МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

(КАФЕДРА 43)

|  |
| --- |
| КУРСОВОЙ ПРОЕКТ ЗАЩИЩЁН С ОЦЕНКОЙ  РУКОВОДИТЕЛЬ |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Ст.преподаватель |  |  |  |  |  | Е. О. Шумова |
| (должность, учёная степень, звание) |  | (подпись) |  | (дата защиты) |  | (инициалы, фамилия) |

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

ОБСЛУЖИВАНИЕ ЧИТАТЕЛЕЙ В БИБЛИОТЕКЕ

по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

|  |
| --- |
| РАБОТУ ВЫПОЛНИЛ |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 4236 |  |  |  | Л. Мвале |
|  |  |  | подпись, дата |  | инициалы, фамилия |

**СОДЕРЖАНИЕ**

1. ПОСТАНОВКА ЗАДАЧИ……………………………………………….4
   1. Анализ предметной области………………………………………4
      1. Сущности предметной области……………………………..…………..4
      2. Субъекты проекта…………………...………………………5
      3. Словарь предметной области………………………………6
   2. Функционал, который будет разработан…………………………8
   3. Функциональные требования…………………………………....11
2. РАЗРАБОТКА КЛАССОВ…………………………………………………………….…….14
   1. Классы сущностей…………………………………………………………….……….14
   2. Управляющие классы…………………………………………..…………………….19
   3. Интерфейсные классы……………………………………….……………………….22
   4. Используемые паттерны проектирования………………….…………….27
   5. Диаграмма классов……………………………………………………………………..28
3. Разработка приложения…………………………………………………………………..….32
   1. Разработка интерфейса приложения………………………………………….32
   2. Реализация методов классов……………………………………………….…….40
   3. Разработка тестового приложения…………………………………………….46
4. ТЕСТИРОВАНИЕ…………………………………………………………………………..….52
5. ЗАКЛЮЧЕНИЕ…………………………………………………………………….…………….56
6. ПРИЛОЖЕНИЕ С ПОЛНЫМ КОДОМ ПРОГРАММ.………………………57
7. **ПОСТАНОВКА ЗАДАЧИ**
   1. **Анализ предметной области**

Разработка системы для автоматизации обслуживания читателей в библиотеке, которая позволит управлять данными о читателях, книгах, а также процессом выдачи и возврата книг. Система должна обеспечивать удобное взаимодействие пользователей с библиотечными ресурсами, а также учитывать необходимость многозначной зависимости и базовые ООП (объектно-ориентированные принципы).

**1.1.1. Сущности предметной области:**

**1. Читатели:**

* ID – уникальный идентификатор читателя.
* Name – ФИО читателя.
* Birthday – дата рождения читателя.
* Email – электронная почта для связи.
* Address – адрес проживания читателя.
* Password – пароль для доступа к системе.

**2. Книги:**

* ID – уникальный идентификатор книги.
* Writer Name – имя автора книги.
* Title – название книги.
* Publisher – издательство.
* Year of Publication – год издания.
* Total Copies – общее количество экземпляров.
* Available Copies – количество доступных экземпляров для выдачи.

**3. Выдача книг:**

* Reader ID – ID читателя, которому выдана книга.
* Book ID – ID книги, которая выдается.
* Date of Lending – дата выдачи книги.
* Date of Return – дата возврата книги.

**1.1.2. Субъекты проекта**

Субъекты представляют собой пользователей системы или участников, которые взаимодействуют с объектами предметной области.

1. Читатель

* Пользователь системы, который регистрируется, просматривает доступные книги и берет книги на временное пользование.
* Действия:
* Регистрация в системе
* Просмотр списка книг
* Поиск книги
* Получение книги
* Возврат книги
* Купить книг
* Свяжитесь с администратором

1. Библиотекарь

* Администратор системы, который управляет книгами и читателями.
* Действия:
* Добавление и удаление книг
* Просмотр и удаление читателей
* Регистрация выдачи и возврата книг
* Поиск книг и читателей
* Очистка данных
* Общайтесь с читателями

1. Система (Автоматизированная библиотека)

* Автоматизированная система, которая хранит данные и управляет процессами.
* Действия:
* Обновление количества доступных экземпляров книг
* Хранение и обработка данных о читателях, книгах и выдаче книг
* Генерация уведомлений об ошибках или подтверждения действий

**1.1.3. Словарь предметной области**

Словарь предметной области содержит термины и их определения, с учетом использования файловой системы вместо традиционной базы данных для хранения информации. В данном случае данные хранятся в файлах, структура которых организована в виде текстовых файлов

Таблица 1

|  |  |
| --- | --- |
| **Термин** | **Описание** |
| Читатель | Пользователь библиотеки, зарегистрированный в системе, имеющий возможность брать книги. |
| Книга | Издание, доступное в библиотеке для временного пользования читателями. |
| ID | Уникальный идентификатор для однозначной идентификации записи о читателе или книге. |
| Выдача книги | Процесс передачи экземпляра книги читателю на определяенный срок. |
| Дата выдачи | Дата, когда книга была выдана читателю. |
| Дата возврата | Дата, до которой книга должна быть возвращена в библиотеку. |
| Доступные экземпляры | Количество копий книги, которые могут быть выданы читателям в данный момент. |
| Общее количество | Общее число копий книги, находящихся в библиотечном фонде. |
| Регистрация | Процесс внесения новой записи в систему, например, добавление нового читателя или книги. |
| Удаление данных | Процесс удаления записи о читателе, книге или всех данных системы. |
| Поиск | Функция системы, которая позволяет находить книги и читателей по определенным параметрам. |
| Администратор | Библиотекарь или ответственный пользователь, управляющий данными в системе. |
| Пароль | Защитный код, используемый для аутентификации читателя в системе. |
| Очистка данных | Удаление всех записей о читателях, книгах или выдачах из базы данных. |
| Система | Программное обеспечение, автоматизирующее работу библиотеки. |
| ФИО | Полное имя читателя, включая фамилию, имя и отчество. |
| Издательство | Организация, осуществившая выпуск книги. |
| Год издания | Год, когда книга была опубликована. |

**1.2 Функционал, который будет разработан:**

1. **Регистрация нового читателя:**

- Ввод информации о читателе (номер читательского билета, ФИО, год рождения, адрес, место работы/учёбы).

- Валидация введённой информации.

- Создание записи о новом читателе.

- Обработка ошибок при регистрации.

- Отображение сообщений о результатах регистрации.

**2) Снятие с обслуживания читателя:**

- Ввод номера читательского билета для удаления.

- Проверка наличия выданных книг у читателя.

- Удаление записи о читателе.

- Обработка ошибок при снятии с обслуживания.

- Отображение сообщений о результатах удаления.

**3) Просмотр всех зарегистрированных читателей:**

- Получение актуального списка читателей.

- Отображение списка читателей с основными данными (например, ФИО и номер читательского билета).

**4) Очистка данных о читателях:**

- Удаление всех записей о читателях.

- Подтверждение действия пользователем.

- Обработка ошибок при очистке данных.

- Отображение сообщений о результатах очистки.

**5) Поиск читателя по номеру читательского билета:**

- Ввод номера читательского билета для поиска.

- Отображение всех данных о найденном читателе.

- Отображение шифров книг, выданных читателю.

- Обработка ошибок при поиске.

**6) Поиск читателя по ФИО:**

- Ввод ФИО или его части для поиска.

- Отображение списка найденных читателей с указанием номера читательского билета и ФИО.

- Обработка ошибок при поиске.

**7) Добавление новой книги:**

- Ввод информации о книге (шифр, автор(ы), название, издательство, год издания, количество экземпляров).

- Валидация введённой информации.

- Создание записи о новой книге.

- Обработка ошибок при добавлении книги.

- Отображение сообщений о результатах добавления.

1. **Удаление сведений о книге:**

- Ввод шифра книги для удаления.

- Проверка наличия выданных экземпляров книги.

- Удаление записи о книге.

- Обработка ошибок при удалении.

- Отображение сообщений о результатах удаления.

1. **Просмотр всех имеющихся книг:**

- Получение актуального списка книг.

- Отображение списка книг с основными данными (например, шифр, автор(ы), название).

1. **Очистка данных о книгах:**

- Удаление всех записей о книгах.

- Подтверждение действия пользователем.

- Обработка ошибок при очистке данных.

- Отображение сообщений о результатах очистки.

1. **Поиск книги по шифру:**

- Ввод шифра книги для поиска.

- Отображение всех данных о найденной книге.

- Отображение номеров читательских билетов и ФИО читателей, которым выданы экземпляры книги.

- Обработка ошибок при поиске.

1. **Поиск книги по фрагментам ФИО автора(ов) или названия:**

- Ввод фрагмента ФИО автора(ов) или названия для поиска.

- Отображение списка найденных книг с указанием шифра, автора(ов), названия, издательства, года издания и количества экземпляров в наличии.

- Обработка ошибок при поиске.

1. **Регистрация выдачи экземпляра книги читателю:**

- Ввод номера читательского билета и шифра книги.

- Проверка наличия свободных экземпляров книги.

- Регистрация выдачи книги.

- Обновление количества экземпляров в наличии.

- Обработка ошибок при регистрации.

- Отображение сообщений о результатах регистрации.

1. **Регистрация приема экземпляра книги от читателя:**

- Ввод номера читательского билета и шифра книги.

- Регистрация возврата книги.

- Обновление количества экземпляров в наличии.

- Обработка ошибок при регистрации.

- Отображение сообщений о результатах регистрации.

**1.3 Функциональные требования**

1. Регистрация нового читателя:

1. Генерация уникального ID читателя

* Метод: generateString()
* ID формируется в формате:
* NNNN-YY, где:
* NNNN — случайное 4-значное число (генерируется с помощью rand()).
* YY — текущий год (две последние цифры).
* Дополнительно ID префиксируется значением выбранного типа регистрации:
* A для подписки.
* R для доступа в читальный зал.
* B для обоих вариантов.
* Пример результата: A1234-24.

1. Ввод данных о пользователе

* Данные вводятся через текстовые поля и комбинированные списки:
* **Имя** (nameedit\_register)
* Дата рождения (день, месяц, год из comboBoxDay\_register, comboBoxMonth\_register, birthDateedit\_register)
* Email (emailedit\_register)
* Адрес (addressedit\_register)
* Пароль (passwordedit\_register)

1. Валидация данных

* Валидация даты рождения:
* Проверка корректности даты (метод isValidDate()):
* Проверяет, что введённые день и месяц находятся в допустимых пределах.
* В случае ошибки выводит сообщение: "Неверная дата"
* Валидация email:
* Проверка правильности формата email (метод isValidEmail()):
* В случае некорректного email выводит сообщение: "Адрес электронной почты указан неверно".
* Уникальность данных:
* ID генерируется заново при каждой регистрации, гарантируя его уникальность.
* Дополнительная проверка дублирования email отсутствует на этапе генерации, но предполагается на уровне HashTable и базы данных.
* Обработка ошибок:
* В случае пустых полей выводится сообщение: "Ваших данных недостаточно, регистрация не удалась".

1. Очистка формы после регистрации

* Метод: cleanScreen()
* Очищает все текстовые поля на форме.
* Дополнительное сообщение пользователю:
* Отображается сгенерированный ID и пароль:  
  "Ваше имя пользователя: ...".

1. Выход из окна регистрации

* Метод: on\_exit\_regi\_clicked()
* Очищает форму и комментарии.
* Переключается на главное окно (MainWindow).

1. Показ окна регистрации

* Метод: Show()
* Отображает диалог регистрации

1. **РАЗРАБОТКА КЛАССОВ**

**2.1. Классы сущностей**

Классы сущностей представляют собой основные структуры данных, которые содержат информацию в системе.

1. **abv\_tree**

* Описывает информацию о книге.
* Класс Node представляет отдельную запись в AVL-дереве и содержит атрибуты, такие как:
* key: Уникальный идентификатор записи (int)
* id: Уникальный идентификатор книги (string)
* writter: Автор книги (string)
* title: Название книги (string)
* publisher: Издатель книги (string)
* year\_publication: Год публикации (string)
* totalNumberOfCopies: Общее количество экземпляров книги (int)
* copiesAvailable: Количество доступных экземпляров в данный момент (int)
* Node left\*: Указатель на левое дочернее дерево (Node)
* Node right\*: Указатель на правое дочернее дерево (Node)
* int height: Высота узла для поддержания сбалансированности AVL-дерева (Node)Основные Методы:

*struct* *Node*

{

*int* key;

    std::string id, writter, title, publisher, year\_publication;

*int* totalNumberOfCopies, copiesAvailable;

*Node*\* left;

*Node*\* right;

*int* height;

};

**Основные Методы:**

1. ****Вставка (insert):** Добавляет новые узлы, при этом поддерживая свойства BST. Если узел становится несбалансированным (на основе коэффициента баланса), выполняются повороты для обеспечения сбалансированности дерева.**
2. ****Удаление (deleteNode):** Удаляет узлы из дерева, при этом корректируя структуру для поддержания сбалансированности.**
3. ****Поиск (search и searchbytitle):** Находит конкретные узлы по key или writter и возвращает их.**
4. ****Обход (preOrder, inOrderTraverse, Traverse):** Проходит по дереву для сбора и хранения данных узлов, включая вставку в базу данных и запись в внешний файл.**
5. ****Хранение данных (downloadDataToDatabase)**: Вставляет все данные узлов в базу данных и записывает их в файл.**
6. ****Проверка баланса:** Обеспечивает поддержание свойства самобалансировки AVL-дерева путем выполнения поворотов**

*int* height(*Node*\* *N*);

*int* max(*int* *a*, *int* *b*);

*Node*\* newNode(NewNode *a*);

*Node*\* rightRotate(*Node*\* *y*);

*Node*\* leftRotate(*Node*\* *x*);

*int* getBalance(*Node*\* *N*);

*Node*\* insert(*Node*\* *node*, NewNode *a*);

*void* preOrder(*Node*\* *root*);

*bool* validateFormat(QString& *input*);

*Node* \*deleteNode(*Node* \**root*, *int* *key*);

*void* inOrderTraverse(*Node*\* *node*, std::vector<NewNode>& *nodeList*);

*void* insertNodeIntoDatabase(*Node*\* *node*, MainWindow& *conn*, std::ofstream& *outFile*);

*void* Traverse(*Node*\* *root*, MainWindow& *conn*, std::ofstream& *outFile*);

*void* downloadDataToDatabase(*Node*\* *root*) ;

*void* deleteAllBooksData();

*Node*\* search(*Node*\* *root*, *int* *key*);

*Node*\* searchbytitle(*Node*\* *root*, const std::string& *title*);

*bool* searchByStringID(*Node*\*& *root*, const  Datatype& *id*,std::string& *file*);

1. **Hashtable**

* Описывает управление данными для читателей с использованием хэш-таблицы
*  Класс HashNode представляет отдельную запись в хэш-таблице и содержит атрибуты, такие как:
* key: Уникальный идентификатор записи (int)
* username: Имя пользователя (string)
* name: Имя (string)
* birthDate: Дата рождения (string)
* email: Электронная почта (string)
* address: Адрес (string)
* password: Пароль (string)
* link bookstatus\*: Ссылка на список книг с их состоянием (link)
* occupied: Флаг, указывающий, занят ли текущий элемент (bool)
* HashNode(): Конструктор, инициализирующий key = -1 и occupied = false

*struct* *link*{

*int* num=0;

    std::string NNNMMM;

*link*\* next = nullptr;

};

*struct* *HashNode* {

*int* key;

    std::string username;

    std::string name;

    std::string birthDate;

    std::string email;

    std::string address;

    std::string password;

*link*\* bookstatus;

*bool* occupied;

    HashNode() : key(-1), occupied(false) {}

};

**Основные Методы:**

1. ****insert:** Вставляет новые записи в хэш-таблицу, используя квадратичное пробирование.**
2. ****search**: Ищет записи по ключу и возвращает соответствующие данные.**
3. ****remove:** Удаляет запись из хэш-таблицы по ключу.**
4. ****add:** Добавляет книгу к пользователю по имени и ID книги.**
5. ****clear**: Очищает таблицу.**
6. ****displayRow:** Отображает данные в модели таблицы QStandardItemModel.**
7. ****display:** Отображает всю хэш-таблицу в виде таблицы QTableView.**
8. ****isDataPresent:** Проверяет, присутствуют ли данные с данным ключом в таблице.**
9. ****downloadToFile**: Загружает данные из хэш-таблицы в файл.**
10. ****retrieveFromDatabase:** Загружает данные из базы данных в хэш-таблицу.**

*void* insert(HashNode *data*);

*void* search(*int* *key*);

*void* remove(*int* *key*);

*void* add(std::string *key*,std::string *bookID*, *int* *num*);

*void* clear();

*void* displayRow(QStandardItemModel\* *model*, const HashNode& *data*);

*void* display(QTableView\* *tableView*);

*bool* isDataPresent(*int* *key*);

*void* downloadToFile(std::string& *fileName*);

*void* retrieveFromDatabase(HashTable& *TABLE*);

1. **Linked\_list**

* Описывает структуру и методы управления односвязным списком с использованием данных типа data для выдачи и возврата книг.
* Класс record представляет отдельную запись в односвязном списке и содержит атрибуты, такие как:
* data: Структура данных (Datatype), содержащая student\_ID, book\_ID, lending\_date и return\_date.
* next: Указатель на следующую запись (record\*).

*struct* *Datatype* {

    std::string student\_ID, book\_ID, lending\_date, return\_date;

};

*struct* *record* {

*Datatype* data;

*record*\* next = nullptr;

};

**Основные Методы:**

1. f****ree\_All\_record:** Освобождает память для всех узлов в списке.**
2. ****isCompanyNumPresent:** Проверяет, существует ли компания с указанным именем в списке.**
3. ****addToBeginning:** Добавляет новый элемент в начало списка и обновляет хэш-таблицу и дерево.**
4. ****DelElem:** Удаляет элемент из списка, указанный указателем.**
5. ****Find\_Element:** Ищет элемент в списке по ключу.**
6. ****FindElemCompany:** Находит элемент в списке по имени компании.**
7. ****deleteByCompanyNum:** Удаляет все элементы с указанным именем компании.**
8. ****editByCompanyNum:** Редактирует данные элемента по имени компании и возвращает указатель на изменённый элемент.**
9. ****downloadToFile:** Сохраняет содержимое списка в файл.**
10. ****addToEnd:** Добавляет новый элемент в конец списка.**

*void* free\_All\_record(record\*\* *begin*);

*bool* isCompanyNumPresent(record\* *tail*, const std::string& *name*);

*bool* addToBeginning(record\*\* *begin*, Datatype *elem*, HashTable& *myHashTable*, Node\*& *root*,std::string& *file*);

*void* DelElem(record\*\* *begin*, record\* *ptrCur*);

*void* Find\_Element( record\* *tail*, const std::string& *k*);

record\* FindElemCompany(record\* *tail*, const std::string& *elem*);

*void* deleteByCompanyNum(record\*\* *tail*, const std::string& *name*);

record\* editByCompanyNum(record\* *tail*, const std::string& *name*, const Datatype& *newData*);

*void* downloadToFile(record\* *tail*, const std::string& *fileName*);

*void* addToEnd(record\*\* *tail*, const Datatype& *data*);

**2.2. Управляющие классы**

Управляющие классы (контроллеры и менеджеры) координируют взаимодействие между сущностями и интерфейсами. Они отвечают за выполнение бизнес-логики и вызывают необходимые методы классов сущностей.

1. **CalculationFacade**

* Реализует Паттерн Фасада для упрощения расчетов стоимости.  
  Метод:
* getCost(Book\_data \*value, CalcFactory \*factory): Возвращает стоимость книги, используя предоставленный фабрику и калькулятор.

*class* *AbstractCalc*;

*class* *CalcFactory*;

#include "headers.h"

*class* *Book\_data*;

*class* *CalculationFacade*

{

*public:*

    static *int* getCost(*Book\_data* \**value*, *CalcFactory* \**factory*);

};

1. **CalcFactory**

* Абстрактный интерфейс фабрики для создания калькуляторов.  
  Метод:
* **createCalculator():** Чисто виртуальный метод для создания калькуляторов.

*class* *AbstractCalc*;

*class* *CalcFactory*

{

*public:*

    virtual *AbstractCalc*\* createCalculator() = 0;

    virtual ~CalcFactory() {}

};

1. **CalcFactoryImpl**

* Реализация интерфейса CalcFactory.
* Атрибуты:
* **calculator**: Ссылка на реализацию AbstractCalc.
* Метод:
* **createCalculator():** Создает и возвращает экземпляр AbstractCalc.

*class* *CalcFactoryImpl* : *public* *CalcFactory*

{

*public:*

    CalcFactoryImpl(*AbstractCalc* \**calculator*);

*AbstractCalc*\* createCalculator() override;

*private:*

*AbstractCalc* \*calculator;

};

1. **AbstractCalc**

* Абстрактный базовый класс для всех стратегий расчета стоимости.
* Метод:
* **getCost(Book\_data \*value)**: Чисто виртуальный метод для расчета стоимости книг.

#include "book\_data.h"

*class* *AbstractCalc*

{

*public:*

    virtual ~AbstractCalc() {}

    virtual *int* getCost(Book\_data \**value*) = 0;

};

1. **Cost Calculation Classes**

* Конкретные реализации интерфейса AbstractCalc, каждая из которых представляет стратегию расчета стоимости для конкретного издателя.
* UNZA: Рассчитывает стоимость на основе стратегии UNZA.
* Ted\_talkscalc: Рассчитывает стоимость для TED Talks.
* Te\_librarycalc: Рассчитывает стоимость для TE Library.
* Jmax\_limcalc: Стратегия стоимости для Jmax Limited.
* Guapcalc: Стратегия стоимости для GUAP.
* Каждая из них переопределяет метод getCost() для предоставления специфичной логики расчета стоимости.

**2.3. Интерфейсные классы**

Основные интерфейсные классы используются для взаимодействия с пользователем и обработки событий в графическом интерфейе.

1. **MainWindow**

* Класс главного окна приложения.
* Обрабатывает вход в систему, регистрацию и подключения к базе данных.
* Методы:
* **connOpen() и connClose():** Управляют подключениями к базе данных.
* **on\_enter\_login\_clicked() и on\_Register\_Home\_clicked():** Слоты для обработки событий входа и регистрации.
* **on\_cancel\_login\_clicked()**; для выхода из приложения
* **void on\_\_checkBox\_Home\_clicked()**; Чтобы установить флажок перед входом в систему в соответствии с правилами приложения

*void* on\_cancel\_login\_clicked();

*void* on\_enter\_login\_clicked();

*void* on\_Register\_Home\_clicked();

*void* on\_\_checkBox\_Home\_clicked();

1. **ownerDialog**

* Обрабатывает функционал управления книгами и пользователями.
* Методы:
* **loadFromDatabase()**: Загружает данные книг/пользователей из файла.
* **createNewNode()**: Создает новый узел для AVL-дерева.

-Назначение: Отключает указанную кнопку (QPushButton\* button).

-Применение: Предотвращает взаимодействие с кнопкой (например, клики), чтобы избежать нежелательных действий со стороны пользователя.

* **clear\_inputdata()**

-Назначение: Очищает все поля ввода в диалоге.

-Применение: Обеспечивает сброс всех полей ввода, особенно при работе с формами или вводом данных.

* **loadFromDatabase()**

-Назначение: Загружает данные (предположительно из базы данных) в соответствующие компоненты UI, такие как таблицы или списки.

-Применение: Восстанавливает и отображает актуальные данные из базы данных, что полезно для синхронизации данных в реальном времени.

* **createNewNode()**

-Назначение: Создает и возвращает новый узел типа NewNode.

-Применение: Вероятно используется для создания нового узла в каком-то структурированном виде данных, например, в AVL-дереве (NewNode упоминается ранее).

* **uploadDatatypeFromFile(const std::string& fileName, std::vector<Datatype>& data, QListWidget\* listWidget)**

-Назначение: Загружает данные из файла и заполняет ими QListWidget.

-Применение: Облегчает чтение и отображение данных из внешних файлов в UI, возможно, для отображения информации о пользователях или книгах.

* **displayNextPicture()**

-Назначение: Отображает следующую картинку/изображение в последовательности.

-Применение: Вероятно используется в слайдшоу или галерее для отображения визуального контента, таких как изображения.

* **startPictureTimer()**

-Назначение: Запускает таймер, который контролирует автоматическое переключение на следующую картинку/изображение.

-Применение: Используется для автоматизации перехода между картинками, особенно в галерее или просмотре изображений.

* **clearTableView(QTableView\* tableView)**

-Назначение: Очищает данные, отображаемые в QTableView.

-Применение: Полезно для обновления или сброса таблиц данных, особенно при отображении информации о пользователях или результатах поиска.

* **loadMessage()**

-Назначение: Загружает и отображает сообщения в диалоге.

-Применение: Обеспечивает обновление UI для отображения сообщений, возможно, для уведомлений пользователей или системных логов.

* Каждая из этих функций предназначена для упрощения взаимодействия между GUI и базой данных, обеспечивая управление информацией о пользователях, книгах и функциями системы.

1. **Portaldialog**

* Обеспечивает пользовательский портал для поиска и покупки книг.
* Методы:
* on\_search\_button\_clicked(): Искать книги в системе.
* on\_buy\_book\_clicked(): Позволяет покупать книги.
* void func(QString, QString);

-Назначение: Метод для выполнения определенных операций с входными строковыми параметрами.

-Применение: Может использоваться для обработки данных, взаимодействия с базой данных или другими компонентами приложения.

* void displayNextP();

-Назначение: Отображает следующую картинку/изображение.

-Применение: Используется для перехода к следующему изображению в слайд-шоу или галерее.

* void startPictureT();

-Назначение: Запускает таймер для автоматического перехода между изображениями.

-Применение: Входит в обработку слайд-шоу или отображения графических элементов.

* void databasetofile();

-Назначение: Сохраняет данные из базы данных в файл.

-Применение: Позволяет выгрузить данные в файл, возможно, для архивирования или дальнейшего использования.

* Book\_data\* fillform();

-Назначение: Заполняет форму данными из объекта Book\_data.

-Применение: Используется для сбора данных о книгах из формы ввода.

* void showCost(Book\_data \*value);

-Назначение: Отображает стоимость книги, предоставленной через указатель на объект Book\_data.

-Применение: Помогает показать расчет стоимости книги, например, перед покупкой.

* void on\_minimize\_clicked();

-Назначение: Обработчик события клика на кнопку "свернуть".

-Применение: Свертывает диалоговое окно.

* void on\_message\_button\_clicked();

-Назначение: Обработчик события клика на кнопку "сообщение".

-Применение: Открывает окно сообщений или диалог для взаимодействия с пользователем.

* void on\_search\_button\_clicked();

-Назначение: Обработчик события клика на кнопку "поиск".

-Применение: Осуществляет поиск книг или информации в системе.

* void on\_buy\_book\_clicked();

-Назначение: Обработчик события клика на кнопку "купить книгу".

-Применение: Позволяет приобрести книгу, взаимодействуя с соответствующими механизмами.

* void on\_exit\_clicked();

-Назначение: Обработчик события клика на кнопку "выход".

-Применение: Закрывает диалоговое окно или завершает приложение.

* bool readCredentialsFromFile(QString &username, QString &password);

1. **Registerdialog**

* Обрабатывает формы регистрации пользователей и их ввод.
* Метод:
* cleanScreen(): Очищает поля формы после регистрации пользователя.
* on\_enter\_register\_clicked(): Регистрация нового пользователя в библиотеке

**2.4. Используемые паттерны проектирования**

1. Фабричный метод

// calcfactory.h

#ifndef CALCFACTORY\_H

#define CALCFACTORY\_H

*class* *AbstractCalc*;

*class* *CalcFactory*

{

*public:*

    virtual *AbstractCalc*\* createCalculator() = 0;

    virtual ~CalcFactory() {}

};

#endif // CALCFACTORY\_H

// calcfactoryimpl.h

#ifndef CALCFACTORYIMPL\_H

#define CALCFACTORYIMPL\_H

#include "headers.h"

*class* *CalcFactoryImpl* : *public* *CalcFactory*

{

*public:*

    CalcFactoryImpl(*AbstractCalc* \**calculator*);

*AbstractCalc*\* createCalculator() override;

*private:*

*AbstractCalc* \*calculator;

};

#endif // CALCFACTORYIMPL\_H

* Описание: Паттерн, используемый для создания объектов без указания их конкретного класса. Используется для динамического создания экземпляров калькулятора.
* Применение: В классе CalcFactory и его реализации CalcFactoryImpl.

2. Фасад (Facade)

// calculationfacade.h

#ifndef CALCULATIONFACADE\_H

#define CALCULATIONFACADE\_H

*class* *AbstractCalc*;

*class* *CalcFactory*;

#include "headers.h"

*class* *Book\_data*;

*class* *CalculationFacade*

{

*public:*

    static *int* getCost(*Book\_data* \**value*, *CalcFactory* \**factory*);

};

#endif // CALCULATIONFACADE\_H

* Описание: Паттерн, который предоставляет единый интерфейс для взаимодействия с подсистемой.
* Применение: CalculationFacade объединяет работу с фабрикой и калькуляторами.

**2.5. Диаграмма классов**

На диаграммах ниже показаны основные классы и их связи:

Диаграмма 1: Иерархия классов калькуляторов

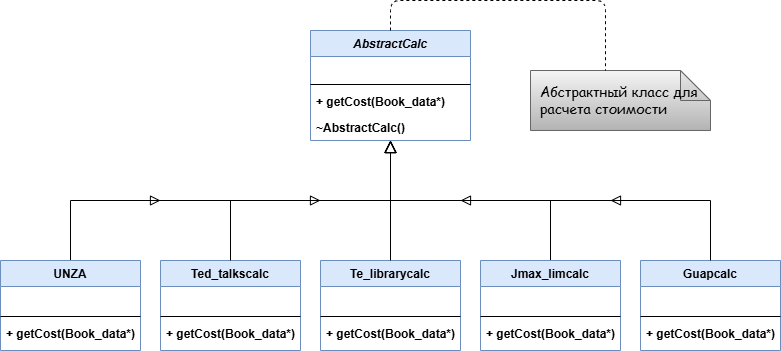


Рисунок 1 — Диаграмма классов для абстрактного класса

**Описание:**

* AbstractCalc - интерфейс, реализуемый всеми калькуляторами.
* Наследующие классы: UNZA, Ted\_talkscalc, Te\_librarycalc, Jmax\_limcalc, Guapcalc - представляют конкретные реализации метода getCost() для различных расчетов.

**Диаграмма 2: Фабрика и фасад**

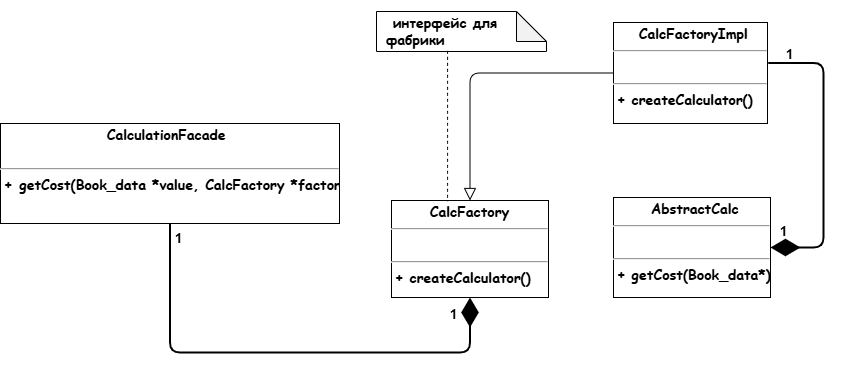
****

Рисунок 2— Диаграмма классов для а Фабрики и фасад класса

**Описание :**

CalcFactory - интерфейс для фабрики.

CalcFactoryImpl - конкретная реализация фабрики.

CalculationFacade - фасад для работы с фабрикой и калькуляторами.

**Диаграмма для Observer:**

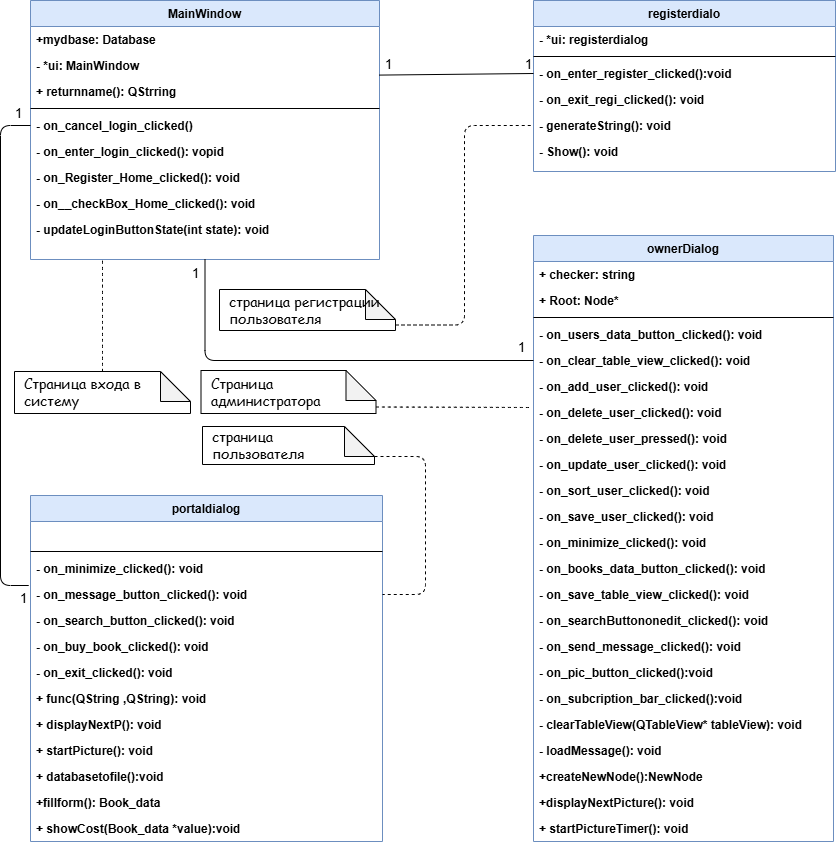
****

Рисунок 3— Диаграмма классов для наблюдателя

На каждом компьютере пользователь может переключаться только с одного интерфейса на другой. Когда активен один интерфейс, предыдущий скрыт, а переключение между пользователями осуществляется с помощью пароля

**Общая диаграмма:**

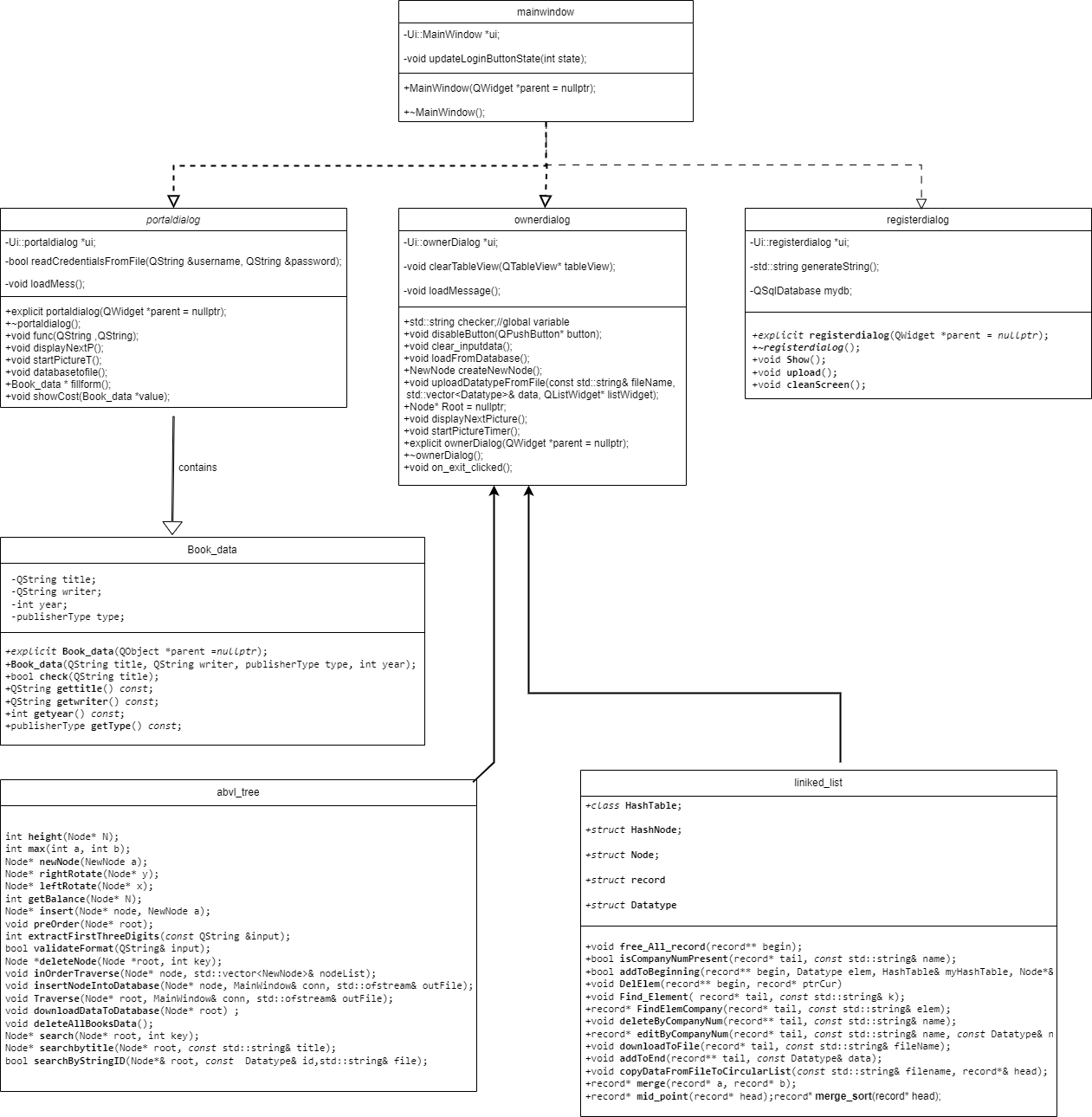
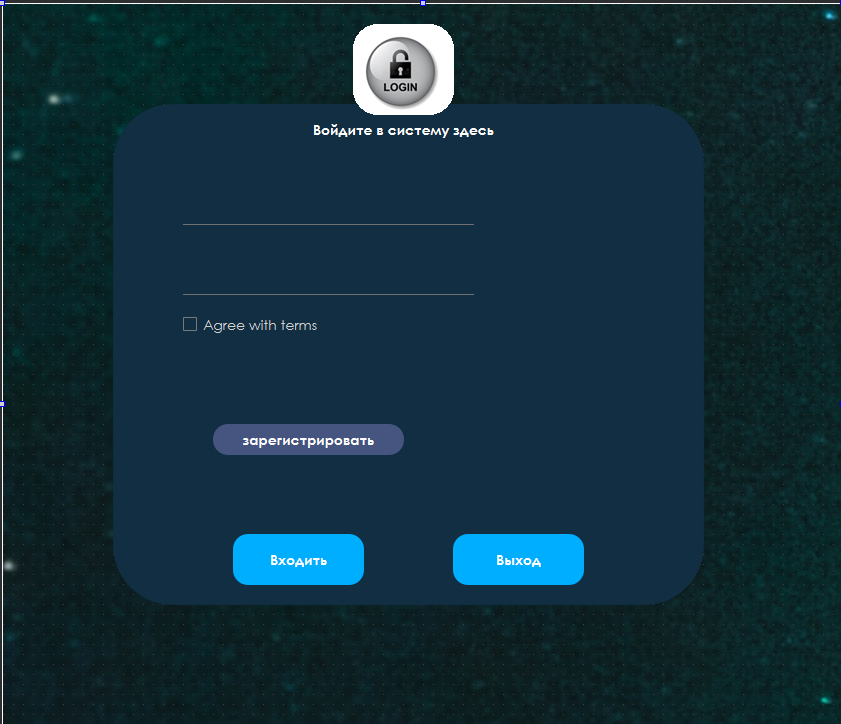
****

Рисунок 4— Диаграмма классов для общая программа

1. **РАЗРАБОТКА ПРИЛОЖЕНИЯ**

**3.1. Разработка интерфейса приложения**

Скриншоты всех разработанных программных форм в режиме дизайнера с указанием программных имён всех виджетов:



QCheckBox

QPushButton

MainWindow

QlineEdit

-edit

QLabel

Рисунок 5— форма входа в систему в режиме разработки

**Описание виджета к программе:**

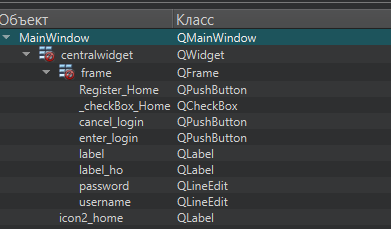
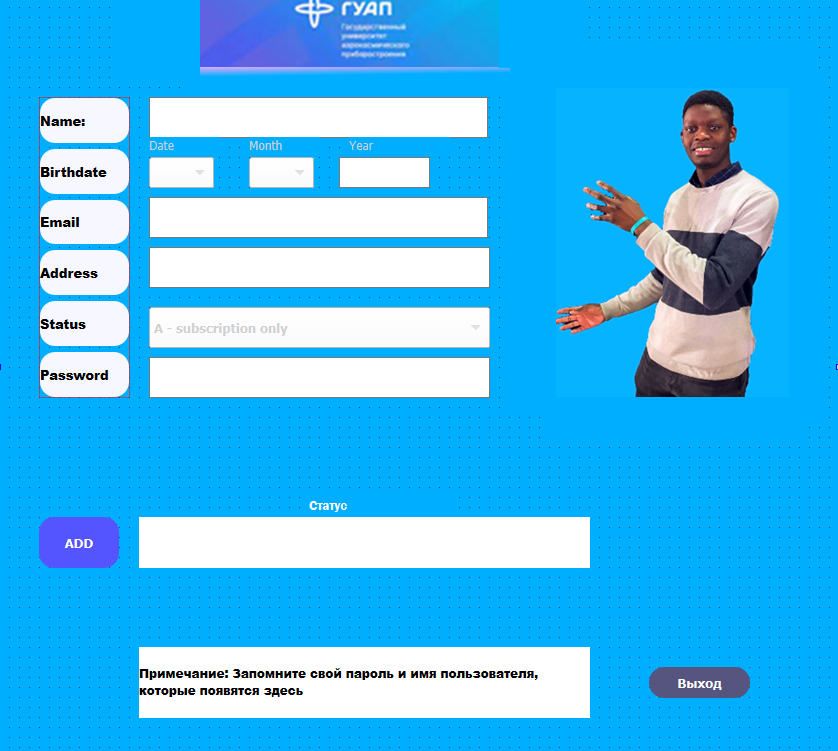


Рисунок 6—Описание виджет

Для mainwindow

Таблица 2

|  |  |
| --- | --- |
| **Имя Виджета** | **Что он делает** |
| centralwidget(QWidget) | страница для входа в систему |
| frame(QFrame) | рамка, содержащая все виджеты |
| Register\_Home(QPushButton) | для регистрации новых пользователей |
| \_checkBox\_Home(QcheckBox) | принять условия библиотек |
| cancel\_login(QPushButton) | чтобы выйти из системы |
| enter\_login(QPushButton) | чтобы войти в систему |
| Password(QLineEdit) | для ввода паролей |
| Username(QLineEdit) | для ввода имени пользователя |



QGraphicsView

QComboBox

QPushButton

QLineEdit

QComboBox

QLabel

Рисунок 7— регистрационная форма в режиме разработки



Рисунок 8—Описание виджет

Registerdialog(QDialog) -

address\_register(QLineEdit) - для ввода адреса пользователя

birthDatedit\_register (QLineEdit) - для ввода год рождения

comboBoxDate(QComboBox) - для ввода дата рождения

Emailedit\_register(QLineEdit) - для ввода почтовый адрес

Enter\_register(QPushButton) - Кнопка для добавления

nameedit\_register(QLineEdit) - для ввод имия

passwordedit\_register(QLineEdit) - для ввод пороль

status\_register(QLabel) - Для ввода статуса доступа к библиотеке

titlegraph(QGraphicsView) - для отображения изображения

verticalLayout(QVBoxLayout) - Макет, хранящий данные

address\_register(QLabel) - чтобы отобразить адресную метку

birthDate(QLabel) -чтобы отобразить дату метку

Email\_register(QLabel) - чтобы отобразить почтовую метку

name\_register(QLabel) - чтобы отобразить имена метку

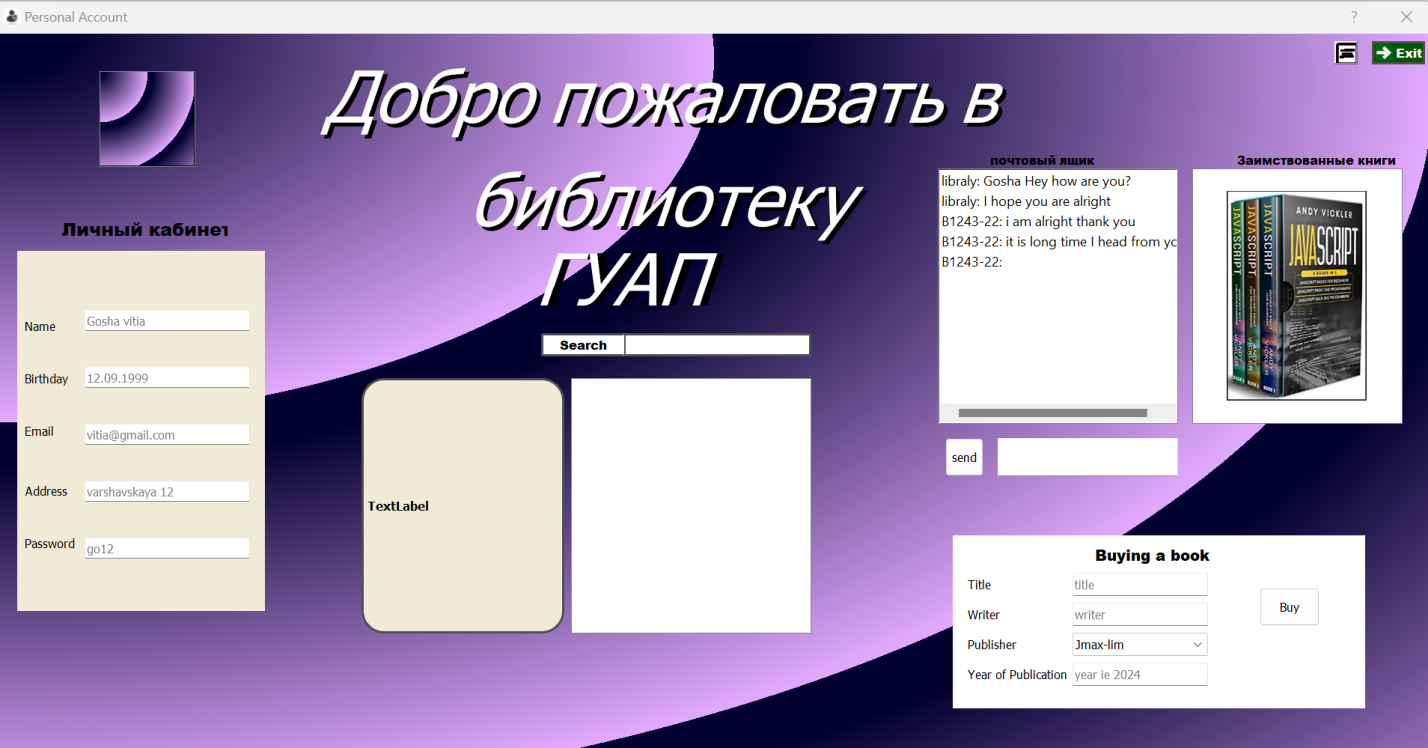
password\_register(QLabel) - чтобы отобразить пароль метку

status\_combo(QLabel) - чтобы отобразить метку состояния

QListWidget

QGraphicsView

QGraphicsView



QLabel

Qverticallayout

QFrame

QPushButton

QComboBox

Рисунок 9—форма страницы пользователя в режиме разработки

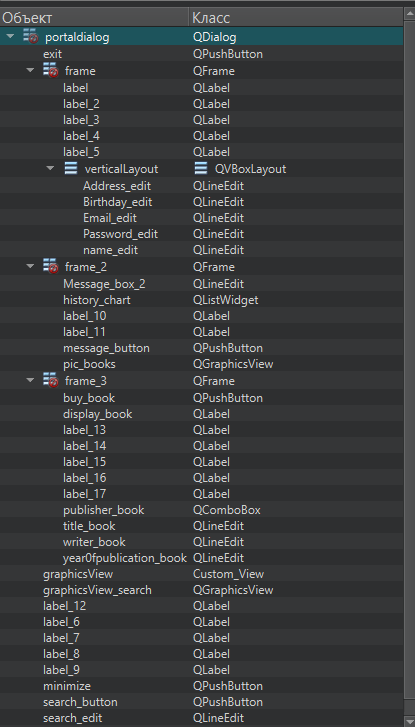


Рисунок 10—Описание виджет

Frame(QFrame) - рамка, содержащая виджеты

verticalLayout(QVBoxLayout) - макет, хранящий данные

Address\_edit(QLineEdit) - Для редактирования адреса

Birthday\_edit(QLineEdit) - Для редактирования дата рождения

Email\_edit(QLineEdit) - Для редактирования почта

Password\_edit(QLineEdit) - Для редактирования пароль

Name\_edit(QLineEdit) - Для редактирования имени

Frame\_2(QFrame) - - рамка, содержащая все виджеты

Message\_boxFrame(QLineEdit) - для ввода сообщений

History\_chart(QListWidget) - для хранения истории разговоров

message\_button(QPushButton) - для отправки сообщений администрации библиотеки

Pic\_books(QGraphicsView) - для отображения изображений заимствованных книг

buy\_book(QPushButton) - заказать книгу в библиотеке

display\_book(QLabel) - этикетка для отображения данных о книге

publisher\_book(QComboBox) - для отображения доступных издателей книг

title\_book(QLineEdit) - чтобы ввести название книги

writer\_book(QLineEdit) - чтобы стать писателем книг

yearOfpublication\_book(QLineEdit) - указать год публикации

graphicsView\_search(QGraphicsView) - для отображения книг, в которых выполняется поиск

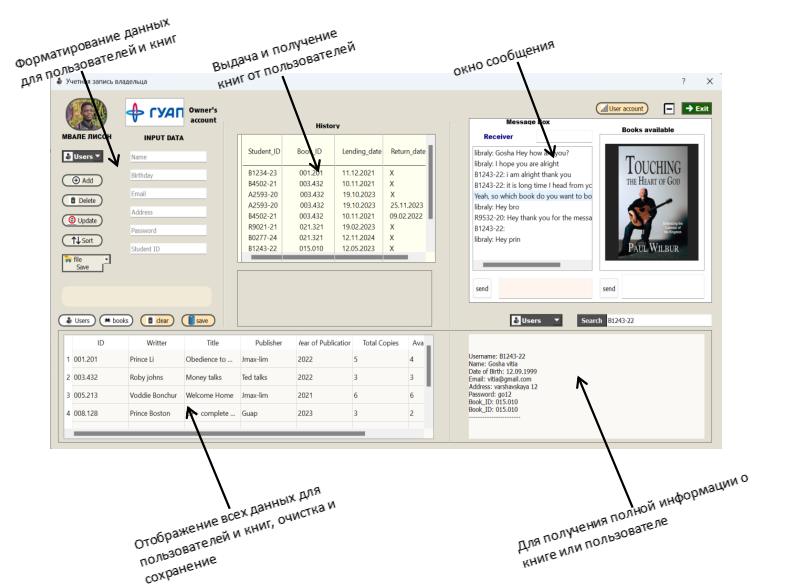
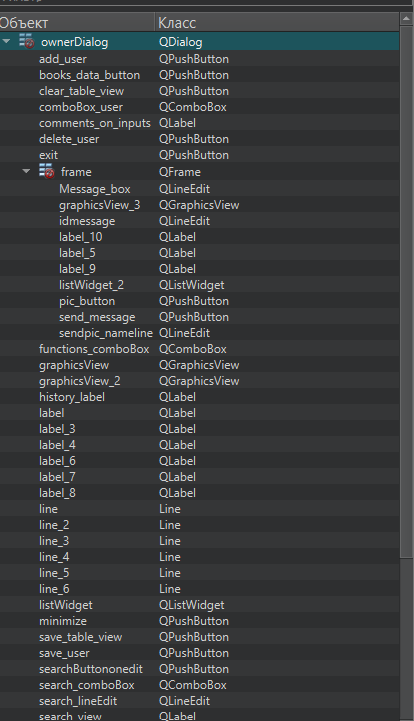


Рисунок 11—форма страницы администратора в режиме разработки



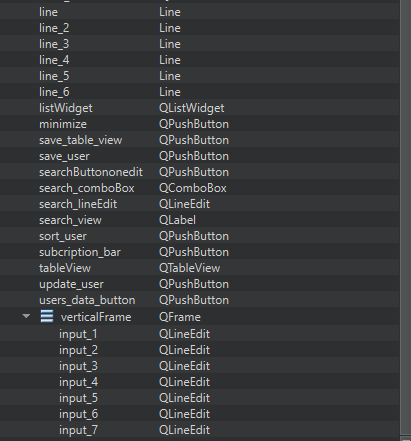


Рисунок 12—Описание виджет

Functions\_comboBox(QComboBox) - отображает функциональные возможности для книги, пользователей и выдачи книги

History\_label(QLabel) - хранит историю выпуска книг

listWidget(QListWidget) - Виджет списка для отображения данных о книгах и пользователях

Minimize(QPushButton) - чтобы свернуть экран

Save\_table\_view(QPushButton) - чтобы сохранить таблицу книг в файле

Save\_user(QPushButton) - чтобы сохранить таблицу пользователя в файле

searchButtononedit(QPushButton) - кнопка для поиска

Search\_comboBox(QComboBox) - отображает, какие данные для поиска, книги или пользователей

Search\_lineEdit(QLineEdit) - предоставляет поле для поиска

Search\_view(QLable) - предоставляет подробную информацию о найденных данных

graphicsView(QGraphicsView) - отображает изображение аватарки

graphicsView\_2(QGraphicsView) - отображает изображения книгframe(QFrame) - рамка, содержащая виджеты

graphicsView\_3(QGraphicsView) - отображает изображения книг, доступных в библиотеке

send\_message(QPushButton) - кнопка для отправки сообщения определенным пользователям

sendpic\_nameline(QLineEdit) - метка для ввода получателя изображения

pic\_button(QPushButton) - кнопка для отправки изображения

listWidget\_2(QListWidget) - виджет списка для хранения сообщений

idmessage(QLineEdit) - метка для ввода ID для отправки

Message\_box(QLineEdit) - метка для ввода сообщения для отправки

exit(QPushButton) - за существующую программу

delete\_user(QPushButton) - чтобы удалить пользователя

comments\_on\_inputs(QLabel) - метка для комментирования запущенной программы

clear\_table\_view(QPushButton) - чтобы очистить табличный вид

comboBox\_user(QComboBox) - чтобы установить связь между пользователем и книгой

Add\_user(QPushButton) - чтобы добавить пользователя

Books\_data\_button(QPushButton) - кнопка "книги"

**3.2. Реализация методов классов**

**3.2.1. Объяснение функций в классе, взаимодействующих с пользовательским интерфейсом (UI) mainwindow:**

### 1. ****Конструктор**** MainWindow::MainWindow

* **Назначение:**
  + Инициализирует элементы интерфейса при создании окна.
  + Устанавливает начальные значения для полей:
    - Поле username получает подсказку "Username".
    - Поле password получает подсказку "Password".
    - Флажок \_checkBox\_Home устанавливается в начальное состояние (false).
  + Подключает сигналы и слоты:
    - Состояние флажка \_checkBox\_Home отслеживается через updateLoginButtonState.
    - Обрабатывается событие клика на флажке \_checkBox\_Home с помощью on\_\_checkBox\_Home\_clicked.
* **Связь с интерфейсом:**
  + Обеспечивает инициализацию графических компонентов (username, password, \_checkBox\_Home).
  + Устанавливает подключение событий (через connect).

### 2. ****Метод**** void MainWindow::on\_cancel\_login\_clicked()

* **Назначение:**
  + Закрывает главное окно при нажатии кнопки "Cancel".
* **Связь с интерфейсом:**
  + Отвечает за реакцию на нажатие кнопки cancel\_login.

### 3. ****Метод**** void MainWindow::updateLoginButtonState(int state)

* **Назначение:**
  + Включает или отключает кнопку enter\_login в зависимости от состояния флажка \_checkBox\_Home.
    - Если флажок включен, кнопка активна.
    - Если флажок выключен, кнопка неактивна.
* **Связь с интерфейсом:**
  + Управляет состоянием доступности кнопки enter\_login.

### 4. ****Метод**** void MainWindow::on\_enter\_login\_clicked()

* **Назначение:**
  + Обрабатывает нажатие кнопки enter\_login для входа в систему.
  + Считывает введенные пользователем данные (username и password) и проверяет:
    - Локальные данные для администратора.
    - Данные в базе данных.
  + Открывает новое окно (например, ownerDialog или portaldialog) при успешной проверке.
  + Выводит сообщение об ошибке при неудачной проверке.
* **Связь с интерфейсом:**
  + Считывает текст из полей username и password.
  + Управляет текстом метки label\_ho, чтобы показать ошибки.
  + Скрывает текущее окно и отображает новое окно при успешной проверке.

### 5. ****Метод**** QString MainWindow::returnname()

* **Назначение:**
  + Возвращает текст, введенный в поле username.
* **Связь с интерфейсом:**
  + Предоставляет доступ к текущему значению текстового поля username.

### 6. ****Метод**** void MainWindow::on\_Register\_Home\_clicked()

* **Назначение:**
  + Закрывает текущее окно.
  + Открывает диалоговое окно регистрации (registerdialog).
* **Связь с интерфейсом:**
  + Реагирует на нажатие кнопки Register\_Home, чтобы переключиться на окно регистрации.

### 7. ****Метод**** void MainWindow::on\_\_checkBox\_Home\_clicked()

* **Назначение:**
  + Предотвращает установку флажка \_checkBox\_Home, если поля username или password пусты.
* **Связь с интерфейсом:**
  + Проверяет содержимое текстовых полей username и password.
  + Управляет состоянием флажка \_checkBox\_Home.

**3.2.2. Резюме взаимодействия класса registerdialog и пользовательского интерфейса**

**1. Конструктор** registerdialog

* Инициализирует интерфейс и наполняет выпадающие списки для выбора даты (дни и месяцы).
* Выполняет связь между UI-компонентами и логикой класса.
* Метод on\_enter\_register\_clicked
  1. Реализует обработку данных, введённых пользователем для регистрации:

1. Получает значения из полей nameedit\_register, birthDateedit\_register, emailedit\_register, passwordedit\_register, а также из выпадающего списка comboBox\_register.
2. Проверяет корректность введённой даты с помощью функции isValidDate.
3. Генерирует уникальный ID пользователя через метод generateString и определяет тип подписки.
4. Вставляет данные в локальную хэш-таблицу (HashTable) и базу данных login\_data.
   1. Выводит сообщение об успешной или неудачной регистрации в UI-компоненты (comment\_register и getpersonal\_data\_2).

* Метод cleanScreen
  1. Очищает все поля ввода (nameedit\_register, birthDateedit\_register, emailedit\_register, passwordedit\_register), чтобы подготовить экран к следующему пользователю.
* Метод generateString
  1. Генерирует уникальный идентификатор пользователя в формате NNNN-YY (случайное 4-значное число и последние две цифры текущего года).
  2. Используется в логике создания ID для нового пользователя в методе on\_enter\_register\_clicked.
* Метод on\_exit\_regi\_clicked
  1. Закрывает диалог регистрации, очищает UI-компоненты, проверяет подключение к базе данных и переключает интерфейс на главное окно.
  2. Создаёт новое окно (MainWindow) и отображает его.

### Передача данных между классом и интерфейсом:

**Получение данных из интерфейса:**  
Данные вводятся пользователем в поля ввода (QLineEdit) и выпадающие списки (QComboBox), затем извлекаются через методы text() и currentText(). Например:

QString date = ui->comboBoxDay\_register->currentText();

QString year = ui->birthDateedit\_register->text();

**Обновление интерфейса:**  
Обновление текста в интерфейсе происходит с использованием методов setText(). Например:

ui->comment\_register->setText("Регистрация прошла успешно");

**Очистка полей интерфейса:**  
Метод cleanScreen очищает введённые данные через clear() для соответствующих полей:

ui->nameedit\_register->clear();

**3.3. Разработка тестового приложения**

Тестовое приложение создаётся для проверки всех компонентов системы, включая реализацию классов, методов и пользовательского интерфейса. Оно помогает выявить ошибки и улучшить функциональность.

Основные этапы тестирования:

1. Интеграционное тестирование:

Проверка взаимодействия между сущностями, контроллерами и интерфейсами;

При запуске MainWindow сущност

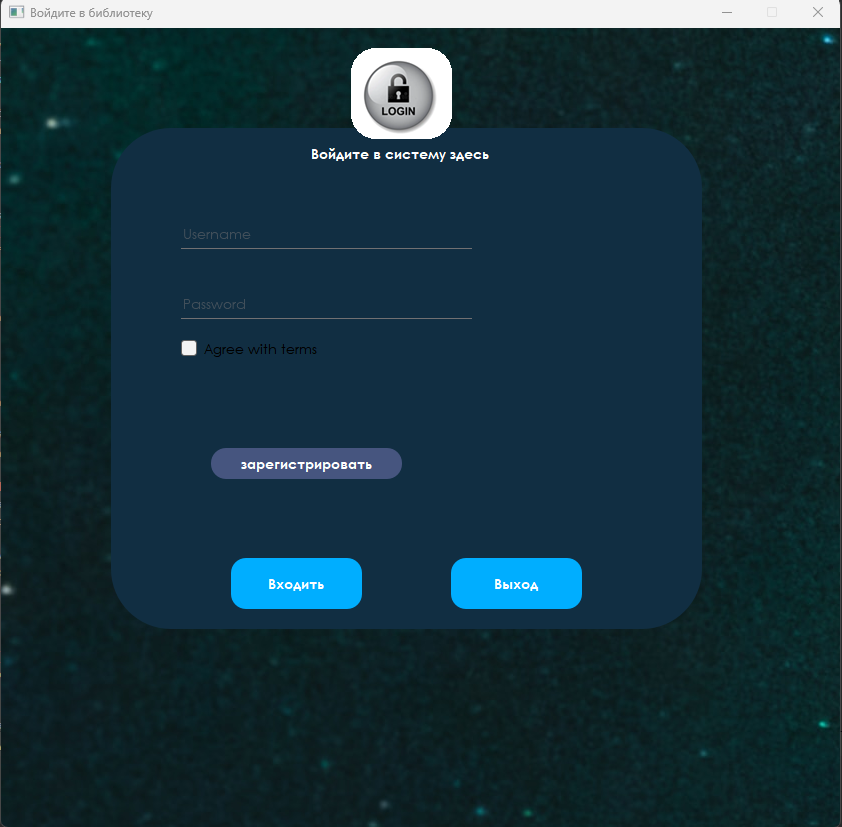


Рисунок 13.1- Логин страница

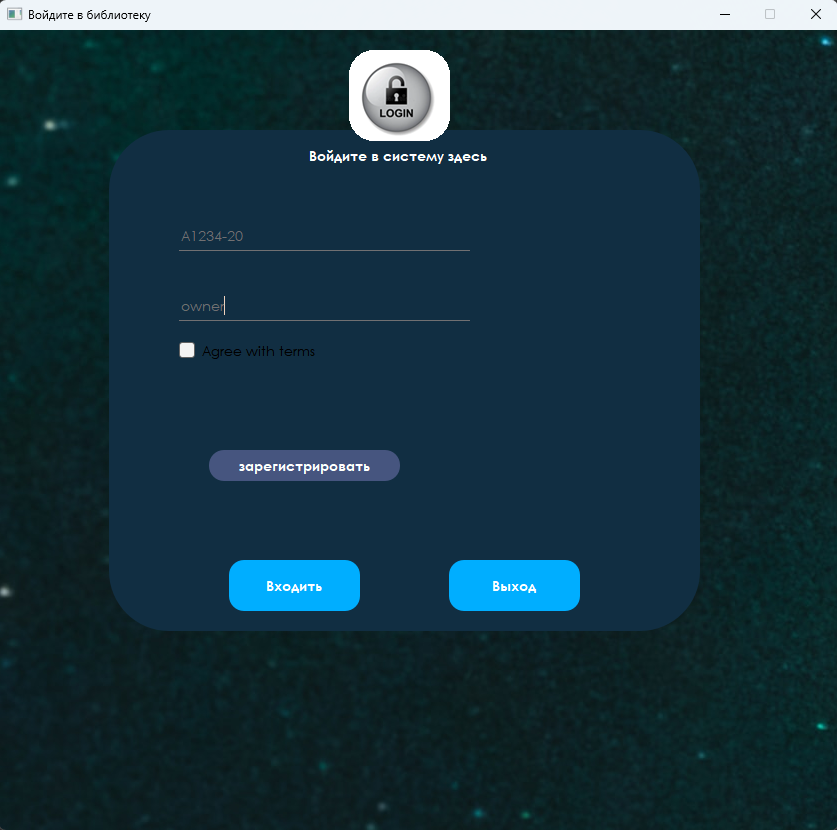


Рисунок 13.2- Логин страница

Кнопки не могут быть нажаты, если не отмечена кнопка "Проверить соблюдение правил". Когда я нажимаю "Зарегистрироваться", меня перенаправляют на страницу регистрации

1. **Интерфейс регистрации**

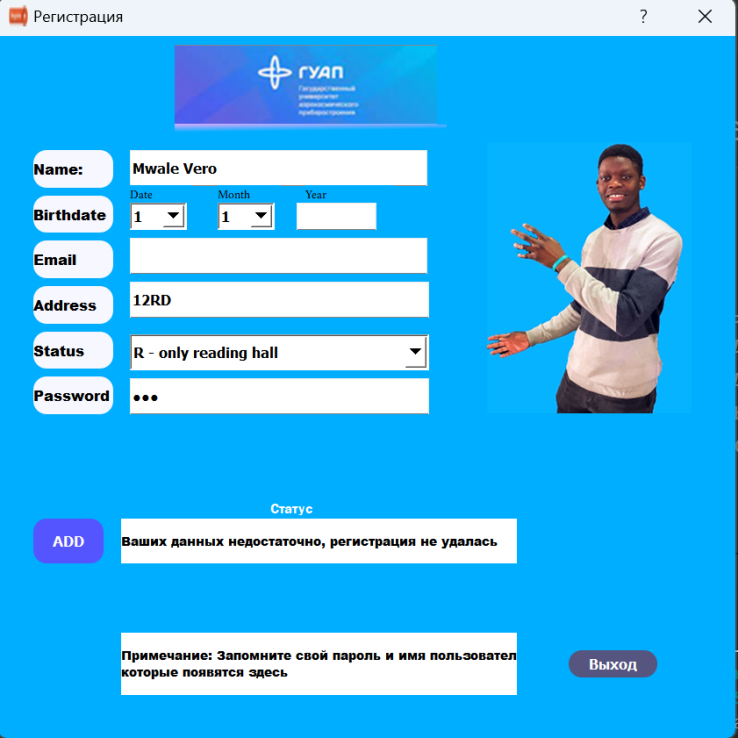


Рисунок 13.3- Регистрация страница

Когда я выхожу, меня возвращают на страницу входа в систему



Рисунок 14.1- Логин страница

Не останавливая работу приложения, теперь я вхожу в систему в роли администратора

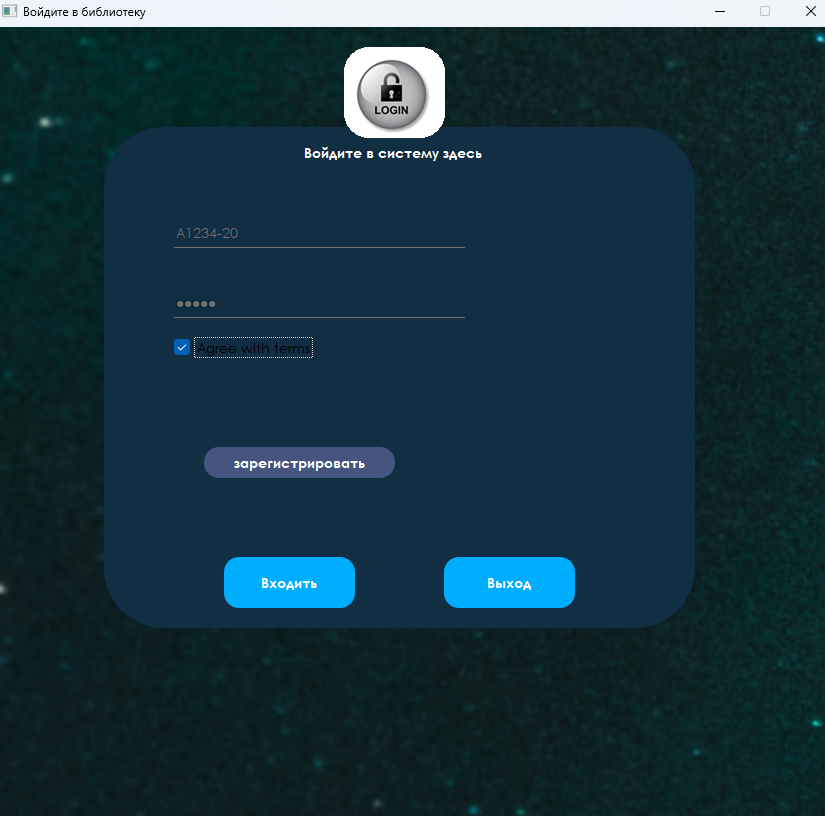


Рисунок 14.2- Логин страница

Б. Как выглядит интерфейс администрирования, когда вы просто загружаете его

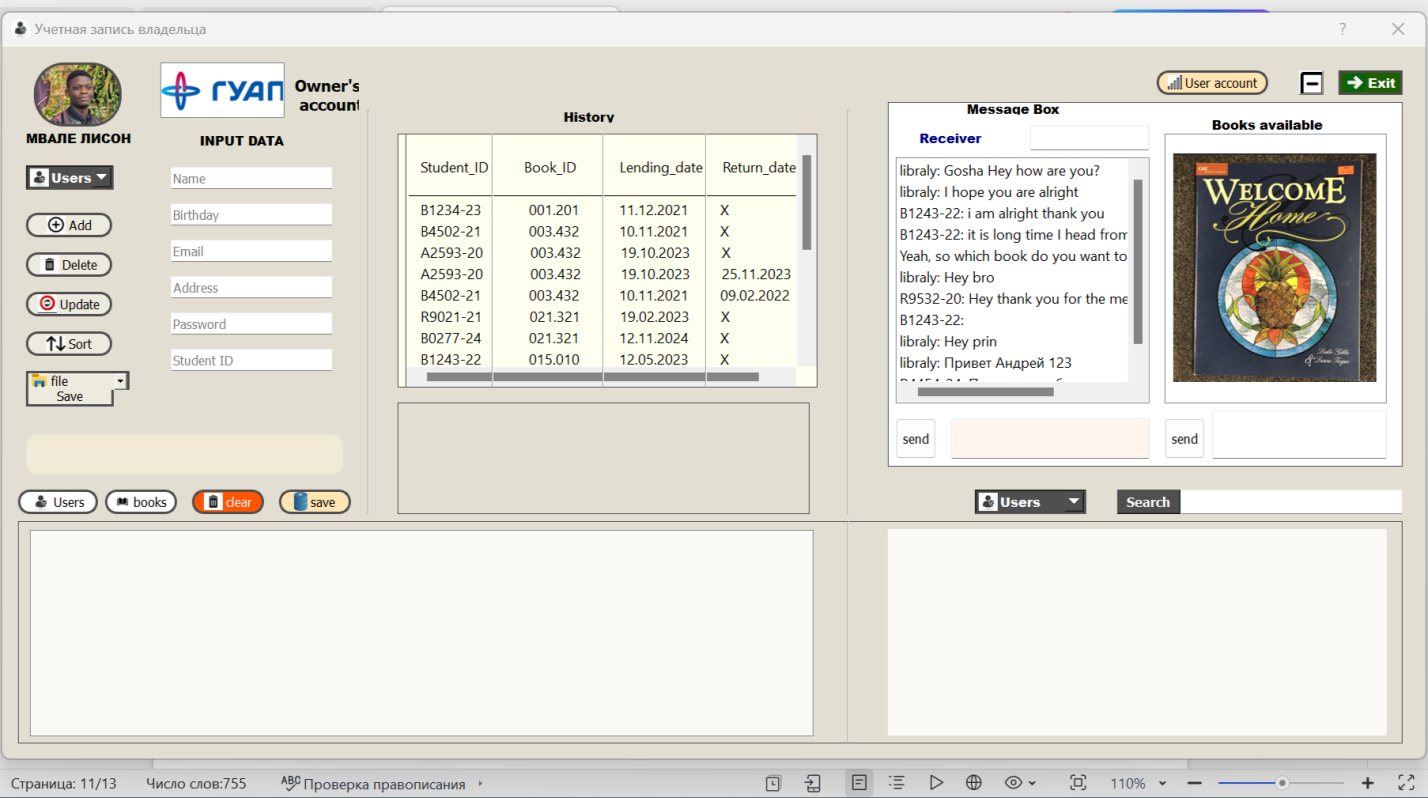


Рисунок 14.2- интерфейса владельца учетной записи

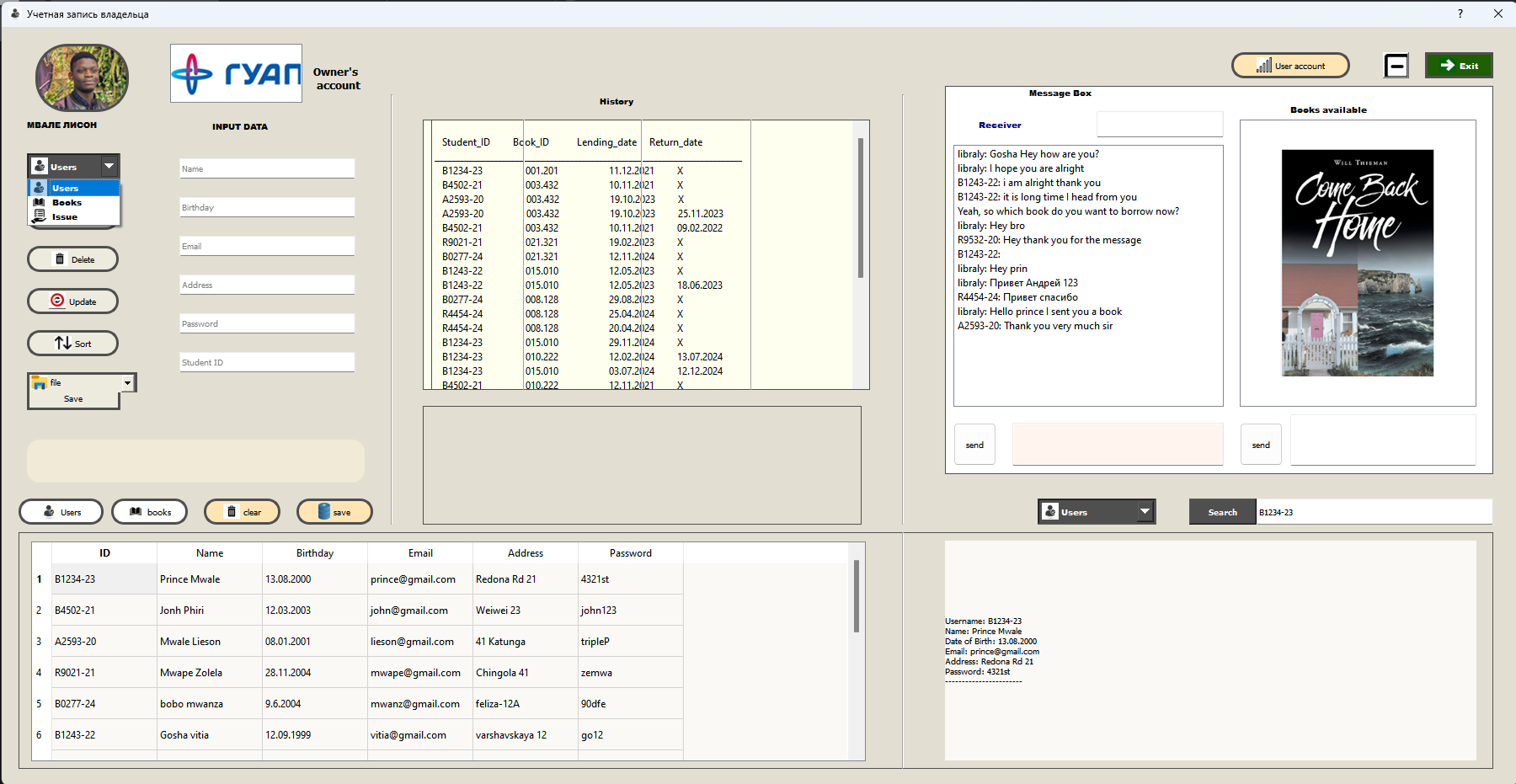


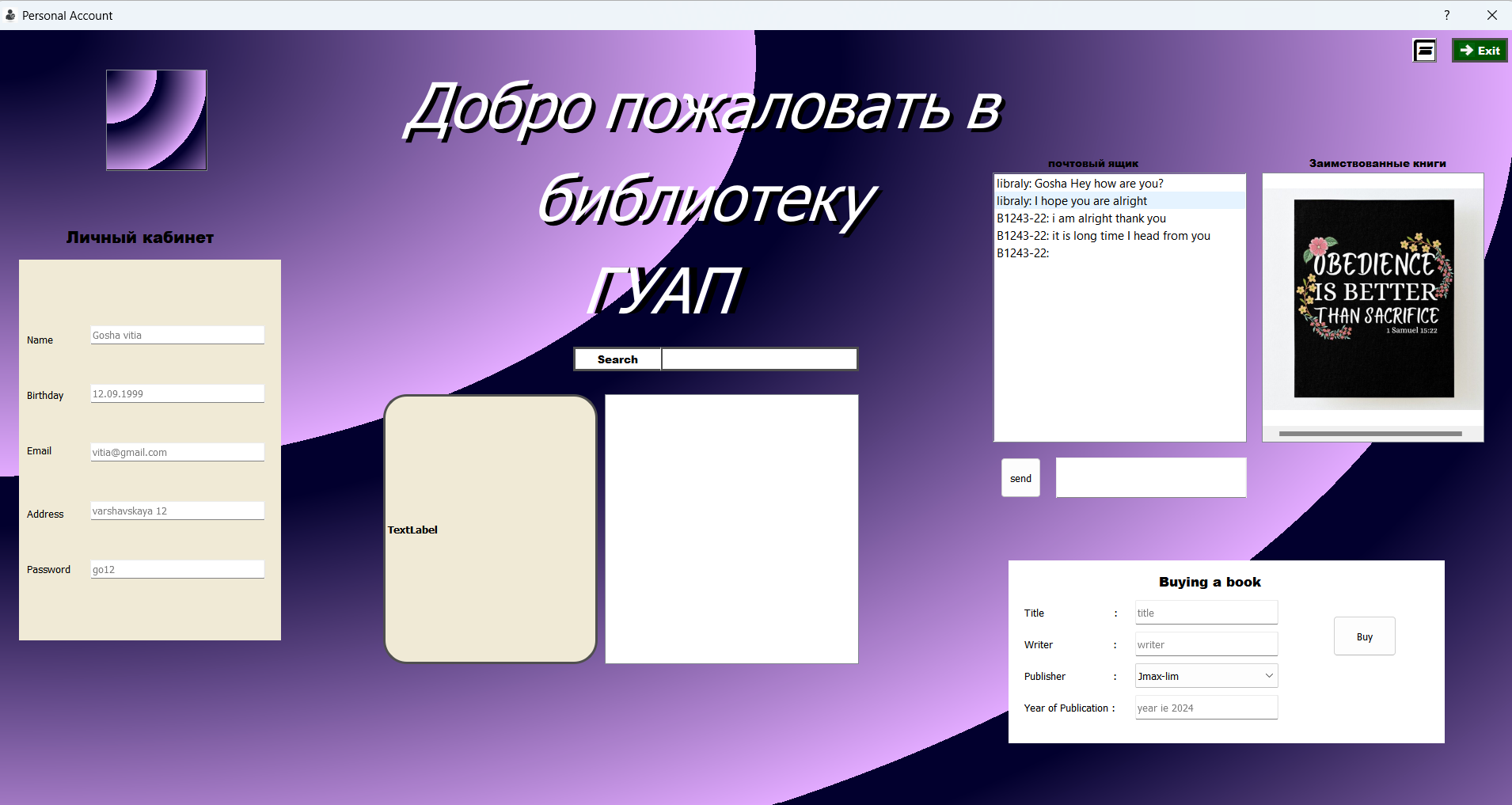
Рисунок 14.3- интерфейса владельца учетной записи

Общие свойства этого интерфейса работают, и теперь я выйду из системы и войду в систему как пользователь

Когда я выхожу, меня возвращают на страницу входа в систему

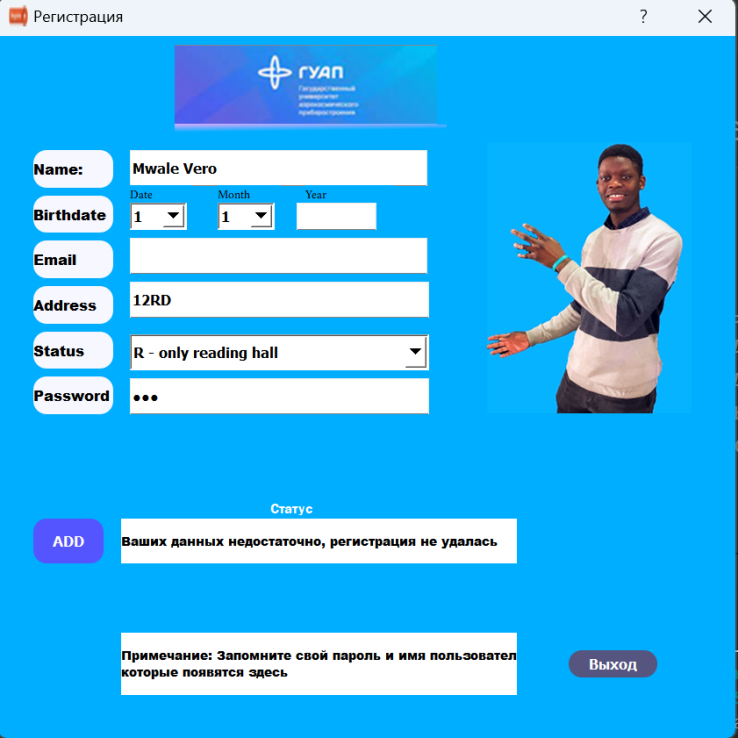


Успешно вошел в систему как пользователь



**4. ТЕСТИРОВАНИЕ**

1. **Интерфейс регистрации**



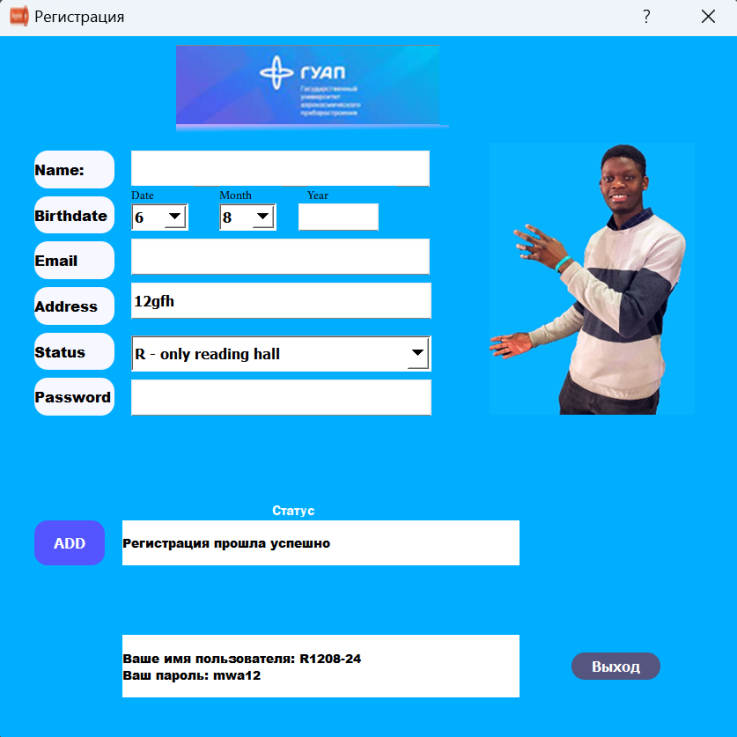


Рисунок -15. тестирование интерфейса регистрации

Из двух приведённых выше скриншотов для регистрации один показывает, что любой неправильный формат ввода данных вызовет уведомление о том, что данные не были сохранены из-за неправильного ввода данных, а на другом скриншоте показан успешный вход с именем пользователя и паролем, которые пользователь должен использовать для входа в систему библиотеки

1. **тестирование интерфейса владельца учётной записи**

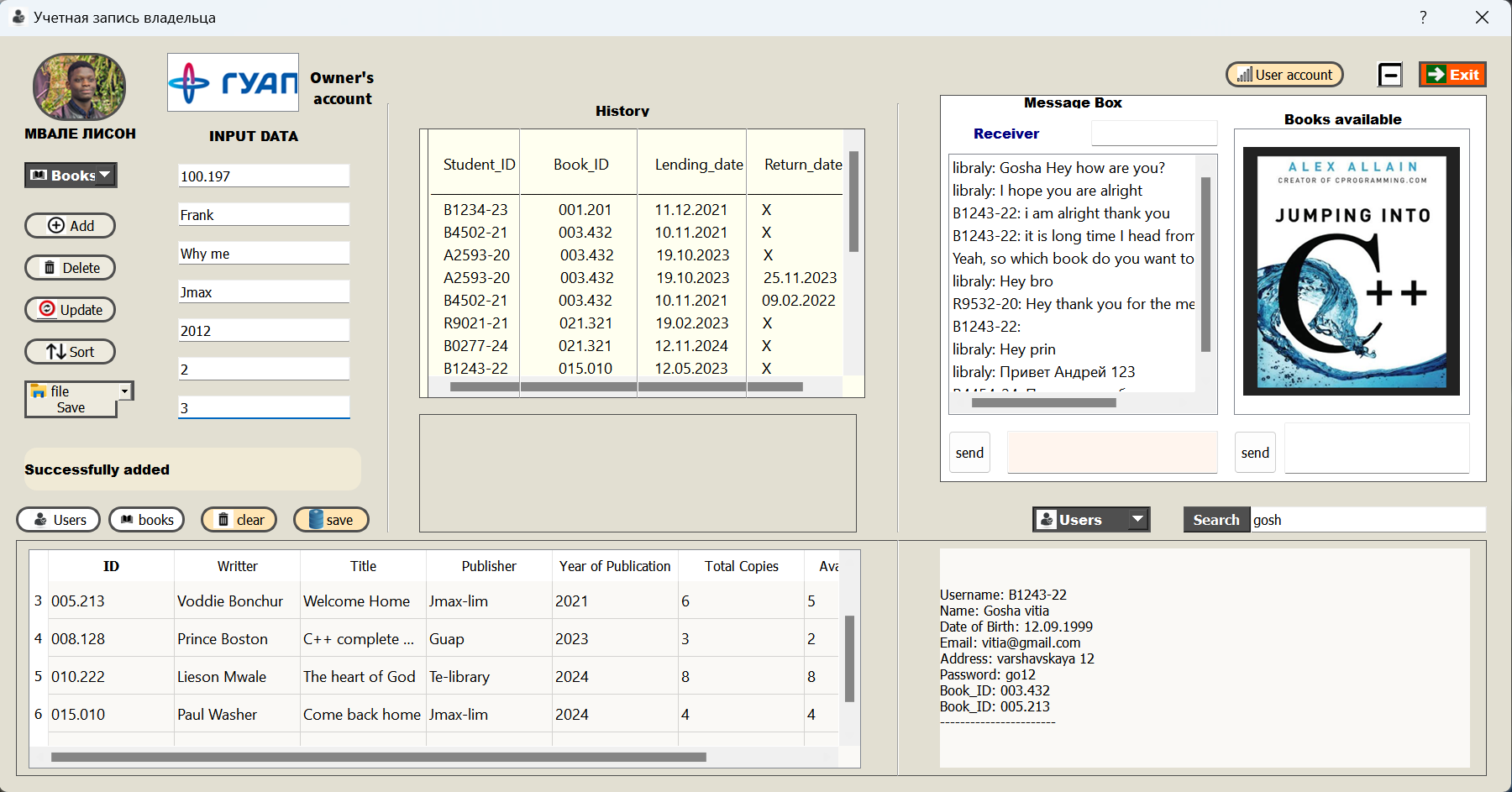


Рисунок 16- *тестирование интерфейса владельца учетной записи*

Скриншоты выше показывают успешное тестирование различных функций на интерфейсе учётной записи владельца. В левом верхнем углу указано, что владелец может форматировать данные для книг, пользователей, а также выдавать книги. Он имеет возможность добавлять, удалять, обновлять, сортировать и сохранять данные о книгах и пользователях. Любой неверный или успешный ввод данных отображается на виджете метки ниже него. Кроме того, владелец может просматривать всех своих пользователей и книги для своей библиотеки, которые отображаются на виджете таблицы в скриншоте. Кроме того, история выдачи и приёма книг хранится в виджете истории, отображённом на интерфейсе. Кроме того, владелец учётной записи успешно может взаимодействовать с различными пользователями, и информация направляется в конкретную учётную запись пользователя. Он также может отправлять изображение книги, которую конкретный пользователь взял в библиотеке, чтобы она отображалась на его учётной записи. Наконец, в режиме поиска владелец может переключаться между поиском деталей пользователей или книг. Пример успешного поиска пользователя по имени Гоша Витя отображается, указывая также, что он взял в библиотеке две разные книги, которые ещё не вернул.

1. **тестирование интерфейса учётной записи пользователя**

Когда пользователь просто выпускает свой интерфейс, показывая предыдущую беседу и книги, которыми он владел в библиотеке

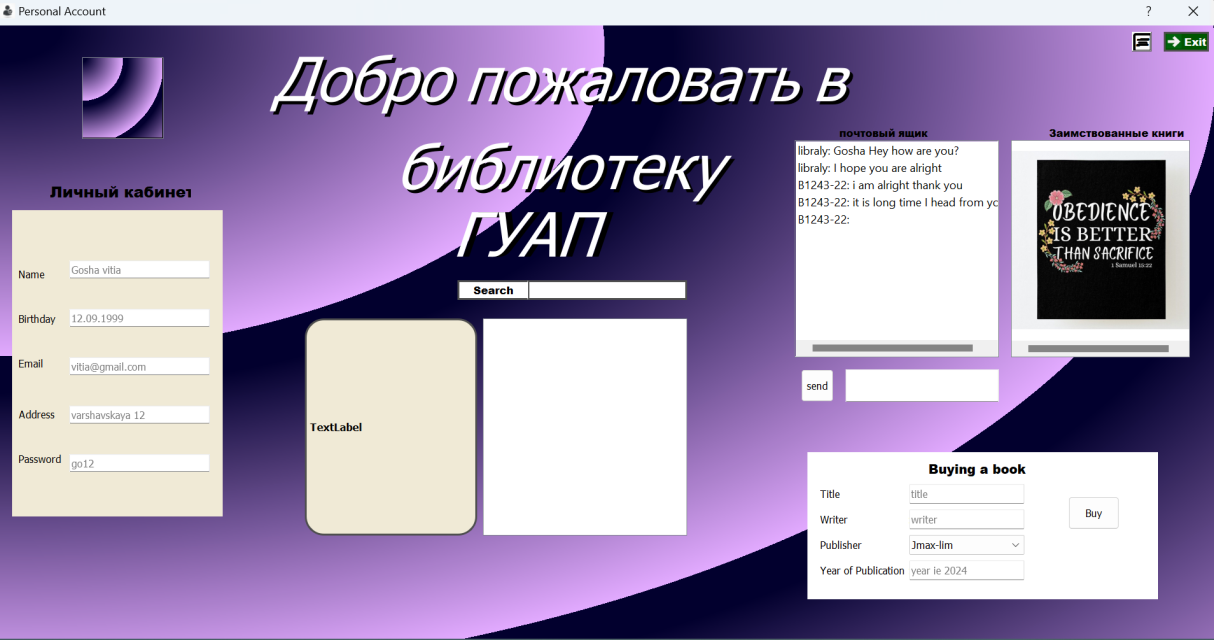


Рисунок 16- тестирование интерфейса учётной записи пользователя

Это показывает, что метод fabric, реализованный для расчёта продаж книг, работает должным образом, и количество книг в библиотеке уменьшается на 1, плюс это показывает, что пользователь может добавить картинку с помощью перетаскивания, а также показывает успешный поиск любой книги Бойера и Мура, для которой не требуется поиск по каталогу. точный ввод слова в поле поиска

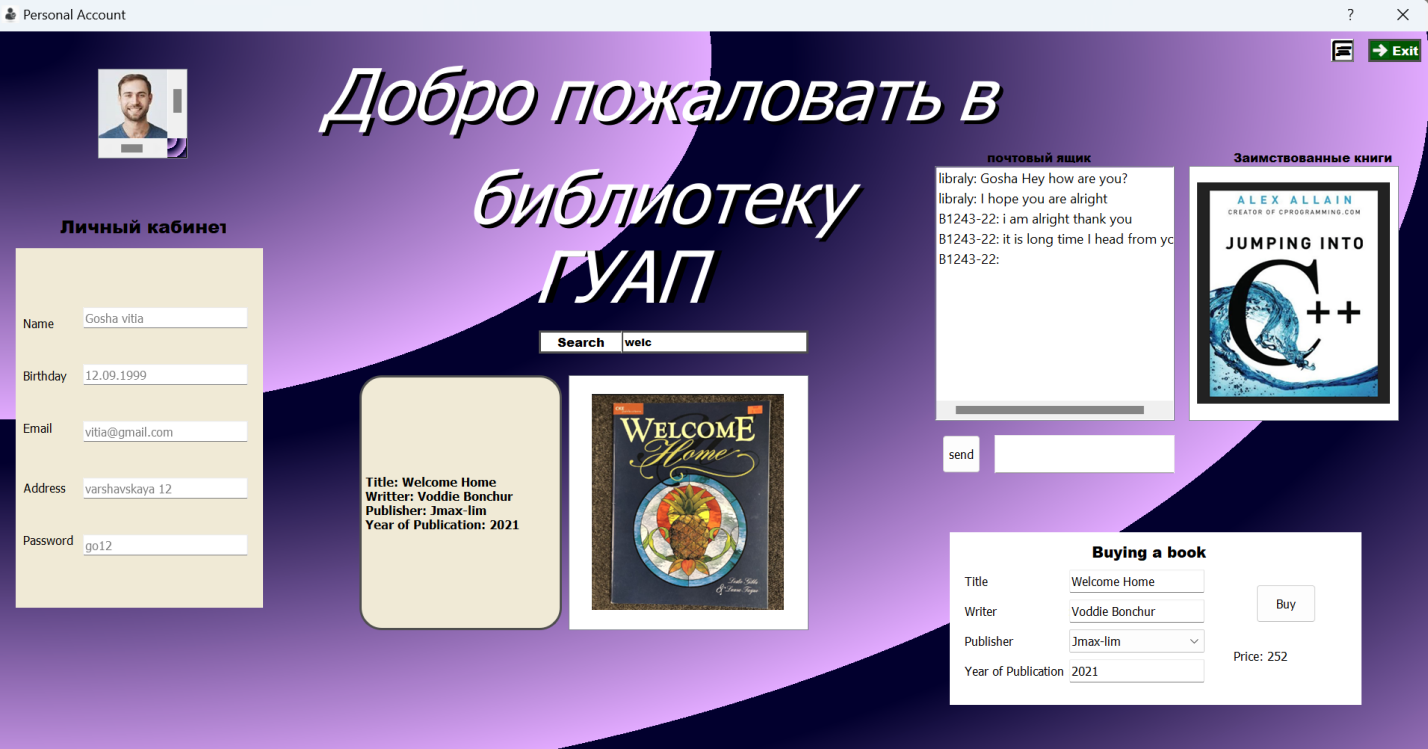


Рисунок 17-тестирование интерфейса учётной записи пользователя получилось

Снимок экрана выше показывает данные индивидуального аккаунта слева от виджета. На этом интерфейсе пользователь может общаться с владельцем библиотеки и просматривать книги, которые он должен библиотеке, справа, где находится виджет графического представления. Кроме того, выделяется функциональность поиска книг в библиотеке с использованием не точного слова, но все равно находящего наиболее близкое совпадение. Этот поиск возвращает наиболее подходящую книгу в библиотеке к тому, что пользователь искал, вместе с изображением этой книги в результатах поиска.

**5. ЗАКЛЮЧЕНИЕ**

Разработка автоматизированной системы управления библиотекой направлена на создание удобного и функционального инструмента для автоматизации библиотечных процессов. В ходе выполнения курсового проекта были решены ключевые задачи:

1. **Анализ предметной области** – изучение процессов учета читателей, книг и выдачи позволило выделить сущности, их атрибуты и взаимосвязи, а также определить функциональные требования.
2. **Разработка структуры системы** – использование объектно-ориентированного подхода обеспечило четкую структуру и возможность дальнейшего расширения.
3. **Реализация функциональности** – система поддерживает основные функции, такие как регистрация читателей и книг, поиск, выдача и возврат, с использованием файловой системы для хранения данных.
4. **Тестирование и проверка** – система успешно прошла тестирование на корректность выполнения операций, включая регистрацию, поиск и обработку данных.
5. **Удобство использования** – интерфейс интуитивно понятен, система предоставляет уведомления об ошибках и подтверждение успешных операций.

### Перспективы развития

Будущее развитие системы может включать внедрение реляционной базы данных, веб-интерфейса для удаленного доступа, модуля аналитики и автоматизированных напоминаний о возврате книг. Разработанная система демонстрирует потенциал для улучшения обслуживания читателей и повышения общей эффективности работы библиотеки.

1. **ПРИЛОЖЕНИЕ С ПОЛНЫМ КОДОМ ПРОГРАММ.**

#ifndef HEADERS\_H

#define HEADERS\_H

#include <iostream>

#include <iomanip>

#include <stack>

#include <vector>

#include <fstream>

#include <sstream>

#include <string>

#include <algorithm>

#include <cctype>

#include <QLabel>

#include <QDebug>

#include <cstdio>

#include <QGraphicsView>

#include <QGraphicsScene>

#include <QDragEnterEvent>

#include <QDragLeaveEvent>

#include <QDragMoveEvent>

#include <QDropEvent>

#include <QWidget>

#include <QMimeData>

#include <QGraphicsPixmapItem>

#include <QPixmap>

#include <QString>

#include <QApplication>

#include <QMessageBox>

#include <cmath>

#include <string.h>

#include <QSqlDatabase>

#include <QSqlQuery>

#include <QVariant>

#include <QTableWidget>

#include <QTableWidgetItem>

#include <QStandardItemModel>

#include <limits>

#include <QMainWindow>

#include <QLineEdit>

#include <QtSql>

#include <QFileInfo>

#include <QDialog>

#include <QTimer>

#include <QTime>

#include <QDesktopServices>

#include <QUrl>

#include <QListWidget>

#include <QFile>

#include <QTextStream>

#include "ownerdialog.h"

#include "mainwindow.h"

#include "ValidateInput.h"

#include "linked\_list.h"

#include "boyer\_moore.h"

#include "hashtable.h"

#include "abv\_tree.h"

#include "registerdialog.h"

#include "portaldialog.h"

#ifndef ABSTRACTCALC\_H

#define ABSTRACTCALC\_H

#include "book\_data.h"

*class* **AbstractCalc**

{

*public*:

*virtual* ~***AbstractCalc***() {}

*virtual* int ***getCost***(Book\_data \*value) = 0;

};

#endif *//* *ABSTRACTCALC\_H*

#ifndef BOOK\_DATA\_H

#define BOOK\_DATA\_H

#include <QObject>

#include <QString>

*class* Book\_data: *public* QObject

{

Q\_OBJECT

*public*:

*enum* publisherType{

Jmax\_lim,

Ted\_talks,

Guap,

Te\_library,

UNZA

};

*explicit* Book\_data(QObject \*parent =*nullptr*);

Book\_data(QString title, QString writer, publisherType type, int year);

bool check(QString title);

QString gettitle() *const*;

QString getwriter() *const*;

int getyear() *const*;

publisherType getType() *const*;

*private*:

QString title;

QString writer;

int year;

publisherType type;

};

#endif *//* *BOOK\_DATA\_H*

#include "book\_data.h"

#include "headers.h"

Book\_data::Book\_data(QObject \*parent)

: QObject(parent), title(""), writer(""), year(0), type(Jmax\_lim) {}

Book\_data::Book\_data(QString title, QString writer, publisherType type, int year)

: title(title), writer(writer), year(year), type(type) {}

bool Book\_data::check(QString title) {

MainWindow conn;

*if* (!conn.connOpen()) {

qDebug() << "Failed to open the database when reading for book data";

*return* *false*;

}

QSqlQuery qry;

qry.prepare("SELECT \* FROM books\_table WHERE title=:title");

qry.bindValue(":title", title);

*if* (qry.exec() && qry.next()) {

qDebug() << "Book found in the database";

int totalNumofCopies = qry.value("totalNumofCopies").toInt();

int numofCopiesAvail = qry.value("NumofCopiesAvail").toInt();

*//* *Decrease* *the* *values* *by* *1*

totalNumofCopies -= 1;

numofCopiesAvail -= 1;

*//* *Update* *the* *database* *with* *the* *new* *values*

QSqlQuery updateQry;

updateQry.prepare("UPDATE books\_table SET totalNumofCopies=:totalNumofCopies, NumofCopiesAvail=:NumofCopiesAvail WHERE title=:title");

updateQry.bindValue(":totalNumofCopies", totalNumofCopies);

updateQry.bindValue(":NumofCopiesAvail", numofCopiesAvail);

updateQry.bindValue(":title", title);

*if* (updateQry.exec()) {

qDebug() << "Book copies updated successfully";

conn.connClose(); *//* *Close* *the* *connection*

*return* *true*;

} *else* {

qDebug() << "Failed to update book copies: " << updateQry.lastError().text();

conn.connClose(); *//* *Close* *the* *connection*

*return* *false*;

}

} *else* {

qDebug() << "Book not found in the database";

*if* (qry.lastError().isValid()) {

qDebug() << "Error: " << qry.lastError().text();

}

conn.connClose(); *//* *Close* *the* *connection*

*return* *false*;

}

}

QString Book\_data::gettitle() *const*

{

*return* title;

}

QString Book\_data::getwriter() *const*

{

*return* writer;

}

int Book\_data::getyear() *const*

{

*return* year;

}

Book\_data::publisherType Book\_data::getType() *const*

{

*return* type;

}

*//* *calcfactory.h*

#ifndef CALCFACTORY\_H

#define CALCFACTORY\_H

*class* **AbstractCalc**;

*class* **CalcFactory**

{

*public*:

*virtual* AbstractCalc\* ***createCalculator***() = 0;

*virtual* ~***CalcFactory***() {}

};

#endif *//* *CALCFACTORY\_H*

*//* *calcfactoryimpl.h*

#ifndef CALCFACTORYIMPL\_H

#define CALCFACTORYIMPL\_H

#include "headers.h"

*class* **CalcFactoryImpl** : *public* CalcFactory

{

*public*:

**CalcFactoryImpl**(AbstractCalc \*calculator);

AbstractCalc\* ***createCalculator***() *override*;

*private*:

AbstractCalc \*calculator;

};

#endif *//* *CALCFACTORYIMPL\_H*

#include "calcfactoryimpl.h"

CalcFactoryImpl::**CalcFactoryImpl**(AbstractCalc \*calculator)

: calculator(calculator)

{}

AbstractCalc\* CalcFactoryImpl::***createCalculator***()

{

*return* calculator;

}

*//* *calculationfacade.h*

#ifndef CALCULATIONFACADE\_H

#define CALCULATIONFACADE\_H

*class* **AbstractCalc**;

*class* **CalcFactory**;

#include "headers.h"

*class* **Book\_data**;

*class* **CalculationFacade**

{

*public*:

*static* int **getCost**(Book\_data \*value, CalcFactory \*factory);

};

#endif *//* *CALCULATIONFACADE\_H*

#include "calculationfacade.h"

int CalculationFacade::getCost(Book\_data \*value, CalcFactory \*factory)

{

AbstractCalc \*calculator = factory->createCalculator();

int cost = calculator->getCost(value);

*delete* calculator;

*return* cost;

}

#ifndef GUAPCALC\_H

#define GUAPCALC\_H

#include "headers.h"

*class* **Guapcalc**: *public* AbstractCalc

{

*public*:

int ***getCost***(Book\_data \*value) *override*;

};

#endif *//* *GUAPCALC\_H*

#include "guapcalc.h"

int Guapcalc::***getCost***(Book\_data \*value)

{

*return* value->getyear()/30+9;

}

#ifndef JMAX\_LIMCALC\_H

#define JMAX\_LIMCALC\_H

#include "headers.h"

*class* **Jmax\_limcalc**: *public* AbstractCalc

{

*public*:

int ***getCost***(Book\_data \*value) *override*;

};

#endif *//* *JMAX\_LIMCALC\_H*

#include "jmax\_limcalc.h"

int Jmax\_limcalc::getCost(Book\_data \*value)

{

*return* value->getyear() / 10 +50 ;

}

#ifndef TE\_LIBRARYCALC\_H

#define TE\_LIBRARYCALC\_H

#include "headers.h"

*class* Te\_librarycalc : *public* AbstractCalc

{

*public*:

int getCost(Book\_data \*value) override;

};

#endif *//* *TE\_LIBRARYCALC\_H*

#include "te\_librarycalc.h"

int Te\_librarycalc::getCost(Book\_data \*value)

{

*return* value->getyear()/120+20;

}

#ifndef TED\_TALKSCALC\_H

#define TED\_TALKSCALC\_H

#include "headers.h"

*class* **Ted\_talkscalc** : *public* AbstractCalc

{

*public*:

int ***getCost***(Book\_data \*value) *override*;

};

#endif *//* *TED\_TALKSCALC\_H*

#include "ted\_talkscalc.h"

int Ted\_talkscalc::getCost(Book\_data \*value)

{

*return* value->getyear()/5 +20;

}

#ifndef UNZA\_H

#define UNZA\_H

#include "abstractcalc.h"

*class* UNZA : *public* AbstractCalc

{

*public*:

int getCost(Book\_data \*value) override;

};

#endif *//* *UNZA\_H*

#include "unza.h"

int UNZA::getCost(Book\_data \*value)

{

*return* value->getyear()/200+70;

}

#endif *//* *HEADERS\_H*

#ifndef HASHTABLE\_H

#define HASHTABLE\_H

#include "headers.h"

*const* int TABLE\_SIZE = 20;

*const* int SIZE = 10;

*struct* **Datatype**;

*struct* **Node**;

*struct* **NewNode**;

*struct* **link**{

int num=0;

std::string NNNMMM;

link\* next = *nullptr*;

};

*struct* **HashNode** {

int key;

std::string username;

std::string name;

std::string birthDate;

std::string email;

std::string address;

std::string password;

link\* bookstatus;

bool occupied;

**HashNode**() : key(-1), occupied(*false*) {}

};

*class* **HashTable** {

*private*:

HashNode table[TABLE\_SIZE];

HashNode fail[SIZE];

*//* *Hash* *function* *using* *quadratic* *probing*

int **hash**(int key, int a);

*public*:

void **insert**(HashNode data);

void **search**(int key);

void **remove**(int key);

void **add**(std::string key,std::string bookID, int num);

void **clear**();

void **displayRow**(QStandardItemModel\* model, *const* HashNode& data);

void **display**(QTableView\* tableView);

bool **isDataPresent**(int key);

void **downloadToFile**(std::string& fileName);

*//void* *uploadFromFile(const* *std::string&* *fileName,const* *std::string&* *pattern);*

void **retrieveFromDatabase**(HashTable& TABLE);

bool **isUsernamePresent**(*const* std::string& username);

bool **editByUsername**(*const* std::string& username, Datatype newData);

void **freehashtablelinkedlist**();

void **deleteLink**(link\*& head, *const* std::string& NNNMMM);

bool **deleteLinkByUsername**(*const* std::string& username, *const* std::string& NNNMMM);

void **displayUserDataHelper**(*const* HashNode& node, QLabel\* label, Node\* head,std::string status,std::string book);

void **displayUserData**(QLabel\* label, Node\* root, std::string status,std::string book);

void **uploadFromFile**(*const* std::string& fileName) ;

};

#endif *//* *HASHTABLE\_H*

#include "hashtable.h"

int HashTable::**hash**(int key, int a) {

*return* (key + a \* a) % TABLE\_SIZE;

}

void HashTable::**insert**(HashNode data) {

int attempt = 0;

*static* int i = 0;

int index;

*//* *Check* *the* *entire* *hash* *table* *for* *an* *existing* *username*

*for* (int j = 0; j < TABLE\_SIZE; ++j) {

*if* (table[j].occupied && table[j].username == data.username) {

*//* *User* *with* *the* *same* *username* *already* *exists,* *skip* *insertion*

*return*;

}

}*for* (int j = 0; j < SIZE; ++j) {

*if* (fail[j].occupied && fail[j].username == data.username) {

*//* *User* *with* *the* *same* *username* *already* *exists,* *skip* *insertion*

*return*;

}

}

*do* {

index = hash(data.key, attempt);

*if* (!table[index].occupied) {

table[index].key = data.key;

table[index].username = data.username;

table[index].name = data.name;

table[index].birthDate = data.birthDate;

table[index].email = data.email;

table[index].address = data.address;

table[index].password = data.password;

table[index].occupied = *true*;

*//QMessageBox::information(nullptr,* *"Insert",* *"Key* *",* *QString::fromStdString(data.username));*

*return*;

}

attempt++;

} *while* (attempt < TABLE\_SIZE);

*//QMessageBox::warning(nullptr,* *"Insert",* *"Unable* *to* *insert* *key.* *Hash* *table* *is* *full.");*

*//handling* *collision*

*if* (i < SIZE && !fail[i].occupied) {

fail[i].key = data.key;

fail[i].username = data.username;

fail[i].name = data.name;

fail[i].birthDate = data.birthDate;

fail[i].email = data.email;

fail[i].address = data.address;

fail[i].password = data.password;

fail[i].occupied = *true*;

i++;

*return*;

}

}

void HashTable::**search**(int key) {

int attempt = 0;

int index;

*do* {

index = hash(key, attempt);

*if* (table[index].key == key && table[index].occupied) {

std::cout << "Key " << key << " found at index " << index << std::endl;

*return*;

}

attempt++;

} *while* (attempt < TABLE\_SIZE);

*for* (int i = 0; i < SIZE; i++) {

*if* (fail[i].key == key && fail[i].occupied) {

std::cout << "Key " << key << " found at index " << i << std::endl;

*return*;

}

}

std::cout << "Key " << key << " not found" << std::endl;

}

void HashTable::**remove**(int key) {

int attempt = 0;

int index;

*do* {

index = hash(key, attempt);

*if* (table[index].key == key && table[index].occupied) {

table[index].occupied = *false*;

std::cout << "Key " << key << " deleted from index " << index << std::endl;

*return*;

}

attempt++;

} *while* (attempt < TABLE\_SIZE);

*for* (int i = 0; i < SIZE; i++) {

*if* (fail[i].key == key && fail[i].occupied) {

fail[i].occupied = *false*;

std::cout << "Key " << key << " deleted from index " << i << std::endl;

*return*;

}

}

std::cout << "Key " << key << " not found in the hash table." << std::endl;

}

void HashTable::**add**(std::string key,std::string bookID, int num)

{

*for* (int i = 0; i < TABLE\_SIZE; i++) {

*if* (table[i].username == key && table[i].occupied) {

*//* *Create* *a* *new* *link* *with* *the* *new* *data*

link\* newLink = *new* link;

newLink->num = num;

newLink->NNNMMM = bookID;

*//* *Add* *the* *new* *link* *to* *the* *beginning* *of* *the* *linked* *list*

newLink->next = table[i].bookstatus;

table[i].bookstatus = newLink;

*return*;

}

}

*for* (int i = 0; i < SIZE; i++) {

*if* (fail[i].username == key && fail[i].occupied) {

*//* *Create* *a* *new* *link* *with* *the* *new* *data*

link\* newLink = *new* link;

newLink->num = num;

newLink->NNNMMM = bookID;

*//* *Add* *the* *new* *link* *to* *the* *beginning* *of* *the* *linked* *list*

newLink->next = fail[i].bookstatus;

fail[i].bookstatus = newLink;

*return*;

}

}

std::cout << "Key " << key << " not found" << std::endl;

}

void HashTable::**displayRow**(QStandardItemModel\* model, *const* HashNode& data) {

QList<QStandardItem\*> items;

items << *new* QStandardItem(QString::fromStdString(data.username))

<< *new* QStandardItem(QString::fromStdString(data.name))

<< *new* QStandardItem(QString::fromStdString(data.birthDate))

<< *new* QStandardItem(QString::fromStdString(data.email))

<< *new* QStandardItem(QString::fromStdString(data.address))

<< *new* QStandardItem(QString::fromStdString(data.password));

model->appendRow(items);

}

void HashTable::**display**(QTableView\* tableView) {

QStandardItemModel\* model = *new* QStandardItemModel(0, 6, *tableView*);

tableView->*setModel*(*model*);

*//* *Set* *horizontal* *header* *labels*

model->setHorizontalHeaderLabels({"ID", "Name", "Birthday", "Email", "Address","Password" });

*//* *Display* *data* *from* *the* *main* *table*

*for* (int i = 0; i < TABLE\_SIZE; i++) {

*if* (table[i].occupied) {

displayRow(*model*, table[i]);

}

}

*//* *Display* *data* *from* *the* *fail* *table*

*for* (int i = 0; i < SIZE; i++) {

*if* (fail[i].occupied) {

displayRow(*model*, fail[i]);

}

}

}

bool HashTable::**isDataPresent**(int key) {

int attempt = 0;

int index;

*//* *Check* *the* *main* *table* *using* *quadratic* *probing*

*do* {

index = hash(key, attempt);

*if* (table[index].occupied && table[index].key == key) {

*return* *true*;

}

attempt++;

} *while* (attempt < TABLE\_SIZE);

attempt = 0;

*//* *Check* *the* *fail* *table* *using* *quadratic* *probing*

*do* {

index = hash(key, attempt);

*if* (fail[index].occupied && fail[index].key == key) {

*return* *true*;

}

attempt++;

} *while* (attempt < SIZE);

*return* *false*;

}

void HashTable::**clear**() {

*for* (int i = 0; i < TABLE\_SIZE; i++) {

table[i].occupied = *false*;

}

*for* (int i = 0; i < SIZE; i++) {

fail[i].occupied = *false*;

}

}

void HashTable::**downloadToFile**(std::string& fileName) {

std::ofstream outputFile(fileName, std::ios::out);

*//* *Write* *data* *from* *the* *main* *table*

*for* (int i = 0; i < TABLE\_SIZE; i++) {

*if* (table[i].occupied) {

outputFile << table[i].username << ","

<< table[i].name << ","

<< table[i].birthDate << ","

<< table[i].email << ","

<< table[i].address<< ","

<< table[i].password;

link\* currentLink = table[i].bookstatus;

*while* (currentLink != *nullptr*) {

outputFile << "," << currentLink->NNNMMM;

currentLink = currentLink->next;

}

outputFile << std::endl;

}

}

*//* *Write* *data* *from* *the* *fail* *table*

*for* (int i = 0; i < SIZE; i++) {

*if* (fail[i].occupied) {

outputFile << fail[i].username << ","

<< fail[i].name << ","

<< fail[i].birthDate << ","

<< fail[i].email << ","

<< fail[i].address << ","

<< fail[i].password ;

link\* currentLink = fail[i].bookstatus;

*while* (currentLink != *nullptr*) {

outputFile << "," << currentLink->NNNMMM;

currentLink = currentLink->next;

}

outputFile << std::endl;

}

}

outputFile.close();

}

void HashTable::**uploadFromFile**(*const* std::string& fileName) {

std::ifstream inputFile(fileName);

*if* (!inputFile) {

qDebug() << "Error opening file.\n";

*return*;

}

std::string line;

*while* (std::getline(inputFile, *line*)) {

*//* *Parse* *comma-separated* *values* *from* *the* *line*

std::istringstream iss(line);

std::string username, name, birthDate, email, address, password;

std::getline(iss, *username*, ',');

std::getline(iss, *name*, ',');

std::getline(iss, *birthDate*, ',');

std::getline(iss, *email*, ',');

std::getline(iss, *address*, ',');

std::getline(iss, *password*, ',');

*//* *Create* *a* *new* *HashNode* *and* *copy* *parsed* *data*

HashNode newNode;

newNode.username = username;

newNode.name = name;

newNode.birthDate = birthDate;

newNode.email = email;

newNode.address = address;

newNode.password = password;

*//* *Initialize* *linked* *list* *to* *nullptr*

newNode.bookstatus = *nullptr*;

*//* *Read* *and* *insert* *book* *statuses* *from* *the* *line*

std::string bookStatus;

*while* (std::getline(iss, *bookStatus*, ',')) {

*//* *Append* *book* *status* *to* *linked* *list*

link\* newLink = *new* link;

newLink->NNNMMM = bookStatus;

newLink->next = newNode.bookstatus; *//* *Insert* *at* *the* *beginning*

newNode.bookstatus = newLink;

}

*//* *Insert* *the* *new* *node* *into* *the* *hash* *table*

insert(newNode); *//* *Assuming* *insertIntoHashTable* *exists*

qDebug() << "Data inserted from file:" << QString::fromStdString(line);

}

inputFile.close();

}

void HashTable::**retrieveFromDatabase**(HashTable& TABLE) {

MainWindow conn;

QSqlQuery \*qry = *new* QSqlQuery(conn.mydbase);

qry->prepare("select \* from login\_data");

qry->exec();

*while* (qry->next()) {

HashNode data;

data.key = qry->value("ID").toInt();

data.username = qry->value("ID").toString().toStdString();

data.name = qry->value("Name").toString().toStdString();

data.birthDate = qry->value("Birthday").toString().toStdString();

data.email = qry->value("Email").toString().toStdString();

data.address = qry->value("Address").toString().toStdString();

data.password = qry->value("Password").toString().toStdString();

TABLE.insert(data);

}

conn.connClose();

}

bool HashTable::**isUsernamePresent**(*const* std::string& username) {

*//* *Check* *the* *main* *table*

*for* (int i = 0; i < TABLE\_SIZE; ++i) {

*if* (table[i].occupied && table[i].username == username) {

*//table[i].bookstatus->num++;*

*return* *true*;

}

}

*//* *Check* *the* *fail* *table*

*for* (int i = 0; i < SIZE; ++i) {

*if* (fail[i].occupied && fail[i].username == username) {

*//fail[i].bookstatus->num++;*

*return* *true*;

}

}

*return* *false*;

}

void **addLink**(link\*& head, *const* std::string& NNNMMM) {

link\* newLink = *new* link;

newLink->NNNMMM = NNNMMM;

newLink->next = head;

head = newLink;

}

void **deleteLinkedList**(link\*& head) {

*while* (head != *nullptr*) {

link\* temp = head;

head = head->next;

*delete* temp;

}

}

bool HashTable::**editByUsername**(*const* std::string& username, Datatype newData) {

*//* *Check* *the* *main* *table*

*for* (int i = 0; i < TABLE\_SIZE; ++i) {

*if* (table[i].occupied && table[i].username == username) {

*//* *Found* *the* *entry,* *update* *the* *data*

addLink(*table[i].bookstatus*, newData.book\_ID);

table[i].bookstatus->num++;

*return* *true*;

}

}

*//* *Check* *the* *fail* *table*

*for* (int i = 0; i < SIZE; ++i) {

*if* (fail[i].occupied && fail[i].username == username) {

*//* *Found* *the* *entry,* *update* *the* *data*

addLink(*fail[i].bookstatus*, newData.book\_ID);

fail[i].bookstatus->num++;

*return* *true*;

}

}

*//* *The* *specified* *username* *was* *not* *found*

*return* *false*;

}

void HashTable::**freehashtablelinkedlist**()

{

*for* (int i = 0; i < TABLE\_SIZE; ++i) {

deleteLinkedList(*table[i].bookstatus*);

}

*for* (int i = 0; i < SIZE; ++i) {

deleteLinkedList(*fail[i].bookstatus*);

}

}

void HashTable::**deleteLink**(link\*& head, *const* std::string& NNNMMM) {

link\* current = head;

link\* previous = *nullptr*;

*while* (current != *nullptr*) {

*if* (current->NNNMMM == NNNMMM) {

*if* (previous == *nullptr*) {

*//* *Node* *to* *be* *deleted* *is* *the* *head*

head = current->next;

} *else* {

previous->next = current->next;

}

*delete* current;

*return*;

}

previous = current;

current = current->next;

}

}

bool HashTable::**deleteLinkByUsername**(*const* std::string& username, *const* std::string& NNNMMM) {

*//* *Check* *the* *main* *table*

*for* (int i = 0; i < TABLE\_SIZE; ++i) {

*if* (table[i].occupied && table[i].username == username) {

*//* *Found* *the* *entry,* *update* *the* *data*

deleteLink(*table[i].bookstatus*, NNNMMM);

*return* *true*;

}

}

*//* *Check* *the* *fail* *table*

*for* (int i = 0; i < SIZE; ++i) {

*if* (fail[i].occupied && fail[i].username == username) {

*//* *Found* *the* *entry,* *update* *the* *data*

deleteLink(*fail[i].bookstatus*, NNNMMM);

*return* *true*;

}

}

*//* *The* *specified* *username* *was* *not* *found*

*return* *false*;

}

void **displayBookData**(*const* Node\* bookNode, QLabel\* label) {

*if* (bookNode != *nullptr*) {

*//* *Display* *book* *data* *on* *QLabel*

std::string bookData = "Title: " + bookNode->title +

"\nID: " + bookNode->id +

"\nWritter: " + bookNode->writter +

"\nPublisher: " + bookNode->publisher +

"\nYear of Publication: " + bookNode->year\_publication +

"\nTotal Number of Copies: " + std::to\_string(bookNode->totalNumberOfCopies) +

"\nCopies Available: " + std::to\_string(bookNode->copiesAvailable);

label->setText(QString::fromStdString(bookData));

} *else* {

*//* *If* *the* *title* *is* *not* *found,* *display* *a* *message*

label->setText("Book with the specified title not found.");

}

}

void HashTable::**displayUserDataHelper**(*const* HashNode& node, QLabel\* label, Node\* root, std::string status,std::string book ) {

*if*(status == "Users"){

std::string userData = "Username: " + node.username +

"\nName: " + node.name +

"\nBirth Date: " + node.birthDate +

"\nEmail: " + node.email +

"\nAddress: " + node.address +

"\nPassword: " + node.password;

*//* *Display* *linked* *list* *data*

link\* currentLink = node.bookstatus;

*while* (currentLink != *nullptr*) {

userData += "\nBookStatus " + std::to\_string(currentLink->num) + ": " + currentLink->NNNMMM;

currentLink = currentLink->next;

}

label->setText(QString::fromStdString(userData));

}*else*{

label->setText("I got her");

Node \*data = searchbytitle(*root*, book);

displayBookData(data, *label*);

}

}

void HashTable::**displayUserData**(QLabel\* label, Node\* root, std::string status, std::string book) {

*if*(status == "Books") { *//* *Check* *if* *the* *status* *is* *for* *displaying* *book* *data*

Node\* data = searchbytitle(*root*, book); *//* *Search* *for* *the* *book* *data*

*if*(data != *nullptr*) { *//* *If* *book* *data* *is* *found*

*//* *Call* *the* *helper* *function* *to* *display* *book* *data*

displayBookData(data, *label*);

} *else* { *//* *If* *book* *data* *is* *not* *found*

label->setText("Book not found"); *//* *Set* *label* *text* *to* *indicate* *that* *the* *book* *was* *not* *found*

}

} *else* { *//* *If* *status* *is* *not* *for* *displaying* *book* *data*

label->setText("Invalid status"); *//* *Set* *label* *text* *to* *indicate* *invalid* *status*

}

}

#ifndef AVL\_TREE\_H

#define AVL\_TREE\_H

#include "headers.h"

*class* **MainWindow** ;

*struct* **Datatype**;

*struct* **NewNode** {

std::string id,writter, title, publisher, year\_publication;

int key,totalNumberOfCopies, copiesAvailable;

};

*struct* **Node**

{

int key;

std::string id, writter, title, publisher, year\_publication;

int totalNumberOfCopies, copiesAvailable;

Node\* left;

Node\* right;

int height;

};

int **height**(Node\* N);

int **max**(int a, int b);

Node\* **newNode**(NewNode a);

Node\* **rightRotate**(Node\* y);

Node\* **leftRotate**(Node\* x);

int **getBalance**(Node\* N);

Node\* **insert**(Node\* node, NewNode a);

void **preOrder**(Node\* root);

int **extractFirstThreeDigits**(*const* QString &input);

bool **validateFormat**(QString& input);

Node \***deleteNode**(Node \*root, int key);

void **inOrderTraverse**(Node\* node, std::vector<NewNode>& nodeList);

void **insertNodeIntoDatabase**(Node\* node, MainWindow& conn, std::ofstream& outFile);

void **Traverse**(Node\* root, MainWindow& conn, std::ofstream& outFile);

void **downloadDataToDatabase**(Node\* root) ;

void **deleteAllBooksData**();

Node\* **search**(Node\* root, int key);

Node\* **searchbytitle**(Node\* root, *const* std::string& title);

bool **searchByStringID**(Node\*& root, *const* Datatype& id,std::string& file);

#endif *//* *AVL\_TREE\_H*

#include "abv\_tree.h"

int **height**(Node\* N)

{

*if* (N == *nullptr*)

*return* 0;

*return* N->height;

}

int **max**(int a, int b)

{

*return* (a > b) ? a : b;

}

Node\* **newNode**(NewNode a)

{

Node\* node = *new* Node();

node->id = a.id;

node->key = a.key;

node->writter = a.writter;

node->title = a.title;

node->publisher = a.publisher;

node->year\_publication = a.year\_publication;

node->totalNumberOfCopies = a.totalNumberOfCopies;

node->copiesAvailable = a.copiesAvailable;

node->left = NULL;

node->right = NULL;

node->height = 1; *//* *new* *node* *is* *initially*

*//* *added* *at* *leaf*

*return*(node);

}

Node\* **rightRotate**(Node\* y)

{

Node\* x = y->left;

Node\* T2 = x->right;

*//* *Perform* *rotation*

x->right = y;

y->left = T2;

*//* *Update* *heights*

y->height = max(height(*y->left*),

height(*y->right*)) + 1;

x->height = max(height(*x->left*),

height(*x->right*)) + 1;

*//* *Return* *new* *root*

*return* x;

}

Node\* **leftRotate**(Node\* x)

{

Node\* y = x->right;

Node\* T2 = y->left;

*//* *Perform* *rotation*

y->left = x;

x->right = T2;

*//* *Update* *heights*

x->height = max(height(*x->left*),

height(*x->right*)) + 1;

y->height = max(height(*y->left*),

height(*y->right*)) + 1;

*//* *Return* *new* *root*

*return* y;

}

*//* *Get* *Balance* *factor* *of* *node* *N*

int **getBalance**(Node\* N)

{

*if* (N == NULL)

*return* 0;

*return* height(*N->left*) - height(*N->right*);

}

Node\* **insert**(Node\* node, NewNode a)

{

*/\** *1.* *Perform* *the* *normal* *BST* *insertion* *\*/*

*if* (node == NULL)

*return*(newNode(a));

*if* (a.key < node->key)

node->left = insert(*node->left*, a);

*else* *if* (a.key > node->key)

node->right = insert(*node->right*, a);

*else* *//* *Equal* *keys* *are* *not* *allowed* *in* *BST*

*return* node;

*/\** *2.* *Update* *height* *of* *this* *ancestor* *node* *\*/*

node->height = 1 + max(height(*node->left*),

height(*node->right*));

*/\** *3.* *Get* *the* *balance* *factor* *of* *this* *ancestor*

*node* *to* *check* *whether* *this* *node* *became*

*unbalanced* *\*/*

int balance = getBalance(*node*);

*//* *If* *this* *node* *becomes* *unbalanced,* *then*

*//* *there* *are* *4* *cases*

*//* *Left* *Left* *Case*

*if* (balance > 1 && a.key < node->left->key)

*return* rightRotate(*node*);

*//* *Right* *Right* *Case*

*if* (balance < -1 && a.key > node->right->key)

*return* leftRotate(*node*);

*//* *Left* *Right* *Case*

*if* (balance > 1 && a.key > node->left->key)

{

node->left = leftRotate(*node->left*);

*return* rightRotate(*node*);

}

*//* *Right* *Left* *Case*

*if* (balance < -1 && a.key < node->right->key)

{

node->right = rightRotate(*node->right*);

*return* leftRotate(*node*);

}

*/\** *return* *the* *(unchanged)* *node* *pointer* *\*/*

*return* node;

}

void **preOrder**(Node\* root)

{

*if* (root != NULL)

{

preOrder(*root->left*);

qDebug()<< root->key << " "<< QString::fromStdString(root->writter)<<" "<< QString::fromStdString(root->title)<<" "<< QString::fromStdString(root->publisher)<<" "<< QString::fromStdString(root->year\_publication)<<" "<<root->totalNumberOfCopies<<" "<<root->copiesAvailable<<" ";

preOrder(*root->right*);

}

}

int **extractFirstThreeDigits**(*const* QString &input) {

*//* *Take* *the* *first* *three* *characters* *of* *the* *input* *string*

QString firstThreeDigits = input.left(3);

*//* *Convert* *the* *extracted* *string* *to* *an* *integer*

bool conversionOk;

int integerValue = firstThreeDigits.toInt(*&conversionOk*);

*//* *Check* *if* *the* *conversion* *was* *successful*

*if* (conversionOk) {

*return* integerValue;

} *else* {

qDebug() << "Conversion to integer failed!";

*//* *Handle* *the* *error* *or* *return* *a* *default* *value*

*return* 0;

}

}

bool **validateFormat**(QString& input) {

*//* *Regular* *expression* *pattern* *for* *the* *format* *NNN.NNN*

QRegularExpression pattern("\\d{3}\\.\\d{3}");

*//* *Check* *if* *the* *input* *matches* *the* *pattern*

*return* pattern.match(input).hasMatch();

}

Node \***nodeWithMimumValue**(Node \*node) {

Node \*current = node;

*while* (current->left != NULL)

current = current->left;

*return* current;

}

Node \***deleteNode**(Node \*root, int key) {

*//* *Find* *the* *node* *and* *delete* *it*

*if* (root == NULL)

*return* root;

*if* (key < root->key)

root->left = deleteNode(*root->left*, key);

*else* *if* (key > root->key)

root->right = deleteNode(*root->right*, key);

*else* {

*if* ((root->left == NULL) ||

(root->right == NULL)) {

Node \*temp = root->left ? root->left : root->right;

*if* (temp == NULL) {

temp = root;

root = NULL;

} *else*

\*root = \*temp;

free(*temp*);

} *else* {

Node \*temp = nodeWithMimumValue(*root->right*);

root->key = temp->key;

root->right = deleteNode(*root->right*,

temp->key);

}

}

*if* (root == NULL)

*return* root;

*//* *Update* *the* *balance* *factor* *of* *each* *node* *and*

*//* *balance* *the* *tree*

root->height = 1 + max(height(*root->left*),

height(*root->right*));

int balanceFactor = getBalance(*root*);

*if* (balanceFactor > 1) {

*if* (getBalance(*root->left*) >= 0) {

*return* rightRotate(*root*);

} *else* {

root->left = leftRotate(*root->left*);

*return* rightRotate(*root*);

}

}

*if* (balanceFactor < -1) {

*if* (getBalance(*root->right*) <= 0) {

*return* leftRotate(*root*);

} *else* {

root->right = rightRotate(*root->right*);

*return* leftRotate(*root*);

}

}

*return* root;

}

void **inOrderTraverse**(Node\* node, std::vector<NewNode>& nodeList) {

*if* (node) {

*//* *Traverse* *left* *subtree*

inOrderTraverse(*node->left*, *nodeList*);

NewNode data;

data.id=node->id;

data.key=node->key;

data.writter=node->writter;

data.title=node->title;

data.publisher=node->publisher;

data.year\_publication=node->year\_publication;

data.totalNumberOfCopies=node->totalNumberOfCopies;

data.copiesAvailable=node->copiesAvailable;

*//* *Visit* *current* *node*

nodeList.push\_back(data);

*//* *Traverse* *right* *subtree*

inOrderTraverse(*node->right*, *nodeList*);

}

}

void **insertNodeIntoDatabase**(Node\* node, MainWindow& conn, std::ofstream& outFile) {

QSqlQuery qry(conn.mydbase);

qry.prepare("INSERT INTO books\_table (ID, writter, title, publisher, yearOfpublication, totalNumofCopies, NumofCopiesAvail) "

"VALUES (:id, :writter, :title, :publisher, :yearOfPublication, :totalNumofCopies, :NumofCopiesAvail)");

qry.bindValue(":id", QString::fromStdString(node->id));

qry.bindValue(":writter", QString::fromStdString(node->writter));

qry.bindValue(":title", QString::fromStdString(node->title));

qry.bindValue(":publisher", QString::fromStdString(node->publisher));

qry.bindValue(":yearOfPublication", QString::fromStdString(node->year\_publication));

qry.bindValue(":totalNumofCopies", node->totalNumberOfCopies);

qry.bindValue(":NumofCopiesAvail", node->copiesAvailable);

*if* (qry.exec()) {

qDebug() << "Node with ID" << node->key << "inserted into the database";

*//* *Write* *the* *node's* *data* *to* *the* *file*

outFile << node->id << ","

<< node->writter << ","

<< node->title << ","

<< node->publisher << ","

<< node->year\_publication << ","

<< node->totalNumberOfCopies << ","

<< node->copiesAvailable << "\n";

} *else* {

qDebug() << "Failed to insert node with ID" << node->key << "Error: " << qry.lastError().text();

}

}

void **Traverse**(Node\* root, MainWindow& conn, std::ofstream& outFile) {

*if* (root != *nullptr*) {

Traverse(*root->left*, *conn*, *outFile*);

*//* *Insert* *the* *node* *into* *the* *database* *and* *write* *to* *file*

insertNodeIntoDatabase(*root*, *conn*, *outFile*);

Traverse(*root->right*, *conn*, *outFile*);

}

}

void **downloadDataToDatabase**(Node\* root) {

*try* {

MainWindow conn;

*if* (!conn.connOpen()) {

qDebug() << "Failed to open database connection.";

*return*;

}

*//* *Open* *the* *file* *for* *writing*

std::ofstream outFile("C:\\Users\\1\\OneDrive\\Desktop\\Library\_Reader\_Service\\books\_data.txt");

*if* (!outFile.is\_open()) {

qDebug() << "Failed to open file for writing.";

*return*;

}

*//* *Traversing* *the* *tree* *and* *inserting* *each* *node* *into* *the* *database*

Traverse(*root*, *conn*, *outFile*);

outFile.close();

conn.connClose();

} *catch* (*const* std::exception& e) {

qDebug() << "Exception caught: " << e.*what*();

}

}

void **deleteAllBooksData**() {

*try* {

MainWindow conn;

*if* (!conn.connOpen()) {

qDebug() << "Failed to open database connection.";

*return*;

}

QSqlQuery \*qry = *new* QSqlQuery(conn.mydbase);

qry->prepare("DELETE FROM books\_table");

*if* (qry->exec()) {

qDebug()<<("All books data deleted successfully");

} *else* {

qDebug()<<("Error occured");

}

conn.connClose();

} *catch* (*const* std::exception& e) {

qDebug() << "Exception caught: " << e.*what*();

}

}

Node\* **search**(Node\* root, int key) {

*if* (root == NULL || root->key == key)

*return* root;

*if* (key < root->key)

*return* search(*root->left*, key);

*return* search(*root->right*, key);

}

Node\* **searchbytitle**(Node\* root, *const* std::string& writter) {

*//* *Base* *Cases:* *root* *is* *null* *or* *key* *is* *present* *at* *the* *root*

*if* (root == NULL || root->writter == writter)

*return* root;

*//* *Key* *is* *greater* *than* *root's* *key*

*if* (writter > root->writter)

*return* searchbytitle(*root->right*, writter);

*//* *Key* *is* *smaller* *than* *root's* *key*

*if* (writter < root->writter)

*return* searchbytitle(*root->left*, writter);

*return* root;

}

bool **searchByStringID**(Node\*& root, *const* Datatype& id,std::string& file) {

*//* *Traverse* *the* *tree*

Node\* current = root;

*while* (current != *nullptr*) {

*//* *Check* *if* *the* *current* *node's* *ID* *matches* *the* *target* *ID*

*if* (id.book\_ID == current->id) {

*//* *Ensure* *copiesAvailable* *is* *not* *reduced* *below* *zero*

*if* (!isValidFo(id.return\_date) && current->copiesAvailable > 0) {

current->copiesAvailable--;

*return* *true*;

} *else* *if* (current->copiesAvailable < current->totalNumberOfCopies && isValidFo(id.return\_date) && isValidFo(id.return\_date)) {

current->copiesAvailable++;

*//qDebug()<<QString::fromStdString(id.book\_ID)+"* *"* *+QString::fromStdString(id.student\_ID);*

deletelast(*file*,id);

*return* *true*;

}

*return* *false*;

*//* *Found* *the* *node* *with* *the* *given* *ID*

}

*//* *If* *the* *target* *ID* *is* *smaller,* *move* *to* *the* *left* *subtree*

*else* *if* (id.book\_ID < current->id) {

current = current->left;

}

*//* *If* *the* *target* *ID* *is* *larger,* *move* *to* *the* *right* *subtree*

*else* {

current = current->right;

}

}

*//* *Node* *with* *the* *given* *ID* *not* *found*

*//qDebug()<<"The* *elements* *are* *done";*

*return* *false*;

}

#ifndef LINKED\_LIST\_H

#define LINKED\_LIST\_H

#include "headers.h"

*struct* **Datatype** {

std::string student\_ID, book\_ID, lending\_date, return\_date;

};

*struct* **record** {

Datatype data;

record\* next = *nullptr*;

};

*class* **HashTable**;

*struct* **HashNode**;

*struct* **Node**;

void **free\_All\_record**(record\*\* begin);

bool **isCompanyNumPresent**(record\* tail, *const* std::string& name);

bool **addToBeginning**(record\*\* begin, Datatype elem, HashTable& myHashTable, Node\*& root,std::string& file);

void **DelElem**(record\*\* begin, record\* ptrCur);

void **Find\_Element**( record\* tail, *const* std::string& k);

record\* **FindElemCompany**(record\* tail, *const* std::string& elem);

void **deleteByCompanyNum**(record\*\* tail, *const* std::string& name);

record\* **editByCompanyNum**(record\* tail, *const* std::string& name, *const* Datatype& newData);

void **downloadToFile**(record\* tail, *const* std::string& fileName);

void **addToEnd**(record\*\* tail, *const* Datatype& data);

void **copyDataFromFileToCircularList**(*const* std::string& filename, record\*& head);

record\* **merge**(record\* a, record\* b);

record\* **mid\_point**(record\* head);

record\* **merge\_sort**(record\* head);

#endif *//* *LINKED\_LIST\_H*

#include "linked\_list.h"

bool **isCompanyNumPresent**(record\* tail, *const* std::string& name) {

*if* (!tail)

*return* *false*;

record\* current = tail;

*while* (current) {

*if* (current->data.student\_ID == name)

*return* *true*;

current = current->next;

}

*return* *false*;

}

bool **addToBeginning**(record\*\* begin, Datatype elem, HashTable& myHashTable, Node\*& root,std::string& file)

{

bool isUserPresent = myHashTable.isUsernamePresent(elem.student\_ID);

bool present = searchByStringID(*root*, elem,*file*);

*if* (isUserPresent && present ) {

record\* p = *new* record;

p->data = elem;

*//myHashTable.add(elem.student\_ID,elem.book\_ID,1);*

*//проверка,* *является* *ли* *список* *пустым*

*if* (\*begin == NULL)

{

p->next = NULL;

}

*else*

{

p->next = \*begin;

}

\*begin = p;

*return* *true*;

} *else* {

qDebug()<< "User with the specified username does not exist." << endl;

*return* *false*;

}

}

void **DelElem**(record\*\* begin, record\* ptrCur)

{

record\* p;

*if* (ptrCur == \*begin) {*//удаляем* *первый* *элемент*

\*begin = (\*begin)->next;

}

*else* {

p = \*begin;

*while* (p->next != ptrCur)

p = p->next;

*//* *удаление* *элемента*

p->next = ptrCur->next;

}

*delete* ptrCur;

}

void **Find\_Element**( record\* tail, *const* std::string& k) {

*if* (!tail) {

qDebug()<< "Список пуст!";

*return*;

}

record\* p = tail;

bool f = *false*;

*while* (p != NULL) {

*if* (p->data.student\_ID == k) {

f = *true*;

*//out* *<<* *(p->data);*

}

p = p->next;

}

*if* (!f) qDebug() << "Номер сотрудника не найден" << endl;

}

record\* **FindElemCompany**(record\* tail, *const* std::string& elem)

{

*if* (tail == *nullptr*) {

*return* *nullptr*;

}

record\* p = tail;

*while* (p != NULL) {

*if* ((p->data.student\_ID == elem))

*return* p;

p = p->next;

}

*return* *nullptr*;

}

void **deleteByCompanyNum**(record\*\* tail, *const* std::string& name) {

*if* (!tail) {

std::cout << "Ссылка пуста" << std::endl;

*return*;

}

record\* ptr;

ptr = FindElemCompany(*\*tail*, name);

*if* (ptr == NULL)qDebug()<< "Запись не найдена!" << endl;

*else* { DelElem(*tail*, *ptr*); qDebug() << "Запись удалена!" << endl; };

}

record\* **editByCompanyNum**(record\* tail, *const* std::string& name, *const* Datatype& newData) {

*if* (!tail) {

*return* *nullptr*;

}

record\* current = tail;

*while* (current != NULL) {

*if* (current->data.student\_ID == name) {

current->data = newData;

*return* tail;

}

current = current->next;

}

*return* tail;

}

void **downloadToFile**(record\* tail, *const* std::string& fileName) {

*if* (!tail) {

qDebug() << "Круговой связанный список пуст. Нечего скачивать." <<

endl;

*return*;

}

std::ofstream outFile(fileName);

*if* (!outFile) {

qDebug()<< "Failed to open file: " << QString::fromStdString(fileName) << endl;

*return*;

}

record\* current = tail;

*while* (current != NULL) {

outFile << std::setfill(' ') << current->data.student\_ID << std::setw(8) <<

" " << current->data.book\_ID << " " << current->data.lending\_date << std::setw(8)

<< " " << current->data.return\_date << std::endl;

current = current->next;

};

outFile.close();

qDebug() << "Циклический связанный список, загруженный в файл: " << QString::fromStdString(fileName)<<

endl;

}

void **addToEnd**(record\*\* tail, *const* Datatype& data) {

*if* (isCompanyNumPresent(*\*tail*, data.student\_ID)) {

qDebug() << "Этот номер компании уже существует в списке" << endl;

*return*;

}

record\* new\_record = *new* record();

new\_record->data = data;

record\* last = \*tail;

new\_record->next = NULL;

*if* (\*tail == NULL) {

\*tail = new\_record;

*return*;

}

*while* (last->next != NULL) {

last = last->next;

}

last->next = new\_record;

}

void **copyDataFromFileToCircularList**(*const* std::string& filename, record\*& head) {

std::ifstream inputFile(filename);

*if* (!inputFile.is\_open()) {

std::cerr << "Failed to open the file." << std::endl;

*return*;

}

Datatype data;

std::string line;

*while* (std::getline(inputFile, *line*)) {

std::istringstream lineStream(line);

*if* (!(lineStream >> data.student\_ID >> data.book\_ID >> data.lending\_date >>

data.return\_date))

{

std::cerr << "Неверные данные в файле. Пропуск строки." << std::endl;

*continue*;

}

addToEnd(*&head*, data);

}

*if* (inputFile.eof()) {

std::cout << "Дошел до конца файла." << std::endl;

}

*else* {

std::cerr << "An error occurred while reading the file." << std::endl;

}

inputFile.close();

}

void **free\_All\_record**(record\*\* begin) {

*if* (\*begin == 0) *return*;

record\* p = \*begin;

record\* t;

*while* (p)

{

t = p;

p = p->next;

*delete* t;

}\*begin = NULL;

}

record\* **merge**(record\* a, record\* b) {

*//* *base* *case*

*if* (a == NULL) *return* b;

*if* (b == NULL) *return* a;

*//* *рекурсивный* *случай,* *take* *a* *head* *pointer*

record\* c;

*if* (a->data.student\_ID < b->data.student\_ID) {

c = a;

c->next = merge(*a->next*, *b*);

}

*else* {

c = b;

c->next = merge(*a*, *b->next*);

}

*return* c;

}

record\* **mid\_point**(record\* head) {

*//* *base* *case*

*if* (head == NULL || head->next == NULL) *return* head;

*//* *recursive* *case*

record\* fast = head;

record\* slow = head;

*while* (fast != NULL && fast->next != NULL) {

fast = fast->next;

*if* (fast->next == NULL)

*break*;

fast = fast->next;

slow = slow->next;

}

*return* slow;

}

record\* **merge\_sort**(record\* head) {

*//* *base* *case*

*if* (head == NULL || head->next == NULL)

*return* head;

*//* *recursive* *case*

*//* *Step* *1:* *divide* *the* *linked* *list* *into*

*//* *two* *equal* *linked* *lists*

record\* mid = mid\_point(*head*);

record\* a = head;

record\* b = mid->next;

mid->next = NULL;

*//* *Step* *2:* *recursively* *sort* *the* *smaller*

*//* *linked* *lists*

a = merge\_sort(*a*);

b = merge\_sort(*b*);

*//* *Step* *3:* *merge* *the* *sorted* *linked* *lists*

record\* c = merge(*a*, *b*);

*return* c;

}

#ifndef BOYER\_MOORE\_H

#define BOYER\_MOORE\_H

#include "headers.h"

*struct* **Datatype**;

*struct* **link**;

*struct* **HashNode**;

void **badCharHeurist**(*const* std::string& pattern, std::vector<int>& badChar);

bool **searchBoyerMoore**(*const* std::string& text, *const* std::string& pattern);

bool **isDataPrese**(*const* std::string& username, *const* std::string& fileContents);

void **displayHashNo**(*const* HashNode& data, QLabel\* label);

void **uploadFromFi**(*const* std::string& fileName, *const* std::string& targetWord, QLabel\* label);

void **deletelast**(std::string& file , *const* Datatype& target);

void **retrieveBookInfo**(*const* std::string& fileContents, *const* std::string& username);

void **uploadFromF22**(*const* std::string& fileName, *const* std::string& targetWord, QLabel\* label);

void **uploadFromF222**(*const* std::string& fileName, *const* std::string& targetWord, QLabel\* label,QGraphicsView\* graph);

#endif *//* *BOYER\_MOORE\_H*

#include "boyer\_moore.h"

void **badCharHeurist**(*const* std::string& pattern, std::vector<int>& badChar) {

int m = pattern.size();

*for* (int i = 0; i < 256; i++)

badChar[i] = -1;

*for* (int i = 0; i < m; i++)

badChar[*static\_cast*<int>(pattern[i])] = i;

}

bool **searchBoyerMoore**(*const* std::string& text, *const* std::string& pattern) {

int m = pattern.size();

int n = text.size();

std::vector<int> badChar(256, 0);

badCharHeurist(pattern, *badChar*);

int s = 0; *//* *shift* *of* *the* *pattern* *with* *respect* *to* *the* *text*

*while* (s <= (n - m)) {

int j = m - 1;

*//* *Keep* *reducing* *index* *j* *of* *the* *pattern* *while* *characters* *of*

*//* *the* *pattern* *and* *text* *are* *matching* *at* *this* *shift* *s*

*while* (j >= 0 && tolower(pattern[j]) == tolower(text[s + j]))

j--;

*//* *If* *the* *pattern* *is* *present* *at* *the* *current* *shift,* *return* *true*

*if* (j < 0) {

*return* *true*;

}

*else* {

*//* *Shift* *the* *pattern* *to* *align* *the* *bad* *character* *in* *the* *text* *with* *the* *last* *occurrence* *in* *the* *pattern*

s += std::max(1, j - badChar[tolower(text[s + j])]);

}

}

*return* *false*;

}

bool **isDataPrese**(*const* std::string& username, *const* std::string& fileContents) {

*//* *Use* *Boyer-Moore* *to* *check* *if* *the* *username* *is* *present* *in* *the* *fileContents*

*return* searchBoyerMoore(fileContents, username);

}

void **displayHashNo**(*const* HashNode& data, QLabel\* label) {

std::string bookData = "Username: " + data.username +

"\nName: " + data.name +

"\nDate of Birth: " + data.birthDate +

"\nEmail: " + data.email +

"\nAddress: " + data.address +

"\nPassword: " + data.password

+ "\n";

*if* (data.bookstatus) {

link\* currentLink = data.bookstatus;

*while* (currentLink) {

bookData += "Book\_ID: " + currentLink->NNNMMM + "\n";

currentLink = currentLink->next;

}

}

bookData += "-----------------------";

label->setText(QString::fromStdString(bookData));

}

std::vector<std::string> **split**(*const* std::string& str, char delimiter) {

std::vector<std::string> tokens;

std::string token;

std::istringstream tokenStream(str);

*while* (std::getline(tokenStream, *token*, delimiter)) {

tokens.push\_back(token);

}

*return* tokens;

}

void **replaceCommasWithSpaces**(std::string& str) {

*for* (char& ch : str) {

*if* (ch == ',') {

ch = ' '; *//* *Replace* *comma* *with* *space*

}

}

}

void **replaceSpaces**(std::string& str) {

*for* (char& ch : str) {

*if* (ch == ' ') {

ch = ','; *//* *Replace* *comma* *with* *space*

}

}

}

void **deleteSubstring**(std::string& str, *const* std::string& substr) {

*//* *Find* *the* *position* *of* *the* *substring* *in* *the* *string*

size\_t pos = str.find(substr);

*//* *If* *the* *substring* *is* *found,* *erase* *it* *from* *the* *string*

*if* (pos != std::string::npos) {

str.erase(pos, substr.length());

}

}

void **deleteWord**(std::string& str, *const* std::string& word) {

*//* *Find* *the* *position* *of* *the* *word* *in* *the* *string*

size\_t pos = str.find(word);

*//* *If* *the* *word* *is* *found,* *erase* *it* *from* *the* *string*

*if* (pos != std::string::npos) {

*//* *Determine* *the* *position* *of* *the* *next* *space* *after* *the* *word*

size\_t nextSpace = str.find(' ', pos);

*//* *Erase* *the* *word* *from* *the* *string*

*if* (nextSpace != std::string::npos) {

str.erase(pos, nextSpace - pos + 1); *//* *+1* *to* *include* *the* *space*

} *else* {

*//* *If* *there's* *no* *space* *after* *the* *word,* *erase* *until* *the* *end* *of* *the* *string*

str.erase(pos);

}

}

}

void **deletelast**(std::string& file, *const* Datatype& target) {

std::ifstream inputFile(file);

*if* (!inputFile) {

qDebug() << "Error opening file.\n";

*return*;

}

std::ofstream tempFile("C:\\Users\\1\\OneDrive\\Desktop\\Library\_Reader\_Service\\temp.txt");

*if* (!tempFile) {

qDebug() << "Error creating temporary file.\n";

inputFile.close();

*return*;

}

std::string line;

*while* (std::getline(inputFile, *line*)) {

*if* (searchBoyerMoore(line, target.book\_ID) && searchBoyerMoore(line, target.student\_ID)) {

*//* *replaceCommasWithSpaces(line);*

*//* *deleteWord(line,* *target.book\_ID);*

deleteSubstring(*line*,target.book\_ID);

*//replaceSpaces(line);*

tempFile << line << std::endl;

qDebug() <<QString::fromStdString(line);

} *else* {

tempFile << line << std::endl;

qDebug() <<QString::fromStdString(line);

}

}

inputFile.close();

tempFile.close();

*if* (std::remove(file.c\_str()) != 0) {

std::perror("Error removing original file");

} *else* {

qDebug() << "Original file removed successfully.\n";

}

*if* (std::rename("C:\\Users\\1\\OneDrive\\Desktop\\Library\_Reader\_Service\\temp.txt", file.c\_str()) != 0) {

std::perror("Error renaming temporary file");

} *else* {

qDebug() << "Temporary file renamed successfully.\n";

}

}

void **uploadFromFi**(*const* std::string& fileName, *const* std::string& targetWord, QLabel\* label) {

std::ifstream inputFile(fileName, std::ios::in);

*if* (!inputFile) {

qDebug() << "Error opening file.\n";

*return*;

}

HashNode data;

std::string line;

*while* (getline(inputFile, *line*)) {

*if* (searchBoyerMoore(line, targetWord)) {

*//* *Create* *a* *new* *HashNode* *and* *copy* *basic* *data*

std::istringstream iss(line);

getline(iss, *data.username*, ',');

getline(iss, *data.name*, ',');

getline(iss, *data.birthDate*, ',');

getline(iss, *data.email*, ',');

getline(iss, *data.address*, ',');

getline(iss, *data.password*, ',');

*//* *Initialize* *the* *linked* *list* *to* *nullptr*

data.bookstatus = *nullptr*;

*//* *Read* *the* *linked* *list* *data*

link\* currentLink = *nullptr*;

std::string linkData;

*while* (getline(iss, *linkData*, ',')) {

link\* newLink = *new* link;

newLink->NNNMMM = linkData;

newLink->next = *nullptr*;

*if* (!data.bookstatus) {

data.bookstatus = newLink; *//* *Set* *the* *head* *of* *the* *list*

currentLink = newLink;

} *else* {

currentLink->next = newLink;

currentLink = newLink;

}

}

*//* *Display* *the* *HashNode* *data*

displayHashNo(data,*label*);

*//* *Reset* *data* *for* *the* *next* *iteration*

data = HashNode();

}

}

inputFile.close();

}

void **retrieveBookInfo**(*const* std::string &fileContents, *const* std::string &username)

{

*//* *Convert* *username* *to* *lowercase* *for* *case-insensitive* *search*

std::string lowercaseUsername = username;

std::transform(lowercaseUsername.begin(), lowercaseUsername.end(), lowercaseUsername.begin(), ::tolower);

std::istringstream iss(fileContents);

std::string line;

*while* (std::getline(iss, *line*)) {

*if* (searchBoyerMoore(line, lowercaseUsername)) {

qDebug()<< "Found book information: " << QString::fromStdString(line) <<endl;

*return*; *//* *Stop* *searching* *after* *finding* *the* *first* *match*

}*else*{

qDebug()<< "I didn't find any Found book information: ";

}

}

}

void **displayBookDataFromNode**(*const* Node& rootNode, *const* std::string& targetWord, QLabel\* label) {

*if* (rootNode.title.empty()) {

label->setText("Node is empty.");

*return*;

}

*if* (searchBoyerMoore(rootNode.title, targetWord)) {

*//* *Display* *book* *data* *on* *QLabel*

std::string bookData = "Title: " + rootNode.title +

"\nID: " + rootNode.id +

"\nWritter: " + rootNode.writter +

"\nPublisher: " + rootNode.publisher +

"\nYear of Publication: " + rootNode.year\_publication +

"\nTotal Number of Copies: " + std::to\_string(rootNode.totalNumberOfCopies) +

"\nCopies Available: " + std::to\_string(rootNode.copiesAvailable);

label->setText(QString::fromStdString(bookData));

} *else* {

*//* *If* *the* *title* *is* *not* *found,* *display* *a* *message*

label->setText("Book with the specified title not found.");

}

}

void **displayBookDat**(*const* Node& rootNode, *const* std::string& targetWord, QLabel\* label, QGraphicsView\* graph) {

*if* (rootNode.title.empty()) {

label->setText("Node is empty.");

*return*;

}

*if* (searchBoyerMoore(rootNode.title, targetWord)) {

*//* *Display* *book* *data* *on* *QLabel*

std::string bookData = "Title: " + rootNode.title +

"\nWritter: " + rootNode.writter +

"\nPublisher: " + rootNode.publisher +

"\nYear of Publication: " + rootNode.year\_publication;

QString path = ":/img\_book/" + QString::fromStdString(rootNode.title) + ".png";

QPixmap pixmap(path);

*if* (!pixmap.isNull()) {

QGraphicsScene\* scene = *new* QGraphicsScene();

scene->addPixmap(pixmap);

graph->setScene(*scene*);

label->setText(QString::fromStdString(bookData));

} *else* {

*//* *If* *the* *pixmap* *is* *null,* *display* *an* *error* *message*

label->setText("Failed to load book cover image.");

}

} *else* {

*//* *If* *the* *title* *is* *not* *found,* *display* *a* *message*

label->setText("Book with the specified title not found.");

}

}

void **uploadFromF22**(*const* std::string& fileName, *const* std::string& targetWord, QLabel\* label) {

std::ifstream inputFile(fileName);

*if* (!inputFile) {

qDebug() << "Error opening file.";

*return*;

}

Node data ;

std::string line;

*while* (getline(inputFile, *line*)) {

*if* (searchBoyerMoore(line, targetWord)) {

std::istringstream iss(line);

getline(iss, *data.id*, ',');

getline(iss, *data.writter*, ',');

getline(iss, *data.title*, ',');

getline(iss, *data.publisher*, ',');

getline(iss, *data.year\_publication*, ',');

std::string totalNumCopiesStr, copiesAvailStr;

getline(iss, *totalNumCopiesStr*, ',');

getline(iss, *copiesAvailStr*, ',');

data.totalNumberOfCopies = std::stoi(totalNumCopiesStr);

data.copiesAvailable = std::stoi(copiesAvailStr);

*//* *Display* *book* *data* *from* *the* *Node*

displayBookDataFromNode(data, targetWord, *label*);

data = Node(); *//* *Clean* *up* *memory* *if* *needed*

}

}

inputFile.close();

}

void **uploadFromF222**(*const* std::string& fileName, *const* std::string& targetWord, QLabel\* label,QGraphicsView\* graph) {

std::ifstream inputFile(fileName);

*if* (!inputFile) {

qDebug() << "Error opening file.";

*return*;

}

Node data ;

std::string line;

*while* (getline(inputFile, *line*)) {

*if* (searchBoyerMoore(line, targetWord)) {

std::istringstream iss(line);

getline(iss, *data.id*, ',');

getline(iss, *data.writter*, ',');

getline(iss, *data.title*, ',');

getline(iss, *data.publisher*, ',');

getline(iss, *data.year\_publication*, ',');

std::string totalNumCopiesStr, copiesAvailStr;

getline(iss, *totalNumCopiesStr*, ',');

getline(iss, *copiesAvailStr*, ',');

*try* {

data.totalNumberOfCopies = std::stoi(totalNumCopiesStr);

data.copiesAvailable = std::stoi(copiesAvailStr);

} *catch* (*const* std::invalid\_argument& e) {

qDebug() << "Error converting string to integer: " << e.*what*();

*//* *Handle* *the* *error* *(e.g.,* *provide* *default* *values* *or* *skip* *the* *line)*

}

*//* *Display* *book* *data* *from* *the* *Node*

displayBookDat(data, targetWord, *label*,*graph*);

data = Node(); *//* *Clean* *up* *memory* *if* *needed*

}

}

inputFile.close();

}

#ifndef EVENTS\_H

#define EVENTS\_H

#include "headers.h"

*class* **Events**

{

*public*:

**Events**();

};

*class* **Custom\_View**: *public* QGraphicsView

{

Q\_OBJECT

*public*:

*explicit* **Custom\_View**(QWidget \*parent = *nullptr*);

*protected*:

void ***dragEnterEvent***(QDragEnterEvent \* event);

void **dragLeaverEvent**(QDragLeaveEvent \* event);

void ***dragMoveEvent***(QDragMoveEvent \* event);

void ***dropEvent***(QDropEvent \*event);

*protected*:

QGraphicsScene \*scene;

QString configFilePath; *//* *Path* *to* *the* *configuration* *file*

void **loadPersistedData**();

void **savePersistedData**(*const* QString &filePath);

};

#endif *//* *EVENTS\_H*

#include "events.h"

#include <QSettings>

QString name;

Events::**Events**()

{

}

Custom\_View::**Custom\_View**(QWidget \*parent) : QGraphicsView(*parent*)

{

setAcceptDrops(*true*);

scene = *new* QGraphicsScene(*this*);

setScene(*scene*);

viewport()->installEventFilter(*this*);

*//* *Set* *the* *path* *to* *the* *configuration* *file*

configFilePath = QApplication::applicationDirPath() + "/config.ini";

*//* *Load* *persisted* *data* *(image* *file* *path)* *on* *initialization*

loadPersistedData();

}

void Custom\_View::***dragEnterEvent***(QDragEnterEvent \*event)

{

QDragEnterEvent \*DragEnterEvent = *static\_cast*<QDragEnterEvent\*>(event);

*if*(DragEnterEvent->mimeData()->hasUrls())

{

DragEnterEvent->acceptProposedAction();

}

}

void Custom\_View::**dragLeaverEvent**(QDragLeaveEvent \*event)

{

event->accept();

}

void Custom\_View::***dragMoveEvent***(QDragMoveEvent \*event)

{

event->accept();

event->acceptProposedAction();

}

void Custom\_View::***dropEvent***(QDropEvent \*event)

{

*if* (event->source() == *this*)

*return*;

QDropEvent \*dropEvent = *static\_cast*<QDropEvent \*>(event);

*if* (dropEvent->mimeData()->hasUrls())

{

QList<QUrl> urlList = dropEvent->mimeData()->urls();

foreach (QUrl url, urlList)

{

*if* (url.isLocalFile())

{

MainWindow another;

name =another.returnname();

QString filePath = url.toLocalFile();

QPixmap pixmap(filePath);

*if* (!pixmap.isNull())

{

*//* *Clear* *scene* *when* *a* *new* *image* *file* *is* *dropped*

*if* (scene->items().count() > 0)

scene->clear();

QGraphicsPixmapItem \*item = *new* QGraphicsPixmapItem(pixmap);

*//* *Set* *the* *scene* *rect* *based* *on* *the* *pixmap's* *rect*

scene->setSceneRect(pixmap.rect());

*//* *Ensure* *aspect* *ratio* *is* *preserved*

*//fitInView(scene->sceneRect(),* *Qt::KeepAspectRatio);*

*//* *Set* *the* *fixed* *width* *and* *height*

setFixedSize(120, 120);

scene->addItem(*item*);

*//* *Save* *the* *dropped* *image* *file* *path* *to* *the* *configuration* *file*

savePersistedData(filePath);

dropEvent->acceptProposedAction();

}

}

}

}

}

void Custom\_View::**loadPersistedData**()

{

QSettings settings(configFilePath, QSettings::*IniFormat*);

QString lastImagePath = settings.value("LastImagePath").toString();

*if* (!lastImagePath.isEmpty())

{

QPixmap pixmap(lastImagePath);

*if* (!pixmap.isNull())

{

*//* *Clear* *scene* *when* *loading* *a* *persisted* *image*

*if* (scene->items().count() > 0)

scene->clear();

scene->setSceneRect(pixmap.rect());

*//* *Ensure* *aspect* *ratio* *is* *preserved*

*//fitInView(scene->sceneRect(),* *Qt::KeepAspectRatio);*

*//* *Set* *the* *pixmap* *size* *directly*

QGraphicsPixmapItem \*item = *new* QGraphicsPixmapItem(pixmap.scaled(120, 120, Qt::*KeepAspectRatio*));

*//* *Set* *the* *fixed* *size* *for* *the* *QGraphicsView*

setFixedSize(120, 120);

scene->addItem(*item*);

}

}

}

void Custom\_View::**savePersistedData**(*const* QString &filePath)

{

QSettings settings(configFilePath, QSettings::*IniFormat*);

settings.setValue("LastImagePath", filePath);

}

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include "headers.h"

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*public*:

QSqlDatabase mydbase;

QString **returnname**();

void **connClose**(){

mydbase.close();

mydbase.removeDatabase(QSqlDatabase::defaultConnection);

};

bool **connOpen**(){

mydbase = QSqlDatabase::addDatabase("QSQLITE");

mydbase.setDatabaseName("C:/sqlite3/Usersdatabase/usersdata.db");

*if*(!mydbase.open()){

qDebug()<<("Failed to open database");

*return* *false*;

}*else*{

*return* *true*;

}

}

*public*:

**MainWindow**(QWidget \*parent = *nullptr*);

~***MainWindow***();

*private* slots:

void **on\_cancel\_login\_clicked**();

void **on\_enter\_login\_clicked**();

void **on\_Register\_Home\_clicked**();

void **on\_\_checkBox\_Home\_clicked**();

*private*:

Ui::MainWindow \*ui;

*//registerdialog* *\*ptrregisterdialog;*

void **updateLoginButtonState**(int state);

};

#endif *//* *MAINWINDOW\_H*

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(*parent*)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

ui->\_checkBox\_Home->setChecked(*false*);

ui->username->setPlaceholderText("Username");

ui->password->setPlaceholderText("Password");

*//ptrregisterdialog* *=* *new* *registerdialog(this);*

connect(ui->\_checkBox\_Home, &QCheckBox::stateChanged, *this*, &MainWindow::updateLoginButtonState);

updateLoginButtonState(ui->\_checkBox\_Home->checkState());

connect(ui->\_checkBox\_Home, &QCheckBox::clicked, *this*, &MainWindow::on\_\_checkBox\_Home\_clicked);

*if*(!connOpen()){

ui->label\_ho->setText("Failed to open database");

}

}

MainWindow::~***MainWindow***()

{

*delete* ui;

*//delete* *ptrregisterdialog;*

}

void MainWindow::**on\_cancel\_login\_clicked**()

{

*this*->close();

}

void MainWindow::**updateLoginButtonState**(int state)

{

*//* *Enable* *or* *disable* *the* *login* *button* *based* *on* *the* *checkbox* *state*

ui->enter\_login->setEnabled(state == Qt::*Checked*);

}

void MainWindow::**on\_enter\_login\_clicked**()

{

QString Username,Password;

Username = ui->username->text();

Password = ui->password->text();

*if*(Username==QString::fromStdString("A1234-20") && Password == QString::fromStdString("owner")){

connClose();

*this*->hide();

ownerDialog owner;

owner.setModal(*true*);

owner.*exec*();

}*else*{

*if*(!connOpen()){

qDebug()<<"Failed to open the database";

*return*;

}

connOpen();

QSqlQuery qry;

qry.prepare("select \* from login\_data where ID='"+Username+"' and Password='"+Password+"'");

*if*(qry.exec( )){

int c=0;

*while*(qry.next()){

c++;

}

*if*(c==1){

QFile file("C:\\Users\\1\\OneDrive\\Desktop\\Library\_Reader\_Service\\credentials.txt");

*if* (file.*open*(QIODevice::*WriteOnly* | QIODevice::*Text*))

{

QTextStream out(*&file*);

out << Username << "\n";

out << Password << "\n";

file.*close*();

}

connClose();

*this*->hide();

portaldialog studentinf;

studentinf.setModal(*true*);

studentinf.*exec*();

}

*if*(c>1)

ui->label\_ho->setText("Дублируйте имя пользователя и пароль!!");

*if*(c<1)

ui->label\_ho->setText("Неверные детали");

}

}

}

QString MainWindow::**returnname**(){

*return* ui->username->text();

}

void MainWindow::**on\_Register\_Home\_clicked**()

{

*//ptrregisterdialog->Show();*

connClose();

*this*->hide();

registerdialog owner;

owner.setModal(*true*);

owner.*exec*();

}

void MainWindow::**on\_\_checkBox\_Home\_clicked**()

{

*if* (ui->username->text().isEmpty() || ui->password->text().isEmpty()) {

*//* *If* *either* *field* *is* *empty,* *prevent* *the* *checkbox* *from* *being* *checked*

ui->\_checkBox\_Home->setChecked(*false*);

}

}

#ifndef OWNERDIALOG\_H

#define OWNERDIALOG\_H

#include "headers.h"

*struct* **NewNode**;

*struct* **Node**;

*class* **HashTable**;

*struct* **HashNode**;

*struct* **Datatype**;

*namespace* **Ui** {

*class* **ownerDialog**;

}

*class* **ownerDialog** : *public* QDialog

{

Q\_OBJECT

*public*:

std::string checker;*//global* *variable* *to* *check* *the* *status* *of* *the* *combobox* *on* *functionalities*

void **disableButton**(QPushButton\* button);

void **clear\_inputdata**();

void **loadFromDatabase**();

NewNode **createNewNode**();

void **uploadDatatypeFromFile**(*const* std::string& fileName, std::vector<Datatype>& data, QListWidget\* listWidget);

Node\* Root = *nullptr*;

void **displayNextPicture**();

void **startPictureTimer**();

*explicit* **ownerDialog**(QWidget \*parent = *nullptr*);

~***ownerDialog***();

*private* slots:

void **on\_exit\_clicked**();

QString **on\_functions\_comboBox\_currentTextChanged**(*const* QString &arg1);

void **on\_users\_data\_button\_clicked**();

void **on\_clear\_table\_view\_clicked**();

void **on\_add\_user\_clicked**();

void **on\_delete\_user\_clicked**();

void **on\_delete\_user\_pressed**();

void **on\_update\_user\_clicked**();

void **on\_sort\_user\_clicked**();

void **on\_save\_user\_clicked**();

void **on\_minimize\_clicked**();

void **on\_books\_data\_button\_clicked**();

void **on\_save\_table\_view\_clicked**();

void **on\_search\_comboBox\_currentTextChanged**(*const* QString &arg1);

void **on\_searchButtononedit\_clicked**();

void **on\_send\_message\_clicked**();

void **on\_pic\_button\_clicked**();

*private*:

Ui::ownerDialog \*ui;

void **clearTableView**(QTableView\* tableView);

void **loadMessage**();

};

#endif *//* *OWNERDIALOG\_H*

#include "ownerdialog.h"

#include "ui\_ownerdialog.h"

#include <QComboBox>

#include <QIcon>

#include <QToolTip>

#include <QDebug>

#include "abv\_tree.h"

HashTable TABLE;

record \*head = *nullptr*;

std::vector<Datatype> uploadedData;

std::string fil = "C:\\Users\\1\\OneDrive\\Desktop\\Library\_Reader\_Service\\history.txt";

std::string fileName = "C:\\Users\\1\\OneDrive\\Desktop\\Library\_Reader\_Service\\users\_datails.txt";

std::string filename2 = "C:\\Users\\1\\OneDrive\\Desktop\\Library\_Reader\_Service\\books\_data.txt";

std::string status = " ";

ownerDialog::**ownerDialog**(QWidget \*parent) :

QDialog(*parent*),

ui(*new* Ui::ownerDialog)

{

ui->setupUi(*this*);

connect(ui->functions\_comboBox, &QComboBox::currentTextChanged, *this*, &ownerDialog::on\_functions\_comboBox\_currentTextChanged);

on\_functions\_comboBox\_currentTextChanged(ui->functions\_comboBox->currentText());

ui->functions\_comboBox->addItem(QIcon(":/img/Scre.png"), "Users");

ui->functions\_comboBox->addItem(QIcon(":/img/book.png"), "Books");

ui->functions\_comboBox->addItem(QIcon(":/img/receiving.png"), "Issue");

*//ui->functions\_comboBox->addItem(QIcon(":/img/receiving.png"),* *"Receive");*

ui->comboBox\_user->addItem(QIcon(":/img/file.png"), "file");

ui->comboBox\_user->addItem(QIcon(":/img/database.png"), "database");

ui->search\_comboBox->addItem(QIcon(":/img/Scre.png"), "Users");

ui->search\_comboBox->addItem(QIcon(":/img/book.png"), "Books");

loadFromDatabase();

TABLE.retrieveFromDatabase(*TABLE*);

uploadDatatypeFromFile(fil, *uploadedData*,*ui->listWidget*);

loadMessage();

startPictureTimer();

}

ownerDialog::~***ownerDialog***()

{

*delete* ui;

}

void ownerDialog::**on\_exit\_clicked**()

{

TABLE.freehashtablelinkedlist();

*this*->close();

}

QString ownerDialog::**on\_functions\_comboBox\_currentTextChanged**(*const* QString &arg1)

{

*if* (arg1 == "Users") {

checker="Users";

clear\_inputdata();

ui->input\_1->setPlaceholderText("Name");

ui->input\_2->setPlaceholderText("Birthday");

ui->input\_3->setPlaceholderText("Email");

ui->input\_4->setPlaceholderText("Address");

ui->input\_5->setPlaceholderText("Password");

ui->input\_6->setPlaceholderText("Student ID");

ui->input\_7->setPlaceholderText("");

ui->input\_7->setStyleSheet("background-color: transparent;"

"border-top: 0px solid #4f4f4f;"

"border-bottom: 0px solid #4f4f4f;"

"border-left: 0px solid #4f4f4f;"

"border-right: 0px solid #4f4f4f;");

ui->input\_5->setStyleSheet("background-color:white");

ui->input\_6->setStyleSheet("background-color:white");

ui->input\_5->setDisabled(*false*);

ui->input\_6->setDisabled(*false*);

ui->input\_7->setDisabled(*true*);

} *else* *if* (arg1 == "Books") {

checker="Books";

clear\_inputdata();

ui->input\_1->setPlaceholderText("Book ID");

ui->input\_2->setPlaceholderText("Writter");

ui->input\_3->setPlaceholderText("Title");

ui->input\_4->setPlaceholderText("Publisher");

ui->input\_5->setPlaceholderText("Year of publication");

ui->input\_6->setPlaceholderText("Total number of copies");

ui->input\_7->setPlaceholderText("Number of copies available");

ui->input\_5->setStyleSheet("background-color:white");

ui->input\_6->setStyleSheet("background-color:white");

ui->input\_7->setStyleSheet("background-color:white");

ui->input\_5->setDisabled(*false*);

ui->input\_6->setDisabled(*false*);

ui->input\_7->setDisabled(*false*);

}*else* *if*(arg1=="Issue"){

checker = "Issuing";

clear\_inputdata();

ui->input\_1->setPlaceholderText("Student ID");

ui->input\_2->setPlaceholderText("Book ID");

ui->input\_3->setPlaceholderText("Date of Borrowing");

ui->input\_4->setPlaceholderText("Date of Return");

ui->input\_5->setPlaceholderText("");

ui->input\_5->setStyleSheet("background-color: transparent;"

"border-top: 0px solid #4f4f4f;"

"border-bottom: 0px solid #4f4f4f;"

"border-left: 0px solid #4f4f4f;"

"border-right: 0px solid #4f4f4f;");

ui->input\_5->setDisabled(*true*);

ui->input\_6->setPlaceholderText("");

ui->input\_6->setStyleSheet("background-color: transparent;"

"border-top: 0px solid #4f4f4f;"

"border-bottom: 0px solid #4f4f4f;"

"border-left: 0px solid #4f4f4f;"

"border-right: 0px solid #4f4f4f;");

ui->input\_6->setDisabled(*true*);

ui->input\_7->setPlaceholderText("");

ui->input\_7->setStyleSheet("background-color: transparent;"

"border-top: 0px solid #4f4f4f;"

"border-bottom: 0px solid #4f4f4f;"

"border-left: 0px solid #4f4f4f;"

"border-right: 0px solid #4f4f4f;");

ui->input\_5->setDisabled(*true*);

ui->input\_6->setDisabled(*true*);

ui->input\_7->setDisabled(*true*);

}

*return* arg1;

}

void ownerDialog::**on\_users\_data\_button\_clicked**()

{

TABLE.retrieveFromDatabase(*TABLE*);

TABLE.display(*ui->tableView*);

}

void ownerDialog::**loadFromDatabase**() {

*try* {

MainWindow conn;

*if* (!conn.connOpen()) {

qDebug() << "Failed to open database connection.";

*return*;

}

QSqlQuery \*qry = *new* QSqlQuery(conn.mydbase);

qry->prepare("select \* from books\_table");

qry->exec();

*while* (qry->next()) {

QString Name = qry->value("ID").toString();

int ID;

ID= extractFirstThreeDigits(Name);

NewNode newNode ;

newNode.key = ID;

newNode.id = Name.toStdString();

newNode.writter = qry->value("writter").toString().toStdString();

newNode.title = qry->value("title").toString().toStdString();

newNode.publisher = qry->value("publisher").toString().toStdString();

newNode.year\_publication = qry->value("yearOfPublication").toString().toStdString();

newNode.totalNumberOfCopies = qry->value("totalNumofCopies").toInt();

newNode.copiesAvailable = qry->value("NumofCopiesAvail").toInt();

Root = insert(*Root*,newNode);

*//AddTreeElem(Root,newNode);*

qDebug() << "This was succesfull " ;

}

conn.connClose();

} *catch* (*const* std::exception& e) {

qDebug() << "Exception caught: " << e.*what*();

}

}

void ownerDialog::**clearTableView**(QTableView\* tableView) {

QStandardItemModel\* model = *dynamic\_cast*<QStandardItemModel\*>(tableView->model());

*if* (model) {

model->*removeRows*(0, model->*rowCount*());

model->setHorizontalHeaderLabels({"ID", "Name", "Birthday", "Email", "Address", "Password"});

model->clear();

}

}

void ownerDialog::**loadMessage**()

{

QFile file("C:\\Users\\1\\OneDrive\\Desktop\\Library\_Reader\_Service\\Messages\_text.txt");

*if* (file.*open*(QIODevice::*ReadOnly* | QIODevice::*Text*))

{

QTextStream in(*&file*);

*while* (!in.atEnd())

{

QString line = in.readLine();

*//* *Split* *the* *line* *into* *words*

QStringList words = line.split(' ');

*//* *Remove* *empty* *words*

words.removeAll("");

*if* (words.size() > 1)

{

words.removeFirst(); *//* *Remove* *the* *first* *word* *(username)*

QString remainingWords = words.join(' ');

ui->listWidget\_2->addItem(remainingWords);

}

}

file.*close*();

}

}

void ownerDialog::**clear\_inputdata**(){

ui->input\_1->clear();

ui->input\_2->clear();

ui->input\_3->clear();

ui->input\_4->clear();

ui->input\_5->clear();

ui->input\_6->clear();

ui->input\_7->clear();

}

void ownerDialog::**on\_clear\_table\_view\_clicked**()

{

TABLE.clear();

clearTableView(*ui->tableView*);

}

NewNode ownerDialog::**createNewNode**(){

QString Name = ui->input\_1->text();

int ID;

ID= extractFirstThreeDigits(Name);

NewNode newNode;

newNode.key = ID;

newNode.id = Name.toStdString();

newNode.writter = ui->input\_2->text().toStdString();

newNode.title = ui->input\_3->text().toStdString();

newNode.publisher = ui->input\_4->text().toStdString();

newNode.year\_publication = ui->input\_5->text().toStdString();

newNode.totalNumberOfCopies = ui->input\_6->text().toInt();

newNode.copiesAvailable = ui->input\_7->text().toInt();

*return* newNode;

}

void **DatatypeToFile**(*const* std::string& fileName, *const* Datatype& elem) {

std::ofstream outputFile(fileName, std::ios::out | std::ios::app);

outputFile << elem.student\_ID << "\t"

<< elem.book\_ID << "\t"

<< elem.lending\_date << "\t"

<< elem.return\_date <<std::endl;

outputFile.close();

}

void ownerDialog::**uploadDatatypeFromFile**(*const* std::string& fileName, std::vector<Datatype>& data, QListWidget\* listWidget) {

std::ifstream inputFile(fileName, std::ios::in);

*if* (!inputFile) {

*//* *Handle* *file* *opening* *error*

*return*;

}

Datatype elem;

listWidget->addItem(" ");

listWidget->addItem(" Student\_ID Book\_ID Lending\_date Return\_date ");

listWidget->addItem("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

*while* (inputFile >> elem.student\_ID >> elem.book\_ID >> elem.lending\_date >> elem.return\_date ) {

data.push\_back(elem);

*//* *Add* *the* *item* *to* *the* *QListWidget*

QString listItem = QString::fromStdString(" "+elem.student\_ID + " " + elem.book\_ID + " " + elem.lending\_date + " " + elem.return\_date)+" ";

listWidget->addItem(listItem);

}

inputFile.close();

}

void ownerDialog::**displayNextPicture**()

{

*static* int currentIndex = 0;

*const* QStringList picturePaths = {":/img\_book/Obedience to God.png", ":/img\_book/Welcome Home.png", ":/img\_book/C++ complete course.png",":/img\_book/The heart of God.png",":/img\_book/Java Script done.png", ":/img\_book/Money talks.png", ":/img\_book/Come back home.png"};

*if* (currentIndex < picturePaths.size()) {

QPixmap pixmap(picturePaths[currentIndex]);

*if* (!pixmap.isNull()) {

QGraphicsScene\* scene = *new* QGraphicsScene(*this*);

scene->addPixmap(pixmap);

ui->graphicsView\_3->setScene(*scene*);

currentIndex = (currentIndex + 1) % picturePaths.size();

} *else* {

qDebug() << "Failed to load image at index" << currentIndex;

}

}

}

void ownerDialog::**startPictureTimer**()

{

*//* *Create* *a* *QTimer* *for* *updating* *the* *picture* *every* *4* *seconds*

QTimer\* timer = *new* QTimer(*this*);

connect(timer, &QTimer::timeout, *this*, &ownerDialog::displayNextPicture);

*//* *Start* *the* *timer* *with* *an* *interval* *of* *4* *seconds* *(4000* *milliseconds)*

timer->start(4000);

}

void ownerDialog::**on\_add\_user\_clicked**()

{

MainWindow conn;

QString Name, Birthday, Email, Address, password, key;

Name = ui->input\_1->text();

Birthday = ui->input\_2->text();

Email = ui->input\_3->text();

Address = ui->input\_4->text();

password = ui->input\_5->text();

key = ui->input\_6->text();

*if* (!conn.connOpen()) {

qDebug() << "Failed to open the database";

*return*;

}

conn.connOpen();

QSqlQuery qry;

*if* (checker == "Users") {

*//* *Check* *if* *the* *username* *already* *exists* *before* *attempting* *to* *insert*

qry.prepare("SELECT \* FROM login\_data WHERE ID='" + key + "'");

*if* (qry.exec() && !qry.next()) {

qry.prepare("INSERT INTO login\_data (ID, Name, Birthday, Email, Address, Password) "

"VALUES ('" + key + "','" + Name + "','" + Birthday + "','" + Email + "','" + Address + "','" + password + "')");

*if* (qry.exec()) {

ui->comments\_on\_inputs->setText("User saved to database");

} *else* {

QMessageBox::critical(*this*, tr("error::"), qry.lastError().text());

}

} *else* {

ui->comments\_on\_inputs->setText("User with ID " + key + " already exists ");

}

clear\_inputdata();

conn.connClose();

} *else* *if* (checker == "Books") {

*if*(validateFormat(*Name*)){

NewNode newNode =createNewNode();

Root = insert(*Root*,newNode);

clear\_inputdata();

ui->comments\_on\_inputs->setText("The book "+Name+" is saved");

}*else*{

ui->comments\_on\_inputs->setText("Incorrect formate № book ID ");

}

qDebug() << "On books now";

} *else* {

Datatype elem;

elem.student\_ID = ui->input\_1->text().toStdString();

elem.book\_ID = ui->input\_2->text().toStdString();

elem.lending\_date = ui->input\_3->text().toStdString();

elem.return\_date = ui->input\_4->text().toStdString();

*if*(addToBeginning(*&head*, elem, *TABLE*,*Root*,*fileName*)){

*if*(TABLE.editByUsername(elem.student\_ID, elem)){

DatatypeToFile(fil, elem);

*//TABLE.downloadToFile(fileName);*

*//TABLE.uploadFromFile(fileName);*

ui->listWidget->addItem(QString::fromStdString(" "+elem.student\_ID + " " + elem.book\_ID + " " + elem.lending\_date + " " + elem.return\_date)+" ");

}

ui->comments\_on\_inputs->setText("Successfully added");

clear\_inputdata();

}*else*{

ui->comments\_on\_inputs->setText("Username unavailable");

}

}

}

void ownerDialog::**on\_delete\_user\_clicked**()

{

MainWindow conn;

QString username = ui->input\_6->text();

QString bookID = ui->input\_1->text();

*if* (!conn.connOpen()) {

qDebug() << "Failed to open the database";

*return*;

}

conn.connOpen();

QSqlQuery qry;

*if* (checker == "Users") {

*//* *Check* *if* *the* *username* *exists* *before* *attempting* *to* *delete*

qry.prepare("SELECT \* FROM login\_data WHERE ID='" + username + "'");

*if* (qry.exec() && qry.next()) {

qry.prepare("DELETE FROM login\_data WHERE ID='" + username + "'");

*if* (qry.exec()) {

ui->comments\_on\_inputs->setText(username + " has been deleted");

} *else* {

ui->comments\_on\_inputs->setText("Failed to delete " + username);

}

} *else* {

ui->comments\_on\_inputs->setText(username + " not found in the database");

}

clear\_inputdata();

conn.connClose();

} *else* *if* (checker == "Books") {

int ID;

ID= extractFirstThreeDigits(bookID);

Root = deleteNode(*Root*,ID);

clear\_inputdata();

ui->comments\_on\_inputs->setText(bookID +" is deleted");

qDebug() << "On books now";

} *else* {

ui->comments\_on\_inputs->setText("On Issuing now");

}

}

void ownerDialog::**on\_delete\_user\_pressed**()

{

ui->comments\_on\_inputs->setText("Delete the user by key");

}

void ownerDialog::**on\_update\_user\_clicked**()

{

MainWindow conn;

QString Name, Birthday, Email, Address, password, key;

Name = ui->input\_1->text();

Birthday = ui->input\_2->text();

Email = ui->input\_3->text();

Address = ui->input\_4->text();

password = ui->input\_5->text();

key = ui->input\_6->text();

*if* (!conn.connOpen()) {

qDebug() << "Failed to open the database";

*return*;

}

conn.connOpen();

QSqlQuery qry;

*if* (checker == "Users") {

*//* *Check* *if* *the* *user* *with* *the* *given* *ID* *exists*

qry.prepare("SELECT \* FROM login\_data WHERE ID='" + key + "'");

*if* (qry.exec() && qry.next()) {

*//* *User* *with* *the* *given* *ID* *found,* *proceed* *to* *update* *information*

qry.prepare("UPDATE login\_data SET Name='" + Name + "', Birthday='" + Birthday + "', "

"Email='" + Email + "', Address='" + Address + "', Password='" + password + "' "

"WHERE ID='" + key + "'");

*if* (qry.exec()) {

ui->comments\_on\_inputs->setText("User information updated");

} *else* {

QMessageBox::critical(*this*, tr("error::"), qry.lastError().text());

}

} *else* {

ui->comments\_on\_inputs->setText("User with ID " + key + " not found in the database");

}

clear\_inputdata();

conn.connClose();

} *else* *if* (checker == "Books") {

int ID;

ID= extractFirstThreeDigits(Name);

Root = deleteNode(*Root*, ID);

NewNode newNode =createNewNode();

*//* *Then,* *insert* *a* *new* *node* *with* *the* *new* *key*

Root = insert(*Root*, newNode);

clear\_inputdata();

ui->comments\_on\_inputs->setText("updated successful");

qDebug() << "On books now";

} *else* {

ui->comments\_on\_inputs->setText("On Issuing now");

}

}

void ownerDialog::**disableButton**(QPushButton\* button) {

*if* (button) {

button->setEnabled(*false*);

}

}

void ownerDialog::**on\_sort\_user\_clicked**()

{

Datatype id;

id.book\_ID = "008.128";

id.student\_ID ="B4502-21";

id.lending\_date ="12.11.2001";

id.return\_date ="31.03.2001";

std::string file ="C:\\Users\\1\\OneDrive\\Desktop\\Library\_Reader\_Service\\users\_datails.txt";

qDebug()<<QString::fromStdString(id.book\_ID)+" " +QString::fromStdString(id.student\_ID);

deletelast(*file*,id);

}

void ownerDialog::**on\_save\_user\_clicked**()

{

TABLE.downloadToFile(*fileName*);

ui->comments\_on\_inputs->setText("Downloaded to file");

}

void ownerDialog::**on\_minimize\_clicked**()

{

showMinimized();

}

void ownerDialog::**on\_books\_data\_button\_clicked**()

{

*try* {

std::vector<NewNode> nodeList; *//* *Store* *AVL* *tree* *nodes* *in* *a* *vector*

*//* *Traverse* *the* *AVL* *tree* *and* *populate* *the* *vector*

inOrderTraverse(*Root*, *nodeList*); *//* *You* *need* *to* *implement* *inOrderTraverse*

*//* *Create* *a* *model* *to* *hold* *the* *data*

QStandardItemModel \*model = *new* QStandardItemModel(nodeList.size(), 7, *this*);

*//* *Set* *headers* *for* *the* *columns*

model->setHorizontalHeaderItem(0, *new* QStandardItem*(*QString*(*"ID"*))*);

model->setHorizontalHeaderItem(1, *new* QStandardItem*(*QString*(*"Writter"*))*);

model->setHorizontalHeaderItem(2, *new* QStandardItem*(*QString*(*"Title"*))*);

model->setHorizontalHeaderItem(3, *new* QStandardItem*(*QString*(*"Publisher"*))*);

model->setHorizontalHeaderItem(4, *new* QStandardItem*(*QString*(*"Year of Publication"*))*);

model->setHorizontalHeaderItem(5, *new* QStandardItem*(*QString*(*"Total Copies"*))*);

model->setHorizontalHeaderItem(6, *new* QStandardItem*(*QString*(*"Available Copies"*))*);

*//* *Populate* *the* *model* *with* *data* *from* *the* *vector*

*for* (unsigned long long row = 0; row < nodeList.size(); ++row) {

*const* NewNode& node = nodeList[row];

model->setItem(row, 0, *new* QStandardItem*(*QString*::fromStdString(node.id))*);

model->setItem(row, 1, *new* QStandardItem*(*QString*::fromStdString(node.writter))*);

model->setItem(row, 2, *new* QStandardItem*(*QString*::fromStdString(node.title))*);

model->setItem(row, 3, *new* QStandardItem*(*QString*::fromStdString(node.publisher))*);

model->setItem(row, 4, *new* QStandardItem*(*QString*::fromStdString(node.year\_publication))*);

model->setItem(row, 5, *new* QStandardItem*(*QString*::number(node.totalNumberOfCopies))*);

model->setItem(row, 6, *new* QStandardItem*(*QString*::number(node.copiesAvailable))*);

}

*//* *Set* *the* *model* *for* *the* *tableView*

ui->tableView->*setModel*(*model*);

loadFromDatabase();

} *catch* (*const* std::exception& e) {

qDebug() << "Exception caught: " << e.*what*();

}

}

void ownerDialog::**on\_save\_table\_view\_clicked**()

{

deleteAllBooksData();

downloadDataToDatabase(*Root*);

ui->comments\_on\_inputs->setText("Saved in the database");

}

void ownerDialog::**on\_search\_comboBox\_currentTextChanged**(*const* QString &arg1)

{

*if*(arg1 == "Users"){

status = "Users";

}*else* *if*(arg1 == "Books"){

status = "Books";

}

}

void ownerDialog::**on\_searchButtononedit\_clicked**()

{

std::string word = ui->search\_lineEdit->text().toStdString();

*if*(status =="Users"){

uploadFromFi(fileName,word,*ui->search\_view*);

}*else*{

uploadFromF22(filename2, word, *ui->search\_view*);

*//TABLE.displayUserData(ui->search\_view,* *Root,* *status,word);*

}

}

void ownerDialog::**on\_send\_message\_clicked**()

{

QString data = ui->Message\_box->text();

QString id = ui->idmessage->text();

*//* *Append* *data* *to* *the* *file*

QFile file("C:\\Users\\1\\OneDrive\\Desktop\\Library\_Reader\_Service\\Messages\_text.txt");

*if* (file.*open*(QIODevice::*Append* | QIODevice::*Text*))

{

QTextStream out(*&file*);

out <<id+" libraly: "+ data << "\n";

file.*close*();

}

*//* *Update* *QListWidget*

ui->listWidget\_2->addItem(" libraly: "+ data );

ui->Message\_box->clear();

}

void ownerDialog::**on\_pic\_button\_clicked**()

{

QString data = ui->sendpic\_nameline->text();

*//* *Append* *data* *to* *the* *file*

QFile file("C:\\Users\\1\\OneDrive\\Desktop\\Library\_Reader\_Service\\pic\_user.txt");

*if* (file.*open*(QIODevice::*Append* | QIODevice::*Text*))

{

QTextStream out(*&file*);

out << data << "\n";

file.*close*();

ui->sendpic\_nameline->clear();

qDebug() <<"updated";

}

}

#ifndef PORTALDIALOG\_H

#define PORTALDIALOG\_H

#include "headers.h"

*namespace* **Ui** {

*class* **portaldialog**;

}

*class* **portaldialog** : *public* QDialog

{

Q\_OBJECT

*public*:

*explicit* **portaldialog**(QWidget \*parent = *nullptr*);

~***portaldialog***();

void **func**(QString ,QString);

void **displayNextP**();

void **startPictureT**();

void **databasetofile**();

*private* slots:

void **on\_minimize\_clicked**();

void **on\_message\_button\_clicked**();

void **on\_search\_button\_clicked**();

*private*:

Ui::portaldialog \*ui;

bool **readCredentialsFromFile**(QString &username, QString &password);

void **loadMess**();

};

#endif *//* *PORTALDIALOG\_H*

#include "portaldialog.h"

#include "ui\_portaldialog.h"

std::string filenam = "C:\\Users\\1\\OneDrive\\Desktop\\Library\_Reader\_Service\\books\_data.txt";

portaldialog::**portaldialog**(QWidget \*parent) :

QDialog(*parent*),

ui(*new* Ui::portaldialog)

{

ui->setupUi(*this*);

QString username, password;

*if* (readCredentialsFromFile(*username*, *password*))

{

func(username,password);

}

*else*{

qDebug() << "Failed to read credentials from file.";

}

loadMess();

startPictureT();

databasetofile();

std::string file = "C:\\Users\\1\\OneDrive\\Desktop\\Library\_Reader\_Service\\books\_data.txt";

std::string name = "java";

*//retrieveBookInfo(file,* *name);*

}

portaldialog::~***portaldialog***()

{

*delete* ui;

}

void portaldialog::**func**(QString username, QString password)

{

password = "word";

*//* *Now* *query* *the* *database* *to* *retrieve* *other* *information*

MainWindow conn;

*if* (!conn.connOpen()) {

qDebug() << "Failed to open the database";

*return*;

}

QSqlQuery qry;

qry.prepare("SELECT \* FROM login\_data WHERE ID=:username");

qry.bindValue(":username", username);

*if* (qry.exec() && qry.next()) {

QString Name = qry.value("Name").toString();

QString Birthday = qry.value("Birthday").toString();

QString Email = qry.value("Email").toString();

QString Address = qry.value("Address").toString();

QString password\_ = qry.value("Password").toString();

ui->name\_edit->setPlaceholderText(Name);

ui->Birthday\_edit->setPlaceholderText(Birthday);

ui->Email\_edit->setPlaceholderText(Email);

ui->Address\_edit->setPlaceholderText(Address);

ui->Password\_edit->setPlaceholderText(password\_);

*//* *You* *can* *update* *the* *fields* *as* *needed* *(e.g.,* *display* *them* *in* *labels,* *etc.)*

qDebug() << "User information retrieved successfully";

} *else* {

QMessageBox::critical(*this*, tr("Error"), qry.lastError().text());

}

conn.connClose(); *//* *Close* *the* *connection*

}

void portaldialog::**displayNextP**()

{

QString username, password;

*if* (readCredentialsFromFile(*username*, *password*))

{

func(username,password);

}

*static* int currentIndex = 0;

*//* *Read* *picture* *paths* *from* *a* *file* *and* *append* *them* *to* *the* *picturePaths* *list*

QStringList picturePaths;

*//* *Open* *the* *file* *containing* *picture* *paths*

QFile file("C:\\Users\\1\\OneDrive\\Desktop\\Library\_Reader\_Service\\pic\_user.txt");

*if* (!file.*open*(QIODevice::*ReadOnly* | QIODevice::*Text*)) {

qDebug() << "Failed to open file:" << file.errorString();

*return*;

}

*//* *Read* *and* *process* *each* *line* *from* *the* *file*

QTextStream in(*&file*);

*while* (!in.atEnd()) {

QString line = in.readLine();

QStringList words = line.split(","); *//* *Split* *the* *line* *into* *words*

*if* (words.size() > 1 && words.first() == username) {

*//* *Remove* *the* *first* *word* *and* *append* *the* *remaining* *words* *to* *picturePaths*

words.removeFirst();

QString path = ":/img\_book/";

*for* (int i = 0; i < words.size(); ++i) {

QString word = words.at(i);

path.append(word);

}

path.append(".png");

picturePaths.append(path);

}

}

file.*close*();

*//* *Your* *existing* *code* *for* *displaying* *pictures*

*if* (currentIndex < picturePaths.size()) {

QPixmap pixmap(picturePaths[currentIndex]);

*if* (!pixmap.isNull()) {

QGraphicsScene\* scene = *new* QGraphicsScene(*this*);

scene->addPixmap(pixmap);

ui->pic\_books->setScene(*scene*);

currentIndex = (currentIndex + 1) % picturePaths.size();

} *else* {

qDebug() << "Failed to load image at index" << currentIndex;

}

}

}

void portaldialog::**startPictureT**()

{

*//* *Create* *a* *QTimer* *for* *updating* *the* *picture* *every* *4* *seconds*

QTimer\* timer = *new* QTimer(*this*);

connect(timer, &QTimer::timeout, *this*, &portaldialog::displayNextP);

*//* *Start* *the* *timer* *with* *an* *interval* *of* *4* *seconds* *(4000* *milliseconds)*

timer->start(4000);

}

void portaldialog::**databasetofile**() {

MainWindow conn;

*if* (!conn.connOpen()) {

qDebug() << "Failed to open the database";

*return*;

}

QSqlQuery qry;

*if* (!qry.exec("SELECT \* FROM books\_table")) {

qDebug() << "Error executing query:" << qry.lastError().text();

conn.connClose(); *//* *Close* *the* *connection*

*return*;

}

QFile outputFile("C:\\Users\\1\\OneDrive\\Desktop\\Library\_Reader\_Service\\books\_data.txt");

*if* (outputFile.*open*(QIODevice::*WriteOnly* | QIODevice::*Text*)) {

QTextStream out(*&outputFile*);

*//* *Fetch* *and* *write* *data*

*while* (qry.next()) {

QStringList rowData;

*for* (int i = 0; i < qry.record().count(); ++i) {

rowData << qry.value(i).toString();

}

out << rowData.join(",") << "\n"; *//* *Separate* *arguments* *by* *comma*

}

outputFile.*close*();

qDebug() << "Data downloaded successfully to books\_data.txt";

} *else* {

qDebug() << "Error opening output file:" << outputFile.errorString();

}

conn.connClose(); *//* *Close* *the* *connection*

}

void portaldialog::**on\_minimize\_clicked**()

{

showMinimized();

}

bool portaldialog::**readCredentialsFromFile**(QString &username, QString &password)

{

QFile file("C:\\Users\\1\\OneDrive\\Desktop\\Library\_Reader\_Service\\credentials.txt");

*if* (file.*open*(QIODevice::*ReadOnly* | QIODevice::*Text*))

{

QTextStream in(*&file*);

username = in.readLine(); *//* *Read* *the* *first* *line* *(username)*

password = in.readLine(); *//* *Read* *the* *second* *line* *(password)*

file.*close*();

*return* *true*;

}

*return* *false*;

}

void portaldialog::**loadMess**()

{

QString username, password;

*if* (readCredentialsFromFile(*username*, *password*))

{

func(username, password);

}

QFile file("C:\\Users\\1\\OneDrive\\Desktop\\Library\_Reader\_Service\\Messages\_text.txt");

*if* (file.*open*(QIODevice::*ReadOnly* | QIODevice::*Text*))

{

QTextStream in(*&file*);

*while* (!in.atEnd())

{

QString line = in.readLine();

*//* *Check* *if* *the* *line* *starts* *with* *the* *username*

*if* (line.startsWith(username, Qt::*CaseInsensitive*))

{

*//* *Split* *the* *line* *into* *words*

QStringList words = line.split(' ');

*//* *Remove* *empty* *words*

words.removeAll("");

*if* (words.size() > 1)

{

words.removeFirst(); *//* *Remove* *the* *first* *word* *(username)*

QString remainingWords = words.join(' ');

ui->history\_chart->addItem(remainingWords);

}

}

}

file.*close*();

}

}

void portaldialog::**on\_message\_button\_clicked**()

{

QString data = ui->Message\_box\_2->text();

QString username, password;

*if* (readCredentialsFromFile(*username*, *password*))

{

func(username,password);

}

*//* *Append* *data* *to* *the* *file*

QFile file("C:\\Users\\1\\OneDrive\\Desktop\\Library\_Reader\_Service\\Messages\_text.txt");

*if* (file.*open*(QIODevice::*Append* | QIODevice::*Text*))

{

QTextStream out(*&file*);

out <<username+" "+ username+": "+data << "\n";

file.*close*();

}

*//* *Update* *QListWidget*

ui->history\_chart->addItem(username +": "+data);

ui->Message\_box\_2->clear();

}

void portaldialog::**on\_search\_button\_clicked**()

{

std::string search = ui->search\_edit->text().toStdString();

uploadFromF222(filenam, search, *ui->show\_up\_books*,*ui->graphicsView\_search*);

}

#ifndef REGISTERDIALOG\_H

#define REGISTERDIALOG\_H

#include "headers.h"

*namespace* Ui {

*class* registerdialog;

}

*class* registerdialog : *public* QDialog

{

Q\_OBJECT

*public*:

*explicit* registerdialog(QWidget \*parent = *nullptr*);

~registerdialog();

void Show();

void upload();

void cleanScreen();

*private* *slots*:

void on\_enter\_register\_clicked();

void on\_exit\_regi\_clicked();

*private*:

Ui::registerdialog \*ui;

std::string generateString();

QSqlDatabase mydb;

};

#endif *//* *REGISTERDIALOG\_H*

#include "registerdialog.h"

#include "ui\_registerdialog.h"

#include "mainwindow.h"

#include "hashtable.h"

#include <iostream>

#include <sstream>

#include <iomanip>

#include <ctime>

#include <cstdlib>

#include <QMessageBox>

registerdialog::registerdialog(QWidget \*parent) :

QDialog(parent),

ui(*new* Ui::registerdialog)

{

ui->setupUi(*this*);

*for*(int i=1;i<32;i++){

ui->comboBoxDay\_register->addItem(QString::number(i));

}

*for*(int i=1;i<13;i++){

ui->comboBoxMonth\_register->addItem(QString::number(i));

}

}

registerdialog::~registerdialog()

{

*delete* ui;

}

std::string registerdialog::generateString() {

*//* *Seed* *the* *random* *number* *generator* *based* *on* *the* *current* *time*

std::srand(*static\_cast*<unsigned>(std::time(*nullptr*)));

*//* *Get* *the* *current* *year*

std::time\_t now = std::time(*nullptr*);

std::tm localTime = {};

localtime\_s(&localTime, &now);

int year = localTime.tm\_year + 1900; *//* *Years* *since* *1900*

*//* *Generate* *a* *random* *4-digit* *number*

int randomDigits = rand() % 10000;

*//* *Format* *the* *string* *as* *"NNNN-YY"*

std::ostringstream oss;

oss << std::setw(4) << std::setfill('0') << randomDigits << "-";

oss << std::setw(2) << std::setfill('0') << (year % 100);

*return* oss.str();

}

void registerdialog::Show()

{

*this*->show();

}

void registerdialog::cleanScreen(){

ui->nameedit\_register->clear();

ui->birthDateedit\_register->clear();

ui->emailedit\_register->clear();

ui->passwordedit\_register->clear();

}

*//globall* *variable* *for* *table*

HashTable table;

void registerdialog::on\_enter\_register\_clicked()

{

MainWindow conn;

QString date = ui->comboBoxDay\_register->currentText();

QString month = ui->comboBoxMonth\_register->currentText();

QString year = ui->birthDateedit\_register->text();

std::string fulldate = date.toStdString().append("."+month.toStdString()+"."+year.toStdString());

std::string result = generateString();

int key = 0;

std::string finalstring;

QString selectedText = ui->comboBox\_register->currentText();

*if*(selectedText == "A - subscription only"){

std::string firststring= "A";

finalstring = firststring.append(result);

}*else* *if*(selectedText == "R - only reading hall"){

std::string fstring= "R";

finalstring = fstring.append(result);

}*else* *if*(selectedText == "B - reading hall and subsccription"){

std::string fstr= "B";

finalstring = fstr.append(result);

}

*for*(char ch : finalstring){

key += *static\_cast*<int>(ch);

}

HashNode data;

*if*(!(ui->nameedit\_register->text().isEmpty() || ui->birthDateedit\_register->text().isEmpty() || ui->emailedit\_register->text().isEmpty() || ui->passwordedit\_register->text().isEmpty())){

*if*(!isValidDate(date.toInt(),month.toInt(),year.toInt())){

ui->comment\_register->setText("Неверная дата");

}*else*{

std::string email = ui->emailedit\_register->text().toStdString();

*if*(!isValidEmail(email)){

ui->comment\_register->setText("Адрес электронной почты указан неверно");

}*else*{

QString Name,birthday,email,address,password;

Name=ui->nameedit\_register->text();

birthday=QString::fromStdString(fulldate);

email=ui->emailedit\_register->text();

address= ui->addressedit\_register->text();

password= ui->passwordedit\_register->text();

data.key = key;

std::string name = Name.toStdString();

*if* (!name.empty()) {

name[0] = std::toupper(name[0]);

data.name = name;

}

data.birthDate = fulldate;

data.email = email.toStdString();

data.address = address.toStdString();

std::string pass = password.toStdString();

data.password = pass;

table.insert(data);

*if*(!conn.connOpen()){

qDebug()<<"Failed to open the database";

*return*;

}

conn.connOpen();

QSqlQuery qry;

qry.prepare("insert into login\_data (ID,Name,Birthday,Email,Address,Password) values ('"+QString::fromStdString(finalstring)+"','"+QString::fromStdString(name)+"','"+birthday+"','"+email+"','"+address+"','"+password+"')");

*if*(qry.exec( )){

QMessageBox::critical(*this*,tr("Database"),tr("saved"));

}*else*{

QMessageBox::critical(*this*,tr("Database"),tr("not saved"));

}

ui->getpersonal\_data\_2->setText("Ваше имя пользователя: " +QString::fromStdString(finalstring) +"\n" + "Ваш пароль: " + QString::fromStdString(pass));

ui->comment\_register->setText("Регистрация прошла успешно");

conn.connClose();

cleanScreen();

}

}

}*else*{

ui->comment\_register->setText("Ваших данных недостаточно, регистрация не удалась");

}

}

void registerdialog::upload(){

}

void registerdialog::on\_exit\_regi\_clicked()

{

cleanScreen();

*if*(!mydb.open()){

ui->comment\_register->setText("Failed to open database");

}*else*{

ui->comment\_register->setText("База данных подключена.....");

}

ui->getpersonal\_data\_2->clear();

ui->getpersonal\_data\_2->setText("Примечание: Запомните свой пароль и имя пользователя,<br> которые появятся здесь");

*this*->close();

}

.