Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
«Библиотека»
по дисциплине
«Системное программное обеспечение вычислительных машин»

Выполнил:
студент гр. 550504
Оверченко А.С.

Руководитель:
Костенич А.М.

Минск 2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Программное обеспечение (ПО) каталогизации библиотеки можно отнести к нескольким категориям, а именно:

- каталог;
- автоматизация управления.

Оно упрощает работу обычных библиотекарей и автоматизирует ее. Также оно позволяет посмотреть обычному читателю список литературы, который у него на данный момент на руках и поможет найти книгу по автору, названию или категории. Это не может полноценно заменить библиотекаря, но однозначно уменьшает время обслуживания одного читателя, упрощает формировать отчетность по конкретному читателю или по библиотеке в целом.

В дальнейшем данное ПО можно легко доработать, поскольку в данном приложении используется несколько независимых баз данных, и обновление или добавление информации в одну из них приведет к тому, что информация автоматически обновится в других, т. к. для их связи используется система уникальных ключей.

На данный момент существует весьма скудный выбор ПО, который используется для этих целей. И это не позволяет работникам библиотеки выбирать, какое ПО более удобно в использовании в следствии его слабой распространенности.

Личная заинтересованность в разработке заключается в том, что я сам хожу в городские библиотеки, и крайне неудобно стоять в очереди с целью поиска какой-либо конкретной книги или просто какой-либо интересной книги из какой-то конкретной категории. К тому же когда есть немного времени и тебе нужно всего узнать, что конкретно в тебя на руках (т. е. проверить реально ли всё, что ты взял, ты знаешь, где искать) и сверить с тем, что лежит у тебя дома, например.

Исходя их всех перечисленных выше аргументов можно сделать вывод о том, что данное направление является перспективным в развитии и тема до сих пор актуальна.

1 Обзор источников

Для разработки данного проекта нам понадобятся следующие паттерны проектирования: MVC, Singleton, DAO, Repository.

В качестве основы приложения будем использовать паттерн MVC. Паттерн MVC (модель-вид-контроллер) — составной паттерн, который представляет собой разделение труда:

- модель — это та часть приложения, в которую заложена вся бизнес-логика (объект приложения)[1]. Именно она будет заниматься соединением с базой данных и предоставит общий интерфейс;
- вид — это та часть приложения, которая будет отвечать за представление данных модели[2]. Именно благодаря этой части можно будет увидеть доступные операции и результаты их выполнения;
- контроллер — это связующий компонент модели и вида. Он позволит соединять вид с моделью и передавать данные между ними.

У данного паттерна присутствуют важные преимущества, которые и является причиной его частого использования в промышленной разработке:

- масштабируемость;
- принцип разделения труда.

Начнем с последнего. Принцип разделения труда заключается в том, что каждый компонент паттерна выполняет только свою роль: модель содержит в себе всю логику работы с данными, а вид (представление) только занимается их отображением конечному пользователю.

Отсюда же и вытекает масштабируемость: чтобы сделать новый вид (например, вы захотели добавить мобильный клиент к своему сайту), вам не нужно переписывать или видоизменять модель. Вам нужно будет написать новый вид и сделать контроллер, который позволит использовать имеющуюся модель. Контроллер будет выступать в роли «обертки» интерфейса, предоставляемого моделью, до такого вида, который требует представление.

Также в качестве модели можно использовать и другие паттерны: например, DAO или Repository.

DAO (Data Access Object, объект доступа к данным) — паттерн, который используют в качестве связующего звена между приложением и БД (базой данных), и который предоставляет абстрактный интерфейс к какому-либо типу базы данных или механизму хранения. Иными словами это просто система управления базой данных (СУБД).

Паттерн Repository преследует принципы, схожие с DAO. Но «Respotory представляет собой все объекты определенного типа в виде концептуального множества. Его поведение похоже на поведение коллекции, за исключением более развитых возможностей для построения запросов» [3]. И именно поэтому в Repository явно не используются поля объектов для их идентификации. Это добавляет определенные неудобства с одной стороны:

надо каждый раз создавать экземпляр класса, чтобы записать в его поля требуемые для идентификации данные. Но если вдуматься, то, в отличии от DAO, не будет нужно создавать несколько методов для фильтрации или добавления объектов по разным свойствам (полям). Это и будет являться плюсом данного подхода. Иными словами, Repository в отличие от DAO позволяет создать более простой, понятный и гибкий интерфейс для работы с данными или базами данных.

Также в приложении предполагается использование паттерна Singleton (одиночка). Особенность данного паттерна заключается в том, что у нас в течении работы всей программы предполагается не более одного экземпляра класса, для которого применен данный паттерн[1]. Но зачем это вообще может понадобиться?

Singleton используют тогда, когда не нужно каждый раз создавать экземпляр какого-либо класса или нужно иметь доступ к какому-либо объекту из различных частей программы. Но ведь можно создать глобальную переменную, зачем де все-таки это нужно?

1) Глобальные переменные не является примером хорошего тона программиста, т. к. могут возникнуть коллизии имен.

2) Не во всех языках программирования присутствуют глобальные переменные

3) Глобальную переменную крайне сложно контролировать (т. к. обычно хранят не саму переменную (экземпляр класса), а ссылку или адрес на нее.

К тому же Singleton можно использовать для файлов конфигурации, т. к. они должны быть загружены в программу один раз, и все могут использовать их в режиме чтения. Аналогичное верно и для системы логирования (системы, ведущей записи, пометки о работе программы).

Таким образом, в данном разделе были выделены основные особенности и пути разработки данного ПО, которые в дальнейшем будут реализованы.

2 Системное проектирование

В данном разделе разобьем программу на модули и опишем сторонние библиотеки, используемые для разработки.

Для реализации паттерна MVC нам понадобится три модуля: представление (GUI, графический интерфейс пользователя), контроллер (GUI_controller) и модель. Поскольку модель реализует паттерн Repository, она должна выступать в роли контроллера базы данных (СУБД контроллер).

Модули, которые взаимодействуют с базой данных, должны сохранять историю своих запросов и результатов: успешен был запрос или нет. Для этого нужно вынести отдельно модуль логирования, реализованного с помощью паттерна Singleton.

Также требуется, чтобы система могла идентифицировать каждого клиента. Для этого потребуется создание отдельного окна в графическом интерфейсе. Но это окно будет использовать очень схожие и пересекающиеся с основным модулем GUI, поэтому выносить это окно в отдельный модуль не имеет смысла.

При запуске программы должно запускаться окно авторизации, в котором пользователь может ввести свои данные для его идентификации: логин и пароль. Эти данные через GUI_controller отправляются в модель. Модель проверяет наличие учетной записи в базе данных и возвращает ответ. Если были введены некорректные учетные данные, пользователь получает соответствующее уведомление. При успешной авторизации открывается основное окно программы, которое содержит только доступные для пользователя функции.

Все классы текущего представления для соединения с базой данных будет использовать контроллер (GUI_controller). Поскольку количество доступных функций контроллера будет невелико и одни и те же функции будут использоваться в разных классах представления, реализуем класс GUI_controller с использованием паттерна Singleton. Это позволит получить выигрыш в читабельности кода, и он не будет дублирован.

В качестве библиотеки для создания графического интерфейса будет использован Qt Framework версии 5.7.1 32bit. Данная версия является одной из самых последних, и позволит использовать все возможности данной библиотеки.

Поскольку в качестве языка программирования, на котором будет написан проект, заявлен C++, для графического интерфейса будет использованы именно компоненты QWidget, которые позволяют сделать это средствами требуемого языка без использования других языков программирования.

В качестве базы данных будет использоваться MongoDB, которая является объектно-документарной базой данных. Идеологически это некий

симбиоз между привычной реляционной БД и key-value хранилищем [4]. Эта база данных использует JSON (JavaScript Object Notation, объектное представление языка JavaScript) и BSON (binary JSON, бинарный JSON) форматы для хранения данных. Это позволяет базе данных хранить объекты (т. е. теперь можно хранить не только строго фиксированные поля, которые задаются шапкой таблицы в реляционных БД), быстро сохранять, искать и передавать их по сети интернет с минимальными затратами на сервере. Хоть данная база данных и является достаточно молодой, она очень быстро набирает популярность. Также данная БД позволяет получать данные без использования SQL (structured query language, язык структурированных запросов) запросов, поскольку сама является NoSQL (Not only SQL, не только SQL).

MongoDB имеет драйвер для различных языков программирования (в том числе под C и C++), что позволяет использовать ее без изучения технологий общения с сервером (REST), а сразу использовать функции создания, добавления, чтения, обновления и удаления информации (стандартные CRUD (create-read-update-delete) операции). Данный драйвер будет использоваться в модели для соединения с базой данных.

Для того, чтобы обезопасить систему от взлома, пароли не должны храниться в не измененном виде. Поэтому будут храниться результаты выполнения хэш-функции. Хэш-функции – это функции, предназначенные для «сжатия» произвольного сообщения или набора данных, записанных, как правило, в двоичном алфавите, в некоторую битовую комбинацию фиксированной длины, называемую сверткой [5]. Кажется, что хранение результата хэш-функции увеличивает шансы хакера на получение пароля, но это не так. Основная особенность современных хэш-функций, используемых в криптографии, заключается в том, что:

1) длина результата выполнения хэш-функции имеет фиксированную длину;

2) при небольшом изменении входных данных результат выполнения хэш-функции очень сильно изменяется.

Именно эти свойства положены в основу алгоритмов семейства SHA (Secure Hash Algorithm, безопасный алгоритм хеширования). Последним стандартом данного семейства является алгоритм SHA-3 Keccak [6].

Для шифрования будем использовать библиотеку [7], в которой реализована эта хэш-функция. Также данная библиотека позволяет изменять длину выходного слова, что добавляет гибкости в ее использовании. Длина выходного слова не обязательно должна соответствовать стандарту (224, 254, 384 или 512 бит).

Таким образом, вариант структурной схемы, описанной в данном разделе, можно наблюдать в приложении А.

3 Функциональное проектирование

В данном разделе будет рассмотрены основные функциональные составляющие проектируемого ПО. Диаграмма классов продемонстрирована в приложении Б.

Для каждого структурного блока предусмотрены один или несколько классов. Также присутствует система классов, которая будет использована для общения между структурными блоками.

У каждого класса присутствует набор методов, которые устанавливает значение одноименных полей – набор set* методов (например, void setName()) и набор методов, которые возвращают копии значений, хранящихся в одноименных полях — набор get* методов (например, String getName() const). Описание данных методов будет опущено.

Также у большинства классов имеется специализированный конструктор вида explicit <имя_класса>(const DB_ID& id = DB_ID()), который предназначен для создания экземпляра класса с заранее известным идентификатором (например, при получении информации из базы данных). Данный конструктор может быть использован без параметров, что будет означать автоматическую выдачу уникального идентификатора. Его описание и наличие в классах будет опущено

DBConnector

Данный класс описывает способы обмена информации с базой данных. Поля класса:

AJIOB::ConfigClass config — поле, которое содержит все конфигурации для обмена данными с базой данных;

mongocxx::pool pool — соединение с базой данных;

bool isAuthorized — поле, хранящее значение true, если пользователь авторизован, иначе хранящее false;

UserPriveleges privelege – поле, хранящее уровень доступа авторизованного пользователя. Если пользователь не авторизован, хранится значение UserPriveleges::none.

Конструктор класса:

DBConnector(String connectTo = "localhost") — в конструктор передается название поля из файла конфигурации, значение которого будет считаться за URI (Uniform Resource Identifier, унифицированный идентификатор ресурса) для соединения с базой данных. В файле конфигураций должна быть запись вида uri_<значение параметра connectTo>=<значение>.

Методы класса:

bsoncxx::types::value Add(const std::string& collectionName, const

bsoncxx::document::view_or_value& view) — приватный метод, который добавляет в коллекцию с именем collectionName запись из view. Возвращает информацию о добавлении;

void CheckAuth() const — приватный метод, который проверяет авторизацию пользователя и, если пользователь не авторизован, генерирует исключение вида NoAuthException;

static std::chrono::system_clock::time_point getCurrentTimePoint() — приватный статический метод, который возвращает текущее время;

bool isItBookCopyID(DB_ID idToCheck) — приватный метод, который проверяет, является ли idToCkeck идентификатором какого-либо экземпляра какого-либо издания. Если является — то метод возвращает true, иначе false;

User Authorize(const String& login, const String& password) — метод, который возвращает пользователя по логину login и паролю password. Если авторизация прошла неудачно, метод генерирует исключение типа NoAuthException;

User LoginAsGuest() — метод, который авторизует пользователя как гостя и возвращает соответствующую информацию;

DB_ID GiveOutBook(BookCopy& bookCopy, User& user) — метод, который выдает экземпляр bookCopy пользователю user и возвращает идентификатор операции;

void RenewBookTime(BookCopy& bookCopy) — метод, который продлевает экземпляр bookCopy;

void ReturnBookCopy(BookCopy& bookCopy) — метод, который возвращает экземпляр bookCopy;

void ArchieveBookCopy(BookCopy& bookCopy) — метод, который архивирует экземпляр bookCopy;

bool isCopyGettedOut(BookCopy& bookCopy) — метод возвращает true, если экземпляр bookCopy выдана читателю, иначе возвращает false;

bool isCopyArchieved(const BookCopy& bookCopy) в — метод возвращает true, если экземпляр bookCopy находится в архиве, иначе возвращает false;

Остальные методы данного класса подходят под один из следующих видов:

void Add(<класс>& <имя экземпляра>);

void Get(std::list<<класс>>& <имя группы экземпляров>, const bsoncxx::document::view_or_value& filter = bsoncxx::builder::stream::document {}.view());

void Update(<класс>& <имя экземпляра>);

void Delete(<класс>& <имя экземпляра>).

Каждый их этих типов имеет общее назначение, поэтому далее будут описаны только эти шаблоны и указаны, для каких классов каждый из них нужно будет определить.

Таблица 3.1 — Описание типовых методов класса DBConnector

Имя шаблона	Для каких классов определен	Предназначение
Add	User, Book, Author	Метод, который добавляет элемент <имя экземпляра> в базу данных.
Get	User (принимает два фильтра: для информации о данных авторизиции пользователя и для персональной информации), Book, Author, Transfer	Метод, который получает элементы типа <класс>, которые соответствуют фильтру filter из базы данных и записывают результат в <имя группы экземпляров>.
Update	User, Book, Author	Метод, который обновляет элемент <имя экземпляра> в базе данных на текущее значение <имя экземпляра>.
Delete	User, Book, Author	Метод, который удаляет элемент <имя экземпляра> из базы данных.

Author

Этот класс, который хранит всю информацию об авторе. Поля класса:
DB_ID _id — уникальный идентификатор автора;
String name — имя автора;
String surname — фамилия автора;
String fatherName — отчество автора.
Методы класса (без get* и set* методов):
bool operator==(const Author& that) const – данный метод является перегрузкой оператора ==, используемого для сравнения двух объектов класса Author. Возвращает значение true, если объекты эквивалентны, иначе возвращает false.

Book

Этот класс, который хранит всю информацию о книге. Поля класса:
DB_ID _id — уникальный идентификатор издания;
ISBNClass ISBN — ISBN номер (International Standard Book Number, международный стандартный книжный номер) издания;
std::list<Author> authors — список авторов книги;
String name — название книги;
li year — год издания книги;

uli pageCount — количество страниц в издании;

std::list<BookCopy> copies — список экземпляров, которые числятся в библиотеке.

Конструктор класса:

explicit Book(const DB_ID& id = DB_ID(), const uli& copiesNum = 0) — предназначен для создания экземпляра класса с заранее известным идентификатором (например, при получении информации из базы данных) и количеством экземпляров издания. Данный конструктор может быть использован без параметров, что будет означать автоматическую выдачу уникального идентификатора и установлению количества копий на значение, равное нулю.

Методы класса (без get* и set* методов):

uli GetNumOfAllCopies() const — метод, который возвращает количество экземпляров, которые числятся в библиотеке;

usi GetNumOfCopiesInLibrary() — метод, который возвращает количество экземпляров, которые не выданы на руки;

void addCopy(const BookCopy& copy) — метод, который добавляет определенный экземпляр издания к уже имеющимся;

std::string getAuthorsAsString() const — метод, который возвращает список авторов данного издания в виде строки;

static std::string getAuthorsAsString(const std::list<Author>& authors) — метод, который возвращает список авторов authors, которые переданы в виде параметра, в виде строки. Данный метод может быть использован без конкретного экземпляра класса.

BookCopy

Данный класс является конкретной копией какого-либо издания. Поля класса:

DB_ID _id — уникальный идентификатор издания;

bool isArchieved — поле, хранящее информацию о том, находится ли данный экземпляр в архиве (true) или нет (false);

bool isGettedOut — поле, хранящее информацию о том, выдан ли этот экземпляр на руки (true) или его можно взять (false).

Transfer

Данный класс описывает все операции выдачи с экземплярами. Поля класса:

DB_ID cl_ID — уникальный идентификатор операции;

DB_ID cl_copyID — уникальный идентификатор экземпляра издания;

DB_ID cl_userID — уникальный идентификатор пользователя;

Date cl_firstGetDate — дата выдачи экземпляра на руки;

Date cl_lastContinueDate — дата последнего продления экземпляра;

Date cl_returnDate — дата возврата экземпляра в библиотеку.

Конструктор класса:

Transfer(const DB_ID& id, const DB_ID& copyId, const DB_ID& userId, const Date& firstGetDate, const Date& lastContinueDate = Date(), const Date& returnDate = Date()) — конструктор, инициализирующий все поля класса. Единственный способ установить значение полей объекта. Сделано, чтобы нельзя было случайно или специально поменять значения полей.

Методы класса (без get* и set* методов):

bool operator<(const Transfer& that) const — метод, перегружающий оператор сравнения «меньше». Может быть использован при создании контейнера типа ключ-значение (std::map, std::multimap) в качестве ключа;

bool operator>(const Transfer& that) const — метод, перегружающий оператор сравнения «больше». Может быть использован при создании контейнера типа ключ-значение (std::map, std::multimap) в качестве ключа.

User

Класс является описанием некоторого пользователя, зарегистрированного в системе или подлежащего регистрации. Поля класса:

DB_ID _id — уникальный идентификатор пользователя;

String login — уникальный логин пользователя;

String cryptedPassword — пароль пользователя, хранящийся в зашифрованном виде;

UserPriveleges privelege — уровень привилегий пользователя (гость, пользователь, администратор);

UserPersonalInfo personalInfo — персональная информация о пользователе.

Конструктор класса:

explicit User(const DB_ID& id = DB_ID(), const String& cryptedPassword = "") — конструктор, который позволяет установить значение зашифрованного пароля и идентификатор пользователя.

Методы класса (без get* и set* методов):

bool isPasswordCorrect(const String& password) const — метод проверяет корректность пароля и возвращает true, если пароль, который хранится в зашифрованном виде, совпадает с паролем password, иначе возвращает false;

bool setPassword(const String& newPassword, const String& oldPassword = "") — метод, который устанавливает значение пароля на newPassword, если текущий пароль совпадает с oldPassword. В этом случае функция вернет true, иначе функция вернет false и не изменит текущий пароль;

void resetPassword() — метод сбрасывает пароль на значение по

умолчанию Пароль по умолчанию всегда совпадает с логином.

UserPersonalInfo

Данный класс описывает персональные данные о пользователе, которые библиотека знает о читателе. В данном классе будут описаны только поля, потому что все методы в данном классе — это get* и set* функции.

String name — имя пользователя;

String surname — фамилия пользователя;

String fatherName — отчество пользователя;

String passportNumber — номер паспорта пользователя;

Другое

В данном разделе будут описаны некоторые особенности разработки и расшифровка некоторых типов.

String – тип, который является обычной строкой: std::string. Добавлено с целью увеличения читабельности кода и ускорения разработки.

uli – тип, который является обычным беззнаковым четырехбайтным целым числом: unsigned long int. Добавлено с целью увеличения читабельности кода и ускорения разработки.

usi, ulli, ui, – аналогичные типы. Расшифровываются как unsigned short int, unsigned long long int, unsigned int, std::string соответственно.

Таким образом, в данном разделе была описана модель и классы, с помощью которых можно с этой моделью взаимодействовать. Это позволит разрабатывать графический, консольный и другой и любой другой интерфейс для этой модели.

4 Разработка программных модулей

В данном разделе будут рассмотрены алгоритмы, используемые в разрабатываемом ПО.

1 Конвертирование ISBN номера из строки в специальный класс

Данный алгоритм реализован в методе ParseConsoleString класса ISBNClass. Метод принимает константную ссылку на строку (std::string) с именем str. Опишем его по шагам.

Шаг 1. Инициализируем константы, которые хранят возможную длину принимаемой как параметр строки str, в которой располагается ISBN номер.

Шаг 2. Разделяем строку на массив строк по разделительному символу '-' и записываем его в переменную vectorOfStrings.

Шаг 3. Создаем временную копию buffer класса ISBNClass, чтобы случайно не повредить данные в текущем экземпляре. Далее работаем именно с ним, если не оговорено иное.

Шаг 4. Проверяем длину строки str. Устанавливаем значение поля cl_type переменной buffer в соответствии с инициализированными на шаге 1 константами. Если cl_type будет свидетельствовать о том, что передан ISBN13 номер, идем к шагу 6. Если cl_type не соответствует ни одной константе, идем к шагу 13.

Шаг 5. Добавляем в начало массива vectorOfStrings пустую строку, чтобы нормально определились позиции

Шаг 6. Если количество элементов массива vectorOfStrings не соответствует требуемому (а именно четырем), идем к шагу 13.

Шаг 7. Запоминаем каждую строку из vectorOfStrings в соответствующий элемент массива cl_fields (cl_fields является полем класса ISBNClass). Если одно из полей вернуло, что операция завершена некорректно, идем к шагу 13.

Шаг 8. Рассчитываем требуемую контрольную цифру и записываем ее в переменную controlNumber.

Шаг 9. Переводим controlNumber в символ этой контрольной цифры (числу 10 в ISBN10 соответствует большой латинский символ 'X').

Шаг 10. Проверяем контрольную цифру на корректность: сравниваем рассчитанную и полученную из строку контрольную цифру. Если они не совпали, идем к шагу 13.

Шаг 11. Переписываем содержимое буфера в вводимый объект, т. к. всё введено правильно.

Шаг 12. Возвращаем результат удачного завершения операции (true). Идем к шагу 14.

Шаг 13. Возвращаем результат неудачного завершения операции (false).

Шаг 14. Завершение алгоритма.

2 Конвертирование ISBN части номера из строки в специальный класс

Данный алгоритм реализован в методе SetString класса ISBNOnePart. Метод принимает константную ссылку на строку (std::string) с именем num. Опишем его по шагам.

Шаг 1. Создаем временную копию buffer класса ISBNOnePart, чтобы случайно не повредить данные в текущем экземпляре. Далее работаем именно с ним, если не оговорено иное.

Шаг 2. Создаем переменную i, которая будет считать итерации цикла, и инициализируем его нулем.

Шаг 3. Если длина num больше или равно i, идем к шагу 9.

Шаг 4. Если элемент не является цифрой, идем к шагу *.

Шаг 5. Умножаем поле ISBNPart на 10 (сдвигаем влево на одну позицию).

Шаг 6. Прибавляем к имеющему числу в ISBNPart цифру, которая записана в строке num на позиции i.

Шаг 7. Увеличиваем встроенный счетчик длины символов numOfElements.

Шаг 8. Увеличиваем счетчик i на единицу. Идем к шагу 3.

Шаг 9. Переписываем buffer в текущий элемент.

Шаг 10. Возвращаем результат удачного завершения операции (true). Идем к шагу 12.

Шаг 11. Возвращаем результат неудачного завершения операции (false).

Шаг 12. Завершение алгоритма.

3 Добавление пользователя в базу данных

Данный алгоритм реализован в методе Add(User& user) класса DBConnector. Метод принимает пользователя для добавления (user). Данный алгоритм описан с помощью блок-схемы (приложение В, лист 1).

4 Выдача издания пользователю

Данный алгоритм реализован в методе GiveOutBook класса DBConnector. Метод принимает копию издания для выдачи (bookCopy) и пользователя (user), которому выдача будет произведена. Данный алгоритм описан с помощью блок-схемы (приложение В, лист 2).

Таким образом в данном разделе были рассмотрены основные алгоритмы, применяемые в данном курсовом проекте.

5 Руководство пользователя

В данном разделе приведена вся информация, которая должна публиковаться вместе с продуктом.

1 Минимальные системные требования

- ОС Windows 7, 8, 8.1, 10 любой разрядности (x32 либо x64);
- ОЗУ: 1 ГБ;
- клавиатура.

2 Рекомендуемые системные требования

- ОС Windows 7, 8, 8.1, 10 любой разрядности (x32 либо x64);
- ОЗУ: 2 ГБ и более;
- клавиатура, мышь.

3 Комплект поставляемого ПО

В комплекте с поставляемым программным обеспечением (далее — ПО) должны быть:

- исполняемый файл “LibraryAppSetup.exe”;
- исполняемый файл “vcredist_x86.exe”.

При отсутствии каких-либо файлов, пожалуйста, обратитесь к поставщику данной копии ПО либо свяжитесь с производителем по адресу <aleskandr9809@gmail.com>.

4 Первый запуск программы

Для запуска программы требуется:

1) проверить комплект поставляемого ПО;

2) запустить файл “LibraryAppSetup.exe”. Следовать инструкциям установщика;

3) запустить файл “vcredist_x86.exe”. Следовать инструкциям установщика;

4) запустить файл “LibraryApp” или одноименный ярлык с рабочего стола (см. рисунок 5.1);

5) Если после запуска вы увидели окно с текстом, отличным от рисунка 5.2, пожалуйста свяжитесь с поставщиком данного ПО.

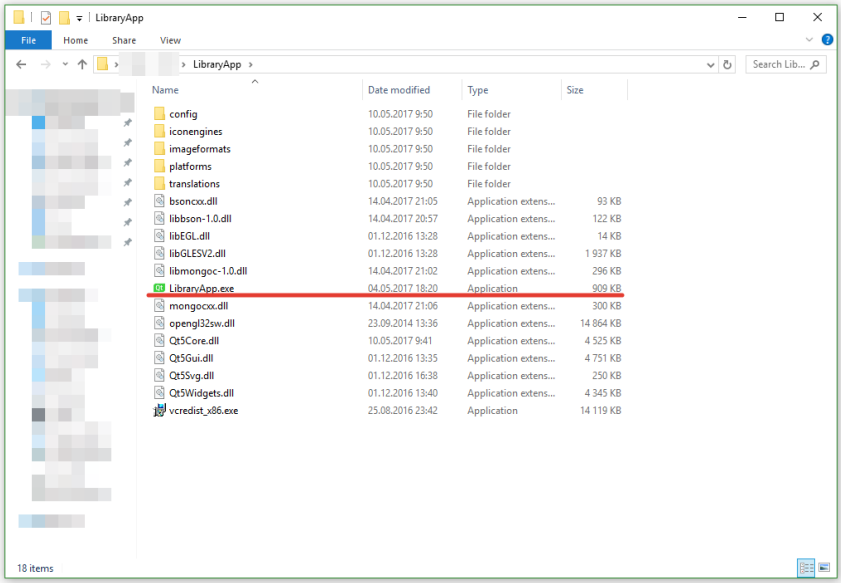

Рисунок 5.1 — Папка с программой

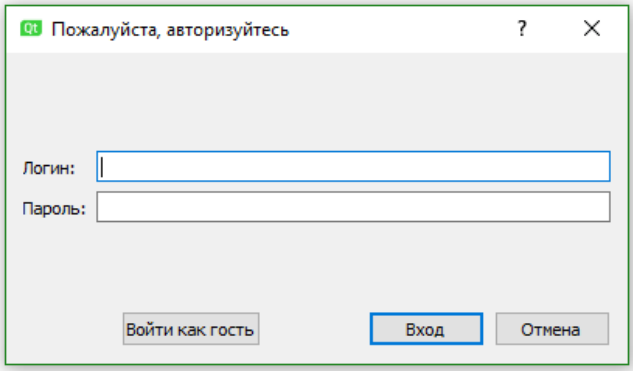

Рисунок 5.2 — Форма авторизации

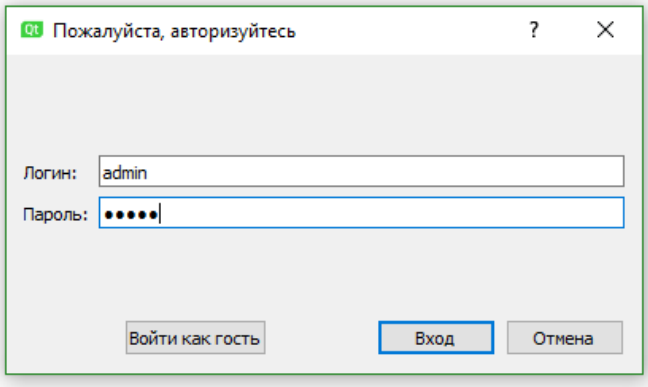

Рисунок 5.3 — Пример ввода логина и пароля

6) Если вы работник и запускаете программу впервые, в качестве логина и пароля по умолчанию введите «admin» (без кавычек). Если вы являетесь читателем, введите логин и пароль, предоставленный работником библиотеки. Пример ввода приведен на рисунке 5.3.

Внимание! При вводе любого пароля отображаются символы '*'. Так и должно быть.

Внимание! Настоятельно рекомендуется сменить пароль для этого пользователя на какой-либо более защищенный.

7) Если вы получили уведомление, как на рисунке 5.4, или окно, не схожее с рисунком 5.5, 5.6 или 5.7, пожалуйста, обратитесь к поставщику ПО.

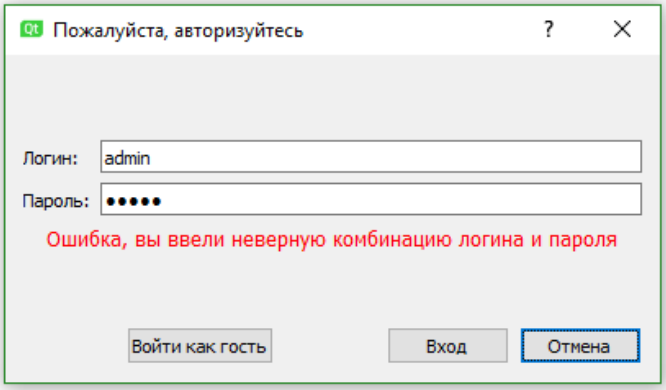

Рисунок 5.4 — Ошибочный ввод логина и пароля

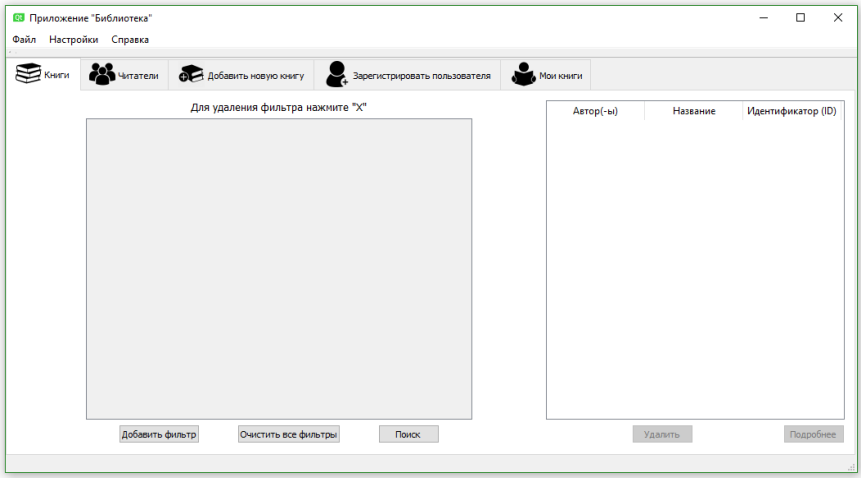

Рисунок 5.5 — Главное меню для работника

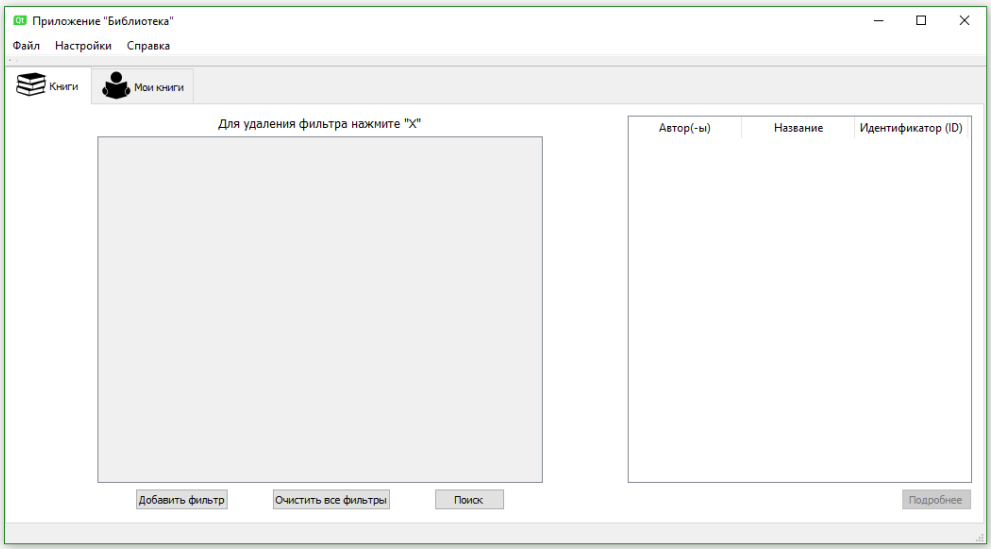

Рисунок 5.6 — Главное меню для читателя

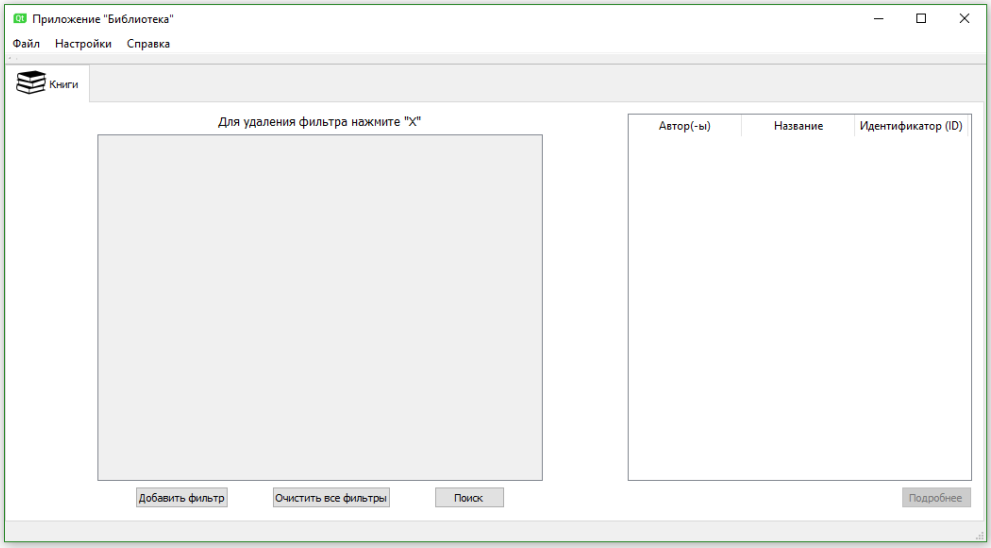

Рисунок 5.7 — Главное меню для гостя

5 Как пользоваться программой

В данном пункте будут показаны некоторые примеры работы с программой.

1) Запустите программу из папки или используя ярлык;

2) Авторизуйтесь (введите правильные логин и пароль);

3) Выберите операцию, которую вы хотите сделать;

4) Введите данные, которые требуются для корректного выполнения операции, и подтвердите их;

5) Поздравляем, действие выполнено.

6 «ЧаВо?» (Часто задаваемые вопросы)

В данном разделе описаны часто задаваемые вопросы (В — вопрос, О — ответ).

В: Программа выдает ошибку: «Ошибка, вы ввели несуществующую комбинацию логина и пароля». Что делать?

О: Возможно, вам сбросили пароль на версию «по умолчанию». В графы «Логин» и «Пароль» введите свой логин. Не забудьте сменить пароль. Если это не помогло, обратитесь к администратору ПО;

В: Были забыты/потеряны все пароли от аккаунта администраторов (или вовсе были удалены аккаунты). Что делать?

О: В этом случае рекомендуется еще раз попробовать вспомнить хоть один пароль от аккаунта администратора. Если это не помогло, напишите об этом распространителю или производителю, и они вышлют вам стандартную версию баз.

В: При попытке добавления нового издания появляется ошибка «Ошибка распознавания ISBN номера. Пожалуйста, повторите ввод». Что делать?

О: Номер нужно вводить в том формате, в котором он дан в издании с соблюдением всех дефисов. Например, 978-985-533-538-3. Программа поддерживает как ISBN-10, так и ISBN-13 стандарты.

6 Тестирование

В данном разделе будет рассмотрен способ проверки программы на корректность установки, а также примеры работы программы.

Описание действий, которые нужно выполнить, чтобы проверить работоспособность программы после сборки, установки или настройки можно найти в предыдущем разделе в подпункте «Первый запуск программы».

Далее приведены некоторые примеры работы программы и продемонстрирована обработка исключительных ситуаций:

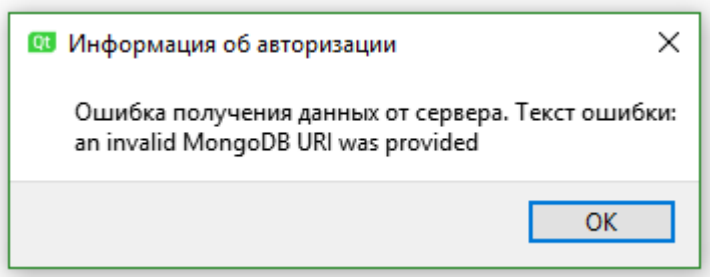

Рисунок 6.1 — Некорректный файл конфигураций

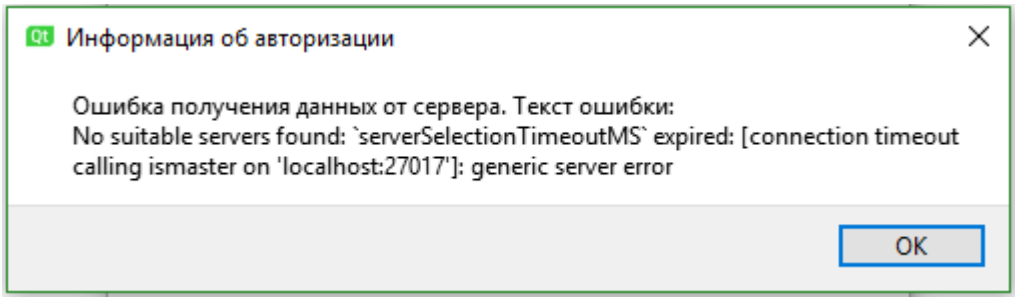

Рисунок 6.2 — Ошибка подключения к серверу

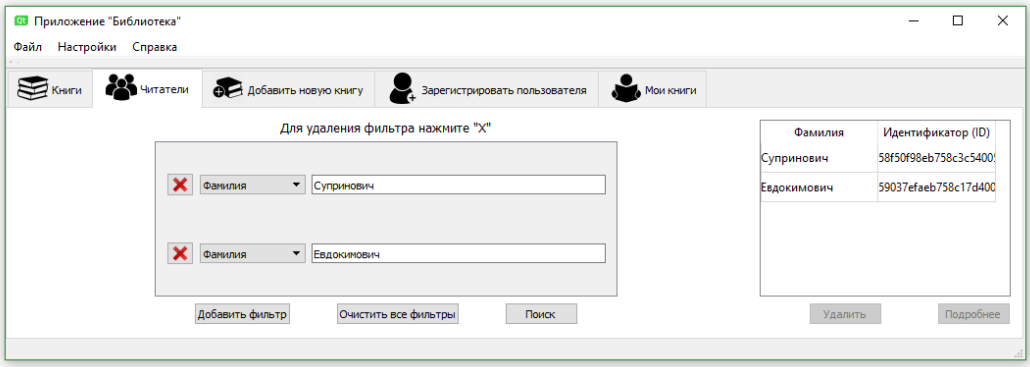

Рисунок 6.3 — Пример выполнения операции «Поиск» по базе данных читателей с использованием фильтров

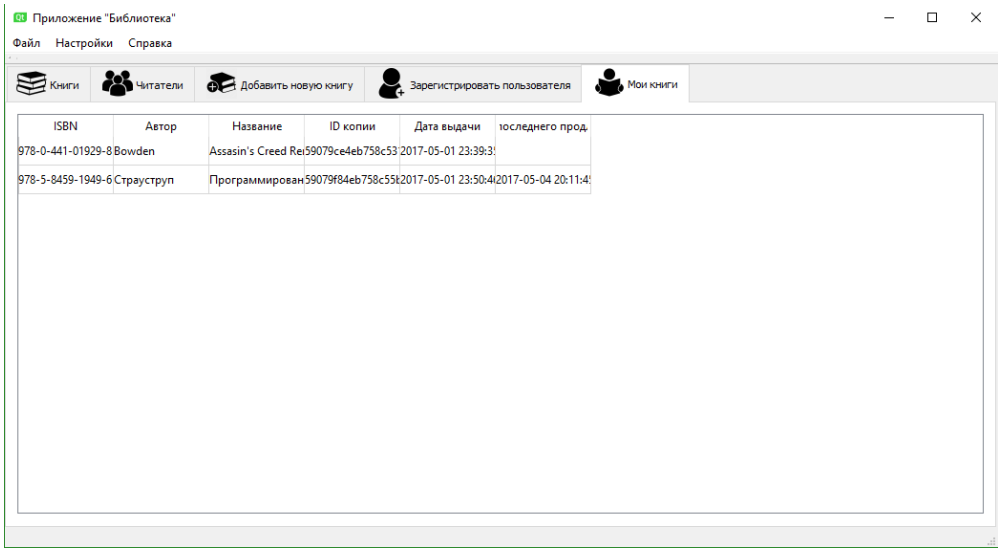

Рисунок 6.4 — Просмотр книг, которые выданы на текущий аккаунт

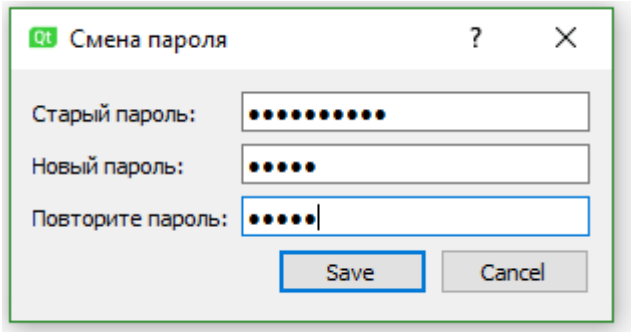

Рисунок 6.5 — Смена пароля текущего пользователя

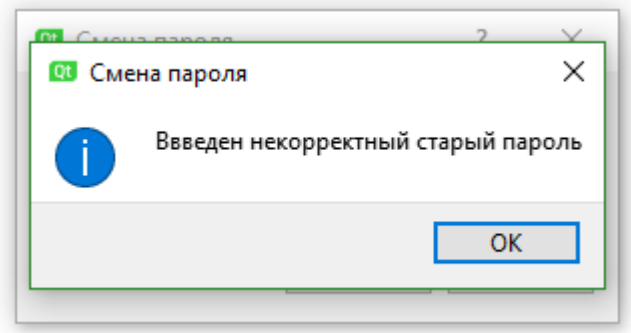

Рисунок 6.6 — При попытки смены пароля был некорректно введен старый пароль

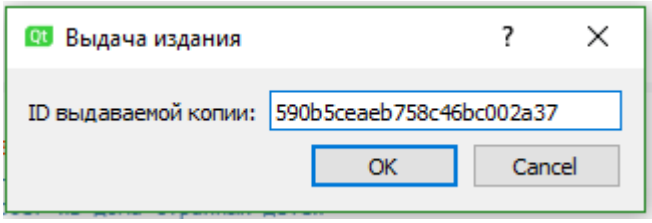

Рисунок 6.7 — Выдача издания

ЗАКЛЮЧЕНИЕ

В данном разделе будут подведены итоги разработки программного обеспечения.

Для реализации данного курсового проекты были использованы:

- язык программирования C++;
- алгоритм шифрования SHA-3 Keccah;
- библиотека Qt;
- база данных MongoDB.

Это позволило реализовать программное обеспечение, которое имеет клиент-серверную архитектуру. Данная архитектура обеспечивает возможность совместного использования базы данных и получения доступа к ней из любой точки мира, где имеется доступ в интернет. Это позволяет пользователям смотреть информацию о книгах, которые им были выданы, где-бы они не находились.

Также стоит заметить, что приложение, разработанное в данном курсовом проекте, является расширяемым, что позволит легко добавлять новые функции в данное приложение, расширяя его функциональность.

Для разработки графической оболочки под другую операционную систему можно будет использовать значительные наработки (систему классов проекта и способ взаимодействия с базой данных) из данного курсового проекта.

Также в ходе и тестирования данного ПО было проверено и практически обосновано, что оно выполняет все требуемые функции и имеет удобный графический интерфейс.

Таким образом, данный программный продукт получил хорошее развитие в процессе написания данного курсового проекта, но также может разрабатываться дальше. Например, с целью получения коммерческой выгоды.

ЛИТЕРАТУРА

1. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб: Питер, 2016. — 366 с.

2. Э. Фримен, Э. Фримен, К. Сьерра, Б. Бейтс. Паттерны проектирования. — СПб: Питер, 2011. — 656 с.

3. Эванс Э. Предметно-ориентированное проектирование (DDD). Структуризация сложных программных систем. – М.: ООО «И.Д. Вильямс». 2010. – 448 с.

4. Habrahabr [Электронный ресурс]. – MongoDB или как разлюбить SQL. – Режим доступа: https://habrahabr.ru/post/74144/

5. Habrahabr [Электронный ресурс]. – Хэш-алгоритмы. – Режим доступа: https://habrahabr.ru/post/93226/

6. The Keccak sponge function family [Электронный ресурс]. – Официальный сайт функции Keccak. – Режим доступа: http://keccak.noekeon.org/

7. Лафоре Р. Объектно-ориентированное программирование в C++/ 4-е издание М.:Питер, 2004. – 923 с.

8. Страуструп, Б. Программирование. Принципы и пратика с испоьзованием C++/Б. Страуструп, второе издание: Пер. с англ. – М.: ООО «И.Д. Вильямс». 2016. – 1328 с.

Приложение А

(обязательное)

Схема структурная (чертеж)

Приложение Б
(обязательное)

Диаграмма классов (чертеж)

Приложение В
(обязательное)

Схема программы

Приложение Г
(обязательное)

Листинг кода