МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Національний університет «Запорізька політехніка»**

Кафедра програмних засобів

(найменування кафедри)

**КУРСОВИЙ ПРОЕКТ**

з дисципліни «Об’єктно-орієнтоване програмування»

(назва дисципліни)

на тему: «Offline лікар»

Студента 2 курсу групи КНТ-119 СП

спеціальності Інженерія програмного забезпечення

напряму підготовки: Інженерія програмного забезпе -

чення Мінаєв С. Г.

(прізвище та ініціали)

Керівник Доцент к.н.т., Каплієнко Т. І.

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала

Кількість балів: \_\_\_\_\_\_\_ Оцінка: ECTS

Члени комісії Каплієнко Т. І.

(підпис) (прізвище та ініціали)

Миронова Н. О.

(підпис) (прізвище та ініціали)

Табунщик Г. В.

(підпис) (прізвище та ініціали)

м. Запоріжжя

2019 рік

**ЗАВДАННЯ  
на курсовий проект студента**

Мінаєв С. Г.

1. Тема проекту: Offline лікар

2. Термін здачі студентами закінченого проекту: 24 квітня 2020

3. Вихідні дані до проекту: Створити програму, що емулює роботу медичного чат-боту: замовлення ліків, діагностування захворювання, бесіда з лікарем;

вихідні дані: список аптек з найдешевшими цінами на ліки, список вірогідних захворювань, журнал бесіди з лікарем.

4. Зміст розрахунково-пояснювальної записки:

1. Аналіз предметної області;

2. Аналіз програмних засобів;

3. Основні рішення з реалізації компонентів системи;

4. Керівництво програміста;

5. Керівництво користувача;

Висновки

Додаток А. Текст програми

Додаток Б. Інтерфейс програми

Додаток В. Слайди презентації

5. Дата видачі завдання: 17 вересня 2019 року

Календарний план

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів курсового  проекту (роботи) | Термін виконання етапів  проекту (роботи) | Примітка |
| 1. | Аналіз індивідуального завдання. | 1-2 тиждень |  |
| 2. | Аналіз програмних засобів, що будуть використовуватись в роботі. | 3-4 тиждень |  |
| 3. | Аналіз структур даних, що необхідно використати в курсової роботі. | 4-5 тиждень |  |
| 4. | Вивчення можливостей програмної реалізації структур даних та інтерфейсу користувача. | 5-9 тиждень |  |
| 5. | Аналіз вимог до апаратних засобів | 9 тиждень |  |
| 6. | Розробка програмного забезпечення | 9-15 тиждень |  |
| 7. | Оформлення, відповідних пунктів пояснювальної записки. | 10-16 тиждень | Розділи 1-5 ПЗ |
| 8. | Захист курсової роботи. | 17 тиждень |  |

Студент Мінаєв С. Г.

Керівник Каплієнко Т. І.

«\_\_\_\_\_\_\_» 2019 р.

реферат

Мета роботи – створення програмного забезпечення для емуляції роботи справжнього медичного чат-боту в режимі офлайн.

Проведено аналіз предметної області, досліджено аналогічне програмне забезпечення, а також існуючі методи та програмні засоби для вирішення завдання.

Для реалізації програмного продукту використовувалася мова програмування С++ та середа розробки Qt Creator.

Здійснено опис прийнятих рішень, реалізованих класів, наведено опис полів та методів реалізованих класів.

У даній роботі було проведено дослідження процесу створення чат-ботів, розглянуто особливості мови програмування С++ у середовищі розробки Qt Creator. За основу для розробки графічного інтерфейсу для користувача була узята мова розмітки сторінки QML. Потреби у збереженні, виведенні, конвертації та роботі з даними реалізовані у роботі із реляційною базою даних на основі SQLite3.

C++, CHATBOT, PATTERN MATCHING, SQLITE3, QML, ЕВРИСТИЧНІ АЛГОРИТМИ, МЕДИЦИНА.

**ЗМІСТ**

[Календарний план](#_Toc27629364) 3

[реферат](#_Toc27629365) 4

[зміст](#_Toc27629366) 5

[перелік умовних позначень, символів, одиниць, скорочень і термінів](#_Toc27629367) 7

[вступ](#_Toc27629368) 8

[1 Аналіз предметної області](#_Toc27629369) 9

[1.1 Аналіз медицини, як основи предметної області](#_Toc27629370) 9

[1.2 Огляд існуючих методів вирішення завдання](#_Toc27629371) 9

[1.2 Огляд існуючих програм та сервісів 1](#_Toc27629372)0

[1.2.1 Передмова 1](#_Toc27629373)0

[1.2.2](#_Toc27629374) ELIZA [1](#_Toc27629374)0

[2 Аналіз програмних засобів 1](#_Toc27629377)2

[2.1 Огляд особливостей мови програмування 1](#_Toc27629378)2

[2.2 Огляд особливостей обраного компілятору 1](#_Toc27629379)2

[2.3 Огляд клас](#_Toc27629380)ової ієрархії [1](#_Toc27629380)3

[2.4 Висновки з розділу 1](#_Toc27629381)3

[3 основні рішення з реалізації компонентів системи 1](#_Toc27629382)4

[3.1 Основні рішення щодо розроблених класів 1](#_Toc27629383)4

[3.1.1 Клас](#_Toc27629384) SqlTopicModel15

[3.1.2 Клас S](#_Toc27629385)qlConversationModel15

[3.1.3 Клас A](#_Toc27629386)nswerManager [1](#_Toc27629386)6

[3.1.4 Клас M](#_Toc27629387)edicinesOrderProcessor [1](#_Toc27629387)7

[3.1.5 Клас D](#_Toc27629388)iaseaseProcessor [1](#_Toc27629388)8

[3.1.6 Клас D](#_Toc27629389)octorProcessor19

[3.2 Основні розроблені алгоритми](#_Toc27629401) 19

[3.2.1 Аналіз](#_Toc27629402) алгоритмів19

[3.2.2 Аналіз алгоритму](#_Toc27629403) пошуку аптек [2](#_Toc27629403)0

[3.2.3 А](#_Toc27629404)наліз алгоритму визначення списку вірогідних хвороб [2](#_Toc27629404)0

[3.2.4](#_Toc27629405) Аналіз алгоритму визначення коректної відповіді лікаря [2](#_Toc27629405)1

[3.3 Основні рішення щодо розробки інтерфейсу](#_Toc27629408) 23

[3.3.1 Вибір платформи.](#_Toc27629409) 23

[3.3.2 Основні рішення з впорядкування елементів інтерфейсу](#_Toc27629410) 23

[3.4 Основні рішення щодо розробки бази даних](#_Toc27629413) 24

[3.4.1 Основні рішення щодо вибору бази даних](#_Toc27629414) 24

[3.4.2 Структура збереження даних у БД](#_Toc27629415) 24

[3.4.2 Основні запити SQL, що було використано](#_Toc27629416) 28

[4 керівництво програміста](#_Toc27629417) 30

[4.1 Призначення та умови застосування програми](#_Toc27629418) 30

[4.1.1 Призначення програми](#_Toc27629419) 30

[4.1.2 Функції програми](#_Toc27629420) 30

[4.1.3 Умови застосування програми](#_Toc27629421) 31

[4.2 Характеристика програми](#_Toc27629422) 31

[4.3 Звертання до програми](#_Toc27629426) 31

[4.4 Вхідні та вихідні дані](#_Toc27629427) 31

[4.4.1 Вхідні дані](#_Toc27629428) 31

[4.4.2 Вихідні дані](#_Toc27629429) 32

[5 керівництво користувача](#_Toc27629431) 33

[5.1 Призначення програми](#_Toc27629432) 33

[5.2 Умови виконання програми](#_Toc27629433) 33

[5.2.1 Апаратні вимоги програми](#_Toc27629434) 33

[5.2.2 Вимоги до користувача](#_Toc27629435) 33

[5.3 Виконання програми](#_Toc27629436) 34

[5.3.1 Запуск програми](#_Toc27629437) 34

[5.3.2 Виконання роботи з програмою](#_Toc27629438) 35

[5.4](#_Toc27629439) Форма діалогу з чат-ботом [3](#_Toc27629439)5

[Висновки](#_Toc27629441) 37

[ПЕРЕЛІК ПОСИЛАНЬ](#_Toc27629442) 38

[Додаток А Діаграма класів](#_Toc27629443) 39

[Додаток Б Код Програми](#_Toc27629444) 40

[Додаток В інтерфейс програми](#_Toc27629445) 41

перелік умовних позначень, символів, одиниць, скорочень і термінів

БД – база даних.

SQL – Structured Query Language.

СУБД – система маніпулювання БД.

Запит – запит до БД за допомогою SQL.

ООП – об’єктно-орієнтоване програмування.

QML – Qt Meta Language (Qt Modeling Language).

вступ

Найбільш цінним скарбом, що є в кожної людини, безперечно, є її життя та здоров’я. Як фізичний так і психічний стан людини потрібно постійно підтримувати в задовільному стані та вживати превентивних та профілактичних заходів.

В той же час, розвиток комп’ютерних технологій та їх всебічне проникнення в життя людей, дозволяє винаходити нові шляхи для вирішення повсякденних людських проблем. Персональний комп’ютер та розроблюване для нього ПЗ пройшло певний шлях власної еволюції і сьогодні людство має у себе на поруках потужний кластер апаратно-програмного комплексу, що став для всіх нас незамінним помічником та вірним другом. Роботи стали одним із найпоширеніших варіантів взаємодії комп’ютера та людини.

Поєднуючи турботу про здоров’я людей та розвиток інформаційних технологій, було прийняте рішення про створення програми, що дозволила б тестувати алгоритми та логіку створення так званого чат-боту, тобто програми, що являла б собою певний візуальний інтерфейс для взаємодії людини та комп’ютера шляхом надсилання людиною повідомлення, обробки його машиною та надсилання машиною відповіді в залежності від контексту з метою вирішення певної проблеми людини на даний момент часу.

Актуальність цієї курсової роботи підтверджується тим, що наразі вже існує декілька спроб створення схожих програмних комплексів, тобто ця тема — є досить актуальною.

Отже, з метою вирішення вищезазначеної проблеми, було поставлено задачу створити відкритий програмний застосунок, що може стати прототипом потужної програмної системи і бути в нагоді для безлічі людей.

1 Аналіз предметної області

В даному розділі розглянуті існуючі методи вирішення завдання створення медичного чат-боту та програми, що реалізують дану потребу, а саме створення медичного чат-боту.

1.1 Аналіз медицини, як основи предметної області

Проблема створення медичного чат-боту стоїть перед багатьма медичними структурами: лікарнями, аптеками тощо. Основними вимогами до такого програмного комплексу такі:

* простота у використанні кінцевим користувачем;
* використання сучасних концепцій користувацького інтерфейсу;
* використання людської мови спілкування з ботом замість команд;
* можливість підтримувати існуючу кодову базу;
* можливість інтегрувати нові модулі в існуючу кодову базу;
* відкритість вихідного коду побудованого програмного комплексу.

1.2 Огляд існуючих методів вирішення завдання

Створення чат-ботів, у тому числі й медичних — задача, яка стояла перед компʼютерними інженерами-науковцями ще починаючи із середини минулого ХХ-го століття.

Ще в далекому 1966 році Жосеф Вейценбаум написав першу в світі програму, яка поводила себе ніби справжній лікар-психотерапевт. Це було справжнє відкриття, адже програма приймала на вхід звичайну людську мову і видавала як результат вираховувані судження, що видавалися за судження справжньої людини.

Користувачі далеко не одразу здогадувалися, що розмовляють вони усього лише з майстерно виготовленим алгоритмом, що має назву — Pattern Matching.

Цей алгоритм приймає на вхід висловлювання звичайною людською мовою, яке далі досліджується на предмет відомих програмі так званих ключових слів, словосполучень або фраз. Реагуючи на ключове слово, програма, що використовує такий алгоритм, може знайти необхідну шаблонну фразу. Також, розібравши вхідне повідомлення на складові частини, програма може підставити тему висловлювання в знайдену шаблонну фразу і отримати таким чином певну відповідь, яка на перший погляд здається цілком осмисленою.

**1.2 Огляд існуючих програм та сервісів**

1.2.1 Передмова

Перш за все слід зазначити, що головними критеріями чат-боту є:

* сучасна та швидка мова програмування;
* сучасний та легкий користувацький інтерфейс;
* інтеграція в застосунку більше ніж однієї функції;

1.2.2 ***ELIZA***

Посилання: <https://en.wikipedia.org/wiki/ELIZA>.

Короткий опис: ELIZA — перша в світі програма, що емулює роботу справжнього лікаря-психотерапевта, написана на мові програмування SLIP та має тільки консольну версію інтерфейсу, була портована також на Emacs LISP та запущена в текстовому редакторі GNU Emacs, що дозволило використовувати це програмне забезпечення у вигляді звичайного плагіну.

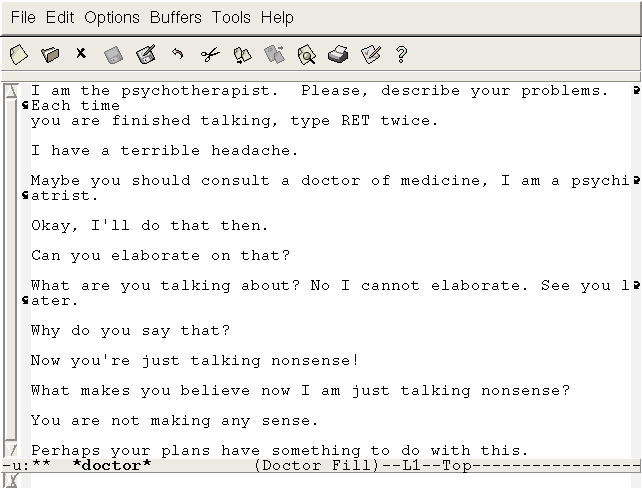
Переваги:

* ця програма була перша в своєму роді;
* скриптова мова програмування, що спрощувала розробку;
* для роботи цієї програми не потрібні великі ресурси;
* програма сприймає на вхід людську мову;

Недоліки

* програма має застарілий інтерфейс командного рядку, що в наш час багатьох людей може здивувати, а деяких навіть шокувати;
* програма вже давно не розвивається, хоча в мережі є безліч її клонів, зокрема на таких мовах програмування як LISP або Clojure;
* програма має на меті дозволити людині спілкуватися з лікарем-психологом та не вміщує в собі інший функціонал: замовлення медикаментів, побудова списку вірогідних захворювань пацієнта.

Робота програми продемонстрована на рисунку 1.1.

Рисунок 1.1 – Робота програми «ELIZA»

1.2.3 ChatterBot

Посилання: https://github.com/gunthercox/ChatterBot.

Короткий опис: ChatterBox — Open Source платформа, що побудована на основі Python 3. Ця платформа використовує штучний інтелект та натренована на безлічі розмов живих людей. Мета цієї платформи — зробити можливим побудову незалежного кінцевого застосунку, що буде являти собою офлайн чат-бот.

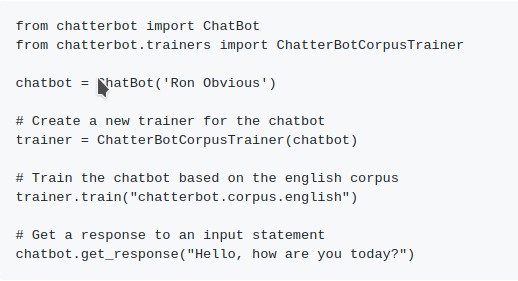
Переваги:

* відкритість платформи;
* скриптова мова програмування, що спрощує розробку;
* обробка висловлювання людською мовою.

Недоліки

* Python 3 не дозволяє створювати досить швидкодіючі застосунки;
* проект не пропонує кінцевого рішення для користувача.

Робота програми продемонстрована на рисунку 1.2.

Рисунок 1.2 – Робота програми «ChatterBot»

1.2.4 ***EasyNLU***

Посилання: https://github.com/kolloldas/EasyNLU.

Короткий опис: EasyNLU — Open Source бібліотека на мові програмування Java, що дозволяє структурувати дані отримані шляхом обробки висловлювання людською мовою. Використовуючи цю бібліотеку можна створити власний застосунок, що буде являти собою повноцінний офлайн чат-бот.

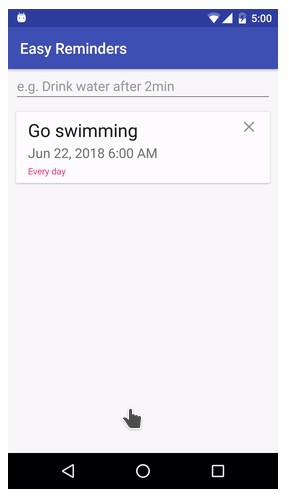
Переваги:

* відкритість бібліотеки;
* широкий простір використання бібліотеки;
* обробка висловлювання людською мовою.

Недоліки

* Java не дозволяє створювати досить швидкодіючі застосунки;
* проект не пропонує кінцевого рішення для користувача;
* останнє оновлення кодової бази проекту — 2 роки тому.

Робота програми продемонстрована на рисунку 1.3.

Рисунок 1.3 – Робота програми «EasyNLU»

2 Аналіз програмних засобів

В даному розділі розглянуті основні особливості програмних засобів, за допомогою який реалізовано курсовий проект.

2.1 Огляд особливостей мови програмування

C++ (Сі-плюс-плюс) — мова програмування високого рівня з підтримкою кількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної. Розроблена Б'ярном Страуструпом (англ. Bjarne Stroustrup) в AT&T Bell Laboratories (Мюррей-Хілл, Нью-Джерсі) 1979 року та початково отримала назву «Сі з класами». Згодом Страуструп перейменував мову на C++ у 1983 р. Базується на мові С. Вперше описана стандартом ISO/IEC 14882:1998, найбільш актуальним же є стандарт ISO/IEC 14882:2014.

У 1990-х роках С++ стала однією з найуживаніших мов програмування загального призначення. Мову використовують для системного програмування, розробки програмного забезпечення, написання драйверів, потужних серверних та клієнтських програм, а також для розробки розважальних програм, наприклад, відеоігор. С++ суттєво вплинула на інші популярні сьогодні мови програмування: С# та Java. [1].

2.2 Огляд особливостей обраного компілятору

Набір компіля́торів GNU (GNU Compiler Collection, GCC) — набір компіляторів для різних мов програмування. GCC — вільне програмне забезпечення, розроблене Фондом Вільних Програм під ліцензією GNU GPL та GNU LGPL, і є ключовою складовою набору знарядь розробки GNU (GNU development toolchain). Це стандартний компілятор для вільних Юнікс-подібних операційних систем, і деяких пропрієтарних систем, що з них розвинулись, наприклад Mac OS X.

Спочатку називався GNU Компілятор Сі, оскільки підтримував лише одну мову програмування — C. Пізніше був розширений для підтримки C++, Fortran, Java (компілятор GCJ), Ada, D, та інших. [2]

2.3 Огляд класової ієрархії

SqlTopicModel – модель розділів чат-боту;

SqlConversationModel – модель діалогів з чат-ботом;

AnswerManager – клас-диспетчер для надання коректної відповіді;

MedicinesOrderProcessor – процесор відповіді «Medicines Order»;

DiaseaseProcessor – процесор відповіді «Guess Your Diasease»;

DoctorProcessor – процесор відповіді «Live Consultaion with Doctor».

2.4 Висновки з розділу

У даному розділі було описано мову програмування та її особливості. Також було описано усі класи програмного застосунку.

3 основні рішення з реалізації компонентів системи

В даному розділі розглянуті основні рішення з розробки класів, основні розроблені алгоритми, рішення щодо розробки інтерфейсу користувача, рішення щодо збереження даних та використання бази даних.

3.1 Основні рішення щодо розроблених класів

Структура програми складається з 6 класів, 4 з яких – користувацькі, 2 – класи форм. Діаграми користувацьких класів знаходяться у додатку А.

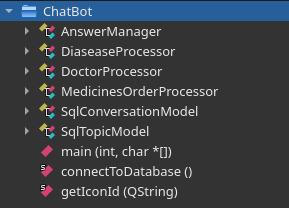
Класову структуру програми зображено на рисунку 3.1.

Рисунок 3.1 – Класова структура програми

3.1.1 Клас ***SqlTopicModel***

Клас SqlTopicModel містить код для відображення усіх існуючих розділів чат-боту в БД. Код файлу визначення класу знаходиться в додатку Б. Дані та методи класу наведені у таблиці 3.1.

Таблиця 3.1 – Опис полів та методів класу SqlTopicModel

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| SqlTopicModel(QObject \*parent = 0) | Конструктор класу |
| Qvariant data(const QModelIndex &index, int role) const override; | Повертає дані моделі |
| QHash<int, QByteArray> roleNames() const override; | Повертає імена стовпців моделі |

3.1.2 Клас ***SqlConverstionModel***

Клас SqlConverstionModel містить код для відображення взаємодії користувача з певним розділом чат-боту. Дані зберігаються в БД. Код файлу визначення класу знаходиться в додатку Б. Дані та методи класу наведені у таблиці 3.2.

Таблиця 3.2 – Опис полів та методів класу SqlConverstionModel

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| SqlConversationModel(QObject \*parent = 0); | Конструктор класу |
| QString icon() const; | Повертає імʼя іконки розділу |
| QString getRecipient() const; | Повертає ім’я розділу (адресанта) |
| void setRecipient(const QString &recipient); | Встановлює ім’я розділу (адресанта) |
| QVariant data(const QModelIndex &index, int role) const override; | Повертає дані моделі |

Продовження таблиці 3.2

|  |  |
| --- | --- |
| **1** | **2** |
| Q\_INVOKABLE void sendMessage(const QString &recipient, const QString &topicId, const QString &message); | Реєструє нове повідомлення від користувача |

3.1.3 Клас ***AnswerManager***

Клас AnswerManager містить код для вибору необхідного процесору для відповіді на повідомлення-запит користувача. Код файлу визначення класу знаходиться в додатку Б. Дані та методи класу наведені у таблиці 3.2.

Таблиця 3.3 – Опис полів та методів класу AnswerManager

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| AnswerManager(); | Конструктор класу |
| QString appropriateMessage(); | Повертає вираховуване відповідне повідомлення-відповідь |
| void setRecipient(QString recipient); | Встановлює імʼя розділу (адресанта) |
| void setMessage(QString message); | Встановлює повідомлення-запит |
| ***private:*** | |
| QString whenMedicinesOrder(); | Повертає результат відповідного процесора |
| QString whenGuessYourDiasease(); | Повертає результат відповідного процесора |
| Qstring whenLiveConsultationWithDoctor(); | Повертає результат відповідного процесора |
| QString recipient; | Ім’я розділу |
| QString message; | Повідомлення-запит |
| MedicinesOrderProcessor mrProcessor; | Процесор відповідного розділу |
| DiaseaseProcessor diProcessor; | Процесор відповідного розділу |
| DoctorProcessor drProcessor | Процесор відповідного розділу |

3.1.4 Клас ***MedicinesOrderProcessor***

Клас MedicinesOrderProcessor містить код для обробки повідомлення-запиту користувача до відповідного розділу. Код файлу визначення класу знаходиться в додатку Б. Дані та методи класу наведені у таблиці 3.2.

Таблиця 3.4 – Опис полів та методів класу MedicinesOrderProcessor

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| MedicinesOrderProcessor(); | Конструктор класу |
| void setSourceMessage(QString message); | Встановлює повідомлення-запит |
| QString processAndGetProcessedMessage(); | Оброблює та повертає повідомлення-відповідь |
| ***private:*** | |
| QString previousSourceMessage; | Попереднє повідомлення користувача |
| QString sourceMessage; | Повідомлення-запит користувача |
| QString processedMesage; | Оброблене повідомлення-відповідь |
| QString currenntStage; | Поточна фаза діалогу |
| QString newStage; | Наступна фаза діалогу |
| bool stageValid; | Чи є повідомлення-запит коректним |
| bool retrieveStage(); | Отримує фазу діалогу з БД |
| void setResult(bool valid, QString msg, QString stage=NULL); | Встановлює проміжний результат обробки |
| bool isValidForStage(); | Чи є повідомлення-запит коректним |
| bool isFinalStage(); | Чи є фаза діалогу останньою |
| bool updateStage(); | Оновлює фазу діалогу |
| bool resetStage(); | Встановлює фазу діалогу в початок |
| bool currentStage(QString stage); | Порівнює фази діалогу |
| bool containsKeyWords(QStringList keywords); | Чи містить повідомлення-запит відповідні ключові слова |
| void processMessage(); | Оброблює повідомлення-запит |
| bool isMatchingInitInput(); | Повідомлення-запит є початковим |
| bool isMatchingAddingVerifyPositiveAnswer(); | Повідомлення-запит є позитивною відповіддю |
| bool isMatchingAddingFinishedInput(); | Повідомлення-запит є завершальним |
| bool isMatchingToBeDoneInput(); | Повідомлення-запит є остаточним |
| QString processUserDrugInfoInput(); | Пошук медикаменту в повідомленні |

Продовження таблиці 3.4

|  |  |
| --- | --- |
| **1** | **2** |
| QStringList processedDrugStringList; | Оброблений список медикаментів |

3.1.4 Клас ***DiaseaseProcessor***

Клас DiaseaseProcessor містить код для обробки запиту користувача відповідного розділу. Код файлу визначення класу знаходиться в додатку Б. Дані та методи класу наведені у таблиці 3.2.

Таблиця 3.4 – Опис полів та методів класу MedicinesOrderProcessor

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| DiaseaseProcessor(); | Конструктор класу |
| void setSourceMessage(QString message); | Встановлює повідомлення-запит |
| QString processAndGetProcessedMessage(); | Оброблює та повертає повідомлення-відповідь |
| ***private:*** | |
| QString previousSourceMessage; | Попереднє повідомлення користувача |
| QString sourceMessage; | Повідомлення-запит користувача |
| QString processedMesage; | Оброблене повідомлення-відповідь |
| QString currenntStage; | Поточна фаза діалогу |
| QString newStage; | Наступна фаза діалогу |
| bool stageValid; | Чи є повідомлення-запит коректним |
| bool retrieveStage(); | Отримує фазу діалогу з БД |
| void setResult(bool valid, QString msg, QString stage=NULL); | Встановлює проміжний результат обробки |
| bool isValidForStage(); | Чи є повідомлення-запит коректним |
| bool isFinalStage(); | Чи є фаза діалогу останньою |
| bool updateStage(); | Оновлює фазу діалогу |
| bool resetStage(); | Встановлює фазу діалогу в початок |
| bool currentStage(QString stage); | Порівнює фази діалогу |
| bool containsKeyWords(QStringList keywords); | Чи містить повідомлення-запит відповідні ключові слова |
| void processMessage(); | Оброблює повідомлення-запит |
| bool isMatchingInitInput(); | Повідомлення-запит є початковим |

Продовження таблиці 3.4

|  |  |
| --- | --- |
| **1** | **2** |
| bool isMatchingAddingFinishedInput(); | Повідомлення-запит є завершальним |
| bool isMatchingToBeDoneInput(); | Повідомлення-запит є остаточним |
| QStringList processUserSymptomInput(); | Пошук хвороб згідно симптому |
| QStringList diaseasesList; | Список знайдених хвороб |

***3.1.4 Клас DoctorProcessor***

Клас DoctorProcessor містить код для обробки запиту користувача відповідного розділу. Код файлу визначення класу знаходиться в додатку Б. Дані та методи класу наведені у таблиці 3.2.

Таблиця 3.6 – Опис полів та методів класу DoctorProcessor

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| DoctorProcessor() | Конструктор класу |
| void setSourceMessage(QString message); | Встановлює повідомлення-запит |
| QString processAndGetProcessedMessage(); | Оброблює та повертає результат |
| ***private:*** | |
| QString sourceMessage; | Повідомлення-запит користувача |
| QString withConvertedViewpoint(QString string); | Змінює типові слова у відповіді |
| QStringList retrieveAnswerList(QString keyword); | Дістає шаблони відповідей з БД |

3.2 Основні розроблені алгоритми

3.2.1 Аналіз алгоритмів

В ході написання курсової роботи було створено декілька алгоритмів.

Розроблені алгоритми не є ідеальними та мають вигляд прототипа.

***3.2.1.1 Аналіз алгоритму пошуку аптек***

Вхідними даними для алгоритму пошуку аптек є:

* набір строк із назвами медикаментів та їх кількістю.

Під час підрахунку остаточного результату алгоритм працює з проміжними даними:

* список вже оброблених аптек та медикаментів.

У кінці виконання алгоритм формує вихідні данні:

* форматована строка з назвами аптек, цінами та назвами медикаментів.

Усі значення мають бути коректними та строго формалізованими (це виконується на етапі створення бази даних).

Увесь алгоритм можна поділити на 2 етапи:

1. Прийняття повідомлення користувача.
2. Пошук аптеки з найменшою ціною на препарат.

Виконання кожного етапу базується та таких алгоритмах:

* алгоритм лінійного пошуку;
* алгоритм сортування вставками;
* елементи динамічного програмування.

***3.2.1.1 Аналіз алгоритму визначення списку вірогідних хвороб***

Вхідними даними для алгоритму визначення списку хвороб є:

* набір строк із симптомами хворого.

Під час підрахунку остаточного результату алгоритм працює з проміжними даними:

* список вже оброблених хвороб з відповідними симптомами.

У кінці виконання алгоритм формує вихідні данні:

* форматована строка зі списком вірогідних хвороб.

Усі значення мають бути коректними та строго формалізованими (це виконується на етапі створення бази даних).

Увесь алгоритм можна поділити на 2 етапи:

1. Прийняття повідомлення користувача.

2. Пошук хвороб з відповідними симптомами.

Виконання кожного етапу базується та таких алгоритмах:

* алгоритм лінійного пошуку;
* алгоритм сортування вставками;
* елементи динамічного програмування.

***3.2.1.1 Аналіз алгоритму визначення коректної відповіді лікаря***

Вхідними даними для алгоритму визначення відповіді лікаря є:

* повідомлення користувача.

У кінці виконання алгоритм формує вихідні данні:

* відповідь лікаря-психолога.

Усі значення мають бути коректними та строго формалізованими (це виконується на етапі створення бази даних).

Увесь алгоритм можна поділити на 2 етапи:

1. Прийняття повідомлення користувача.

2. Пошук найбільш коректної відповіді.

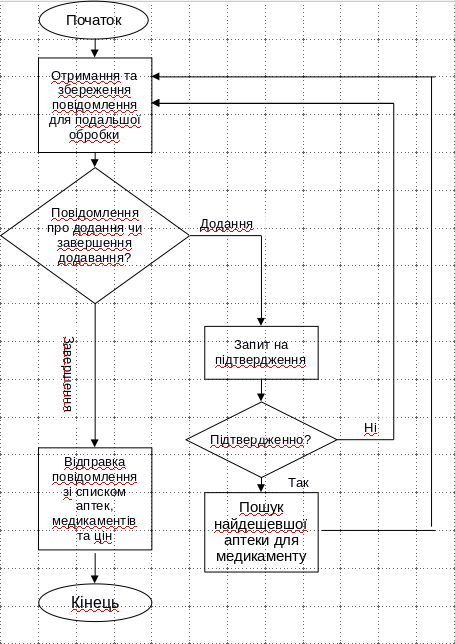
Виконання кожного етапу базується та таких алгоритмах:

* алгоритм лінійного пошуку;
* алгоритм сортування вставками;
* елементи динамічного програмування.
* елементи співпадання з шаблонами.

Розглянемо основні типові алгоритми.

Алгоритм пошуку аптек та медикаментів зображено у вигляді блок-схеми на рисунку 3.2.

Алгоритм пошуку хвороби за симптомами зображено у вигляді блок-схеми на рисунку 3.3.

Алгоритм пошуку коректної відповіді лікаря зображено у вигляді блок-схеми на рисунку 3.4.

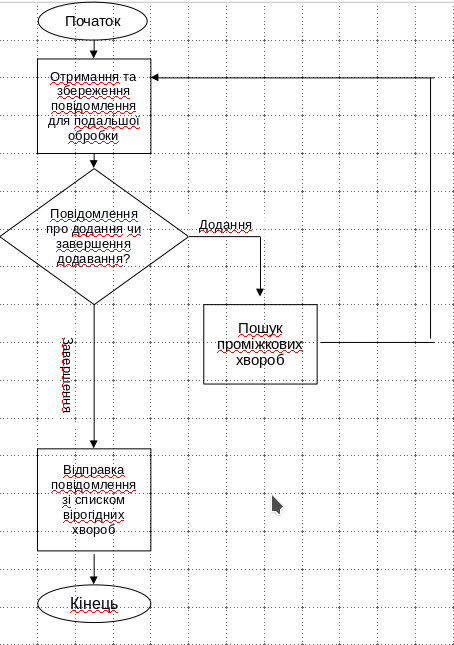
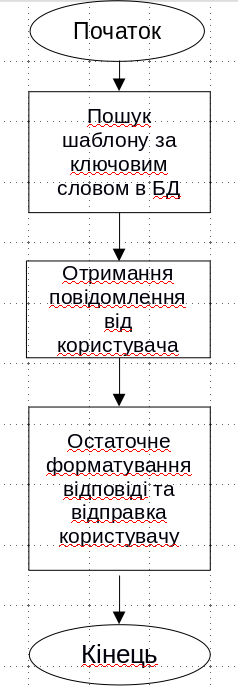
Рисунок 3.2 – Пошук аптек та медикаментів

Рисунок 3.3 – Пошук хвороби за симптомами

Рисунок 3.4 – Пошук пошуку коректної відповіді лікаря

3.3 Основні рішення щодо розробки інтерфейсу

3.3.1 Вибір платформи.

Головним рішенням ,щодо розробки інтерфейсу, було вибрати нову та сучасну платформу для розробки інтерфейсу QML (Qt Modeling/Meta Language). Ця платформа не залежить від платформи та більш зручним інтерфейсом для програміста. [3]

Для побудови інтерфейсу було використані такі основні компоненти як: StackView, ListView, QAbstractModel тощо.

3.3.2 Основні рішення з впорядкування елементів інтерфейсу

Під час розробки інтерфейсу користувача було використано наступні елементи графічного інтерфейсу: ListView, ColumnView, Pane.

3.4 Основні рішення щодо розробки бази даних

3.4.1 Основні рішення щодо вибору бази даних

База даних, як один з основних компонентів програми, потребує зваженого вибору середовища та платформи розробки, бо від цього залежатиме якість та комфорт не тільки у користуванні, а й у написанні програмного продукту, отже, саме тому основними критеріями для вибору БД були:

* відкритість платформи;
* портабельність платформи;
* вбудовуванність платформи;

Отже, проаналізувавши наявні пропозиції у цій сфері, було обрано платформу Sqlite3, яка вирізняється з-поміж інших варіантів, та відповідає усім заданим критеріям.

Особливістю SQLite є те, що вона не використовує парадигму клієнт-сервер, тобто рушій SQLite не є окремим процесом, з яким взаємодіє застосунок, а надає бібліотеку, з якою програма компілюється і рушій стає складовою частиною програми. Таким чином, як протокол обміну використовуються виклики функцій (API) бібліотеки SQLite. Такий підхід зменшує накладні витрати, час відгуку і спрощує програму. SQLite зберігає всю базу даних (включаючи визначення, таблиці, індекси і дані) в єдиному стандартному файлі на тому комп'ютері, на якому виконується застосунок. Простота реалізації досягається за рахунок того, що перед початком виконання транзакції весь файл, що зберігає базу даних, блокується; ACID-функції досягаються зокрема за рахунок створення файлу-журналу.

Кілька процесів або потоків можуть одночасно без жодних проблем читати дані з однієї бази. Запис в базу можна здійснити тільки в тому випадку, коли жодних інших запитів у цей час не обслуговується; інакше спроба запису закінчується невдачею, і в програму повертається код помилки. Іншим варіантом розвитку подій є автоматичне повторення спроб запису протягом заданого інтервалу часу.

У комплекті постачання йде також функціональна клієнтська частина у вигляді виконуваного файлу sqlite3, за допомогою якого демонструється реалізація функцій основної бібліотеки. Клієнтська частина працює з командного рядка, і дозволяє звертатися до файлу БД на основі типових функцій ОС.

Завдяки архітектурі рушія можливо використовувати SQlite як на вбудовуваних (embedded) системах, так і на виділених машинах з гігабайтними масивами даних. [4]

3.4.2 Структура збереження даних у БД

У процесі збереження даних використовуються таблиці, як ключові носії інформації, та представлені наступними екземплярами:

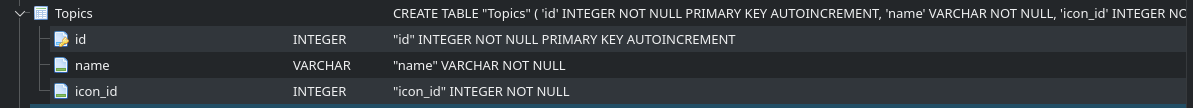
«*Topics*» - таблиця для збереження даних розділів, а саме: імʼя розділу, посилання на іконку розділу (рис. 3.5).

Рисунок 3.5 – Структура таблиці «*Topics*»

Первинним ключем є поле Id, яке ідентифікує кожен запис. Таблиці «*Topics*» та «*TopicIcon*» пов’язані зовнішнім ключем з допоміжними таблицями для економії пам’яті та збереження цілісності даних, бо зовнішній ключ виступає перешкодою для внесення неіснуючих чи нерелевантних даних.

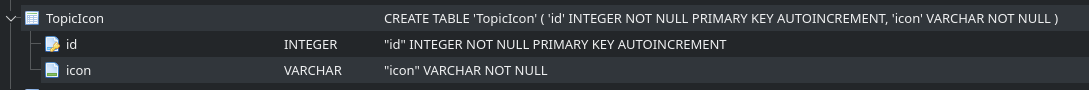
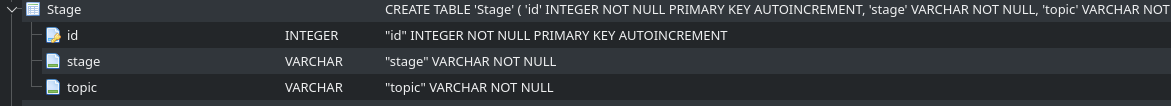
«*TopicIcon*» - таблиця для збереження іконок розділів, а саме: імʼя графічного файлу (рис. 3.6).

Рисунок 3.6 – Структура таблиці «*TopicIcon*»

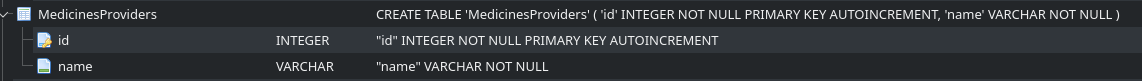
Первинним ключем є поле Id, яке ідентифікує кожен запис.

«*Stage*» - таблиця для збереження даних про поточну фазу діалогу, а саме: код фази, код розділу (рис. 3.7).

Рисунок 3.7 – Структура таблиці «*Stage*»

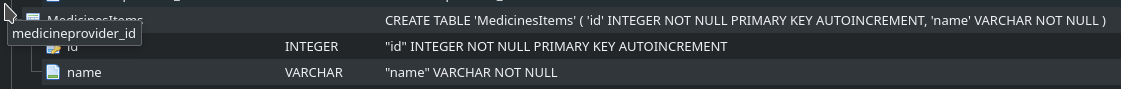
Первинним ключем є поле Id, яке ідентифікує кожен запис.

«*MedicinesProviders*» - таблиця для збереження даних аптек, а саме: імʼя аптеки (рис. 3.8).

Рисунок 3.8 – Структура таблиці «*MedicinesProviders*»

Первинним ключем є поле Id, яке ідентифікує кожен запис.

«M*edicinesItems*» - таблиця для збереження даних медикаментів, а саме: імʼя медикаменту (рис. 3.9).

Рисунок 3.9 – Структура таблиці «M*edicinesItems*»

Первинним ключем є поле Id, яке ідентифікує кожен запис.

«*Medicines*» - таблиця для представлення взаємовідношення аптек та медикаментів, а саме: ціна, посилання на медикамент, посилання на аптеку (рис. 3.10).

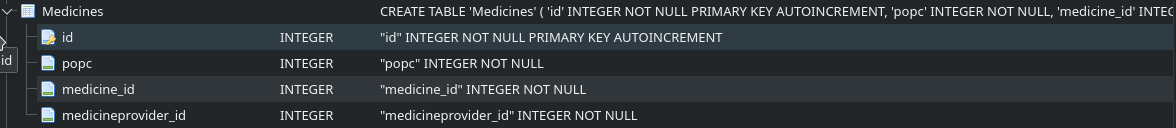


Рисунок 3.10 – Структура таблиці «*Medicines*»

Первинним ключем є поле Id, яке ідентифікує кожен запис. Таблиці «*Medicines*» та «*MedicinesProviders*», «*Medicines*» та «*MedicinesItems*» пов’язані зовнішнім ключем з допоміжними таблицями для економії пам’яті та збереження цілісності даних, бо зовнішній ключ виступає перешкодою для внесення неіснуючих чи нерелевантних даних.

«*Keywords*» - таблиця для збереження ключових слів, а саме: ключове слово (рис. 3.11).

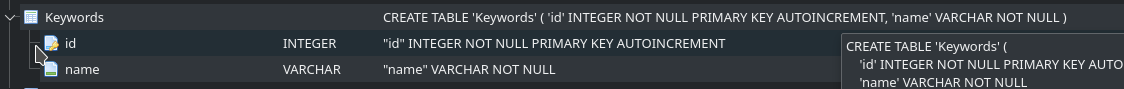
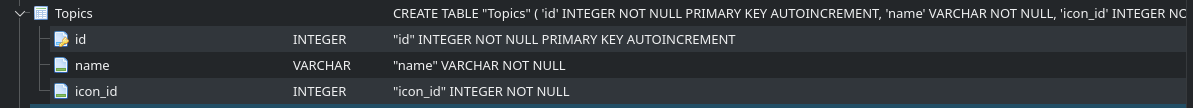


Рисунок 3.11 – Структура таблиці «*Keywords*»

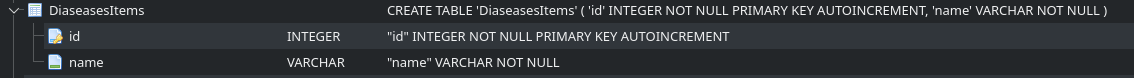
Первинним ключем є поле Id, яке ідентифікує кожен запис.

«D*iaseasesSymptoms*» - таблиця для збереження даних симптомів, а саме: симптом, чи є важливим (рис. 3.12).

Рисунок 3.12 – Структура таблиці «D*iaseasesSymptoms*»

Первинним ключем є поле Id, яке ідентифікує кожен запис.

«D*iaseasesItems*» - таблиця для збереження даних хвороб, а саме: хвороба (рис. 3.13).

Рисунок 3.13 – Структура таблиці «D*iaseasesItems*»

Первинним ключем є поле Id, яке ідентифікує кожен запис.

«*Diaseases*» - таблиця для збереження представлення взаємовідношення симптомів та хвороб, а саме: посилання на хворобу, посилання на симптом (рис. 3.14).

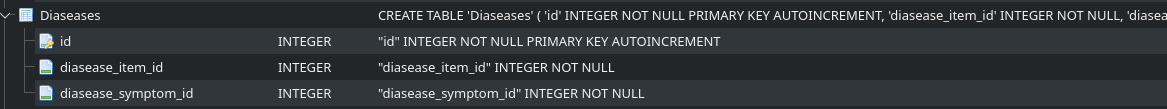


Рисунок 3.14 – Структура таблиці «*Diaseases*»

Первинним ключем є поле Id, яке ідентифікує кожен запис та допомагає зекономити пам'ять у інших таблицях, адже замість сполучення номеру, літери та кафедри використовується саме Id. Таблиці «D*iaseasesItems*» та «*Diaseases*», «D*iaseasesSymptoms*» та «*Diaseases*» пов’язані зовнішнім ключем з допоміжними таблицями для економії пам’яті та збереження цілісності даних, бо зовнішній ключ виступає перешкодою для внесення неіснуючих чи нерелевантних даних.

«*Conversations*» - таблиця для збереження діалогів, а саме: автора, адресанта, дату, повідомлення, посилання на іконку розділу (рис. 3.15).

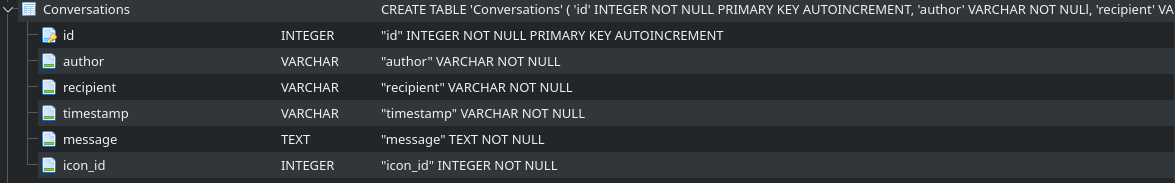


Рисунок 3.15 – Структура таблиці «*Conversations*»

Первинним ключем є поле Id, яке ідентифікує кожен запис. Таблиці «*Conversations*» та «*TopicIcon*» пов’язані зовнішнім ключем з допоміжними таблицями для економії пам’яті та збереження цілісності даних, бо зовнішній ключ виступає перешкодою для внесення неіснуючих чи нерелевантних даних.

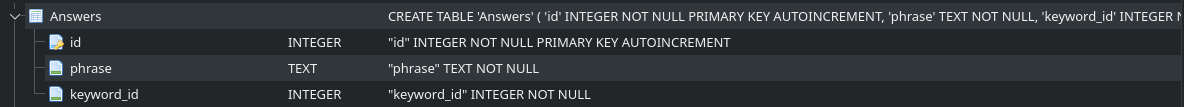
«*Answers*» - таблиця для збереження шаблонів фраз, а саме: фразу-шаблон, посилання на ключове слово (рис. 3.16).

Рисунок 3.16 – Структура таблиці «*Answers*»

Первинним ключем є поле Id, яке ідентифікує кожен запис. Таблиці «*Answers*» та «*Keywords*» пов’язані зовнішнім ключем з допоміжними таблицями для економії пам’яті та збереження цілісності даних, бо зовнішній ключ виступає перешкодою для внесення неіснуючих чи нерелевантних даних.

3.4.2 Основні запити SQL, що було використано

Мова для створення запитів SQlite є SQL-подібна, відрізняючись лише синтаксисом (несуттєво) та функціоналом. В ході роботи з базою даних були використані наступні запити для взаємодії:

* SELECT MedicinesProviders.name, MedicinesItems.name, MIN(Medicines.popc) FROM Medicines INNER JOIN MedicinesProviders ON MedicinesProviders.id = Medicines.medicineprovider\_id INNER JOIN MedicinesItems ON MedicinesItems.id = Medicines.medicine\_id WHERE MedicinesItems.name = '%1
* SELECT DiaseasesItems.name FROM Diaseases INNER JOIN DiaseasesSymptoms ON DiaseasesSymptoms.id = Diaseases.diasease\_symptom\_id INNER JOIN DiaseasesItems ON DiaseasesItems.id = Diaseases.diasease\_item\_id WHERE DiaseasesSymptoms.name LIKE UPPER('\%%1\%')"
* SELECT Answers.phrase FROM Answers INNER JOIN Keywords ON Answers.keyword\_id = Keywords.id WHERE Keywords.name = '%1"

4 керівництво програміста

В даному розділі розглянуті призначення, умови застосування, характеристика програми, звертання до програми, початкові та вихідні дані та представлені повідомлення.

4.1 Призначення та умови застосування програми

4.1.1 Призначення програми

Програма ChatBot призначена для емуляції та демонстрації роботи справжнього медичного чат-боту в режимі офлайн (використовується локальна база даних Sqlite3) в режимі настільного графічного застосунку.

4.1.2 Функції програми

Функції, що виконує програма:

* надає доступ до існуючих розділів (наявних в локальній БД);
* надає можливість знайти та замовити ліки за найнижчою ціною (ліки мають бути наявні в БД);
* надає можливість приблизно визначити хворобу пацієнта за вказаними симптомами (алгоритм може скласти та видати лише приблизний перелік вірогідних хвороб, особливо за малої кількості симптомів у локальній базі даних, тож останнє слово залишається за Вашим живим особистим лікарем);
* надає можливість провести бесіду з лікарем-психологом (емуляція старої програми ELIZA). [5]

4.1.3 Умови застосування програми

До комп’ютеру, на якому виконується програма, висуваються наступні вимоги:

* операційна система Windows 10, Ubuntu 20 04 LTS, macOS;
* наявність дисплею;
* наявність миші та клавіатури;
* до 4 ГБ оперативної пам’яті (для додатку достатньо 100 МБ)

Для налаштування середовища розробки на комп’ютері повинні бути встановлені: бібліотека Qt5 (бажано Long Time Support версія); GCC-сумісний компілятор (GCC або MinGW64): Qt Creator (не є обов’язковим, проте спрощує життя); система контролю версій Git (не є обов’язковою, проте дозволяє гнучко управляти сирцевими даними програмного комплексу, зокрема отримувати останні оновлення з репозитарію на GitHub без потреби завантажувати ZIP архів, а також вести розробку у виділеній гілці).

Для налаштування та підключення бази даних до програмного комплексу необхідно запустити на виконання наступний скрипт:

- Ubuntu 20.04 LTS / macOS: ./db-bootstrap.sh

- Windows 10: ./db-bootstrap.bat

Усі файли класів мови C++, файли графічного інтерфейсу QML, Shel скрипти, файл бази даних Sqlite3, супутня документація Doxygen — розміщенні в корені проекту. Файли SQL для створення бази даних розміщені в ./db-bootstrap-data. Ресурси PNG графічного інтерфейсу розміщені в ./resources/public.

4.2 Характеристика програми

Програма виконана за допомогою мови програмування високого рівня С++ в середовищі розробки Qt Creator за допомогою платформи QML.

4.3 Звертання до програми

Для запуску програми потрібно попередньо упевнитися в відповідності характеристик комп’ютера, на який встановлена програма, до системних вимог програми. Потрібно упевнитися у наявності всіх файлів програми.

Звертання до програми передбачене одним способом: через виконуваний файл ChatBox.exe (Windows 10) або ChatBot.desktop (Ubuntu 20 04 LTS). Запустити можна також через інтерфейс командного рядка: ./ChatBot.exe (Windows 10), ./ChatBox (Ubuntu 20 04 LTS).

4.4 Вхідні та вихідні дані

4.4.1 Вхідні дані

Вхідні дані уявляють собою повідомлення-запити користувача. Для кожного розділу медичного чат-боту вони можуть бути унікальними. Також слід зазначити, що повідомлення-запити мають бути строго сформульованими та тільки англійською мовою.

4.4.2 Вихідні дані

Вихідні дані представляють собою журнал бесіди користувача з відповідним розділом медичного чат-боту. Кожне повідомлення програми є чітко сформульованим та тільки англійською мовою, що є основною мовою застосунку, тож на це слід одразу ж звернути увагу кінцевому користувачеві перед звертанням до застосунку.

5 керівництво користувача

В даному розділі розглянуто призначення програми, умови її виконання, процес виконання програми та повідомлення для користувача.

5.1 Призначення програми

Програма ChatBot призначена для емуляції та демонстрації роботи справжнього медичного чат-боту в режимі офлайн (використовується локальна база даних Sqlite3) в режимі настільного графічного застосунку.

5.2 Умови виконання програми

5.2.1 Апаратні вимоги програми

До комп’ютеру, на якому виконується програма, висуваються наступні вимоги:

* операційна система Windows 10, Ubuntu 20 04 LTS, macOS;
* наявність дисплею;
* наявність миші та клавіатури;
* до 4 ГБ оперативної пам’яті (для додатку достатньо 100 МБ)

5.2.2 Вимоги до користувача

До користувачів висуваються наступні вимоги:

* володіння англійською мовою на початковому або середньому рівні;
* терпіння, бо інколи чат-бот може не зрозуміти користувача через те, що він — лише компʼютер, а не жива людина;
* базовий набір навичок користування ПК в цілому.

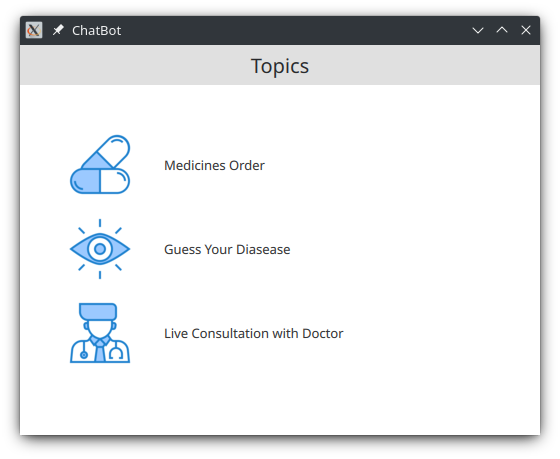
5.3 Виконання програми

5.3.1 Запуск програми

Для запуску програми потрібно попередньо упевнитися в відповідності характеристик комп’ютера, на який встановлена програма, до системних вимог програми. Другим кроком потрібно упевнитися в наявності всіх бібліотек для роботи програми та в наявності бази даних для повноцінного функціонування.

Звертання до програми передбачене одним способом: через виконуваний файл ChatBox.exe (Windows 10) або ChatBot.desktop (Ubuntu 20 04 LTS). Запустити можна також через інтерфейс командного рядка: ./ChatBot.exe (Windows 10), ./ChatBox (Ubuntu 20 04 LTS).

Після запуску виконуваного файлу з’являється головна форма програми, що свідчить про початок роботи з програмою. Головна форма програми зображена на рисунку 5.1.

Рисунок 5.1 – Головна форма програми

5.3.2 Виконання роботи з програмою

Спершу користувач має на головній формі серед запропонованих йому розділів медичного чат-боту обрати той, що необхідний йому на даний момент шляхом натискання лівої клавіші компʼютерної миші на відповідний розділ. Програма одразу ж відобразить форму історію діалогу та форму відправлення повідомлення, що згодом буде неодмінно опрацьоване певним вбудованим алгоритмом чат-боту.

***5.4 Форма діалогу з чат-ботом***

На формі діалогу користувач має змогу побачити та прокрутити знизу-вверх усю наявну в локальній базі даних історію діалогу з чат-ботом. Також знизу є усі необхідні елементи керування для відправлення повідомлення-запиту: поле для введення тексту, яке може динамічно змінювати розмір в залежності від кількості уведених літер, кнопка Send, при натиснені якої запуститься відповідний вбудований в застосунок алгоритм обробки повідомлення-запиту. Зверху форми знаходиться кнопка Back, що дозволяє повернутися до головного меню, якщо користувач закінчив діалог або помилився з вибором розділу чат-боту.

Спілкування з чат-ботом в розділах Medicines Order та Guess Your Diasease має декілька спільних рис:

- спочатку користувач розпочинає діалог: I want to order; I want to start;

- чат-бот надсилає у відповідь підказку стосовно подальшої роботи та надалі чекає від користувача першого медикаменту: Medicine name 10 (де 10 — кількість медикаменту, а Medicine name — його назва);

- на кожному кроці чат-бот просить користувача виправити повідомлення або підтвердити його;

- якщо користувач закінчив додавання медикаментів він сповіщує про це бота: Enough, Done with it;

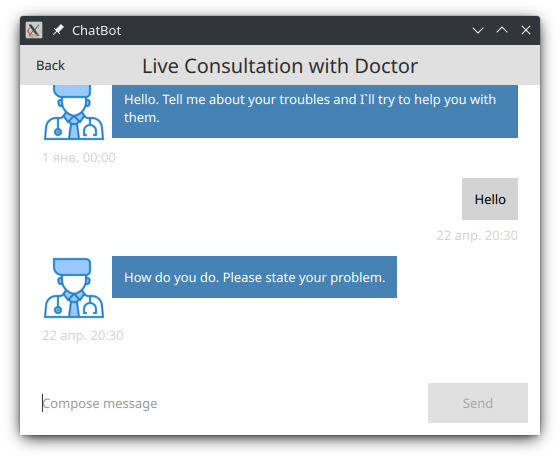
- чат-бот видає повідомлення зі списком аптек та медикаментів;

- користувач пише щось на зразок: Thanks a lot, та фаза діалогу повертається на початок.

Цей порядок дій справедливий і для Guess Your Diasease, окрім запрошення на підтвердження вводу користувача.

У разі спілкування з лікарем-психологом (Live Consultation with Doctor), користувач вільний писати все що завгодно, не дотримуючись обмежень фаз діалогу. Проте користувач має не забувати, що у компʼютера теж є серце, а користувач, перш за все, є людина вихованою та чемною.

Вигляд форми діалогу зображений на рисунку 5.2.

Рисунок 5.2 — Вигляд форми діалогу

Висновки

Під час виконання курсового проекту було розроблено програмне забезпечення для емуляції роботи медичного чат-боту.

Було виконано огляд існуючих методів вирішення завдання та аналогів, проаналізовано переваги та недоліки і розроблено функціонал власного програмного продукту.

Для програми було реалізовано інструментарій для роботи у формі програми-месенджеру, що є звичним та простим для цільової аудиторії програмного продукту та полегшує роботу з ним.

Було розроблено декілька алгоритмів, основними ознаками яких є:

* подача на вхід вислову людською мовою;
* видання користувачеві результату у зручній для нього формі;

Було розроблено максимально прости та зрозумілий інтерфейс, у якому не має зайвих функцій.

Одною з ключових переваг розробленого продукту було закладення засад для подальшого розвитку та модернізації. Зважаючи на актуальність технологій, які були використані, якісне проектування бази даних, алгоритму та інтерфейсу, цей програмним продукт має усі перспективи до подальшої розробки, а саме:

* створення алгоритму, який знаходить найбільш оптимальне рішення;
* синтез інтерфейсу та бази даних, для перетворення програми на повноцінний чат-бот.

ПЕРЕЛІК ПОСИЛАНЬ

1) Mark Allen Weiss, Using a Table Instead of Recursion // Data Structures and Algorithm in C++ (fourth edition). — Florida International University, 2013. –482 – 485p.

2) Компиляторы: принципы, технологии и инструментарий. — Джеффри Ульман, Рави Сети, Моника Лам, 2018. – 1184 c.

3) “What Is SQLite?” [Electronic resource]. – Режим доступу: <https://www.sqlite.org/index.html>.

4) A Book about Qt5. — Github, 2019. – 335 c.

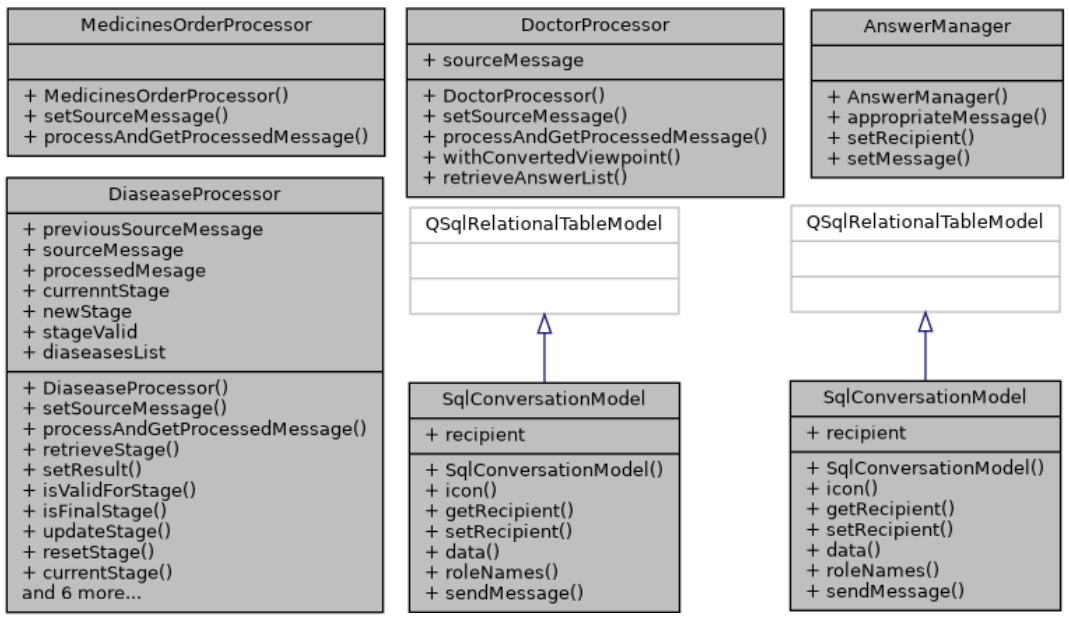
5) Возможности вычислительных машин и человеческий разум. От суждений к вычислениям. — Джозеф Вейценбаум. 1982 – 368 c.

6) «Евристичні алгоритми» [Електрон. ресурс]. – Режим доступу: <http://mmsa.kpi.ua/sancho/ASD_HTM/Arti04.html>.

7) Кормен Т., Лейзерсон Ч., Ривест Р. Введение // Алгоритмы: построение и анализ(третье издание). — М.: «МІТ», 1990 — С. 7—17.

8) «Алгоритм лінійного пошуку» [Електрон. ресурс]. – Режим доступу: <http://kytok.org.ua/?p=311>.

9) Кормен Т., Лейзерсон Ч., Ривест Р. Глава 16. Динамическое программирование // Алгоритмы: построение и анализ (третье издание). — М.: «МІТ», 1990 — С. 299—317.

Додаток А Діаграма класів

Додаток Б Код Програми

#include "medicinesorderprocessor.h"   
  
#include <QDebug>   
#include <QVariant>   
#include <QSqlQuery>   
#include <QSqlError>   
#include <QRegularExpression>   
#include <QRegularExpressionMatch>   
  
// Public API   
  
MedicinesOrderProcessor::MedicinesOrderProcessor() {   
   this->stageValid = false;   
   this->newStage = "S\_FIN";   
}   
  
void MedicinesOrderProcessor::setSourceMessage(QString message) {   
   this->sourceMessage = message;   
}   
  
QString MedicinesOrderProcessor::processAndGetProcessedMessage() {   
   if(!retrieveStage()) return "Sorry, an error occured in my core "   
                               "and I can not remember point where we stopped at.";   
   processMessage();   
   if (!isValidForStage()) return "Sorry, I do not understand you.";   
   else {   
       if(!isFinalStage()) {   
           if(!updateStage()) return "Sorry, an error occured in my core and we can not proceed.";   
       } else if (!resetStage()) return "Sorry, an error occured in my core and we can not begin from scratch.";   
       return this->processedMesage;   
   }   
}   
  
// Internal API   
  
bool MedicinesOrderProcessor::retrieveStage() {   
   QSqlQuery query("SELECT stage FROM Stage WHERE topic = 'medicinesorder'");   
   if(!query.exec()) {   
       qWarning() << "Unable to retrieve stage from DB: " << query.lastError();   
       return false;   
   } else {   
       query.first();   
       this->currenntStage = query.value(0).toString();   
       return true;   
   }   
}   
  
void MedicinesOrderProcessor::setResult(bool valid, QString message, QString stage) {   
   this->stageValid = valid;   
   this->processedMesage = message;   
   if(stage != NULL) this->newStage = stage;   
}   
  
bool MedicinesOrderProcessor::isValidForStage() {   
   return this->stageValid;   
}   
  
bool MedicinesOrderProcessor::isFinalStage() {   
   return this->currenntStage == "S\_FIN";   
}   
  
bool MedicinesOrderProcessor::updateStage() {   
   QSqlQuery query(QString("UPDATE Stage SET stage = '%1' WHERE topic = 'medicinesorder'").arg(this->newStage));   
   if (!query.exec()) {   
       qWarning() << "Unable to update stage in DB: " << query.lastError();   
       return false;   
   } else {   
       return true;   
   }   
}   
  
bool MedicinesOrderProcessor::resetStage() {   
   QSqlQuery query(QString("UPDATE Stage SET stage = 'S\_INIT' WHERE topic = 'medicinesorder'"));   
   if (!query.exec()) {   
       qWarning() << "Unable to reset stage in DB: " << query.lastError();   
       return false;   
   } else {   
       return true;   
   }   
}   
  
bool MedicinesOrderProcessor::currentStage(QString stage) {   
   return this->currenntStage == stage;   
}   
  
bool MedicinesOrderProcessor::containsKeyWords(QStringList keywords) {   
   QRegularExpression re(keywords.join("|"), QRegularExpression::CaseInsensitiveOption);   
   return re.match(this->sourceMessage).hasMatch();   
}   
  
// Specific API   
  
void MedicinesOrderProcessor::processMessage() {   
   if(isMatchingInitInput() && currentStage("S\_INIT")) {   
       // On S\_INIT stage ...   
       setResult(true, "Ok, please send me name of first medicament.\n"   
                       "As example: Medicine Name Amount[P].\n"   
                       "You can edit your requests on any stage if needed.\n"   
                       "When you'll going to finish, just let me know :)", "S\_ADDING");   
       return;   
   } else if(!isMatchingAddingFinishedInput() && currentStage("S\_ADDING")) {   
       // Store current user input because we will use it on the next stage   
       this->previousSourceMessage = this->sourceMessage;   
       // On S\_ADDING stage ...   
       setResult(true, "Sure? Can i remember and process your input?", "S\_ADDING\_VERIFY");   
       return;   
   } else if(currentStage("S\_ADDING\_VERIFY")) {   
       // On S\_ADDING\_VERIFY stage ...   
       if(isMatchingAddingVerifyPositiveAnswer()) {   
           this->processedDrugStringList.append(processUserDrugInfoInput());   
           setResult(true, "Got it! Send me a next medicine.", "S\_ADDING");   
           return;   
       } else {   
           setResult(true, "Don`t worry, just end me a correct item.", "S\_ADDING");   
           return;   
       }   
   } else if(isMatchingAddingFinishedInput() && currentStage("S\_ADDING")) {   
       // On S\_ADDING stage ...   
       setResult(true, "Ok! I`ve processed all your inputs.\n"   
                       "And here is a list of providers "   
                       "which currently provide required "   
                       "medicines for the most lowest cost!\n"   
                 + this->processedDrugStringList.join("\n"), "S\_FIN");   
       return;   
   } else if(isMatchingToBeDoneInput() && currentStage("S\_FIN")) {   
       // On S\_FIN stage ...   
       setResult(true, "Ok. It was so great to help you! See you later!");   
       return;   
   } else return;   
}   
  
QString MedicinesOrderProcessor::processUserDrugInfoInput() {   
   QRegularExpression re("([a-z\\.A-Z\\W?\\d?\\s]+)\\s(\\d+)");   
   QRegularExpressionMatch match = re.match(this->previousSourceMessage);   
   QSqlQuery query(QString("SELECT MedicinesProviders.name, MedicinesItems.name, MIN(Medicines.popc) FROM Medicines "   
                           "INNER JOIN MedicinesProviders ON MedicinesProviders.id = Medicines.medicineprovider\_id "   
                           "INNER JOIN MedicinesItems ON MedicinesItems.id = Medicines.medicine\_id "   
                           "WHERE MedicinesItems.name = '%1'").arg(match.captured(1)));   
   if (!query.exec()) {   
       qWarning() << "Unable to retrieve medicine from DB: " << query.lastError();   
       return QString("Sorry, an error occured in my core "   
                      "and I can not retrieve info about").arg(match.captured(1));   
   }   
   query.first();   
   if (query.value(0).toString() == "") {   
       return QString("Found nothing for %1. Impossible... Maybe my database is incomplete?").arg(match.captured(1));   
   }   
   return QString("%1: (%2 - %3)").arg(   
               query.value(0).toString(),   
               query.value(1).toString(),   
               QString::number(query.value(2).toInt() \* match.captured(2).toInt()));   
}   
  
bool MedicinesOrderProcessor::isMatchingInitInput() {   
   return containsKeyWords(QStringList {"hello", "start", "want", "order"});   
}   
  
bool MedicinesOrderProcessor::isMatchingAddingVerifyPositiveAnswer() {   
   return containsKeyWords(QStringList {"yes", "yep", "yeah", "sure"});   
}   
  
bool MedicinesOrderProcessor::isMatchingAddingFinishedInput() {   
   return containsKeyWords(QStringList {"done", "finished", "enough"});   
}   
  
bool MedicinesOrderProcessor::isMatchingToBeDoneInput() {   
   return containsKeyWords(QStringList {"thank"});   
}