

Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных технологий

Кафедра инженерной психологии и эргономики

Дисциплина: Основы конструирования программ

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе  
на тему

**РАЗРАБОТКА ПРОГРАММЫ УЧЕТА СВЕДЕНИЙ О  
ПАЦИЕНТАХ МЕДИЦИНСКОГО ЦЕНТРА**

Студент:  
гр. 680971 Микулич В.А.

Руководитель:  
к.т.н. Меженная М.М.

Минск 2017

# СОДЕРЖАНИЕ

Содержание .....	2
1 Требования к программе.....	3
2 Конструирование программы .....	7
2.1 Разработка структуры программы .....	7
2.2 Выбор способа организации данных .....	10
2.3 Разработка перечня пользовательских функций программы .....	13
3 Разработка алгоритмов работы программы.....	17
3.1 Алгоритм функции main.....	17
3.2 Алгоритм функции Application::StartAndWaitExit .....	18
3.3 Алгоритм функции UsersRegistry::CheckUserPassword .....	19
4 Описание работы программы.....	20
4.1 Авторизация.....	20
4.2 Модуль администратора.....	21
4.3 Модуль пользователя.....	29
4.4 Исключительные ситуации .....	30
Приложение А (обязательное) Листинг кода с комментариями .....	33

# **1 ТРЕБОВАНИЯ К ПРОГРАММЕ**

Разработать программу учета сведений о пациентах медицинского центра.

Сведения о пациентах медицинского центра содержат: Ф.И.О. пациента, пол, дату рождения, место проживания (город), контактный телефон, диагнозы.

Вывести иногородних пациентов. Вывести список пациентов старше  $x$  лет, у которых диагноз  $y$  ( $x$ ,  $y$  вводятся с клавиатуры).

Реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя, как минимум три вида поиска, как минимум три вида сортировки.

## **Исходные требования к курсовой работе**

1. Язык программирования C++.
2. Среда разработки Microsoft Visual Studio версии 2010 и выше.
3. Вид приложения – консольное.
4. Парадигма программирования – объектно-ориентированная.
5. Способ организации данных – поля соответствующих классов.
6. Способ хранения данных – базы данных.
7. Каждая логически завершенная задача программы должна быть реализована в виде метода.
8. Текст пояснительной записки оформляется в соответствии со стандартом предприятия «СТП 01–2013».

## **Функциональные требования к курсовой работе**

Первым этапом работы программы является авторизация – предоставление прав. В рамках данного этапа необходимо считать данные базы данных, содержащей учетные записи пользователей следующего вида:

- login;
- password;
- role (данное поле служит для разделения в правах администраторов и пользователей).

После ввода пользователем своих персональных данных (логина и пароля) и сверки с информацией, находящейся в базе данных пользователя,

необходимо предусмотреть возможность входа в качестве администратора (в этом случае, например,  $role = 1$ ) или в качестве пользователя (в этом случае, например,  $role = 0$ ).

Если база данных с учетными записями пользователей не существует, то необходимо её программно создать и записать учетные данные администратора.

Регистрация новых пользователей при входе в систему не предусмотрена. Данную задачу выполняет администратор в режиме работы с учетными записями пользователей.

Вторым этапом работы программы является собственно работа с данными, которая становится доступной только после прохождения авторизации. Данные хранятся в отдельной базе.

Для работы с данными должны быть предусмотрены два функциональных модуля: модуль администратора и модуль пользователя.

Модуль администратора включает следующие подмодули (с указанием функциональных возможностей):

1. Управление учетными записями пользователей:
  - i. просмотр всех учетных записей;
  - ii. добавление новой учетной записи;
  - iii. редактирование учетной записи;
  - iv. удаление учетной записи.
2. Работа с файлом данных:
  - a. создание файла;
  - b. открытие существующего файла;
  - c. удаление файла.
3. Работа с данными:
  - a. режим редактирования:
    - i. просмотр всех данных;
    - ii. добавление новой записи;
    - iii. удаление записи;
    - iv. редактирование записи;
  - b. режим обработки данных:
    - i. вывести иногородних пациентов;
    - ii. вывести список пациентов старше  $x$  лет, у которых диагноз  $y$ ;
    - iii. поиск данных (как минимум три вида):
      - поиск пациентов по имени;
      - поиск пациентов по городу;

- поиск пациентов по диагнозу;
- поиск пациентов по номеру телефона;
- iv. сортировка (как минимум три вида):
  - сортировка пациентов по имени;
  - сортировка пациентов по городу;
  - сортировка пациентов по дате рождения.

Модуль пользователя включает подмодуль работы с данными со следующими функциональными возможностями:

1. Просмотр всех данных.
2. Вывести иногородних пациентов.
3. Вывести список пациентов старше x лет, у которых диагноз у.
4. Поиск данных:
  - a. поиск пациентов по имени;
  - b. поиск пациентов по городу;
  - c. поиск пациентов по диагнозу;
  - d. поиск пациентов по номеру телефона.
5. Сортировка:
  - a. сортировка пациентов по имени;
  - b. сортировка пациентов по городу;
  - c. сортировка пациентов по дате рождения.

Для реализации перечисленных модулей/подмодулей необходимо создавать меню с соответствующими пунктами.

Предусмотреть:

1. обработку исключительных ситуаций:
  - a. имя пользователя или пароль не верны;
  - b. запись с указанным идентификатором не найдена;
  - c. пользователь с таким именем уже существует;
  - d. ведённые данные не соответствуют формату поля;
2. возможность возврата назад (навигация);
3. запрос на подтверждение удаления вида «Вы действительно хотите удалить файл (запись)?»;
4. вывод сообщения об успешности создания/создания файла/записи.

### **Требования к программной реализации**

1. Все переменные и константы должны иметь осмысленные имена в рамках тематики варианта к курсовой работе.

2. Имена функций должны быть осмысленными и строиться по принципу «глагол + существительное». Если функция выполняет какую-либо проверку и возвращает результат типа `bool`, то ее название должно начинаться с глагола `is` (например, `isFileExist`, `isUnicLogin`).
3. Код не должен содержать неименованных числовых констант (так называемых «магических» чисел), неименованных строковых констант (например, имен файлов и др.). Подобного рода информацию следует выносить в глобальные переменные с атрибутом `const`. По правилам хорошего стиля программирования тексты всех информационных сообщений, выводимых пользователю в ответ на его действия, также оформляются как константы.
4. Код необходимо комментировать (как минимум в части нетривиальной логики).
5. Код не должен дублироваться – для этого существуют методы и функции.
6. Одна функция решает только одну задачу (например, не допускается в одной функции считывать данные из файла и выводить их на консоль – это две разные функции). При этом внутри функции возможен вызов других функций.
7. Следует избегать длинных функций и глубокой вложенности: текст функции должен уместиться на один экран, а вложенность блоков и операторов должна быть не более трёх.

## 2 КОНСТРУИРОВАНИЕ ПРОГРАММЫ

Реализация программы будет осуществляться на языке C++ в IDE-среде Microsoft Visual Studio 2015 Update 3. Программа будет компилироваться и использоваться в операционных системах семейства Microsoft Windows версии 7 и выше.

### 2.1 Разработка структуры программы

Согласно требованиям к программе, необходимо наличие исполняемой программы, которая работает с базой пользователей программы и базой пациентов медицинского учреждения. Следовательно, с точки зрения верхней архитектуры программы, можно выделить два основных модуля: модуль работы с базой данных пользователей и модуль работы с базой данных пациентов. На рисунке 1 показаны основные модули программы.

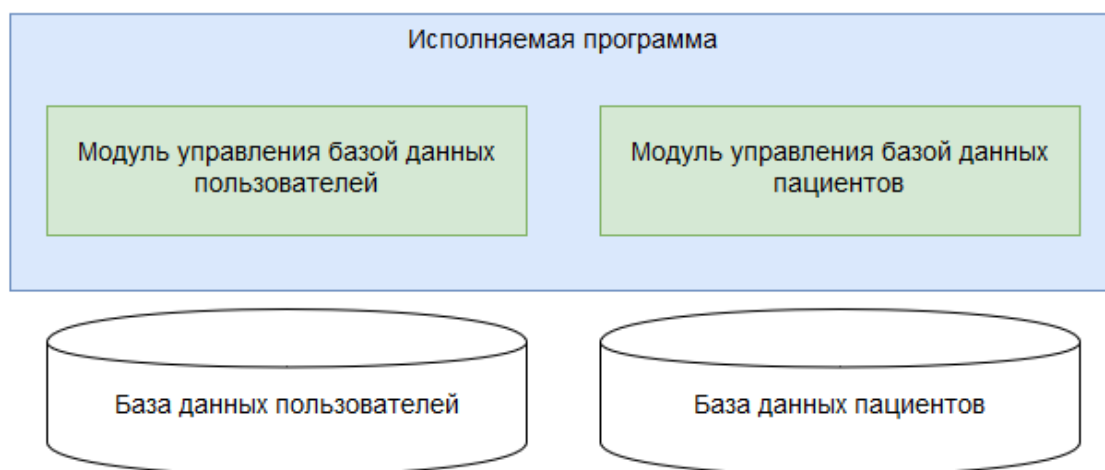


Рисунок 1 - Основные модули программы

Функция аутентификации пользователя заключается в проверке существования в базе данных пользователей введённого логина и соответствующего ему пароля. Авторизация пользователя подразумевает получение его роли из базы данных и предоставление ему соответствующих привилегий. Задачи аутентификации пользователя и запрос его роли можно реализовать в отдельном модуле либо непосредственно в модуле управления базой данных пользователей.

После успешной авторизации пользователя создаётся пользовательская сессия с соответствующими привилегиями, согласно роли пользователя. Сессия администратора имеет полный доступ к модулям управления обеими

базами данных, а сессия пользователя, в свою очередь, имеет доступ только к модулю управления базой данных пациентов в режиме «только для чтения».

Программа подразумевает наличие консольного пользовательского интерфейса. Для организации связи слоёв интерфейса пользователя, логики и доменной модели выбран паттерн «MVC» (Model-View-Controller). В качестве модели (Model) здесь выступают модули управления базами данных.

Связь сессий пользователей с модулями управления базами данных будут осуществляться через контроллеры (Controller). Так, для сессии администратора требуется доступ к контроллеру базы пользователей и к контроллеру базы пациентов, а сессии пользователя – только к контроллеру базы пациентов.

На рисунке 2 показаны взаимосвязи основных классов программы.

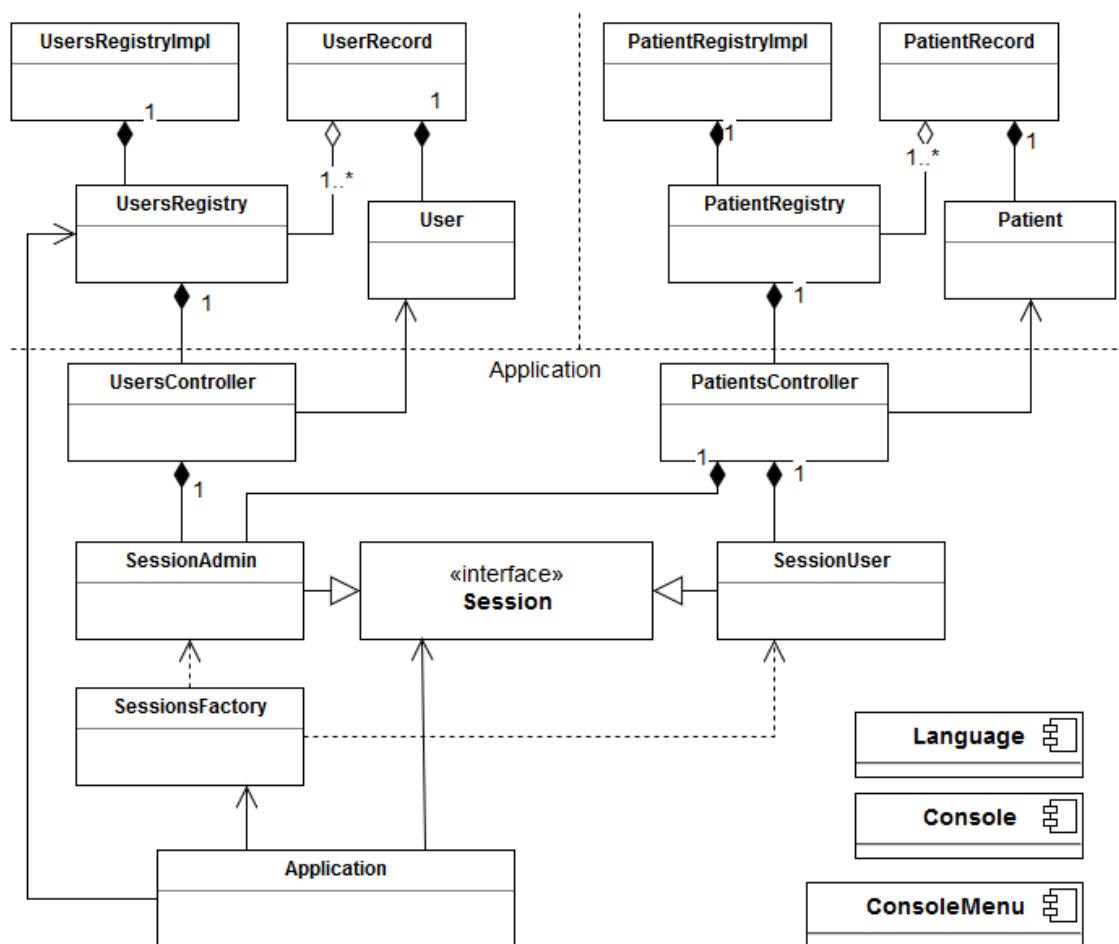


Рисунок 2 - UML-диаграмма основных классов

На рисунке 2 также обозначены компоненты, связи с которыми не отображены на диаграмме: «Language» - поставщик отображаемых в интерфейсе строк, подразумевающий возможность локализации (добавления



новых языков); «Console» - класс для работы с консолью (ввод/вывод текста); «ConsoleMenu» - компонент для работы с консольным меню пользователя.

На рисунке 3 показана структура программного проекта в контексте разработки программы в Microsoft Visual Studio для ОС Windows.

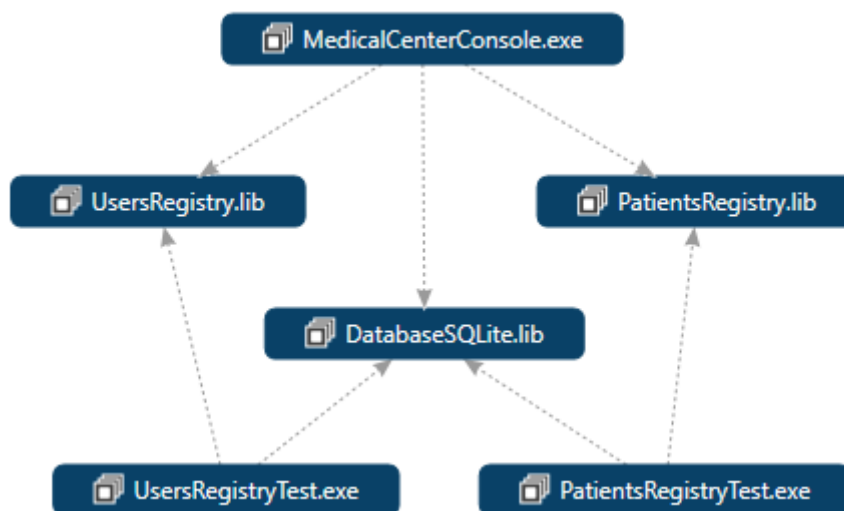


Рисунок 3 - Структура программного проекта

Описание модулей, указанных на рисунке 3:

- MedicalCenterConsole.exe – исполняемый программный модуль;
- UsersRegistry.lib – модуль управления базой данных пользователей;
- PatientsRegistry.lib – модуль управления базой данных пациентов;
- DatabaseSQLite.lib – компонентный модуль взаимодействия с СУБД SQLite;
- UsersRegistryTest.exe – unit-тест для тестирования модуля управления базой данных пользователей;
- PatientsRegistryTest.exe – unit-тест для тестирования модуля управления базой данных пациентов.

## 2.2 Выбор способа организации данных

Для представления в программе объекта пользователя вводится класс User, содержащий имя пользователя и его роль.

```
enum class UserRole
{
    Unknown,
    Admin,
    User,
};

class User
{
    std::string m_login;
    UserRole m_role;
};
```

Пароль должен храниться в памяти максимально короткое время, необходимое для вычисления секретного ключа. Потому класс не содержит поля для хранения пароля. Секретный ключ, формируемый на основе пароля, хранится в базе данных и необходим только для проверки корректности вводимого пользователем пароля при прохождении процедуры аутентификации. Следовательно, хранение секретного ключа также не требуется в полях класса User.

Для представления в программе объекта пациента вводится класс Patient, содержащий: ФИО пациента, пол, дату рождения, город проживания, номер телефона и список диагнозов.

```
enum class Gender
{
    Male,
    Feemale,
    Schemale,
};

class BirthDate
{
    uint16_t m_year;
    uint8_t m_month;
    uint8_t m_day;
};

class Patient
{
    std::string m_name;
    Gender m_gender;
    BirthDate m_birthDate;
    std::string m_city;
    std::string m_phone;
    std::vector<std::string> m_diagnoses;
};
```

В перечислении Gender указано три пола для демонстрации возможности дальнейшего расширения списка, согласно современным тенденциям.

Класс BirthDate представляет собой дату рождения.

Следует заметить, что классы предметной области User и Patient не содержат какой-либо идентификатор, сопоставляющий их с хранилищем. Вместо этого вводится понятие (тип) записи (Record), которое агрегирует идентификатор записи и объект класса. Следовательно, для пользователя и для пациента вводятся следующие типы соответственно:

```
using UserId_t = uint64_t;  
using UserRecord_t = std::pair<UserId_t, User>;  
  
using PatientId_t = uint64_t;  
using PatientRecord_t = std::pair<PatientId_t, Patient>;
```

Для работы со списками объектов будет использоваться наиболее эффективный для списков переменной длины STL-контейнер «vector».

В качестве СУБД выбрана встраиваемая бесплатная библиотека SQLite с открытым исходным кодом, так как её база данных располагается в одном файле, что наиболее близко соответствует заданию.

Для хранения текстовых данных выбрана кодировка UTF-8, чтобы не привязывать данные и программу к локальной кодировке операционной системы и иметь возможность хранения и обработки данных на разных языках. Для выборки данных по ряду текстовых условий (поиск без учёта регистра, поиск подстроки в строке) библиотеку SQLite требуется скомпилировать в режиме интеграции с библиотекой ICU (указать ключ «-D SQLITE\_ENABLE\_ICU» при компиляции и предоставить .LIB- и .DLL-модули библиотеки ICU). Библиотека ICU представляет собой набор программных компонент с открытым кодом для поддержки в программном обеспечении интернационализации (I18N) и глобализации (G11N), включающий полноценную поддержку стандарта Unicode.

Программа обслуживает данные, хранимые в двух базах данных: базе данных пользователей и базе данных пациентов. Эти базы имеют различные схемы данных.

Полный SQL-код схем баз данных приведён в «Приложение А».

### 2.2.1 Схема базы данных пользователей

Для базы данных пользователей требуется хранить имя пользователя (login) и некий ключ его пароля (пароль в открытом виде хранить нельзя с точки зрения безопасности). Роли пользователей вынесены в отдельную таблицу. Схема базы данных пользователей показана на рисунке 4.

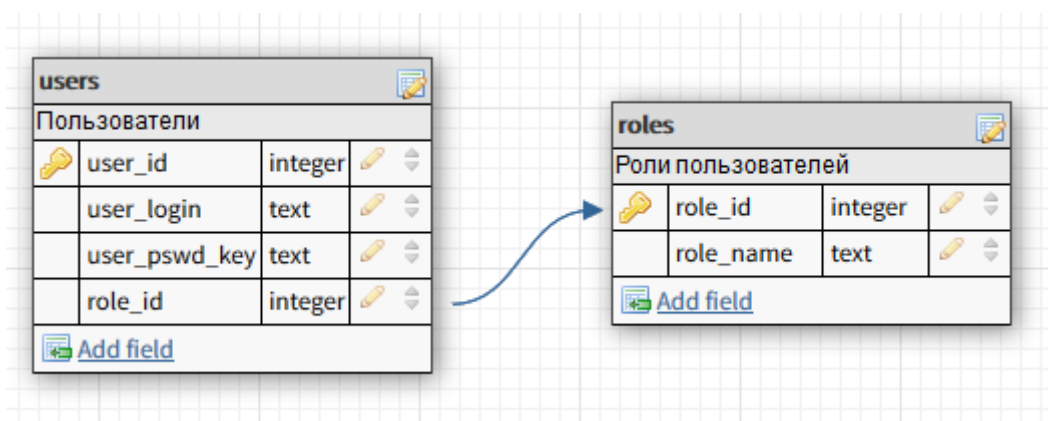


Рисунок 4 - Схема базы данных пользователей

### 2.2.2 Схема базы данных пациентов

База данных пациентов, согласно требованиям, должна содержать следующую информацию:

- ФИО пациента;
- дата рождения пациента;
- пол пациента;
- город проживания;
- номер телефона;
- диагнозы.

Нормализация схемы данных пользователей: города, пол и диагнозы выносятся в отдельные таблицы и связываются с таблицей пациентов. Так как диагнозы вынесены в отдельную таблицу и могут повторяться у разных пациентов, для обеспечения возможности указывать для одного пациента несколько диагнозов, организуется связь «многие ко многим» посредством ввода дополнительной таблицы «patients\_diagnoses».

Схема базы данных пациентов показана на рисунке 5.

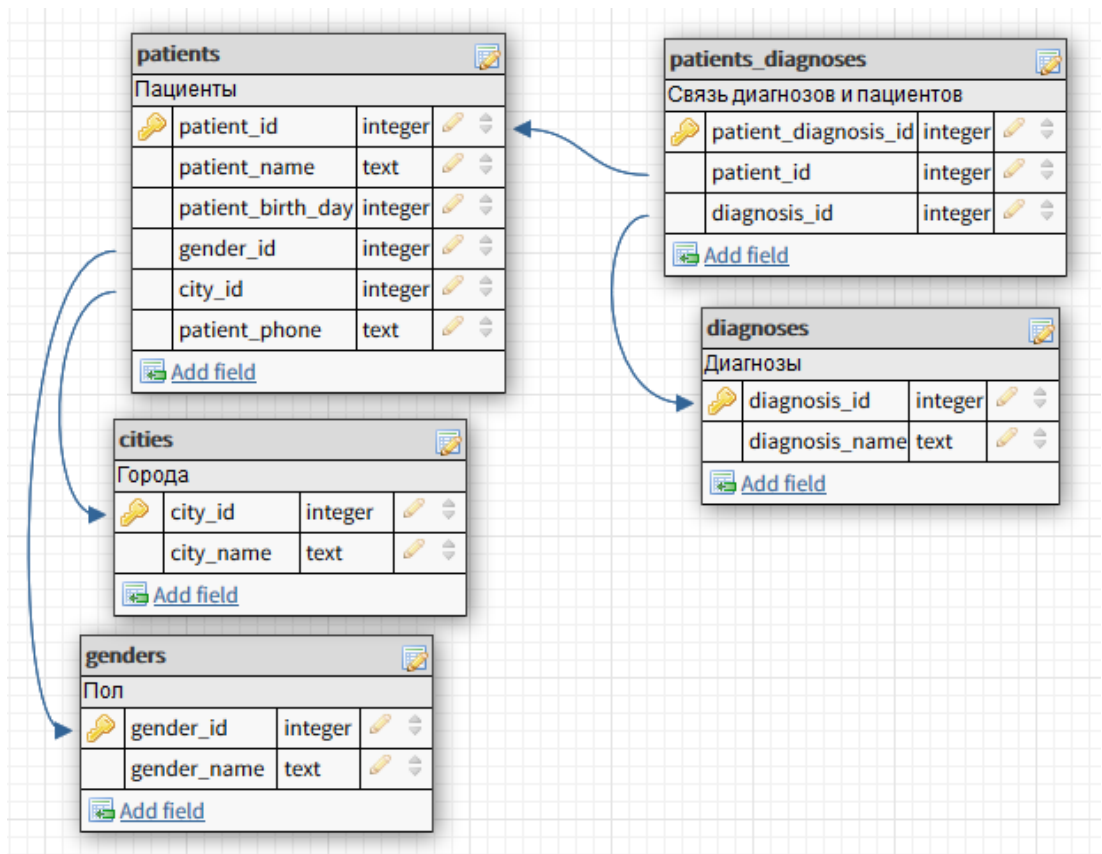


Рисунок 5 - Схема базы данных пациентов

## 2.3 Разработка перечня пользовательских функций программы

Ниже приведено описание публичных интерфейсов всех основных классов.

```
class User
{
public:
    const std::string& GetLogin() const;
    const UserRole GetRole() const;
};

using UserId_t = uint64_t;
using UserRecord_t = std::pair<UserId_t, User>;

class UsersRegistry
{
public:
    uint64_t GetAllUsersCount() const;
    uint64_t GetAdminUsersCount() const;
    const std::vector<UserRecord_t> GetAllUsers() const;
    bool AddUser(const User& user, const std::string& password);
    bool DeleteUser(const UserId_t userId);
    bool DeleteUserByLogin(const std::string& login);
    bool UpdateUser(const UserId_t userId, const User& user);
    bool UpdateUserAndPassword(const UserId_t userId, const User& user, const
std::string& password);
```

```

    bool IsExistUser(const std::string& login) const;
    UserRecord_t GetUserByLogin(const std::string& login) const;
    UserRecord_t GetUserById(const UserId_t userId) const;

    bool CheckUserPassword(const std::string& login, const std::string& password)
const;
};

class UsersController
{
public:
    void PrintAllUsers() const;
    void AddNewUser();
    void RemoveUser();
    void EditUser();
};

class Patient
{
public:
    const std::string& GetName() const;
    const Gender GetGender() const;
    const BirthDate& GetBirthDate() const;
    const std::string& GetCity() const;
    const std::string& GetPhone() const;
    const std::vector<std::string>& GetDiagnoses() const;

    const std::string GetDiagnosesAsString() const;
};

using PatientId_t = uint64_t;
using PatientRecord_t = std::pair<PatientId_t, Patient>;

class PatientsRegistry
{
public:
    const std::vector<PatientRecord_t> GetAllPatients() const;
    void AddPatient(const Patient& patient);
    bool UpdatePatient(const PatientId_t patientId, const Patient& patient);
    bool DeletePatient(const PatientId_t patientId);
    PatientRecord_t GetPatientById(const PatientId_t patientId);

    const std::vector<PatientRecord_t> SearchPatientsByName(const std::string&
name) const;
    const std::vector<PatientRecord_t> SearchPatientsByCity(const std::string&
city) const;
    const std::vector<PatientRecord_t> SearchPatientsByDiagnose(const std::string&
diagnose) const;
    const std::vector<PatientRecord_t> SearchPatientsByPhone(const std::string&
phone) const;

    const std::vector<PatientRecord_t> OrderPatientsByName() const;
    const std::vector<PatientRecord_t> OrderPatientsByCity() const;
    const std::vector<PatientRecord_t> OrderPatientsByBirthDate() const;

    const std::vector<PatientRecord_t> ViewNonresidentPatients(const std::string&
city) const;
    const std::vector<PatientRecord_t> ViewPatientsByAgeAndDiagnose(uint16_t age,
const std::string& diagnose) const;
};

class PatientsController

```

```

{
public:
    void OpenFileForReadAndWrite();
    void OpenFileForReadOnly();
    void CreateFile();
    void CloseFile();
    void RemoveFile();

    void ViewAllPatients() const;
    void AddPatient();
    void EditPatient();
    void RemovePatient();

    void SearchPatientsByName() const;
    void SearchPatientsByCity() const;
    void SearchPatientsByDiagnose() const;
    void SearchPatientsByPhone() const;

    void OrderPatientsByName() const;
    void OrderPatientsByCity() const;
    void OrderPatientsByBirthDate() const;

    void ViewNonresidentPatients() const;
    void ViewPatientsByAgeAndDiagnose() const;
};

class Session
{
public:
    virtual void StartSessionAndWaitExit() = 0;
};

class SessionAdmin : public Session
{
    void StartSessionAndWaitExit() override;
};

class SessionUser : public Session
{
public:
    void StartSessionAndWaitExit() override;
};

class SessionsFactory
{
public:
    std::shared_ptr<Session> CreateSession(const std::string& userName, const
UserRole userRole,
        std::shared_ptr<PatientsRegistry> patientsRegistry);
};

class ConsoleMenu
{
    struct MenuItem
    {
        std::string name;
        std::function<void()> action;
    };

public:

```

```

    void AddItem(const std::string& itemName, std::function<void()> action);
    void AddMenu(const std::string& itemName, const ConsoleMenu menu);

    void StartMenuAndWaitExit() const;
};

class Language
{
public:
    const char* GetString(const LanguageString languageStringId) const;
};

class Application
{
public:
    void StartAndWaitExit();
};

```



### 3 РАЗРАБОТКА АЛГОРИТМОВ РАБОТЫ ПРОГРАММЫ

#### 3.1 Алгоритм функции main

Функция main является точкой входа в программу, вызывает основные функции инициализации, создаёт объект класса Application и передаёт ему управление. На рисунке 6 показана блок-схема алгоритма функции main.



Рисунок 6 - Блок-схема алгоритма функции main

### 3.2 Алгоритм функции Application::StartAndWaitExit

Метод StartAndWaitExit класса Application авторизует пользователя и запускает сеанс диалогового взаимодействия с пользователем (рисунок 7).

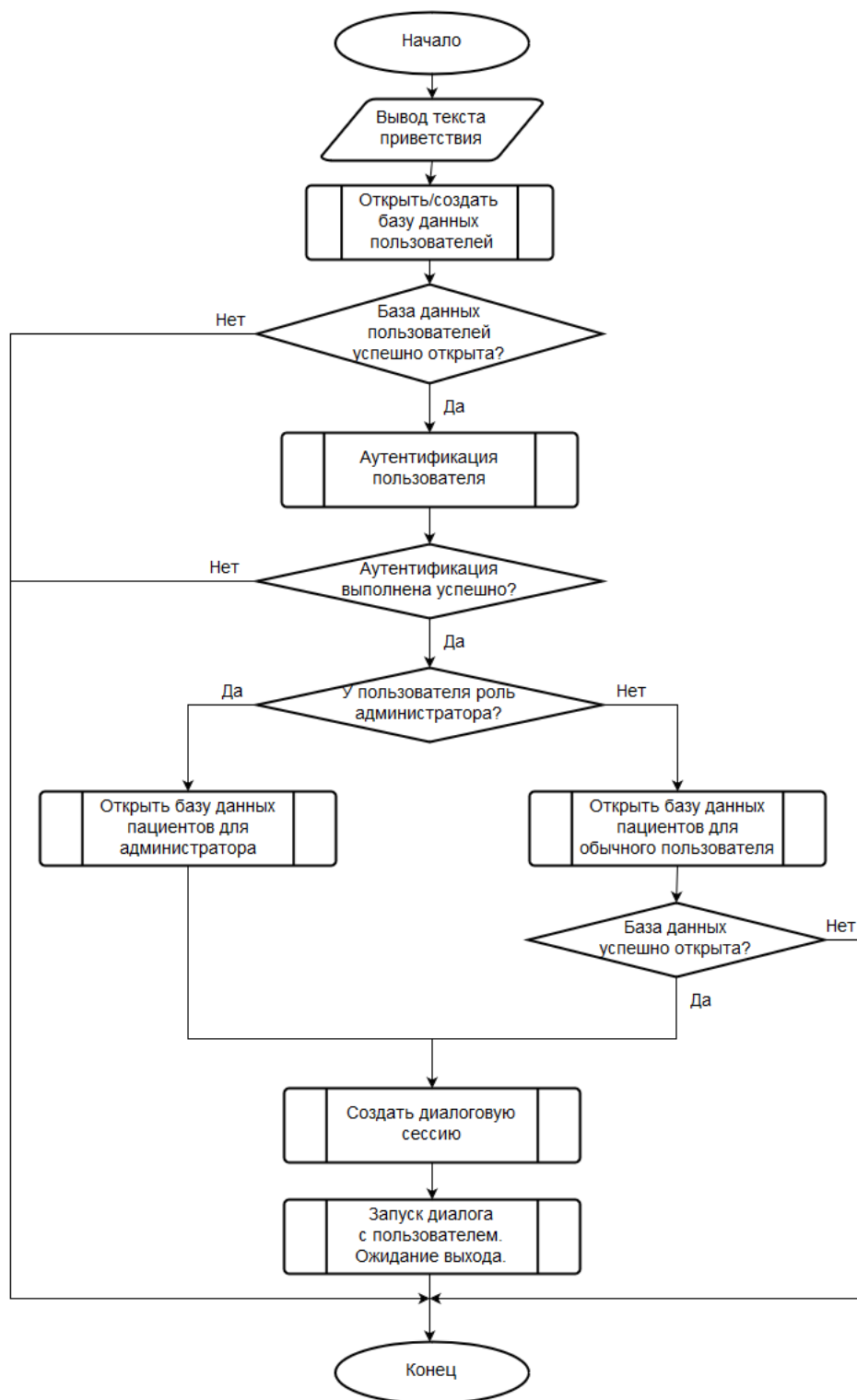


Рисунок 7 - Блок-схема алгоритма метода Application::StartAndWaitExit

### 3.3 Алгоритм функции UsersRegistry::CheckUserPassword

Проверка правильности ввода логина пользователя и пароля при аутентификации заключается в поиске по базе данных записи с указанным логином и ключом, полученным из пароля.

С целью защиты программы от атаки методом «грубой силы» (перебор паролей), для вычисления ключа выбран алгоритм PKDF2 (Password-Based Key Derivation Function), являющийся частью спецификации PKCS #5 v2.0 (RFC 2898). В качестве источника псевдослучайных чисел выбрана HMAC-функция с применением хеш-функции SHA512.

Алгоритм метода CheckUserPassword класса UsersRegistry показан на рисунке 8.

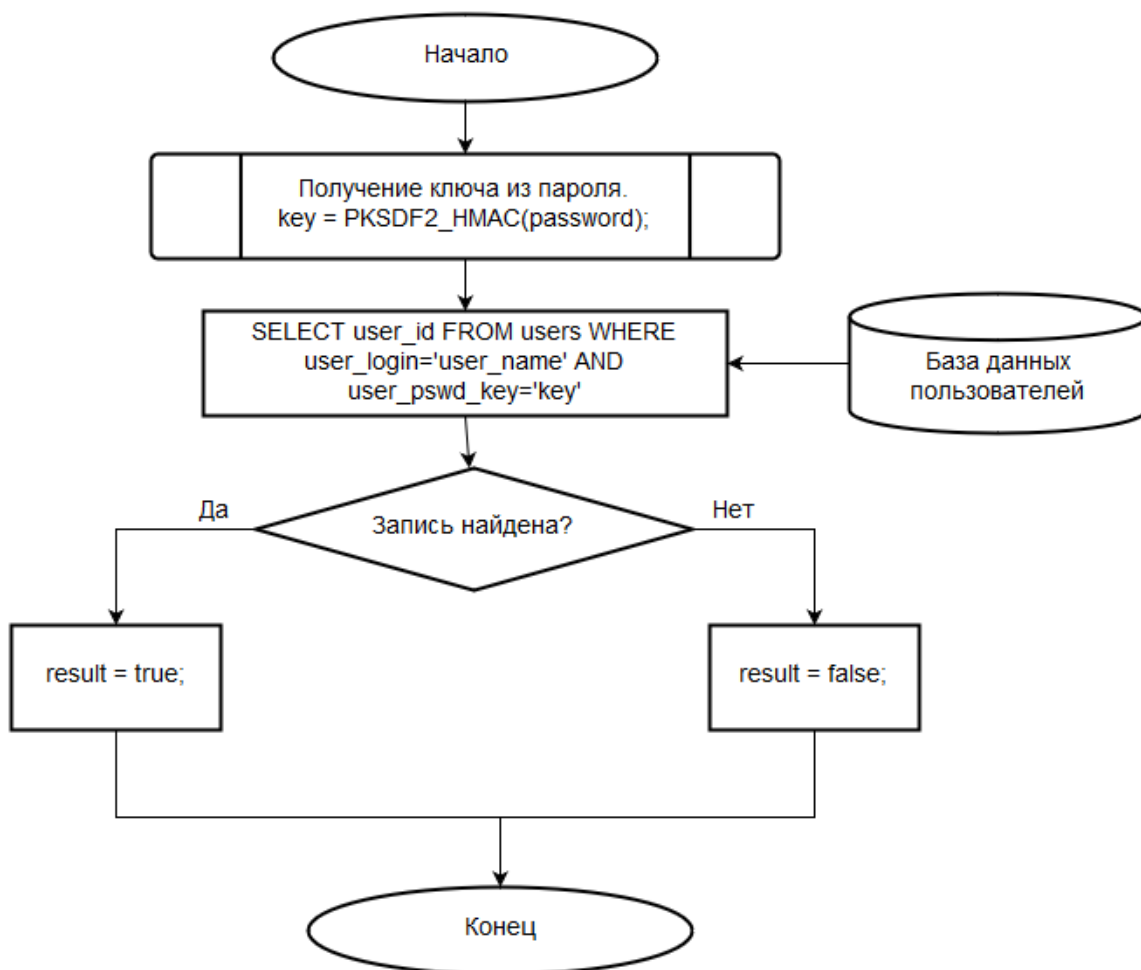


Рисунок 8 - Блок-схема алгоритма метода UsersRegistry::CheckUserPassword

## 4 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

При запуске программа печатает приветствие и пытается найти и открыть базу данных пользователей. Если база не найдена, то программа предлагает создать её. При создании новой базы необходимо сразу добавить в неё учётную запись администратора.

Поиск и создание файла базы данных пользователей производится в текущей директории программы. Имя файла жёстко прописано в коде программы: «MedicalCenterUsers.db».

Если база данных успешно открыта, то программа проверяет в ней наличие учётных записей администратора и, если их нет, предлагает добавить (рисунок 9).

```
* Программа учета сведений о пациентах медицинского центра.
* ИИТ БГУИР. Факультет Компьютерных Технологий.
* Специальность "Инженерно-психологическое обеспечение информационных технологий" (ИПОИТ).
* Курсовой проект по дисциплине "Основы конструирования программ".
* Выполнил: Студент 1го курса группы 680971, Микулич Василий Александрович.
* Руководитель: Меженная Марина Михайловна

Первый запуск. Будет создана новая база данных пользователей.
Продолжить? (y/n) [y]: y
При создании базы данных будет создана учётная запись администратора.
Введите данные для новой учётной записи.
Имя пользователя (login): admin
Пароль: *****
Повторите ввод пароля: *****
База данных пользователей успешно создана.
Учётная запись администратора успешно добавлена в базу данных пользователей.
```

Рисунок 9 - Создание базы данных пользователей при запуске программы

### 4.1 Авторизация

После успешного открытия базы данных пользователей (только для чтения) программа запрашивает учётные записи пользователя (логин и пароль) для его авторизации до тех пор, пока авторизация не будет успешно пройдена, либо пока пользователь не решит отменить процедуру входа: в этом случае программа завершит работу.

Для проверки корректности ввода логина и пароля программа обращается напрямую к модулю базы пользователей, открытой только для чтения, который инкапсулирует алгоритм обработки и проверки пароля и его соответствия имени пользователя.

После успешной аутентификации пользователя программа получает из базы данных пользователей роль аутентифицированного пользователя и авторизует его, создав соответствующую роли сессию: сессию администратора, которой будет соответствовать в дальнейшем описании модуль администратора либо сессию пользователя, которой будет соответствовать модуль пользователя.

Если пользователь авторизован с правами администратора, то программа выведет соответствующее сообщение (рисунок 10).

```
Имя пользователя: admin
Пароль: *****
Вход в систему выполнен успешно.
Вы зашли в систему с правами администратора.
```

Рисунок 10 - Авторизация пользователя (администратора)

## 4.2 Модуль администратора

После входа администратора программа, для удобства, автоматически открывает базу данных пациентов «по умолчанию», если она есть (рисунок 11). Файл базы данных «по умолчанию» должен находиться в текущей директории программы, а его имя зашифровано в коде программы: «MedicalCenterPatients.db».

```
Имя пользователя: admin
Пароль: *****
Вход в систему выполнен успешно.
Вы зашли в систему с правами администратора.
Создан и открыт файл базы данных пациентов по умолчанию: MedicalCenterPatients.db.
```

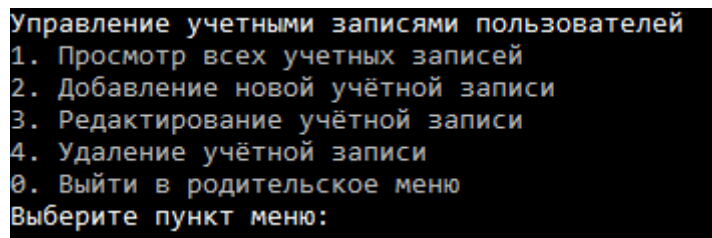
Рисунок 11 - Автоматическое открытие базы данных пациентов в модуле администратора

Далее программа запускает диалоговое взаимодействие с пользователем путём отображения пронумерованных пунктов меню и запросом у пользователя ввода желаемого номера. На рисунке 12 показано главное меню модуля администратора.

```
1. Управление учетными записями пользователей
2. Управление файлами данных
3. Обработка данных
0. Выйти из программы
Выберите пункт меню:
```

Рисунок 12 - Главное меню модуля администратора

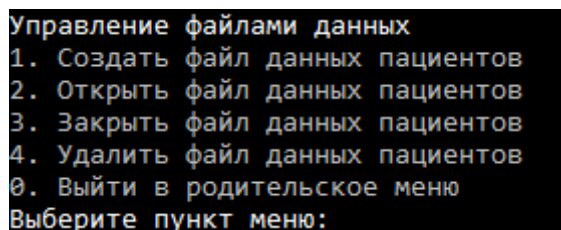
На рисунке 13 показано меню «Управление учётными записями пользователей».



```
Управление учетными записями пользователей
1. Просмотр всех учетных записей
2. Добавление новой учётной записи
3. Редактирование учётной записи
4. Удаление учётной записи
0. Выйти в родительское меню
Выберите пункт меню:
```

Рисунок 13 - Меню «Управление учётными записями пользователей»

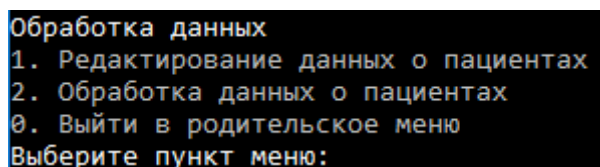
На рисунке 14 показано меню «Управление файлами данных».



```
Управление файлами данных
1. Создать файл данных пациентов
2. Открыть файл данных пациентов
3. Закрыть файл данных пациентов
4. Удалить файл данных пациентов
0. Выйти в родительское меню
Выберите пункт меню:
```

Рисунок 14 - Меню «Управление файлами данных»

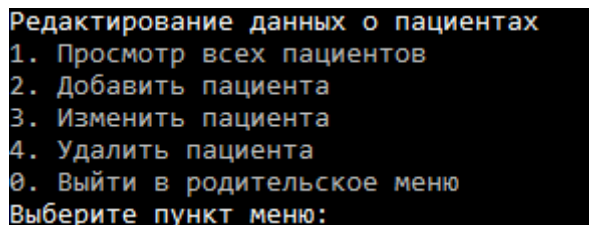
На рисунке 15 показано меню «Обработка данных».



```
Обработка данных
1. Редактирование данных о пациентах
2. Обработка данных о пациентах
0. Выйти в родительское меню
Выберите пункт меню:
```

Рисунок 15 - Меню «Обработка данных»

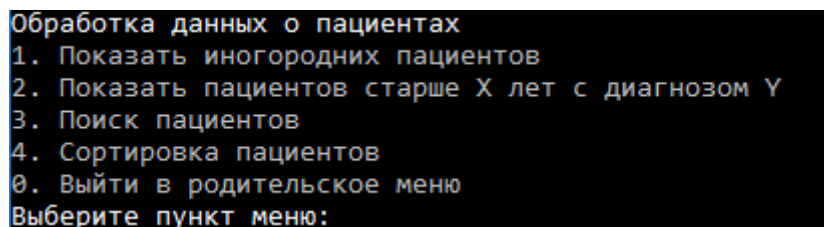
На рисунке 16 показано меню «Редактирование данных о пациентах».



```
Редактирование данных о пациентах
1. Просмотр всех пациентов
2. Добавить пациента
3. Изменить пациента
4. Удалить пациента
0. Выйти в родительское меню
Выберите пункт меню:
```

Рисунок 16 - Меню «Редактирование данных о пациентах»

На рисунке 17 показано меню «Обработка данных о пациентах».



```
Обработка данных о пациентах
1. Показать иногородних пациентов
2. Показать пациентов старше X лет с диагнозом Y
3. Поиск пациентов
4. Сортировка пациентов
0. Выйти в родительское меню
Выберите пункт меню:
```

Рисунок 17 - Меню «Обработка данных о пациентах»

На рисунке 18 показано меню «Поиск пациентов».

```

Поиск пациентов
1. Поиск пациентов по имени
2. Поиск пациентов по городу
3. Поиск пациентов по диагнозу
4. Поиск пациентов по номеру телефона
0. Выйти в родительское меню
Выберите пункт меню:

```

Рисунок 18 - Меню «Поиск пациентов»

На рисунке 19 показано меню «Сортировка пациентов».

```

Сортировка пациентов
1. Сортировка пациентов по имени
2. Сортировка пациентов по городу
3. Сортировка пациентов по дате рождения
0. Выйти в родительское меню
Выберите пункт меню:

```

Рисунок 19 - Меню «Сортировка пациентов»

На рисунке 20 показан пример вывода всех учётных записей пользователей.

```

Управление учетными записями пользователей
1. Просмотр всех учетных записей
2. Добавление новой учётной записи
3. Редактирование учётной записи
4. Удаление учётной записи
0. Выйти в родительское меню
Выберите пункт меню: 1
-----
ID: 1, Имя пользователя: admin, Роль: Администратор
ID: 2, Имя пользователя: vm, Роль: Администратор
ID: 3, Имя пользователя: vm2, Роль: Пользователь
ID: 4, Имя пользователя: user, Роль: Пользователь
-----
Найдено записей: 4

```

Рисунок 20 - Просмотр всех учётных записей пользователей

На рисунке 21 показан пример добавления новой учётной записи пользователя.

```

Управление учетными записями пользователей
1. Просмотр всех учетных записей
2. Добавление новой учётной записи
3. Редактирование учётной записи
4. Удаление учётной записи
0. Выйти в родительское меню
Выберите пункт меню: 2
Предоставить пользователю права администратора? (y/n) [n]: y
Введите данные для новой учётной записи.
Имя пользователя (login): vm3
Пароль: *
Повторите ввод пароля: *
Новый пользователь успешно добавлен.

```

Рисунок 21 - Добавление новой учётной записи пользователя

На рисунке 22 показан пример изменения существующей учётной записи пользователя.

```
Управление учетными записями пользователей
1. Просмотр всех учетных записей
2. Добавление новой учётной записи
3. Редактирование учётной записи
4. Удаление учётной записи
0. Выйти в родительское меню
Выберите пункт меню: 3
Введите ID пользователя: 5
Пользователь найден:
ID: 5, Имя пользователя: vm3, Роль: Администратор
Вы действительно хотите изменить этого пользователя? (y/n) [n]: y
Изменить роль пользователя? (y/n) [n]: y
Предоставить пользователю права администратора? (y/n) [n]: n
Изменить имя пользователя? (y/n) [n]: y
Введите новое имя пользователя (login): vm4
Изменить пароль пользователя? (y/n) [n]: y
Введите новый пароль: *
Повторите ввод нового пароля: *
Пользователь успешно изменён.
```

Рисунок 22 - Изменение учётной записи пользователя

На рисунке 23 показан пример удаления существующей учётной записи пользователя.

```
Управление учетными записями пользователей
1. Просмотр всех учетных записей
2. Добавление новой учётной записи
3. Редактирование учётной записи
4. Удаление учётной записи
0. Выйти в родительское меню
Выберите пункт меню: 4
Введите ID пользователя: 5
Пользователь найден:
ID: 5, Имя пользователя: vm4, Роль: Пользователь
Вы действительно хотите удалить этого пользователя? (y/n) [n]: y
Пользователь успешно удалён.
```

Рисунок 23 - Удаление учётной записи пользователя

На рисунке 24 показан пример создания нового файла данных с пациентами. После успешного создания нового файла он автоматически открывается (становится активным).

```
Управление файлами данных
1. Создать файл данных пациентов
2. Открыть файл данных пациентов
3. Закрыть файл данных пациентов
4. Удалить файл данных пациентов
0. Выйти в родительское меню
Выберите пункт меню: 1
Введите путь к файлу базы данных пациентов: D:\1.db
Файл успешно создан и открыт.
```

Рисунок 24 - Создание файла данных



На рисунке 25 показан пример закрытия открытого файла данных с пациентами.

```
Управление файлами данных
1. Создать файл данных пациентов
2. Открыть файл данных пациентов
3. Закрыть файл данных пациентов
4. Удалить файл данных пациентов
0. Выйти в родительское меню
Выберите пункт меню: 3
Файл закрыт.
```

Рисунок 25 - Закрытие файла данных

На рисунке 26 показан пример открытия существующего файла данных с пациентами. Если до этого был открыт другой файл, то он будет автоматически закрыт.

```
Управление файлами данных
1. Создать файл данных пациентов
2. Открыть файл данных пациентов
3. Закрыть файл данных пациентов
4. Удалить файл данных пациентов
0. Выйти в родительское меню
Выберите пункт меню: 2
Введите путь к файлу базы данных пациентов: d:\1.db
Файл успешно открыт.
```

Рисунок 26 - Открытие файла данных

На рисунке 27 показан пример удаления существующего файла данных с пациентами. Если пользователь пытается удалить открытый файл, то программа выведет соответствующее предупреждение и откажет в удалении.

```
Управление файлами данных
1. Создать файл данных пациентов
2. Открыть файл данных пациентов
3. Закрыть файл данных пациентов
4. Удалить файл данных пациентов
0. Выйти в родительское меню
Выберите пункт меню: 4
Введите путь к файлу базы данных пациентов: d:\1.db
Вы действительно хотите удалить файл? (y/n) [n]: y
Файл успешно удалён.
```

Рисунок 27 - Удаление файла данных

На рисунке 28 показан пример добавления новой записи о пациенте в открытый файл базы данных пациентов. Если при попытке добавления новой записи открытых (активных) файлов не будет, то программа выведет соответствующее предупреждение.

```

Редактирование данных о пациентах
1. Просмотр всех пациентов
2. Добавить пациента
3. Изменить пациента
4. Удалить пациента
0. Выйти в родительское меню
Выберите пункт меню: 2
ФИО пациента: Микулич Всеий Александрович
Пол (м/ж/т): м
Дата рождения (ДД.ММ.ГГГГ): 27.06.1984
Город: Минск
Телефон: +375292287777
Диагнозы (через запятую): микролит, сколиоз
Пациент успешно добавлен.

```

Рисунок 28 - Добавление пациента

На рисунке 29 показан пример вывода всех записей пациентов, содержащихся в открытом файле базы данных с пациентами.

```

Редактирование данных о пациентах
1. Просмотр всех пациентов
2. Добавить пациента
3. Изменить пациента
4. Удалить пациента
0. Выйти в родительское меню
Выберите пункт меню: 1
-----
ID: 1; Имя: Микулич Всеий Александрович; Пол: Мужчина; Дата рождения: 27.06.1984; Город: Минск; Телефон: +375292287777;
Диагнозы: микролит, сколиоз
ID: 2; Имя: Киррилов Киррил Киррилович; Пол: Транссексуал; Дата рождения: 26.12.1983; Город: Лунинец; Телефон: +37529212
0201; Диагнозы: аутизм
-----
Найдено записей: 2

```

Рисунок 29 - Просмотр всех записей о пациентах

На рисунке 30 показан пример изменения существующей записи о пациенте в открытом файле базы данных с пациентами.

```

Редактирование данных о пациентах
1. Просмотр всех пациентов
2. Добавить пациента
3. Изменить пациента
4. Удалить пациента
0. Выйти в родительское меню
Выберите пункт меню: 3
Введите ID пациента: 2
Пациент найден:
ID: 2; Имя: Киррилов Киррил Киррилович; Пол: Транссексуал; Дата рождения: 26.12.1983; Город: Лунинец; Телефон: +37529212
0201; Диагнозы: аутизм
Вы действительно хотите изменить этого пациента? (y/n) [y]: д
ФИО пациента: Киррилов Киррил Киррилович
Пол (м/ж/т): м
Дата рождения (ДД.ММ.ГГГГ): 26.12.1983
Город: Микашевичи
Телефон: +375292120201
Диагнозы (через запятую): аутизм, шизофрения
Пациент успешно изменён.

```

Рисунок 30 - Изменение записи о пациенте

На рисунке 31 показан пример удаления существующей записи о пациенте в открытом файле базы данных с пациентами.

```

Редактирование данных о пациентах
1. Просмотр всех пациентов
2. Добавить пациента
3. Изменить пациента
4. Удалить пациента
0. Выйти в родительское меню
Выберите пункт меню: 4
Введите ID пациента: 3
Пациент найден:
ID: 3; Имя: ыффыв; Пол: Мужчина; Дата рождения: 12.12.1233; Город: ыфв; Телефон: 123; Диагнозы:
Вы действительно хотите удалить этого пациента? (y/n) [n]: y
Пациент успешно удалён.

```

Рисунок 31 - Удаление записи о пациенте

На рисунке 32 показан пример вывода иногородних пациентов, содержащихся в открытом файле базы данных с пациентами. Так как программа не хранит информацию о текущем городе, то она запрашивает у пользователя ввод города и выводит на экран всех пациентов, которые проживают в других городах.

```

Обработка данных о пациентах
1. Показать иногородних пациентов
2. Показать пациентов старше X лет с диагнозом Y
3. Поиск пациентов
4. Сортировка пациентов
0. Выйти в родительское меню
Выберите пункт меню: 1
Введите название города: минск
-----
ID: 2; Имя: Кирилл Кирил Кирилович; Пол: Мужчина; Дата рождения: 26.12.1983; Город: Микашевичи; Телефон: +3752921202
01; Диагнозы: аутизм, шизофрения
-----
Найдено записей: 1

```

Рисунок 32 - Показ иногородних пациентов

На рисунке 33 показан пример вывода пациентов старше какого-то возраста (вводит пользователь) и имеющих в записи некоторый диагноз (вводит пользователь).

```

Обработка данных о пациентах
1. Показать иногородних пациентов
2. Показать пациентов старше X лет с диагнозом Y
3. Поиск пациентов
4. Сортировка пациентов
0. Выйти в родительское меню
Выберите пункт меню: 2
Введите диагноз: аутизм
Введите возраст (полных лет): 33
-----
ID: 2; Имя: Кирилл Кирил Кирилович; Пол: Мужчина; Дата рождения: 26.12.1983; Город: Микашевичи; Телефон: +3752921202
01; Диагнозы: аутизм, шизофрения
-----
Найдено записей: 1

```

Рисунок 33 - Показ пациентов старше X лет с диагнозом Y

На рисунке 34 показан пример поиска пациентов по имени. Программа выполняет поиск вхождения введённого значения без учёта регистра.

```

Поиск пациентов
1. Поиск пациентов по имени
2. Поиск пациентов по городу
3. Поиск пациентов по диагнозу
4. Поиск пациентов по номеру телефона
0. Выйти в родительское меню
Выберите пункт меню: 1
Введите имя пациента: Микучи
-----
ID: 1; Имя: Микучи Всеий Александрович; Пол: Мужчина; Дата рождения: 27.06.1984; Город: Минск; Телефон: +375292287777;
Диагнозы: микролит, сколиоз
-----
Найдено записей: 1

```

Рисунок 34 - Поиск пациентов по имени

На рисунке 35 показан пример поиска пациентов по городу. Программа выполняет поиск вхождения введённого значения без учёта регистра.

```
Поиск пациентов
1. Поиск пациентов по имени
2. Поиск пациентов по городу
3. Поиск пациентов по диагнозу
4. Поиск пациентов по номеру телефона
0. Выйти в родительское меню
Выберите пункт меню: 2
Введите название города: Мика
-----
ID: 2; Имя: Кирилл Кириллович; Пол: Мужчина; Дата рождения: 26.12.1983; Город: Микашевичи; Телефон: +375292120201; Диагнозы: аутизм, шизофрения
-----
Найдено записей: 1
```

Рисунок 35 - Поиск пациентов по городу

На рисунке 36 показан пример поиска пациентов по диагнозу. Программа выполняет поиск вхождения введённого значения без учёта регистра.

```
Поиск пациентов
1. Поиск пациентов по имени
2. Поиск пациентов по городу
3. Поиск пациентов по диагнозу
4. Поиск пациентов по номеру телефона
0. Выйти в родительское меню
Выберите пункт меню: 3
Введите диагноз: микроли
-----
ID: 1; Имя: Микулич Всеий Александрович; Пол: Мужчина; Дата рождения: 27.06.1984; Город: Минск; Телефон: +375292287777; Диагнозы: микролит, сколиоз
-----
Найдено записей: 1
```

Рисунок 36 - Поиск пациентов по диагнозу

На рисунке 37 показан пример поиска пациентов по номеру телефона. Программа выполняет поиск вхождения введённого значения без учёта регистра.

```
Поиск пациентов
1. Поиск пациентов по имени
2. Поиск пациентов по городу
3. Поиск пациентов по диагнозу
4. Поиск пациентов по номеру телефона
0. Выйти в родительское меню
Выберите пункт меню: 4
Введите номер телефона: 1202
-----
ID: 2; Имя: Кирилл Кириллович; Пол: Мужчина; Дата рождения: 26.12.1983; Город: Микашевичи; Телефон: +375292120201; Диагнозы: аутизм, шизофрения
-----
Найдено записей: 1
```

Рисунок 37 - Поиск пациентов по номеру телефона

На рисунке 38 показан пример вывода пациентов, отсортированных по имени в порядке возрастания.

```
Сортировка пациентов
1. Сортировка пациентов по имени
2. Сортировка пациентов по городу
3. Сортировка пациентов по дате рождения
0. Выйти в родительское меню
Выберите пункт меню: 1
-----
ID: 4; Имя: Анохин Пётр Оксанович; Пол: Транссексуал; Дата рождения: 12.03.2001; Город: Заславль; Телефон: ; Диагнозы: кариес
ID: 2; Имя: Кирилл Кириллович; Пол: Мужчина; Дата рождения: 26.12.1983; Город: Микашевичи; Телефон: +375292120201; Диагнозы: аутизм, шизофрения
ID: 1; Имя: Микулич Всеий Александрович; Пол: Мужчина; Дата рождения: 27.06.1984; Город: Минск; Телефон: +375292287777; Диагнозы: микролит, сколиоз
-----
Найдено записей: 3
```

Рисунок 38 - Сортировка пациентов имени

На рисунке 39 показан пример вывода пациентов, отсортированных по городу в порядке возрастания.

```
Сортировка пациентов
1. Сортировка пациентов по имени
2. Сортировка пациентов по городу
3. Сортировка пациентов по дате рождения
0. Выйти в родительское меню
Выберите пункт меню: 2
-----
ID: 4; Имя: Анохин Пётр Оксанович; Пол: Транссексуал; Дата рождения: 12.03.2001; Город: Заславль; Телефон: ; Диагнозы: к
ариес
ID: 2; Имя: Кириллов Кирил Кирилович; Пол: Мужчина; Дата рождения: 26.12.1983; Город: Микашевичи; Телефон: +3752921202
01; Диагнозы: аутизм, шизофрения
ID: 1; Имя: Микулич Всеий Александрович; Пол: Мужчина; Дата рождения: 27.06.1984; Город: Минск; Телефон: +375292287777;
Диагнозы: микролит, сколиоз
-----
Найдено записей: 3
```

Рисунок 39 - Сортировка пациентов по городу

На рисунке 40 показан пример вывода пациентов, отсортированных по дате рождения в порядке возрастания.

```
Сортировка пациентов
1. Сортировка пациентов по имени
2. Сортировка пациентов по городу
3. Сортировка пациентов по дате рождения
0. Выйти в родительское меню
Выберите пункт меню: 3
-----
ID: 2; Имя: Кириллов Кирил Кирилович; Пол: Мужчина; Дата рождения: 26.12.1983; Город: Микашевичи; Телефон: +3752921202
01; Диагнозы: аутизм, шизофрения
ID: 1; Имя: Микулич Всеий Александрович; Пол: Мужчина; Дата рождения: 27.06.1984; Город: Минск; Телефон: +375292287777;
Диагнозы: микролит, сколиоз
ID: 4; Имя: Анохин Пётр Оксанович; Пол: Транссексуал; Дата рождения: 12.03.2001; Город: Заславль; Телефон: ; Диагнозы: к
ариес
-----
Найдено записей: 3
```

Рисунок 40 - Сортировка пациентов по дате рождения

### 4.3 Модуль пользователя

После входа пользователя программа запрашивает у пользователя путь к базе данных пациентов (рисунок 41). Таким образом, пользователь может работать с пациентами в любой базе данных. Если пользователь ничего не введёт (введёт пустой путь), то программа попытается открыть файл базы данных пациентов «по умолчанию», который должен находиться в текущей директории программы.

```
Имя пользователя: user
Пароль: *
Вход в систему выполнен успешно.
Введите путь к файлу базы данных пациентов [MedicalCenterPatients.db]:
```

Рисунок 41 - Запрос пути к базе данных пациентов в модуле пользователя

После успешного открытия базы данных пациентов программа запускает диалоговое взаимодействие с пользователем путём отображения пронумерованных пунктов меню и запросом у пользователя ввода желаемого номера (рисунок 42).

```
1. Просмотр всех пациентов
2. Показать иногородних пациентов
3. Показать пациентов старше X лет с диагнозом Y
4. Поиск пациентов
5. Сортировка пациентов
0. Выйти из программы
Выберите пункт меню:
```

Рисунок 42 - Главное меню модуля пользователя

Снимки экрана для всех операций в модуле пользователя совпадают со снимками экрана соответствующих операций в модуле администратора.

#### 4.4 Исключительные ситуации

Обработка исключительных ситуаций, как правило, занимает существенную часть кода программы. Это неизбежно, потому что программа без обработки ошибок и пограничных ситуаций выглядит для пользователя некачественной и может приводить к множеству негативных последствий: повреждению данных, неверной трактовке результатов работы программы, нагрузкой на службу поддержки или разработчика и т.п. В связи с этим, в программу встроено достаточно много проверок данных и результатов выполнения функций, обработок программных исключений, сопровождающихся уведомлением пользователя о проблеме и, при возможности, логичной реакцией на ситуацию либо прерыванием работы программы, если программа не смогла обработать исключительную ситуацию.

Первая, простейшая исключительная ситуация: введён неверный логин пользователя либо пароль. Программа сообщает об этом пользователю и спрашивает у пользователя, желает ли он повторить ввод данных (рисунок 43). Если пользователь ответит отрицательно, то программа завершит работу.

```
Имя пользователя: salkdjlsajdlsad
Пароль: *****
Имя пользователя или пароль не верны.
Повторить ввод? (y/n) [y]:
Имя пользователя: 21k3;21l.dsma.,masd
Пароль: *****
Имя пользователя или пароль не верны.
Повторить ввод? (y/n) [y]: n
```

Рисунок 43 - Имя пользователя или пароль не верны

При вводе пользователем несуществующего ID пользователя или пациента для таких операций, как редактирование или удаление, программ выведет соответствующее сообщение. На рисунке 44 показан пример

отображения ошибки при вводе несуществующего ID учётной записи пользователя.

```
Управление учетными записями пользователей
1. Просмотр всех учетных записей
2. Добавление новой учётной записи
3. Редактирование учётной записи
4. Удаление учётной записи
0. Выйти в родительское меню
Выберите пункт меню: 4
Введите ID пользователя: 10
Пользователя с таким ID не существует.
```

Рисунок 44 - Попытка удаления несуществующей записи

При попытке создания новой учётной записи с именем, которое уже существует, программа выведет соответствующее сообщение об ошибке (рисунок 45).

```
Введите данные для новой учётной записи.
Имя пользователя (login): vm
Пароль: ***
Повторите ввод пароля: ***
Пользователь с таким именем уже существует.
```

Рисунок 45 - Попытка создания учётной записи пользователя с уже существующим именем

Аналогичное сообщение будет выведено и при попытке изменения имени пользователя на такое, какое уже существует в базе данных (рисунок 46).

```
Управление учетными записями пользователей
1. Просмотр всех учетных записей
2. Добавление новой учётной записи
3. Редактирование учётной записи
4. Удаление учётной записи
0. Выйти в родительское меню
Выберите пункт меню: 3
Введите ID пользователя: 3
Пользователь найден:
ID: 3, Имя пользователя: vm2, Роль: Пользователь
Вы действительно хотите изменить этого пользователя? (y/n) [n]: y
Изменить роль пользователя? (y/n) [n]: n
Изменить имя пользователя? (y/n) [n]: y
Введите новое имя пользователя (login): vm
Пользователь с таким именем уже существует.
```

Рисунок 46 - Попытка изменения у пользователя имени на такое, какое уже существует в базе данных

При открытии или удалении файла с данными программа проверяет его существование. Если файла не существует, то программа выводит соответствующее сообщение (рисунок 47).



```

Управление файлами данных
1. Создать файл данных пациентов
2. Открыть файл данных пациентов
3. Заккрыть файл данных пациентов
4. Удалить файл данных пациентов
0. Выйти в родительское меню
Выберите пункт меню: 2
Введите путь к файлу базы данных пациентов: D:\12321.db
Не удалось найти файл по указаному пути.

```

Рисунок 47 - Попытка открытия несуществующего файла

При вводе данных о пациентах программа проверяет корректность ввода полей. На рисунке 48 показан пример вывода сообщений об ошибках ввода значений для ФИО пациента (оно не может быть пустым), пола, дате рождения (должна быть в формате ДД.ММ.ГГГГ) и городе (также не может быть пустым).

```

Редактирование данных о пациентах
1. Просмотр всех пациентов
2. Добавить пациента
3. Изменить пациента
4. Удалить пациента
0. Выйти в родительское меню
Выберите пункт меню: 2
ФИО пациента:
Вы ввели некорректное значение. Повторите ввод.
Поле не может быть пустым.
ФИО пациента: Петрович
Пол (м/ж/т): к
Вы ввели некорректное значение. Повторите ввод.
Пол (м/ж/т): м
Дата рождения (ДД.ММ.ГГГГ): фыв
Вы ввели некорректное значение. Повторите ввод.
Дата рождения (ДД.ММ.ГГГГ): 12.07.1927
Город:
Вы ввели некорректное значение. Повторите ввод.
Поле не может быть пустым.
Город: Минск

```

Рисунок 48 - Проверка корректности вводимых пользователем данных



# ПРИЛОЖЕНИЕ А

## (обязательное)

### Листинг кода с комментариями

#### Модуль MedicalCenterConsole.exe

##### Файл main.cpp

```
#include "stdafx.h"
#include "Console.h"
#include "Application.h"

int main()
{
    // set utf-8 locale for console and boost filesystem
    std::locale::global(boost::locale::generator().generate(""));
    boost::filesystem::path::imbue(std::locale());

    ::SetConsoleCP(1251);
    ::SetConsoleOutputCP(CP_UTF8);

    try
    {
        Application app;
        app.StartAndWaitExit();
    }
    catch (const std::exception& ex)
    {
        printf("\r\n\t Internal exception: %s", ex.what());
    }

    return 0;
}
```

##### Файл stdafx.h

```
#pragma once

#include "targetver.h"

#include <stdarg.h>
#include <iostream>
#include <io.h>
#include <fcntl.h>
#include <conio.h>

// windows
#include <windows.h>

// stl
#include <string>
#include <vector>
#include <memory>
#include <codecvt>
#include <locale>
#include <unordered_map>
#include <map>
#include <array>

// boost
#include <boost/filesystem.hpp>
#include <boost/locale.hpp>
#include <boost/tokenizer.hpp>
#include <boost/algorithm/string.hpp>
```

##### Файл Application.h

```
#pragma once
```

```

#include "Console.h"
#include "../PatientsRegistry/PatientsRegistry.h"
#include "../UsersRegistry/UsersRegistry.h"
#include "Language.h"

class Application
{
    static const std::string USERS_REGISTRY_DATABASE_FILENAME;
    static const std::string PATIENTS_REGISTRY_DATABASE_FILENAME_DEFAULT;

public:
    Application();

    void StartAndWaitExit();

private:
    void PrintWelcomText();
    bool ValidateUsersDatabase();
    bool AuthenticateUser(std::string& userName, UserRole& userRole);
    bool CreateDatabaseWithAdmin(bool databaseIsExist);

    std::shared_ptr<PatientsRegistry> OpenPatientsRegistryForAdmin();
    std::shared_ptr<PatientsRegistry> OpenPatientsRegistryForUser();

    const Language& GetLanguage() const;

private:
    Language m_language;
    Console m_console;
};

```

## Файл Application.cpp

```

#include "stdafx.h"
#include "Application.h"
#include "Utils.h"
#include "ConsoleInputFormAddUser.h"
#include "ConsoleInputFormLogin.h"
#include "SessionsFactory.h"

const std::string Application::USERS_REGISTRY_DATABASE_FILENAME = "MedicalCenterUsers.db";
const std::string Application::PATIENTS_REGISTRY_DATABASE_FILENAME_DEFAULT = "MedicalCenterPatients.db";

Application::Application() :
    m_console(m_language)
{
}

void Application::StartAndWaitExit()
{
    PrintWelcomText();

    try
    {
        if (!ValidateUsersDatabase())
        {
            return;
        }
    }
    catch (const std::exception& ex)
    {
        m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::CantOpenUsersDatabase));
        m_console.PrintErrorWithNewLine("\t %s", ex.what());

        return;
    }

    try
    {
        std::string userName;
        UserRole userRole;
        bool authenticationSuccess = AuthenticateUser(userName, userRole);

        if (!authenticationSuccess)

```

```

    {
        return;
    }

    m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::LoginSuccess));

    std::shared_ptr<PatientsRegistry> patientsRegistry;

    if (userRole == UserRole::Admin)
    {
        m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::LoggedInUserIsAdmin));

        patientsRegistry = OpenPatientsRegistryForAdmin();
    }
    else
    {
        patientsRegistry = OpenPatientsRegistryForUser();

        if (patientsRegistry == nullptr)
        {
            return;
        }
    }

    SessionsFactory sessionsFactory(m_language, m_console, USERS_REGISTRY_DATABASE_FILENAME);
    std::shared_ptr<Session> session = sessionsFactory.CreateSession(userName, userRole,
patientsRegistry);

    session->StartSessionAndWaitExit();
}
catch (...)
{
    m_console.PrintErrorWithNewLine(GetLanguage().GetString(
        LanguageString::UnknowError));

    throw;
}
}

bool Application::AuthenticateUser(std::string& userName, UserRole& userRole)
{
    bool authenticationSuccess = false;

    std::shared_ptr<UsersRegistry> usersRegistry =
        UsersRegistry::OpenUsersRegistry(USERS_REGISTRY_DATABASE_FILENAME, true);

    while (!authenticationSuccess)
    {
        ConsoleInputFormLogin formLogin(m_console, m_language);

        authenticationSuccess = usersRegistry->CheckUserPassword(formLogin.GetUsername(),
formLogin.GetPassword());

        if (authenticationSuccess)
        {
            UserRecord_t userRecord = usersRegistry->GetUserByLogin(formLogin.GetUsername());

            userName = userRecord.second.GetLogin();
            userRole = userRecord.second.GetRole();
        }
        else
        {
            m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::LoginFailed));

            if
(!m_console.RequestInputYesNo(GetLanguage().GetString(LanguageString::QuestionRepeatEnterYesNo), true))
            {
                break;
            }
        }
    }

    return authenticationSuccess;
}

std::shared_ptr<PatientsRegistry> Application::OpenPatientsRegistryForAdmin()
{

```

```

        std::shared_ptr<PatientsRegistry> patientsRegistry;

        if (!boost::filesystem::exists(PATIENTS_REGISTRY_DATABASE_FILENAME_DEFAULT))
        {
            patientsRegistry =
            PatientsRegistry::CreatePatientsRegistry(PATIENTS_REGISTRY_DATABASE_FILENAME_DEFAULT);

            if (patientsRegistry != nullptr)
            {
                m_console.PrintYellowWithNewLine(
                    GetLanguage().GetString(LanguageString::PatientsDatabaseDefaultHasCreatedAndOpenend),
                    PATIENTS_REGISTRY_DATABASE_FILENAME_DEFAULT.c_str());
            }
        }
        else
        {
            patientsRegistry =
            PatientsRegistry::OpenPatientsRegistry(PATIENTS_REGISTRY_DATABASE_FILENAME_DEFAULT, false);

            if (patientsRegistry != nullptr)
            {
                m_console.PrintYellowWithNewLine(
                    GetLanguage().GetString(LanguageString::PatientsDatabaseDefaultHasOpenend),
                    PATIENTS_REGISTRY_DATABASE_FILENAME_DEFAULT.c_str());
            }
        }

        return patientsRegistry;
    }

    std::shared_ptr<PatientsRegistry> Application::OpenPatientsRegistryForUser()
    {
        std::shared_ptr<PatientsRegistry> patientsRegistry;

        while (true)
        {
            std::string filePath;

            if (Utils::RequestExistFilePathWithDefaultValue(m_console, m_language,
                PATIENTS_REGISTRY_DATABASE_FILENAME_DEFAULT, filePath))
            {
                patientsRegistry =
                PatientsRegistry::OpenPatientsRegistry(PATIENTS_REGISTRY_DATABASE_FILENAME_DEFAULT, true);

                if (patientsRegistry != nullptr)
                {
                    break;
                }
                else
                {
                    m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::CantOpenPatientsDatabase));
                }
            }

            if
            (!m_console.RequestInputYesNo(GetLanguage().GetString(LanguageString::QuestionRepeatEnterYesNo), true))
            {
                break;
            }
        }

        return patientsRegistry;
    }

    bool Application::ValidateUsersDatabase()
    {
        bool databaseIsExist = false;

        if (boost::filesystem::exists(USERS_REGISTRY_DATABASE_FILENAME))
        {
            std::shared_ptr<UsersRegistry> usersRegistry =
            UsersRegistry::OpenUsersRegistry(USERS_REGISTRY_DATABASE_FILENAME, true);

            databaseIsExist = true;

            if (usersRegistry->GetAdminUsersCount() > 0)
            {

```

```

        return true;
    }
}

if (databaseIsExist)
{
    m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::AdminNotFoundInUsersDatabase)
);

    if
(!m_console.RequestInputYesNo(GetLanguage().GetString(LanguageString::QuestionCreateFirstAdmin), true))
    {
        return false;
    }
}
else
{
    m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::CreateNewUsersDatabaseInfo));

    if (!m_console.RequestInputYesNo(GetLanguage().GetString(LanguageString::QuestionContinueYesNo),
true))
    {
        return false;
    }

    m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::CreateAdminInNewUsersDatabase
Info));
}

return CreateDatabaseWithAdmin(databaseIsExist);
}

bool Application::CreateDatabaseWithAdmin(bool databaseIsExist)
{
    User adminUser;
    std::string adminUserPassword;

    while (true)
    {
        m_console.PrintWhiteWithNewLine(GetLanguage().GetString(LanguageString::NewUserFormCaption));

        ConsoleInputFormAddUser formAddUserAdmin(m_console, m_language);

        if (formAddUserAdmin.GetPassword() == formAddUserAdmin.GetPasswordConfirm())
        {
            adminUser = User(formAddUserAdmin.GetUsername(), UserRole::Admin);
            adminUserPassword = formAddUserAdmin.GetPassword();

            break;
        }
        else
        {
            m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::PasswordsNotEq));

            if
(!m_console.RequestInputYesNo(GetLanguage().GetString(LanguageString::QuestionRepeatEnterYesNo), true))
            {
                return false;
            }
        }
    }

    std::shared_ptr<UsersRegistry> usersRegistry;

    if (databaseIsExist)
    {
        usersRegistry = UsersRegistry::OpenUsersRegistry(USERS_REGISTRY_DATABASE_FILENAME, false);
    }
    else
    {
        usersRegistry = UsersRegistry::CreateUsersRegistry(USERS_REGISTRY_DATABASE_FILENAME);

        m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::UsersDatabaseCreated));
    }

    usersRegistry->AddUser(adminUser, adminUserPassword);
}

```

```

        m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::AdminCreatedInUsersDatabase))
;

        return true;
}

const Language& Application::GetLanguage() const
{
    return m_language;
}

void Application::PrintWelcomText()
{
    m_console.PrintWhiteWithNewLine(GetLanguage().GetString(LanguageString::Welcom));
}

```

## Файл UsersController.h

```

#pragma once

#include "../UsersRegistry/UsersRegistry.h"

class Console;
class Language;

class UsersController
{
public:
    explicit UsersController(const std::string& usersDatabaseFilePath, Console& console, const Language& language);
    ~UsersController();

    void PrintAllUsers() const;
    void AddNewUser();
    void RemoveUser();
    void EditUser();

private:
    const Language& GetLanguage() const;
    void PrintUsers(const std::vector<UserRecord_t>& usersRecords) const;
    void PrintUser(const UserRecord_t& userRecord) const;
    const std::string GetUserDisplayRole(const UserRole userRole) const;
    bool RequestUserId(uint64_t& userId);

private:
    const Language& m_language;
    Console& m_console;
    std::shared_ptr<UsersRegistry> m_usersRegistry;
};

```

## Файл UsersController.cpp

```

#include "stdafx.h"
#include "UsersController.h"
#include "Console.h"
#include "ConsoleInputFormAddUser.h"
#include "Language.h"

UsersController::UsersController(const std::string& usersDatabaseFilePath, Console& console, const Language& language)
:
    m_usersRegistry(UsersRegistry::OpenUsersRegistry(usersDatabaseFilePath, false)),
    m_language(language),
    m_console(console)
{
}

UsersController::~UsersController()
{
}

void UsersController::PrintAllUsers() const
{
}

```

```

        const std::vector<UserRecord_t> allUsers = m_usersRegistry->GetAllUsers();

        PrintUsers(allUsers);
    }

void UsersController::AddNewUser()
{
    User user;
    UserRole role(UserRole::User);
    std::string password;

    if (m_console.RequestInputYesNo(GetLanguage().GetString(LanguageString::CreateNewUserAsAdmin), false))
    {
        role = UserRole::Admin;
    }

    while (true)
    {
        m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::NewUserFormCaption));

        ConsoleInputFormAddUser formAddUser(m_console, m_language);

        if (formAddUser.GetPassword() == formAddUser.GetPasswordConfirm())
        {
            user = User(formAddUser.GetUsername(), role);
            password = formAddUser.GetPassword();

            break;
        }
        else
        {
            m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::PasswordsNotEq));

            if
(!m_console.RequestInputYesNo(GetLanguage().GetString(LanguageString::QuestionRepeatEnterYesNo), true))
            {
                return;
            }
        }
    }

    if (m_usersRegistry->IsExistUser(user.GetLogin()))
    {
        m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::UserLoginExistAlready));

        return;
    }

    if (m_usersRegistry->AddUser(user, password))
    {
        m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::UsersAdded));
    }
    else
    {
        m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::CantAddUser));
    }
}

void UsersController::RemoveUser()
{
    uint64_t userId;

    if (!RequestUserId(userId))
    {
        return;
    }

    if (!m_console.RequestInputYesNo(GetLanguage().GetString(LanguageString::RemoveUserConfirm), false))
    {
        return;
    }

    if (m_usersRegistry->DeleteUser(userId))
    {
        m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::UserRemoved));
    }
    else

```

```

    {
        m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::CantRemoveUser));
    }
}

bool UsersController::RequestUserId(uint64_t& userId)
{
    m_console.PrintWhite(GetLanguage().GetString(LanguageString::EnterUserId));
    uint64_t requestUserId = m_console.RequestInputInteger();

    const UserRecord_t existUserRecord = m_usersRegistry->GetUserById(requestUserId);

    if (existUserRecord.second == User::GetEmptyUser())
    {
        m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::UserIdNotFound));

        return false;
    }

    m_console.PrintInfoWithNewLine(GetLanguage().GetString(LanguageString::UserFound));
    PrintUser(existUserRecord);

    userId = requestUserId;

    return true;
}

void UsersController::EditUser()
{
    uint64_t userId;

    if (!RequestUserId(userId))
    {
        return;
    }

    if (!m_console.RequestInputYesNo(GetLanguage().GetString(LanguageString::EditUserConfirm), false))
    {
        return;
    }

    const UserRecord_t existUserRecord = m_usersRegistry->GetUserById(userId);
    User existUser = existUserRecord.second;

    UserRole newRole;
    std::string newName;
    bool changePassword = false;
    std::string newPassword;

    if (m_console.RequestInputYesNo(GetLanguage().GetString(LanguageString::EditUserChangeRole), false))
    {
        if (m_console.RequestInputYesNo(GetLanguage().GetString(LanguageString::EditUserAsAdmin), false))
        {
            newRole = UserRole::Admin;
        }
        else
        {
            newRole = UserRole::User;
        }
    }
    else
    {
        newRole = existUser.GetRole();
    }

    if (m_console.RequestInputYesNo(GetLanguage().GetString(LanguageString::EditUserChangeName), false))
    {
        m_console.PrintWhite(GetLanguage().GetString(LanguageString::EditUserNewName));
        newName = m_console.RequestInputNonEmptyString();

        if (m_usersRegistry->IsExistUser(newName) &&
            m_usersRegistry->GetUserByLogin(newName).first != userId)
        {
            m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::UserLoginExistAlready));

            return;
        }
    }
}

```



```

    }
    else
    {
        newName = existUser.GetLogin();
    }

    while (true)
    {
        if (!m_console.RequestInputYesNo(GetLanguage().GetString(LanguageString::EditUserChangePassword),
false))
        {
            break;
        }

        m_console.PrintWhite(GetLanguage().GetString(LanguageString::EditUserNewPassword));
        std::string tempNewPassword = m_console.RequestInputPassword();

        m_console.PrintWhite(GetLanguage().GetString(LanguageString::EditUserNewPasswordConfirm));
        std::string tempNewPasswordConfirm = m_console.RequestInputPassword();

        if (tempNewPassword == tempNewPasswordConfirm)
        {
            newPassword = tempNewPassword;
            changePassword = true;

            break;
        }
        else
        {
            m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::PasswordsNotEq));

            if
(!m_console.RequestInputYesNo(GetLanguage().GetString(LanguageString::QuestionRepeatEnterYesNo), true))
            {
                break;
            }
        }
    }

    User updateUser(newName, newRole);
    bool updateUserSuccess;

    if (changePassword)
    {
        updateUserSuccess = m_usersRegistry->UpdateUserAndPassword(userId, updateUser, newPassword);
    }
    else
    {
        updateUserSuccess = m_usersRegistry->UpdateUser(userId, updateUser);
    }

    if (updateUserSuccess)
    {
        m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::UserChanged));
    }
    else
    {
        m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::CantChangeUser));
    }
}

void UsersController::PrintUsers(const std::vector<UserRecord_t>& usersRecords) const
{
    if (usersRecords.empty())
    {
        m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::NotFoundAnyRecord));

        return;
    }

    m_console.PrintWhiteWithNewLine("-----");

    for (const auto& userRecord : usersRecords)
    {
        PrintUser(userRecord);
    }
}

```

```

        m_console.PrintWhiteWithNewLine("-----");
        m_console.PrintWhite(GetLanguage().GetString(LanguageString::FoundRecords));
        m_console.PrintWhite(u8"%zu\r\n", usersRecords.size());
    }

void UsersController::PrintUser(const UserRecord_t& userRecord) const
{
    const uint64_t userId = userRecord.first;
    const User& user = userRecord.second;

    m_console.PrintInfoWithNewLine("%s: %I64u, %s: %s, %s: %s",
        GetLanguage().GetString(LanguageString::UserID),
        userId,
        GetLanguage().GetString(LanguageString::UserName),
        user.GetLogin().c_str(),
        GetLanguage().GetString(LanguageString::UserRole),
        GetUserDisplayRole(user.GetRole()).c_str());
}

const std::string UsersController::GetUserDisplayRole(const UserRole userRole) const
{
    switch (userRole)
    {
        case UserRole::Admin:
            return GetLanguage().GetString(LanguageString::UserRoleIsAdmin);
            break;

        case UserRole::User:
            return GetLanguage().GetString(LanguageString::UserRoleIsUser);
            break;
    }

    throw std::logic_error("unknown user role");
}

const Language& UsersController::GetLanguage() const
{
    return m_language;
}

```

## Файл PatientsController.h

```

#pragma once

#include "../PatientsRegistry/PatientsRegistry.h"
#include "ConsoleInputForm.h"

class Console;
class Language;

class PatientsController
{
public:
    static const std::unordered_map<std::string, Gender> GENDER_STRINGS_MAP;

    explicit PatientsController(Console& console, const Language& language,
        std::shared_ptr<PatientsRegistry> patientsRegistry);
    ~PatientsController();

    PatientsController() = delete;
    PatientsController(const PatientsController&) = delete;
    PatientsController& operator=(const PatientsController&) = delete;

    void OpenFileForReadAndWrite();
    void OpenFileForReadOnly();
    void CreateFile();
    void CloseFile();
    void RemoveFile();

    void ViewAllPatients() const;
    void AddPatient();
    void EditPatient();
    void RemovePatient();

    void SearchPatientsByName() const;
    void SearchPatientsByCity() const;
}

```

```

void SearchPatientsByDiagnose() const;
void SearchPatientsByPhone() const;

void OrderPatientsByName() const;
void OrderPatientsByCity() const;
void OrderPatientsByBirthDate() const;

void ViewNonresidentPatients() const;
void ViewPatientsByAgeAndDiagnose() const;

private:
    const Language& GetLanguage() const;
    void OpenFile(bool readOnly);
    bool CheckFileIsOpened() const;
    void PrintPatients(const std::vector<PatientRecord_t>& patientsRecords) const;
    void PrintPatient(const PatientRecord_t& patientRecord) const;
    const Patient CreatePatientForm() const;
    bool RequestPatientId(uint64_t& patientId) const;

    const std::string GetGenderDisplayString(const Gender gender) const;
    const std::string GetBirthDateDisplayString(const BirthDate& birthDate) const;

    const Gender GetGenderByInputString(const std::string& genderString) const;
    const BirthDate GetBirthDateByInputString(const std::string& birthDateString) const;
    const std::vector<std::string> GetDiagnosesByInputString(const std::string& diagnosesString) const;

    bool CheckFormFieldGenderString(const ConsoleInputForm::FieldName& fieldName,
        const std::string& fieldValue, std::string& fieldValueFormatDescription) const;
    bool CheckFormFieldBirthDateString(const ConsoleInputForm::FieldName& fieldName,
        const std::string& fieldValue, std::string& fieldValueFormatDescription) const;

private:
    const Language& m_language;
    Console& m_console;
    std::shared_ptr<PatientsRegistry> m_patientsRegistry;
};

```

## Файл PatientsController.cpp

```

#include "stdafx.h"
#include "PatientsController.h"
#include "Console.h"
#include "ConsoleInputFormPatient.h"
#include "Utils.h"
#include "Language.h"

namespace ph = std::placeholders;

const std::unordered_map<std::string, Gender> PatientsController::GENDER_STRINGS_MAP = {
    { u8"M", Gender::Male },
    { u8"М", Gender::Male },
    { u8"m", Gender::Male },
    { u8"Ж", Gender::Feemale },
    { u8"ж", Gender::Feemale },
    { u8"F", Gender::Feemale },
    { u8"f", Gender::Feemale },
    { u8"Т", Gender::Schemale },
    { u8"т", Gender::Schemale },
    { u8"S", Gender::Schemale },
    { u8"s", Gender::Schemale },
};

PatientsController::PatientsController(Console& console, const Language& language,
    std::shared_ptr<PatientsRegistry> patientsRegistry)
:
    m_console(console),
    m_language(language),
    m_patientsRegistry(patientsRegistry)
{
}

PatientsController::~PatientsController()
{
}

```

```

void PatientsController::CreateFile()
{
    std::string filePath;

    if (!Utils::RequestFilePath(m_console, m_language, false, filePath))
    {
        return;
    }

    if (boost::filesystem::exists(filePath))
    {
        m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::PatientsFileExistAlready));

        return;
    }

    m_patientsRegistry = PatientsRegistry::CreatePatientsRegistry(filePath);

    m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::PatientsFileCreated));
}

void PatientsController::OpenFileForReadAndWrite()
{
    OpenFile(false);
}

void PatientsController::OpenFileForReadOnly()
{
    OpenFile(true);
}

void PatientsController::OpenFile(bool readOnly)
{
    std::string filePath;

    if (!Utils::RequestFilePath(m_console, m_language, true, filePath))
    {
        return;
    }

    m_patientsRegistry = PatientsRegistry::OpenPatientsRegistry(filePath, readOnly);

    m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::PatientsFileOpened));
}

void PatientsController::CloseFile()
{
    if (m_patientsRegistry != nullptr)
    {
        m_patientsRegistry.reset();

        m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::PatientsFileClosed));
    }
    else
    {
        m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::NotFoundOpenedPatientsFile));
    }
}

void PatientsController::RemoveFile()
{
    std::string filePath;

    if (!Utils::RequestFilePath(m_console, m_language, true, filePath))
    {
        return;
    }

    if (m_patientsRegistry != nullptr)
    {
        if (boost::filesystem::equivalent(m_patientsRegistry->GetConnectionString(), filePath))
        {
            m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::CantRemoveOpenedPatientsFile));
        }
        return;
    }
}

```

```

        if (!m_console.RequestInputYesNo(GetLanguage().GetString(LanguageString::RemoveFileConfirm), false))
        {
            return;
        }

        try
        {
            boost::filesystem::remove(filePath);

            m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::PatientsFileRemoved));
        }
        catch (const boost::filesystem::filesystem_error&)
        {
            m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::CantRemovePatientsFile));
        }
    }

bool PatientsController::CheckFileIsOpened() const
{
    if (m_patientsRegistry != nullptr)
    {
        return true;
    }
    else
    {
        m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::NotFoundOpenedPatientsFile));

        return false;
    }
}

void PatientsController::ViewAllPatients() const
{
    if (!CheckFileIsOpened())
    {
        return;
    }

    PrintPatients(m_patientsRegistry->GetAllPatients());
}

void PatientsController::AddPatient()
{
    if (!CheckFileIsOpened())
    {
        return;
    }

    m_patientsRegistry->AddPatient(CreatePatientForm());

    m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::PatientAdded));
}

void PatientsController::EditPatient()
{
    if (!CheckFileIsOpened())
    {
        return;
    }

    uint64_t patientId;

    if (!RequestPatientId(patientId))
    {
        return;
    }

    if (!m_console.RequestInputYesNo(GetLanguage().GetString(LanguageString::PatientEditConfirm), true))
    {
        return;
    }

    if (m_patientsRegistry->UpdatePatient(patientId, CreatePatientForm()))
    {
        m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::PatientChanged));
    }
}

```

```

        else
        {
            m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::CantChangePatient));
        }
    }

void PatientsController::RemovePatient()
{
    if (!CheckFileIsOpened())
    {
        return;
    }

    uint64_t patientId;

    if (!RequestPatientId(patientId))
    {
        return;
    }

    if (!m_console.RequestInputYesNo(GetLanguage().GetString(LanguageString::PatientRemoveConfirm),
false))
    {
        return;
    }

    if (m_patientsRegistry->DeletePatient(patientId))
    {
        m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::PatientRemoved));
    }
    else
    {
        m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::CantRemovePatient));
    }
}

void PatientsController::SearchPatientsByName() const
{
    if (!CheckFileIsOpened())
    {
        return;
    }

    m_console.PrintWhite(GetLanguage().GetString(LanguageString::SearchPatientName));
    const std::string searchName = m_console.RequestInputString();

    PrintPatients(m_patientsRegistry->SearchPatientsByName(searchName));
}

void PatientsController::SearchPatientsByCity() const
{
    if (!CheckFileIsOpened())
    {
        return;
    }

    m_console.PrintWhite(GetLanguage().GetString(LanguageString::SearchPatientCity));
    const std::string searchCity = m_console.RequestInputString();

    PrintPatients(m_patientsRegistry->SearchPatientsByCity(searchCity));
}

void PatientsController::SearchPatientsByDiagnose() const
{
    if (!CheckFileIsOpened())
    {
        return;
    }

    m_console.PrintWhite(GetLanguage().GetString(LanguageString::SearchPatientDiagnose));
    const std::string searchDiagnose = m_console.RequestInputString();

    PrintPatients(m_patientsRegistry->SearchPatientsByDiagnose(searchDiagnose));
}

void PatientsController::SearchPatientsByPhone() const
{

```

```

        if (!CheckFileIsOpened())
        {
            return;
        }

        m_console.PrintWhite(GetLanguage().GetString(LanguageString::SearchPatientPhone));
        const std::string searchPhone = m_console.RequestInputString();

        PrintPatients(m_patientsRegistry->SearchPatientsByPhone(searchPhone));
    }

void PatientsController::OrderPatientsByName() const
{
    if (!CheckFileIsOpened())
    {
        return;
    }

    PrintPatients(m_patientsRegistry->OrderPatientsByName());
}

void PatientsController::OrderPatientsByCity() const
{
    if (!CheckFileIsOpened())
    {
        return;
    }

    PrintPatients(m_patientsRegistry->OrderPatientsByCity());
}

void PatientsController::OrderPatientsByBirthDate() const
{
    if (!CheckFileIsOpened())
    {
        return;
    }

    PrintPatients(m_patientsRegistry->OrderPatientsByBirthDate());
}

void PatientsController::ViewNonresidentPatients() const
{
    if (!CheckFileIsOpened())
    {
        return;
    }

    m_console.PrintWhite(GetLanguage().GetString(LanguageString::SearchPatientCity));
    const std::string searchCity = m_console.RequestInputString();

    PrintPatients(m_patientsRegistry->ViewNonresidentPatients(searchCity));
}

void PatientsController::ViewPatientsByAgeAndDiagnose() const
{
    if (!CheckFileIsOpened())
    {
        return;
    }

    m_console.PrintWhite(GetLanguage().GetString(LanguageString::SearchPatientDiagnose));
    const std::string searchDiagnose = m_console.RequestInputString();

    m_console.PrintWhite(GetLanguage().GetString(LanguageString::SearchPatientAge));
    const uint16_t searchAge = static_cast<uint16_t>(m_console.RequestInputInteger());

    PrintPatients(m_patientsRegistry->ViewPatientsByAgeAndDiagnose(searchAge, searchDiagnose));
}

const Patient PatientsController::CreatePatientForm() const
{
    ConsoleInputFormPatient inputFormPatient(m_console, m_language,
        std::bind(&PatientsController::CheckFormFieldGenderString, this, ph::_1, ph::_2, ph::_3),
        std::bind(&PatientsController::CheckFormFieldBirthDateString, this, ph::_1, ph::_2, ph::_3));

    Patient newPatient(

```

```

        inputFormPatient.GetName(),
        GetGenderByInputString(inputFormPatient.GetGender()),
        GetBirthDateByInputString(inputFormPatient.GetBirthDate()),
        inputFormPatient.GetCity(),
        inputFormPatient.GetPhone(),
        GetDiagnosesByInputString(inputFormPatient.GetDiagnoses()));
    return newPatient;
}

bool PatientsController::RequestPatientId(uint64_t& patientId) const
{
    m_console.PrintWhite(GetLanguage().GetString(LanguageString::EnterPatientId));
    uint64_t requestPatientId = m_console.RequestInputInteger();

    const PatientRecord_t existPatientRecord = m_patientsRegistry->GetPatientById(requestPatientId);

    if (existPatientRecord.second == Patient::GetEmptyPatient())
    {
        m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::PatientIdNotFound));

        return false;
    }

    m_console.PrintInfoWithNewLine(GetLanguage().GetString(LanguageString::PatientFound));
    PrintPatient(existPatientRecord);

    patientId = requestPatientId;

    return true;
}

void PatientsController::PrintPatients(const std::vector<PatientRecord_t>& patientsRecords) const
{
    if (patientsRecords.empty())
    {
        m_console.PrintYellowWithNewLine(GetLanguage().GetString(LanguageString::NotFoundAnyRecord));

        return;
    }

    m_console.PrintWhiteWithNewLine("-----");

    for (const auto& patientRecord : patientsRecords)
    {
        PrintPatient(patientRecord);
    }

    m_console.PrintWhiteWithNewLine("-----");
    m_console.PrintWhite(GetLanguage().GetString(LanguageString::FoundRecords));
    m_console.PrintWhite(u8"%zu\r\n", patientsRecords.size());
}

void PatientsController::PrintPatient(const PatientRecord_t& patientRecord) const
{
    const uint64_t patientId = patientRecord.first;
    const Patient& patient = patientRecord.second;

    m_console.PrintInfoWithNewLine("%s: %I64u; %s: %s; %s: %s; %s: %s; %s: %s; %s: %s; %s: %s",
        GetLanguage().GetString(LanguageString::PatientId),
        patientId,
        GetLanguage().GetString(LanguageString::PatientName),
        patient.GetName().c_str(),
        GetLanguage().GetString(LanguageString::PatientGender),
        GetGenderDisplayString(patient.GetGender()).c_str(),
        GetLanguage().GetString(LanguageString::PatientBirthDate),
        GetBirthDateDisplayString(patient.GetBirthDate()).c_str(),
        GetLanguage().GetString(LanguageString::PatientCity),
        patient.GetCity().c_str(),
        GetLanguage().GetString(LanguageString::PatientPhone),
        patient.GetPhone().c_str(),
        GetLanguage().GetString(LanguageString::PatientDiagnoses),
        patient.GetDiagnosesAsString().c_str()
    );
}

const std::string PatientsController::GetGenderDisplayString(const Gender gender) const

```



```

{
    switch (gender)
    {
    case Gender::Male:
        return GetLanguage().GetString(LanguageString::GenderMale);
        break;

    case Gender::Feemale:
        return GetLanguage().GetString(LanguageString::GenderFeemale);
        break;

    case Gender::Schemale:
        return GetLanguage().GetString(LanguageString::GenderShemale);
        break;
    }

    throw std::logic_error("unknown gender");
}

const std::string PatientsController::GetBirthDateDisplayString(const BirthDate& birthDate) const
{
    char birthDateAsString[24] = { 0 };
    sprintf_s(birthDateAsString, _countof(birthDateAsString), "%02u.%02u.%04u",
        birthDate.GetDay(), birthDate.GetMonth(), birthDate.GetYear());

    return std::string(birthDateAsString);
}

const BirthDate PatientsController::GetBirthDateByInputString(const std::string& birthDateString) const
{
    // date format: 31.12.2016
    static const size_t DATE_STRING_LENGTH = 10;
    static const char DATE_SEPARATOR_CHAR = '.';

    if (birthDateString.size() != DATE_STRING_LENGTH ||
        birthDateString[2] != DATE_SEPARATOR_CHAR || birthDateString[5] != DATE_SEPARATOR_CHAR)
    {
        throw std::invalid_argument("Invalid birth date format");
    }

    return BirthDate(
        static_cast<int16_t>(std::stoul(birthDateString.substr(6, 4))),
        static_cast<int8_t>(std::stoul(birthDateString.substr(3, 2))),
        static_cast<int8_t>(std::stoul(birthDateString.substr(0, 2))));
}

bool PatientsController::CheckFormFieldBirthDateString(const ConsoleInputForm::FieldName& fieldName,
    const std::string& fieldValue, std::string& fieldValueFormatDescription) const
{
    try
    {
        GetBirthDateByInputString(fieldValue);
    }
    catch (const std::invalid_argument&)
    {
        return false;
    }

    return true;
}

const std::vector<std::string> PatientsController::GetDiagnosesByInputString(const std::string&
    diagnosesString) const
{
    std::vector<std::string> diagnoses;

    boost::char_separator<char> sep(",");
    boost::tokenizer<boost::char_separator<char>, std::string::const_iterator, std::string>
        tokens(diagnosesString, sep);

    for (const auto& t : tokens)
    {
        diagnoses.push_back(boost::trim_copy(t));
    }

    return diagnoses;
}

```

```

const Gender PatientsController::GetGenderByInputString(const std::string& genderString) const
{
    auto it = GENDER_STRINGS_MAP.find(genderString);

    if (it == GENDER_STRINGS_MAP.cend())
    {
        throw std::invalid_argument("Invalid gender string");
    }

    return it->second;
}

bool PatientsController::CheckFormFieldGenderString(const ConsoleInputForm::FieldName& fieldName,
const std::string& fieldValue, std::string& fieldValueFormatDescription) const
{
    if (GENDER_STRINGS_MAP.find(fieldValue) != GENDER_STRINGS_MAP.cend())
    {
        return true;
    }
    else
    {
        return false;
    }
}

const Language& PatientsController::GetLanguage() const
{
    return m_language;
}

```

## Файл Session.h

```

#pragma once

class Session
{
public:
    virtual ~Session()
    {
        ;
    }

    virtual void StartSessionAndWaitExit() = 0;
};

```

## Файл SessionAdmin.h

```

#pragma once

#include "Session.h"
#include "ConsoleMenu.h"
#include "UsersController.h"
#include "PatientsController.h"

class Console;
class Language;
class PatientsRegistry;

class SessionAdmin : public Session
{
public:
    explicit SessionAdmin(const std::string& userName, const std::string& usersDatabaseFilePath,
const Language& language, Console& console, std::shared_ptr<PatientsRegistry> patientsRegistry);
    ~SessionAdmin();

    void StartSessionAndWaitExit() override;

private:
    void CreateMenu();

private:
    const std::string m_userName;
    const Language& m_language;
}

```

```

        Console& m_console;
        UsersController m_usersController;
        PatientsController m_patientsController;
        ConsoleMenu m_menu;
};

```

## Файл SessionAdmin.cpp

```

#include "stdafx.h"
#include "SessionAdmin.h"
#include "../PatientsRegistry/PatientsRegistry.h"
#include "../UsersRegistry/UsersRegistry.h"
#include "Console.h"
#include "Language.h"

SessionAdmin::SessionAdmin(const std::string& userName, const std::string& usersDatabaseFilePath,
    const Language& language, Console& console, std::shared_ptr<PatientsRegistry> patientsRegistry)
    :
    m_userName(userName),
    m_usersController(usersDatabaseFilePath, console, language),
    m_patientsController(console, language, patientsRegistry),
    m_language(language),
    m_console(console),
    m_menu(console, language)
{
    CreateMenu();
}

SessionAdmin::~SessionAdmin()
{
}

void SessionAdmin::StartSessionAndWaitExit()
{
    m_menu.StartMenuAndWaitExit();
}

void SessionAdmin::CreateMenu()
{
    ConsoleMenu menuUsers(m_console, m_language);
    menuUsers.AddItem(m_language.GetString(LanguageString::MenuViewAllUsers),
        std::bind(&UsersController::PrintAllUsers, &m_usersController));
    menuUsers.AddItem(m_language.GetString(LanguageString::MenuAddNewUser),
        std::bind(&UsersController::AddNewUser, &m_usersController));
    menuUsers.AddItem(m_language.GetString(LanguageString::MenuEditUser),
        std::bind(&UsersController::EditUser, &m_usersController));
    menuUsers.AddItem(m_language.GetString(LanguageString::MenuRemoveUser),
        std::bind(&UsersController::RemoveUser, &m_usersController));

    ConsoleMenu menuFile(m_console, m_language);
    menuFile.AddItem(m_language.GetString(LanguageString::MenuPatientsFileCreateFile),
        std::bind(&PatientsController::CreateFile, &m_patientsController));
    menuFile.AddItem(m_language.GetString(LanguageString::MenuPatientsFileOpen),
        std::bind(&PatientsController::OpenFileForReadAndWrite, &m_patientsController));
    menuFile.AddItem(m_language.GetString(LanguageString::MenuPatientsCloseFile),
        std::bind(&PatientsController::CloseFile, &m_patientsController));
    menuFile.AddItem(m_language.GetString(LanguageString::MenuPatientsRemoveFile),
        std::bind(&PatientsController::RemoveFile, &m_patientsController));

    ConsoleMenu menuDataEdit(m_console, m_language);
    menuDataEdit.AddItem(m_language.GetString(LanguageString::MenuViewAllPatients),
        std::bind(&PatientsController::ViewAllPatients, &m_patientsController));
    menuDataEdit.AddItem(m_language.GetString(LanguageString::MenuAddPatient),
        std::bind(&PatientsController::AddPatient, &m_patientsController));
    menuDataEdit.AddItem(m_language.GetString(LanguageString::MenuEditPatient),
        std::bind(&PatientsController::EditPatient, &m_patientsController));
    menuDataEdit.AddItem(m_language.GetString(LanguageString::MenuRemovePatient),
        std::bind(&PatientsController::RemovePatient, &m_patientsController));

    ConsoleMenu menuDataSearch(m_console, m_language);
    menuDataSearch.AddItem(m_language.GetString(LanguageString::MenuSearchPatientsByName),
        std::bind(&PatientsController::SearchPatientsByName, &m_patientsController));
    menuDataSearch.AddItem(m_language.GetString(LanguageString::MenuSearchPatientsByCity),
        std::bind(&PatientsController::SearchPatientsByCity, &m_patientsController));
    menuDataSearch.AddItem(m_language.GetString(LanguageString::MenuSearchPatientsByDiagnose),
        std::bind(&PatientsController::SearchPatientsByDiagnose, &m_patientsController));
}

```

```

menuDataSearch.AddItem(m_language.GetString(LanguageString::MenuSearchPatientsByPhone),
    std::bind(&PatientsController::SearchPatientsByPhone, &m_patientsController));

ConsoleMenu menuDataOrder(m_console, m_language);
menuDataOrder.AddItem(m_language.GetString(LanguageString::MenuOrderPatientsByName),
    std::bind(&PatientsController::OrderPatientsByName, &m_patientsController));
menuDataOrder.AddItem(m_language.GetString(LanguageString::MenuOrderPatientsByCity),
    std::bind(&PatientsController::OrderPatientsByCity, &m_patientsController));
menuDataOrder.AddItem(m_language.GetString(LanguageString::MenuOrderPatientsByBirthDate),
    std::bind(&PatientsController::OrderPatientsByBirthDate, &m_patientsController));

ConsoleMenu menuDataView(m_console, m_language);
menuDataView.AddItem(m_language.GetString(LanguageString::MenuViewNonresidentPatients),
    std::bind(&PatientsController::ViewNonresidentPatients, &m_patientsController));
menuDataView.AddItem(m_language.GetString(LanguageString::MenuViewPatientsByAgeAndDiagnose),
    std::bind(&PatientsController::ViewPatientsByAgeAndDiagnose, &m_patientsController));
menuDataView.AddMenu(m_language.GetString(LanguageString::MenuSearchPatients),
    menuDataSearch);
menuDataView.AddMenu(m_language.GetString(LanguageString::MenuOrderPatients),
    menuDataOrder);

ConsoleMenu menuData(m_console, m_language);
menuData.AddMenu(m_language.GetString(LanguageString::MenuEditPatientsData),
    menuDataEdit);
menuData.AddMenu(m_language.GetString(LanguageString::MenuProcessPatientsData),
    menuDataView);

m_menu.AddMenu(m_language.GetString(LanguageString::MenuControlUsers),
    menuUsers);
m_menu.AddMenu(m_language.GetString(LanguageString::MenuControlPatientsDatabasesFiles),
    menuFile);
m_menu.AddMenu(m_language.GetString(LanguageString::MenuControlData),
    menuData);
}

```

## Файл SessionUser.h

```

#pragma once

#include "Session.h"
#include "ConsoleMenu.h"
#include "PatientsController.h"

class Console;
class Language;

class SessionUser : public Session
{
public:
    explicit SessionUser(const std::string& userName, const Language& language, Console& console,
        std::shared_ptr<PatientsRegistry> patientsRegistry);
    ~SessionUser();

    void StartSessionAndWaitExit() override;

private:
    void CreateMenu();

private:
    const std::string m_userName;
    const Language& m_language;
    Console& m_console;
    PatientsController m_patientsController;
    ConsoleMenu m_menu;
};

```

## Файл SessionUser.cpp

```

#include "stdafx.h"
#include "SessionUser.h"
#include "../PatientsRegistry/PatientsRegistry.h"
#include "Console.h"
#include "Language.h"

```

```

SessionUser::SessionUser(const std::string& userName, const Language& language, Console& console,
    std::shared_ptr<PatientsRegistry> patientsRegistry)
:
    m_userName(userName),
    m_language(language),
    m_console(console),
    m_patientsController(console, language, patientsRegistry),
    m_menu(console, language)
{
    CreateMenu();
}

SessionUser::~SessionUser()
{
}

void SessionUser::StartSessionAndWaitExit()
{
    m_menu.StartMenuAndWaitExit();
}

void SessionUser::CreateMenu()
{
    ConsoleMenu menuDataSearch(m_console, m_language);
    menuDataSearch.AddItem(m_language.GetString(LanguageString::MenuSearchPatientsByName),
        std::bind(&PatientsController::SearchPatientsByName, &m_patientsController));
    menuDataSearch.AddItem(m_language.GetString(LanguageString::MenuSearchPatientsByCity),
        std::bind(&PatientsController::SearchPatientsByCity, &m_patientsController));
    menuDataSearch.AddItem(m_language.GetString(LanguageString::MenuSearchPatientsByDiagnose),
        std::bind(&PatientsController::SearchPatientsByDiagnose, &m_patientsController));
    menuDataSearch.AddItem(m_language.GetString(LanguageString::MenuSearchPatientsByPhone),
        std::bind(&PatientsController::SearchPatientsByPhone, &m_patientsController));

    ConsoleMenu menuDataOrder(m_console, m_language);
    menuDataOrder.AddItem(m_language.GetString(LanguageString::MenuOrderPatientsByName),
        std::bind(&PatientsController::OrderPatientsByName, &m_patientsController));
    menuDataOrder.AddItem(m_language.GetString(LanguageString::MenuOrderPatientsByCity),
        std::bind(&PatientsController::OrderPatientsByCity, &m_patientsController));
    menuDataOrder.AddItem(m_language.GetString(LanguageString::MenuOrderPatientsByBirthDate),
        std::bind(&PatientsController::OrderPatientsByBirthDate, &m_patientsController));

    m_menu.AddItem(m_language.GetString(LanguageString::MenuViewAllPatients),
        std::bind(&PatientsController::ViewAllPatients, &m_patientsController));
    m_menu.AddItem(m_language.GetString(LanguageString::MenuViewNonresidentPatients),
        std::bind(&PatientsController::ViewNonresidentPatients, &m_patientsController));
    m_menu.AddItem(m_language.GetString(LanguageString::MenuViewPatientsByAgeAndDiagnose),
        std::bind(&PatientsController::ViewPatientsByAgeAndDiagnose, &m_patientsController));

    m_menu.AddMenu(m_language.GetString(LanguageString::MenuSearchPatients),
        menuDataSearch);
    m_menu.AddMenu(m_language.GetString(LanguageString::MenuOrderPatients),
        menuDataOrder);
}

```

## Файл SessionsFactory.h

```

#pragma once

#include "Session.h"
#include "../UsersRegistry/UserRole.h"

class Language;
class Console;
class PatientsRegistry;

class SessionsFactory
{
public:
    explicit SessionsFactory(const Language& language, Console& console, const std::string&
        usersDatabaseFilePath);
    ~SessionsFactory();

    std::shared_ptr<Session> CreateSession(const std::string& userName, const UserRole userRole,
        std::shared_ptr<PatientsRegistry> patientsRegistry);
}

```

```
private:
    const Language& m_language;
    Console& m_console;
    const std::string m_usersDatabaseFilePath;
};
```

## Файл SessionsFactory.cpp

```
#include "stdafx.h"
#include "SessionsFactory.h"
#include "SessionAdmin.h"
#include "SessionUser.h"
#include "Language.h"

SessionsFactory::SessionsFactory(const Language& language, Console& console, const std::string&
usersDatabaseFilePath) :
    m_language(language),
    m_console(console),
    m_usersDatabaseFilePath(usersDatabaseFilePath)
{
}

SessionsFactory::~SessionsFactory()
{
}

std::shared_ptr<Session> SessionsFactory::CreateSession(const std::string& userName, const UserRole
userRole,
    std::shared_ptr<PatientsRegistry> patientsRegistry)
{
    switch (userRole)
    {
        case UserRole::Admin:
            return std::make_shared<SessionAdmin>(userName, m_usersDatabaseFilePath, m_language, m_console,
patientsRegistry);
            break;

        case UserRole::User:
            return std::make_shared<SessionUser>(userName, m_language, m_console, patientsRegistry);
            break;

        default:
            throw std::logic_error("unknown user role id");
    }
}
```

## Файл Console.h

```
#pragma once

class Language;

class Console
{
    enum ConsoleColors
    {
        CC_BLACK = 0x0000,
        CC_RED = 0x0004,
        CC_GREEN = 0x0002,
        CC_BROWN = 0x0006,
        CC_BLUE = 0x0001,
        CC_MAGENTA = 0x0005,
        CC_CYAN = 0x0003,
        CC_GRAY = 0x0007,
        CC_DARKGRAY = 0x0008,
        CC_LIGHTRED = 0x000C,
        CC_LIGHTGREEN = 0x000A,
        CC_YELLOW = 0x000E,
        CC_LIGHTBLUE = 0x0009,
        CC_LIGHTMAGENTA = 0x000D,
        CC_LIGHTCYAN = 0x000B,
        CC_WHITE = 0x000F
    };
};
```

```

        static const size_t FORMAT_TEXT_BUFFER_SIZE;

public:
    explicit Console(const Language& language);

    void PrintInfo(const char* msg, ...);
    void PrintWhite(const char* msg, ...);
    void PrintYellow(const char* msg, ...);
    void PrintError(const char* msg, ...);

    void PrintInfoWithNewLine(const char* msg, ...);
    void PrintWhiteWithNewLine(const char* msg, ...);
    void PrintYellowWithNewLine(const char* msg, ...);
    void PrintErrorWithNewLine(const char* msg, ...);

    bool RequestInputYesNo(const char* requestText, bool defaultValue);
    const std::string RequestInputString();
    const std::string RequestInputNonEmptyString();
    const uint64_t RequestInputInteger();
    const std::string RequestInputPassword();

private:
    const Language& GetLanguage() const;

    void PrintTextWithColor(WORD color, bool newline, const char* msg, va_list args);

    void SetConsoleColor(WORD color);
    void ResetConsoleColor();

private:
    const Language& m_language;
    HANDLE m_consoleHandle;
    WORD m_defaultAttributes;
    std::wstring_convert<std::codecvt_utf8<wchar_t>> m_utf8Conveter;
};

```

## Файл Console.cpp

```

#include "stdafx.h"
#include "Console.h"
#include "Utils.h"
#include "Language.h"

const size_t Console::FORMAT_TEXT_BUFFER_SIZE = 4096;

Console::Console(const Language& language) :
    m_language(language)
{
    m_consoleHandle = ::GetStdHandle(STD_OUTPUT_HANDLE);

    CONSOLE_SCREEN_BUFFER_INFO csbi;

    if (::GetConsoleScreenBufferInfo(m_consoleHandle, &csbi))
    {
        m_defaultAttributes = csbi.wAttributes;
    }
    else
    {
        m_defaultAttributes = CC_WHITE;
    }
}

void Console::SetConsoleColor(WORD color)
{
    WORD attr = m_defaultAttributes;
    attr &= 0xFFF0;
    attr |= color;
    ::SetConsoleTextAttribute(m_consoleHandle, attr);
}

void Console::ResetConsoleColor()
{
    ::SetConsoleTextAttribute(m_consoleHandle, m_defaultAttributes);
}

```

```

void Console::PrintTextWithColor(WORD color, bool newLine, const char* msg, va_list args)
{
    char formattedLine[FORMAT_TEXT_BUFFER_SIZE] = { 0 };
    vsnprintf_s((char*)formattedLine, _countof(formattedLine), _TRUNCATE, msg, args);

    SetConsoleColor(color);

    printf("%s", formattedLine);

    if (newLine)
    {
        printf("\r\n");
    }

    ResetConsoleColor();
}

void Console::PrintInfo(const char* msg, ...)
{
    va_list args;
    va_start(args, msg);

    PrintTextWithColor(CC_GRAY, false, msg, args);

    va_end(args);
}

void Console::PrintWhite(const char* msg, ...)
{
    va_list args;
    va_start(args, msg);

    PrintTextWithColor(CC_WHITE, false, msg, args);

    va_end(args);
}

void Console::PrintYellow(const char* msg, ...)
{
    va_list args;
    va_start(args, msg);

    PrintTextWithColor(CC_YELLOW, false, msg, args);

    va_end(args);
}

void Console::PrintError(const char* msg, ...)
{
    va_list args;
    va_start(args, msg);

    PrintTextWithColor(CC_LIGHTRED, false, msg, args);

    va_end(args);
}

void Console::PrintInfoWithNewLine(const char* msg, ...)
{
    va_list args;
    va_start(args, msg);

    PrintTextWithColor(CC_GRAY, true, msg, args);

    va_end(args);
}

void Console::PrintWhiteWithNewLine(const char* msg, ...)
{
    va_list args;
    va_start(args, msg);

    PrintTextWithColor(CC_WHITE, true, msg, args);

    va_end(args);
}

```



```

void Console::PrintYellowWithNewLine(const char* msg, ...)
{
    va_list args;
    va_start(args, msg);

    PrintTextWithColor(CC_YELLOW, true, msg, args);

    va_end(args);
}

void Console::PrintErrorWithNewLine(const char* msg, ...)
{
    va_list args;
    va_start(args, msg);

    PrintTextWithColor(CC_LIGHTRED, true, msg, args);

    va_end(args);
}

bool Console::RequestInputYesNo(const char* requestText, bool defaultValue)
{
    bool reponseValue = defaultValue;

    PrintYellow(requestText);
    PrintYellow(" (y/n) [");
    PrintYellow(defaultValue ? "y" : "n");
    PrintYellow("]: ");

    SetConsoleColor(CC_YELLOW);

    while (true)
    {
        std::string inputString;
        getline(std::cin, inputString);

        if (inputString.empty())
        {
            break;
        }

        char responseSymbol = inputString.front();

        // responseSymbol is in local codepage
        if (responseSymbol == 'Y' || responseSymbol == 'y' ||
            responseSymbol == 'Д' || responseSymbol == 'д')
        {
            reponseValue = true;

            break;
        }
        else if (responseSymbol == 'N' || responseSymbol == 'n' ||
            responseSymbol == 'H' || responseSymbol == 'h')
        {
            reponseValue = false;

            break;
        }
    }

    ResetConsoleColor();

    return reponseValue;
}

const std::string Console::RequestInputString()
{
    SetConsoleColor(CC_GRAY);

    std::string inputString;
    getline(std::cin, inputString);

    ResetConsoleColor();

    return Utils::ConvertStringFromLocalCodepageToUtf8(inputString);
}

```

```

const std::string Console::RequestInputNonEmptyString()
{
    while (true)
    {
        const std::string inputValue = RequestInputString();

        if (inputValue.empty())
        {
            continue;
        }

        return inputValue;
    }
}

const std::string Console::RequestInputPassword()
{
    static const char RETURN_SYMBOL_CODE = 13;

    SetConsoleColor(CC_GRAY);

    std::string inputString;

    int ch = _getch();

    while (ch != RETURN_SYMBOL_CODE)
    {
        inputString.push_back(ch);
        std::cout << '*';

        ch = _getch();
    }

    std::cout << "\r\n";

    ResetConsoleColor();

    return Utils::ConvertStringFromLocalCodepageToUtf8(inputString);
}

const uint64_t Console::RequestInputInteger()
{
    while (true)
    {
        const std::string inputValue = RequestInputNonEmptyString();

        uint64_t inputValueInteger;

        try
        {
            inputValueInteger = std::stoull(inputValue);
        }
        catch (const std::invalid_argument&)
        {
            PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::IncorrectValue));

            continue;
        }
        catch (const std::out_of_range&)
        {
            PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::IncorrectValue));

            continue;
        }

        return inputValueInteger;
    }
}

const Language& Console::GetLanguage() const
{
    return m_language;
}

```

## Файл ConsoleMenu.h

```
#pragma once

class Console;
class Language;

class ConsoleMenu
{
    struct MenuItem
    {
        std::string name;
        std::function<void()> action;
    };

public:
    explicit ConsoleMenu(Console& console, const Language& language);
    ~ConsoleMenu();

    void AddItem(const std::string& itemName, std::function<void()> action);
    void AddMenu(const std::string& itemName, const ConsoleMenu menu);

    void StartMenuAndWaitExit() const;

private:
    const Language& GetLanguage() const;

    void StartMenuDialog(bool childMenu, const std::string& itemName) const;
    bool DoMenuDialog(bool childMenu, const std::string& itemName) const;
    void PrintMenu(bool childMenu, const std::string& itemName) const;
    void CallAction(const std::function<void()>& actionCallback) const;

private:
    const Language& m_language;
    Console& m_console;
    std::map<size_t, MenuItem> m_items;
};
```

## Файл ConsoleMenu.cpp

```
#include "stdafx.h"
#include "ConsoleMenu.h"
#include "Console.h"
#include "Language.h"

ConsoleMenu::ConsoleMenu(Console& console, const Language& language) :
    m_language(language),
    m_console(console)
{
}

ConsoleMenu::~~ConsoleMenu()
{
}

void ConsoleMenu::AddItem(const std::string& itemName, std::function<void()> action)
{
    size_t menuItemNumber = m_items.size() + 1;
    m_items[menuItemNumber] = MenuItem{ itemName, action };
}

void ConsoleMenu::AddMenu(const std::string& itemName, const ConsoleMenu menu)
{
    std::function<void()> actionFunc = [itemName, menu]()
    {
        menu.StartMenuDialog(true, itemName);
    };

    m_items[m_items.size() + 1] = MenuItem{ itemName, actionFunc };
}

void ConsoleMenu::StartMenuAndWaitExit() const
{
    StartMenuDialog(false, "");
}
```

```

void ConsoleMenu::StartMenuDialog(bool childMenu, const std::string& itemName) const
{
    while (true)
    {
        if (!DoMenuDialog(childMenu, itemName))
        {
            break;
        }
    }
}

bool ConsoleMenu::DoMenuDialog(bool childMenu, const std::string& itemName) const
{
    PrintMenu(childMenu, itemName);

    m_console.PrintWhite(GetLanguage().GetString(LanguageString::MenuSelectItem));

    uint64_t selectedMenuItemNum = m_console.RequestInputInteger();

    if (selectedMenuItemNum == 0)
    {
        return false;
    }

    auto it = m_items.find(selectedMenuItemNum);

    if (it == m_items.cend())
    {
        m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::IncorrectValue));

        return true;
    }

    CallAction(it->second.action);

    return true;
}

void ConsoleMenu::PrintMenu(bool childMenu, const std::string& itemName) const
{
    if (childMenu)
    {
        m_console.PrintWhite(u8"%s\r\n", itemName.c_str());
    }

    for (const auto& i : m_items)
    {
        m_console.PrintInfo(u8"%zu. %s\r\n", i.first, i.second.name.c_str());
    }

    if (childMenu)
    {
        m_console.PrintInfoWithNewLine(GetLanguage().GetString(LanguageString::MenuReturnToParent));
    }
    else
    {
        m_console.PrintInfoWithNewLine(GetLanguage().GetString(LanguageString::MenuExit));
    }
}

void ConsoleMenu::CallAction(const std::function<void()>& actionCallback) const
{
    if (actionCallback == nullptr)
    {
        return;
    }

    try
    {
        actionCallback();
    }
    catch (const std::exception& ex)
    {
        m_console.PrintErrorWithNewLine(GetLanguage().GetString(LanguageString::UnknowError));
        m_console.PrintErrorWithNewLine("\t %s", ex.what());
    }
}

```

```

}

const Language& ConsoleMenu::GetLanguage() const
{
    return m_language;
}

```

## Файл ConsoleInputForm.h

```

#pragma once

class Console;
class Language;

class ConsoleInputForm
{
public:
    using FieldName = std::string;
    using FieldDisplayText = std::string;
    using FieldCheckValuePredicat = std::function<bool(const FieldName& fieldName,
        const std::string& fieldValue, std::string& fieldValueFormatDescription)>;

    struct InputFieldItem
    {
        FieldName name;
        bool isPassword;
        FieldDisplayText display;
        FieldCheckValuePredicat predicat;
    };

    using InputFields = std::vector<InputFieldItem>;
    using InputFieldsValues = std::map<FieldName, std::string>;

public:
    explicit ConsoleInputForm(Console& console, const Language& language, const InputFields& inputFields);

    void DoInputFormFields();
    const InputFieldsValues& GetInputFieldsValues() const;

private:
    void DoInputFormField(const InputFieldItem& fieldItem);
    bool CheckFieldValue(const FieldCheckValuePredicat& predicat, const std::string& fieldName,
        const std::string& fieldValue);

private:
    const Language& m_language;
    Console& m_console;
    const InputFields m_inputFields;
    InputFieldsValues m_fieldsValues;
};

```

## Файл ConsoleInputForm.cpp

```

#include "stdafx.h"
#include "ConsoleInputForm.h"
#include "Console.h"
#include "Language.h"

ConsoleInputForm::ConsoleInputForm(Console& console, const Language& language, const InputFields&
inputFields) :
    m_console(console),
    m_language(language),
    m_inputFields(inputFields)
{
}

const ConsoleInputForm::InputFieldsValues& ConsoleInputForm::GetInputFieldsValues() const
{
    return m_fieldsValues;
}

void ConsoleInputForm::DoInputFormFields()
{
}

```

```

        for (const auto& fieldItem : m_inputFields)
        {
            DoInputFormField(fieldItem);
        }
    }

void ConsoleInputForm::DoInputFormField(const InputFieldItem& fieldItem)
{
    while (true)
    {
        m_console.PrintWhite(fieldItem.display.c_str());
        m_console.PrintWhite(": ");

        std::string fieldValue;

        if (fieldItem.isPassword)
        {
            fieldValue = m_console.RequestInputPassword();
        }
        else
        {
            fieldValue = m_console.RequestInputString();
        }

        if (!CheckFieldValue(fieldItem.predicat, fieldItem.name, fieldValue))
        {
            continue;
        }

        m_fieldsValues.emplace(fieldItem.name, fieldValue);

        break;
    }
}

bool ConsoleInputForm::CheckFieldValue(const FieldCheckValuePredicat& predicat,
const std::string& fieldName, const std::string& fieldValue)
{
    if (predicat == nullptr)
    {
        return true;
    }

    std::string fieldValueFormatDescription;

    if (predicat(fieldName, fieldValue, fieldValueFormatDescription))
    {
        return true;
    }

    m_console.PrintErrorWithNewLine(m_language.GetString(LanguageString::IncorrectValue));

    if (!fieldValueFormatDescription.empty())
    {
        m_console.PrintInfo(u8"\t%s\r\n", fieldValueFormatDescription.c_str());
    }

    return false;
}

```

## Файл ConsoleInputFormLogin.h

```

#pragma once

#include "ConsoleInputForm.h"

class Language;

class ConsoleInputFormLogin
{
public:
    static const std::string FIELD_USERNAME;
    static const std::string FIELD_PASSWORD;

    explicit ConsoleInputFormLogin(Console& console, const Language& language);

```

```

        const std::string& GetUsername() const;
        const std::string& GetPassword() const;

private:
    ConsoleInputForm m_consoleInputForm;
};

```

## Файл ConsoleInputFormLogin.cpp

```

#include "stdafx.h"
#include "ConsoleInputFormLogin.h"
#include "Language.h"

const std::string ConsoleInputFormLogin::FIELD_USERNAME = "username";
const std::string ConsoleInputFormLogin::FIELD_PASSWORD = "password";

ConsoleInputFormLogin::ConsoleInputFormLogin(Console& console, const Language& language)
:
    m_consoleInputForm(console, language, ConsoleInputForm::InputFields
    {
        ConsoleInputForm::InputFieldItem
        {
            FIELD_USERNAME,
            false,
            language.GetString(LanguageString::LoginName),
            nullptr
        },
        ConsoleInputForm::InputFieldItem
        {
            FIELD_PASSWORD,
            true,
            language.GetString(LanguageString::LoginPasword),
            nullptr
        }
    })
{
    m_consoleInputForm.DoInputFormFields();
}

const std::string& ConsoleInputFormLogin::GetUsername() const
{
    return m_consoleInputForm.GetInputFieldsValues().at(FIELD_USERNAME);
}

const std::string& ConsoleInputFormLogin::GetPassword() const
{
    return m_consoleInputForm.GetInputFieldsValues().at(FIELD_PASSWORD);
}

```

## Файл ConsoleInputFormAddUser.h

```

#pragma once

#include "ConsoleInputForm.h"

class Language;

class ConsoleInputFormAddUser
{
public:
    static const std::string FIELD_USERNAME;
    static const std::string FIELD_PASSWORD;
    static const std::string FIELD_CONFIRM_PASSWORD;

public:
    explicit ConsoleInputFormAddUser(Console& console, const Language& language);

    const std::string& GetUsername() const;
    const std::string& GetPassword() const;
    const std::string& GetPasswordConfirm() const;

private:
    bool CheckFieldValuePredicate(const ConsoleInputForm::FieldName& fieldName,
        const std::string& fieldValue, std::string& fieldValueFormatDescription);

private:

```

```

    const Language& m_language;
    ConsoleInputForm m_consoleInputForm;
};

```

## Файл ConsoleInputFormAddUser.cpp

```

#include "stdafx.h"
#include "ConsoleInputFormAddUser.h"
#include "Language.h"

const std::string ConsoleInputFormAddUser::FIELD_USERNAME = "username";
const std::string ConsoleInputFormAddUser::FIELD_PASSWORD = "password";
const std::string ConsoleInputFormAddUser::FIELD_CONFIRM_PASSWORD = "password_confirm";

namespace ph = std::placeholders;

ConsoleInputFormAddUser::ConsoleInputFormAddUser(Console& console, const Language& language)
:
    m_language(language),
    m_consoleInputForm(console, language, ConsoleInputForm::InputFields
    {
        ConsoleInputForm::InputFieldItem
        {
            FIELD_USERNAME,
            false,
            language.GetString(LanguageString::UserFormName),
            std::bind(&ConsoleInputFormAddUser::CheckFieldValuePredicate, this, ph::_1, ph::_2, ph::_3)
        },
        ConsoleInputForm::InputFieldItem
        {
            FIELD_PASSWORD,
            true,
            language.GetString(LanguageString::UserFormPassword),
            nullptr
        },
        ConsoleInputForm::InputFieldItem
        {
            FIELD_CONFIRM_PASSWORD,
            true,
            language.GetString(LanguageString::UserFormConfirmPassword),
            nullptr
        },
    })
{
    m_consoleInputForm.DoInputFormFields();
}

bool ConsoleInputFormAddUser::CheckFieldValuePredicate(const ConsoleInputForm::FieldName& fieldName,
    const std::string& fieldValue, std::string& fieldValueFormatDescription)
{
    if (fieldName == FIELD_USERNAME)
    {
        if (fieldValue.empty())
        {
            fieldValueFormatDescription = m_language.GetString(LanguageString::EmptyValueNotAllowed);

            return false;
        }
    }

    return true;
}

const std::string& ConsoleInputFormAddUser::GetUsername() const
{
    return m_consoleInputForm.GetInputFieldsValues().at(FIELD_USERNAME);
}

const std::string& ConsoleInputFormAddUser::GetPassword() const
{
    return m_consoleInputForm.GetInputFieldsValues().at(FIELD_PASSWORD);
}

const std::string& ConsoleInputFormAddUser::GetPasswordConfirm() const
{

```



```

        return m_consoleInputForm.GetInputFieldsValues().at(FIELD_CONFIRM_PASSWORD);
    }

```

## Файл ConsoleInputFormPatient.h

```

#pragma once

#include "ConsoleInputForm.h"

class Language;

class ConsoleInputFormPatient
{
    static const std::string FIELD_NAME;
    static const std::string FIELD_GENDER;
    static const std::string FIELD_BIRTHDATE;
    static const std::string FIELD_CITY;
    static const std::string FIELD_PHONE;
    static const std::string FIELD_DIAGNOSES;

public:
    explicit ConsoleInputFormPatient(Console& console, const Language& language,
        ConsoleInputForm::FieldCheckValuePredicat checkFieldGenderPredicate,
        ConsoleInputForm::FieldCheckValuePredicat checkFieldBirthDatePredicate);

    const std::string& GetName() const;
    const std::string& GetGender() const;
    const std::string& GetBirthDate() const;
    const std::string& GetCity() const;
    const std::string& GetPhone() const;
    const std::string& GetDiagnoses() const;

private:
    bool CheckFieldNoEmptyPredicate(const ConsoleInputForm::FieldName& fieldName,
        const std::string& fieldValue, std::string& fieldValueFormatDescription);

private:
    const Language& m_language;
    ConsoleInputForm m_consoleInputForm;
};

```

## Файл ConsoleInputFormPatient.cpp

```

#include "stdafx.h"
#include "ConsoleInputFormPatient.h"
#include "Language.h"

const std::string ConsoleInputFormPatient::FIELD_NAME = "name";
const std::string ConsoleInputFormPatient::FIELD_GENDER = "gender";
const std::string ConsoleInputFormPatient::FIELD_BIRTHDATE = "birthdate";
const std::string ConsoleInputFormPatient::FIELD_CITY = "city";
const std::string ConsoleInputFormPatient::FIELD_PHONE = "phone";
const std::string ConsoleInputFormPatient::FIELD_DIAGNOSES = "diagnoses";

namespace ph = std::placeholders;

ConsoleInputFormPatient::ConsoleInputFormPatient(Console& console, const Language& language,
    ConsoleInputForm::FieldCheckValuePredicat checkFieldGenderPredicate,
    ConsoleInputForm::FieldCheckValuePredicat checkFieldBirthDatePredicate)
:
    m_language(language),
    m_consoleInputForm(console, language, ConsoleInputForm::InputFields
    {
        ConsoleInputForm::InputFieldItem
        {
            FIELD_NAME,
            false,
            language.GetString(LanguageString::PatientFormName),
            std::bind(&ConsoleInputFormPatient::CheckFieldNoEmptyPredicate, this, ph::_1, ph::_2, ph::_3)
        },
        ConsoleInputForm::InputFieldItem
        {
            FIELD_GENDER,
            false,

```

```

        language.GetString(LanguageString::PatientFormGender),
        checkFieldGenderPredicate
    },
    ConsoleInputForm::InputFieldItem
    {
        FIELD_BIRTHDATE,
        false,
        language.GetString(LanguageString::PatientFormBirthDate),
        checkFieldBirthDatePredicate
    },
    ConsoleInputForm::InputFieldItem
    {
        FIELD_CITY,
        false,
        language.GetString(LanguageString::PatientFormCity),
        std::bind(&ConsoleInputFormPatient::CheckFieldNoEmptyPredicate, this, ph::_1, ph::_2, ph::_3)
    },
    ConsoleInputForm::InputFieldItem
    {
        FIELD_PHONE,
        false,
        language.GetString(LanguageString::PatientFormPhone),
        nullptr
    },
    ConsoleInputForm::InputFieldItem
    {
        FIELD_DIAGNOSES,
        false,
        language.GetString(LanguageString::PatientFormDiagnoses),
        nullptr
    },
    })
{
    m_consoleInputForm.DoInputFormFields();
}

const std::string& ConsoleInputFormPatient::GetName() const
{
    return m_consoleInputForm.GetInputFieldsValues().at(FIELD_NAME);
}

const std::string& ConsoleInputFormPatient::GetGender() const
{
    return m_consoleInputForm.GetInputFieldsValues().at(FIELD_GENDER);
}

const std::string& ConsoleInputFormPatient::GetBirthDate() const
{
    return m_consoleInputForm.GetInputFieldsValues().at(FIELD_BIRTHDATE);
}

const std::string& ConsoleInputFormPatient::GetCity() const
{
    return m_consoleInputForm.GetInputFieldsValues().at(FIELD_CITY);
}

const std::string& ConsoleInputFormPatient::GetPhone() const
{
    return m_consoleInputForm.GetInputFieldsValues().at(FIELD_PHONE);
}

const std::string& ConsoleInputFormPatient::GetDiagnoses() const
{
    return m_consoleInputForm.GetInputFieldsValues().at(FIELD_DIAGNOSES);
}

bool ConsoleInputFormPatient::CheckFieldNoEmptyPredicate(const ConsoleInputForm::FieldName& fieldName,
    const std::string& fieldValue, std::string& fieldValueFormatDescription)
{
    if (fieldValue.empty())
    {
        fieldValueFormatDescription = m_language.GetString(LanguageString::EmptyValueNotAllowed);

        return false;
    }

    return true;
}

```

```
}
```

## Файл Utils.h

```
#pragma once

class Console;
class Language;

class Utils
{
public:
    static bool RequestExistFilePath(Console& console, const Language& language, std::string& filePath);
    static bool RequestFilePath(Console& console, const Language& language, bool checkExist, std::string& filePath);
    static bool RequestExistFilePathWithDefaultValue(Console& console, const Language& language, const std::string& defaultPath, std::string& filePath);
    static bool RequestFilePathWithDefaultValue(Console& console, const Language& language, bool checkExist, const std::string& defaultPath, std::string& filePath);

    static const std::string ConvertStringFromLocalCodepageToUtf8(const std::string& str);

private:
    static bool CheckExistPatientsFile(Console& console, const Language& language, const std::string& filePath);
};
```

## Файл Utils.cpp

```
#include "stdafx.h"
#include "Utils.h"
#include "Console.h"
#include "Language.h"

bool Utils::RequestExistFilePath(Console& console, const Language& language, std::string& filePath)
{
    return RequestFilePath(console, language, true, filePath);
}

bool Utils::RequestFilePath(Console& console, const Language& language, bool checkExist, std::string& filePath)
{
    return RequestFilePathWithDefaultValue(console, language, checkExist, "", filePath);
}

bool Utils::RequestExistFilePathWithDefaultValue(Console& console, const Language& language, const std::string& defaultPath, std::string& filePath)
{
    return RequestFilePathWithDefaultValue(console, language, true, defaultPath, filePath);
}

bool Utils::RequestFilePathWithDefaultValue(Console& console, const Language& language, bool checkExist, const std::string& defaultPath, std::string& filePath)
{
    while (true)
    {
        console.PrintYellow(language.GetString(LanguageString::EnterPathToPatientsDatabase));

        if (!defaultPath.empty())
        {
            console.PrintYellow(u8" [%s]", defaultPath.c_str());
        }

        console.PrintYellow(": ");

        std::string inputValue = console.RequestInputString();

        if (inputValue.empty() && !defaultPath.empty())
        {
            inputValue = defaultPath;
        }

        if (checkExist &&
```

```

        !CheckExistPatientsFile(console, language, inputValue))
    {
        return false;
    }

    filePath = inputValue;

    return true;
}

return false;
}

bool Utils::CheckExistPatientsFile(Console& console, const Language& language, const std::string&
filePath)
{
    try
    {
        if (!boost::filesystem::exists(filePath))
        {

            console.PrintErrorWithNewLine(language.GetString(LanguageString::PatientsDatabaseFileNotFound));

            return false;
        }
    }
    catch (const boost::filesystem::filesystem_error&)
    {
        console.PrintErrorWithNewLine(language.GetString(LanguageString::CantCheckExistPatientsDatabase));

        return false;
    }

    return true;
}

const std::string Utils::ConvertStringFromLocalCodepageToUtf8(const std::string& str)
{
    int iUnicodeLen = ::MultiByteToWideChar(CP_ACP, 0, str.c_str(), (int)str.size(), NULL, 0);

    if (iUnicodeLen <= 0)
    {
        return "";
    }

    std::vector<wchar_t> unicodeString(iUnicodeLen + 1);
    iUnicodeLen = ::MultiByteToWideChar(CP_ACP, 0, str.c_str(), (int)str.size(),
        unicodeString.data(), (int)iUnicodeLen);

    if (iUnicodeLen <= 0)
    {
        return "";
    }

    unicodeString[iUnicodeLen] = 0;

    int iUtf8Len = ::WideCharToMultiByte(CP_UTF8, 0, unicodeString.data(), iUnicodeLen, NULL, 0, NULL,
    NULL);

    if (iUtf8Len <= 0)
    {
        return "";
    }

    std::vector<char> utf8String(iUtf8Len + 1);
    iUtf8Len = ::WideCharToMultiByte(CP_UTF8, 0, unicodeString.data(), iUnicodeLen, utf8String.data(),
    iUtf8Len, NULL, NULL);

    if (iUtf8Len <= 0)
    {
        return "";
    }

    utf8String[iUtf8Len] = 0;

    return utf8String.data();
}

```

```
}
```

## Файл Language.h

```
#pragma once
```

```
enum class LanguageString : uint32_t
{
    Welcom,
    UnknowError,
    CantOpenUsersDatabase,
    LoginFailed,
    LoginSuccess,
    LoginedUserIsAdmin,
    QuestionRepeatEnterYesNo,
    QuestionContinueYesNo,
    PatientsDatabaseDefaultHasOpenend,
    PatientsDatabaseDefaultHasCreatedAndOpenend,
    CantOpenPatientsDatabase,
    AdminNotFoundInUsersDatabase,
    QuestionCreateFirstAdmin,
    CreateNewUsersDatabaseInfo,
    CreateAdminInNewUsersDatabaseInfo,
    UsersDatabaseCreated,
    AdminCreatedInUsersDatabase,
    NewUserFormCaption,
    PasswordsNotEq,
    IncorrectValue,
    EmptyValueNotAllowed,
    UserFormName,
    UserFormPassword,
    UserFormConfirmPassword,
    LoginName,
    LoginPasword,
    PatientFormName,
    PatientFormGender,
    PatientFormBirthDate,
    PatientFormCity,
    PatientFormPhone,
    PatientFormDiagnoses,
    EnterPathToPatientsDatabase,
    PatientsDatabaseFileNotFound,
    CantCheckExistPatientsDatabase,
    MenuSelectItem,
    MenuReturnToParent,
    MenuExit,
    MenuControlUsers,
    MenuControlPatientsDatabasesFiles,
    MenuControlData,
    MenuEditPatientsData,
    MenuProcessPatientsData,
    MenuViewNonresidentPatients,
    MenuViewPatientsByAgeAndDiagnose,
    MenuSearchPatients,
    MenuOrderPatients,
    MenuSearchPatientsByName,
    MenuSearchPatientsByCity,
    MenuSearchPatientsByDiagnose,
    MenuSearchPatientsByPhone,
    MenuOrderPatientsByName,
    MenuOrderPatientsByCity,
    MenuOrderPatientsByBirthDate,
    MenuViewAllPatients,
    MenuAddPatient,
    MenuEditPatient,
    MenuRemovePatient,
    MenuPatientsFileCreateFile,
    MenuPatientsFileOpen,
    MenuPatientsCloseFile,
    MenuPatientsRemoveFile,
    MenuViewAllUsers,
    MenuAddNewUser,
    MenuEditUser,
    MenuRemoveUser,
    FoundRecords,
    NotFoundAnyRecord,
}
```

```

    UserID,
    UserName,
    UserRole,
    UserRoleIsAdmin,
    UserRoleIsUser,
    CreateNewUserAsAdmin,
    UserLoginExistAlready,
    UsersAdded,
    CantAddUser,
    RemoveUserConfirm,
    UserRemoved,
    CantRemoveUser,
    EnterUserId,
    UserIdNotFound,
    UserFound,
    EditUserConfirm,
    EditUserChangeRole,
    EditUserAsAdmin,
    EditUserChangeName,
    EditUserNewName,
    EditUserChangePassword,
    EditUserNewPassword,
    EditUserNewPasswordConfirm,
    UserChanged,
    CantChangeUser,
    PatientId,
    PatientName,
    PatientGender,
    PatientBirthDate,
    PatientCity,
    PatientPhone,
    PatientDiagnoses,
    GenderMale,
    GenderFeemale,
    GenderShemale,
    PatientAdded,
    PatientEditConfirm,
    PatientChanged,
    CantChangePatient,
    PatientRemoveConfirm,
    PatientRemoved,
    CantRemovePatient,
    SearchPatientName,
    SearchPatientCity,
    SearchPatientDiagnose,
    SearchPatientPhone,
    SearchPatientAge,
    EnterPatientId,
    PatientIdNotFound,
    PatientFound,
    PatientsFileExistAlready,
    PatientsFileCreated,
    PatientsFileOpened,
    PatientsFileClosed,
    RemoveFileConfirm,
    CantRemoveOpenedPatientsFile,
    PatientsFileRemoved,
    CantRemovePatientsFile,
    NotFoundOpenedPatientsFile,
};

class Language
{
    using LanguageStringsMap_t = std::unordered_map<LanguageString, std::string>;

    static const LanguageStringsMap_t DEFAULT_LANGUAGE_STRINGS;

public:
    Language();

    const char* GetString(const LanguageString languageStringId) const;

private:
    LanguageStringsMap_t m_strings;
};

```

## Файл Language.cpp

```
#include "stdafx.h"
#include "Language.h"

const Language::LanguageStringsMap_t Language::DEFAULT_LANGUAGE_STRINGS = {
    {
        LanguageString::Welcom,
        u8"* Программа учета сведений о пациентах медицинского центра.\r\n"
        u8"\r\n"
        u8"* ИИТ БГУИР. Факультет Компьютерных Технологий.\r\n"
        u8"* Специальность \"Инженерно-психологическое обеспечение информационных технологий\" (ИПОИТ).\r\n"
        u8"* Курсовой проект по дисциплине \"Основы конструирования программ\".\r\n"
        u8"* Выполнил: Студент 1го курса группы 680971, Микулич Василий Александрович.\r\n"
        u8"* Руководитель: Меженная Марина Михайловна\r\n"
        u8"\r\n"
    },
    {
        LanguageString::UnknowError,
        u8"В программе произошла неизвестная ошибка. Обратитесь к разработчику."
    },
    {
        LanguageString::CantOpenUsersDatabase,
        u8"Не удалось открыть базу данных пользователей. Обратитесь к системному администратору."
    },
    {
        LanguageString::LoginFailed,
        u8"Имя пользователя или пароль не верны."
    },
    {
        LanguageString::LoginSuccess,
        u8"Вход в систему выполнен успешно."
    },
    {
        LanguageString::LoggedInUserIsAdmin,
        u8"Вы зашли в систему с правами администратора."
    },
    {
        LanguageString::QuestionRepeatEnterYesNo,
        u8"Повторить ввод?"
    },
    {
        LanguageString::QuestionContinueYesNo,
        u8"Продолжить?"
    },
    {
        LanguageString::PatientsDatabaseDefaultHasOpenend,
        u8"Открыт файл базы данных пациентов по умолчанию: %s."
    },
    {
        LanguageString::PatientsDatabaseDefaultHasCreatedAndOpenend,
        u8"Создан и открыт файл базы данных пациентов по умолчанию: %s."
    },
    {
        LanguageString::CantOpenPatientsDatabase,
        u8"Не удалось открыть базу данных. Убедитесь, что Вы указали корректный путь."
    },
    {
        LanguageString::AdminNotFoundInUsersDatabase,
        u8"База данных не содержит учётной записи администратора."
    },
    {
        LanguageString::QuestionCreateFirstAdmin,
        u8"Для продолжения работы необходимо создать учётную запись администратора. Продолжить?"
    },
    {
        LanguageString::CreateNewUsersDatabaseInfo,
        u8"Первый запуск. Будет создана новая база данных пользователей."
    },
    {
        LanguageString::CreateAdminInNewUsersDatabaseInfo,
        u8"При создании базы данных будет создана учётная запись администратора."
    },
    {
        LanguageString::UsersDatabaseCreated,
```

```

        u8"База данных пользователей успешно создана."
    },
    {
        LanguageString::AdminCreatedInUsersDatabase,
        u8"Учётная запись администратора успешно добавлена в базу данных пользователей."
    },
    {
        LanguageString::IncorrectValue,
        u8"Вы ввели некорректное значение. Повторите ввод."
    },
    {
        LanguageString::EmptyValueNotAllowed,
        u8"Поле не может быть пустым."
    },
    {
        LanguageString::NewUserFormCaption,
        u8"Введите данные для новой учётной записи."
    },
    {
        LanguageString::UserFormName,
        u8"Имя пользователя (login)"
    },
    {
        LanguageString::UserFormPassword,
        u8"Пароль"
    },
    {
        LanguageString::UserFormConfirmPassword,
        u8"Повторите ввод пароля"
    },
    {
        LanguageString::PasswordsNotEq,
        u8"Пароли не совпадают."
    },
    {
        LanguageString::LoginName,
        u8"Имя пользователя"
    },
    {
        LanguageString::LoginPasword,
        u8"Пароль"
    },
    {
        LanguageString::PatientFormName,
        u8"ФИО пациента"
    },
    {
        LanguageString::PatientFormGender,
        u8"Пол (м/ж/т)"
    },
    {
        LanguageString::PatientFormBirthDate,
        u8"Дата рождения (ДД.ММ.ГГГГ)"
    },
    {
        LanguageString::PatientFormCity,
        u8"Город"
    },
    {
        LanguageString::PatientFormPhone,
        u8"Телефон"
    },
    {
        LanguageString::PatientFormDiagnoses,
        u8"Диагнозы (через запятую)"
    },
    {
        LanguageString::MenuSelectItem,
        u8"Выберите пункт меню: "
    },
    {
        LanguageString::MenuReturnToParent,
        u8"0. Выйти в родительское меню"
    },
    {
        LanguageString::MenuExit,
        u8"0. Выйти из программы"
    }

```



```

    },
    {
        LanguageString::EnterPathToPatientsDatabase,
        u8"Введите путь к файлу базы данных пациентов"
    },
    {
        LanguageString::PatientsDatabaseFileNotFound,
        u8"Не удалось найти файл по указанному пути."
    },
    {
        LanguageString::CantCheckExistPatientsDatabase,
        u8"Не удалось проверить существование файла."
    },
    {
        LanguageString::MenuControlUsers,
        u8"Управление учетными записями пользователей"
    },
    {
        LanguageString::MenuControlPatientsDatabasesFiles,
        u8"Управление файлами данных"
    },
    {
        LanguageString::MenuControlData,
        u8"Обработка данных"
    },
    {
        LanguageString::MenuEditPatientsData,
        u8"Редактирование данных о пациентах"
    },
    {
        LanguageString::MenuProcessPatientsData,
        u8"Обработка данных о пациентах"
    },
    {
        LanguageString::MenuViewNonresidentPatients,
        u8"Показать иногородних пациентов"
    },
    {
        LanguageString::MenuViewPatientsByAgeAndDiagnose,
        u8"Показать пациентов старше X лет с диагнозом Y"
    },
    {
        LanguageString::MenuSearchPatients,
        u8"Поиск пациентов"
    },
    {
        LanguageString::MenuOrderPatients,
        u8"Сортировка пациентов"
    },
    {
        LanguageString::MenuSearchPatientsByName,
        u8"Поиск пациентов по имени"
    },
    {
        LanguageString::MenuSearchPatientsByCity,
        u8"Поиск пациентов по городу"
    },
    {
        LanguageString::MenuSearchPatientsByDiagnose,
        u8"Поиск пациентов по диагнозу"
    },
    {
        LanguageString::MenuSearchPatientsByPhone,
        u8"Поиск пациентов по номеру телефона"
    },
    {
        LanguageString::MenuOrderPatientsByName,
        u8"Сортировка пациентов по имени"
    },
    {
        LanguageString::MenuOrderPatientsByCity,
        u8"Сортировка пациентов по городу"
    },
    {
        LanguageString::MenuOrderPatientsByBirthDate,
        u8"Сортировка пациентов по дате рождения"
    },
    },

```

```

{
    LanguageString::MenuViewAllPatients,
    u8"Просмотр всех пациентов"
},
{
    LanguageString::MenuAddPatient,
    u8"Добавить пациента"
},
{
    LanguageString::MenuEditPatient,
    u8"Изменить пациента"
},
{
    LanguageString::MenuRemovePatient,
    u8"Удалить пациента"
},
{
    LanguageString::MenuPatientsFileCreateFile,
    u8"Создать файл данных пациентов"
},
{
    LanguageString::MenuPatientsFileOpen,
    u8"Открыть файл данных пациентов"
},
{
    LanguageString::MenuPatientsCloseFile,
    u8"Закрыть файл данных пациентов"
},
{
    LanguageString::MenuPatientsRemoveFile,
    u8"Удалить файл данных пациентов"
},
{
    LanguageString::RemoveFileConfirm,
    u8"Вы действительно хотите удалить файл?"
},
{
    LanguageString::MenuViewAllUsers,
    u8"Просмотр всех учетных записей"
},
{
    LanguageString::MenuAddNewUser,
    u8"Добавление новой учётной записи"
},
{
    LanguageString::MenuEditUser,
    u8"Редактирование учётной записи"
},
{
    LanguageString::MenuRemoveUser,
    u8"Удаление учётной записи"
},
{
    LanguageString::FoundRecords,
    u8"Найдено записей: "
},
{
    LanguageString::NotFoundAnyRecord,
    u8"Записи не найдены"
},
{
    LanguageString::UserID,
    u8"ID"
},
{
    LanguageString::UserName,
    u8"Имя пользователя"
},
{
    LanguageString::UserRole,
    u8"Роль"
},
{
    LanguageString::UserRoleIsAdmin,
    u8"Администратор"
},
{

```

```

    LanguageString::UserRoleIsUser,
    u8"Пользователь"
},
{
    LanguageString::CreateNewUserAsAdmin,
    u8"Предоставить пользователю права администратора?"
},
{
    LanguageString::UserLoginExistAlready,
    u8"Пользователь с таким именем уже существует."
},
{
    LanguageString::UsersAdded,
    u8"Новый пользователь успешно добавлен."
},
{
    LanguageString::CantAddUser,
    u8"При добавлении пользователя произошла ошибка."
},
{
    LanguageString::RemoveUserConfirm,
    u8"Вы действительно хотите удалить этого пользователя?"
},
{
    LanguageString::UserRemoved,
    u8"Пользователь успешно удалён."
},
{
    LanguageString::CantRemoveUser,
    u8"При удалении пользователя произошла ошибка."
},
{
    LanguageString::EnterUserId,
    u8"Введите ID пользователя: "
},
{
    LanguageString::UserIdNotFound,
    u8"Пользователя с таким ID не существует."
},
{
    LanguageString::UserFound,
    u8"Пользователь найден:"
},
{
    LanguageString::EditUserConfirm,
    u8"Вы действительно хотите изменить этого пользователя?"
},
{
    LanguageString::EditUserChangeRole,
    u8"Изменить роль пользователя?"
},
{
    LanguageString::EditUserAsAdmin,
    u8"Предоставить пользователю права администратора?"
},
{
    LanguageString::EditUserChangeName,
    u8"Изменить имя пользователя?"
},
{
    LanguageString::EditUserNewName,
    u8"Введите новое имя пользователя (login): "
},
{
    LanguageString::EditUserChangePassword,
    u8"Изменить пароль пользователя?"
},
{
    LanguageString::EditUserNewPassword,
    u8"Введите новый пароль: "
},
{
    LanguageString::EditUserNewPasswordConfirm,
    u8"Повторите ввод нового пароля: "
},
{
    LanguageString::UserChanged,

```

```

        u8"Пользователь успешно изменён."
    },
    {
        LanguageString::CantChangeUser,
        u8"При изменении пользователя произошла ошибка."
    },
    {
        LanguageString::PatientId,
        u8"ID"
    },
    {
        LanguageString::PatientName,
        u8"Имя"
    },
    {
        LanguageString::PatientGender,
        u8"Пол"
    },
    {
        LanguageString::PatientBirthDate,
        u8"Дата рождения"
    },
    {
        LanguageString::PatientCity,
        u8"Город"
    },
    {
        LanguageString::PatientPhone,
        u8"Телефон"
    },
    {
        LanguageString::PatientDiagnoses,
        u8"Диагнозы"
    },
    {
        LanguageString::GenderMale,
        u8"Мужчина"
    },
    {
        LanguageString::GenderFeemale,
        u8"Женщина"
    },
    {
        LanguageString::GenderShemale,
        u8"Транссексуал"
    },
    {
        LanguageString::PatientAdded,
        u8"Пациент успешно добавлен."
    },
    {
        LanguageString::PatientEditConfirm,
        u8"Вы действительно хотите изменить этого пациента?"
    },
    {
        LanguageString::PatientChanged,
        u8"Пациент успешно изменён."
    },
    {
        LanguageString::CantChangePatient,
        u8"При изменении пациента произошла ошибка."
    },
    {
        LanguageString::PatientRemoveConfirm,
        u8"Вы действительно хотите удалить этого пациента?"
    },
    {
        LanguageString::PatientRemoved,
        u8"Пациент успешно удалён."
    },
    {
        LanguageString::CantRemovePatient,
        u8"При удалении пациента произошла ошибка."
    },
    {
        LanguageString::SearchPatientName,
        u8"Введите имя пациента: "
    }

```

```

    },
    {
        LanguageString::SearchPatientCity,
        u8"Введите название города: "
    },
    {
        LanguageString::SearchPatientDiagnose,
        u8"Введите диагноз: "
    },
    {
        LanguageString::SearchPatientPhone,
        u8"Введите номер телефона: "
    },
    {
        LanguageString::SearchPatientAge,
        u8"Введите возраст (полных лет): "
    },
    {
        LanguageString::EnterPatientId,
        u8"Введите ID пациента: "
    },
    {
        LanguageString::PatientIdNotFound,
        u8"Пациента с таким ID не существует."
    },
    {
        LanguageString::PatientFound,
        u8"Пациент найден:"
    },
    {
        LanguageString::PatientsFileExistAlready,
        u8"По указанному пути файл уже существует."
    },
    {
        LanguageString::PatientsFileCreated,
        u8"Файл успешно создан и открыт."
    },
    {
        LanguageString::PatientsFileOpened,
        u8"Файл успешно открыт."
    },
    {
        LanguageString::PatientsFileClosed,
        u8"Файл закрыт."
    },
    {
        LanguageString::CantRemoveOpenedPatientsFile,
        u8"Файл по указанному пути открыт. Закройте его и повторите попытку удаления."
    },
    {
        LanguageString::PatientsFileRemoved,
        u8"Файл успешно удалён."
    },
    {
        LanguageString::CantRemovePatientsFile,
        u8"Не удалось удалить файл."
    },
    {
        LanguageString::NotFoundOpenedPatientsFile,
        u8"Нет открытых файлов с данными о пациентах. Откройте файл и повторите попытку."
    },
};

Language::Language() :
    m_strings(DEFAULT_LANGUAGE_STRINGS)
{
}

const char* Language::GetString(const LanguageString languageStringId) const
{
    const auto it = m_strings.find(languageStringId);

    if (it == m_strings.cend())
    {
        std::string exceptionText("unknown language string id: ");
        exceptionText += std::to_string(static_cast<uint32_t>(languageStringId));
    }
}

```

```

        throw std::invalid_argument(exceptionText);
    }

    return it->second.c_str();
}

```

## Модуль UsersRegistry.lib

### Файл stdafx.h

```

#pragma once

#include "targetver.h"

#define WIN32_LEAN_AND_MEAN           // Exclude rarely-used stuff from Windows headers

// stl
#include <string>
#include <vector>
#include <memory>
#include <thread>
#include <mutex>
#include <map>
#include <algorithm>

// sqlite
#include <sqlite3.h>

// openssl
#include <openssl/bio.h>
#include <openssl/evp.h>
#include <openssl/hmac.h>
#include <openssl/buffer.h>

```

### Файл UserRole.h

```

#pragma once

enum class UserRole
{
    Unknown,
    Admin,
    User,
};

```

### Файл User.h

```

#pragma once

#include "UserRole.h"

class User
{
public:
    User();
    explicit User(const std::string& login, const UserRole role);

    User(const User&);
    User& operator= (const User&);

    static const User& GetEmptyUser();

    const std::string& GetLogin() const;
    const UserRole GetRole() const;

private:
    std::string m_login;
    UserRole m_role;
};

using UserId_t = uint64_t;
using UserRecord_t = std::pair<UserId_t, User>;

```

```
static const UserId_t USERID_EMPTY = 0;

bool operator==(const User& lhs, const User& rhs);
bool operator!=(const User& lhs, const User& rhs);
```

## Файл User.cpp

```
#include "stdafx.h"
#include "User.h"

bool operator==(const User& lhs, const User& rhs)
{
    return (lhs.GetLogin() == rhs.GetLogin() &&
        lhs.GetRole() == rhs.GetRole());
};

bool operator!=(const User& lhs, const User& rhs)
{
    return (lhs.GetLogin() != rhs.GetLogin() ||
        lhs.GetRole() != rhs.GetRole());
};

User::User() :
    m_login(""),
    m_role(UserRole::Unknown)
{
}

User::User(const std::string& login, const UserRole role) :
    m_login(login),
    m_role(role)
{
}

User::User(const User& other) :
    m_login(other.m_login),
    m_role(other.m_role)
{
}

User& User::operator= (const User& other)
{
    m_login = other.m_login;
    m_role = other.m_role;

    return *this;
}

const User& User::GetEmptyUser()
{
    static User emptyUser;

    return emptyUser;
}

const std::string& User::GetLogin() const
{
    return m_login;
}

const UserRole User::GetRole() const
{
    return m_role;
}
```

## Файл UsersRegistry.h

```
#pragma once

#include "User.h"

class UsersRegistryImpl;

class UsersRegistry
{

```

```

public:
    static std::shared_ptr<UsersRegistry> CreateUsersRegistry(const std::string& connectionString);
    static std::shared_ptr<UsersRegistry> OpenUsersRegistry(const std::string& connectionString, bool
readOnly);

    UsersRegistry() = delete;
    UsersRegistry(const UsersRegistry&) = delete;
    UsersRegistry& operator = (const UsersRegistry&) = delete;

private:
    explicit UsersRegistry(const std::string& connectionString, bool createIfNotExist, bool readOnly);

public:
    uint64_t GetAllUsersCount() const;
    uint64_t GetAdminUsersCount() const;
    const std::vector<UserRecord_t> GetAllUsers() const;
    bool AddUser(const User& user, const std::string& password);
    bool DeleteUser(const UserId_t userId);
    bool DeleteUserByLogin(const std::string& login);
    bool UpdateUser(const UserId_t userId, const User& user);
    bool UpdateUserAndPassword(const UserId_t userId, const User& user, const std::string& password);
    bool IsExistUser(const std::string& login) const;
    UserRecord_t GetUserByLogin(const std::string& login) const;
    UserRecord_t GetUserById(const UserId_t userId) const;

    bool CheckUserPassword(const std::string& login, const std::string& password) const;

private:
    std::unique_ptr<UsersRegistryImpl> m_usersRegistryImpl;
};

```

## Файл UsersRegistry.cpp

```

#include "stdafx.h"
#include "UsersRegistry.h"
#include "UsersRegistryImpl.h"

UsersRegistry::UsersRegistry(const std::string& connectionString, bool createIfNotExist, bool readOnly) :
    m_usersRegistryImpl(std::make_unique<UsersRegistryImpl>(connectionString, createIfNotExist, readOnly))
{
}

std::shared_ptr<UsersRegistry> UsersRegistry::CreateUsersRegistry(const std::string& connectionString)
{
    return std::shared_ptr<UsersRegistry>(new UsersRegistry(connectionString, true, false));
}

std::shared_ptr<UsersRegistry> UsersRegistry::OpenUsersRegistry(const std::string& connectionString, bool
readOnly)
{
    return std::shared_ptr<UsersRegistry>(new UsersRegistry(connectionString, false, readOnly));
}

uint64_t UsersRegistry::GetAllUsersCount() const
{
    return m_usersRegistryImpl->GetAllUsersCount();
}

uint64_t UsersRegistry::GetAdminUsersCount() const
{
    return m_usersRegistryImpl->GetAdminUsersCount();
}

const std::vector<UserRecord_t> UsersRegistry::GetAllUsers() const
{
    return m_usersRegistryImpl->GetAllUsers();
}

bool UsersRegistry::AddUser(const User& user, const std::string& password)
{
    return m_usersRegistryImpl->AddUser(user, password);
}

bool UsersRegistry::DeleteUser(const UserId_t userId)
{
}

```



```

        return m_usersRegistryImpl->DeleteUser(userId);
    }

    bool UsersRegistry::DeleteUserByLogin(const std::string& login)
    {
        return m_usersRegistryImpl->DeleteUserByLogin(login);
    }

    bool UsersRegistry::UpdateUser(const UserId_t userId, const User& user)
    {
        return m_usersRegistryImpl->UpdateUser(userId, user);
    }

    bool UsersRegistry::UpdateUserAndPassword(const UserId_t userId, const User& user, const std::string& password)
    {
        return m_usersRegistryImpl->UpdateUserAndPassword(userId, user, password);
    }

    bool UsersRegistry::IsExistUser(const std::string& login) const
    {
        return m_usersRegistryImpl->IsExistUser(login);
    }

    UserRecord_t UsersRegistry::GetUserByLogin(const std::string& login) const
    {
        return m_usersRegistryImpl->GetUserByLogin(login);
    }

    UserRecord_t UsersRegistry::GetUserById(const UserId_t userId) const
    {
        return m_usersRegistryImpl->GetUserById(userId);
    }

    bool UsersRegistry::CheckUserPassword(const std::string& login, const std::string& password) const
    {
        return m_usersRegistryImpl->CheckUserPassword(login, password);
    }
}

```

## Файл UsersRegistryImpl.h

```

#pragma once

#include "../DatabaseSQLite/DatabaseSQLite.h"
#include "User.h"

class UsersRegistryImpl
{
public:
    explicit UsersRegistryImpl(const std::string& connectionString, bool createIfNotExist, bool readOnly);

    UsersRegistryImpl() = delete;
    UsersRegistryImpl(const UsersRegistryImpl&) = delete;
    UsersRegistryImpl& operator = (const UsersRegistryImpl&) = delete;

public:
    const std::vector<UserRecord_t> GetAllUsers() const;
    uint64_t GetAllUsersCount() const;
    uint64_t GetAdminUsersCount() const;
    bool AddUser(const User& user, const std::string& password);
    bool DeleteUser(const UserId_t userId);
    bool DeleteUserByLogin(const std::string& login);
    bool UpdateUser(const UserId_t userId, const User& user);
    bool UpdateUserAndPassword(const UserId_t userId, const User& user, const std::string& password);
    bool IsExistUser(const std::string& login) const;
    UserRecord_t GetUserByLogin(const std::string& login) const;
    UserRecord_t GetUserById(const UserId_t userId) const;

    bool CheckUserPassword(const std::string& login, const std::string& password) const;

private:
    void ValidateDatabaseSQLiteSchema();

    bool IsExistUserInDatabase(const std::string& login) const;
}

```

```

        UserRole GetRoleById(const uint32_t roleId) const;
        uint32_t GetRoleId(const UserRole role);

private:
        mutable std::mutex m_mutex;

        DatabaseSQLite m_sqliteDatabase;

        SQLiteStatement_t m_checkUserPasswordStatement;
};

```

## Файл UsersRegistryImpl.cpp

```

#include "stdafx.h"
#include "UsersRegistryImpl.h"
#include "DatabaseSchemaSql.h"
#include "PasswordKeyGenerator.h"
#include "../DatabaseSQLite/DatabaseSQLiteException.h"

UsersRegistryImpl::UsersRegistryImpl(const std::string& connectionString, bool createIfNotExist, bool
readOnly) :
        m_sqliteDatabase(connectionString, createIfNotExist, readOnly)
{
        if (!readOnly)
        {
                ValidateDatabaseSQLiteSchema();
        }

        m_checkUserPasswordStatement =
m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_CHECK_USER_AND_PASSWORD);
}

bool UsersRegistryImpl::IsExistUser(const std::string& login) const
{
        std::lock_guard<std::mutex> lock(m_mutex);

        return IsExistUserInDatabase(login);
}

bool UsersRegistryImpl::AddUser(const User& user, const std::string& password)
{
        std::lock_guard<std::mutex> lock(m_mutex);

        if (IsExistUserInDatabase(user.GetLogin()))
        {
                return false;
        }

        const std::string passwordKey = PasswordKeyGenerator::GenerateKey_PBKDF2_HMAC(password);

        SQLiteStatement_t addUserStatement =
m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_INSERT_USER);

        sqlite3_bind_text(addUserStatement.get(), 1, user.GetLogin().c_str(), (int)user.GetLogin().size(),
SQLITE_STATIC);
        sqlite3_bind_text(addUserStatement.get(), 2, passwordKey.c_str(), (int)passwordKey.size(),
SQLITE_STATIC);
        sqlite3_bind_int(addUserStatement.get(), 3, GetRoleId(user.GetRole()));

        int sqliteStepResult = sqlite3_step(addUserStatement.get());

        if (sqliteStepResult != SQLITE_DONE)
        {
                const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSQLiteConnectionPtr());

                throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
                        sqliteStepResult);
        }

        return true;
}

bool UsersRegistryImpl::DeleteUser(const UserId_t userId)
{
        std::lock_guard<std::mutex> lock(m_mutex);

```

```

        SQLiteStatement_t deleteUserStatement =
m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_DELETE_USER_BY_ID);

        sqlite3_bind_int64(deleteUserStatement.get(), 1, userId);

        int sqliteStepResult = sqlite3_step(deleteUserStatement.get());

        if (sqliteStepResult == SQLITE_DONE)
        {
            if (sqlite3_changes(m_sqliteDatabase.GetSqliteConnectionPtr()) == 0)
            {
                return false;
            }
            else
            {
                return true;
            }
        }
        else
        {
            const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

            throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
                sqliteStepResult);
        }
    }

bool UsersRegistryImpl::DeleteUserByLogin(const std::string& login)
{
    std::lock_guard<std::mutex> lock(m_mutex);

    SQLiteStatement_t deleteUserStatement =
m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_DELETE_USER_BY_LOGIN);

    sqlite3_bind_text(deleteUserStatement.get(), 1, login.c_str(), (int)login.size(), SQLITE_STATIC);

    int sqliteStepResult = sqlite3_step(deleteUserStatement.get());

    if (sqliteStepResult == SQLITE_DONE)
    {
        if (sqlite3_changes(m_sqliteDatabase.GetSqliteConnectionPtr()) == 0)
        {
            return false;
        }
        else
        {
            return true;
        }
    }
    else
    {
        const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

        throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
            sqliteStepResult);
    }
}

bool UsersRegistryImpl::UpdateUser(const UserId_t userId, const User& user)
{
    std::lock_guard<std::mutex> lock(m_mutex);

    SQLiteStatement_t addUserStatement = m_sqliteDatabase.CreateStatement(
        DatabaseSchemaSql::SQL_UPDATE_USER_WITHOUT_PASSWORD);

    sqlite3_bind_text(addUserStatement.get(), 1, user.GetLogin().c_str(), (int)user.GetLogin().size(),
        SQLITE_STATIC);
    sqlite3_bind_int(addUserStatement.get(), 2, GetRoleId(user.GetRole()));
    sqlite3_bind_int64(addUserStatement.get(), 3, userId);

    int sqliteStepResult = sqlite3_step(addUserStatement.get());

    if (sqliteStepResult == SQLITE_DONE)
    {
        if (sqlite3_changes(m_sqliteDatabase.GetSqliteConnectionPtr()) == 0)
        {

```

```

        return false;
    }
    else
    {
        return true;
    }
}
else
{
    const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

    throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
        sqliteStepResult);
}
}

bool UsersRegistryImpl::UpdateUserAndPassword(const UserId_t userId, const User& user, const std::string&
password)
{
    std::lock_guard<std::mutex> lock(m_mutex);

    const std::string passwordKey = PasswordKeyGenerator::GenerateKey_PBKDF2_HMAC(password);

    SQLiteStatement_t addUserStatement =
m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_UPDATE_USER);

    sqlite3_bind_text(addUserStatement.get(), 1, user.GetLogin().c_str(), (int)user.GetLogin().size(),
    SQLITE_STATIC);
    sqlite3_bind_text(addUserStatement.get(), 2, passwordKey.c_str(), (int)passwordKey.size(),
    SQLITE_STATIC);
    sqlite3_bind_int(addUserStatement.get(), 3, GetRoleId(user.GetRole()));
    sqlite3_bind_int64(addUserStatement.get(), 4, userId);

    int sqliteStepResult = sqlite3_step(addUserStatement.get());

    if (sqliteStepResult == SQLITE_DONE)
    {
        if (sqlite3_changes(m_sqliteDatabase.GetSqliteConnectionPtr()) == 0)
        {
            return false;
        }
        else
        {
            return true;
        }
    }
    else
    {
        const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

        throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
            sqliteStepResult);
    }
}

const std::vector<UserRecord_t> UsersRegistryImpl::GetAllUsers() const
{
    std::lock_guard<std::mutex> lock(m_mutex);

    std::vector<UserRecord_t> allUsers;

    SQLiteStatement_t selectUsersStatement =
m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_SELECT_ALL_USERS);

    int sqliteStepResult = sqlite3_step(selectUsersStatement.get());

    while (sqliteStepResult == SQLITE_ROW)
    {
        const uint64_t userId = sqlite3_column_int64(selectUsersStatement.get(), 0);
        const std::string login = (const char*)sqlite3_column_text(selectUsersStatement.get(), 1);
        const uint32_t roleId = (uint32_t)sqlite3_column_int(selectUsersStatement.get(), 2);

        allUsers.emplace_back(std::make_pair(userId, User{
            login,
            GetRoleById(roleId)
        }));
    }
}

```

```

        sqliteStepResult = sqlite3_step(selectUsersStatement.get());
    }

    if (sqliteStepResult != SQLITE_DONE)
    {
        const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

        throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
            sqliteStepResult);
    }

    return allUsers;
}

uint64_t UsersRegistryImpl::GetAllUsersCount() const
{
    uint64_t allUsersCount = 0;

    std::lock_guard<std::mutex> lock(m_mutex);

    SQLiteStatement_t selectStatement =
        m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_SELECT_USERS_COUNT);

    int sqliteStepResult = sqlite3_step(selectStatement.get());

    if (sqliteStepResult == SQLITE_ROW)
    {
        allUsersCount = sqlite3_column_int64(selectStatement.get(), 0);
    }

    return allUsersCount;
}

uint64_t UsersRegistryImpl::GetAdminUsersCount() const
{
    uint64_t adminUsersCount = 0;

    std::lock_guard<std::mutex> lock(m_mutex);

    SQLiteStatement_t selectStatement = m_sqliteDatabase.CreateStatement(
        DatabaseSchemaSql::SQL_SELECT_ADMIN_USERS_COUNT);

    int sqliteStepResult = sqlite3_step(selectStatement.get());

    if (sqliteStepResult == SQLITE_ROW)
    {
        adminUsersCount = sqlite3_column_int64(selectStatement.get(), 0);
    }

    return adminUsersCount;
}

bool UsersRegistryImpl::CheckUserPassword(const std::string& login, const std::string& password) const
{
    const std::string passwordKey = PasswordKeyGenerator::GenerateKey_PBKDF2_HMAC(password);

    std::lock_guard<std::mutex> lock(m_mutex);

    sqlite3_reset(m_checkUserPasswordStatement.get());
    sqlite3_bind_text(m_checkUserPasswordStatement.get(), 1, login.c_str(), (int)login.size(),
        SQLITE_STATIC);
    sqlite3_bind_text(m_checkUserPasswordStatement.get(), 2, passwordKey.c_str(), (int)passwordKey.size(),
        SQLITE_STATIC);

    int sqliteStepResult = sqlite3_step(m_checkUserPasswordStatement.get());

    if (sqliteStepResult == SQLITE_ROW)
    {
        return true;
    }
    else if (sqliteStepResult == SQLITE_DONE)
    {
        return false;
    }
    else
    {

```

```

        const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

        throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
            sqliteStepResult);
    }
}

UserRecord_t UsersRegistryImpl::GetUserByLogin(const std::string& login) const
{
    std::lock_guard<std::mutex> lock(m_mutex);

    SQLiteStatement_t selectUserStatement = m_sqliteDatabase.CreateStatement(
        DatabaseSchemaSql::SQL_SELECT_USER_ID_AND_ROLE_BY_LOGIN);

    sqlite3_bind_text(selectUserStatement.get(), 1, login.c_str(), (int)login.size(), SQLITE_STATIC);

    int sqliteStepResult = sqlite3_step(selectUserStatement.get());

    if (sqliteStepResult == SQLITE_ROW)
    {
        const uint64_t userId = sqlite3_column_int64(selectUserStatement.get(), 0);
        const uint32_t roleId = (uint32_t)sqlite3_column_int(selectUserStatement.get(), 1);

        UserRecord_t userRecord = std::make_pair(userId, User{
            login,
            GetRoleById(roleId)
        });

        return userRecord;
    }
    else if (sqliteStepResult == SQLITE_DONE)
    {
        return std::make_pair(USERID_EMPTY, User::GetEmptyUser());
    }
    else
    {
        const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

        throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
            sqliteStepResult);
    }
}

UserRecord_t UsersRegistryImpl::GetUserById(const UserId_t userId) const
{
    if (userId == USERID_EMPTY)
    {
        return std::make_pair(userId, User::GetEmptyUser());
    }

    std::lock_guard<std::mutex> lock(m_mutex);

    SQLiteStatement_t selectUserStatement = m_sqliteDatabase.CreateStatement(
        DatabaseSchemaSql::SQL_SELECT_USER_LOGIN_AND_ROLE_BY_ID);

    sqlite3_bind_int64(selectUserStatement.get(), 1, userId);

    int sqliteStepResult = sqlite3_step(selectUserStatement.get());

    if (sqliteStepResult != SQLITE_ROW)
    {
        return std::make_pair(USERID_EMPTY, User::GetEmptyUser());
    }

    const std::string login = (const char*)sqlite3_column_text(selectUserStatement.get(), 0);
    const uint32_t roleId = (uint32_t)sqlite3_column_int(selectUserStatement.get(), 1);

    UserRecord_t userRecord = std::make_pair(userId, User{
        login,
        GetRoleById(roleId)
    });

    return userRecord;
}

bool UsersRegistryImpl::IsExistUserInDatabase(const std::string& login) const
{

```

```

        SQLiteStatement_t selectUserStatement = m_sqliteDatabase.CreateStatement(
            DatabaseSchemaSql::SQL_SELECT_USER_ID_BY_LOGIN);

        sqlite3_bind_text(selectUserStatement.get(), 1, login.c_str(), (int)login.size(), SQLITE_STATIC);

        int sqliteStepResult = sqlite3_step(selectUserStatement.get());

        if (sqliteStepResult == SQLITE_ROW)
        {
            return true;
        }
        else if (sqliteStepResult == SQLITE_DONE)
        {
            return false;
        }
        else
        {
            const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

            throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
                sqliteStepResult);
        }
    }

    UserRole UsersRegistryImpl::GetRoleById(const uint32_t roleId) const
    {
        return DatabaseSchemaSql::DATABASE_SQLITE_USER_ROLES_BIND_TABLE.at(roleId);
    }

    uint32_t UsersRegistryImpl::GetRoleId(const UserRole role)
    {
        const auto& bindMap = DatabaseSchemaSql::DATABASE_SQLITE_USER_ROLES_BIND_TABLE;

        auto it = std::find_if(bindMap.cbegin(), bindMap.cend(), [&role](const std::pair<uint32_t, UserRole>&
t) -> bool
        {
            return (t.second == role);
        });

        if (it == bindMap.cend())
        {
            throw std::invalid_argument("Invalid role value");
        }

        return it->first;
    }

    void UsersRegistryImpl::ValidateDatabaseSQLiteSchema()
    {
        m_sqliteDatabase.BeginTransaction();

        try
        {
            m_sqliteDatabase.ExecuteSql(DatabaseSchemaSql::USERS_DATABASE_SQLITE_SCHEMA);
        }
        catch (...)
        {
            m_sqliteDatabase.RollbackTransaction();

            throw;
        }

        m_sqliteDatabase.CommitTransaction();
    }
}

```

## Файл DatabaseSchemaSql.h

```

#pragma once

#include "UserRole.h"

struct DatabaseSchemaSql
{
    static const std::map<uint32_t, UserRole> DATABASE_SQLITE_USER_ROLES_BIND_TABLE;

    static const std::string USERS_DATABASE_SQLITE_SCHEMA;
}

```

```

static const std::string SQL_CHECK_USER_AND_PASSWORD;
static const std::string SQL_INSERT_USER;
static const std::string SQL_UPDATE_USER_WITHOUT_PASSWORD;
static const std::string SQL_UPDATE_USER;
static const std::string SQL_DELETE_USER_BY_ID;
static const std::string SQL_DELETE_USER_BY_LOGIN;
static const std::string SQL_SELECT_ALL_USERS;
static const std::string SQL_SELECT_USERS_COUNT;
static const std::string SQL_SELECT_ADMIN_USERS_COUNT;
static const std::string SQL_SELECT_USER_ID_AND_ROLE_BY_LOGIN;
static const std::string SQL_SELECT_USER_LOGIN_AND_ROLE_BY_ID;
static const std::string SQL_SELECT_USER_ID_BY_LOGIN;
};

```

## Файл DatabaseSchemaSql.cpp

```

#include "stdafx.h"
#include "DatabaseSchemaSql.h"

const std::map<uint32_t, UserRole> DatabaseSchemaSql::DATABASE_SQLITE_USER_ROLES_BIND_TABLE = {
    { 1, UserRole::Admin },
    { 2, UserRole::User },
};

const std::string DatabaseSchemaSql::USERS_DATABASE_SQLITE_SCHEMA = R"(
CREATE TABLE IF NOT EXISTS roles (
    role_id INTEGER PRIMARY KEY NOT NULL,
    role_name TEXT COLLATE NOCASE NOT NULL UNIQUE
);

CREATE UNIQUE INDEX IF NOT EXISTS roles_name_idx ON roles (role_name);

INSERT OR REPLACE INTO roles(role_id, role_name) VALUES(1, 'admin');
INSERT OR REPLACE INTO roles(role_id, role_name) VALUES(2, 'user');

CREATE TABLE IF NOT EXISTS users (
    user_id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_login TEXT NOT NULL UNIQUE,
    user_pswd_key TEXT NOT NULL,
    role_id INTEGER NOT NULL,

    FOREIGN KEY(role_id) REFERENCES roles(role_id) ON UPDATE CASCADE
);

CREATE UNIQUE INDEX IF NOT EXISTS users_login_idx ON users (user_login);
CREATE INDEX IF NOT EXISTS users_login_with_keys_idx ON users (user_login, user_pswd_key);

)";

const std::string DatabaseSchemaSql::SQL_CHECK_USER_AND_PASSWORD =
    "select user_id, role_id from users where user_login = ? and user_pswd_key = ?;";

const std::string DatabaseSchemaSql::SQL_INSERT_USER =
    "insert into users (user_login, user_pswd_key, role_id) values (?, ?, ?);";

const std::string DatabaseSchemaSql::SQL_UPDATE_USER_WITHOUT_PASSWORD =
    "update users set user_login = ?, role_id = ? where user_id = ?;";

const std::string DatabaseSchemaSql::SQL_UPDATE_USER =
    "update users set user_login = ?, user_pswd_key = ?, role_id = ? where user_id = ?;";

const std::string DatabaseSchemaSql::SQL_DELETE_USER_BY_ID =
    "delete from users where user_id = ?;";

const std::string DatabaseSchemaSql::SQL_DELETE_USER_BY_LOGIN =
    "delete from users where user_login = ?;";

const std::string DatabaseSchemaSql::SQL_SELECT_ALL_USERS =
    "select user_id, user_login, role_id from users;";

const std::string DatabaseSchemaSql::SQL_SELECT_USERS_COUNT =
    "select count(*) from users;";

```



```

const std::string DatabaseSchemaSql::SQL_SELECT_ADMIN_USERS_COUNT =
    "select count(*) from users where role_id = 1;";

const std::string DatabaseSchemaSql::SQL_SELECT_USER_ID_AND_ROLE_BY_LOGIN =
    "select user_id, role_id from users where user_login = ?;";

const std::string DatabaseSchemaSql::SQL_SELECT_USER_LOGIN_AND_ROLE_BY_ID =
    "select user_login, role_id from users where user_id = ?;";

const std::string DatabaseSchemaSql::SQL_SELECT_USER_ID_BY_LOGIN =
    "select user_id from users where user_login = ?;";

```

## Файл PasswordKeyGenerator.h

```

#pragma once

class PasswordKeyGenerator
{
    static const std::string PBKDF2_SALT;
    static const uint32_t PBKDF2_ITERATIONS_COUNT;
    static const uint32_t PBKDF2_KEY_SIZE;

public:
    static std::string GenerateKey_PBKDF2_HMAC(const std::string& password);
};

```

## Файл PasswordKeyGenerator.cpp

```

#include "stdafx.h"
#include "PasswordKeyGenerator.h"

const std::string PasswordKeyGenerator::PBKDF2_SALT = "N}i82s7kU+W^mcS7hI9ec|0z9A4X+f";
const uint32_t PasswordKeyGenerator::PBKDF2_ITERATIONS_COUNT = 64000;
const uint32_t PasswordKeyGenerator::PBKDF2_KEY_SIZE = 64;

std::string PasswordKeyGenerator::GenerateKey_PBKDF2_HMAC(const std::string& password)
{
    std::vector<uint8_t> res(PBKDF2_KEY_SIZE);

    if (PKCS5_PBKDF2_HMAC(
        password.data(), (int)password.size(),
        (const unsigned char*)PBKDF2_SALT.data(), (int)PBKDF2_SALT.size(),
        PBKDF2_ITERATIONS_COUNT,
        EVP_sha512(),
        PBKDF2_KEY_SIZE,
        res.data()) == 0)
    {
        throw std::exception("PKCS5_PBKDF2_HMAC error");
    }

    BIO* b64 = BIO_new(BIO_f_base64());

    if (b64 == nullptr)
    {
        throw std::bad_alloc();
    }

    BIO* bmem = BIO_new(BIO_s_mem());

    if (bmem == nullptr)
    {
        BIO_free_all(b64);

        throw std::bad_alloc();
    }

    BIO_push(b64, bmem);
    BIO_set_flags(b64, BIO_FLAGS_BASE64_NO_NL);

    BIO_write(b64, res.data(), (int)res.size());

    BIO_flush(b64);

    int len = BIO_pending(bmem);
}

```

```

        std::vector<char> buf(len + 1);

        len = BIO_read(bmem, buf.data(), len);
        buf[len] = '\\0';

        BIO_free_all(b64);

        return std::string(buf.data());
    }

```

## Модуль PatientsRegistry.lib

### Файл stdafx.h

```

#pragma once

#include "targetver.h"

#define WIN32_LEAN_AND_MEAN           // Exclude rarely-used stuff from Windows headers

// stl
#include <string>
#include <vector>
#include <memory>
#include <chrono>
#include <thread>
#include <mutex>
#include <unordered_set>
#include <map>
#include <algorithm>

// sqlite
#include <sqlite3.h>

```

### Файл Gender.h

```

#pragma once

enum class Gender
{
    Unknown,
    Male,
    Feemale,
    Schemale,
};

```

### Файл BirthDate.h

```

#pragma once

class BirthDate
{
public:
    BirthDate();
    explicit BirthDate(uint16_t year, uint8_t month, uint8_t day);
    explicit BirthDate(uint32_t birthDateInt);

    static const BirthDate& GetEmptyBirthDate();
    static const BirthDate GetCurrentDate();

    uint32_t GetBirthDateAsInt() const;

    uint16_t GetYear() const;
    uint8_t GetMonth() const;
    uint8_t GetDay() const;

private:
    uint16_t m_year;
    uint8_t m_month;
    uint8_t m_day;

    // date in integer format (YYYYMMDD)
    uint32_t m_birthDateInt;

```

```
};

bool operator==(const BirthDate& lhs, const BirthDate& rhs);
bool operator!=(const BirthDate& lhs, const BirthDate& rhs);
```

## Файл BirthDate.cpp

```
#include "stdafx.h"
#include "BirthDate.h"

bool operator==(const BirthDate& lhs, const BirthDate& rhs)
{
    return (lhs.GetBirthDateAsInt() == rhs.GetBirthDateAsInt());
};

bool operator!=(const BirthDate& lhs, const BirthDate& rhs)
{
    return (lhs.GetBirthDateAsInt() != rhs.GetBirthDateAsInt());
};

BirthDate::BirthDate() :
    BirthDate(0)
{
}

BirthDate::BirthDate(uint32_t birthDateInt) :
    BirthDate(birthDateInt / 10000, (birthDateInt % 10000) / 100, birthDateInt % 100)
{
}

BirthDate::BirthDate(uint16_t year, uint8_t month, uint8_t day) :
    m_year(year),
    m_month(month),
    m_day(day),
    m_birthDateInt(year * 10000 + month * 100 + day)
{
}

const BirthDate& BirthDate::GetEmptyBirthDate()
{
    static BirthDate emptyBirthDate(0);

    return emptyBirthDate;
}

const BirthDate BirthDate::GetCurrentDate()
{
    auto now = std::chrono::system_clock::now();
    auto inTimeT = std::chrono::system_clock::to_time_t(now);

    tm tmNow;

    if (localtime_s(&tmNow, &inTimeT) != 0)
    {
        throw std::runtime_error("localtime_s error");
    }

    return BirthDate(tmNow.tm_year + 1900, tmNow.tm_mon + 1, tmNow.tm_mday);
}

uint32_t BirthDate::GetBirthDateAsInt() const
{
    return m_birthDateInt;
}

uint16_t BirthDate::GetYear() const
{
    return m_year;
}

uint8_t BirthDate::GetMonth() const
{
    return m_month;
}

uint8_t BirthDate::GetDay() const
```

```
{
    return m_day;
}
```

## Файл Patient.h

```
#pragma once

#include "BirthDate.h"
#include "Gender.h"

class Patient
{
public:
    Patient();
    explicit Patient(const std::string name, const Gender gender, const BirthDate& birthDate, const
std::string& city,
        const std::string& phone, const std::vector<std::string>& diagnoses);

    static const Patient& GetEmptyPatient();

    const std::string& GetName() const;
    const Gender GetGender() const;
    const BirthDate& GetBirthDate() const;
    const std::string& GetCity() const;
    const std::string& GetPhone() const;
    const std::vector<std::string>& GetDiagnoses() const;

    const std::string GetDiagnosesAsString() const;

private:
    std::string m_name;
    Gender m_gender;
    BirthDate m_birthDate;
    std::string m_city;
    std::string m_phone;
    std::vector<std::string> m_diagnoses;
};

using PatientId_t = uint64_t;
using PatientRecord_t = std::pair<PatientId_t, Patient>;

static const PatientId_t PATIENTID_EMPTY = 0;

bool operator==(const Patient& lhs, const Patient& rhs);
bool operator!=(const Patient& lhs, const Patient& rhs);
```

## Файл Patient.cpp

```
#include "stdafx.h"
#include "Patient.h"

bool operator==(const Patient& lhs, const Patient& rhs)
{
    return (lhs.GetName() == rhs.GetName() &&
        lhs.GetGender() == rhs.GetGender() &&
        lhs.GetBirthDate() == rhs.GetBirthDate() &&
        lhs.GetCity() == rhs.GetCity() &&
        lhs.GetPhone() == rhs.GetPhone() &&
        lhs.GetDiagnosesAsString() == rhs.GetDiagnosesAsString());
};

bool operator!=(const Patient& lhs, const Patient& rhs)
{
    return (lhs.GetName() != rhs.GetName() ||
        lhs.GetGender() != rhs.GetGender() ||
        lhs.GetBirthDate() != rhs.GetBirthDate() ||
        lhs.GetCity() != rhs.GetCity() ||
        lhs.GetPhone() != rhs.GetPhone() ||
        lhs.GetDiagnosesAsString() != rhs.GetDiagnosesAsString());
};

Patient::Patient() :
    m_gender(Gender::Unknown),
    m_birthDate(BirthDate::GetEmptyBirthDate())
```

```

{
}

Patient::Patient(const std::string name, const Gender gender, const BirthDate& birthDate, const
std::string& city,
    const std::string& phone, const std::vector<std::string>& diagnoses)
:
    m_name(name),
    m_gender(gender),
    m_birthDate(birthDate),
    m_city(city),
    m_phone(phone),
    m_diagnoses(diagnoses)
{
}

const Patient& Patient::GetEmptyPatient()
{
    static Patient emptyPatient;

    return emptyPatient;
}

const std::string& Patient::GetName() const
{
    return m_name;
}

const Gender Patient::GetGender() const
{
    return m_gender;
}

const BirthDate& Patient::GetBirthDate() const
{
    return m_birthDate;
}

const std::string& Patient::GetCity() const
{
    return m_city;
}

const std::string& Patient::GetPhone() const
{
    return m_phone;
}

const std::vector<std::string>& Patient::GetDiagnoses() const
{
    return m_diagnoses;
}

const std::string Patient::GetDiagnosesAsString() const
{
    std::string diagnosesAsString;

    for (const auto& i : m_diagnoses)
    {
        if (!diagnosesAsString.empty())
        {
            diagnosesAsString += ", ";
        }

        diagnosesAsString += i;
    }

    return diagnosesAsString;
}

```

## Файл PatientsRegistry.h

```

#pragma once

#include "Patient.h"

```

```

class PatientsRegistryImpl;

class PatientsRegistry
{
public:
    static std::shared_ptr<PatientsRegistry> CreatePatientsRegistry(const std::string& connectionString);
    static std::shared_ptr<PatientsRegistry> OpenPatientsRegistry(const std::string& connectionString,
        bool readOnly);

    PatientsRegistry() = delete;
    PatientsRegistry(const PatientsRegistry&) = delete;
    PatientsRegistry& operator = (const PatientsRegistry&) = delete;

private:
    explicit PatientsRegistry(const std::string& connectionString, bool createIfNotExist, bool readOnly);

public:
    const std::string& GetConnectionString() const;

    const std::vector<PatientRecord_t> GetAllPatients() const;
    void AddPatient(const Patient& patient);
    bool UpdatePatient(const PatientId_t patientId, const Patient& patient);
    bool DeletePatient(const PatientId_t patientId);
    PatientRecord_t GetPatientById(const PatientId_t patientId);

    const std::vector<PatientRecord_t> SearchPatientsByName(const std::string& name) const;
    const std::vector<PatientRecord_t> SearchPatientsByCity(const std::string& city) const;
    const std::vector<PatientRecord_t> SearchPatientsByDiagnose(const std::string& diagnose) const;
    const std::vector<PatientRecord_t> SearchPatientsByPhone(const std::string& phone) const;

    const std::vector<PatientRecord_t> OrderPatientsByName() const;
    const std::vector<PatientRecord_t> OrderPatientsByCity() const;
    const std::vector<PatientRecord_t> OrderPatientsByBirthDate() const;

    const std::vector<PatientRecord_t> ViewNonresidentPatients(const std::string& city) const;
    const std::vector<PatientRecord_t> ViewPatientsByAgeAndDiagnose(uint16_t age, const std::string&
        diagnose) const;

private:
    std::unique_ptr<PatientsRegistryImpl> m_patientsRegistryImpl;
};

```

## Файл PatientsRegistry.cpp

```

#include "stdafx.h"
#include "PatientsRegistry.h"
#include "PatientsRegistryImpl.h"

PatientsRegistry::PatientsRegistry(const std::string& connectionString, bool createIfNotExist, bool
    readOnly) :
    m_patientsRegistryImpl(std::make_unique<PatientsRegistryImpl>(connectionString, createIfNotExist,
        readOnly))
{
}

std::shared_ptr<PatientsRegistry> PatientsRegistry::CreatePatientsRegistry(const std::string&
    connectionString)
{
    return std::shared_ptr<PatientsRegistry>(new PatientsRegistry(connectionString, true, false));
}

std::shared_ptr<PatientsRegistry> PatientsRegistry::OpenPatientsRegistry(const std::string&
    connectionString, bool readOnly)
{
    return std::shared_ptr<PatientsRegistry>(new PatientsRegistry(connectionString, false, readOnly));
}

const std::string& PatientsRegistry::GetConnectionString() const
{
    return m_patientsRegistryImpl->GetConnectionString();
}

const std::vector<PatientRecord_t> PatientsRegistry::GetAllPatients() const
{
    return m_patientsRegistryImpl->GetAllPatients();
}

```

```

}

void PatientsRegistry::AddPatient(const Patient& patient)
{
    m_patientsRegistryImpl->AddPatient(patient);
}

bool PatientsRegistry::UpdatePatient(const PatientId_t patientId, const Patient& patient)
{
    return m_patientsRegistryImpl->UpdatePatient(patientId, patient);
}

bool PatientsRegistry::DeletePatient(const PatientId_t patientId)
{
    return m_patientsRegistryImpl->DeletePatient(patientId);
}

PatientRecord_t PatientsRegistry::GetPatientById(const PatientId_t patientId)
{
    return m_patientsRegistryImpl->GetPatientById(patientId);
}

const std::vector<PatientRecord_t> PatientsRegistry::SearchPatientsByName(const std::string& name) const
{
    return m_patientsRegistryImpl->SearchPatientsByName(name);
}

const std::vector<PatientRecord_t> PatientsRegistry::SearchPatientsByCity(const std::string& city) const
{
    return m_patientsRegistryImpl->SearchPatientsByCity(city);
}

const std::vector<PatientRecord_t> PatientsRegistry::SearchPatientsByDiagnose(const std::string& diagnose) const
{
    return m_patientsRegistryImpl->SearchPatientsByDiagnose(diagnose);
}

const std::vector<PatientRecord_t> PatientsRegistry::SearchPatientsByPhone(const std::string& phone) const
{
    return m_patientsRegistryImpl->SearchPatientsByPhone(phone);
}

const std::vector<PatientRecord_t> PatientsRegistry::OrderPatientsByName() const
{
    return m_patientsRegistryImpl->OrderPatientsByName();
}

const std::vector<PatientRecord_t> PatientsRegistry::OrderPatientsByCity() const
{
    return m_patientsRegistryImpl->OrderPatientsByCity();
}

const std::vector<PatientRecord_t> PatientsRegistry::OrderPatientsByBirthDate() const
{
    return m_patientsRegistryImpl->OrderPatientsByBirthDate();
}

const std::vector<PatientRecord_t> PatientsRegistry::ViewNonresidentPatients(const std::string& city) const
{
    return m_patientsRegistryImpl->ViewNonresidentPatients(city);
}

const std::vector<PatientRecord_t> PatientsRegistry::ViewPatientsByAgeAndDiagnose(uint16_t age,
const std::string& diagnose) const
{
    return m_patientsRegistryImpl->ViewPatientsByAgeAndDiagnose(age, diagnose);
}

```

## Файл PatientsRegistryImpl.h

```

#pragma once

#include "Patient.h"
#include "../DatabaseSQLite/DatabaseSQLite.h"

```

```

class PatientsRegistryImpl
{
public:
    explicit PatientsRegistryImpl(const std::string& connectionString, bool createIfNotExist, bool
readOnly);

    PatientsRegistryImpl() = delete;
    PatientsRegistryImpl(const PatientsRegistryImpl&) = delete;
    PatientsRegistryImpl& operator = (const PatientsRegistryImpl&) = delete;

public:
    const std::vector<PatientRecord_t> GetAllPatients() const;
    void AddPatient(const Patient& patient);
    bool UpdatePatient(const PatientId_t patientId, const Patient& patient);
    bool DeletePatient(const PatientId_t patientId);
    PatientRecord_t GetPatientById(const PatientId_t patientId);
    const std::string& GetConnectionString() const;

    const std::vector<PatientRecord_t> SearchPatientsByName(const std::string& name) const;
    const std::vector<PatientRecord_t> SearchPatientsByCity(const std::string& city) const;
    const std::vector<PatientRecord_t> SearchPatientsByDiagnose(const std::string& diagnose) const;
    const std::vector<PatientRecord_t> SearchPatientsByPhone(const std::string& phone) const;

    const std::vector<PatientRecord_t> OrderPatientsByName() const;
    const std::vector<PatientRecord_t> OrderPatientsByCity() const;
    const std::vector<PatientRecord_t> OrderPatientsByBirthDate() const;

    const std::vector<PatientRecord_t> ViewNonresidentPatients(const std::string& city) const;
    const std::vector<PatientRecord_t> ViewPatientsByAgeAndDiagnose(uint16_t age, const std::string&
diagnose) const;

private:
    Gender GetGenderById(const uint32_t genderId) const;
    uint32_t GetGenderId(const Gender gender);

    const std::string GetCityById(const uint64_t cityId) const;
    uint64_t GetCityId(const std::string& city);

    const std::vector<std::string> GetDiagnosesOfPatient(const uint64_t patientId) const;
    void SetDiagnosesOfPatient(const uint64_t patientId, const std::vector<std::string>& diagnoses);
    void DeleteDiagnosesOfPatient(const uint64_t patientId);

private:
    void ValidateDatabaseSQLiteSchema();
    void PrepareSqlStatements();

    const std::vector<PatientRecord_t> GetPatientsFromStatement(sqlite3_stmt* statementPtr) const;

private:
    mutable std::mutex m_mutex;

    DatabaseSQLite m_sqliteDatabase;

    SQLiteStatement_t m_insertPatientStatement;
    SQLiteStatement_t m_deletePatientStatement;
    SQLiteStatement_t m_selectPatientStatement;
    SQLiteStatement_t m_selectAllPatientsStatement;
    SQLiteStatement_t m_selectAllPatientsOrderByNameStatement;
    SQLiteStatement_t m_selectAllPatientsOrderByBirthDateStatement;
    SQLiteStatement_t m_selectAllPatientsOrderByCityStatement;
    SQLiteStatement_t m_updatePatientStatement;
    SQLiteStatement_t m_insertDiagnosisStatement;
    SQLiteStatement_t m_selectDiagnosisStatement;
    SQLiteStatement_t m_insertDiagnosisOfPatientStatement;
    SQLiteStatement_t m_selectDiagnosesOfPatientStatement;
    SQLiteStatement_t m_deleteDiagnosesOfPatientStatement;
    SQLiteStatement_t m_insertCityStatement;
    SQLiteStatement_t m_selectCityByNameStatement;
    SQLiteStatement_t m_selectCityByIdStatement;

    SQLiteStatement_t m_searchPatientsByNameStatement;
    SQLiteStatement_t m_searchPatientsByPhoneStatement;
    SQLiteStatement_t m_searchPatientsByCityStatement;
    SQLiteStatement_t m_searchPatientsByDiagnoseStatement;

    SQLiteStatement_t m_searchPatientsByCityNonresidentStatement;

```



```

        SQLiteStatement_t m_searchPatientsByAgeAndDiagnoseStatement;
    };

```

## Файл PatientsRegistryImpl.cpp

```

#include "stdafx.h"
#include "PatientsRegistryImpl.h"
#include "../DatabaseSQLite/DatabaseSQLiteException.h"
#include "DatabaseSchemaSql.h"

PatientsRegistryImpl::PatientsRegistryImpl(const std::string& connectionString, bool createIfNotExist,
bool readOnly) :
    m_sqliteDatabase(connectionString, createIfNotExist, readOnly)
{
    if (!readOnly)
    {
        ValidateDatabaseSQLiteSchema();
    }

    PrepareSqlStatements();
}

void PatientsRegistryImpl::PrepareSqlStatements()
{
    m_insertPatientStatement = m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_INSERT_PATIENT);
    m_updatePatientStatement = m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_UPDATE_PATIENT);
    m_deletePatientStatement = m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_DELETE_PATIENT);
    m_selectPatientStatement =
m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_SELECT_PATIENT_BY_ID);
    m_selectAllPatientsStatement =
m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_SELECT_ALLS_PATIENTS);

    m_selectAllPatientsOrderByNameStatement =
        m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_SELECT_ALLS_PATIENTS_ORDER_BY_NAME);
    m_selectAllPatientsOrderByBirthDateStatement =
        m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_SELECT_ALLS_PATIENTS_ORDER_BY_BIRTHDATE);
    m_selectAllPatientsOrderByCityStatement =
        m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_SELECT_ALLS_PATIENTS_ORDER_BY_CITY);

    m_selectDiagnosesOfPatientStatement =
        m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_SELECT_DIAGNOSES_BY_PATIENT_ID);

    m_insertDiagnosisStatement = m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_INSERT_DIAGNOSE);
    m_selectDiagnosisStatement =
m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_SELECT_DIAGNOSE_ID_BY_NAME);
    m_insertDiagnosisOfPatientStatement =
m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_INSERT_DIAGNOSE_OF_PATIENT);
    m_deleteDiagnosesOfPatientStatement =
m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_DELETE_DIAGNOSES_OF_PATIENT);

    m_insertCityStatement = m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_INSERT_CITY);
    m_selectCityByNameStatement =
m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_SELECT_CITY_ID_BY_NAME);
    m_selectCityByIdStatement =
m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_SELECT_CITY_NAME_BY_ID);

    m_searchPatientsByNameStatement =
m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_SELECT_PATIENTS_BY_NAME);
    m_searchPatientsByPhoneStatement =
m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_SELECT_PATIENTS_BY_PHONE);
    m_searchPatientsByCityStatement =
m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_SELECT_PATIENTS_BY_CITY_NAME);

    m_searchPatientsByDiagnoseStatement =
        m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_SELECT_PATIENTS_BY_DIAGNOSE_NAME);

    m_searchPatientsByCityNonresidentStatement =
        m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_SELECT_PATIENTS_BY_NOT_CITY_NAME);

    m_searchPatientsByAgeAndDiagnoseStatement =
        m_sqliteDatabase.CreateStatement(DatabaseSchemaSql::SQL_SELECT_PATIENTS_BY_AGE_AND_DIAGNOSE_NAME);
}

const std::string& PatientsRegistryImpl::GetConnectionString() const
{

```

```

        return m_sqliteDatabase.GetConnectionString();
    }

void PatientsRegistryImpl::AddPatient(const Patient& patient)
{
    std::lock_guard<std::mutex> lock(m_mutex);

    m_sqliteDatabase.BeginTransaction();

    try
    {
        sqlite3_reset(m_insertPatientStatement.get());

        sqlite3_bind_text(m_insertPatientStatement.get(), 1, patient.GetName().c_str(),
(int)patient.GetName().size(), SQLITE_STATIC);
        sqlite3_bind_int(m_insertPatientStatement.get(), 2, patient.GetBirthDate().GetBirthDateAsInt());
        sqlite3_bind_int64(m_insertPatientStatement.get(), 3, GetGenderId(patient.GetGender()));
        sqlite3_bind_int64(m_insertPatientStatement.get(), 4, GetCityId(patient.GetCity()));
        sqlite3_bind_text(m_insertPatientStatement.get(), 5, patient.GetPhone().c_str(),
(int)patient.GetPhone().size(), SQLITE_STATIC);

        int sqliteStepResult = sqlite3_step(m_insertPatientStatement.get());

        if (sqliteStepResult != SQLITE_DONE)
        {
            const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

            throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
                sqliteStepResult);
        }

        uint64_t patientId = sqlite3_last_insert_rowid(m_sqliteDatabase.GetSqliteConnectionPtr());

        SetDiagnosesOfPatient(patientId, patient.GetDiagnoses());
    }
    catch (...)
    {
        m_sqliteDatabase.RollbackTransaction();

        throw;
    }

    m_sqliteDatabase.CommitTransaction();
}

bool PatientsRegistryImpl::UpdatePatient(const PatientId_t patientId, const Patient& patient)
{
    std::lock_guard<std::mutex> lock(m_mutex);

    m_sqliteDatabase.BeginTransaction();

    try
    {
        sqlite3_reset(m_updatePatientStatement.get());

        sqlite3_bind_text(m_updatePatientStatement.get(), 1, patient.GetName().c_str(),
(int)patient.GetName().size(), SQLITE_STATIC);
        sqlite3_bind_int(m_updatePatientStatement.get(), 2, patient.GetBirthDate().GetBirthDateAsInt());
        sqlite3_bind_int64(m_updatePatientStatement.get(), 3, GetGenderId(patient.GetGender()));
        sqlite3_bind_int64(m_updatePatientStatement.get(), 4, GetCityId(patient.GetCity()));
        sqlite3_bind_text(m_updatePatientStatement.get(), 5, patient.GetPhone().c_str(),
(int)patient.GetPhone().size(), SQLITE_STATIC);
        sqlite3_bind_int64(m_updatePatientStatement.get(), 6, patientId);

        int sqliteStepResult = sqlite3_step(m_updatePatientStatement.get());

        if (sqliteStepResult != SQLITE_DONE)
        {
            const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

            throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
                sqliteStepResult);
        }

        if (sqlite3_changes(m_sqliteDatabase.GetSqliteConnectionPtr()) == 0)
        {
            m_sqliteDatabase.RollbackTransaction();
        }
    }
}

```

```

        return false;
    }

    DeleteDiagnosesOfPatient(patientId);
    SetDiagnosesOfPatient(patientId, patient.GetDiagnoses());
}
catch (...)
{
    m_sqliteDatabase.RollbackTransaction();

    throw;
}

m_sqliteDatabase.CommitTransaction();

return true;
}

bool PatientsRegistryImpl::DeletePatient(const PatientId_t patientId)
{
    std::lock_guard<std::mutex> lock(m_mutex);

    sqlite3_reset(m_deletePatientStatement.get());
    sqlite3_bind_int64(m_deletePatientStatement.get(), 1, patientId);

    int sqliteStepResult = sqlite3_step(m_deletePatientStatement.get());

    if (sqliteStepResult == SQLITE_DONE)
    {
        if (sqlite3_changes(m_sqliteDatabase.GetSqliteConnectionPtr()) == 0)
        {
            return false;
        }
        else
        {
            return true;
        }
    }
    else
    {
        const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

        throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
            sqliteStepResult);
    }
}

const std::vector<PatientRecord_t> PatientsRegistryImpl::GetAllPatients() const
{
    std::lock_guard<std::mutex> lock(m_mutex);

    sqlite3_reset(m_selectAllPatientsStatement.get());

    return GetPatientsFromStatement(m_selectAllPatientsStatement.get());
}

const std::vector<PatientRecord_t> PatientsRegistryImpl::GetPatientsFromStatement(sqlite3_stmt*
statementPtr) const
{
    std::vector<PatientRecord_t> patientsRecords;

    int sqliteStepResult = sqlite3_step(statementPtr);

    while (sqliteStepResult == SQLITE_ROW)
    {
        const uint64_t patientId = sqlite3_column_int64(statementPtr, 0);
        const std::string name = (const char*)sqlite3_column_text(statementPtr, 1);
        const uint32_t birthDay = (uint32_t)sqlite3_column_int(statementPtr, 2);
        const uint32_t genderId = (uint32_t)sqlite3_column_int(statementPtr, 3);
        const uint32_t cityId = (uint32_t)sqlite3_column_int(statementPtr, 4);
        const std::string phone = (const char*)sqlite3_column_text(statementPtr, 5);

        patientsRecords.emplace_back(std::make_pair(patientId, Patient{
            name,
            GetGenderById(genderId),
            BirthDate{ birthDay },

```

```

        GetCityById(cityId),
        phone,
        GetDiagnosesOfPatient(patientId)
    )));

    sqliteStepResult = sqlite3_step(statementPtr);
}

if (sqliteStepResult != SQLITE_DONE)
{
    const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

    throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
        sqliteStepResult);
}

return patientsRecords;
}

PatientRecord_t PatientsRegistryImpl::GetPatientById(const PatientId_t patientId)
{
    if (patientId == PATIENTID_EMPTY)
    {
        return std::make_pair(patientId, Patient::GetEmptyPatient());
    }

    std::lock_guard<std::mutex> lock(m_mutex);

    sqlite3_reset(m_selectPatientStatement.get());
    sqlite3_bind_int64(m_selectPatientStatement.get(), 1, patientId);

    const std::vector<PatientRecord_t> patientsRecords =
        GetPatientsFromStatement(m_selectPatientStatement.get());

    if (patientsRecords.empty())
    {
        return std::make_pair(PATIENTID_EMPTY, Patient::GetEmptyPatient());
    }
    else
    {
        return patientsRecords.front();
    }
}

const std::vector<PatientRecord_t> PatientsRegistryImpl::SearchPatientsByName(const std::string& name)
const
{
    std::lock_guard<std::mutex> lock(m_mutex);

    sqlite3_reset(m_searchPatientsByNameStatement.get());
    sqlite3_bind_text(m_searchPatientsByNameStatement.get(), 1, name.c_str(), (int)name.size(),
        SQLITE_STATIC);

    return GetPatientsFromStatement(m_searchPatientsByNameStatement.get());
}

const std::vector<PatientRecord_t> PatientsRegistryImpl::SearchPatientsByCity(const std::string& city)
const
{
    std::lock_guard<std::mutex> lock(m_mutex);

    sqlite3_reset(m_searchPatientsByCityStatement.get());
    sqlite3_bind_text(m_searchPatientsByCityStatement.get(), 1, city.c_str(), (int)city.size(),
        SQLITE_STATIC);

    return GetPatientsFromStatement(m_searchPatientsByCityStatement.get());
}

const std::vector<PatientRecord_t> PatientsRegistryImpl::SearchPatientsByDiagnose(const std::string&
diagnose) const
{
    std::lock_guard<std::mutex> lock(m_mutex);

    sqlite3_reset(m_searchPatientsByDiagnoseStatement.get());
    sqlite3_bind_text(m_searchPatientsByDiagnoseStatement.get(), 1, diagnose.c_str(),
        (int)diagnose.size(), SQLITE_STATIC);

```

```

        return GetPatientsFromStatement(m_searchPatientsByDiagnoseStatement.get());
    }

const std::vector<PatientRecord_t> PatientsRegistryImpl::SearchPatientsByPhone(const std::string& phone)
const
{
    std::lock_guard<std::mutex> lock(m_mutex);

    sqlite3_reset(m_searchPatientsByPhoneStatement.get());
    sqlite3_bind_text(m_searchPatientsByPhoneStatement.get(), 1, phone.c_str(), (int)phone.size(),
        SQLITE_STATIC);

    return GetPatientsFromStatement(m_searchPatientsByPhoneStatement.get());
}

const std::vector<PatientRecord_t> PatientsRegistryImpl::OrderPatientsByName() const
{
    std::lock_guard<std::mutex> lock(m_mutex);

    sqlite3_reset(m_selectAllPatientsOrderByNameStatement.get());

    return GetPatientsFromStatement(m_selectAllPatientsOrderByNameStatement.get());
}

const std::vector<PatientRecord_t> PatientsRegistryImpl::OrderPatientsByCity() const
{
    std::lock_guard<std::mutex> lock(m_mutex);

    sqlite3_reset(m_selectAllPatientsOrderByCityStatement.get());

    return GetPatientsFromStatement(m_selectAllPatientsOrderByCityStatement.get());
}

const std::vector<PatientRecord_t> PatientsRegistryImpl::OrderPatientsByBirthDate() const
{
    std::lock_guard<std::mutex> lock(m_mutex);

    sqlite3_reset(m_selectAllPatientsOrderByBirthDateStatement.get());

    return GetPatientsFromStatement(m_selectAllPatientsOrderByBirthDateStatement.get());
}

const std::vector<PatientRecord_t> PatientsRegistryImpl::ViewNonresidentPatients(const std::string& city)
const
{
    std::lock_guard<std::mutex> lock(m_mutex);

    sqlite3_reset(m_searchPatientsByCityNonresidentStatement.get());
    sqlite3_bind_text(m_searchPatientsByCityNonresidentStatement.get(), 1, city.c_str(), (int)city.size(),
        SQLITE_STATIC);

    return GetPatientsFromStatement(m_searchPatientsByCityNonresidentStatement.get());
}

const std::vector<PatientRecord_t> PatientsRegistryImpl::ViewPatientsByAgeAndDiagnose(uint16_t age,
const std::string& diagnose) const
{
    BirthDate birthDateToday(BirthDate::GetCurrentDate());
    BirthDate birthDateMore(birthDateToday.GetYear() - age, birthDateToday.GetMonth(),
    birthDateToday.GetDay());
    int birthDateMoreInt = static_cast<int>(birthDateMore.GetBirthDateAsInt());

    std::lock_guard<std::mutex> lock(m_mutex);

    sqlite3_reset(m_searchPatientsByAgeAndDiagnoseStatement.get());
    sqlite3_bind_int(m_searchPatientsByAgeAndDiagnoseStatement.get(), 1, birthDateMoreInt);
    sqlite3_bind_text(m_searchPatientsByAgeAndDiagnoseStatement.get(), 2, diagnose.c_str(),
    (int)diagnose.size(), SQLITE_STATIC);

    return GetPatientsFromStatement(m_searchPatientsByAgeAndDiagnoseStatement.get());
}

const std::vector<std::string> PatientsRegistryImpl::GetDiagnosesOfPatient(const uint64_t patientId) const
{
    std::vector<std::string> patientDiagnoses;

    sqlite3_reset(m_selectDiagnosesOfPatientStatement.get());

```

```

sqlite3_bind_int64(m_selectDiagnosesOfPatientStatement.get(), 1, patientId);

int sqliteStepResult = sqlite3_step(m_selectDiagnosesOfPatientStatement.get());

while (sqliteStepResult == SQLITE_ROW)
{
    const std::string name = (const
char*)sqlite3_column_text(m_selectDiagnosesOfPatientStatement.get(), 0);
    patientDiagnoses.emplace_back(name);

    sqliteStepResult = sqlite3_step(m_selectDiagnosesOfPatientStatement.get());
}

if (sqliteStepResult != SQLITE_DONE)
{
    const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

    throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
        sqliteStepResult);
}

return patientDiagnoses;
}

void PatientsRegistryImpl::SetDiagnosesOfPatient(const uint64_t patientId, const std::vector<std::string>&
diagnoses)
{
    std::unordered_set<uint64_t> diagnosesIds;

    for (const auto& diagnosisName : diagnoses)
    {
        sqlite3_reset(m_selectDiagnosisStatement.get());
        sqlite3_bind_text(m_selectDiagnosisStatement.get(), 1, diagnosisName.c_str(),
(int)diagnosisName.size(), SQLITE_STATIC);

        int sqliteStepResult = sqlite3_step(m_selectDiagnosisStatement.get());

        if (sqliteStepResult == SQLITE_ROW)
        {
            const uint64_t diagnosisId = sqlite3_column_int64(m_selectDiagnosisStatement.get(), 0);
            diagnosesIds.insert(diagnosisId);
        }
        else if (sqliteStepResult == SQLITE_DONE)
        {
            sqlite3_reset(m_insertDiagnosisStatement.get());
            sqlite3_bind_text(m_insertDiagnosisStatement.get(), 1, diagnosisName.c_str(),
(int)diagnosisName.size(), SQLITE_STATIC);

            int sqliteInsertStepResult = sqlite3_step(m_insertDiagnosisStatement.get());

            if (sqliteInsertStepResult != SQLITE_DONE)
            {
                const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

                throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
                    sqliteStepResult);
            }

            const uint64_t diagnosisId =
sqlite3_last_insert_rowid(m_sqliteDatabase.GetSqliteConnectionPtr());
            diagnosesIds.insert(diagnosisId);
        }
        else
        {
            const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

            throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
                sqliteStepResult);
        }
    }

    for (const auto diagnoseId : diagnosesIds)
    {
        sqlite3_reset(m_insertDiagnosisOfPatientStatement.get());
        sqlite3_bind_int64(m_insertDiagnosisOfPatientStatement.get(), 1, patientId);
        sqlite3_bind_int64(m_insertDiagnosisOfPatientStatement.get(), 2, diagnoseId);
    }
}

```

```

        int sqliteStepResult = sqlite3_step(m_insertDiagnosisOfPatientStatement.get());

        if (sqliteStepResult != SQLITE_DONE)
        {
            const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

            throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
                sqliteStepResult);
        }
    }
}

void PatientsRegistryImpl::DeleteDiagnosesOfPatient(const uint64_t patientId)
{
    sqlite3_reset(m_deleteDiagnosesOfPatientStatement.get());
    sqlite3_bind_int64(m_deleteDiagnosesOfPatientStatement.get(), 1, patientId);

    int sqliteStepResult = sqlite3_step(m_deleteDiagnosesOfPatientStatement.get());

    if (sqliteStepResult == SQLITE_DONE)
    {
        return;
    }
    else
    {
        const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

        throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
            sqliteStepResult);
    }
}

const std::string PatientsRegistryImpl::GetCityById(const uint64_t cityId) const
{
    sqlite3_reset(m_selectCityByIdStatement.get());
    sqlite3_bind_int64(m_selectCityByIdStatement.get(), 1, cityId);

    int sqliteStepResult = sqlite3_step(m_selectCityByIdStatement.get());

    if (sqliteStepResult != SQLITE_ROW)
    {
        const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

        throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
            sqliteStepResult);
    }

    const std::string name = (const char*)sqlite3_column_text(m_selectCityByIdStatement.get(), 0);

    return name;
}

uint64_t PatientsRegistryImpl::GetCityId(const std::string& city)
{
    uint64_t cityId;

    sqlite3_reset(m_selectCityByNameStatement.get());
    sqlite3_bind_text(m_selectCityByNameStatement.get(), 1, city.c_str(), (int)city.size(),
        SQLITE_STATIC);

    int sqliteStepResult = sqlite3_step(m_selectCityByNameStatement.get());

    if (sqliteStepResult == SQLITE_ROW)
    {
        cityId = sqlite3_column_int64(m_selectCityByNameStatement.get(), 0);
    }
    else if (sqliteStepResult == SQLITE_DONE)
    {
        sqlite3_reset(m_insertCityStatement.get());
        sqlite3_bind_text(m_insertCityStatement.get(), 1, city.c_str(), (int)city.size(), SQLITE_STATIC);

        int sqliteInsertStepResult = sqlite3_step(m_insertCityStatement.get());

        if (sqliteInsertStepResult != SQLITE_DONE)
        {
            const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

```

```

        throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
                                       sqliteStepResult);
    }

    cityId = sqlite3_last_insert_rowid(m_sqliteDatabase.GetSqliteConnectionPtr());
}
else
{
    const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteDatabase.GetSqliteConnectionPtr());

    throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
                                   sqliteStepResult);
}

return cityId;
}

Gender PatientsRegistryImpl::GetGenderById(const uint32_t genderId) const
{
    return DatabaseSchemaSql::DATABASE_SQLITE_GENDERS_BIND_TABLE.at(genderId);
}

uint32_t PatientsRegistryImpl::GetGenderId(const Gender gender)
{
    const auto& bindMap = DatabaseSchemaSql::DATABASE_SQLITE_GENDERS_BIND_TABLE;

    auto it = std::find_if(bindMap.cbegin(), bindMap.cend(),
                           [&gender](const std::pair<uint32_t, Gender>& t) -> bool {
                               return (t.second == gender);
                           });

    if (it == bindMap.cend())
    {
        throw std::invalid_argument("Invalid gender value");
    }

    return it->first;
}

void PatientsRegistryImpl::ValidateDatabaseSQLiteSchema()
{
    m_sqliteDatabase.BeginTransaction();

    try
    {
        m_sqliteDatabase.ExecuteSql(DatabaseSchemaSql::PATIENTS_DATABASE_SQLITE_SCHEMA);
    }
    catch (...)
    {
        m_sqliteDatabase.RollbackTransaction();

        throw;
    }

    m_sqliteDatabase.CommitTransaction();
}

```

## Файл DatabaseSchemaSql.h

```

#pragma once

#include "Gender.h"

struct DatabaseSchemaSql
{
    static const std::map<uint32_t, Gender> DATABASE_SQLITE_GENDERS_BIND_TABLE;

    static const std::string PATIENTS_DATABASE_SQLITE_SCHEMA;

    static const std::string DatabaseSchemaSql::SQL_INSERT_PATIENT;
    static const std::string DatabaseSchemaSql::SQL_UPDATE_PATIENT;
    static const std::string DatabaseSchemaSql::SQL_DELETE_PATIENT;
    static const std::string DatabaseSchemaSql::SQL_SELECT_PATIENT_BY_ID;
    static const std::string DatabaseSchemaSql::SQL_SELECT_ALLS_PATIENTS;
    static const std::string DatabaseSchemaSql::SQL_SELECT_ALLS_PATIENTS_ORDER_BY_NAME;
    static const std::string DatabaseSchemaSql::SQL_SELECT_ALLS_PATIENTS_ORDER_BY_BIRTHDATE;
}

```



```

static const std::string DatabaseSchemaSql::SQL_SELECT_ALLS_PATIENTS_ORDER_BY_CITY;
static const std::string DatabaseSchemaSql::SQL_SELECT_DIAGNOSES_BY_PATIENT_ID;
static const std::string DatabaseSchemaSql::SQL_INSERT_DIAGNOSE;
static const std::string DatabaseSchemaSql::SQL_SELECT_DIAGNOSE_ID_BY_NAME;
static const std::string DatabaseSchemaSql::SQL_INSERT_DIAGNOSE_OF_PATIENT;
static const std::string DatabaseSchemaSql::SQL_DELETE_DIAGNOSES_OF_PATIENT;
static const std::string DatabaseSchemaSql::SQL_INSERT_CITY;
static const std::string DatabaseSchemaSql::SQL_SELECT_CITY_ID_BY_NAME;
static const std::string DatabaseSchemaSql::SQL_SELECT_CITY_NAME_BY_ID;
static const std::string DatabaseSchemaSql::SQL_SELECT_PATIENTS_BY_NAME;
static const std::string DatabaseSchemaSql::SQL_SELECT_PATIENTS_BY_PHONE;
static const std::string DatabaseSchemaSql::SQL_SELECT_PATIENTS_BY_CITY_NAME;
static const std::string DatabaseSchemaSql::SQL_SELECT_PATIENTS_BY_DIAGNOSE_NAME;
static const std::string DatabaseSchemaSql::SQL_SELECT_PATIENTS_BY_NOT_CITY_NAME;
static const std::string DatabaseSchemaSql::SQL_SELECT_PATIENTS_BY_AGE_AND_DIAGNOSE_NAME;
};

```

## Файл DatabaseSchemaSql.cpp

```

#include "stdafx.h"
#include "DatabaseSchemaSql.h"

const std::map<uint32_t, Gender> DatabaseSchemaSql::DATABASE_SQLITE_GENDERS_BIND_TABLE = {
    { 1, Gender::Male },
    { 2, Gender::Feemale },
    { 3, Gender::Schemale },
};

const std::string DatabaseSchemaSql::PATIENTS_DATABASE_SQLITE_SCHEMA = R"(
CREATE TABLE IF NOT EXISTS genders (
    gender_id INTEGER PRIMARY KEY NOT NULL,
    gender_name TEXT COLLATE NOCASE NOT NULL UNIQUE
);

CREATE UNIQUE INDEX IF NOT EXISTS genders_name_idx ON genders (gender_name);

INSERT OR REPLACE INTO genders(gender_id, gender_name) VALUES(1, 'male');
INSERT OR REPLACE INTO genders(gender_id, gender_name) VALUES(2, 'female');
INSERT OR REPLACE INTO genders(gender_id, gender_name) VALUES(3, 'shemale');

CREATE TABLE IF NOT EXISTS cities (
    city_id INTEGER PRIMARY KEY NOT NULL,
    city_name TEXT COLLATE NOCASE NOT NULL UNIQUE
);

CREATE UNIQUE INDEX IF NOT EXISTS cities_name_idx ON cities (city_name);

CREATE TABLE IF NOT EXISTS patients (
    patient_id INTEGER PRIMARY KEY AUTOINCREMENT,
    patient_name TEXT COLLATE NOCASE NOT NULL,
    patient_birth_day INTEGER NOT NULL,
    gender_id INTEGER NOT NULL,
    city_id INTEGER NOT NULL,
    patient_phone TEXT COLLATE NOCASE DEFAULT NULL,

    FOREIGN KEY(gender_id) REFERENCES genders(gender_id) ON UPDATE CASCADE,
    FOREIGN KEY(city_id) REFERENCES cities(city_id) ON UPDATE CASCADE
);

CREATE INDEX IF NOT EXISTS patients_name_idx ON patients (patient_name);
CREATE INDEX IF NOT EXISTS patients_birth_day_idx ON patients (patient_birth_day);
CREATE INDEX IF NOT EXISTS patients_phone_idx ON patients (patient_phone);

CREATE TABLE IF NOT EXISTS diagnoses (
    diagnosis_id INTEGER PRIMARY KEY AUTOINCREMENT,
    diagnosis_name TEXT COLLATE NOCASE NOT NULL UNIQUE
);

CREATE UNIQUE INDEX IF NOT EXISTS diagnoses_name_idx ON diagnoses (diagnosis_name);

CREATE TABLE IF NOT EXISTS patients_diagnoses (
    patient_diagnosis_id INTEGER PRIMARY KEY AUTOINCREMENT,
    patient_id INTEGER NOT NULL,
    diagnosis_id INTEGER NOT NULL,

```

```

        FOREIGN KEY(patient_id) REFERENCES patients(patient_id) ON UPDATE CASCADE ON DELETE CASCADE,
        FOREIGN KEY(diagnosis_id) REFERENCES diagnoses(diagnosis_id) ON UPDATE CASCADE ON DELETE CASCADE
    );

    ");

const std::string DatabaseSchemaSql::SQL_INSERT_PATIENT =
    "insert into patients (patient_name, patient_birth_day, gender_id, city_id, patient_phone) "
    "values (?, ?, ?, ?, ?);";

const std::string DatabaseSchemaSql::SQL_UPDATE_PATIENT =
    "update patients set patient_name = ?, patient_birth_day = ?, gender_id = ?, city_id = ?,
    patient_phone = ? "
    "where patient_id = ?;";

const std::string DatabaseSchemaSql::SQL_DELETE_PATIENT =
    "delete from patients where patient_id = ?;";

const std::string DatabaseSchemaSql::SQL_SELECT_PATIENT_BY_ID =
    "select patient_id, patient_name, patient_birth_day, gender_id, city_id, patient_phone from patients
    where "
    "patient_id = ?;";

const std::string DatabaseSchemaSql::SQL_SELECT_ALLS_PATIENTS =
    "select patient_id, patient_name, patient_birth_day, gender_id, city_id, patient_phone from
    patients;";

const std::string DatabaseSchemaSql::SQL_SELECT_ALLS_PATIENTS_ORDER_BY_NAME =
    "select patient_id, patient_name, patient_birth_day, gender_id, city_id, patient_phone from patients "
    "order by patient_name asc;";

const std::string DatabaseSchemaSql::SQL_SELECT_ALLS_PATIENTS_ORDER_BY_BIRTHDATE =
    "select patient_id, patient_name, patient_birth_day, gender_id, city_id, patient_phone from patients "
    "order by patient_birth_day asc;";

const std::string DatabaseSchemaSql::SQL_SELECT_ALLS_PATIENTS_ORDER_BY_CITY =
    "select patient_id, patient_name, patient_birth_day, gender_id, p.city_id, patient_phone from patients
    as p "
    "join cities as c on c.city_id = p.city_id order by c.city_name asc;";

const std::string DatabaseSchemaSql::SQL_SELECT_DIAGNOSES_BY_PATIENT_ID =
    "select diagnosis_name from diagnoses where diagnosis_id in " \
    "(select diagnosis_id from patients_diagnoses where patient_id = ?);";

const std::string DatabaseSchemaSql::SQL_INSERT_DIAGNOSE =
    "insert into diagnoses (diagnosis_name) values (?);";

const std::string DatabaseSchemaSql::SQL_SELECT_DIAGNOSE_ID_BY_NAME =
    "select diagnosis_id from diagnoses where diagnosis_name = ?;";

const std::string DatabaseSchemaSql::SQL_INSERT_DIAGNOSE_OF_PATIENT =
    "insert into patients_diagnoses (patient_id, diagnosis_id) values (?, ?);";

const std::string DatabaseSchemaSql::SQL_DELETE_DIAGNOSES_OF_PATIENT =
    "delete from patients_diagnoses where patient_id = ?;";

const std::string DatabaseSchemaSql::SQL_INSERT_CITY =
    "insert into cities (city_name) values (?);";

const std::string DatabaseSchemaSql::SQL_SELECT_CITY_ID_BY_NAME =
    "select city_id from cities where city_name = ?;";

const std::string DatabaseSchemaSql::SQL_SELECT_CITY_NAME_BY_ID =
    "select city_name from cities where city_id = ?;";

const std::string DatabaseSchemaSql::SQL_SELECT_PATIENTS_BY_NAME =
    "select patient_id, patient_name, patient_birth_day, gender_id, city_id, patient_phone from patients "
    "where patient_name like '%||?||%'";

const std::string DatabaseSchemaSql::SQL_SELECT_PATIENTS_BY_PHONE =
    "select patient_id, patient_name, patient_birth_day, gender_id, city_id, patient_phone from patients "
    "where patient_phone like '%||?||%'";

const std::string DatabaseSchemaSql::SQL_SELECT_PATIENTS_BY_CITY_NAME =
    "select patient_id, patient_name, patient_birth_day, gender_id, city_id, patient_phone from patients "
    "where city_id in (select city_id from cities where city_name like '%||?||%'");

```

```

const std::string DatabaseSchemaSql::SQL_SELECT_PATIENTS_BY_DIAGNOSE_NAME =
    "select patient_id, patient_name, patient_birth_day, gender_id, city_id, patient_phone from patients "
    "where patient_id in (select patient_id from patients_diagnoses where diagnosis_id in "
    "(select diagnosis_id from diagnoses where diagnosis_name like '%||?||%'));";

const std::string DatabaseSchemaSql::SQL_SELECT_PATIENTS_BY_NOT_CITY_NAME =
    "select patient_id, patient_name, patient_birth_day, gender_id, city_id, patient_phone from patients "
    "where city_id not in (select city_id from cities where city_name like '%||?||%'");

const std::string DatabaseSchemaSql::SQL_SELECT_PATIENTS_BY_AGE_AND_DIAGNOSE_NAME =
    "select patient_id, patient_name, patient_birth_day, gender_id, city_id, patient_phone from patients "
    "where patient_birth_day <= ? and "
    "patient_id in (select patient_id from patients_diagnoses where diagnosis_id in "
    "(select diagnosis_id from diagnoses where diagnosis_name like '%||?||%'));";

```

## Модуль DatabaseSQLite.lib

### Файл stdafx.h

```

#pragma once

#include "targetver.h"

#define WIN32_LEAN_AND_MEAN           // Exclude rarely-used stuff from Windows headers

// stl
#include <string>
#include <thread>
#include <chrono>

// sqlite
#include <sqlite3.h>

```

### Файл SQLiteStatementType.h

```

#pragma once

struct SQLiteStatementDeleter
{
    void operator()(sqlite3_stmt* stmt)
    {
        sqlite3_finalize(stmt);
    }
};

using SQLiteStatement_t = std::unique_ptr < sqlite3_stmt, SQLiteStatementDeleter > ;

```

### Файл DatabaseSQLiteException.h

```

#pragma once

class DatabaseSQLiteException : public std::exception
{
public:
    explicit DatabaseSQLiteException(const std::string& msg, int code);

    const char* what() const throw() override;

    const std::string& GetMessage() const;
    int GetCode() const;

private:
    std::string m_msg;
    int m_code;
};

```

### Файл DatabaseSQLiteException.cpp

```

#include "stdafx.h"
#include "DatabaseSQLiteException.h"

```

```

DatabaseSQLiteException::DatabaseSQLiteException(const std::string& msg, int code) :
    m_msg(msg),
    m_code(code)
{
}

const char* DatabaseSQLiteException::what() const throw()
{
    return m_msg.c_str();
}

const std::string& DatabaseSQLiteException::GetMessage() const
{
    return m_msg;
}

int DatabaseSQLiteException::GetCode() const
{
    return m_code;
}

```

## Файл DatabaseSQLite.h

```

#pragma once

#include "SQLiteStatementType.h"

class DatabaseSQLite
{
    static const int SQLITE_BUSY_WAIT_PERIOD_MS;

    struct SQLiteConnectionDeleter
    {
        void operator()(sqlite3* p)
        {
            int res = sqlite3_close_v2(p);

            if (res != SQLITE_OK)
            {
                res = res;
            }
        }
    };

    static int SqliteBusyWaitHandler(void* ud, int count);

public:
    using SQLiteConnection_t = std::unique_ptr < sqlite3, SQLiteConnectionDeleter >;

public:
    explicit DatabaseSQLite(const std::string& connectionString, bool createIfNotExist, bool readOnly);

    const std::string& GetConnectionString() const;

    SQLiteStatement_t CreateStatement(const std::string& sql) const;
    void ExecuteSql(const std::string& sql);

    void BeginTransaction();
    void CommitTransaction();
    void RollbackTransaction();

    sqlite3* GetSqliteConnectionPtr() const;

private:
    void EnableWaitBusyDatabase();
    void EnableForeignKeys();

private:
    const std::string m_connectionString;
    SQLiteConnection_t m_sqliteConnection;
};

```

## Файл DatabaseSQLite.cpp

```
#include "stdafx.h"
#include "DatabaseSQLite.h"
#include "DatabaseSQLiteException.h"

const int DatabaseSQLite::SQLITE_BUSY_WAIT_PERIOD_MS = 20;

DatabaseSQLite::DatabaseSQLite(const std::string& connectionString, bool createIfNotExist, bool readOnly)
:
    m_connectionString(connectionString)
{
    int openFlags = SQLITE_OPEN_NOMUTEX;

    if (createIfNotExist)
    {
        openFlags |= SQLITE_OPEN_CREATE;
    }

    if (readOnly)
    {
        openFlags |= SQLITE_OPEN_READONLY;
    }
    else
    {
        openFlags |= SQLITE_OPEN_READWRITE;
    }

    sqlite3* pSqliteConnection = nullptr;
    int res = sqlite3_open_v2(connectionString.c_str(), &pSqliteConnection,
        openFlags, nullptr);

    if (res != SQLITE_OK)
    {
        std::string exceptionText = "can't open database file: " + connectionString;
        throw std::exception(exceptionText.c_str());
    }

    m_sqliteConnection = SQLiteConnection_t(pSqliteConnection);

    EnableWaitBusyDatabase();
    EnableForeignKeys();
}

void DatabaseSQLite::EnableWaitBusyDatabase()
{
    sqlite3_busy_handler(m_sqliteConnection.get(), SqliteBusyWaitHandler, this);
}

int DatabaseSQLite::SqliteBusyWaitHandler(void* /*udp*/, int /*count*/)
{
    std::this_thread::sleep_for(
        std::chrono::milliseconds(SQLITE_BUSY_WAIT_PERIOD_MS));

    return 1;
}

void DatabaseSQLite::EnableForeignKeys()
{
    ExecuteSql("PRAGMA foreign_keys = ON;");
}

void DatabaseSQLite::BeginTransaction()
{
    ExecuteSql("BEGIN TRANSACTION;");
}

void DatabaseSQLite::CommitTransaction()
{
    ExecuteSql("COMMIT TRANSACTION;");
}

void DatabaseSQLite::RollbackTransaction()
{
    ExecuteSql("ROLLBACK TRANSACTION;");
}
```

```

}

sqlite3* DatabaseSQLite::GetSqliteConnectionPtr() const
{
    return m_sqliteConnection.get();
}

const std::string& DatabaseSQLite::GetConnectionString() const
{
    return m_connectionString;
}

SQLiteStatement_t DatabaseSQLite::CreateStatement(const std::string& sql) const
{
    sqlite3_stmt* pStatement = nullptr;
    int sqlitePrepareResult = sqlite3_prepare_v2(m_sqliteConnection.get(),
        sql.c_str(), (int)sql.length(),
        &pStatement, nullptr);

    if (sqlitePrepareResult == SQLITE_OK)
    {
        return SQLiteStatement_t(pStatement);
    }
    else
    {
        const char* pExecErrorMessage = sqlite3_errmsg(m_sqliteConnection.get());

        throw DatabaseSQLiteException(pExecErrorMessage != nullptr ? pExecErrorMessage : "",
            sqlitePrepareResult);
    }
}

void DatabaseSQLite::ExecuteSql(const std::string& sql)
{
    bool executeSuccess = false;

    std::string execErrorMessage;
    char* pExecErrorMessage = nullptr;

    int sqliteExecResult = sqlite3_exec(m_sqliteConnection.get(), sql.c_str(),
        nullptr, nullptr, &pExecErrorMessage);

    if (sqliteExecResult == SQLITE_OK ||
        sqliteExecResult == SQLITE_DONE || sqliteExecResult == SQLITE_ROW)
    {
        executeSuccess = true;
    }

    if (pExecErrorMessage != nullptr)
    {
        execErrorMessage = pExecErrorMessage;
        sqlite3_free(pExecErrorMessage);
    }

    if (!executeSuccess)
    {
        throw DatabaseSQLiteException(execErrorMessage, sqliteExecResult);
    }
}

```