столь большого массива информации. Появление электронной публикации позволяет не печатать информацию в виде бумажных книг, а хранить её на гораздо более ёмких электронных носителях в электронных библиотеках [1].

В свою очередь развитие Интернета способствует широкому распространению информации. Электронные библиотеки имеют возможность предоставления доступа к книгам дистанционно.

Также электронные книги решают и другие задачи, такие как упрощение получения книги читателем, уменьшение стоимости издания и распространения книги и т.д. [2]

Согласно некоторым источникам, первая в истории электронная книга была набрана на компьютере в 1971 году Майклом Хартом. В дальнейшем коллекция электронных материалов продолжала пополняться. Была создана первая общедоступная электронная библиотека, в настоящее время известная как проект «Гутенберг» [3].

Одной из крупнейших электронных библиотек русскоязычного сегмента Интернета является Либрусек [4]. Принцип его работы основан на википодобной архитектуре, то есть наполнением сайта, исправлением ошибок занимаются пользователи. Это обеспечивает высокую степень актуальности и обновляемости библиотеки.

В настоящее время пользователи библиотеки начали заниматься независимыми проектами, связанными с Либрусек. Например, появились обновляемые копии базы книг, программы для работы с ними вне сети Интернет [5]. Однако, такие программы для своей работы требуют наличия у пользователя всей базы книг, что приводит к нерациональному использованию дискового пространства.

В данной курсовой работе предлагается решение данной проблемы путем размещения базы книг на одном компьютере в сети и организации доступа к этой базе с других компьютеров; предлагается реализация сетевой электронной библиотеки, которая может быть использована в локальных сетях как зеркало сайта Либрусек.

¹⁾ Информационный взрыв – постоянное увеличение скорости и объёмов публикаций (объёма информации) в масштабах планеты [13].

1 ПОСТАНОВКА ЗАДАЧИ

Входными данными для работы являются упакованные файлы книг, список метаданных книг в специальном формате и список жанров книг. Необходимо изучить форматы хранения этих данных и организовать простой и понятный сетевой интерфейс доступа к ним.

В рамках курсового проекта необходимо спроектировать и реализовать программное средство, включающее в себя клиентскую и серверную часть

Серверное программное средство должно быть реализовано как веб-сервис, осуществляющий импорт входных данных, организующий их хранение и предоставляющий программный интерфейс для удаленного доступа к этим данным без права их модификации: поиск записей книг по шаблону, получение списка найденных книг, получение файла книги. Также необходимо обеспечить возможность получения и вывода информации о запросах.

Реализовать графический интерфейс клиентского приложения, предоставляющее возможность формирования шаблона для поиска, отображения полученного списка книг, возможность выбора книги из списка и ее скачивания. Также необходимо реализовать функцию автообнаружения сервера, если клиент и сервер находятся в одной сети.

Обе части приложения должны иметь возможность простого развертывания на компьютерах пользователей. Необходимо реализовать сохранение вводимых пользователем параметров приложений с возможностью использования их при последующих запусках. Должна быть реализована возможность сохранения книг в различных форматах.

Для закрепления знаний, полученных на учебных занятиях в рамках курсов КСиС, ООТПиСП, ВебТех и ТРПО, языком программирования был выбран язык C#, для проектирования клиент-серверной архитектуры была выбрана технология WCF, для разработки графического интерфейса была выбрана технология WPF.

2 ОБЗОР АНАЛОГОВ

В связи с тем, что большинство людей предпочитает не иметь большие коллекции книг на диске, программ для организации домашних электронных библиотек существует не так много. Рассмотрим наиболее распространенные.

2.1 MyHomeLib

2.1.1 Данное программное средство является наиболее используемым при организации домашней библиотеки в странах русскоязычного сегмента интернета. Данное приложение может использоваться и для каталогизации собственной коллекции книг пользователя, и как клиент для работы с копией библиотеки Либрусек (и других библиотек, использующих такие же форматы служебных файлов).

2.1.2 Главная функциональная возможность данного приложения – поиск по автору, названию, жанру и другим параметрам книг в коллекции. Пример окна поиска приведен на рисунке 2.1.

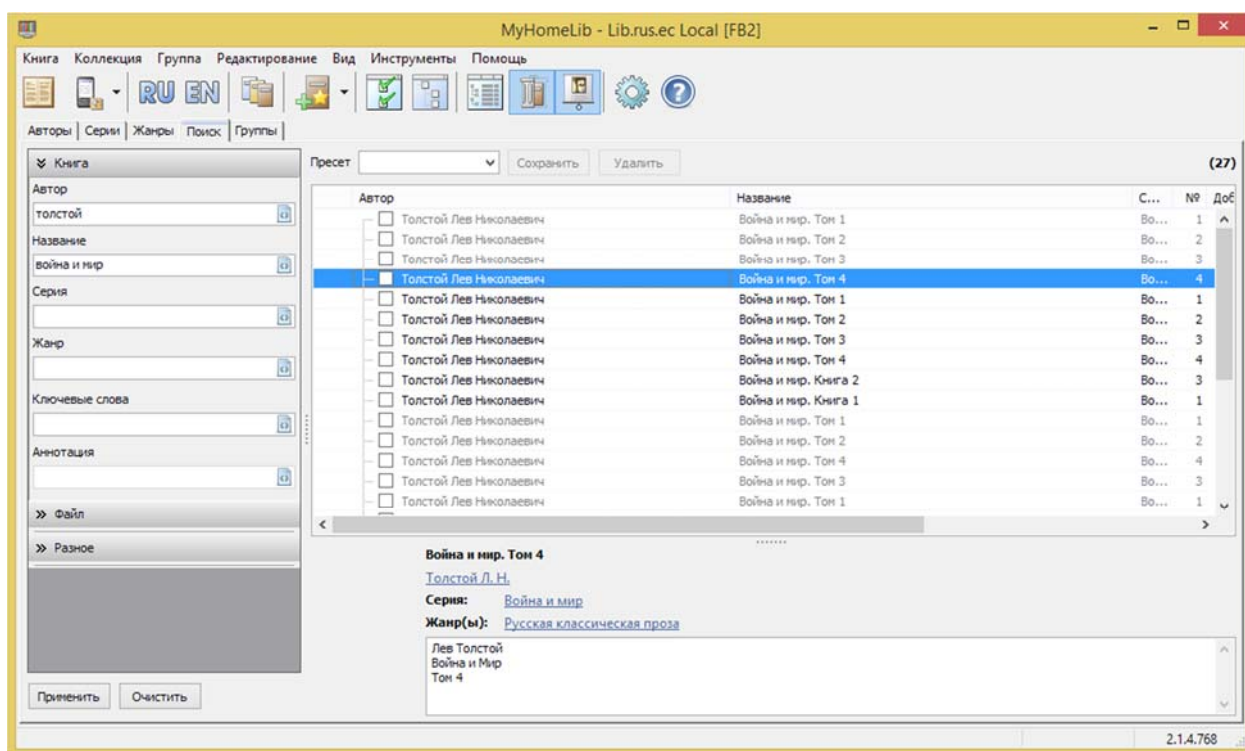


Рисунок 2.1 – Поиск книг в программном средстве MyHomeLib

2.1.3 После нахождения требуемой книги программное средство осуществляет его экспорт в заранее определенную папку на диске. Возможность чтения книг появляется после экспорта с помощью сторонних приложений.

2.1.4 Основными недостатками данного программного средства являются отсутствие возможности работы по сети и медленное выполнение следующих операций: запуск программы, поиск книг, отображение списков книг.

2.2 MyRuLib

2.2.1 Данное программное средство является кроссплатформенным аналогом MyHomeLib. Так же может использоваться для организации и управления домашней библиотекой и для доступа к копии библиотеки Либрусек.

2.2.2 MyRuLib предоставляет возможность поиска по следующим критериям: автор, название. Пример работы программы (поиска книг) представлен на рисунке 2.2.

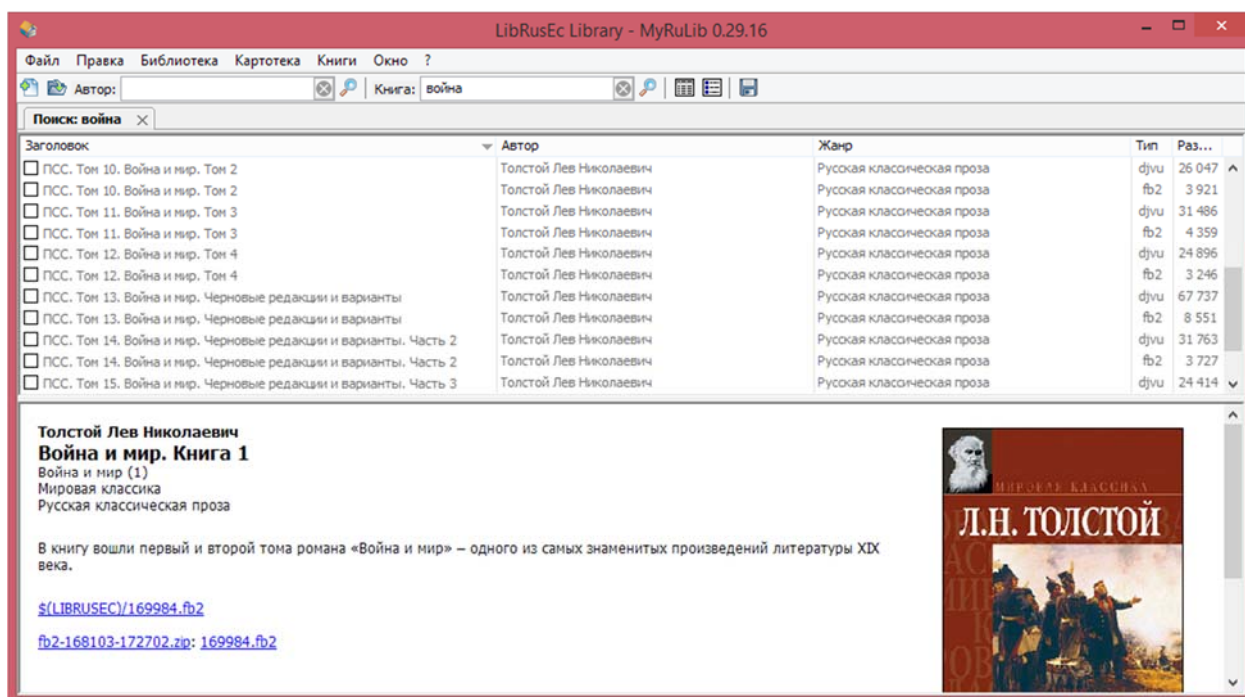


Рисунок 2.2 – Поиск книг в программном средстве MyRuLib

2.2.3 Данному программному средству присущи следующие недостатки:

- Долгое время импорта книг. Это связано с тем, что MyRuLib не использует файлы метаданных, предоставляемые сетевыми электронными библиотеками.

- Невозможность поиска одновременно и по автору, и по названию (отсутствие возможности задания шаблона для поиска).

- Отсутствие возможности выбора папки для экспорта и неудобно реализованная возможность сохранения книг на диск. Вместо этого при выборе книг предлагается сразу запустить приложение для чтения (такие приложения могут быть не установлены у пользователя).

3 ОПИСАНИЕ ФОРМАТОВ ДАННЫХ

В настоящее время люди, занимающиеся распространением электронных книг, организовали унифицированную форму распространения книг из нескольких электронных библиотек. Кроме Либрусека, в таком же формате распространяются копии веб-сайта Флибуста [6] и некоторые другие. Кроме того, и для Либрусека, и для Флибусты существует несколько вариантов распространения: *fb2* копии, содержащие только книги формата .fb2, и *usr* копии, которые могут содержать книги различных форматов: doc, rtf, txt, pdf, epub и т.д.

Перед началом проектирования и разработки необходимо описать структуру входных данных.

3.1 Хранение файлов книг

3.1.1 В связи с большим размером коллекций целесообразно оптимизировать способы их хранения. Принято использовать упаковку книг в zip архивы (пример папки хранилища книг можно увидеть на рисунке 2.1).

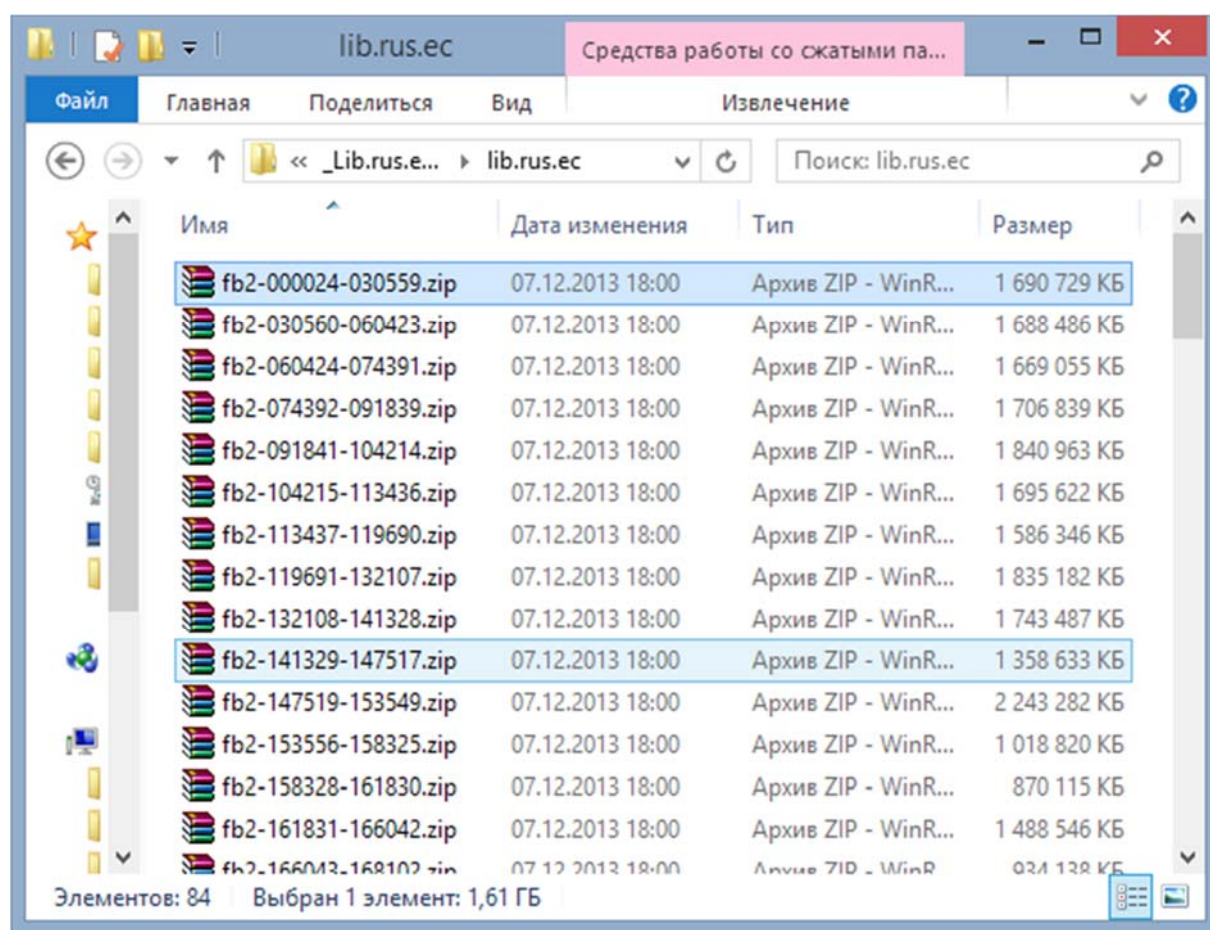


Рисунок 2.1 – Пример папки-хранилища книг

3.1.2 Можно увидеть следующее правило формирования имен архивов:

<Тип-архива>-<ID-первой-книги>-<ID-последней-книги>.zip

Причем <Тип-архива> может быть *fb2* или *usr* (для коллекций разных форматов). Две другие части имени используются для упрощения управлением коллекцией и для уникальной идентификации каждого архива. Кроме того, как будет сказано ниже, имена архивов и имена файлов метаданных, их описывающих, должны совпадать.

3.1.3 На рисунке 2.2 можно увидеть содержание одного из архивов.

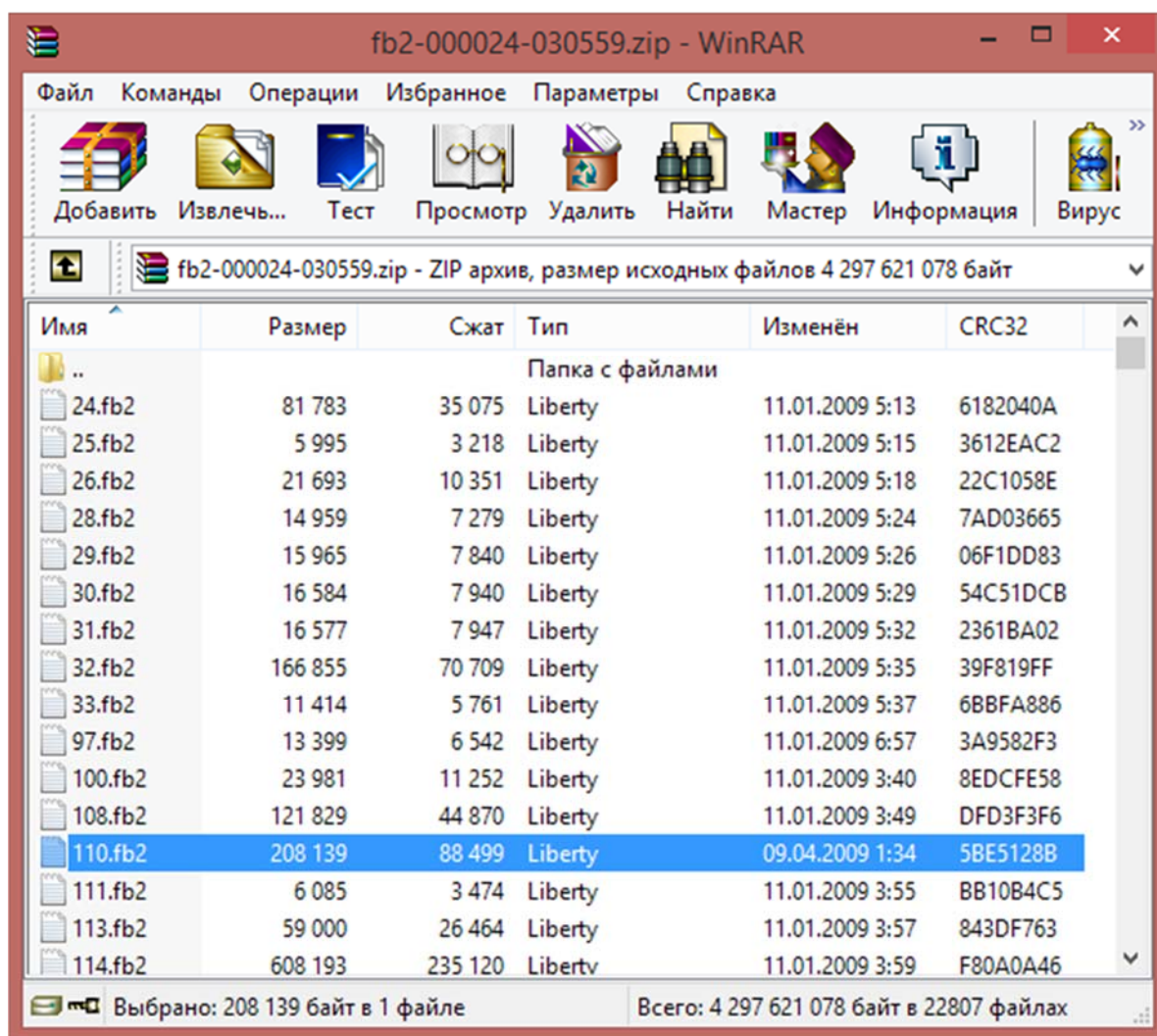


Рисунок 2.2 – Пример содержания архива из хранилища книг

Файлы, представленные на рисунке 2, непосредственно являются книгами. Можно просто распаковывать архивы и выбирать необходимые книги. Однако это неудобно, так как имена файлов книг ничего не говорят об авторе, или названии, или содержании книги – вместо этого используется числовой ключ (при создании пользовательских коллекций имена файлов могут быть произвольными, однако это повлияет на файлы метаданных).

Для коллекций библиотек Либрусек и Флибусты данные ключи используются для уникальной идентификации каждой книги (утверждается что они будут уникальными).

Для организации доступа к книгам из этих архивов и разрабатывается данная курсовая работа.

3.2 Метаданные коллекций книг

3.2.1 Для описания коллекций книг был разработан специальный формат файлов. В настоящее время его используют для создания коллекций копий веб-сайтов электронных библиотек. Также его целесообразно использовать при каталогизации обширных собственных библиотек. Файлы этого формата имеют расширение `inpx`.

По своей структуре файлы данного формата представляют собой `zip`-архив, содержащий файлы с расширением `inpr`. В качестве примера на рисунке 2.3 приведена структура `inpx` файла библиотеки Либрусек.

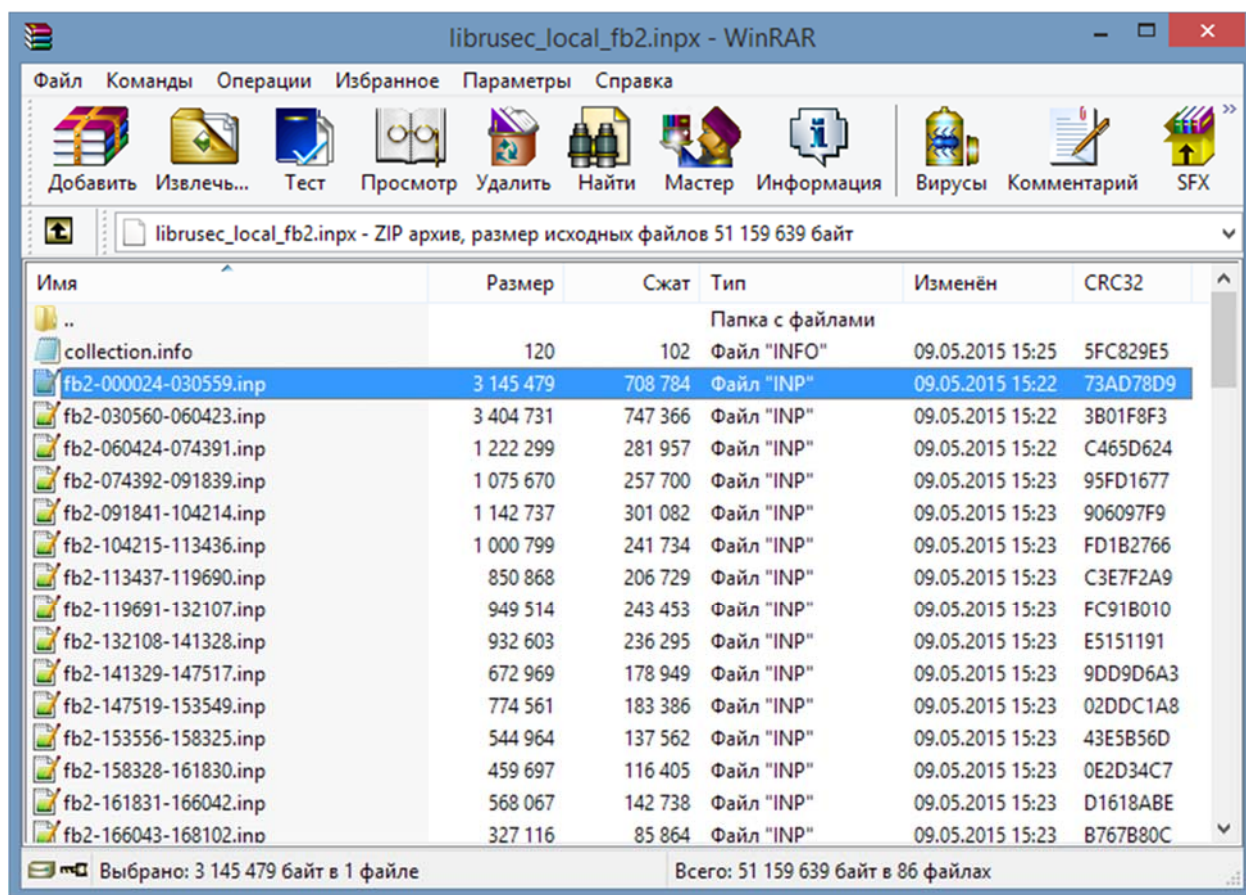


Рисунок 2.3 – Пример содержания `inpx` файла

В качестве первого файла можно увидеть файл `collection.info`. Он является необязательным и содержит комментарий, относящийся к данной коллекции.

Как было сказано в пункте 2.1, имена .inp файлов должны совпадать с именами архивов, к которым они относятся.

3.2.2 Данные файлы являются текстовыми в формате UTF-8. Они построчно содержат информацию обо всех книгах, содержащихся в коллекции. Пример такого файла приведен на рисунке 2.4.



Рисунок 2.4 – Пример inp файла

Кроме того, утверждается, что количество строк такого файла и число книг в соответствующем архиве совпадает, то есть каждая книга имеет свою запись в файле метаданных (безошибочность достигается тем, что создание inpх файла производится автоматически, а не вручную).

3.2.3 Каждая строка содержит определенным образом расположенные метаданные. Некоторые из этих полей необязательны, то есть они могут содержать пустую строку (тем не менее они должны присутствовать), некоторые поля могут содержать несколько значений. Поля разделяются символом с кодом 04 (EOT – End of Transmission). Названия и описания полей приведены в таблице 2.1.

3.2.4 В формате .inp файла реализована возможность наличия у книги нескольких (не менее одного) авторов. Признаком конца имени автора является символ с кодом 58 (':'), разделителем фамилии, имени и отчества служит символ с кодом 44 (','):

<Фамилия1>,<Имя1>,<Отчество1>:<Фамилия2>,<Имя2>,<Отчество2>:

Поля фамилии, имени и отчества не являются обязательными и могут быть пропущены. Наличие разделителей полей обязательно.

Таблица 2.1 – Поля записи книги в файле метаданных

Номер	Условное обозначение	Название	Примечание
0	AUTHOR	Полное имя автора	См. 2.2.4
1	GENRE	Жанр	См. 2.2.5
2	TITLE	Название	
3	SERIES	Название серии	Может быть не заполнено
4	SERNO	Номер в серии	Может быть не заполнено
5	FILE	Имя файла в архиве	Для Либрусека совпадает с полем LIBID
6	SIZE	Размер файла	Размер файла книги в байтах
7	LIBID	ID книги	Совпадает с FILE; уникально идентифицирует каждую книгу
8	DEL	Индикатор удаления файла	См. 2.2.6
9	EXT	Расширение файла	Для fb2 коллекций – всегда fb2
10	DATE	Дата добавления	Дата добавления в базу Либрусека; может быть не заполнено
11	LANG	Язык книги	Может быть не заполнено
12	LIBRATE	Внешний рейтинг	Рейтинг книги на сайте Либрусек; обычно не актуально и не заполнено
13	KEYWORDS	Ключевые слова	Как правило не заполнены

3.2.5 Поле жанров, как и поле авторов, может содержать несколько значений; признаком окончания жанра также служит символ с кодом 58 (':'). Однако, это поле принимает одно из predetermined значений, причем эти значения инициализируются из специального файла жанров. Список жанров формируется библиотекой Либрусек. Подробнее про инициализацию списка жанров см. подраздел 3.3.

3.2.6 Индикатор удаления файла является служебной информацией. Для большинства книг он установлен в 0 (то есть книга не удалена). Если же индикатор установлен в 1, то это может означать, что с книгой что-то произошло: файл был добавлен, а затем удален модераторами, или имеется другая версия книги с существенно лучшим качеством. Тем не менее, если книга уже попала в базу книг, то она не удаляется, а помечается этим флагом.

3.3 Список жанров

3.3.1 В большие сетевые библиотеки пользователи постоянно добавляют новые книги, и необходимо обеспечить их категоризацию по жанрам. Может так получиться, что новая книга не принадлежит ни одному из заранее определенных жанров, может понадобится расширение списка.

Список жанров предоставляется электронной библиотекой, поэтому при создании копии какой-либо библиотеки рекомендуется использовать актуальный список жанров именно этой библиотеки.

Рассмотрим формат файла списка жанров библиотеки Либрусек.

3.3.2 Файлы списка жанров имеют расширение .glst. Файлы этого формата являются текстовыми; в них построчно размещена информация о каждом жанре. Формат каждой строки:

<Номер-группы-жанра>.<Номер-жанра>.<Номер-поджанра> <Имя-жанра>;<Описание-жанра>

3.3.3 Существует возможность упорядочивания по трёхуровневой иерархической структуре: группа жанра, жанр, поджанр. Фактически при этом признак, который устанавливается каждой книге, – это поджанр. Тогда группировка по жанрам и группам жанров является условной, облегчающую присвоение записям книг данного признака.

Номер группы жанра, номер жанра, номер поджанра разделены между собой символом с кодом 46 ('. '); после номера поджанра следует символ с кодом 32 (' ').

3.3.4 Поле имени жанра содержит сокращенное название жанра. Обычно оно записывается с использованием букв латинского алфавита и знаков подчеркивания, например: detective, det_espionage, vers_libre и т.д. Данное поле служит для упрощения идентификации каждого жанра.

3.3.5 За полем имени жанра и до конца строки следует поля описания жанра. В этом поле принято размещать название поджанра в удобном для чтения пользователем виде. Описание поджанров на сайте Либрусек принято заполнять на русском языке, например: Советская классическая проза, Киносценарии, Европейская старинная литература и т.д.

3.3.6 Формат файлов предусматривает также размещение комментариев. Комментарием является текст, ограниченный слева символом с кодом 35 ('#') и справа концом строки. Комментарии должны располагаться на отдельных строках. Комментарии могут быть полезны при группировке поджанров по жанрам и группам. Пример комментария:

#----- 0.2 Детективы и Триллеры -----

4 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

4.1 Проектирование архитектуры

4.1.1 Программное средство будет состоять из двух частей: серверная и клиентская части. Они должны быть независимы, но должен существовать способ их взаимодействия.

4.4.2 Независимость приложений заключается в следующем:

- независимость развертывания, то есть клиент и сервер могут быть развернуты не обязательно на одном компьютере, причем должна обеспечиваться возможность их взаимодействия;

- независимость настройки: и клиент, и сервер обладают присущим только им списком параметров, причем должна существовать возможность изменения этого списка параметров независимо от другой части программного средства;

- независимость старта: предполагается, что развернут один сервер, и с ним взаимодействует большое число клиентов; должна обеспечиваться параллельная обработка запросов от большого числа клиентов.

4.4.3 Необходимо разработать программное средство с учетом разделения функций, выполняемых клиентом и сервером.

Основная функция, выполняемая сервером, – это импорт входных данных. Еще одна немаловажная функция сервера – извлечение и передача клиенту по запросу файла книги.

Клиент же должен предоставлять возможность обнаружения сервера, установления с ним соединения, формирования запроса для поиска, отображение результатов запроса, получение и сохранение файла книги. Преобразование файла книги в один из выбранных форматов должно происходить на клиенте.

4.4.4 Еще один параметр, по которому проводится разделение – это данные. Практически все входные данные доступны только серверному приложению. Клиент же получает данные по мере необходимости или по запросу пользователя.

4.2 Проектирование серверного программного средства

4.2.1 Для начала, в соответствии с форматами, рассмотренными в разделе 3, необходимо обеспечить импорт входных данных. Импорт целесообразно производить в следующей последовательности: инициализация списка жанров, инициализация записей книг с помощью файла метаданных, инициализация хранилища, откуда будет производиться извлечение книг. Важным является то, что инициализация жанров должна обязательно производиться до инициализации метаданными, так как в запись книги обязательно должна входить информация о ее жанре.

4.2.2 Инициализация жанров должна происходить путем построчного считывания файла жанров с игнорированием строк-комментариев. Подходящие строки должны разделяться на поля в соответствии с форматом.

4.2.3 Затем должна производиться инициализация метаданных: для начала из `inpx` файла, представляющий собой по сути архив, считываются поочередно записи файлов. Затем для каждого из этих файлов производится построчное чтение информации о записях книг. После этого необходимо обеспечить разделение данной информации в соответствии с форматом и заполнение полей записей книг. Необходимо предусмотреть возможность наличия у одной книги нескольких авторов и жанров, а также наличие пустых полей. Также необходимо обеспечить сохранение информации о том, из какой записи архива производится считывание данных, так как эта информация соответствует тому архиву из хранилища, в котором располагается книга.

4.2.4 После завершения инициализации необходимо обеспечить возможность взаимодействия клиентов с сервером. Целесообразно реализовать данную возможность с помощью каких-либо технологий RPC (Remote procedure call – удаленный вызов процедур).

4.2.5 Необходимо реализовать интерфейс для ввода пользователем параметров инициализации. Поскольку сервер не предполагает активного прямого взаимодействия с пользователем после его запуска, то может быть реализован текстовый интерфейс консольного приложения. При введенных неверных данных необходимо запросить повторный ввод. Также для удобства пользователя следует реализовать возможность сохранения после закрытия приложения и повторного использования введенных параметров.

4.3 Проектирование клиентского программного средства

4.3.1 Основная функция клиентского приложения – предоставление возможности поиска и получения книг. Графический интерфейс должен соответствовать данной функции. Необходимо предусмотреть просмотр пользователем основной информации о книге перед ее скачиванием. Макет главного окна клиентского приложения представлен на рисунке 4.1.

Макет спроектирован таким образом, чтобы клиенту было удобно с первого раза разобраться в способе использования данного программного средства: предполагается, что взгляд пользователя следует справа налево и сверху вниз.

Для начала пользователь осуществляет заполнение шаблона для поиска, например: заполняет текстовые поля названия книги и имени автора, выбирает требуемый жанр. Данные действия будут производиться в области формирования шаблона для поиска. Затем результат обработки сервером запроса будет отображаться в область вывода списка книг. Ниже области списка присутствует область отображения краткой информации о книге. В этой области появляется информация о текущей выбранной книге. После этого предполагается нажатие кнопки для скачивания книги.

<div> <div></div> <div></div> <div></div> </div> <div> <div>Область формирования шаблона для поиска</div> </div>	<div> <div>Автор книги</div> <div>Название книги</div> <div>Автор книги</div> <div>Название книги</div> <div>Автор книги</div> <div>Название книги</div> </div>	<div> <div>Жанр</div> <div>Жанр</div> <div>Жанр</div> </div>
	<div> <div>Область вывода списка книг</div> </div>	
	<div> <div>Область вывода информации о книге</div> <div>Скачать</div> </div>	

Рисунок 4.1 – Макет главного окна клиентского приложения

4.3.2 Необходимо предусмотреть возможность ввода пользователем сервера для подключения. Однако также должна существовать возможность автообнаружения доступных серверов.

Пользователь должен обладать возможностью выбора папки, в которую будут помещаться скачиваемые с сервера книги.

Для того, чтобы пользователь не вводил каждый раз при запуске одни и те же параметры, необходимо предусмотреть возможность сохранения настроек.

4.3.3 Книги, предоставляемые библиотекой Либрусек, а, следовательно, находящиеся на сервере, представлены в формате fb2. Однако у пользователя может быть электронное устройство или программа для чтения, которые могут не поддерживать данный формат.

Для решения этой проблемы необходимо обеспечить возможность автоматической конвертации скачиваемой книги в нужный формат. Выбор формата так же должен осуществляться пользователем и сохраняться после перезапусков приложения.

5 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

Перед началом реализации возможностей, заявленных на этапе проектирования программного средства, необходимо разработать структуры данных, предназначенные для хранения в оперативной памяти данных, полученных в результате импорта при инициализации серверного приложения. Исходя из принципа ограничения сложности рационально организовать программу в виде некоторого числа классов, каждый из которых будет выполнять определенную задачу.

5.1 Разработка структур данных

5.1.1 Для инициализации и хранения списка жанров был разработан класс GenresList. Схема алгоритма инициализации представлена на рисунке 5.1.

Для хранения каждой записи жанра был реализован класс GenresListEntity. Структура свойств данного класса представлена в таблице 5.1. Описание данных свойств приведено в пункте 3.3.

Таблица 5.1 – Структура класса GenresListEntity

Тип элемента	Название элемента	Назначение элемента
int	genreGroupNumber	Номер группы жанра
int	genreNumber	Номер жанра
int	subgenreNumber	Номер поджанра
string	name	Имя жанра
string	description	Описание жанра

Инициализация экземпляра данного класса производится в блоках 6-8 схемы, представленной на рисунке 5.1.

В качестве хранилища жанров в классе GenresList используется массив типа List<GenresListEntity>.

Для доступа к списку жанров были разработаны специальные методы, которые приведены в таблице 5.2.

Следует отметить, что данный список после его инициализации является единственным и унифицированным хранилищем жанров для программного средства. Поэтому было решено для уникальной идентификации жанров использовать индекс жанра в массиве жанров в классе GenresList. Данное решение значительно сократило место, необходимое для хранения записей книг, а также ускорило обработку данных, в том числе, поиск по жанру.

Сортировка списка жанров, производимая в блоке 11 на схеме на рисунке 5.1, необходима для того, чтобы, после того, как список жанров будет передан клиенту, он отобразился бы в упорядоченном виде, что значительно упростит поиск жанров.

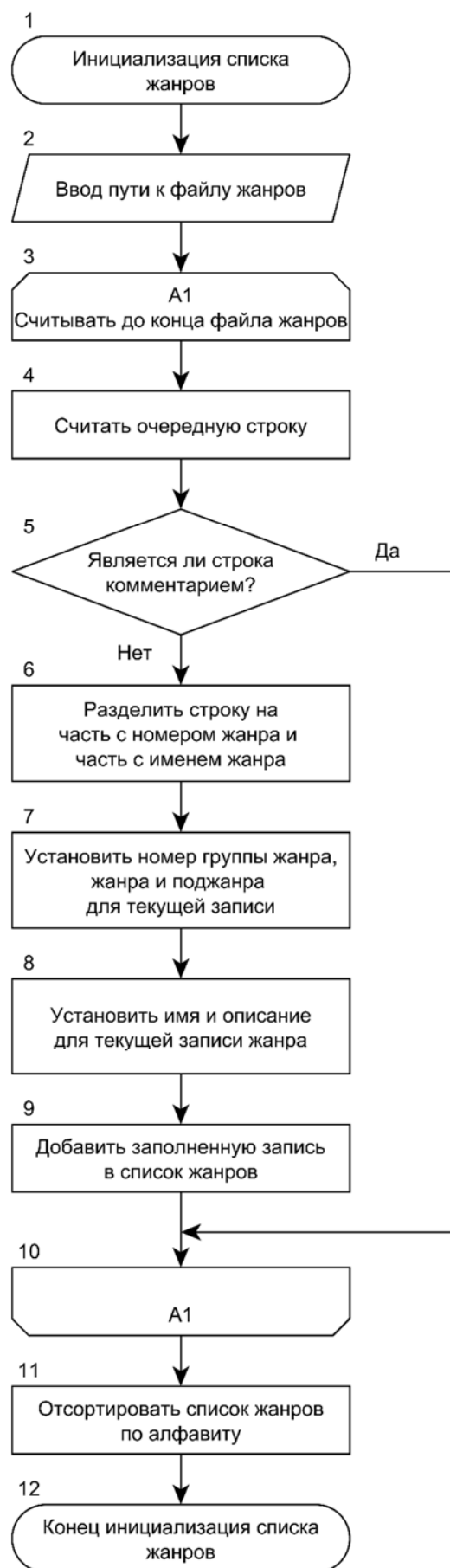


Рисунок 5.1 – Инициализация списка жанров

Таблица 5.2 – Методы класса GenresList для доступа к жанрам

Тип возвращаемого значения	Имя метода	Описание
string	getGenreName	По индексу возвращает имя жанра
string	getGenreDescription	По индексу возвращает описание жанра
int	getGenreID	По имени жанра возвращает его индекс
List<GenresListEntity>	getAvailableGenres	Возвращает список жанров. Используется при инициализации клиентского приложения

5.1.2 Для инкапсуляции методов, предназначенных для извлечения файлов книг, был реализован статический класс BookExtractor. Он включает в себя два доступных извне метода: initialize и extract.

Перед началом работы с этим классом его необходимо проинициализировать. Фактически, инициализация заключается в указании папки на диске, в которой расположены упакованные файлы книг.

После инициализации для получения потока файла необходимой книги был реализован метод extract. Следует отметить, что ему необходимо передать запись книги (экземпляр класса BookEntity, см. пункт 5.1.3), который, при иных входных данных, нужно получить заранее (например, из класса MetadataDB, см. пункт 5.1.7). Он работает следующим образом: получает имя архива, в котором находится файл данной записи книги, открывает данный архив из хранилища, получает запись данного архива, открывает эту запись, возвращает поток, связанный с файлом, находящимся в архиве. Теперь можно передать данный поток куда необходимо и после передачи записать данный поток на диск.

5.1.3 Для хранения записей метаданных книг был реализован класс BookEntity. Он инкапсулирует поля, предоставляемые форматом файла метаданных, методы их инициализации и методы получения информации о записях книг.

На рисунках 5.2 и 5.3 приведен алгоритм инициализации записи книги.

В качестве реализации возможности наличия у одной книги нескольких авторов и жанров были разработаны классы Authors и Genres. Их инициализация производится в блоках 4 и 5. Реализация данных классов рассмотрена в пунктах 5.1.4 и 5.1.5.

Поле archiveName соответствует имени inpr файла из inprx архива. Его инициализация проводится на другом этапе: на этапе импорта метаданных из inpr файла.

Поля, агрегированные в данном классе, соответствуют полям данных inprx файла, пояснение которым дается в пункте 3.2.4 и представлены в таблице 5.3.

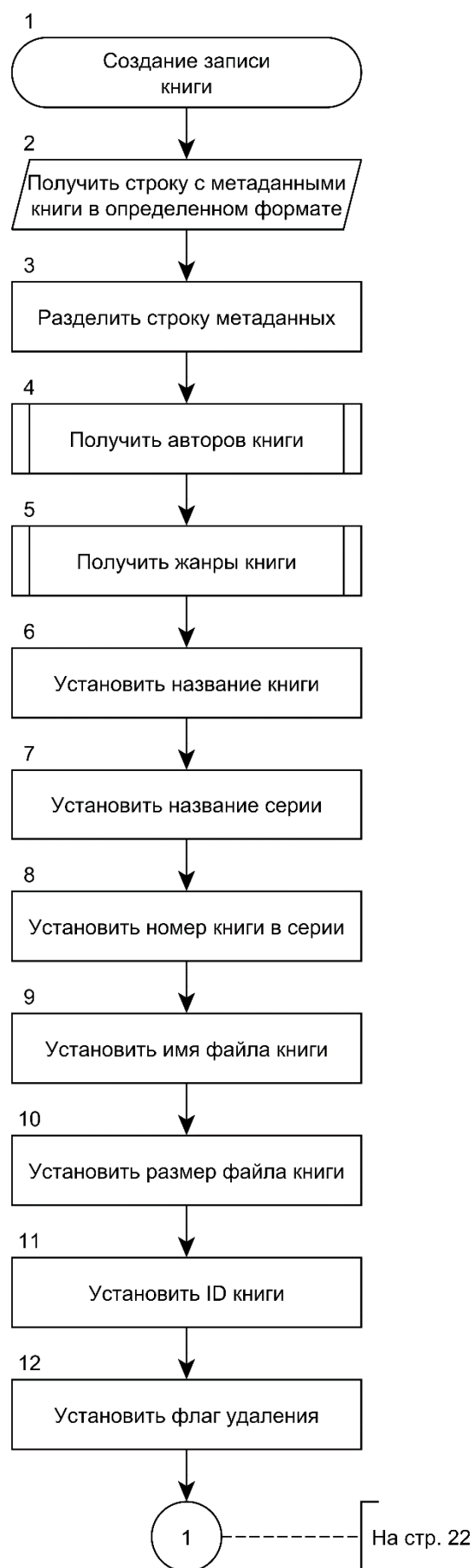


Рисунок 5.2 – Инициализация записи книги



Рисунок 5.3 – Окончание инициализации записи книги

Таблица 5.3 – Свойства, агрегированные в классе BookEntity

Тип данных	Имя свойства	Соответствующее имя поля
Authors	authors	AUTHOR
Genres	genres	GENRE
string	title	TITLE
string	seriesTitle	SERIES
int	numberInSeries	SERNO
string	filename	FILE
int	fileSize	SIZE
int	bookID	LIBID
bool	isDeleted	DEL
string	extension	EXT
string	dateAdded	DATE
string	language	LANG
string	bookRate	LIBRATE
string	keywords	KEYWORDS
string	archiveName	-

5.1.4 Для реализации возможности наличия у одной книги нескольких авторов были реализованы классы Author, агрегирующий поля имени, фамилии, отчества, а также (для упрощения поиска автора по строке, которая может содержать как имя, так и фамилию или отчество или их часть) полного имени, и Authors, агрегирующий список типа List<Author>.

Алгоритм получения списка авторов книг представлен на рисунке 5.4.

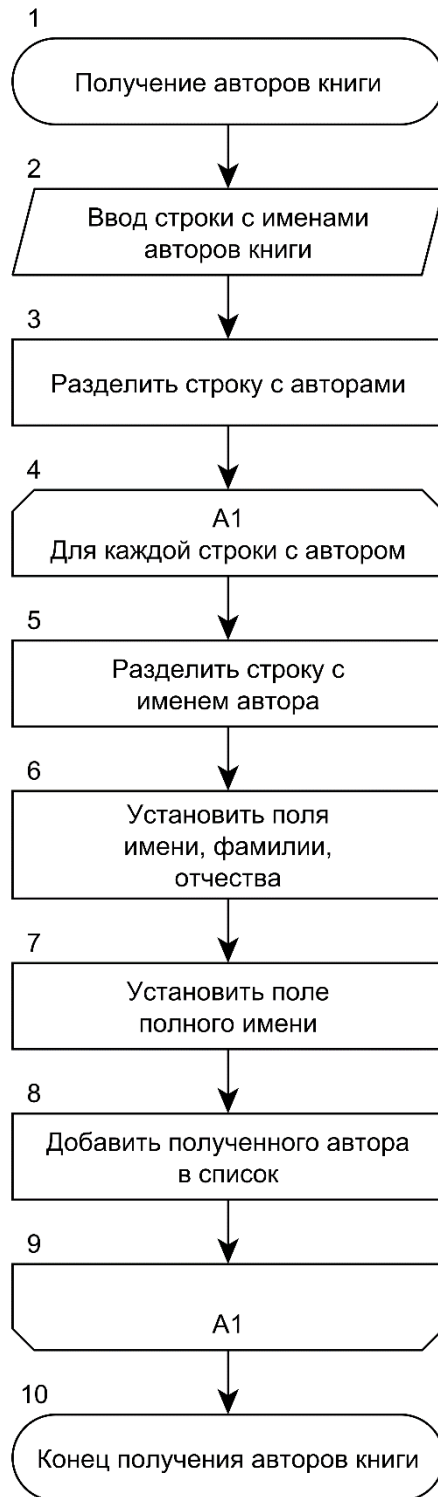


Рисунок 5.4 – Получение авторов книги

5.1.5 Для обеспечения наличия у одной книги нескольких жанров был разработан класс Genres. Он агрегирует список типа List<int>. В данном списке содержатся номера жанров, а для получения имен и описаний жанров следует обратиться к классу GenresList. Как было сказано в пункте 5.1.1, данное решение позволило оптимизировать расход памяти и ускорить время поиска жанров.

Алгоритм получения жанров представлен на рисунке 5.5.

Предполагается, что все жанры, представленные в файле метаданных известны заранее и представлены в особом списке жанров. Тем не менее, нельзя исключать наличие ошибок, полученных при заполнении характеристик книг. Поэтому, при обнаружении жанра, имя которого не содержится в списке жанров, такой жанр игнорируется.

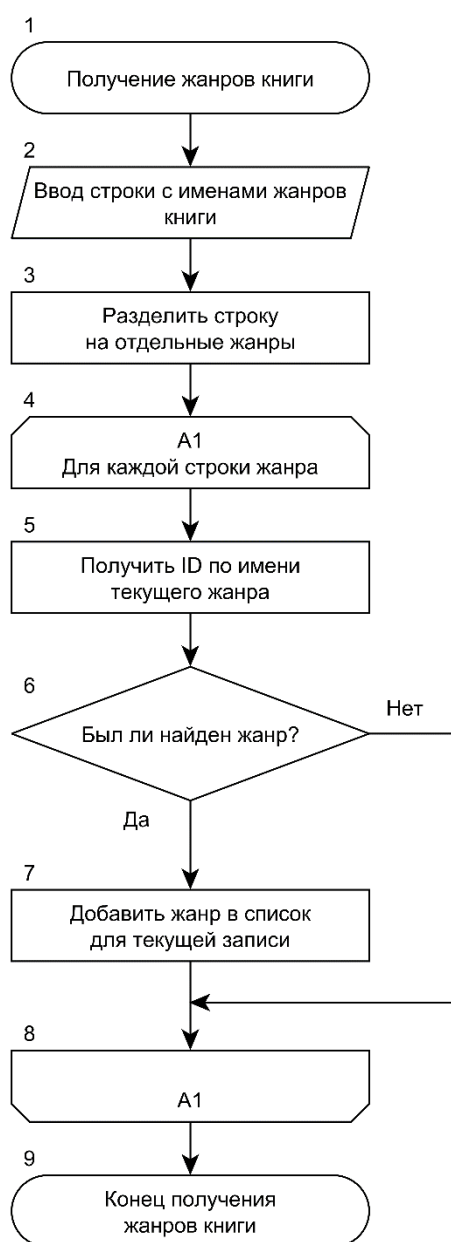


Рисунок 5.5 – Получение жанров книги

5.1.6 Предположим, дан список записей книг типа `List<BookEntity>`, и необходимо извлечь какие-либо данные из этого списка. Для этих целей был разработан статический класс `MetadataQuery`. Методы, агрегированные в данном классе, представлены в таблице 5.4.

Таблица 5.4 – Методы класса `MetadataQuery`

Тип возвращаемого значений	Название	Описание
<code>List<BookEntity></code>	<code>selectBooksByAuthor</code>	Возвращает все книги, принадлежащие всем авторам, в полном имени которых содержится строка для поиска
<code>List<BookEntity></code>	<code>selectBooksByTitle</code>	Возвращает книги, в названии которых содержится строка для поиска
<code>List<BookEntity></code>	<code>selectBooksByGenre</code>	Возвращает все книги, принадлежащие жанру с определенным ID
<code>List<BookEntity></code>	<code>selectBooksByGenres</code>	Возвращает книги, лежащие на пересечении искомых жанров
<code>BookEntity</code>	<code>selectBookByID</code>	Возвращает книгу по ее ID
<code>List<BookEntity></code>	<code>selectBooksByTemplate</code>	Производит отсеивание книг списка в соответствие с полями шаблона

Основной и самый универсальный метод, который будет использоваться при поиске – `selectBooksByTemplate`. Данный метод производит последовательное отсеивание книг по критериям: имя автора, название, жанр.

5.1.7 В качестве декоратора для класса `MetadataQuery` был реализован класс `MetadataList`. Он содержит поле типа `List<BookEntity>`, которое является основным хранилищем метаданных книг в программном средстве. В данном классе определены методы обращения к метаданным, реализованные как более общие в классе `MetadataQuery`.

Перед тем, чтобы использовать методы данного класса, его необходимо проинициализировать. Для этого необходимо вызвать метод `import` класса `InpxImport`, который возвращает список типа `List<BookEntity>`, полученный в результате импорта `inpx` файла. Реализацию данного класса приведена в пункте 5.1.8.

5.1.8 Импорт метаданных инкапсулирован в статическом классе `InpxImport`. Алгоритм импорта представлен на рисунке 5.6.

Данный алгоритм на каждой итерации считывает из файла метаданных строку и создает запись книги. Стоит отметить, что поле имени архива устанавливается в данном алгоритме, а не при создании записи книги.



Рисунок 5.6 – Импорт метаданных

5.2 Разработка серверного программного средства

5.2.1 Сервер не требует активного взаимодействия с пользователем во время своей работы. При запуске пользователь осуществляет ввод необходимы данных, а затем сервис автоматически принимает и обрабатывает запросы от клиентов.

В связи с этим целесообразно реализовать сервер в виде консольного приложения, на котором обеспечивается хостинг (данный вид хостинг называется автохостингом) и работа веб-сервиса.

5.2.2 Для начала необходимо совершить инициализацию сервера. Алгоритм инициализации представлен на графическом материале ГУИР.351006-001 ПД.

Инициализация состоит из трёх частей: инициализация списка жанров (см. пункт 5.1.1), инициализация метаданных (см. пункт 5.1.8), инициализация хранилища книг (см. пункт 5.1.2).

5.2.3 В связи с тем, что в пункте 4.2.4 было запланировано применения технологий RPC, при реализации серверного программного средства было решено применить технологию WCF (Windows Communication Foundation). Данная технология предоставляет все возможности для построения сервис-ориентированных программных средств.

Создание веб-сервиса с использованием WCF базируется на задании трех базовых частей: address (адрес сервиса), binding (привязки сервиса), contract (контракт данных для взаимодействия).

5.2.4 Для реализации сервиса была выбрана привязка NetTcpBinding. Она обеспечивает взаимодействие программных средств, написанных с использованием технологий .Net и WCF по бинарному протоколу передачи данных TCP. Данная привязка наиболее проста в использовании и обеспечивает максимальную скорость обработки (в отличие, например, от привязок, основанных на HTTP протоколе, так как в них передача данных осуществляется в виде текста, а значит необходимы преобразования данных при передаче и приёме).

5.2.5 Один компьютер может обладать несколькими сетевыми картами, и, следовательно, находиться в нескольких сетях и обладать несколькими IP-адресами. Для того, чтобы сервис был доступен из всех подключенных к нему сетей, необходимо в качестве адреса при создании конечной точки указать имя хоста компьютера.

Действия по установке привязки и адреса производятся в блоке 16 на схеме алгоритма, представленной на графическом материале ГУИР.351006-001 ПД.

5.2.6 Для обеспечения взаимодействия сервиса и его клиентов необходимо также реализовать контракт взаимодействия, который представляет собой некий интерфейс. Данный интерфейс необходимо сделать доступным и сервису, и клиентам. Однако, данный интерфейс необходимо реализовать только на сервисе. Клиенты же получают доступ к реализации с помощью технологии WCF – происходит удаленный вызов процедур.

5.2.7 В качестве контракта был реализован интерфейс `IService`. Методы данного интерфейса представлены в таблице 5.5.

Данный интерфейс реализован в классе `Service`. Экземпляры данного класса создаются при подключении клиентов к сервису. По сути, данный класс является заместителем, перенаправляющим запросы от клиентов к классу `MetadataList`. Также, для предоставления пользователю возможности просмотра подключений и запросов от клиентов, данный класс обеспечивает отображение подобной статистической информации.

Таблица 5.5 – Интерфейс контракта взаимодействия `IService`

Возвращаемое значение	Имя метода	Описание
<code>List<BookEntity></code>	<code>selectBooksByAuthor</code>	Получение списка книг на основании имени или любой части имени автора
<code>List<BookEntity></code>	<code>selectBooksByTitle</code>	Получение списка книг на основании названия или части названия книги
<code>List<BookEntity></code>	<code>selectBooksByGenre</code>	Получение списка книг, относящихся к требуемому жанру
<code>List<BookEntity></code>	<code>selectBooksByGenres</code>	Получение списка книг, принадлежащих одновременно всем искомым жанрам
<code>BookEntity</code>	<code>selectBookByID</code>	Получение записи книги на основании ее ID
<code>List<BookEntity></code>	<code>selectBooksByTemplate</code>	Получение списка книг, соответствующих шаблону
<code>Stream</code>	<code>extractBook</code>	Извлечение файла книги по ее записи
<code>Stream</code>	<code>extractBookByID</code>	Извлечение файла книги по ее ID
<code>List<BookEntity></code>	<code>getAvailableGenres</code>	Получение списка доступных жанров

5.2.8 Инициализация сервера входными данными производится при каждом его запуске. Пользователю, осуществляющему администрирование сервера, необходимо каждый раз заново вводить пути к файлам входных данных. Однако, для удобства пользователя, в серверном программном средстве предусмотрена возможность сохранения следующих настроек: пути файла жанров, файла метаданных, хранилища книг. Данные настройки сохраняются в папке пользователя и автоматически используются при запусках сервера.

Также была реализована возможность сброса всех настроек для их повторной установки. Стоит отметить, что программное средство запросит повторного ввода инициализирующих параметров в случае их отсутствия или

произошедшей ошибке при инициализации.

5.2.9 Для упрощения развертывания на компьютерах пользователя была реализована возможность создания инсталлятора. Практически каждый класс реализован с созданием отдельной динамически загружаемой библиотеки, поэтому для запуска программного средства необходимо их наличие. Инсталлятор же в автоматическом режиме размещает все необходимые библиотеки.

5.3 Разработка клиентского программного средства

5.3.1 Основное назначение клиента – предоставление удобного интерфейса для получения информации о книгах и самих книг. Поэтому главная задача при его реализации – создание графического интерфейса согласно макету, представленному на рисунке 4.1 в пункте 4.3.1. Реализованный интерфейс является масштабируемым и легко приспособляется под практически любые размеры экранов.

5.3.2 Для подключения клиента к серверу необходимо, чтобы пользователь указал адрес сервера, на котором размещен сервис электронной библиотеки.

Еще один параметр, который нужно указать при запуске приложения – это путь папки, в которую будут помещаться скачиваемые с сервера электронные книги.

5.3.3 Графический интерфейс предоставляет возможность поиска по жанру. Однако, для этого на клиенте и сервере необходимо наличие одинакового списка жанров. Поэтому при запуске клиента посылается запрос серверу на получение доступного списка жанров. Затем этот список используется для выбора жанров.

Данный запрос выполняет также функцию проверки правильности адреса сервера, указанного клиентом, и проверки доступности сервера: если получить список жанров не удастся, то будет предложено ввести другой адрес сервера.

5.3.4 Предположим, что сервер и клиент запущены в одной локальной сети. Пользователю необходимо узнать адрес сервера, чтобы использовать программное средство. Однако это может быть довольно сложной задачей.

Для решения данной проблемы была реализована возможность автообнаружения сервиса клиентом, предоставляемая технологией WCF. Суть ее в следующем: сервис постоянно широковещательно рассылает UDP пакеты с адресом и контрактом сервиса. На клиенте запускается специальный слушатель, который получает пакеты такого типа. Если клиент обнаруживает, что принят пакет от подходящего по контракту сервиса, то, значит, что сервер найден, можно получить его адрес и установить соединение.

5.3.5 Как уже было сказано выше, основной применяемый запрос для поиска на клиенте – это вызов метода `selectBooksByTemplate`. Для его вызова необходимо передать экземпляр класса `BookEntity`. Причем, большинство по-

лей может быть не заполнено. Этот случай означает, что по данному полю подходят все книги и отсеивание не производится.

5.3.6 Для получения файла книги используется метод `extractBook`, которому необходимо передать экземпляр класса `BookEntity`. Данный метод эффективнее, чем метод `extractBookById`, так как позволяет эффективнее использовать ресурсы сервера, что связано с тем, что нет необходимости на сервере вызывать методы получения записи книги по ее ID.

5.3.7 Для реализации возможности получения электронных книг в различных форматах использовалась программное средство `fb2conv` [7]. Оно предоставляет возможность его запуска с использованием параметров командной строки, что позволяет его добавить как часть функционала разрабатываемого программного средства.

Выходные форматы, предоставляемые программным средством `fb2conv`, представлены в таблице 5.6.

Таблица 5.6 – Выходные форматы конвертора `fb2conv`

Расширение формата	Название формата	Описание
epub	Electronic Publication	Открытый формат электронных версий книг, разработанный Международным форумом по цифровым публикациям [8]
mobi	Mobipocket	Формат, разработанный для использования в электронных книгах Amazon Kindle [9]
azw3	Kindle Format 8	Проприетарный формат, является эквивалентом формата <code>mobi</code> для книг под DRM защитой [10]

Было решено разместить реализацию функционала по конвертации на клиенте. Сначала клиент получает книгу в `fb2` формате, а потом, в соответствии с настройками, осуществляет преобразование в необходимый формат.

Для инкапсуляции преобразования форматов был реализован статический класс `FormatConvertor`. Вызов программного средства `fb2conv` производится программно в следующем форме:

```
fb2conv.exe --delete-source-file --output-format <Формат> <Входной файл>
```

Такой вызов конвертора позволяет осуществить выбор формата выходного файла, выбор входного файла, а также осуществить удаление входного файла по завершению преобразования. Выходной файл в необходимом формате будет создан в той же директории, что и входной.

5.3.8 Для того, чтобы предоставить пользователю не задавать параметры

программного средства при каждом его запуске, была реализована возможность их сохранения в файле настроек. При запуске происходит считывание файла настроек и выполняется попытка подключения к серверу (выполняется попытка скачивания жанров). Если же подключение установить не удастся, то перед запуском основной программы появляется окно, предназначенное для настройки программного средства.

5.3.9 Как и серверное программное приложение, клиент может распространяться и развертываться на компьютерах пользователей в виде инсталлятора. Однако, в отличие от сервера, клиент обращается к внешнему программному средству – fb2conv. Данное программное средство распространяется в виде скомпилированного исполняемого файла, поэтому он не требует установки и может поставляться в составе инсталлятора.

6 РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ

6.1 Администрирование сервера

6.1.1 Серверное программное средство поставляется в виде установочного файла setup.exe. После его запуска пользователь видит окно, представленное на рисунке 6.1.

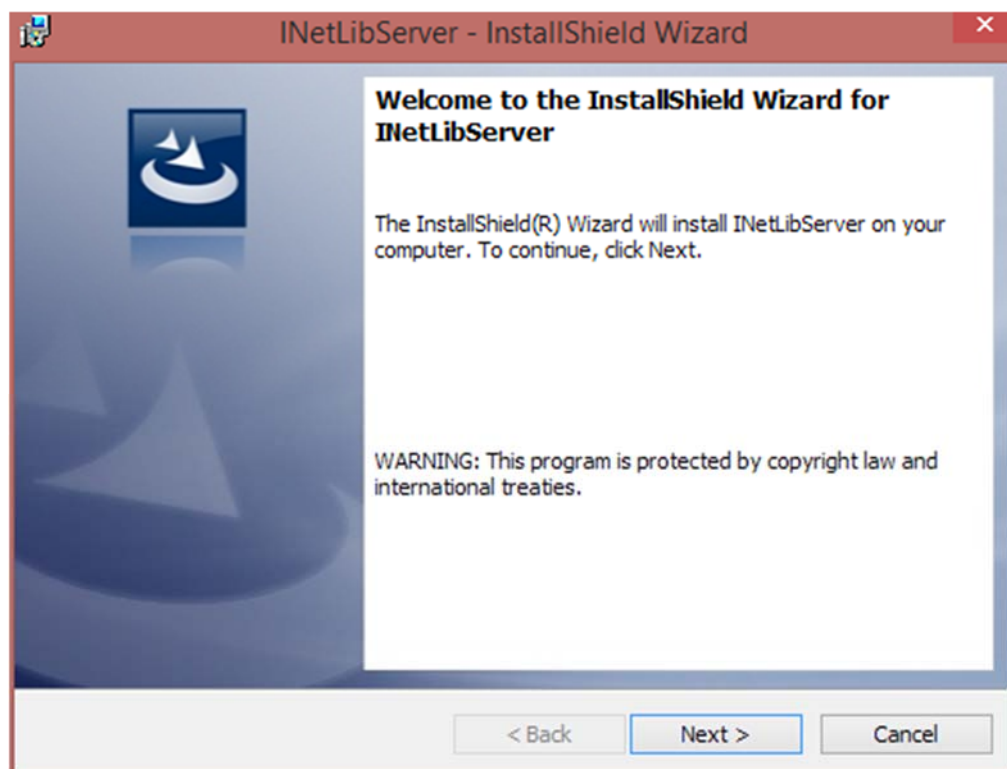


Рисунок 6.1 – Приветственное окно инсталлятора сервера

После нажатия кнопки Next пользователь может увидеть окно, с помощью которого можно осуществить выбор директории для установки серверного программного средства. Данное окно представлено на рисунке 6.2.

Затем пользователь видит окно, представленное на рисунке 6.3, на котором он может просмотреть информацию об установке.

По нажатию на кнопку Install начинается установка программного средства. По завершению установки появляется окно (см. рисунок 6.4) с предложением запустить сервер сразу по завершению установки.

В результате установки на рабочем столе и в меню (на экране) Пуск появляются ярлыки, с помощью которых можно производить запуск программного средства.

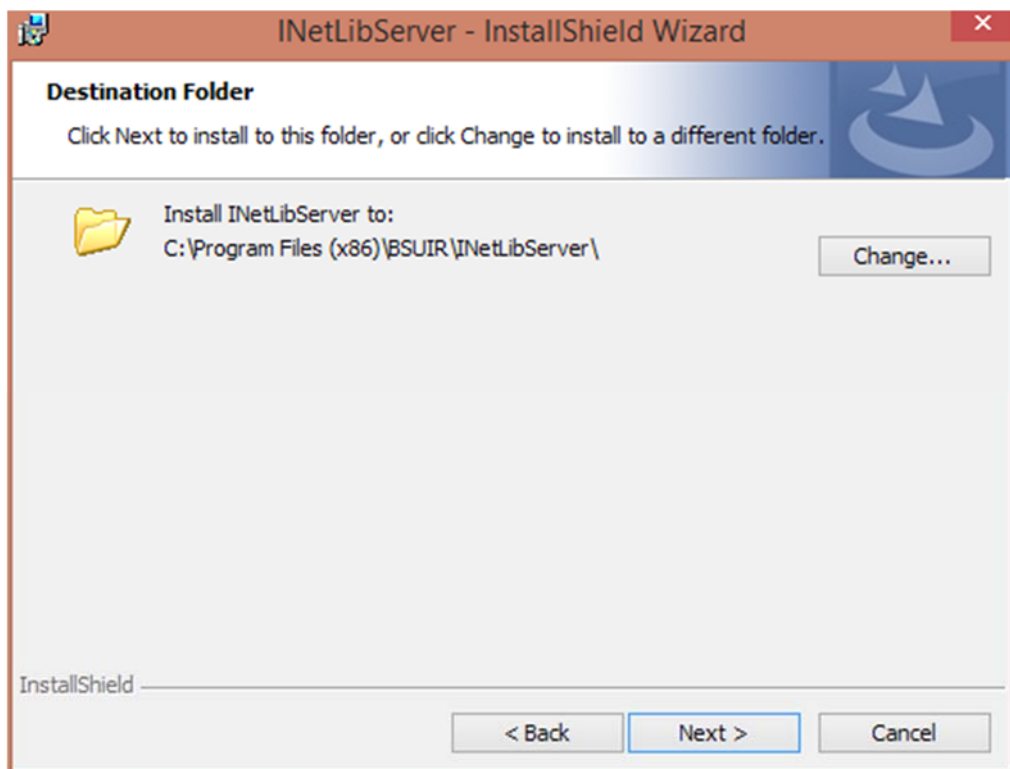


Рисунок 6.2 – Выбор директории установки

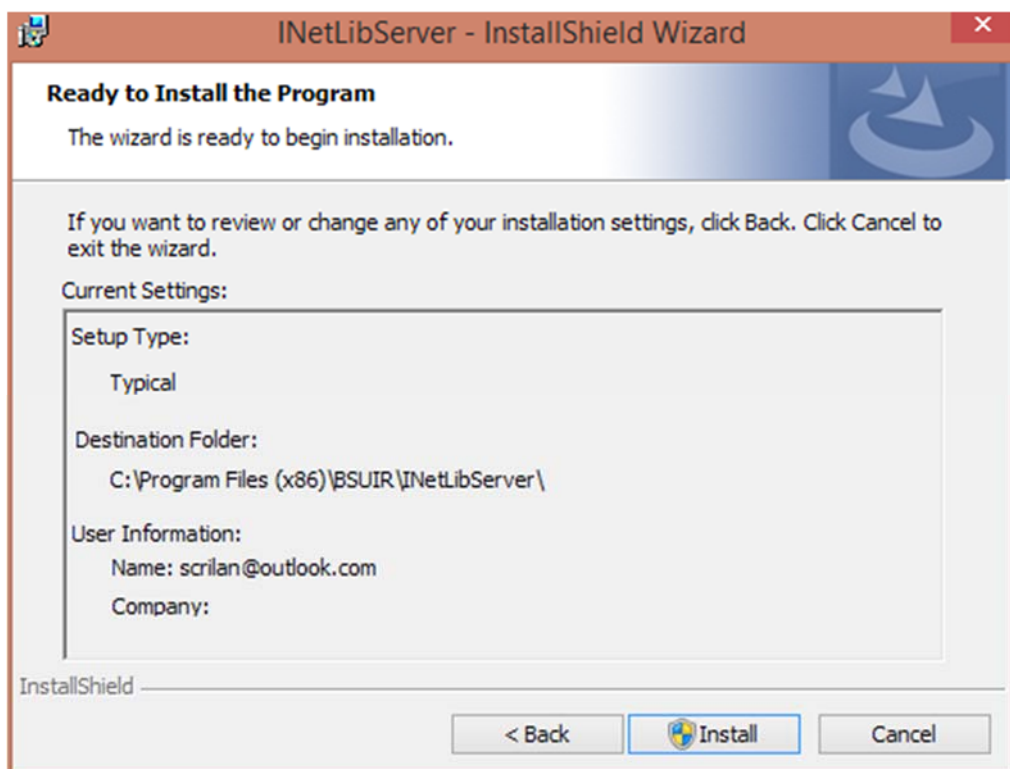


Рисунок 6.3 – Просмотр информации об установке

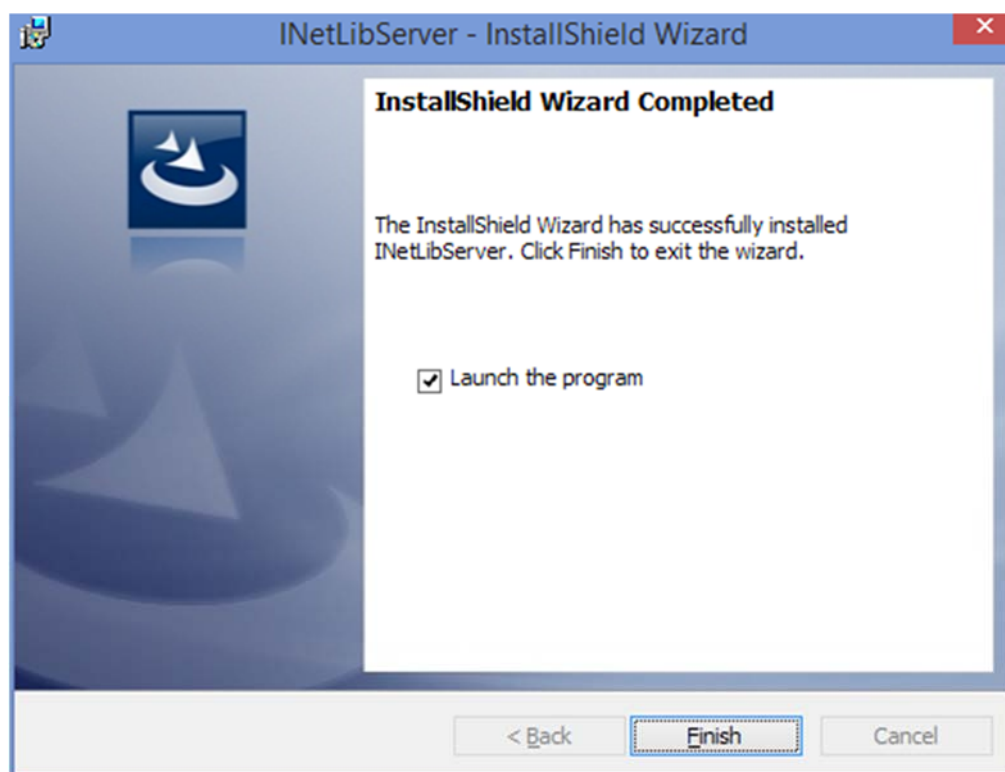


Рисунок 6.4 – Завершение установки сервера

6.1.2 Если программное средство до этого не устанавливалось на компьютер пользователя (если отсутствует файл настроек), то при первом его запуске потребуются ввод следующих данных: путь к файлу жанров, путь к файлу метаданных, путь к хранилищу книг. Если в процессе инициализации сервера выяснится, что какой-либо из путей неверен, то приложение затребует повторный ввод этого пути.

По завершению всех инициализаций об этом отображается сообщение и производится попытка запустить сервис. После его успешного запуска отображается на каком адресе и порту он располагается.

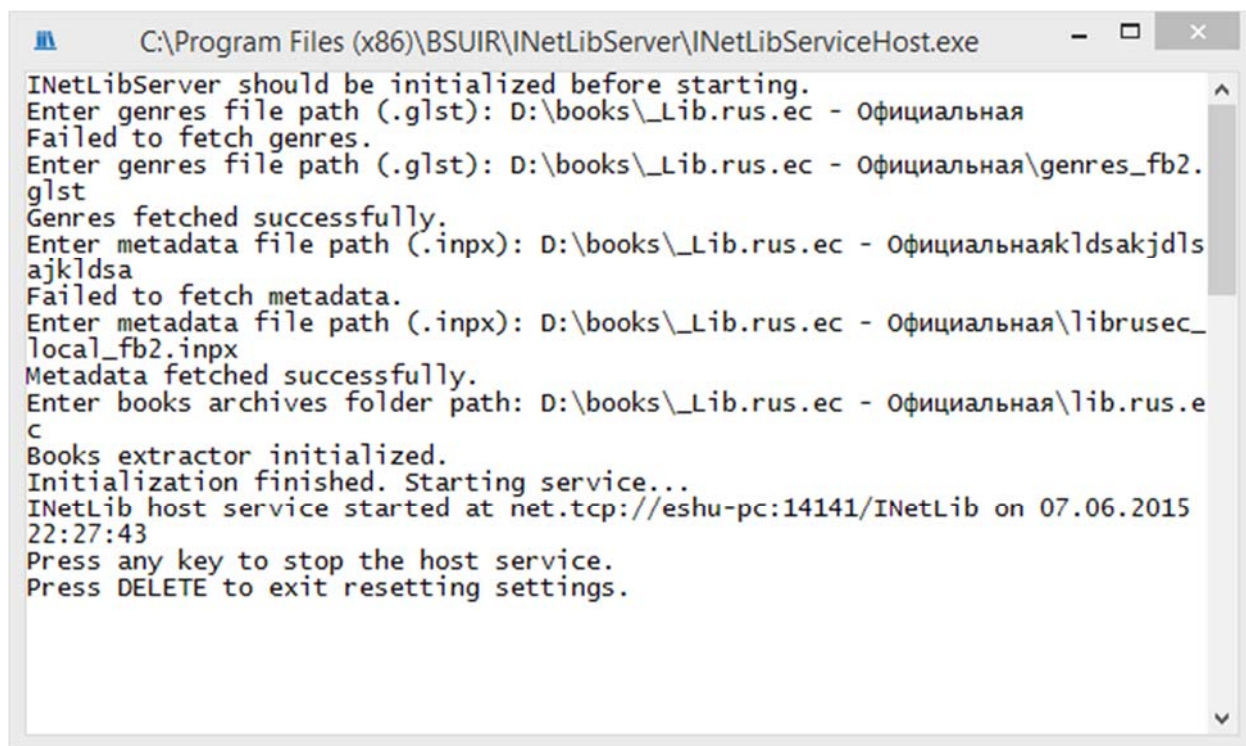
Остановить сервис и выйти из приложения можно по нажатию любой клавиши на клавиатуре. Если же нажать клавишу Delete, то помимо завершения приложения будут удалены все настройки программного средства.

Пример окна сервера приведен на рисунке 6.5.

6.1.3 Если же приложение уже однажды запускалось на компьютере пользователя (повторный запуск мог произойти и после переустановки программного средства; то есть произошло сохранение настроек приложения на компьютере пользователя), то можно увидеть окно, представленное на рисунке 6.6. Программное средство с помощью сообщений оповещает пользователя о том, что был обнаружен файл настроек, была произведена попытка инициализации сервера с использованием этих настроек, попытка оказалось успешной, сервер успешно запущен.

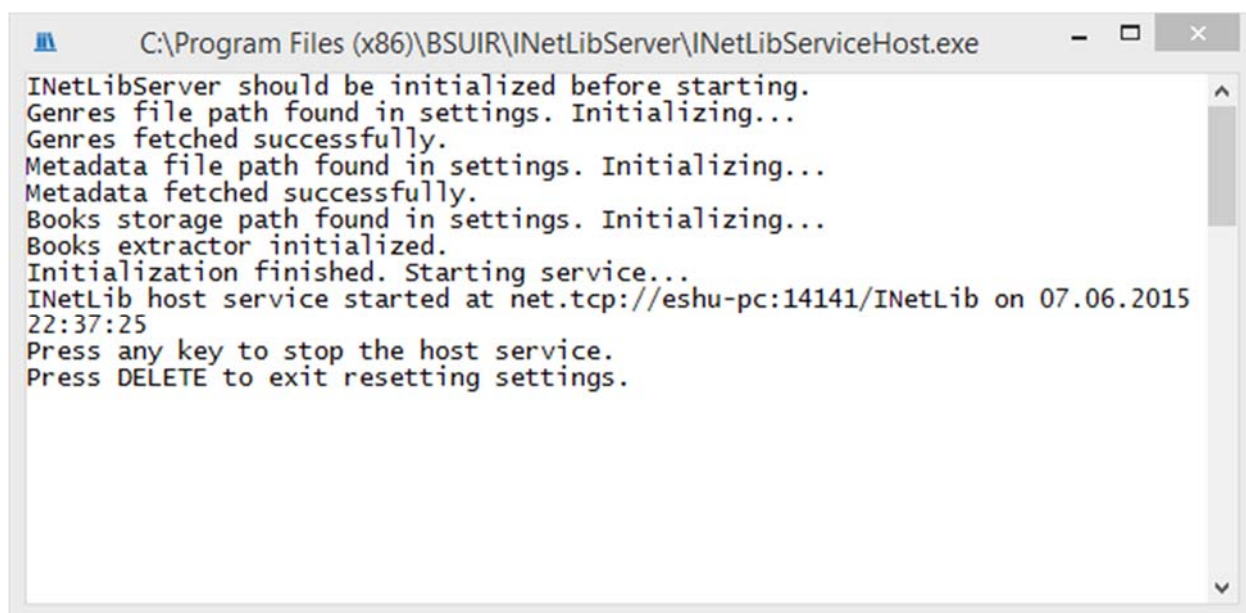
Если только какое-либо одно из полей оказалось умышленно или случайно поврежденным, то при инициализации сервера это выявится и будет

предложено ввести заново значение конкретного ошибочного параметра.



```
C:\Program Files (x86)\BSUIR\INetLibServer\INetLibServiceHost.exe
INetLibServer should be initialized before starting.
Enter genres file path (.glst): D:\books\_Lib.rus.ec - Официальная
Failed to fetch genres.
Enter genres file path (.glst): D:\books\_Lib.rus.ec - Официальная\genres_fb2.
glst
Genres fetched successfully.
Enter metadata file path (.inpx): D:\books\_Lib.rus.ec - Официальная\kldsakjdl
ajklksa
Failed to fetch metadata.
Enter metadata file path (.inpx): D:\books\_Lib.rus.ec - Официальная\librusec_
local_fb2.inpx
Metadata fetched successfully.
Enter books archives folder path: D:\books\_Lib.rus.ec - Официальная\lib.rus.e
c
Books extractor initialized.
Initialization finished. Starting service...
INetLib host service started at net.tcp://eshu-pc:14141/INetLib on 07.06.2015
22:27:43
Press any key to stop the host service.
Press DELETE to exit resetting settings.
```

Рисунок 6.5 – Запуск сервера с вводом параметров пользователем



```
C:\Program Files (x86)\BSUIR\INetLibServer\INetLibServiceHost.exe
INetLibServer should be initialized before starting.
Genres file path found in settings. Initializing...
Genres fetched successfully.
Metadata file path found in settings. Initializing...
Metadata fetched successfully.
Books storage path found in settings. Initializing...
Books extractor initialized.
Initialization finished. Starting service...
INetLib host service started at net.tcp://eshu-pc:14141/INetLib on 07.06.2015
22:37:25
Press any key to stop the host service.
Press DELETE to exit resetting settings.
```

Рисунок 6.6 – Запуск сервера с автоматической инициализацией

6.1.4 Реализация сервиса предусматривает вывод информации о приходящих от клиентов запросах. Пример таких сообщений можно увидеть на рисунке 6.7.

Можно видеть, что после приветственного сообщения и сообщений инициализации выведено несколько строк вида:

Request for genres from [<IP-клиента>]:<Порт-клиента>

Можно проследить, с каких IP адресов подключаются клиенты.

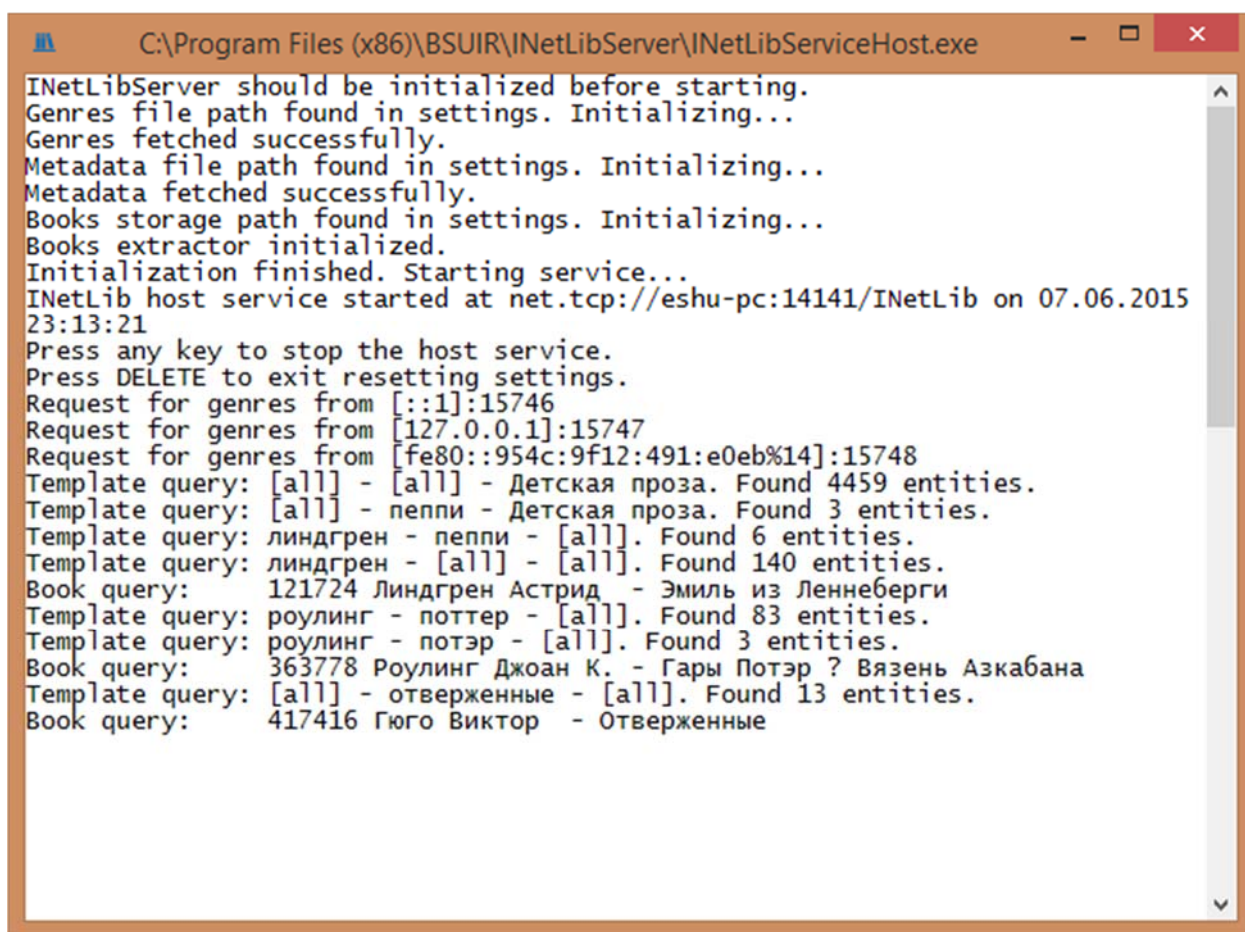
После этого можно видеть приходящие запросы на поиск формата:

Template query: <Автор> - <Название> - <Жанр>. Found <Число-найденных-книг> entities.

Если в приходящем шаблоне для поиска какое-либо из полей не заполнено, то подразумевается, что отсеивание по этому поле производить не нужно. В таком случае вместо содержимого этого поля выводится надпись [all].

Также можно видеть приходящие запросы на получение файлов книг вида:

Book query: <ID-книги> <Автор> - <Название>



```
C:\Program Files (x86)\BSUIR\INetLibServer\INetLibServiceHost.exe
INetLibServer should be initialized before starting.
Genres file path found in settings. Initializing...
Genres fetched successfully.
Metadata file path found in settings. Initializing...
Metadata fetched successfully.
Books storage path found in settings. Initializing...
Books extractor initialized.
Initialization finished. Starting service...
INetLib host service started at net.tcp://eshu-pc:14141/INetLib on 07.06.2015
23:13:21
Press any key to stop the host service.
Press DELETE to exit resetting settings.
Request for genres from [::1]:15746
Request for genres from [127.0.0.1]:15747
Request for genres from [fe80::954c:9f12:491:e0eb%14]:15748
Template query: [all] - [all] - Детская проза. Found 4459 entities.
Template query: [all] - пеппи - Детская проза. Found 3 entities.
Template query: линдгрэн - пеппи - [all]. Found 6 entities.
Template query: линдгрэн - [all] - [all]. Found 140 entities.
Book query: 121724 Линдгрэн Астрид - Эмиль из Леннеберги
Template query: роулинг - поттер - [all]. Found 83 entities.
Template query: роулинг - потэр - [all]. Found 3 entities.
Book query: 363778 Роулинг Джоан К. - Гары Потэр ? Вязень Азкабана
Template query: [all] - отверженные - [all]. Found 13 entities.
Book query: 417416 Гюго Виктор - Отверженные
```

Рисунок 6.7 – Пример вывода информации о подключениях и запросах

6.1.5 Не допускается запуск нескольких сущностей программного средства одновременно. При попытке такого запуска, пользователь увидит сообщение об ошибке. Пример сообщения приведен на рисунке 6.8.

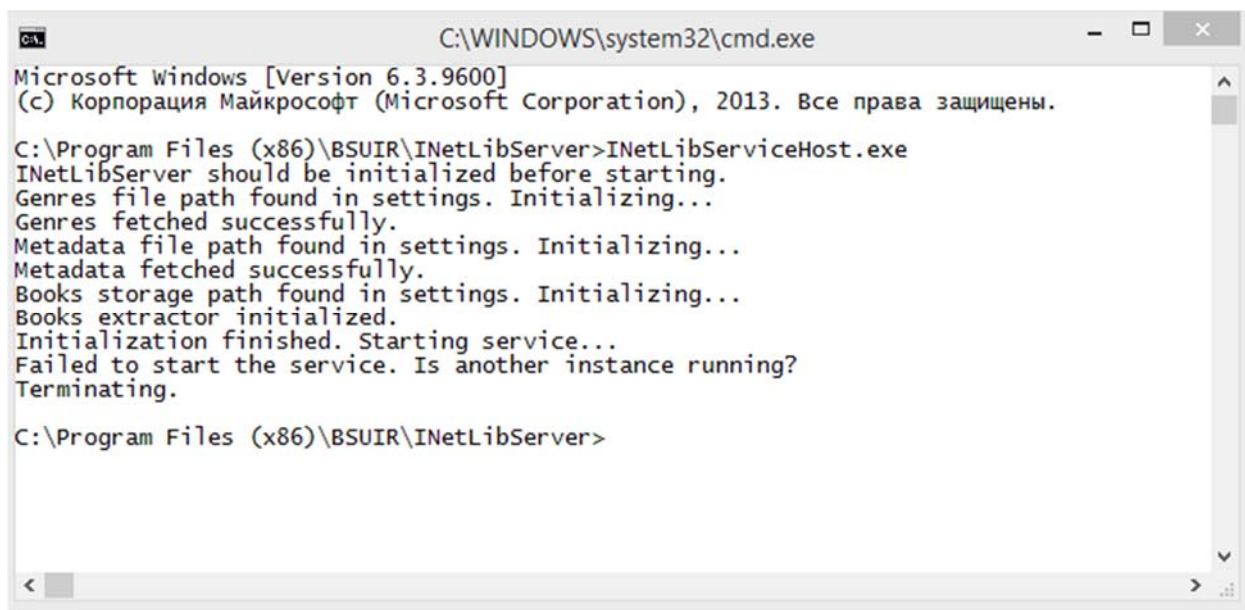


Рисунок 6.8 – Пример ошибки при запуске сервиса

6.2 Руководство по использованию клиента

6.2.1 Клиентское программное средство, как и серверное, может развертываться на компьютерах пользователей с помощью инсталлятора. Процесс установки аналогичен рассмотренному в пункте 6.1.1.

6.2.2 При первом запуске, если программное средство не устанавливалось на компьютер пользователя ранее, потребуется ввод некоторых параметров, необходимых для работы. Пользователь видит окно настройки, представленное на рисунке 6.9.

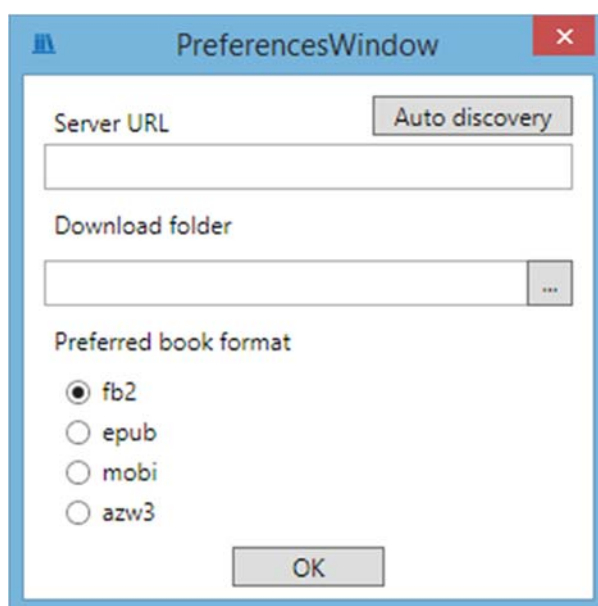


Рисунок 6.9 – Окно настройки клиентского программного средства

Как видно из рисунка 6.9, необходимо задать адрес серверного программного средства, папку для скачивания книг и выбрать желаемый формат книг.

В качестве адреса сервера нужно вводить имя хоста или IP-адрес. Пользователь может не знать адреса сервера, и если и клиент, и сервер развернуты в одной сети, то можно воспользоваться функцией автообнаружения, для чего необходимо нажать кнопку Autodiscovery. По завершении ее работы адрес хоста сервера (если он будет найден) будет подставлен в поле для ввода адреса сервера автоматически.

Для выбора папки для скачивания пользователь может воспользоваться диалогом выбора папки, появляющийся по нажатию кнопки ... и представленный на рисунке 6.10, или же ввести адрес папки вручную. Если будет введен неверный путь к папке, то программное средство затребует его повторный ввод.

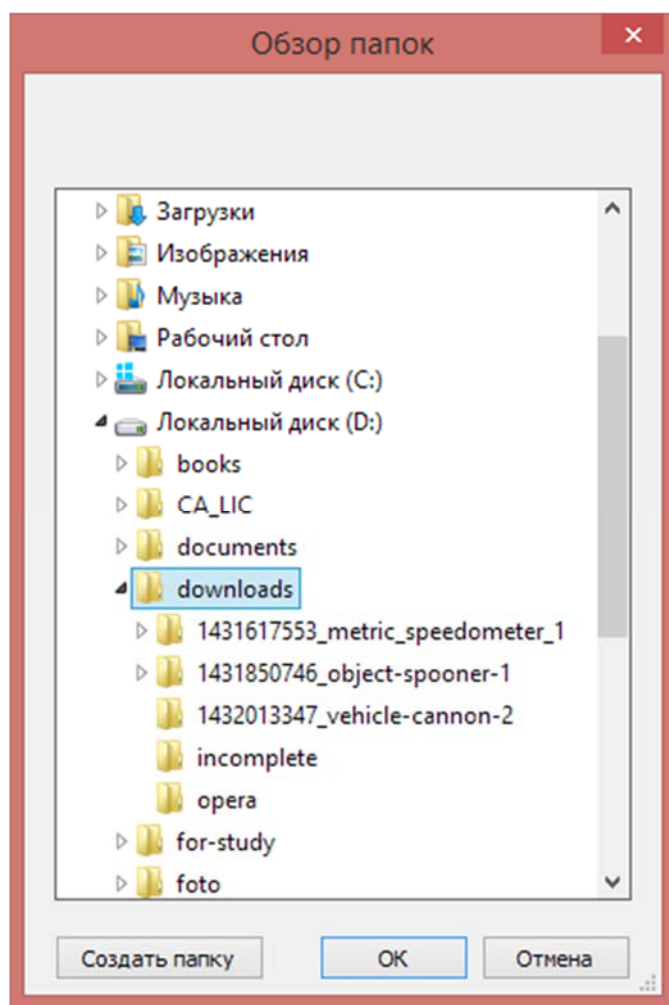


Рисунок 6.10 – Диалог выбора папки скачивания

И последний параметр, который необходимо задать пользователю – это формат, в который будут конвертироваться скачиваемые книги. По умолчанию выбран формат fb2.

Окно настроек со всеми заполненными полями представлено на рисунке 6.11.

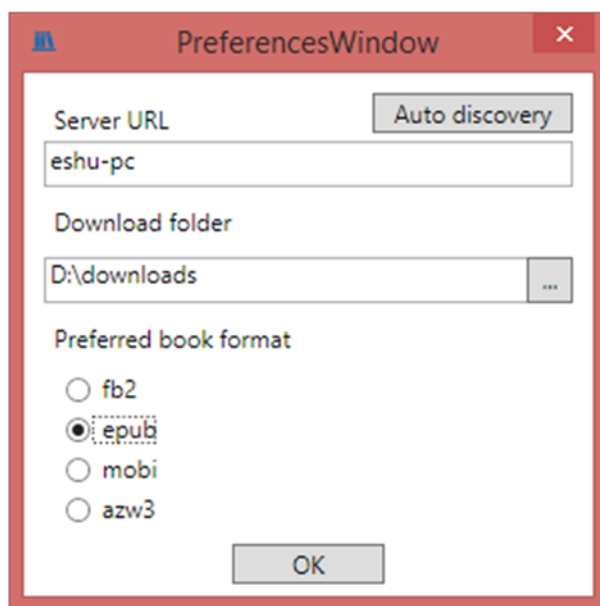


Рисунок 6.11 – Заполненное окно настроек

Если введен неверный адрес сервера и программное средство не может установить соединение, то появится окно, представленное на рисунке 6.12, а после его закрытия снова появится окно ввода настроек.

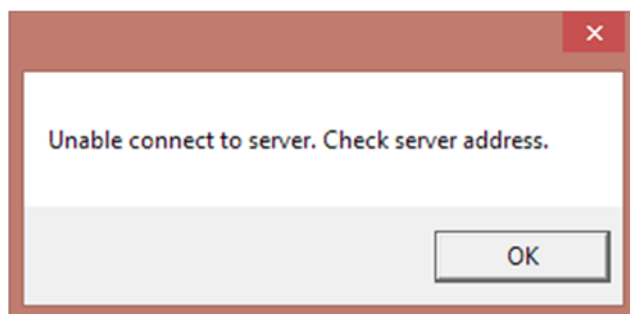


Рисунок 6.12 – Окно сообщения об ошибке при вводе адреса сервера

6.2.3 Если же соединение установить удалось, то появится основное окно клиентского программного средства. Пример окна представлен на рисунке 6.13.

6.2.4 Как уже упоминалось, основная функция клиентского программного средства – поиск книг по шаблону, причем в шаблон входят следующие параметры: имя автора, название книги, жанр книги.

Поля имени автора и названия книг являются текстовыми и допускают ввод неполного имени автора или названия. Поиск не является регистрочувствительным. Поле жанра представляет собой выпадающий список, так как жанры заранее определены. Пример ввода шаблона для поиска представлен на

рисунке 6.14.

6.2.5 По нажатию кнопки Search, если запрос на сервер произойдет успешно, то пользователь увидит список книг, соответствующих его запросу. Он может выбирать элементы списка, при этом в нижней части окна будет обновляться отображаемая информация о книге. Пример окна с выведенными результатами поиска приведен на рисунке 6.15.

6.2.6 После того, как пользователь осуществит выбор книги, он может скачать ее. Для этого необходимо нажать кнопку Download.

Если в настройках программного средства выбран не fb2 формат, то по завершению скачивания в отдельном окне будет вызван конвертер fb2conv. Пример окна конвертера приведен на рисунке 6.16.

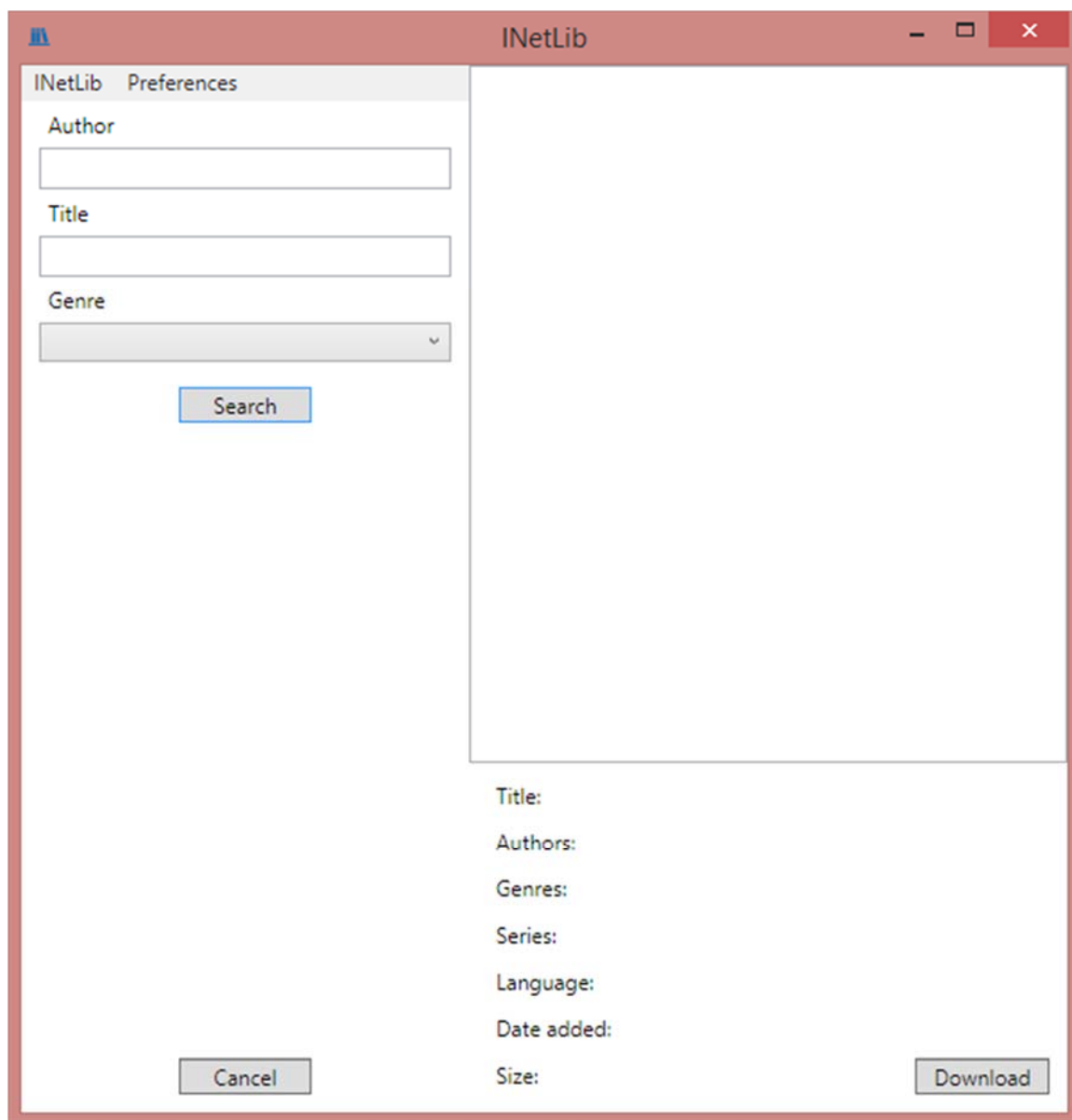


Рисунок 6.13 – Основное окно клиента

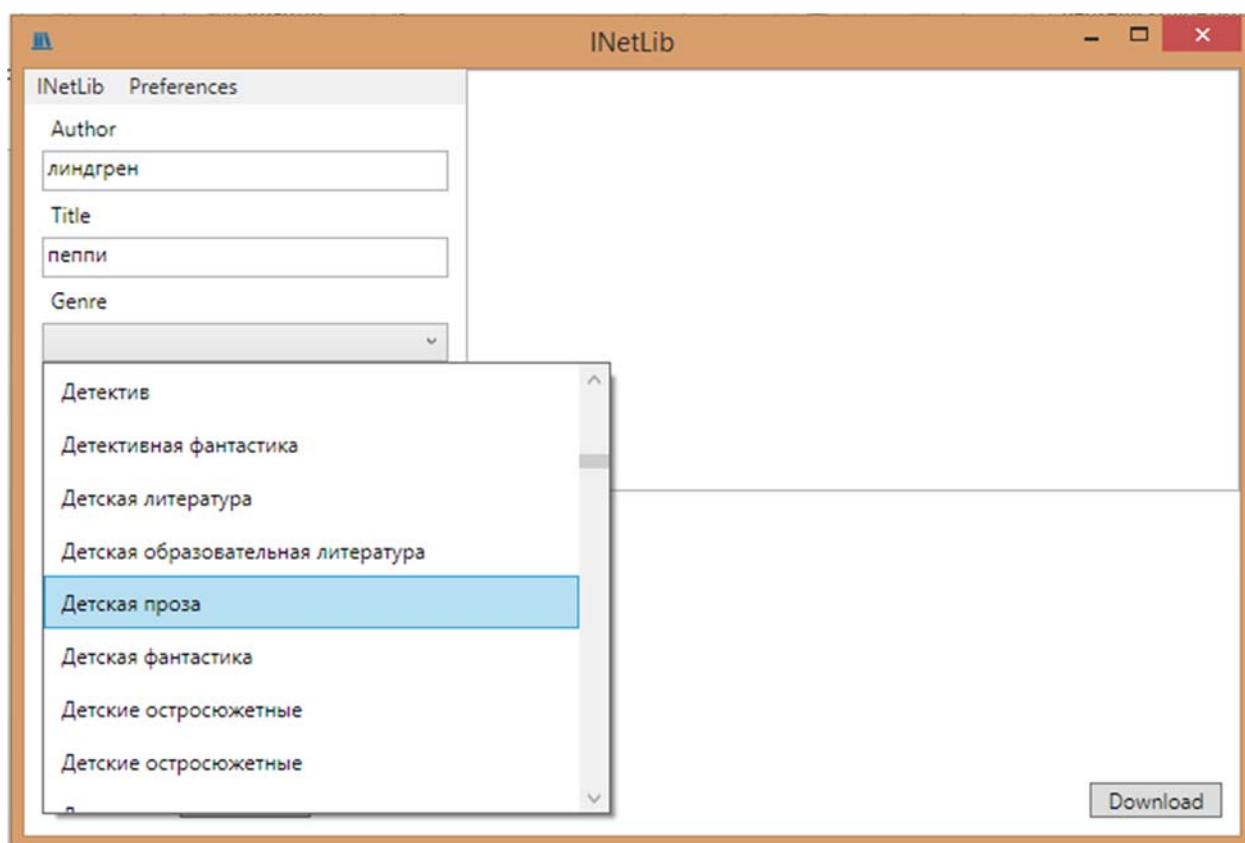


Рисунок 6.14 – Ввод шаблона для поиска

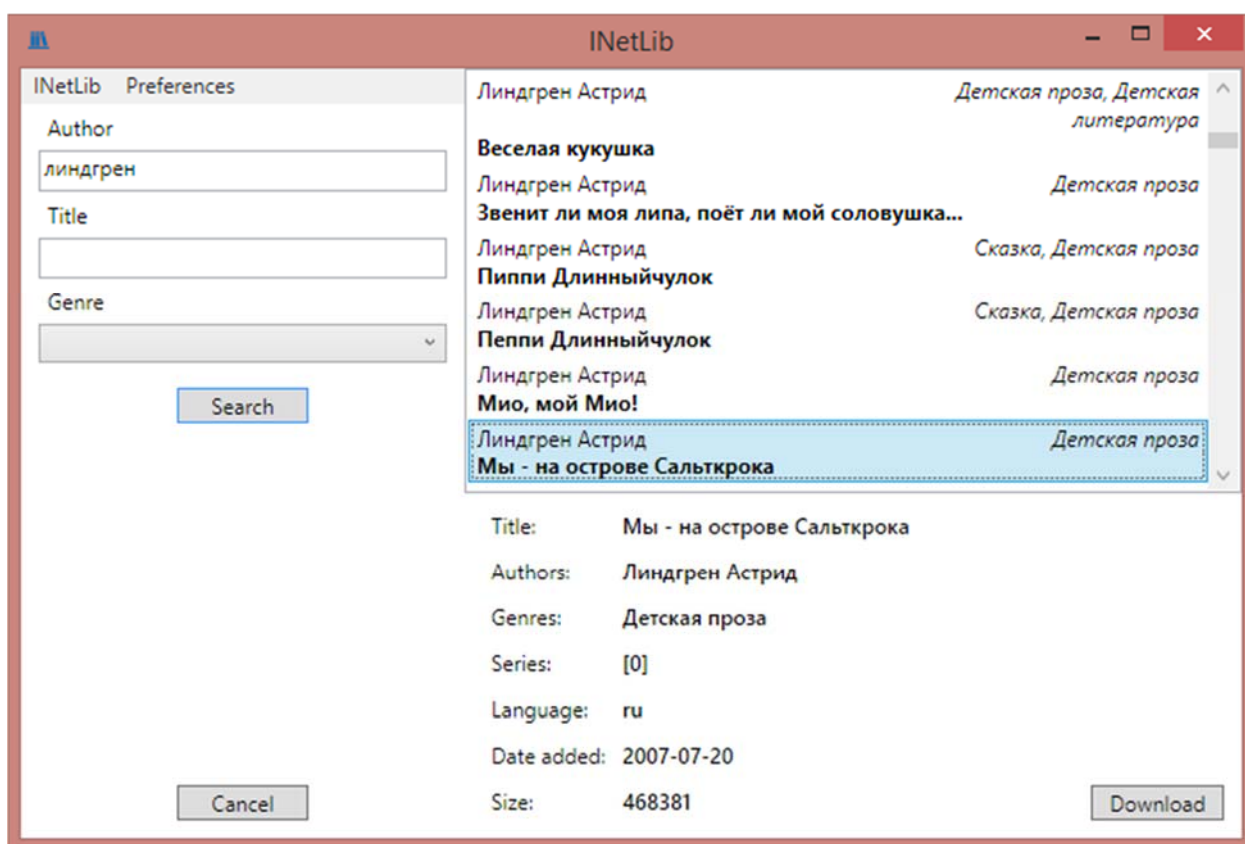


Рисунок 6.15 – Выбор книги и просмотр ее краткой информации

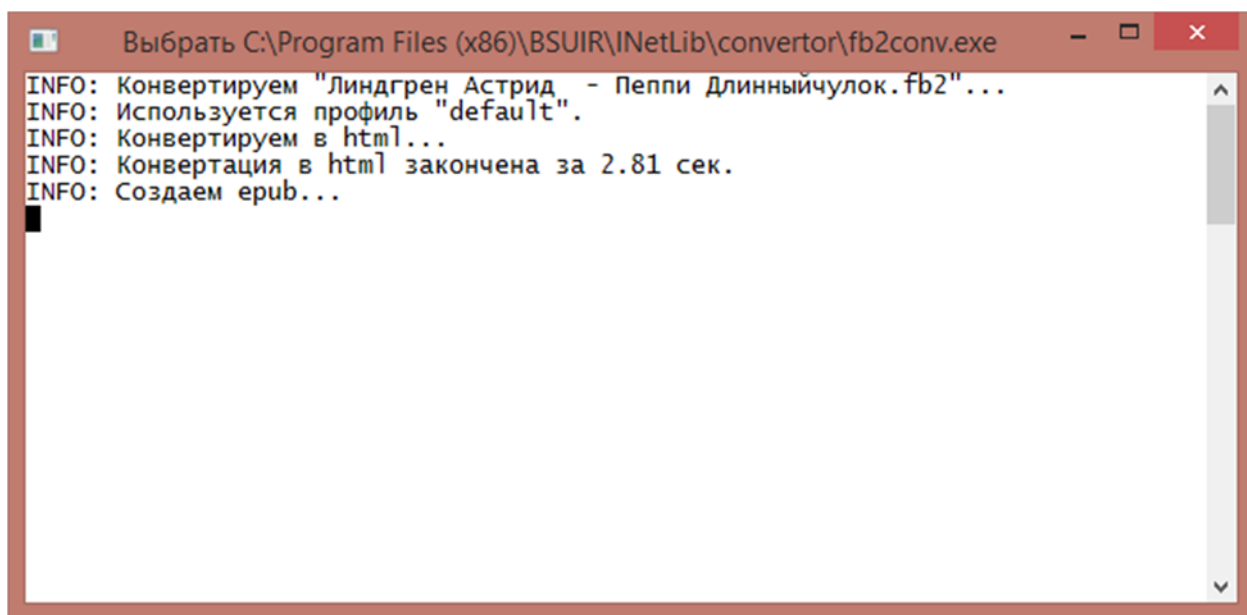


Рисунок 6.16 – Окно конвертера fb2conv

ЗАКЛЮЧЕНИЕ

В ходе работы над курсовым проектом было разработано программное средство, позволяющее организовывать доступ к книгам копии электронной библиотеки Либрусек (так называемое зеркало электронной библиотеки).

Разработанное программное средство является сервис-ориентированным программным средством. Для его реализации была изучена такая технология для организации подобных систем, как WCF – Windows Communication Foundation.

Для создания легко развёртываемых приложений была изучена и использована технология создания инсталляторов.

Для реализации возможности сохранения настроек приложения были изучены и использованы технологии, предоставляемые фреймворком .Net.

При создании графических интерфейсов пользователя были закреплены умения по использованию технологии WPF - Windows Presentation Foundation.

Также были изучены форматы файлов, наиболее часто применяемых для создания служебных файлов электронных библиотек; были реализованы структуры для хранения информации этих файлов в памяти компьютера и модули их импорта.

При реализации курсового проекта были закреплены навыки программирования на языке C# с учетом применения вышеперечисленных технологий.

При дальнейшей разработке программного средства будут реализованы следующие возможности:

- иерархическая структура хранения и отображения жанров книг;
- возможность выбора нескольких жанров для поиска;
- сбор и сохранение статистики подключения клиентов к серверу; на основании этих данных кеширование и оптимизация данных на сервере;
- кеширование результатов запросов клиентом;
- реализация клиентских программных средств для различных платформ и операционных систем, в том числе реализация веб-версии;
- расширение графического интерфейса для включения в шаблон поиска максимального числа доступных полей записей книг;

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Википедия, Книга [Электронный ресурс]. - Электронные данные. - Режим доступа: <https://ru.wikipedia.org/wiki/Книга>.
- [2] Википедия, Электронная книга [Электронный ресурс]. - Электронные данные. - Режим доступа: https://ru.wikipedia.org/wiki/Электронная_книга.
- [3] The Guardian, Michael Hart, inventor of the ebook, dies aged 64 [Электронный ресурс]. - Электронные данные. - Режим доступа: <http://www.theguardian.com/books/2011/sep/08/michael-hart-inventor-ebook-dies>.
- [4] Либрусек [Электронный ресурс]. - Электронные данные. - Режим доступа: <http://lib.rus.ec>.
- [5] Booktracker, Илья Ларин: библиотека Либрусек [Электронный ресурс]. - Электронные данные. - Режим доступа: <http://booktracker.org/viewtopic.php?t=41025>.
- [6] Флибуста [Электронный ресурс]. - Электронные данные. - Режим доступа: <http://flibusta.net>.
- [7] dnk_dz, Конвертер формата fb2 в epub, mobi, azw3 для MS Windows, Mac OS и Linux [Электронный ресурс]. - Электронные данные. - Режим доступа: <http://www.the-ebook.org/forum/viewtopic.php?t=28447>.
- [8] Wikipedia, EPUB [Электронный ресурс]. - Электронные данные. - Режим доступа: <https://en.wikipedia.org/wiki/EPUB>.
- [9] exler.ru, Чем отличаются форматы электронных книг и какой формат предпочесть [Электронный ресурс]. - Электронные данные. - Режим доступа: <http://www.exler.ru/likbez/11-07-2012.htm>.
- [10] Wikipedia, Amazon Kindle [Электронный ресурс]. - Электронные данные. - Режим доступа: http://en.wikipedia.org/wiki/Amazon_Kindle#Proprietary_formats_.28AZW.2C_KF8.29.
- [11] Доманов, А.Т. Стандарт предприятия. Дипломные проекты (работы). Общие требования / А. Т. Доманов, Н. И. Сорока - Минск: БГУИР, 2010.
- [12] ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. - Введ. 01.01.1992, М: Изд-во стандартов, 1991.
- [13] Википедия, Информационный взрыв [Электронный ресурс]. - Электронные данные. - Режим доступа: https://ru.wikipedia.org/wiki/Информационный_взрыв.

ПРИЛОЖЕНИЕ А
(обязательное)
Исходные коды методов

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Runtime.Serialization;
namespace GenresList
{
    [DataContract]
    static public class GenresList
    {
        [DataMember]
        public static List<GenresListEntity> genres = new List<GenresListEntity>();

        public static void initialize(List<GenresListEntity> availableGenres)
        {
            genres = availableGenres;
            genres.Sort((x, y) => String.CompareOrdinal(x.description, y.description));
        }
        public static void initialize(string genresFilePath)
        {
            tryReadGenresFromFile(genresFilePath);
            genres.Sort((x, y) => String.CompareOrdinal(x.description, y.description));
        }
        private static void tryReadGenresFromFile(string genresFilePath)
        {
            readGenresFromFile(genresFilePath);
        }
        private static void readGenresFromFile(string genresFilePath)
        {
            StreamReader genresFile = new StreamReader(genresFilePath);
            readGenresByLine(genresFile);
            genresFile.Close();
        }
        private static void readGenresByLine(TextReader genresFile)
        {
            string line = genresFile.ReadLine();
            while (line != null)
            {
                addGenresIgnoringComments(line);
                line = genresFile.ReadLine();
            }
        }
    }
}
```

```

}
private const int firstCharacterPosition = 0;
private static void addGenresIgnoringComments(string line)
{
    if (line[firstCharacterPosition] == '#') return;
    if (line.Length == 0) return;
    GenresListEntity genre = new GenresListEntity(line);
    genres.Add(genre);
}
static public string getGenreName(int id)
{
    return genres[id].name;
}
static public string getGenreDescription(int id)
{
    try
    {
        return id < 0 ? "[all]" : genres[id].description;
    }
    catch
    {
        return "";
    }
}
public const int notFoundID = -1;
static public int getGenreID(string genreString)
{
    return genres.FindIndex(entity => entity.name == genreString);
}
public static void printGenresListDebug()
{
    foreach (var genre in genres)
    {
        Console.WriteLine("{0}.{1} {2}: {3}", genre.genreNumber, genre.subgenreNumber, genre.name, genre.description);
    }
}
public static List<GenresListEntity> getAvailableGenres()
{
    return genres;
}
}
}
}

```

```

using System.Collections.Generic;
using System.IO;
using System.IO.Compression;
using System.Linq;
namespace InpxImport
{
    public static class InpxImport
    {
        const string inpFileExtension = ".inp";
        public static List<BookEntity.BookEntity> import(string inpxFilePath)
        {
            return getBooksListFromInpxFile(inpxFilePath);
        }
        private static List<BookEntity.BookEntity> getBooksListFromInpxFile(string
inpxFilePath)
        {
            List<BookEntity.BookEntity> booksList = new List<BookEntity.BookEntity>();

            ZipArchive archive = ZipFile.OpenRead(inpxFilePath);
            foreach (var archiveEntry in archive.Entries.Where(isInpFile))
            {
                booksList.AddRange(getBookEntitiesFromInpArchiveEntry(archiveEntry));
            }
            return booksList;
        }
        private static List<BookEntity.BookEntity> getBookEntitiesFromInpAr-
archiveEntry(ZipArchiveEntry archiveEntry)
        {
            StreamReader reader = new StreamReader(archiveEntry.Open());
            List<BookEntity.BookEntity> bookEntitiesFromFile = new
List<BookEntity.BookEntity>();
            while (!reader.EndOfStream)
            {
                var bookEntity = createBookEntity(reader.ReadLine());
                bookEntity.setArchiveName(Path.ChangeExtension(archiveEntry.Name,
"zip")); //Have to initialize archiveName field outside the constructor
                bookEntitiesFromFile.Add(bookEntity); //of the BookEntity class cause the con-
structor designed to parse book metadata
            }
            return bookEntitiesFromFile;
        }
    }
}

```

```

private static void setArchiveName(this BookEntity.BookEntity book, string archiveName)
{
    book.archiveName = archiveName;
}
private static BookEntity.BookEntity createBookEntity(string bookData)
{
    return new BookEntity.BookEntity(bookData);
}
private static bool isInpFile(ZipArchiveEntry inpFile)
{
    return Path.GetExtension(inpFile.Name) == inpFileExtension;
}
}
}
}

```

```

using System;
using System.Collections.Generic;
using System.Runtime.Serialization;
namespace BookEntity
{
    [DataContract]
    public class BookEntity
    {
        //Для совместимости сюда включены все доступные поля.
        [DataMember]
        public Authors authors { get; set; } //Список авторов
        [DataMember]
        public Genres genres { get; set; } //Список жанров
        [DataMember]
        public string title { get; set; } //Название
        [DataMember]
        public string seriesTitle { get; set; } //Название серии
        [DataMember]
        public int numberInSeries { get; set; } //Номер книги в серии
        [DataMember]
        public string fileName { get; set; } //Имя файла книги [DataMember]
        public int fileSize { get; set; } //Размер файла с книгой [DataMember]
        public int bookID { get; set; } //Уникальный ID книги
        [DataMember]
        public bool isDeleted { get; set; } //Признак удаления
        [DataMember]
        public string extension { get; set; } //Расширение файла [DataMember]
        public string dateAdded { get; set; } //Дата добавления файла [DataMember]
    }
}

```

```

public string language { get; set; }//Язык книги
[DataMember]
public string bookRate { get; set; }//Внешний рейтинг книги
[DataMember]
public string keywords { get; set; }//Теги
    [DataMember]
public string archiveName { get; set; }
//Имя архива с файлом книги
//(Имя архива совпадает с именем файла с метаданными в inrx файле)
public BookEntity() { }
public BookEntity(string bookRawInfo)
{
    parseBookRawInfo(bookRawInfo);
}
private void parseBookRawInfo(string bookRawInfo)
{
    try
    {
        splitAndParseBookInfo(bookRawInfo);
    }
    catch
    { }
}

private void splitAndParseBookInfo(string bookRawInfo)
{
    const char bookInfoDelimiter = (char)0x04;
    string[] splittedBookInfo = bookRawInfo.Split(bookInfoDelimiter);
    setBookInfo(splittedBookInfo);
}
private enum BookInfo
{
    Authors,
    Genres,
    Title,
    SeriesTitle,
    NumberInSeries,
    FileName,
    FileSize,
    BookID,
    IsDeleted,
    Extension,
    DateAdded,
    Language,

```

```

BookRate,
Keywords
}
private void setBookInfo(IReadOnlyList<string> bookInfo)
{
    setAuthors(bookInfo[(int)BookInfo.Authors]);
    setGenres(bookInfo[(int)BookInfo.Genres]);
    setTitle(bookInfo[(int)BookInfo.Title]);
    setSeriesTitle(bookInfo[(int)BookInfo.SeriesTitle]);
    setNumberInSeries(bookInfo[(int)BookInfo.NumberInSeries]);
    setFileName(bookInfo[(int)BookInfo.FileName]);
    setFileSize(bookInfo[(int)BookInfo.FileSize]);
    setBookID(bookInfo[(int)BookInfo.BookID]);
    setDeletionFlag(bookInfo[(int)BookInfo.IsDeleted]);
    setExtension(bookInfo[(int)BookInfo.Extension]);
    setDateAdded(bookInfo[(int)BookInfo.DateAdded]);
    setLanguage(bookInfo[(int)BookInfo.Language]);
    setBookRate(bookInfo[(int)BookInfo.BookRate]);
    setKeywords(bookInfo[(int)BookInfo.Keywords]);
}
private void setAuthors(string authorsString)
{
    authors = new Authors(authorsString);
}
private void setGenres(string genresString)
{
    genres = new Genres(genresString);
}
private void setTitle(string title)
{
    this.title = title;
}
private void setSeriesTitle(string seriesTitle)
{
    this.seriesTitle = seriesTitle;
}
private void setNumberInSeries(string numberInSeriesString)
{
    if (numberInSeriesString.Length == 0)
        numberInSeries = 0;
    else
        numberInSeries = Convert.ToInt32(numberInSeriesString);
}
private void setFileName(string fileName)

```



```

{
this.fileName = fileName;
}
private void setFileSize(string fileSizeString)
{
if (fileSizeString.Length == 0)
fileSize = 0;
else
fileSize = Convert.ToInt32(fileSizeString);
}
private void setBookID(string bookIDString)
{
if (bookIDString.Length == 0)
bookID = 0;
else
bookID = Convert.ToInt32(bookIDString);
}
private void setDeletionFlag(string deletedFlagString)
{
isDeleted = (deletedFlagString == "1");
}

private void setExtension(string extension)
{
this.extension = extension;
}
private void setDateAdded(string dateAdded)
{
this.dateAdded = dateAdded;
}
private void setLanguage(string language)
{
this.language = language;
}
private void setBookRate(string bookRate)
{
this.bookRate = bookRate;
}
private void setKeywords(string keywords)
{
this.keywords = keywords;
}
public override string ToString()
{

```

```

return string.Format("{0} {1} - {2}", bookID, authors.getAuthors(), title);
}
public void printInfoDebug()
{
    Console.WriteLine(ToString());
}

}
}

using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.Text;
namespace BookEntity
{
    [DataContract]
    public class Authors : IEnumerable<Author>
    {
        [DataMember]
        private readonly List<Author> authors = new List<Author>();
        public Authors()
        { }
        public Authors(Author author)
        {
            authors.Add(author);
        }
        public Authors(string fullNames)
        {
            parseAuthors(fullNames);
        }
        private const char authorDelimiter = ':';
        private void parseAuthors(string fullName)
        {
            List<string> splittedAuthorsFullNames = fullName.Split(authorDelim-
            iter).ToList();
            deleteLastAuthor(splittedAuthorsFullNames); //Последний автор всегда пустой.
            Таков формат файла.
            foreach (string authorName in splittedAuthorsFullNames)
            {
                Author author = new Author(authorName);
                authors.Add(author);
            }
        }
    }
}

```

```

    }
    }
    private static void deleteLastAuthor(ICollection splittedAuthorsFullNames)
    {
        splittedAuthorsFullNames.RemoveAt( splittedAuthorsFullNames.Count - 1 );
    }
    public IEnumerator<Author> GetEnumerator()
    {
        return authors.GetEnumerator();
    }
    IEnumerator IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }
    public override string ToString()
    {
        return getAllAuthorsString();
    }
    public string getAuthors()
    {
        return getAllAuthorsString();
    }
    private string getAllAuthorsString()
    {
        StringBuilder authorsStringBuilder = new StringBuilder();
        StringBuilder authorFullName;
        for (int i = 0; i < authors.Count - 1; i++)
        {
            authorFullName = getAuthorFullNameStringBuilder(authors[i]);
            authorsStringBuilder.Append(authorFullName);
            authorsStringBuilder.Append(", ");
        }
        authorFullName = getAuthorFullNameStringBuilder(authors[authors.Count - 1]);
        authorsStringBuilder.Append(authorFullName);
        return authorsStringBuilder.ToString();
    }
    private StringBuilder getAuthorFullNameStringBuilder(Author author)
    {
        StringBuilder authorsStringBuilder = new StringBuilder();
        authorsStringBuilder.Append(author.surname);
        authorsStringBuilder.Append(' ');
        authorsStringBuilder.Append(author.name);
        authorsStringBuilder.Append(' ');
        authorsStringBuilder.Append(author.middleName);
    }

```

```

return authorsStringBuilder;
}
public void printAuthorsToConsoleDebug()
{
    foreach (var author in authors)
    {
        Console.WriteLine("{0} {1} {2}", author.surname, author.name, author.middle-
            Name);
    }
}
public Author this[int i]
{
    get { return authors[i]; }
    set { authors[i] = value; }
}
}
}

using System;
using System.Runtime.Serialization;
namespace BookEntity
{
    [DataContract]
    public class Author
    {
        [DataContract]
        enum Name
        {
            [EnumMember]
            LastName,
            [EnumMember]
            FirstName,
            [EnumMember]
            MiddleName
        };
        [DataMember]
        public string surname { get; private set; }    //фамилия
        [DataMember]
        public string name { get; private set; }        //имя
        [DataMember]
        public string middleName { get; private set; } //отчество
        [DataMember]
        public string fullName { get; set; }
        public Author() { }
    }
}

```

```

public Author(string fullName)
{
    tryParseFullName(fullName);
}
private const char nameDelimiter = ',';
private void tryParseFullName(string fullName)
{
    string[] splittedFullName = fullName.Split(nameDelimiter);
    try
    {
        setNameSurnameAndMiddlename(splittedFullName);
        this.fullName = surname + ' ' + name + ' ' + middleName;
    }
    catch (Exception e)
    { }
}
private void setNameSurnameAndMiddlename(string[] splittedFullName)
{
    surname = splittedFullName[(int) Name.LastName];
    name = splittedFullName[(int) Name.FirstName];
    middleName = splittedFullName[(int) Name.MiddleName];
}
}
}
}

```

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.Serialization;
using System.Text;
using GenresList;
namespace BookEntity
{
    [DataContract]
    public class Genres : IEnumerable<int>
    {
        [DataMember]
        public List<int> genresIDs { get; set; }
        public Genres()
        {
            genresIDs = new List<int>();
        }
        public Genres(int genreID) : this()
    }
}

```

```

{
genresIDs.Add(genreID);
}
public Genres(string genresString) : this()
{
parseGenresString(genresString);
}
private void parseGenresString(string genresString)
{
List<string> genresStringSplitted = splitGenresString(genresString);
deleteLastEmptyGenre(genresStringSplitted);

addGenresByNames(genresStringSplitted);
}
private static List<string> splitGenresString(string genresString)
{
const char genresDelimiter = ':';
return new List<string>(genresString.Split(genresDelimiter));
}
private static void deleteLastEmptyGenre(ICollection genresStringSplitted)
{
int lastElement = genresStringSplitted.Count - 1;
genresStringSplitted.RemoveAt(lastElement);
}
private void addGenresByNames(IEnumerable<string> genresStringSplitted)
{
foreach (var genreString in genresStringSplitted)
{
int genreID = GenresList.GenresList.getGenreID(genreString);
if (genreID != GenresList.GenresList.notFoundID)
genresIDs.Add(genreID);
}
}
public string getGenres()
{
StringBuilder genresStringBuilder = new StringBuilder();
if (genresIDs.Count != 0)
{
for (int i = 0; i < genresIDs.Count - 1; i++)
{
genresStringBuilder.Append(GenresList.GenresList.getGenreDescription(genresIDs[i]));
genresStringBuilder.Append(", ");
}
}
}

```

```

    } genresStringBuilder.Append(GenresList.GenresList.getGenreDescription(genresIDs[genresIDs.Count - 1]));
    }
    return genresStringBuilder.ToString();
    }
    public string getGenresFromGenresList(List<GenresListEntity> genresList)
    {
        StringBuilder genresStringBuilder = new StringBuilder();
        if (genresIDs.Count != 0)
        {
            for (int i = 0; i < genresIDs.Count - 1; i++)
            {
                genresStringBuilder.Append(genresList[genresIDs[i]].descriptiongenresStringBuilder.Append(", ");
            }
            genresStringBuilder.Append(GenresList.GenresList.getGenreDescription(genresIDs[genresIDs.Count - 1]));
        }
        return genresStringBuilder.ToString();
    }
    public void printGenresDebug()
    {
        foreach (var id in genresIDs)
        {
            Console.WriteLine("{0} {1} {2}", id, GenresList.GenresList.getGenreName(id), GenresList.GenresList.getGenreDescription(id));
        }
    }
    public IEnumerator<int> GetEnumerator()
    {
        return genresIDs.GetEnumerator();
    }
    IEnumerator IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }
    public override string ToString()
    {
        return getGenres();
    }
    }
    }
    }

```

```

using System;
using System.IO;
using System.IO.Compression;
namespace BookExtractor
{
    public static class BookExtractor
    {
        private static string booksStoragePath;    //Folder path with books archives
        public static void initialize(string booksStoragePathToInitialize)
        {
            booksStoragePath = booksStoragePathToInitialize;
        }
        public static Stream extract(BookEntity.BookEntity book)
        {
            string archivePath = getArchivePath(book.archiveName);

            ZipArchive archive;
            try
            {
                archive = ZipFile.OpenRead(archivePath);
            }
            catch (Exception)
            {
                Console.WriteLine("ERROR: something went wrong while trying to extract a
book {0}", book);
                Console.WriteLine("Please, stop the server resetting settings and restart.");
                return Stream.Null;
            }
            var bookEntry = archive.GetEntry(book.fileName.appendExtension(book.extension));
            return bookEntry.Open();
        }
        private static string getArchivePath(string archiveName)
        {
            return Path.Combine(booksStoragePath, archiveName);
        }
        private static string appendExtension(this string fileName, string extension)
        {
            return Path.ChangeExtension(fileName, extension);
        }
    }
}

```