

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Національний університет «Запорізька політехніка»

Кафедра програмних засобів

(найменування кафедри)

**КУРСОВИЙ ПРОЄКТ  
(РОБОТА)**

з дисципліни «Об'єктно-орієнтоване програмування»

(назва дисципліни)

на тему: «Чат-бот пошуку стоматологічних послуг»

Студентів 2 курсу КНТ-219 групи  
спеціальності 122 Комп'ютерні  
науки та технології  
освітня програма (спеціалізація)  
Комп'ютерні науки

Жир Є.С.

(прізвище та ініціали)

Примаков В. В.

(прізвище та ініціали)

Кобяковський О. І.

(прізвище та ініціали)

Пархоменко А. Д.

(прізвище та ініціали)

Керівник професор, к.т.н., Табунщик Г. В.

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

Національна шкала

Кількість балів: Оцінка: ECTS

Члени комісії

Табунщик Г. В.

(підпис)

(прізвище та ініціали)

Каплієнко Т. І.

(підпис)

(прізвище та ініціали)

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**  
**Національний університет «Запорізька політехніка»**  
 (повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ  
 Кафедра програмних засобів  
 Ступінь бакалавр вищої  
 освіти бакалавр  
 Спеціальність 122 Комп'ютерні науки та технології  
 (код і найменування)  
 Освітня програма (спеціалізація) Комп'ютерні науки  
 (назва освітньої програми (спеціалізації))

**ЗАТВЕРДЖУЮ**

Завідувач кафедри ПЗ, д.т.н, проф.  
С.О. Субботін  
 “ ” 20 року

**З А В Д А Н Н Я**

**НА КУРСОВИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)**

Жир Є.С., Примаков В. В., Кобяковський О. І., Пархоменко А. Д.  
 (прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) «Чат-бот пошуку стоматологічних послуг»  
 керівник проєкту (роботи) Табунщик Галина Володимирівна, к.т.н., професор,  
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)  
 затверджені наказом закладу вищої освіти від \_\_\_\_\_

2. Строк подання студентом проєкту (роботи) 22 грудня 2020 року

3. Вихідні дані до проєкту (роботи) створити застосунок «Чат-бот пошуку стоматологічних послуг»

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області; 2. Аналіз програмних засобів; 3. Основні рішення з реалізації компонентів системи; 4. Керівництво програміста; 5. Керівництво користувача; 6. Додатки.

## 6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	професор, к.т.н., Табунщик Г.В.		

7. Дата видачі завдання 20 вересня 2020 р.**КАЛЕНДАРНИЙ ПЛАН**

№ з/п	Назва етапів курсового проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1.	Аналіз індивідуального завдання.	1 тиждень	
2.	Аналіз програмних засобів, що будуть використовуватись в роботі.	2 тиждень	
3.	Аналіз структур даних, що необхідно використати в курсовій роботі.	3 тиждень	
4.	Затвердження завдання	4 тиждень	
5.	Вивчення можливостей програмної реалізації структур даних та інтерфейсу користувача.	5-9 тиждень	
6.	Розробка програмного забезпечення	9-13 тиждень	
7.	Оформлення, відповідних пунктів пояснювальної записки.	10-14 тиждень	Розділи 1-5 ПЗ
9.	Захист курсової роботи.	15 тиждень	

Студент \_\_\_\_\_ Жир Є.С.  
 ( підпис ) (прізвище та ініціали)

Студент \_\_\_\_\_ Примаков В. В.  
 ( підпис ) (прізвище та ініціали)

Студент \_\_\_\_\_ Кобяковський О. І.  
 ( підпис ) (прізвище та ініціали)

Студент \_\_\_\_\_ Пархоменко А. Д.  
 ( підпис ) (прізвище та ініціали)

Керівник проєкту (роботи) \_\_\_\_\_ Табунщик Г. В.  
 ( підпис ) (прізвище та ініціали)

## РЕФЕРАТ

Об'єкт дослідження – процес створення програмних додатків з використанням ООП.

Предмет дослідження – чат-бот.

Мета роботи – покращення пошуку інформації стоматологічних послуг за рахунок чат-боту.

Ключові слова – чат-бот, графічний інтерфейс, база даних, алгоритми, виняткові ситуації, JSON, QT, C++, стоматологічні послуги.

У даній роботі проведено дослідження: структури чат-боту та подібних аналогів, середовища розробки на прикладі IDE QT Creator, структури стоматологічних послуг, актуальність використання чат-боту , ієрархії класів інтерфейсу, зберігання даних у зовнішніх файлах, контейнерів в середовищі QT.

## ЗМІСТ

	C.
Перелік скорочень та умовних познач	7
Вступ	9
1 Аналіз предметної області	8
1.1 Аналоги	10
1.2 Чат-бот	11
1.3 Висновки та постановка задачі	12
2 Аналіз програмних засобів	15
2.1 Вибір мови програмування	15
2.2 Вибір середовища розробки	16
3 Основні рішення з реалізації компонентів системи	18
3.1 Інтерфейс	18
3.2 Робота за даними	26
3.3 Алгоритми та структури даних	29
3.4 Виключні ситуації	35
4 Керівництво програміста	40
5 Керівництво користувача	44
Висновки	48

Перелік джерел посилання	49
Додаток А Текст програми	50

## ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

Qt Creator – інтегроване середовище розробки, призначене для створення крос-платформових застосунків з використанням бібліотеки Qt;

JSON (JavaScript Object Notatio) - простий формат обміну даними, зручний для читання і написання як людиною, так і комп'ютером;

QList – один із загальних класів контейнерів Qt. Він зберігає елементи у списку, що забезпечує швидкий доступ на основі індексу та вставки та видалення на основі індексу;

ООП – об'єктно-орієнтоване програмування;

STL (Standard Template Library) – стандартна бібліотека шаблонів в мові програмування C ++;

QtWidgets – містить класи для додатків на основі віджетів, модуль виділений з QtGui в Qt 5;

QString – є рядком символів Unicode. QString зберігає рядок 16-бітних QChar, де кожен QChar зберігає символ Unicode 4.0;

Unicode – це міжнародний стандарт, який підтримує більшість існуючих на сьогодні писемних систем;

ASCII (American standard code for information interchange) - назва таблиці (кодування, набору), в якій деяким поширеним друкованим і недрукованим символів зіставлені числові коди;

Latin-1 – кодова сторінка, призначена для західноєвропейських мов;

Set-ери – метод, який використовується в об'єктно-орієнтованому програмуванні для того, щоб привласнити яке-небудь значення інкапсульованому полю;

Get-ери – спеціальний метод, що дозволяє отримати дані, доступ до яких безпосередньо обмежений.

SQLite — полегшена реляційна система керування базами даних.



## ВСТУП

Сьогодні чат-боти пішли далеко вперед, розширивши список сфер для їх застосування. Чат-боти можуть бути використані як для розваг, так і для бізнесу, наприклад, консультування клієнта по продукту. Поспілкуватися з чат-ботом можна 24/7, тому вам не обов'язково наймати співробітників для роботи в нічні зміни. Простіше і вигідніше створити чат-бот, який надасть максимум корисної інформації яка для клієнта.

Ціль роботи – створити чат-бота для зручного пошуку стоматологічних послуг на основі аналізу аналогічних сервісів.

У роботі необхідно виконати наступні задачі для створення чат-боту:

- аналіз предметної області;
- розробити відповідні структури даних;
- створити візуальний інтерфейс;
- розробити програму.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Аналоги

Головне завдання програми - пошук стоматологів за певними критеріями. Перед розробкою потрібно подивитися структуру аналогічних програм. Як аналог було розглянуто веб-сервіс "helsi.me" (рис. 1.1) виходячи з якого були підчеркнуті майбутні поля для класів.

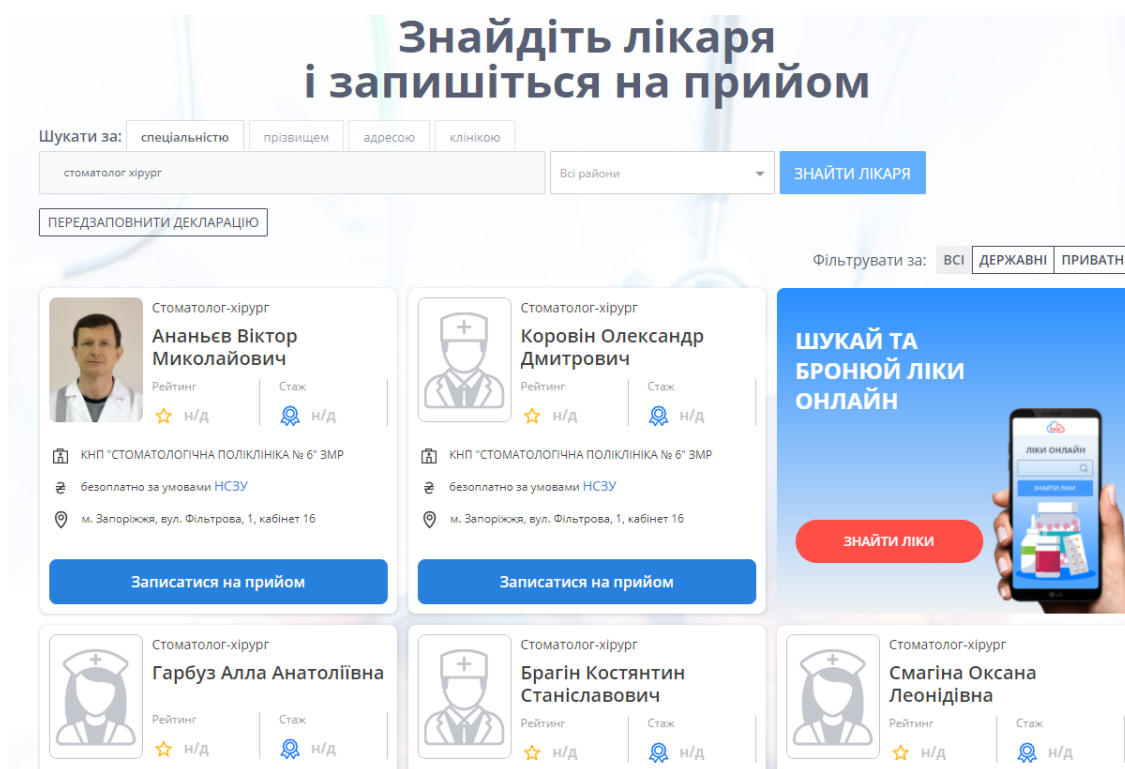


Рисунок 1.1 - Аналог – веб-сервіс пошуку стоматологів

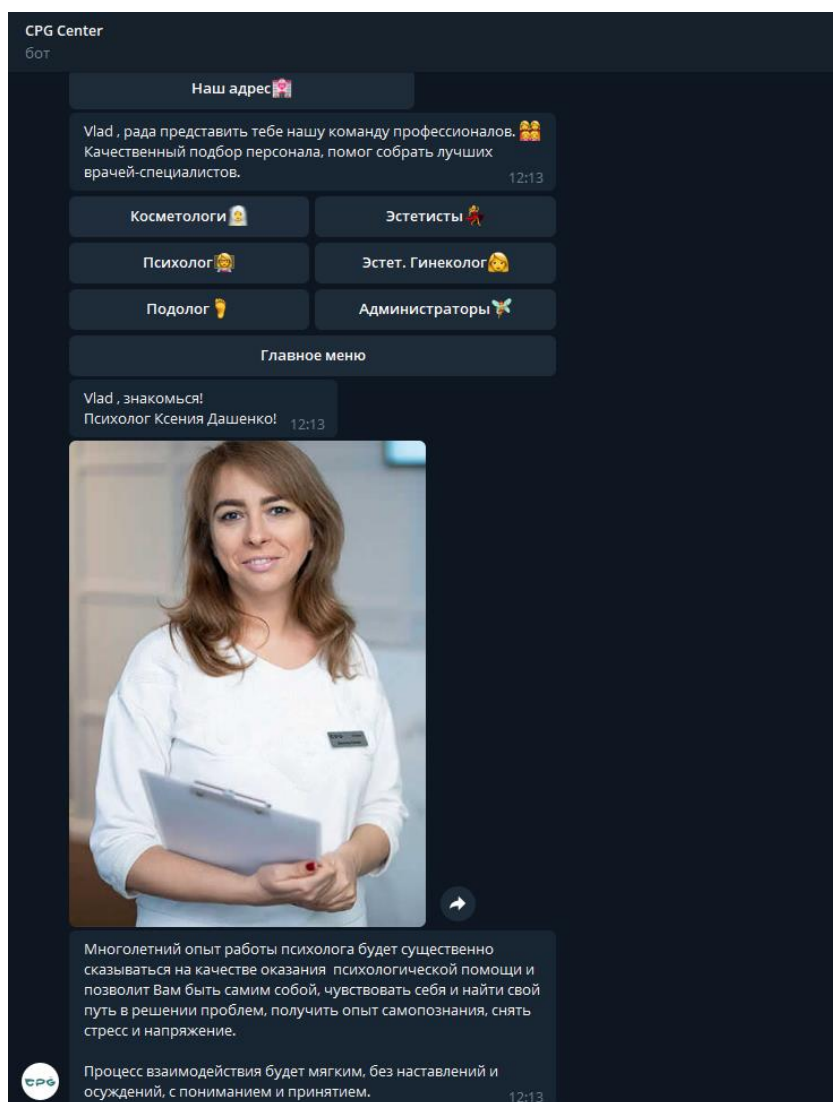


Рисунок 2.2 - Аналог чат-боту телеграму медичного сервісу CPG

Ще один аналог, який було розглянуто - чат-боту телеграму медичного сервісу CPG (рис 1.2) з якого стало зрозуміло, що потрібно поділити лікарів на спеціалізації та створити зручне меню для пошуку всіх лікарів за певним критерієм.

## 1.2 Чат-бот

Чат-бот - це комп'ютерна програма, фактично віртуальний співрозмовник, який працює на основі встановлених правил і алгоритмів. Він

вміє імітувати людську поведінку під час спілкування з потенційним клієнтом. В сучасному світі головна задача чат-боту зекономити час як клієнта та замовника. Чат-бот полегшує навігацію користувача в певній галузі. Щоб кожний раз не відповідати користувачу на поширене питання, бот зможе це робити моментально. Також бот може повідомляти користувача про важливі новини, тощо.

Види чат-ботів на сайтах і в месенджерах:

- Чат-боти - персональні помічники
- Чат-боти з функцією розваги
- Чат-боти з технологією штучного інтелекту
- Чат-боти для бізнесу та маркетингових операцій

Остання в списку група - найцікавіша і активно розвивається. Інтернет-маркетинг бачить в чат-ботах можливість для бізнесу додатково автоматизувати етапи роботи з клієнтом, підвищити лояльність споживача, перевести частину елементарних завдань на «віртуальних продавців», отримуючи прямі конверсії.[1]

### **1.3 Висновки та постановка задачі**

Існують стоматологічні клініки до яких закріплені стоматологи, стоматологи мають свою спеціалізацію яка займається певними стоматологічними послугами. У свою чергу стоматологи повинні мати такі обов'язкові поля дані:

- ПІБ;
- спеціалізація;

- вік;
- стаж роботи;
- фото;
- контакти;
- стоматологія.

Також ми додали додаткові поля:

- про дантиста;
- коментарі;
- рейтинг.

Рейтинг підраховується виходячи з коментарів. Буде потрібно додатковий клас Коментар який має поля:

- ім'я відправника;
- сам коментар;
- оцінка.

Обов'язковою класом є стоматологія яка має такі дані:

- назва стоматології;
- адреса;
- робочий час;
- контакти;
- про стоматологію.

Виходячи з цих класів для компактного зберігання даних потрібно створити поля у стоматологів і стоматологій - ID стоматології, завдяки якому стоматолог буде прив'язаний до певної стоматології. А коментарі будуть прив'язані безпосередньо до певного стоматолога.

Значить функціонал програми повинен бути таким:

- 1) вміти шукати стоматологів за критеріями:
  - прізвище;
  - стоматологія;
  - спеціальність.
- 2) залишати коментарі до стоматолога;
- 3) автоматично підраховувати рейтинг лікаря виходячи з коментарів;
- 4) виведення всіх лікарів за певним критерієм;
- 5) сортування знайдених лікарів за рейтингом і за алфавітом.

Оскільки головне завдання додатку - пошук стоматологів значить при запуску програми повинна відкриватися форма в якій вже можна шукати стоматологів. Список знайдених стоматологів повинен виводиться окремим блоком і на головній формі повинна бути можливість вибору сортування і за яким критерієм буде працювати пошук і звичайно ж текстове поле для введення критерію (наприклад прізвища стоматолога). Якщо користувач не може визначитися з пошуком повинна бути функція виведення всіх стоматологів конкретної спеціалізації або стоматології.

Обов'язкова повинна бути форма "Допомога" в якій описується функціональність програми і як з нею працювати. Оскільки лікарі мають відносно багато даних, то при виводі списку лікарів повинні відображатися тільки основні поля для зручного перегляду для користувача. Для кожного знайденого лікаря повинна бути добавлена форма в якій буде виводиться повна інформація про стоматолога і стоматології в якій він працює.

Для зручності у кожного лікаря буде форма "Коментарі" в якій користувач зможе дивитися і додавати коментарі конкретного стоматолога.

## **2 АНАЛІЗ ПРОГРАМНИХ ЗАСОБІВ**

### **2.1 Вибір мови програмування**

На початку 80-х років минулого століття співробітник Bell Labs Бйорн Страуструп після довгих мук з існуючими мовами програмування провів експеримент зі схрещуванням C і Simula. Він навіть не розраховував, що його дітище, яке отримало назву C ++, приверне стільки уваги. Однак тоді мова викликав справжній фурор: компільований, структурований, об'єктно-орієнтована, неймовірно спрощує роботу з великими програмами і при цьому має величезний потенціал для розвитку. Такий, що ще майже десятиліття знадобилося Страуструпом, щоб наділити C ++ усіма характерними особливостями. Розвиток же триває досі.

Швидкість виконання коду - мабуть, головний аргумент на користь того, чому C ++ був, є і буде затребуваний в ІТ. На противагу цьому часто ставиться швидкість написання коду, яка, наприклад, у різних мов на порядок вище. У цьому є частка істини - C #, Java і звичайно ж Python навіть візуально займають менше місця, з їх допомогою можна створювати складні програми, витративши мінімум часу.

Він універсальний. Компілятори C ++ є на кожній операційній системі, більшість програм легко переноситься з платформи на платформу, з середовищем розробки і бібліотеками у вас точно не виникне проблем. C ++ - це демонстрація ідеї класичного програмування, коли 90% думок пов'язано з кодом і лише 10% з використовуваної периферією.[2]

### **2.2 Вибір середовища розробки**

Основне завдання Qt Creator - спростити розробку програми за допомогою фреймворка Qt на різних платформах. Тому серед можливостей, властивих будь-якому середовищі розробки, є вбудований дизайнер інтерфейсів на QtWidgets.[3]

Бібліотека Qt надає набір контейнерних класів загального призначення заснованих на шаблонах. Ці класи можуть використовуватися для зберігання елементів певного типу.

Ці контейнерні класи розроблені для легкого, безпечного і простого використання замість контейнерів STL.[6]

Серед контейнерів для зберігання класів "Стоматолог" і "Стоматологія" використовується контейнер QList оскільки під час роботи програми постійно будуть додаватися елементи в структуру, то QList підходить для оптимізації додатку оскільки він миттєво може додати елемент в контейнер на відміну від інших контейнерів, що значно прискорює працездатність програми.

Мова C ++ надає два типи рядків: традиційні рядки мови C - масиви символів, що завершуються символом '\ 0' і клас string. Qt надає набагато більш потужний клас QString. Він призначений для зберігання рядків з 16-ти бітними символами Unicode. Unicode містить набори символів ASCII і Latin-1 з їх звичайними числовими значеннями. Але оскільки кожен символ в QString представлений 16-ма бітами, він може містити тисячі інших символів. Завдяки використанню класу QString можна зручно працювати з інтерфейсними класами QT, а так само створити багатомовний додаток в якому алгоритми будуть коректно працювати завдяки Unicode.[7]

Клас QWidget надає базову можливість для відтворення на екрані і для обробки подій для користувача введення. Всі елементи призначеного для



користувача інтерфейсу, що надаються Qt, є підкласами QWidget або використовуються в поєднанні з підкласом QWidget. Створення призначених для користувача віджетів виконується успадкуванням від QWidget або відповідного підкласу і перевизначення віртуальних обробників подій. Компонування - елегантний і гнучкий спосіб для автоматичного розміщення дочірніх віджетів всередині контейнера. Кожен віджет повідомляє компоувальнику свої вимоги до розміру за допомогою властивостей `sizeHint` і `sizePolicy`, а компоновщик відповідно розподіляє доступний обсяг простіру.[8] За допомогою підкласів QWidget і компоновань буде реалізований графічний інтерфейс і функціональність програми.

## 3 ОСНОВНІ РІШЕННЯ З РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

### 3.1 Інтерфейс

Прийнято рішення, щоб всі функції стосовно пошуку стоматологів розміщувалися на головній формі, а також блок в якому виводиться сам список. У кожного знайденого лікаря потрібно щоб створювалися дві кнопки – “Детальніше” та “Коментарі” при натисканні на які будуть відкриватися відповідні форми і в них передаватися дані про лікаря у якого були нажаті ці кнопки. У формі “Детальніше” виводиться повна інформація про лікаря та лікарню в якій він працює. У формі “Коментарі” виводяться всі коментарі конкретного лікаря та форма в якій можливо написати та відправити коментар. В блоці `menubar` знаходитимуться три меню-вкладки – “Меню” “Мова” “Показати всі”. В першій вкладці можливо відкрити форму “Допомога” та “Розробники”, де відповідно виведеться коротка інформація про програму та таблиця зі списком спеціалізацій стоматологів та послуг, які відносяться до певної спеціалізації. У другій вкладці можливо вибрати мову. А в третій вивести всіх стоматологів, стоматологій та спеціалізацій.

В усій програмі використовувалося два стилі елементів:

1) Перший стиль:

- `color: black;`
- `background-color: rgb(199, 199, 199);`
- `font: 14pt "Arial".`

2) Другий стиль:

- `background-color: rgb(72, 60, 72);`

- font: 12pt "Arial";
- color: white.

Перший стиль можливо побачити в блоці повідомлень для користувача на рис. 3.1, а другий в QLabel (синій колір на рис. 3.2). Використання двох стилів дає можливість створити цікаве рішення щодо дизайну програми.

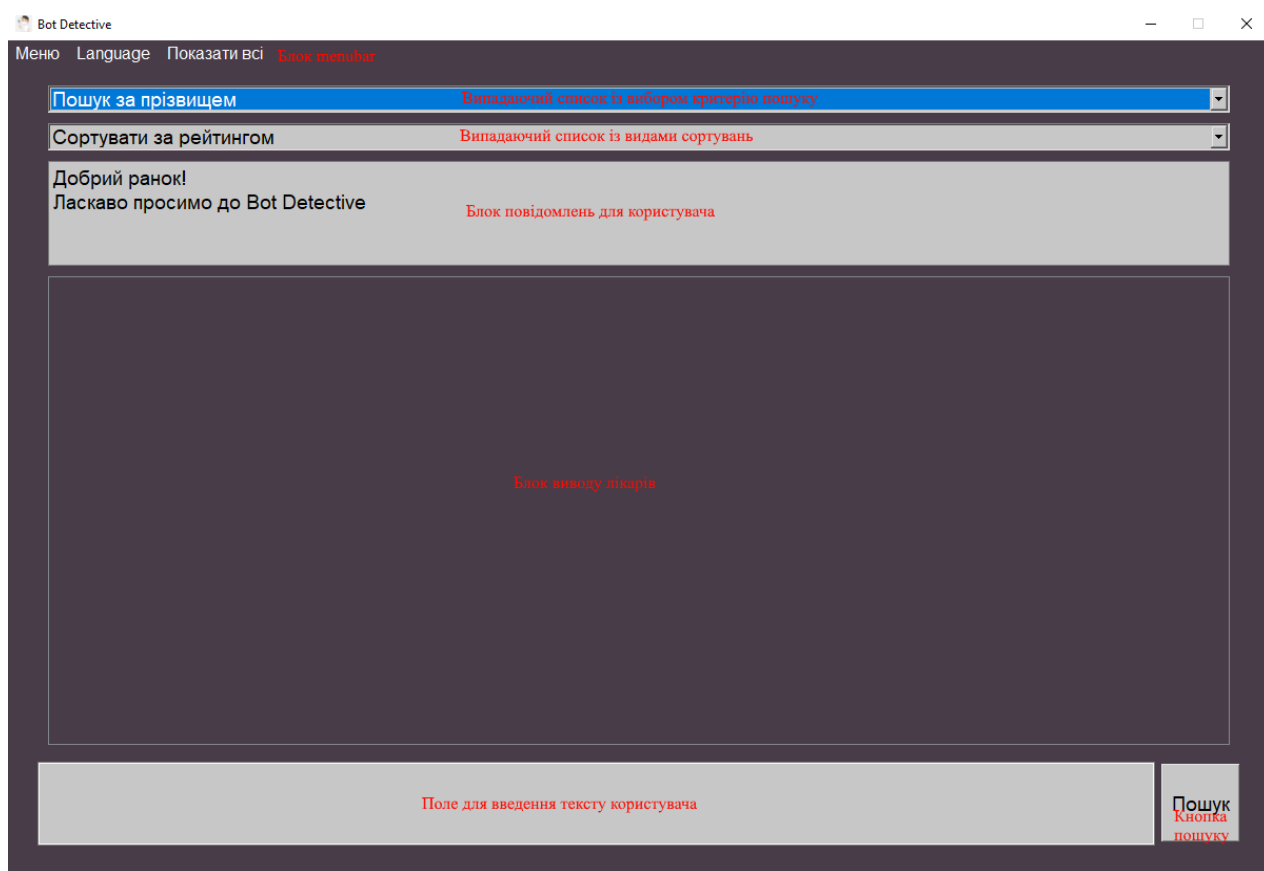


Рисунок 3.1 - Головна форма програми

На рис. 3.1 зображується структура головної форми програми. Випадаючі списки реалізовані за допомогою класу QComboBox. Використовуються його методи для додавання елементів до списку, повернення номеру вибраного елементу, як раз за допомогою цього методу в алгоритмах буде знаходитися за

яким критерієм шукати та як сортувати список стоматологів. А також при зміні мови використовується метод очистки випадваючого списку. В блоці повідомлень використовується клас QTextBrowser, який дозволяє додавати до нього інформацію у вигляді рядків та редагувати текст за допомогою розмітки HTML та CSS-стилів. Блок виводу лікарів – це клас QScrollArea, який дозволяє встановлювати компонування та віджети, на основі цих методів створена функція для виводу стоматологів. Поле для введення тексту користувача – це клас QLineEdit, який дозволяє вводити текст та передавати його до програми. І звичайна кнопка QPushButton, яка має сигнал clicked і дозволяє виконувати функції пошуку в програмі. Блок menubar – це клас QMenuBar, який розділений на класи QMenu в яких знаходяться класи QAction, які вже дозволяються за допомогою сигналу triggered викликати певні функції.

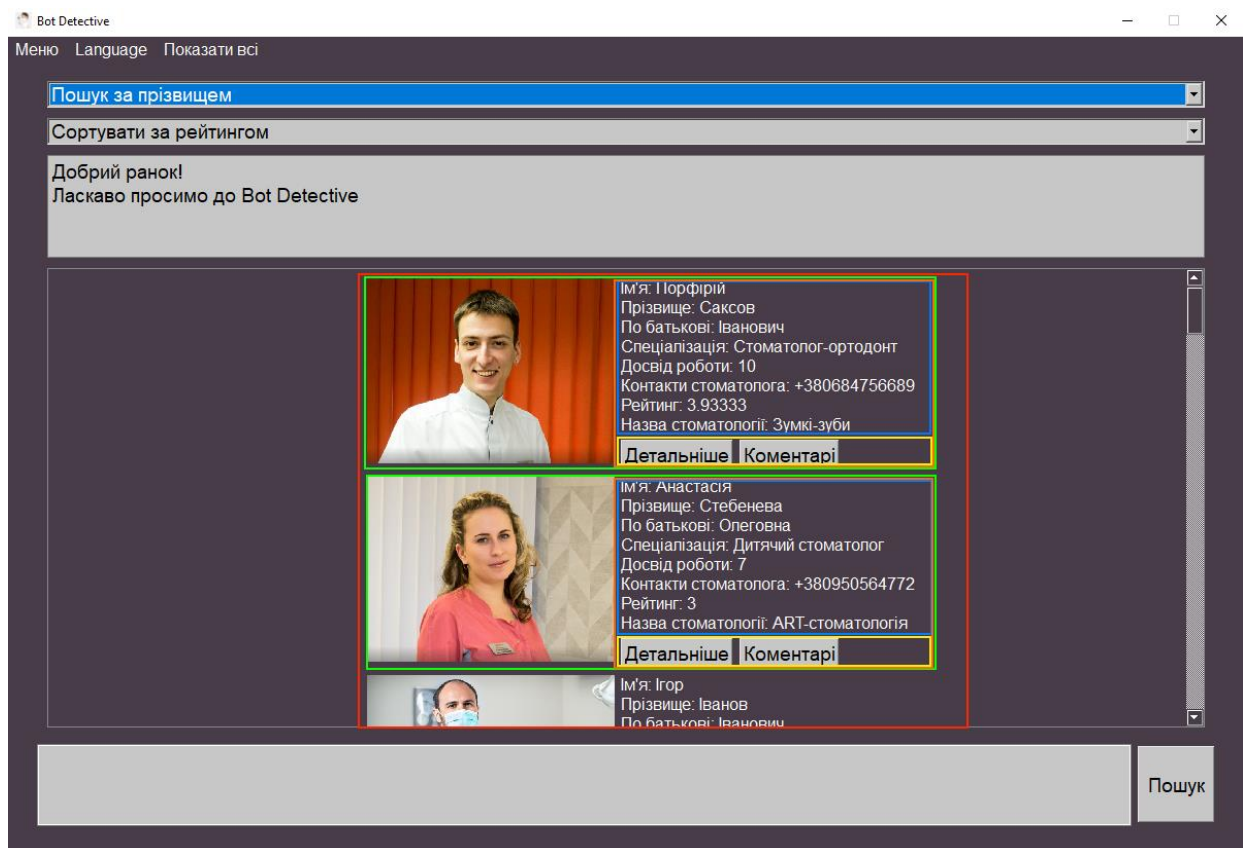


Рисунок 3.2 - Створення списку стоматологів


На рис. 3.2 схематично зображено, як створюється список знайдених лікарів. Спочатку до методу пересилається `QList<QDentist* >`, створюється новий `QWidget` та головне компонування по вертикалі `QVBoxLayout` (червоний колір на рис. 3.2). Потім в циклі для кожного лікаря створюється компонування по горизонталі `QHBoxLayout` (зелений колір на рис. 3.2) до якого додається `QLabel` в якому зберігається фото лікаря, яке створюється за допомогою класу `QPixmap` і встановлюється в `QLabel` за допомогою методу `setPixmap`. Додатково створюється компонування по вертикалі `QVBoxLayout` (помаранчевий колір на рис. 3.2), яке додається до зовнішнього `QHBoxLayout` (зелений колір на рис. 3.2), створюється `QLabel` (синій колір на рис. 3.2) до якого заносяться дані про лікаря і додається до зовнішнього `QVBoxLayout` (помаранчевий колір на рис. 3.2). Створюється компонування по горизонталі `QHBoxLayout` (жовтий колір на рис. 3.2) до якого додаються дві кнопки “Детальніше” та “Коментарі”, ці кнопки з’єднуються з методом для посилення лікаря до нової форми:


```

– connect(detailed, &QPushButton::clicked, [this, tmp]() { this-
>SendDetailed(tmp); });
– connect(comments, &QPushButton::clicked, [this, tmp]() { this-
>SendComments(tmp); });

```

`QHBoxLayout` (жовтий колір на рис. 3.2) додається до зовнішнього `QVBoxLayout` (помаранчевий колір на рис. 3.2). На цьому етапі створено `QHBoxLayout` (зелений колір на рис. 3.2) для лікаря і це компонування додається до головного `QVBoxLayout` (червоний колір на рис. 3.2) і так для кожного лікаря в списку. Після відпрацювання циклу для кожного лікаря в `QWidget` встановлюється головне компонування `QHBoxLayout` (зелений колір на рис. 3.2). І вже в кінці до блоку виводу лікарів (рис 3.1) встановлюється створений віджет.

 Detailed Info — □ ×



Ім'я: Порфірій
Прізвище: Саксов
По батькові: Іванович
Вік: 40
Спеціалізація: Стоматолог-ортодонт
Про стоматолога: вузький спеціаліст
Контакти стоматолога: +380684756689
Рейтинг: 3.93333
Досвід роботи: 10

Назва стоматології: Зумкі-зуби

Адреса стоматології: Запоріжжя, вул.Шкільна,32

Робочі години: 8:00-20:00

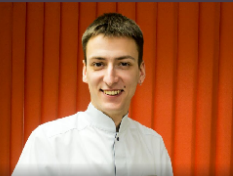
Про нас: This is family dentistry.

Контакти стоматології: zubki-zubi@ukr.net

Рисунок 3.3 – Форма “Детальніше”

На рис. 3.3 показана форма “Детальніше”, яка відкривається під час натискання на кнопку “Детальніше” на рис.3.2. В цій формі за допомогою компоновань та QTextBrowser-ів виводиться вся інформація про лікаря та стоматологію в якій він працює.

Comments of dentist



Ім'я: Порфірій

Прізвище: Саксов

По батькові: Іванович

Спеціалізація: Стоматолог-ортодонт

Денис Сухачев

Кращий

4.1

Влад Примаков

Дуже добре

4.3

Филипп

Ім'я

Коментар

Рейтинг ([0,5])

Додати коментар

Рисунок 3.4 – Форма “Коментарі”

В цій формі виводиться ПІБ та спеціалізація лікаря, а також за аналогією створення списку стоматологів створюється список коментарів в QScrollArea а також форма в якій можна залишити коментар для конкретного лікаря.

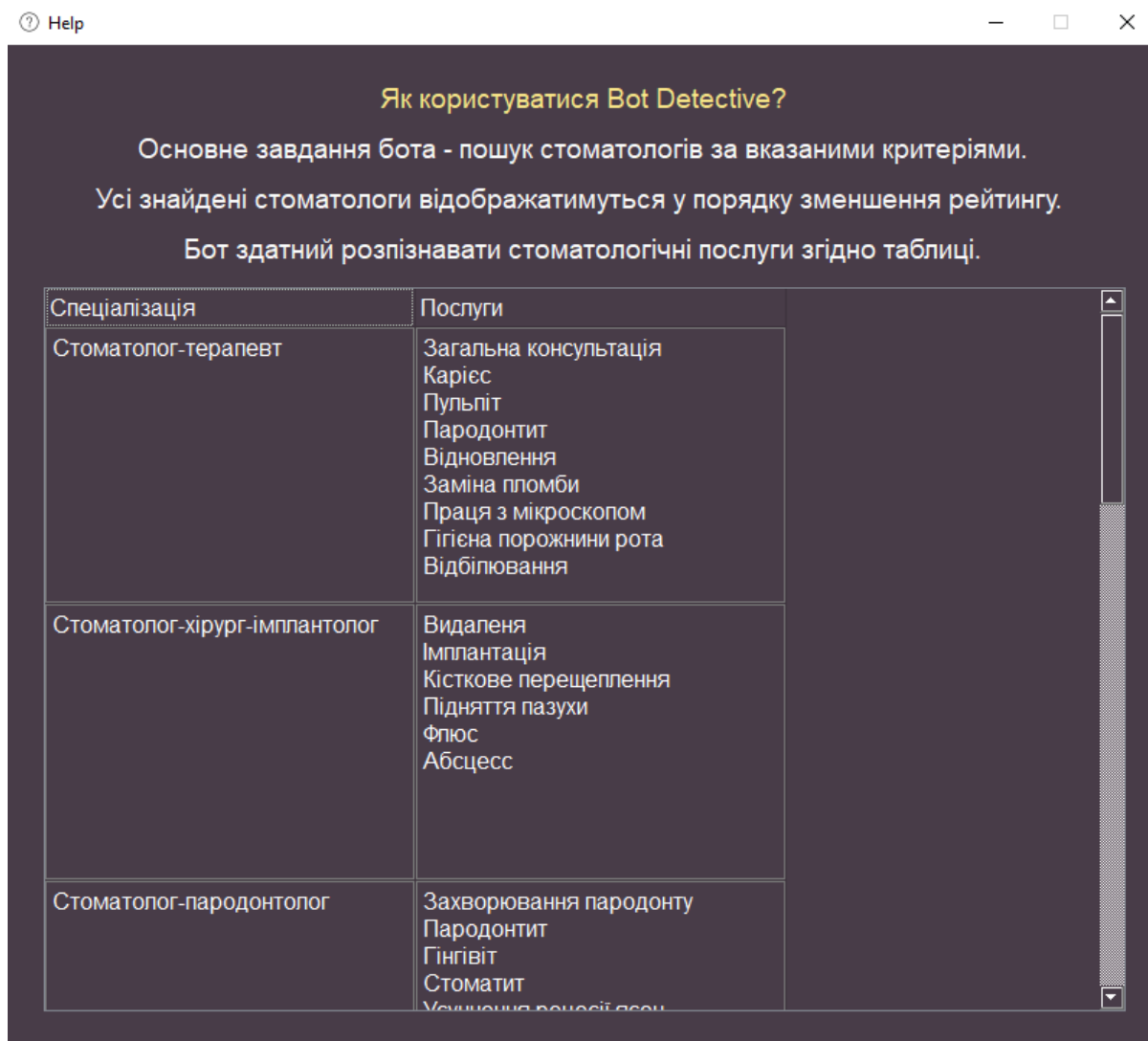


Рисунок 3.5 – Форма “Допомога”

В блоці menubar на рис. 3.1 можна відкрити форму “Допомога” в якій знаходиться інформація про користування програмою та таблиця зі спеціалізаціями та відповідними їм послугами, за якими можливий пошук лікарів.



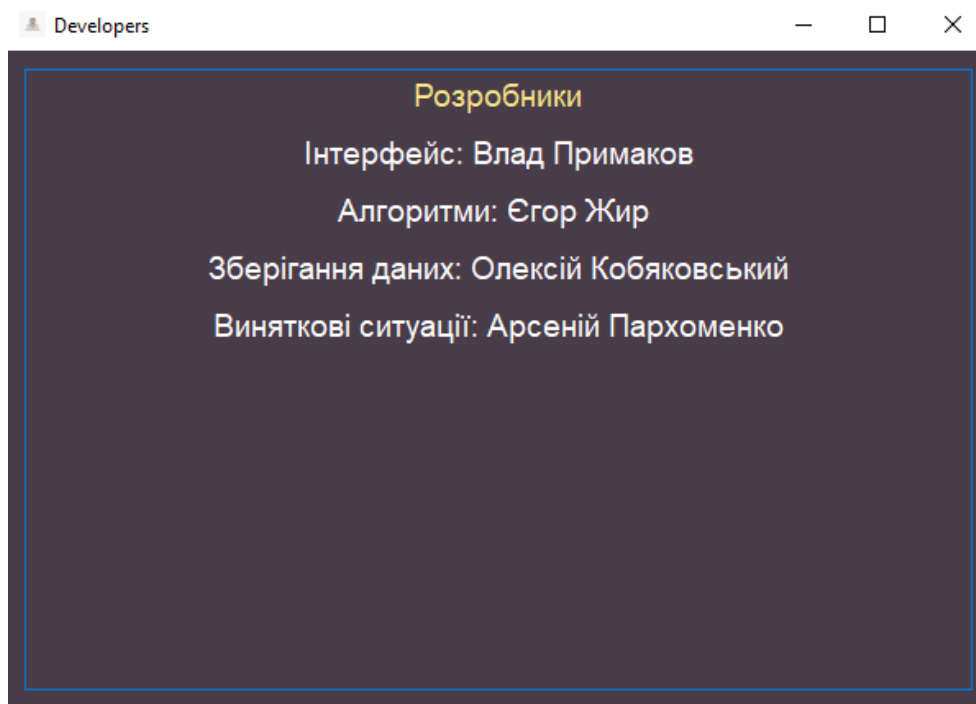


Рисунок 3.6 – Форма “Розробники”

В тому ж блоці менубар на рис. 3.1 можна відкрити форму “Розробники” в якій виводяться безпосередньо розробники цієї програми

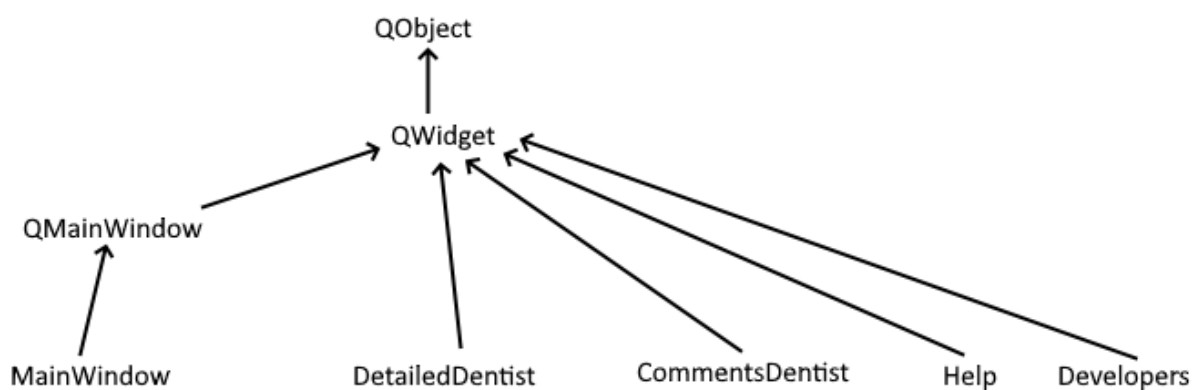


Рисунок 3.7 – Ієрархія класів інтерфейсу в програмі

На рис. 3.7 зображено ієрархію створених класів для відображення графічного інтерфейсу. Оскільки в класі `QMainWindow` реалізований `menubar` за замовчуванням то цей клас використовується для головної форми програми. А оскільки в інших формах не потрібно використовувати додаткових ускладнень класи для форм “Детальніше” “Коментарі” “Допомога” “Розробники” успадковуються від `QWidget`.

### 3.2 Робота з даними

JSON (англ. JavaScript Object Notation, укр. Запис об'єктів *JavaScript*, вимовляється *джейсон*) — це текстовий формат обміну даними між комп'ютерами.

Ми обрали цей формат, тому що з ним зручно працювати в IDE Qt Creator. Qt Creator має спеціальні класи для JSON, такі як: `QJsonObject`, `QJsonParseError`, `QJsonArray`. Ми написали файли JSON, на двох мовах: українській та англійській. На рисунку 3.8, файли з закінченням `eng` написані на англійській, а `ukr` – на українській.

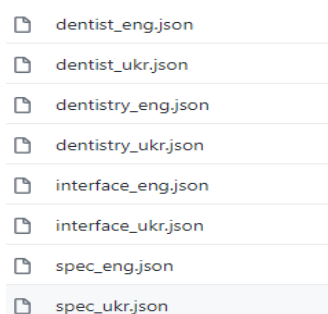


Рисунок 3.8 - каталог JSON-файлів

Структура JSON-файлів.

На рисунку 3.9, приклад одного JSON-об'єкту в масиві. Дані в JSON записуються таким чином: "key" : "value". Цифрою 2 позначені ключі, 1 і 3 –значення. 1-початок масиву.

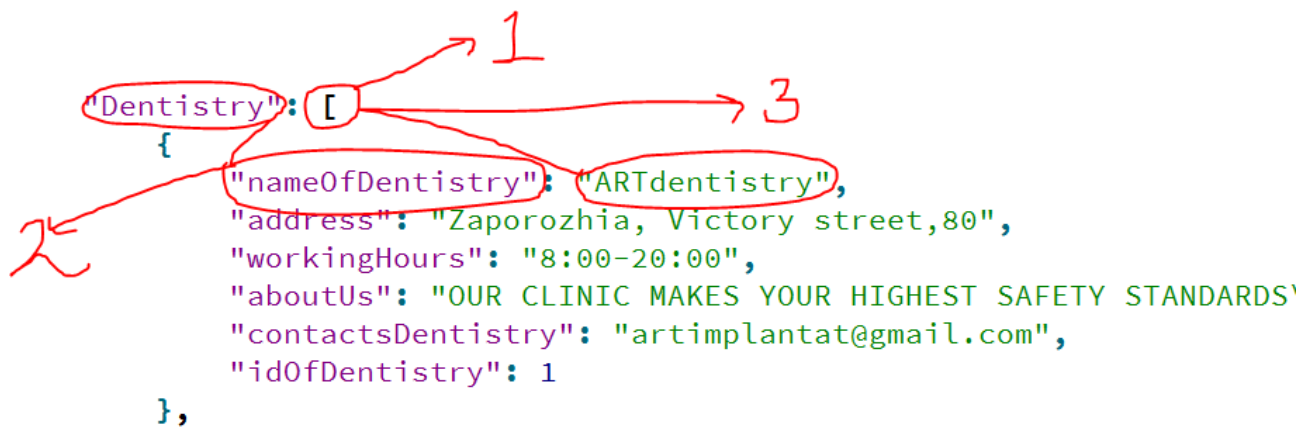


Рисунок 3.9 - Структура JSON-файлів

На рисунку 3.10, робота з даними в проєкті ведеться за допомогою ключів. Знизу приклад.

```
QString Dentistry::getNameOfDentistry() {return this->nameOfDentistry;}
```

Рисунок 3.10 –приклад роботи з ключами в C++

На рисунку 3.11, рейтинг встановлюються з урахуванням коментарів, нижче фрагмент коду.

```

Comments * tmp = new Comments();
tmp->setCommentatorName(dateCommentsArr.at(k).toObject().value("commentatorName").toString());
tmp->setComment(dateCommentsArr.at(k).toObject().value("comment").toString());
tmp->setRating(dateCommentsArr.at(k).toObject().value("rating").toDouble());
tempRating += dateCommentsArr.at(k).toObject().value("rating").toDouble();
temp->addListComments(*tmp);

```

Рисунок 3.11 – фрагмент коду, де через коментарі ведеться підрахування рейтингу.

SQLite – полегшена реляційна система керування базами даних.

На рисунку 3.12, приклад одного SQLite-об'єкту в масиві.

	nameOfDentistry	address	workingHours	aboutUs	contactsDentistry	idOfDentistry
	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр
1	ARTdentistry	Zaporozhia, Victory street,80	8:00-20:00	OUR CLINIC MAKES YOUR HIGHEST ...	artimplantat@gmail.com	1
2	SMILE	Zaporozhia,Alesandrovskaya street,30	8:00-20:00	A complex approach!\nWe consider t...	info@studio-smile.com.ua	2
3	zubki-zubi	Zaporozhia,School street,32	8:00-20:00	This is family dentistry.	zubki-zubi@ukr.net	3
4	Vidnova	Zaporozhia,Peace street,13	9:00-21:00	Vidnova dentistry has a large staff of ...	info@vidnova.dental	4

Рисунок 3.12 - структура таблиці в SQLite

На рисунку 3.13, приклад підключення бази даних та зчитування даних.

```

162 void MainWindow::OpenDentistry(){
163     try{
164         SQLiteDatabase fileDentistry;
165         fileDentistry = SQLiteDatabase::addDatabase("QSQLITE");
166         fileDentistry.setDatabaseName(pathFileDentistry);
167         if(!fileDentistry.open())
168         {
169             throw 6;
170         }
171     }
172     else
173     {
174         QSqlQuery query;
175         query.exec("SELECT nameOfDentistry, address, workingHours, aboutUs, contactsDentistry, idOfDentistry FROM Dentistry");
176         listDentistry.clear();
177         while (query.next())
178         {
179             Dentistry* temp = new Dentistry();
180             temp->setNameOfDentistry(query.value(0).toString());
181             temp->setAddress(query.value(1).toString());
182             temp->setWorkingHours(query.value(2).toString());
183             temp->setAboutUs(query.value(3).toString());
184             temp->setContactsDentistry(query.value(4).toString());
185             temp->setIdOfDentistry(query.value(5).toInt());
186             listDentistry.append(temp);
187         }
188     }
189     fileDentistry.close();
190 }
191 catch(int error){
192     MyException ex(error, interface);
193 }
194 }

```

Рисунок 3.13 - підключення бази даних та зчитування даних

### 3.3 Алгоритми та структури даних

Клас Dentistry:

У класі Dentistry є такі поля:

- QString nameOfDentistry;
- QString address;
- QString workingHours;
- QString aboutUs;
- QString contactsDentistry;
- int idOfDentistry; //ми використовуємо це поле для знаходження

клініки лікаря.

У класі Dentistry є такі методи:

- Set-ери для кожного поля;
- Get-ери для кожного поля.

Клас Dentist:

У класі Dentist є такі поля:

– Dentistry dentistry; // ми використовуємо структуру Dentistry для зберігання даних про стоматологію.

- QString firstName;
- QString lastName;
- QString patronymic;
- QString specialization;
- QString aboutDentist;
- QString contactsDentist;
- int years;

- double rating;
- int workExperience;
- int idOfDentistry; // ми використовуємо це поле для знаходження клініки лікаря.

- QString photo; //ми використовуємо це поле для зберігання назви фотографії,

наприклад: “dentist.png”.

- QList<Comments> listComments; // ми використовуємо QList для зберігання списку коментарів типу Comments

У класі Dentist є такі методи:

- Set-ери для кожного поля;
- Get-ери для кожного поля.

Клас Comments:

У класі Comments є такі поля:

- QString commentatorName;
- QString comment;
- double rating;

У класі Comments є такі методи:

- Set-ери для кожного поля;
- Get-ери для кожного поля.

Хеш-таблиці:

При зчитуванні даних з JSON ми додаємо їх в хеш-таблицю. Хеш-таблиця існує для швидкого пошуку за ключовими словами. Існує пошук (рис. 3.14)

лікаря за прізвищем, спеціалізації та клініки. Так само є алгоритм знаходження відповідного лікаря за симптомами.

При виведенні даних на екран ми маємо можливість вибрати варіант сортування (за рейтингом, за алфавітом, проти алфавіту). У вікні виводу для нас відкритий вибір дій: докладніше про лікаря і залишити коментар. З вибірки коментарів складається рейтинг для кожного стоматолога.

Існують декілька хеш-таблиць:

- HashTableByName;
- HashTableByNameOfDentistry;
- HashTableBySpecialization;

Усі вони використовують відкриту адресацію. Вони в основному схожі, але їх пошук відрізняються, через які не можна використовувати одну хеш-таблицю для всіх випадків. Хешування відбувається за трьома першими літерами.

Алгоритм:

```
for (int i = 0; i < key.size(); i++)  
  
{  
  
    hash_result = (hash_result + key[i].unicode()) % size;  
  
}
```

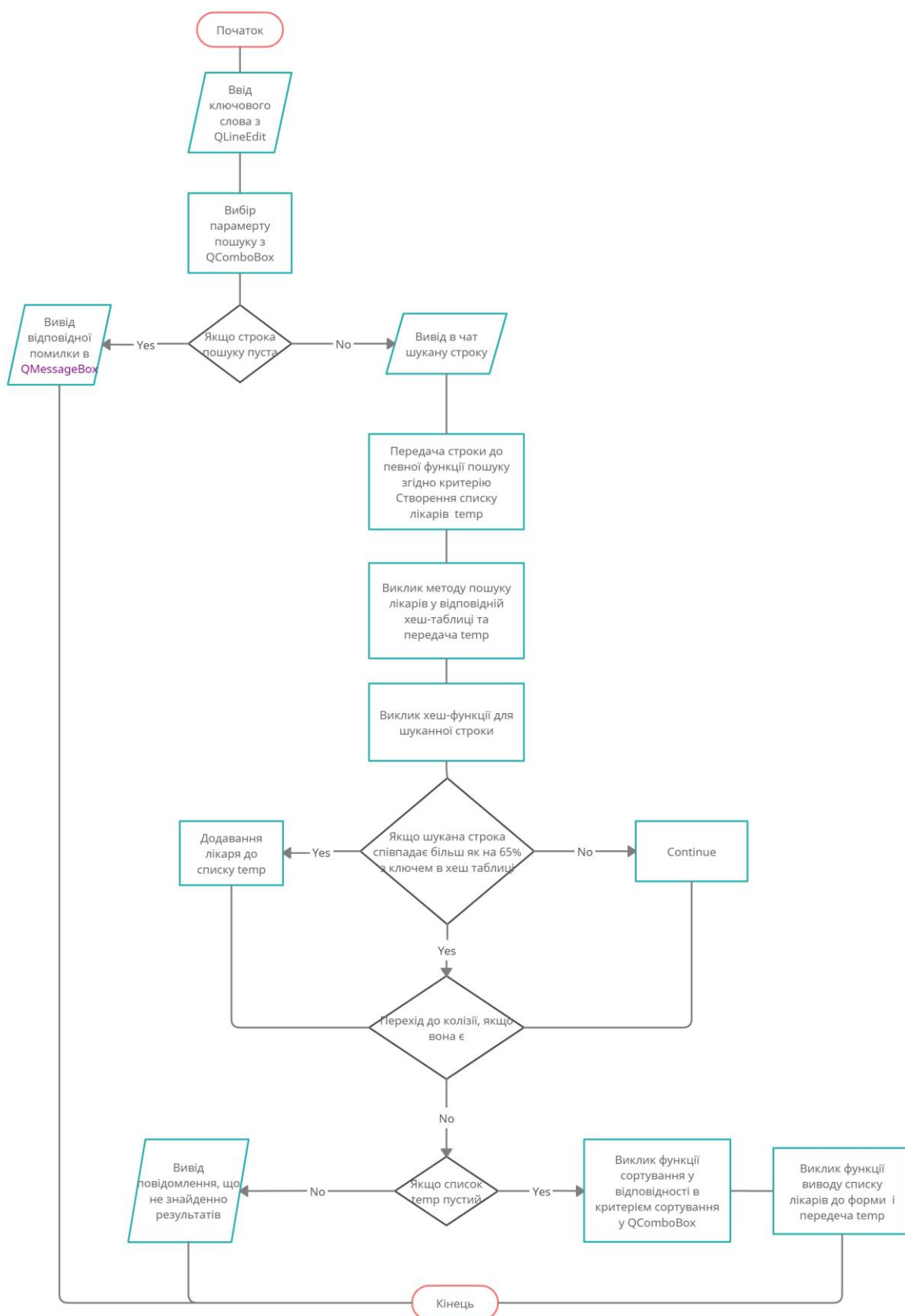


Рисунок 3.14 – схема алгоритму пошуку лікарів за певним критерієм



## Бульбашкове сортування

Алгоритм сортування є методом класу MainWindow рис. 3.15.

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
    QJsonDocument dataDentist;
    QJsonArray dataDentistArr;

    QJsonDocument dataDentistry;
    QJsonArray dataDentistryArr;

    QJsonDocument dataSpec;
    QJsonArray dataSpecArr;

    QJsonDocument interface;
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    void SearchByName(QString SearchWord);
    void SearchBySpecialization(QString SearchWord);
    void SearchByDentistry(QString SearchWord);
    void ShowDentist(QList<Dentist*> *temp);
    void sortByRating(QList<Dentist*> *list);
    void sortByAscending(QList<Dentist*> *list);
    void sortByDescending(QList<Dentist*> *list);
    void OpenSpec();
    void OpenDentist();
    void OpenDentistry();
    void OpenInterface();
    void SaveDentist();
public slots:
    void SendDetailed(Dentist* temp);
    void SendComments(Dentist* temp);
    void SendShowDentistFromDentistry(QString temp);
    void SendShowDentistFromSpecialization(QString temp);
private slots:
    void on_Send_clicked();
    void on_actHelp_triggered();

    void on_actionEng_triggered();

    void on_actionUkr_triggered();

    void on_actDevelops_triggered();

    void on_actionDentists_triggered();

    void on_actionDentistry_triggered();

    void on_actionSpecializations_triggered();

    void on_actionClose_triggered();

private:
    Ui::MainWindow *ui;
    QList<Dentist*> listDentist;
    QList<Dentistry*> listDentistry;
    QList<QPair<QString, QList<QString>>>> specialization;
    HashTableByName hashTableByName;
    HashTableByNameOfDentistry hashTableByNameOfDentistry;
    HashTableBySpecialization hashTableBySpecialization;
    QString pathFileDentist;
    QString pathFileDentistry;
    QString pathFileSpec;
    QString pathInterface;
};
```

Рисунок 3.15 – клас MainWindow

Блок-схема сортування бульбашкою рис. 3.16.

У кодї є три методи сортування:

- по рейтингу (sortByRating);
- по алфавіту (sortByAscending);
- проти алфавіту (sortByDescending).

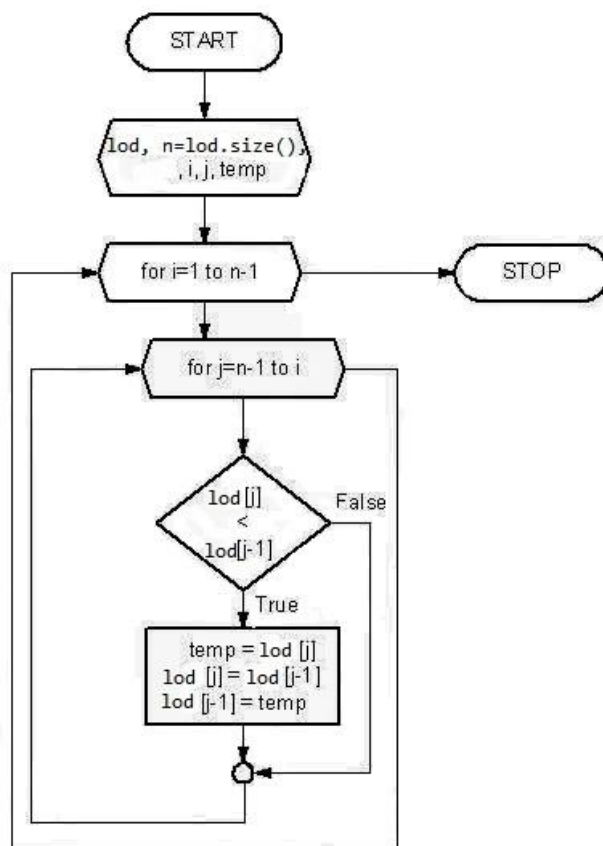


Рисунок 3.16 – схема алгоритму сортування бульбашкою

### 3.4 Виняткові ситуації

Для реалізації виняткових ситуацій в нашому чат-боті, я вирішив створити окремий клас під назвою `MyException` рис. 3.17.

```
#ifndef EXCEPTION_H
#define EXCEPTION_H

#include <QJsonObject>
#include <QJsonParseError>
#include <QJsonArray>
#include <QPixmap>

class MyException {
public:
    MyException(int error_code, QJsonDocument &interface);
    MyException(int error_code, QJsonDocument &interface, QString string_error);
    MyException(QPixmap &pix, QString path);
};

#endif // EXCEPTION_H
```

Рисунок 3.17 – клас `MyException`

У цього класу є конструктори, в яких, завдяки операторам `switch-case`, відбувається пошук кожної виняткової ситуації, і виведення повідомлень про помилки для користувача за допомогою вбудованого у `QtCreator` класу `QMessageBox` рис. 3.18.

```

1 | #include <exception.h>
2 | #include <QMessageBox>
3 |
4 | MyException::MyException(int error_code, QJsonDocument &interface){
5 |     switch(error_code){
6 |         case 1: //пользователь не ввёл данные в диалоговую строку
7 |             QMessageBox::warning(0, interface.object().value("Error").toString(), interface.object().value("EmptyLine").toString());
8 |             break;
9 |         case 2: //пользователь ввёл несуществующий рейтинг
10 |             QMessageBox::warning(0, interface.object().value("Error").toString(), interface.object().value("ErrorRating").toString());
11 |             break;
12 |         case 3: //пользователь не ввёл рейтинг
13 |             QMessageBox::warning(0, interface.object().value("Error").toString(), interface.object().value("ErrorCommentatorName").toString());
14 |             break;
15 |         case 4: //не открывается файл со специализациями
16 |             QMessageBox::warning(0, interface.object().value("FailOpen").toString(), interface.object().value("FailOpenSpec").toString());
17 |             break;
18 |         case 5: //не открывается файл с докторами
19 |             QMessageBox::warning(0, interface.object().value("FailOpen").toString(), interface.object().value("FailOpenDoctors").toString());
20 |             break;
21 |         case 6: //не открывается файл со стоматологиями
22 |             QMessageBox::warning(0, interface.object().value("FailOpen").toString(), interface.object().value("FailOpenDentistry").toString());
23 |             break;
24 |         case 7: //не открывается файл с данными интерфейса: для чтения, для записи
25 |             QMessageBox::warning(0, interface.object().value("FailOpen").toString(), interface.object().value("FailOpenInterface").toString());
26 |             break;
27 |         default:
28 |             break;
29 |     }
30 | };
31 |
32 | MyException::MyException(int error_code, QJsonDocument &interface, QString string_error){
33 |     switch(error_code){
34 |         case 1: //не найдена фамилия
35 |             QMessageBox::warning(0, interface.object().value("NotFound").toString(), interface.object().value("NotFoundName").toString() + string_error + "!");
36 |             break;
37 |         case 2: //не найдена специализация
38 |             QMessageBox::warning(0, interface.object().value("NotFound").toString(), interface.object().value("NotFoundSpec").toString() + string_error + "!");
39 |             break;
40 |         case 3: //не найдена стоматология
41 |             QMessageBox::warning(0, interface.object().value("NotFound").toString(), interface.object().value("NotFoundDentistry").toString() + string_error + "!");
42 |             break;
43 |         default:
44 |             break;
45 |     }
46 | };
47 |
48 | MyException::MyException(QPixmap &pix, QString path){ //не была найдено нужная картинка, поэтому заменена на стандартную
49 |     pix.load(path);
50 | };

```

Рисунок 3.18 – конструктори класу MyException

Для того, щоб виловити виключення, використовувалися оператори: try, catch, throw. try - це блок, в якому відбувається пошук винятків. Якщо виключення не було виявлено, то код продовжує компілюватися. Однак, якщо виключення було знайдено, то використовується оператор throw. Він відправляє сигнал, який ловить оператор catch(), у якому, в свою чергу, оброблюється отримана виняткова ситуація рис. 3.19.

```

void MainWindow::OpenDentistry(){
    try{
        QFile fileDentistry(pathFileDentistry);
        if(!fileDentistry.open(QIODevice::ReadOnly|QFile::Text)){
            throw 6;
        }
        else{
            listDentistry.clear();
            dateDentistry = QJsonDocument::fromJson(QByteArray(fileDentistry.readAll()));
            dateDentistryArr = QJsonValue(dateDentistry.object().value("Dentistry")).toArray();
            for(int i =0; i < dateDentistryArr.count(); i++)
            {
                Dentistry* temp = new Dentistry();
                temp->setNameOfDentistry(dateDentistryArr.at(i).toObject().value("nameOfDentistry").toString());
                temp->setAddress(dateDentistryArr.at(i).toObject().value("address").toString());
                temp->setWorkingHours(dateDentistryArr.at(i).toObject().value("workingHours").toString());
                temp->setAboutUs(dateDentistryArr.at(i).toObject().value("aboutUs").toString());
                temp->setContactsDentistry(dateDentistryArr.at(i).toObject().value("contactsDentistry").toString());
                temp->setIdOfDentistry(dateDentistryArr.at(i).toObject().value("idOfDentistry").toInt());
                listDentistry.append(temp);
            }
        }
        fileDentistry.close();
    }
    catch(int error){
        MyException ex(error, interface);
    }
}

```

Рисунок 3.19 – приклад використання конструкції try-throw-catch у програмі

Загалом у програмі нараховується тринадцять (13) виняткових ситуацій.

Перелік виняткових ситуацій:

Не знайдено прізвище, спеціалізація або стоматологія за запитом користувача  
рис. 3.20.

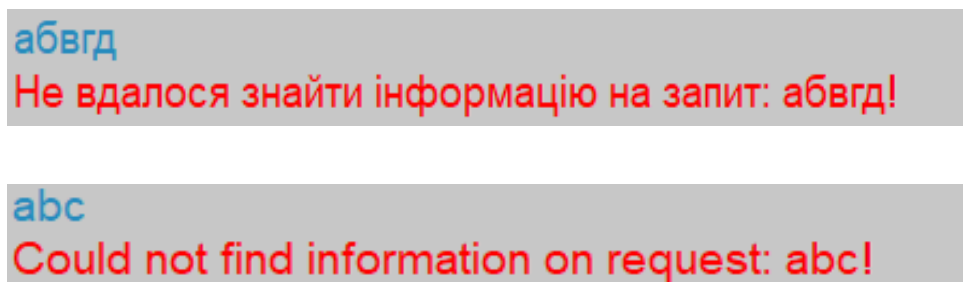


Рисунок 3.20 – виняткова ситуація – не знайдено інформацію за запитом

Користувач нічого не написав у поле для введення інформації та зробив запит рис. 3.21.

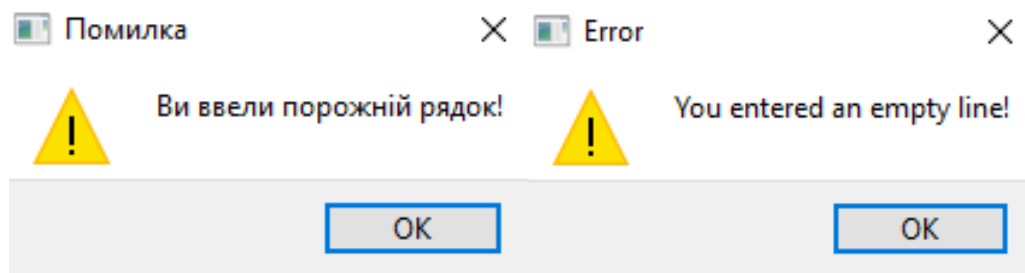


Рисунок 3.21 – виняткова ситуація – пусте поле введення інформації

Користувач не ввів своє ім'я коли залишав відгук рис. 3.22.

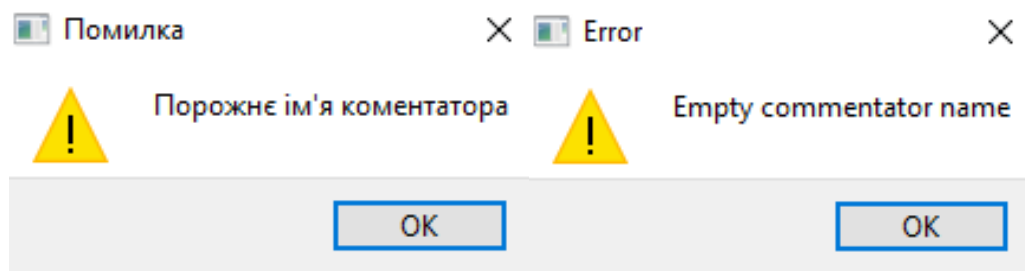


Рисунок 3.22 – виняткова ситуація – нема ім'я користувача у відгуку

Користувач ввів неможливий рейтинг, або не ввів його зовсім рис. 3.23.

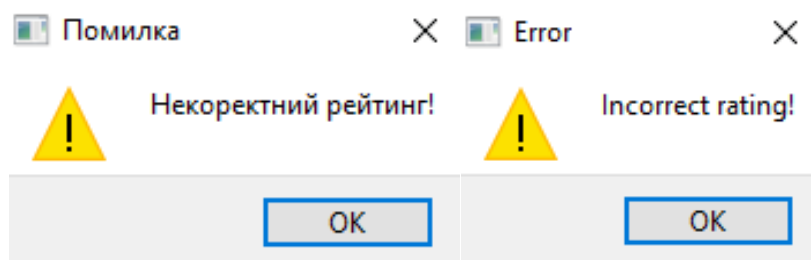


Рисунок 3.23 – виняткова ситуація – невірно введенний рейтинг

Не було знайдено потрібне фото, тому на його місце було завантажено стандартне зображення рис. 3.24.



Рисунок 3.24 – виняткова ситуація – не знайдено потрібне зображення

Не відкривається файл з інформацією про докторів, стоматології, спеціалізації, інтерфейс рис. 3.25.

Інформація з коментарями користувача не записалася до файлу рис. 3.25.

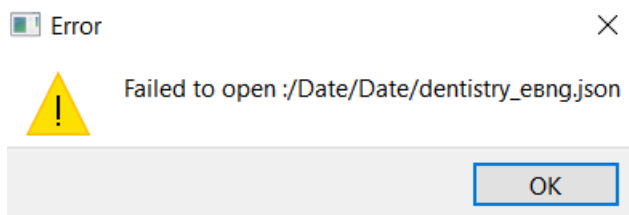


Рисунок 3.25 – виняткова ситуація – проблеми з доступом до файлів

## 4 КЕРІВНИЦТВО ПРОГРАМІСТА

### 4.1 Призначення програми

Призначення програми полягає в пошуку стоматолога по імені, спеціалізації, клініки, а також за ключовими знаходження лікарів.

### 4.2 Перерахування файлів, в яких знаходиться програма

Folder:

- Date;
- Icons;
- ImageDentists.

.pro:

- BotDetective.pro.

.user:

- BotDetective.pro.user.

.h:

- commentsdentist.h;
- dentist.h;
- dentistry.h;
- detaileddentist.h;



- developers.h;
- exception.h;
- hashtablebyname.h;
- hashtablebynameofdentistry.h;
- hashtablebyspecialization.h;
- help.h;
- mainwindow.h.

.cpp:

- commentsdentist.cpp;
- dentist.cpp;
- dentistry.cpp;
- detaileddentist.cpp;
- developers.cpp;
- exception.cpp;
- hashtablebyname.cpp;
- hashtablebynameofdentistry.cpp;
- hashtablebyspecialization.cpp;
- help.cpp;
- main.cpp;
- mainwindow.cpp.

.ui:

- commentsdentist.ui;
- detaileddentist.ui;
- developers.ui;
- help.ui;
- mainwindow.ui.

.qrc:

- resources.qrc.

### 4.3 Вхідні і вихідні дані

1) Date:

- dentist\_eng.json; - json
- dentist\_ukr.json; - json
- dentistry\_eng.db; - sqlite
- dentistry\_ukr.db; - sqlite
- interface\_eng.json; - json
- interface\_ukr.json; - json
- spec\_eng.json; - json
- spec\_ukr.json. – json

2) ImageDentists:

- BBabka.PNG;
- Bedran.PNG;
- Chorna.PNG;
- Cushnarenko.PNG;
- Dovbush.PNG;
- Gesuch.PNG;
- Golovko.PNG;
- Ivanov.PNG;
- Kazimov.PNG;

- Kroka.PNG;
- Kuzemka.PNG;
- Mailov.PNG;
- mosenko.PNG;
- no-avatar.PNG; – якщо не знайшло фотографії стоматолога це буде за замовченням.;
- Onipko.PNG;
- Raeva.PNG;
- Saksov.PNG;
- Stebeneva.PNG;
- Terentiva.PNG;
- Vifak.PNG;
- Yakovenko.PNG;
- Uncle Sam.jpg.

### 3) Icons:

- CommentsDentistIcon.png;
- DetailedDentistIcon.png;
- DevelopersIcon.png;
- HelpIcon.png;
- WindowIcon.png.

## **5 КЕРІВНИЦТВО КОРИСТУВАЧА**

### **5.1 Призначення програми**

Програма призначена для пошуку стоматологів за певним критерієм або ключовим словом. Доступні критерії:

- пошук за прізвищем;
- пошук за спеціалізацією або послугою, яка відноситься до цієї спеціалізації;
- пошук за назвою стоматології.

Також можливо виводити знайдених стоматологів за критерієм сортування:

- сортування за рейтингом;
- сортування за прізвищем (за зростанням);
- сортування за прізвищем (за спаданням).

Можливо додавати коментарі до певного лікаря, таким чином вибудовується рейтинг у стоматологів.

### **5.2 Умови виконання програми**

Програма виконується, якщо вдалося відкрити всі обов'язкові файли з даними стоматологів, стоматологічних послуг та стоматологій. Якщо файли інтерфейсу пошкодженні то програма може працювати некоректно.

### 5.3 Виконання програми

Виконання програми відбувається з відкриття обов'язкових файлів, якщо не вдасться відкрити певний файл, то з'явиться повідомлення який саме файл не вдалося відкрити. Під час завершення програми дані про стоматологів зберігаються у відповідний файл, якщо програма некоректно завершиться, то всі зміни які відбувалися в програмі не зберігуться.

### 5.4 Повідомлення користувачу

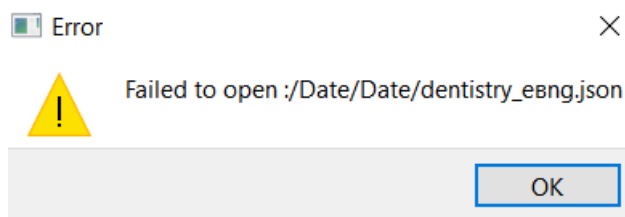


Рисунок 5.1 - Повідомлення, якщо не вдалося відкрити певний файл

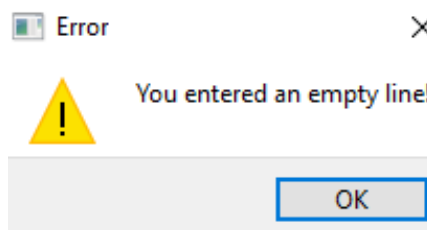


Рисунок 5.2 - Повідомлення, якщо введений пустий рядок

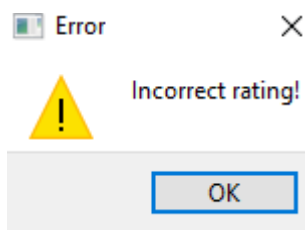


Рисунок 5.3 - Повідомлення, якщо при додаванні коментаря введений некоректний рейтинг

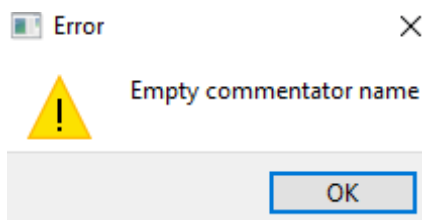


Рисунок 5.4 - Повідомлення, якщо не введено ім'я коментатора



Рисунок 5.5 - Повідомлення, якщо не знайдено стоматологів за заданим критерієм

## 5.5 Керівництво по програмі

Фото-керівництво головного екрану чат-боту представлено на рис. 5.6.

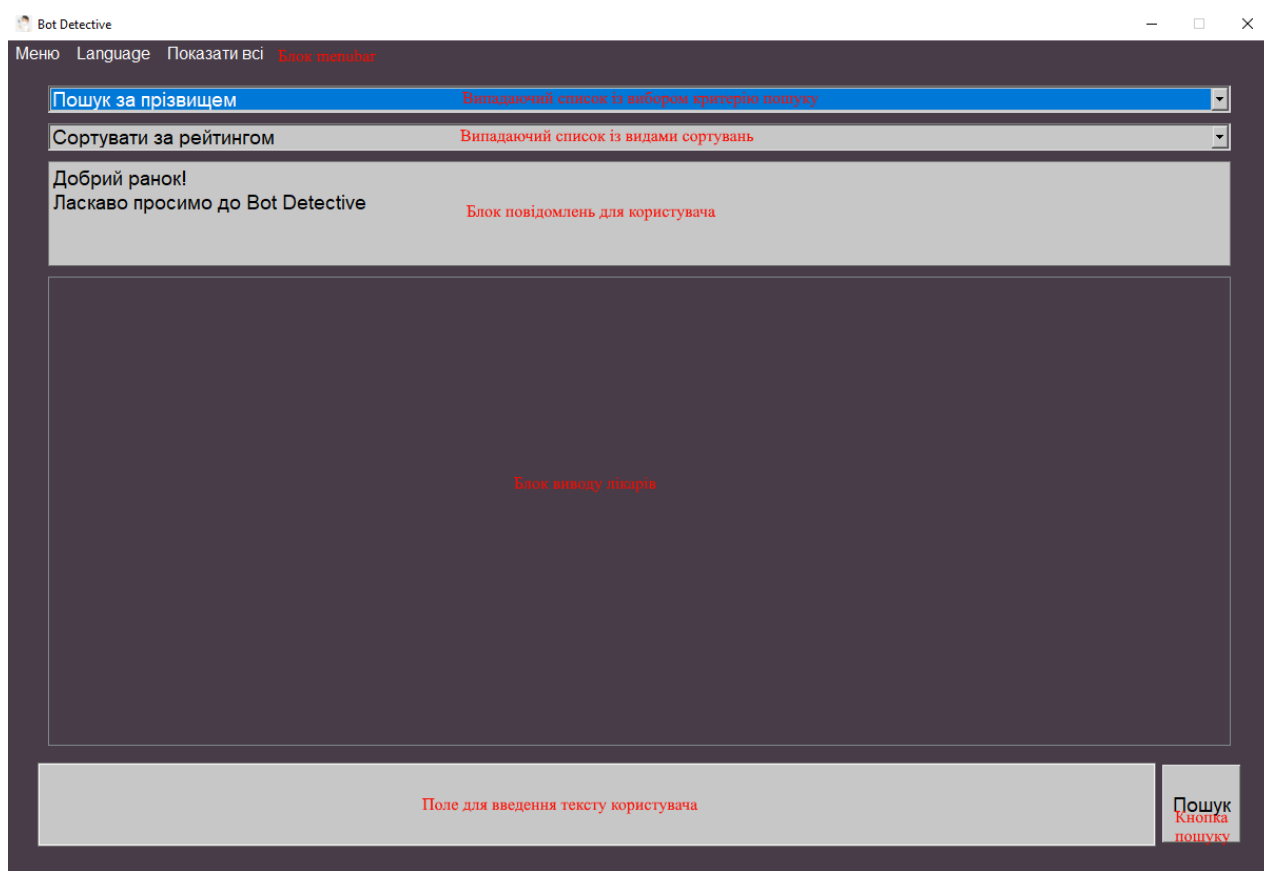


Рисунок 5.6 – фото-керівництво

## ВИСНОВКИ

У цій програмі були досить вдало застосовні хеш-таблиці для зберігання даних. Були добре зроблені структури такі як Dentist, Dentistry. Так само не можна не помітити відмінне використання JSON.

Із недоліків не зовсім коректна сортування на українському (сортування відбувалася при перекладі в unicode і проблема полягала в тому, що “і” менший unicode ніж у “а”).

Другим недоліком можна вважати відсутність збереження прогресу використання програми при переході на іншу мову (всі знайдені лікарі зникали).



## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Чат-бот [Електронний ресурс]: epochta.ru . Стаття. Режим доступа: <https://www.epochta.ru/blog/articles/chat-bots/>
2. C++ [Електронний ресурс]: vc.ru. Стаття. Режим доступа: <https://vc.ru/hr/50161-pochemu-c-krut-aktualen-i-bessmerten>
3. QT [Електронний ресурс]: Вікіпедія. Вільна енциклопедія. Режим доступа: <https://uk.wikipedia.org/wiki/Qt>
4. ООП [Електронний ресурс]: Вікіпедія. Вільна енциклопедія. – Режим доступа: [https://uk.wikipedia.org/wiki/Об'єктно-орієнтоване\\_програмування](https://uk.wikipedia.org/wiki/Об'єктно-орієнтоване_програмування)
5. QList [Електронний ресурс]: doc.qt.io. Документація. – Режим доступа: <https://doc.qt.io/qt-5/qlist.html>
6. Базові контейнери QT [Електронний ресурс]: doc.crossplatform.ru. Вільна енциклопедія. – Режим доступа: <http://doc.crossplatform.ru/qt/4.3.2/containers.html>
7. Класи QString і QVariant [Електронний ресурс]: opennet.ru. Вільна енциклопедія. – Режим доступа: [https://www.opennet.ru/docs/RUS/qt3\\_prog/x5518.html](https://www.opennet.ru/docs/RUS/qt3_prog/x5518.html)
8. Віджети та компоновка QT [Електронний ресурс]: doc.crossplatform.ru. Вільна енциклопедія. – Режим доступа: <http://doc.crossplatform.ru/qt/4.7.x/widgets-and-layouts.html>
9. Хеш-таблиці [Електронний ресурс]: habr.com. Пост. – Режим доступа: <https://habr.com/ru/post/509220/>

## ДОДАТОК А

### Текст програми

#### BotDetective.pro

QT += core gui sql

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

CONFIG += c++11

# You can make your code fail to compile if it uses deprecated APIs.

# In order to do so, uncomment the following line.

#DEFINES += QT\_DISABLE\_DEPRECATED\_BEFORE=0x060000 # disables all  
the APIs deprecated before Qt 6.0.0

SOURCES += \  
 commentsdentist.cpp \  
 dentist.cpp \  
 dentistry.cpp \  
 detaileddentist.cpp \  
 developers.cpp \  
 exception.cpp \  
 hashtablebyname.cpp \  
 hashtablebynameofdentistry.cpp \  
 hashtablebyspecialization.cpp \  
 help.cpp \  
 main.cpp \  
 mainwindow.cpp

HEADERS += \  
 commentsdentist.h \  
 dentist.h \  
 dentistry.h \  
 detaileddentist.h \  
 developers.h \  
 exception.h \  
 hashtablebyname.h \  
 hashtablebynameofdentistry.h \  
 hashtablebyspecialization.h \  
 help.h \  
 mainwindow.h

```
# Default rules for deployment.
qnx: target.path = /tmp/${TARGET}/bin
else: unix:!android: target.path = /opt/${TARGET}/bin
!isEmpty(target.path): INSTALLS += target
```

```
RESOURCES += \
    resources.qrc
```

```
FORMS += \
    commentsdentist.ui \
    detaileddentist.ui \
    developers.ui \
    help.ui \
    mainwindow.ui
```

```
DISTFILES += \
    BotDetective.pro.user
```

### **commentsdentist.h**

```
#ifndef COMMENTSIDENTIST_H
#define COMMENTSIDENTIST_H
```

```
#include <QWidget>
#include "dentist.h"
#include <QJsonObject>
#include <QJsonParseError>
#include <QJsonArray>
#include "dentist.h"
```

```
namespace Ui {
class CommentsDentist;
}
```

```
class CommentsDentist : public QWidget
{
    Q_OBJECT
```

```
public:
    explicit CommentsDentist(QWidget *parent = nullptr);
    ~CommentsDentist();
    void sendDentist(Dentist *temp, QJsonDocument &interface);
    void addComment();
```

```

        void countRating();

private slots:
    void on_AddCommentBtn_clicked();

private:
    Ui::CommentsDentist *ui;
    QJsonDocument interface;
    Dentist *currentDentist;
};

#endif // COMMENTSIDENTIST_H

```

### commentsdentist.cpp

```

#include "commentsdentist.h"
#include "ui_commentsdentist.h"
#include "dentist.h"
#include "exception.h"
#include "QBoxLayout"
#include "QTextBrowser"
#include <QMessageBox>
#include <QDoubleSpinBox>

CommentsDentist::CommentsDentist(QWidget *parent) :QWidget(parent),ui(new
Ui::CommentsDentist)
{
    ui->setupUi(this);
}

CommentsDentist::~CommentsDentist()
{
    delete ui;
}

void CommentsDentist::sendDentist(Dentist *temp, QJsonDocument &interface){
    this->interface = interface;
    this->currentDentist = temp;
    QPixmap pix(":/Images/ImageDentists/ImageDentists/" + currentDentist-
>getPhoto());
    try{
        if (pix.isNull())
        {
            throw QString (":/Images/ImageDentists/ImageDentists/no-avatar.PNG");

```

```

    }
}
catch(QString path){
    MyException ex(pix, path);
}
ui->Photo->setPixmap(pix.scaled(240, 180));
ui->FirstName->setText(interface.object().value("FirstName").toString() + ": " +
currentDentist->getFirstName());
ui->LastName->setText(interface.object().value("LastName").toString() + ": " +
currentDentist->getLastName());
ui->Patronymic->setText(interface.object().value("Patronymic").toString() + ": " +
currentDentist->getPatronymic());
ui->Specialization->setText(interface.object().value("Specialization").toString() +
": " + currentDentist->getSpecialization());
ui->AddCommentBtn-
>setText(interface.object().value("AddCommentBtn").toString());
ui->CommentLabel-
>setText(interface.object().value("CommentLabel").toString());
ui->CommentatorNameLabel-
>setText(interface.object().value("CommentatorNameLabel").toString());
ui->RatingLabel->setText(interface.object().value("Rating").toString() + "
([0,5])");
addComment();
}

void CommentsDentist::addComment(){
    QVBoxLayout *mainLayout = new QVBoxLayout();
    if (currentDentist->getListComments().size() == 0)
    {
        QPixmap pix(":/Images/ImageDentists/ImageDentists/Uncle Sam.jpg");
        QLabel* photoUncleSam = new QLabel;
        photoUncleSam->setPixmap(pix.scaled(522, 293));
        QLabel* nullComments = new
QLabel(interface.object().value("NullComments").toString());
        mainLayout->addWidget(photoUncleSam);
        mainLayout->addWidget(nullComments);
    }
    else{
        for (int i = 0; i < currentDentist->getListComments().size(); i++){
            QVBoxLayout *tmpLayout = new QVBoxLayout();
            tmpLayout->setAlignment(Qt::AlignLeft);
            QTextBrowser *LabelCommentatorName = new QTextBrowser();
            LabelCommentatorName->setText(currentDentist-
>getListComments()[i].getCommentatorName());

```

```

        LabelCommentatorName->setMaximumHeight(35);
        tmpLayout->addWidget(LabelCommentatorName);
        QTextBrowser *LabelComment = new QTextBrowser();
        LabelComment->setText(currentDentist-
>getListComments()[i].getComment());
        LabelComment->setMaximumHeight(75);
        tmpLayout->addWidget(LabelComment);
        QTextBrowser *LabelRating = new QTextBrowser();
        LabelRating->setText(QString::number(currentDentist-
>getListComments()[i].getRating()));
        LabelRating->setMaximumHeight(35);
        tmpLayout->addWidget(LabelRating);
        tmpLayout->setContentsMargins(0,10,0,10);
        mainLayout->addLayout(tmpLayout);
    }
}
QWidget *wd = new QWidget();
wd->setLayout(mainLayout);
ui->scrollArea->setWidget(wd);
}
void CommentsDentist::countRating(){
    double Rating = 0;
    for(int i = 0; i < currentDentist->getListComments().size(); i++){
        Rating += currentDentist->getListComments()[i].getRating();
    }
    currentDentist->setRating(Rating / currentDentist->getListComments().size());
}
void CommentsDentist::on_AddCommentBtn_clicked()
{
    try{
        if( ui->Rating->text() == "" || ui->Rating->text().at(0).unicode() < 48 || ui->Rating-
>text().at(0).unicode() > 53 || (ui->Rating->text().at(0).unicode() == 53 && ui-
>Rating->text().size() != 1))
        {
            throw 2;
        }
        else if (ui->CommentatorName->text() == "")
        {
            throw 3;
        }
        else{
            Comments *temp = new Comments();
            temp->setCommentatorName(ui->CommentatorName->text());
            temp->setComment(ui->Comment->text());

```

```

temp->setRating(ui->Rating->text().toDouble());
ui->CommentatorName->clear();
ui->Comment->clear();
ui->Rating->clear();
currentDentist->addListComments(*temp);
addComment();
countRating();
QMessageBox::information(0,
interface.object().value("Success").toString(),interface.object().value("SuccessAddCo
mment").toString());
}
}
catch(int error){
    MyException ex(error, interface);
}
}

```

### dentist.h

```

#ifndef DENTIST_H
#define DENTIST_H
#include "dentistry.h"
#include <QString>
#include <QList>
class Comments{
public:
    Comments();
    void setCommentatorName(QString CommentatorName);
    void setComment(QString Comment);
    void setRating(double Rating);
    QString getCommentatorName();
    QString getComment();
    double getRating();
private:
    QString commentatorName;
    QString comment;
    double rating;
};

class Dentist
{
public:
    Dentist();
    Dentist(QString objJSON);

```

```

~Dentist();
QString getFirstName();
QString getLastName();
QString getPatronymic();
QString getSpecialization();
QString getAboutDentist();
QString getContactsDentist();
int getYears();
double getRating();
int getWorkExperience();
QString getPhoto();
Dentistry getDentistry();
int getIdOfDentistry();
QList<Comments>& getListComments();

void setFirstName(QString FirstName);
void setLastName(QString LastName);
void setPatronymic(QString Patronymic);
void setSpecialization(QString Specialization);
void setAboutDentist(QString AboutDentist);
void setContactsDentist(QString ContactsDentist);
void setYears(int Years);
void setRating(double Rating);
void setWorkExperience(int WorkExperience);
void setPhoto(QString Photo);
void setDentistry(Dentistry dentistry);
void setIdOfDentistry(int idOfDentistry);
void addListComments(Comments &comment);
Dentist& operator=(Dentist& other);
friend class Comments;
private:
    Dentistry dentistry;
    QString firstName;
    QString lastName;
    QString patronymic;
    QString specialization;
    QString aboutDentist;
    QString contactsDentist;
    int years;
    double rating;
    int workExperience;
    int idOfDentistry;
    QString photo;
    QList<Comments> listComments;

```



```
};
```

```
#endif // DENTIST_H
```

### **dentist.cpp**

```
#include "dentist.h"
```

```
Dentist::Dentist()
```

```
{
    firstName = "Empty";
    lastName = "Empty";
    patronymic = "Empty";
    specialization = "Empty";
    aboutDentist = "Empty";
    contactsDentist = "Empty";
    years = NULL;
    rating = NULL;
    workExperience = NULL;
    idOfDentistry = -1;
}
```

```
Dentist::~Dentist() {}
```

```
QString Dentist::getFirstName() {return this->firstName;}
```

```
QString Dentist::getLastName() {return this->lastName;}
```

```
QString Dentist::getPatronymic() {return this->patronymic;}
```

```
QString Dentist::getSpecialization() {return this->specialization;}
```

```
QString Dentist::getAboutDentist() {return this->aboutDentist;}
```

```
QString Dentist::getContactsDentist() {return this->contactsDentist;}
```

```
QString Dentist::getPhoto(){return this->photo;}
```

```
Dentistry Dentist::getDentistry(){return dentistry;}
```

```
int Dentist::getYears() {return this->years;}
```

```
double Dentist::getRating() {return this->rating;}
```

```
int Dentist::getWorkExperience() {return this->workExperience;}
```

```
int Dentist::getIdOfDentistry(){return idOfDentistry;}
```

```
QList<Comments>& Dentist::getListComments(){return this->listComments;}
```

```
void Dentist::setFirstName(QString FirstName) {this->firstName = FirstName;}
```

```
void Dentist::setLastName(QString LastName){this->lastName = LastName;}
```

```
void Dentist::setPatronymic(QString Patronymic) {this->patronymic = Patronymic;}
```

```
void Dentist::setSpecialization(QString Specialization) {this->specialization =  
Specialization;}
```

```

void Dentist::setAboutDentist(QString AboutDentist) {this->aboutDentist =
AboutDentist;}
void Dentist::setContactsDentist(QString ContactsDentist) {this->contactsDentist =
ContactsDentist;}
void Dentist::setYears(int Years) {this->years = Years;}
void Dentist::setRating(double Rating) {this->rating = Rating;}
void Dentist::setWorkExperience(int WorkExperience) {this->workExperience =
WorkExperience;}
void Dentist::setPhoto(QString Photo){this->photo = Photo;}
void Dentist::setDentistry(Dentistry dentistry){this->dentistry = dentistry;}
void Dentist::setIdOfDentistry(int idOfDentistry){this->idOfDentistry =
idOfDentistry;}
void Dentist::addListComments(Comments
&comment){listComments.append(comment);}
Dentist& Dentist::operator=(Dentist& other)
{
    this->photo = other.photo;
    this->years = other.years;
    this->rating = other.rating;
    this->lastName = other.lastName;
    this->dentistry = other.dentistry;
    this->firstName = other.firstName;
    this->patronymic = other.patronymic;
    this->aboutDentist = other.aboutDentist;
    this->specialization = other.specialization;
    this->workExperience = other.workExperience;
    this->contactsDentist = other.contactsDentist;
    this->listComments = other.listComments;
    this->idOfDentistry = other.idOfDentistry;
    return *this;
}
Comments::Comments(){
    comment = "Empty";
    commentatorName = "Empty";
    rating = -1;
}
void Comments::setCommentatorName(QString CommentatorName){this-
>commentatorName = CommentatorName;}
void Comments::setComment(QString Comment){this->comment = Comment;}
void Comments::setRating(double Rating){this->rating = Rating;}
QString Comments::getCommentatorName(){return this->commentatorName;}
QString Comments::getComment(){return this->comment;}
double Comments::getRating(){return this->rating;}

```

**dentistry.h**

```

#ifndef DENTISTRY_H
#define DENTISTRY_H

#include <QString>

class Dentistry
{
public:
    Dentistry();
    Dentistry(QString objJSON);
    ~Dentistry();
    QString getNameOfDentistry();
    QString getAddress();
    QString getWorkingHours();
    QString getAboutUs();
    QString getContactsDentistry();
    int getIdOfDentistry();

    void setNameOfDentistry(QString NameOfDentistry);
    void setAddress(QString Address);
    void setWorkingHours(QString WorkingHours);
    void setAboutUs(QString AboutUs);
    void setContactsDentistry(QString ContactsDentistry);
    void setIdOfDentistry(int idOfDentistry);

    Dentistry& operator=(Dentistry& other);
private:
    QString nameOfDentistry;
    QString address;
    QString workingHours;
    QString aboutUs;
    QString contactsDentistry;
    int idOfDentistry;
};

#endif // DENTISTRY_H

```

### **dentistry.cpp**

```

#include "dentistry.h"

Dentistry::Dentistry()
{
    nameOfDentistry = "Empty";
}

```

```

    address = "Empty";
    workingHours = "Empty";
    aboutUs = "Empty";
    contactsDentistry = "Empty";
    idOfDentistry = -1;
}
Dentistry::~Dentistry() {}

QString Dentistry::getNameOfDentistry() {return this->nameOfDentistry;}
QString Dentistry::getAddress() {return this->address;}
QString Dentistry::getWorkingHours() {return this->workingHours;}
QString Dentistry::getAboutUs() {return this->aboutUs;}
QString Dentistry::getContactsDentistry() {return this->contactsDentistry;}
int Dentistry::getIdOfDentistry() {return this->idOfDentistry;}

void Dentistry::setNameOfDentistry(QString NameOfDentistry) {this-
>nameOfDentistry = NameOfDentistry;}
void Dentistry::setAddress(QString Address) {this->address = Address;}
void Dentistry::setWorkingHours(QString WorkingHours) {this->workingHours =
WorkingHours;}
void Dentistry::setAboutUs(QString AboutUs) {this->aboutUs = AboutUs;}
void Dentistry::setContactsDentistry(QString ContactsDentistry) { this-
>contactsDentistry = ContactsDentistry;}
void Dentistry::setIdOfDentistry(int idOfDentistry) {this->idOfDentistry =
idOfDentistry;}

Dentistry& Dentistry::operator=(Dentistry& other)
{
    this->nameOfDentistry = other.nameOfDentistry;
    this->workingHours = other.workingHours;
    this->address = other.address;
    this->aboutUs = other.aboutUs;
    this->contactsDentistry = other.contactsDentistry;
    this->idOfDentistry = other.idOfDentistry;
    return *this;
}

```

### **detaileddentist.h**

```

#ifndef DETAILEDDEDENTIST_H
#define DETAILEDDEDENTIST_H

#include <QWidget>
#include "dentist.h"

```

```

#include "dentistry.h"
#include <QJsonObject>
#include <QJsonParseError>
#include <QJsonArray>
namespace Ui {
class DetailedDentist;
}

class DetailedDentist : public QWidget
{
    Q_OBJECT

public:
    explicit DetailedDentist(QWidget *parent = nullptr);
    ~DetailedDentist();
    void sendDentist(Dentist* temp, QJsonDocument &interface);

private:
    Ui::DetailedDentist *ui;
};

#endif // DETAILEDDENTIST_H

```

### **detaileddentist.cpp**

```

#include "detaileddentist.h"
#include "ui_detaileddentist.h"
#include <QPixmap>
DetailedDentist::DetailedDentist(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::DetailedDentist)
{
    ui->setupUi(this);
}

DetailedDentist::~DetailedDentist()
{
    delete ui;
}

void DetailedDentist::sendDentist(Dentist* temp, QJsonDocument &interface){
    QPixmap pix(":/Images/ImageDentists/ImageDentists/" + temp->getPhoto());

```

```

    if (pix.isNull())
    {
        QPixmap noAvatarPix(":/Images/ImageDentists/ImageDentists/no-
avatar.PNG");
        ui->photo->setPixmap(noAvatarPix.scaled(360,270));
    }
    else
    {
        ui->photo->setPixmap(pix.scaled(360,270));
    }
    ui->FirstName->setText(interface.object().value("FirstName").toString() + ": " +
temp->getFirstName());
    ui->LastName->setText(interface.object().value("LastName").toString() + ": " +
temp->getLastName());
    ui->Patronymic->setText(interface.object().value("Patronymic").toString() + ": " +
temp->getPatronymic());
    ui->Years->setText(interface.object().value("Years").toString() + ": " +
QString::number(temp->getYears()));
    ui->WorkExperience-
>setText(interface.object().value("WorkExperience").toString() + ": " +
QString::number(temp->getWorkExperience()));
    ui->WorkingHours-
>setText(interface.object().value("WorkingHours").toString() + ": " + temp-
>getDentistry().getWorkingHours());
    ui->Address->setText(interface.object().value("Address").toString() + ": " +
temp->getDentistry().getAddress());
    ui->AboutUs->setText(interface.object().value("AboutUs").toString() + ": " +
temp->getDentistry().getAboutUs());
    ui->ContactsDentistry-
>setText(interface.object().value("ContactsDentistry").toString() + ": " + temp-
>getDentistry().getContactsDentistry());
    ui->NameOfDentistry-
>setText(interface.object().value("NameOfDentistry").toString() + ": " + temp-
>getDentistry().getNameOfDentistry());
    ui->ContactsDentist-
>setText(interface.object().value("ContactsDentist").toString() + ": " + temp-
>getContactsDentist());
    ui->Rating->setText(interface.object().value("Rating").toString() + ": " +
QString::number(temp->getRating()));
    ui->AboutDentist-
>setText(interface.object().value("AboutDentist").toString() + ": " + temp-
>getAboutDentist());

```

```

        ui->Specialization-
>setText(interface.object().value("Specialization").toString() + ": " + temp-
>getSpecialization());

}

```

## developers.h

```

#ifndef DEVELOPERS_H
#define DEVELOPERS_H

#include <QWidget>
#include <QJsonObject>
#include <QJsonParseError>
#include <QJsonArray>
namespace Ui {
class Developers;
}

class Developers : public QWidget
{
    Q_OBJECT

public:
    explicit Developers(QWidget *parent = nullptr);
    ~Developers();
    void SendInterface(QJsonDocument &interface);

private:
    Ui::Developers *ui;
};

#endif // DEVELOPERS_H

```

## developers.cpp

```

#include "developers.h"
#include "ui_developers.h"

```

```

Developers::Developers(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Developers)
{
    ui->setupUi(this);
}

Developers::~Developers()
{
    delete ui;
}

void Developers::SendInterface(QJsonDocument &interface){
    ui->DevelopersText-
>setText(interface.object().value("DevelopersText").toString());
}

```

### **exception.h**

```

#ifndef EXCEPTION_H
#define EXCEPTION_H

#include <QJsonObject>
#include <QJsonParseError>
#include <QJsonArray>
#include <QPixmap>

class MyException {
public:
    MyException(int error_code, QJsonDocument &interface);
    MyException(int error_code, QJsonDocument &interface, QString string_error);
    MyException(QPixmap &pix, QString path);
};

#endif // EXCEPTION_H

```

### **exception.cpp**

```

#include <exception.h>
#include <QMessageBox>

```



```

MyException::MyException(int error_code, QJsonDocument &interface){
    switch(error_code){
        case 1: //пользователь не ввёл данные в диалоговую строку
            QMessageBox::warning(0, interface.object().value("Error").toString(),
interface.object().value("EmptyLine").toString());
            break;
        case 2: //пользователь ввёл несуществующий рейтинг
            QMessageBox::warning(0,
interface.object().value("Error").toString(),interface.object().value("ErrorRating").toS
tring());
            break;
        case 3: //пользователь не ввёл рейтинг
            QMessageBox::warning(0,
interface.object().value("Error").toString(),interface.object().value("ErrorCommentat
orName").toString());
            break;
        case 4: //не открывается файл со специализациями
            QMessageBox::warning(0, interface.object().value("FailOpen").toString(),
interface.object().value("FailOpenSpec").toString());
            break;
        case 5: //не открывается файл с докторами
            QMessageBox::warning(0, interface.object().value("FailOpen").toString(),
interface.object().value("FailOpenDoctors").toString());
            break;
        case 6: //не открывается файл со стоматологиями
            QMessageBox::warning(0, interface.object().value("FailOpen").toString(),
interface.object().value("FailOpenDentistry").toString());
            break;
        case 7: //не открывается файл с данными интерфейса: для чтения, для записи
            QMessageBox::warning(0, interface.object().value("FailOpen").toString(),
interface.object().value("FailOpenInterface").toString());
            break;
        default:
            break;
    }
};

```

```

MyException::MyException(int error_code, QJsonDocument &interface, QString
string_error){
    switch(error_code){
        case 1: //не найдена фамилия
            QMessageBox::warning(0, interface.object().value("NotFound").toString(),
interface.object().value("NotFoundName").toString() + string_error + "!");

```

```

        break;
    case 2: //не найдена специализация
        QMessageBox::warning(0, interface.object().value("NotFound").toString(),
interface.object().value("NotFoundSpec").toString() + string_error + "!");
        break;
    case 3: //не найдена стоматология
        QMessageBox::warning(0, interface.object().value("NotFound").toString(),
interface.object().value("NotFoundDentistry").toString() + string_error + "!");
        break;
    default:
        break;
}
};

```

```

MyException::MyException(QPixmap &pix, QString path){ //не была найдено
нужная картинка, поэтому заменена на стандартную
    pix.load(path);
};

```

### hashtablebyname.h

```

#ifndef HASHTABLEBYNAME_H
#define HASHTABLEBYNAME_H

#include "dentist.h"
#include <QString>
#include <QList>

class HashTableByName
{
public:
    struct Node
    {
        QString key;
        QList<Dentist*> listOfDentists;
    };

    HashTableByName();
    ~HashTableByName();
    void push(Dentist* currentOfDentist);
    void search(QString key, QList<Dentist*> *tmp);
    void searchAll(QList<Dentist*> *tmp);

```

```

    void clear();
private:
    Node* arr;
    int hashing(QString key);
    int size;
};

#endif // HASHTABLEBYNAME_H

```

### hashtablebyname.cpp

```

#include "hashtablebyname.h"
HashTableByName::HashTableByName()
{
    size = 100;
    arr = new Node[size];
}
HashTableByName::~HashTableByName()
{
    delete[] arr;
}
void HashTableByName::push(Dentist *currentOfDentist)
{
    int id = hashing(currentOfDentist->getLastName());
    arr[id].listOfDentists.push_back(currentOfDentist);
    arr[id].key = currentOfDentist->getLastName().toLowerCase();
}
int HashTableByName::hashing(QString key)
{
    int hash_result = 0;
    key = key.toLowerCase();
    for (int i = 0; i < 3; i++)//хеширование по трем первым буквам
    {
        hash_result = (hash_result + key[i].unicode()) % size;
    }
    return hash_result;
}
void HashTableByName::search(QString key, QList<Dentist*> *temp)
{
    key = key.toLowerCase();
    int id = hashing(key);
    for(int i = 0; i < arr[id].listOfDentists.size(); i++){

```

```

    int success = 0;
    int sizevalue = key.size();
    for(int j = 0; j < sizevalue; j++){
        if (arr[id].listOfDentists.at(i)->getLastName()[j].toLowerCase() == key[j])
            ++success;
    }
    double res = success / sizevalue;
    if(res >= 0.65){
        temp->append(arr[id].listOfDentists.at(i));
    }
}

void HashTableByName::searchAll(QList<Dentist*> *tmp){
    for(int i = 0; i < size; i++){
        if(arr[i].listOfDentists.size() != 0){
            for(int j = 0; j < arr[i].listOfDentists.size(); j++){
                tmp->append(arr[i].listOfDentists.at(j));
            }
        }
    }
}

void HashTableByName::clear()
{
    delete arr;
    arr = new Node[size];
}

```

### hashtablebynameofdentistry.h

```

#ifndef HASHTABLEBYNAMEOFDENTISTRY_H
#define HASHTABLEBYNAMEOFDENTISTRY_H

#include "dentist.h"
#include <QString>
#include <QList>

class HashTableByNameOfDentistry
{
public:

```

```

struct Node
{
    QString key;
    QList<Dentist*> listOfDentists;
};

HashTableByNameOfDentistry();
~HashTableByNameOfDentistry();
void push(Dentist* currentOfDentist);
void search(QString key, QList<Dentist*> *temp);
void clear();
private:
    Node* arr;
    int hashing(QString key);
    int size;
};

#endif // HASHTABLEBYNAMEOFDENTISTRY_H

```

### **hashtablebynameofdentistry.cpp**

```

#include "hashtablebynameofdentistry.h"

HashTableByNameOfDentistry::HashTableByNameOfDentistry()
{
    size = 30;
    arr = new Node[size];
}

HashTableByNameOfDentistry::~HashTableByNameOfDentistry()
{
    delete[] arr;
}

void HashTableByNameOfDentistry::push(Dentist *currentOfDentist)
{
    int id = hashing(currentOfDentist->getDentistry().getNameOfDentistry());
    arr[id].listOfDentists.push_back(currentOfDentist);
    arr[id].key = currentOfDentist->getDentistry().getNameOfDentistry().toLower();
}

int HashTableByNameOfDentistry::hashing(QString key)
{
    int hash_result = 0;
    key = key.toLower();
}

```

```

    for (int i = 0; i < 3; i++)//хеширование по трем первым буквам
    {
        hash_result = (hash_result + key[i].unicode()) % size;
    }
    return hash_result;
}
void HashTableByNameOfDentistry::search(QString key, QList<Dentist*> *temp)
{
    key = key.toLower();
    int id = hashing(key);
    for(int i = 0; i < arr[id].listOfDentists.size(); i++){
        int success = 0;
        int sizevalue = key.size();
        for(int j = 0; j < sizevalue; j++){
            if (arr[id].listOfDentists.at(i)-
>getDentistry().getNameOfDentistry()[j].toLower() == key[j])
                ++success;

        }
        double res = success / sizevalue;
        if(res >= 0.65){
            temp->append(arr[id].listOfDentists.at(i));
        }
    }
}

void HashTableByNameOfDentistry::clear()
{
    for(int i = 0; i < size; i++)
    {
        arr[i].key = "";
        arr[i].listOfDentists.clear();
    }
}

```

### hashtablebyspecialization.h

```

#ifndef HASHTABLEBYSPECIALIZATION_H
#define HASHTABLEBYSPECIALIZATION_H
#include "dentist.h"
#include <QString>
#include <QList>

```

```

#include <QPair>

class HashTableBySpecialization
{
public:
    struct Node
    {
        QString key;
        QList<Dentist*> listOfDentists;
    };

    HashTableBySpecialization();
    ~HashTableBySpecialization();
    void push(Dentist* currentOfDentist);
    void search(QString key, QList<Dentist*> *temp);
    void setKeyWords(QList<QPair<QString, QList<QString>>>
*keyWordsForSearchBySpec);
    void clear();
private:
    Node* arr;
    QList<QPair<QString, QList<QString>>> *keyWordsForSearchBySpec;
    QString searchByKeyWords(QString keyWords);
    int hashing(QString key);
    int size;
};

#endif // HASHTABLEBYSPECIALIZATION_H

```

### **hashtablebyspecialization.cpp**

```

#include "hashtablebyspecialization.h"

HashTableBySpecialization::HashTableBySpecialization()
{
    size = 1000;
    arr = new Node[size];
    keyWordsForSearchBySpec = new QList<QPair<QString, QList<QString>>>;
}
HashTableBySpecialization::~~HashTableBySpecialization()
{
    delete[] arr;
}

```

```

void HashTableBySpecialization::push(Dentist *currentOfDentist)
{
    int id = hashing(currentOfDentist->getSpecialization());
    arr[id].listOfDentists.push_back(currentOfDentist);
    arr[id].key = currentOfDentist->getSpecialization().toLower();
}
int HashTableBySpecialization::hashing(QString key)
{
    int hash_result = 0;
    key = key.toLower();
    for (int i = 0; i < key.size(); i++)//хеширование по трем первым буквам
    {
        hash_result = (hash_result + key[i].unicode()) % size;
    }
    return hash_result;
}
void HashTableBySpecialization::search(QString key, QList<Dentist*> *temp)
{
    key = key.toLower();
    int id = hashing(key);

    if(arr[id].listOfDentists.size() == 0)
    {
        QString newKey = searchByKeyWords(key);
        if (newKey != key)
        {
            search(newKey, temp);
        }
    }

    for(int i = 0; i < arr[id].listOfDentists.size(); i++)
    {
        int success = 0;
        int sizevalue = key.size();
        for(int j = 0; j < sizevalue; j++){
            if (arr[id].listOfDentists.at(i)->getSpecialization()[j].toLower() == key[j])
                ++success;
        }
        double res = success / sizevalue;
        if(res >= 0.65){
            temp->append(arr[id].listOfDentists.at(i));
        }
    }
}

```



```

}

void HashTableBySpecialization::setKeyWords(QList<QPair<QString,
QList<QString>>> *keyWordsForSearchBySpec)
{
//  for(int i = 0; i < keyWordsForSearchBySpec->size(); i++)
//  {
//      for(int j = 0; j < keyWordsForSearchBySpec->at(i).second.size(); j++)
//      {
//          keyWordsForSearchBySpec->at(i).second.at(j) =
keyWordsForSearchBySpec->at(i).second.at(j).toLower();
//      }
//  }
    this->keyWordsForSearchBySpec = keyWordsForSearchBySpec;
}

QString HashTableBySpecialization::searchByKeyWords(QString keyWords)
{
//  keyWords = keyWords.toLower();
    for (int i = 0; i < keyWordsForSearchBySpec->size(); i++)
    {
        for (int j = 0; j < keyWordsForSearchBySpec->at(i).second.size(); j++)
        {
            int firstSym = keyWords.toStdString().find(keyWordsForSearchBySpec-
>at(i).second.at(j).toLower().toStdString());
            if(firstSym >= 0 /*&& firstSym < keyWords.size()*/)
            {
                keyWords = keyWordsForSearchBySpec->at(i).first;
                return keyWords;
            }
        }
    }
    return keyWords;
}

void HashTableBySpecialization::clear()
{
    for(int i = 0; i < size; i++)
    {
        arr[i].key = "";
        arr[i].listOfDentists.clear();
    }
}

```

**help.h**

```

#ifndef HELP_H
#define HELP_H

#include <QWidget>
#include <QPair>
#include <QList>
#include <QJsonObject>
#include <QJsonParseError>
#include <QJsonArray>

namespace Ui {
class Help;
}

class Help : public QWidget
{
    Q_OBJECT

public:
    explicit Help(QWidget *parent = nullptr);
    ~Help();
    QList<QPair<QString, QList<QString>>>> *specialization;
    void SendSpecialization(QList<QPair<QString, QList<QString>>>>
        *specialization, QJsonDocument &interface);
    void setTable(QJsonDocument &interface);

private:
    Ui::Help *ui;
};

#endif // HELP_H

```

**help.cpp**

```

#include "help.h"
#include "ui_help.h"
#include <QTextBrowser>
Help::Help(QWidget *parent) :
    QWidget(parent),

```

```

    ui(new Ui::Help)
{
    ui->setupUi(this);
    ui->verticalLayout->setAlignment(Qt::AlignCenter);
}

Help::~Help()
{
    delete ui;
}

void Help::SendSpecialization(QList<QPair<QString, QList<QString>>>
*specialization, QJsonDocument &interface){
    this->specialization = specialization;
    ui->HowToUse->setText(interface.object().value("HowToUse").toString());
    setTable(interface);
}

void Help::setTable(QJsonDocument &interface){
    ui->table->verticalHeader()->hide();
    ui->table->horizontalHeader()->hide();
    ui->table->setRowCount(7);
    ui->table->setColumnCount(2);
    QTableWidgetItem *temp1 = new
QTableWidgetItem(interface.object().value("Specialization").toString());
    QTableWidgetItem *temp2 = new
QTableWidgetItem(interface.object().value("Service").toString());
    ui->table->setItem(0, 0, temp1);
    ui->table->setItem(0, 1, temp2);
    for(int i = 0; i < specialization->size(); i++){
        QTextBrowser *r1 = new QTextBrowser();
        r1->setText(specialization->at(i).first);
        QTextBrowser *r2 = new QTextBrowser();
        r2->clear();
        for(int j =0; j < specialization->at(i).second.size(); j++){
            r2->append(specialization->at(i).second.at(j));
        }
        ui->table->setCellWidget(i+1, 0 , r1);
        ui->table->setCellWidget(i+1, 1 , r2);
    }
    ui->table->setDragEnabled(false);
    ui->table->resizeColumnsToContents();
    ui->table->resizeRowsToContents();
}

```

**mainwindow.h**

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <exception.h>
#include <QMainWindow>
#include <QJsonObject>
#include <QJsonParseError>
#include <QFile>
#include <QList>
#include <QPair>
#include <QJsonArray>
#include "QtSql/QtSqlDatabase"
#include "QtSql/QtSqlQuery"
#include "dentist.h"
#include "dentistry.h"
#include "hashtablebyname.h"
#include "hashtablebyspecialization.h"
#include "hashtablebynameofdentistry.h"
#include "help.h"
#include "detaileddentist.h"
#include "commentsdentist.h"
#include "developers.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT
    QJsonDocument dateDentist;
    QJsonArray dateDentistArr;

    QJsonDocument dateSpec;
    QJsonArray dateSpecArr;

    QJsonDocument interface;
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    void SearchByName(QString SearchWord);

```

```

void SearchBySpecialization(QString SearchWord);
void SearchByDentistry(QString SearchWord);
void ShowDentist(QList<Dentist*> *temp);
void sortByRating(QList<Dentist*> *list);
void sortByAscending(QList<Dentist*> *list);
void sortByDescending(QList<Dentist*> *list);
void OpenSpec();
void OpenDentist();
void OpenDentistry();
void OpenInterface();
void SaveDentist();
public slots:
    void SendDetailed(Dentist* temp);
    void SendComments(Dentist* temp);
    void SendShowDentistFromDentistry(QString temp);
    void SendShowDentistFromSpecialization(QString temp);
private slots:
    void on_Send_clicked();
    void on_actHelp_triggered();

    void on_actionEng_triggered();

    void on_actionUkr_triggered();

    void on_actDevelops_triggered();

    void on_actionDentists_triggered();

    void on_actionDentistry_triggered();

    void on_actionSpecializations_triggered();

    void on_actionClose_triggered();

private:
    Ui::MainWindow *ui;
    QList<Dentist*> listDentist;
    QList<Dentistry*> listDentistry;
    QList<QPair<QString, QList<QString>>>> specialization;
    HashTableByName hashTableByName;
    HashTableByNameOfDentistry hashTableByNameOfDentistry;
    HashTableBySpecialization hashTableBySpecialization;
    QString pathFileDentist;
    QString pathFileDentistry;

```

```

    QString pathFileSpec;
    QString pathInterface;
};
#endif // MAINWINDOW_H

```

### mainwindow.cpp

```

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "dentist.h"
#include "exception.h"
#include <QTextEdit>
#include <QTextStream>
#include <QMessageBox>
#include <QTime>
#include <QScrollArea>
#include <QLabel>

class dentist;

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    pathFileDentist = "../BotDetective-main/Date/dentist_eng.json";
    pathFileDentistry = "../BotDetective-main/Date/dentistry_eng.db";
    pathFileSpec = "../BotDetective-main/Date/spec_eng.json";
    pathInterface = "../BotDetective-main/Date/interface_eng.json";
    OpenInterface();
    OpenSpec();
    OpenDentistry();
    OpenDentist();
}

void MainWindow::OpenSpec(){
    try{
        QFile fileSpec(pathFileSpec);
        if(!fileSpec.open(QIODevice::ReadOnly|QFile::Text))
        {
            throw 4;
        }
    }
}

```

```

else{
    specialization.clear();
    dateDentist = QJsonDocument::fromJson(QByteArray(fileSpec.readAll()));
    QPair<QString, QList<QString>> temp;
    temp.first = interface.object().value("Dentist-therapist").toString();
    dateSpecArr = QJsonValue(dateDentist.object().value("Dentist-
therapist")).toArray();
    QString tempText;
    for(int i =0; i < dateSpecArr.count();i++){
        tempText = dateSpecArr[i].toString();
        temp.second.append(tempText);
    }
    specialization.append(temp);
    temp.second.clear();
    temp.first = interface.object().value("Dentist-surgeon-
implantologist").toString();
    dateSpecArr = QJsonValue(dateDentist.object().value("Dentist-surgeon-
implantologist")).toArray();
    for(int i =0; i < dateSpecArr.count();i++){
        tempText = dateSpecArr[i].toString();
        temp.second.append(tempText);
    }
    specialization.append(temp);
    temp.second.clear();
    temp.first = interface.object().value("Dentist-periodontist").toString();
    dateSpecArr = QJsonValue(dateDentist.object().value("Dentist-
periodontist")).toArray();
    for(int i =0; i < dateSpecArr.count();i++){
        tempText = dateSpecArr[i].toString();
        temp.second.append(tempText);
    }
    specialization.append(temp);
    temp.second.clear();
    temp.first = interface.object().value("Dentist-orthopedist").toString();
    dateSpecArr = QJsonValue(dateDentist.object().value("Dentist-
orthopedist")).toArray();
    for(int i =0; i < dateSpecArr.count();i++){
        tempText = dateSpecArr[i].toString();
        temp.second.append(tempText);
    }
    specialization.append(temp);
    temp.second.clear();
    temp.first = interface.object().value("Dentist-orthodontist").toString();

```

```

        dateSpecArr = QJsonValue(dateDentist.object().value("Dentist-
orthodontist")).toArray();
        for(int i =0; i < dateSpecArr.count();i++){
            tempText = dateSpecArr[i].toString();
            temp.second.append(tempText);
        }
        specialization.append(temp);
        temp.second.clear();
        temp.first = interface.object().value("Children's dentist").toString();
        dateSpecArr = QJsonValue(dateDentist.object().value("Children's
dentist")).toArray();
        for(int i =0; i < dateSpecArr.count();i++){
            tempText = dateSpecArr[i].toString();
            temp.second.append(tempText);
        }
        specialization.append(temp);
        temp.second.clear();
        hashTableBySpecialization.setKeyWords(&specialization);
    }
    fileSpec.close();
}
catch(int error){
    MyException ex(error, interface);
}
}

void MainWindow::OpenDentist(){
    try{
        QFile fileDoctors(pathFileDentist);
        if(!fileDoctors.open(QIODevice::ReadOnly|QFile::Text)){
            throw 5;
        }
        else {
            listDentist.clear();
            dateDentist =
QJsonDocument::fromJson(QByteArray(fileDoctors.readAll()));
            dateDentistArr =
QJsonValue(dateDentist.object().value("Doctors")).toArray();
            for(int i =0; i < dateDentistArr.count(); i++)
            {
                Dentist* temp = new Dentist();
                temp-
>setFirstName(dateDentistArr.at(i).toObject().value("firstName").toString());

```



```

        temp-
>setLastName(dateDentistArr.at(i).toObject().value("lastName").toString());
        temp-
>setPatronymic(dateDentistArr.at(i).toObject().value("patronymic").toString());
        temp->setYears(dateDentistArr.at(i).toObject().value("years").toInt());
        temp-
>setSpecialization(dateDentistArr.at(i).toObject().value("specialization").toString());
        temp-
>setAboutDentist(dateDentistArr.at(i).toObject().value("aboutDentist").toString());
        temp-
>setContactsDentist(dateDentistArr.at(i).toObject().value("contactsDentis").toString(
));
        temp-
>setWorkExperience(dateDentistArr.at(i).toObject().value("workExperience").toInt()
);
        temp->setPhoto(dateDentistArr.at(i).toObject().value("photo").toString());
        temp-
>setIdOfDentistry(dateDentistArr.at(i).toObject().value("idOfDentistry").toInt());
        QJsonArray dateCommentsArr =
dateDentistArr.at(i).toObject().value("comments").toArray();
        double tempRating = 0;
        for(int k = 0; k < dateCommentsArr.count(); k++){
            Comments * tmp = new Comments();
            tmp-
>setCommentatorName(dateCommentsArr.at(k).toObject().value("commentatorNam
e").toString());
            tmp-
>setComment(dateCommentsArr.at(k).toObject().value("comment").toString());
            tmp-
>setRating(dateCommentsArr.at(k).toObject().value("rating").toDouble());
            tempRating +=
dateCommentsArr.at(k).toObject().value("rating").toDouble();
            temp->addListComments(*tmp);
        }
        if(dateCommentsArr.count() != 0)
            temp->setRating(tempRating / dateCommentsArr.count());
        else
            temp->setRating(0);

        for (int j = 0; j < listDentistry.count(); j++)
        {
            if (temp->getIdOfDentistry() == listDentistry.at(j)->getIdOfDentistry())
            {
                temp->setDentistry(*listDentistry.at(j));
            }
        }
    }
}

```

```

        break;
    }
}
listDentist.append(temp);
}
}
fileDoctors.close();
hashTableByName.clear();
hashTableByNameOfDentistry.clear();
hashTableBySpecialization.clear();
for(int j =0; j < listDentist.size(); j++){
    hashTableByName.push(listDentist.at(j));
    hashTableByNameOfDentistry.push(listDentist.at(j));
    hashTableBySpecialization.push(listDentist.at(j));
}
}
catch(int error){
    MyException ex(error, interface);
}
}

void MainWindow::OpenDentistry(){
    try{
        QSqlDatabase fileDentistry;
        fileDentistry = QSqlDatabase::addDatabase("QSQLITE");
        fileDentistry.setDatabaseName(pathFileDentistry);
        if(!fileDentistry.open())
        {
            throw 6;
        }
        else
        {
            QSqlQuery query;
            query.exec("SELECT nameOfDentistry, address, workingHours, aboutUs,
contactsDentistry, idOfDentistry FROM Dentistry");
            listDentistry.clear();

            while (query.next())
            {
                Dentistry* temp = new Dentistry();
                temp->setNameOfDentistry(query.value(0).toString());
                temp->setAddress(query.value(1).toString());
                temp->setWorkingHours(query.value(2).toString());
                temp->setAboutUs(query.value(3).toString());
            }
        }
    }
}

```

```

        temp->setContactsDentistry(query.value(4).toString());
        temp->setIdOfDentistry(query.value(5).toInt());
        listDentistry.append(temp);
    }
}
fileDentistry.close();
}
catch(int error){
    MyException ex(error, interface);
}
}

void MainWindow::OpenInterface(){
    try{
        QFile fileInterface(pathInterface);
        if(!fileInterface.open(QIODevice::ReadOnly|QFile::Text))
        {
            throw 7;
        }
        else{
            ui->BotOut->clear();
            ui->SearchBy->clear();
            ui->SortBy->clear();
            interface = QJsonDocument::fromJson(QByteArray(fileInterface.readAll()));
            ui->SearchBy-
>addItem(interface.object().value("SearchBySurname").toString());
            ui->SearchBy-
>addItem(interface.object().value("SearchBySpecialization").toString());
            ui->SearchBy-
>addItem(interface.object().value("SearchByDentistry").toString());
            ui->SortBy->addItem(interface.object().value("SortByRating").toString());
            ui->SortBy-
>addItem(interface.object().value("SortByAscending").toString());
            ui->SortBy-
>addItem(interface.object().value("SortByDescending").toString());
            ui->BotOut->setAlignment(Qt::AlignLeft);
            QTime cur = QTime::currentTime();
            if(cur.hour() >= 6 && cur.hour() <= 12)
                ui->BotOut->append(interface.object().value("GoodMorning").toString());
            else if(cur.hour() > 12 && cur.hour() < 18)
                ui->BotOut-
>append(interface.object().value("GoodAfternoon").toString());
            else
                ui->BotOut->append(interface.object().value("GoodEvening").toString());

```

```

        ui->BotOut->append(interface.object().value("Welcome").toString());
        ui->Send->setText(interface.object().value("Search").toString());
        ui->menuHelp->setTitle(interface.object().value("Menu").toString());
        ui->actHelp->setText(interface.object().value("Help").toString());
        ui->actDevelops->setText(interface.object().value("Developers").toString());
        ui->actionDentistry-
>setText(interface.object().value("AllDentistry").toString());
        ui->actionDentists-
>setText(interface.object().value("AllDentists").toString());
        ui->actionSpecializations-
>setText(interface.object().value("AllSpecializations").toString());
        ui->menuShow_All->setTitle(interface.object().value("ShowAll").toString());
        ui->actionClose->setText(interface.object().value("Close").toString());
    }
    fileInterface.close();
}
catch(int error){
    MyException ex(error, interface);
}
}
MainWindow::~MainWindow()
{
    SaveDentist();
    delete ui;
}
void MainWindow::on_Send_clicked()
{
    try{
        int searchBy = ui->SearchBy->currentIndex();
        QString SearchWord = ui->MessageIn->text();
        if(SearchWord == "")
            throw 1;
        else
        {
            ui->BotOut->append("<p style='color:#258BBE'>" + SearchWord + "</p>");
            ui->MessageIn->clear();
            switch(searchBy)
            {
                case 0: SearchByName(SearchWord); break;
                case 1: SearchBySpecialization(SearchWord); break;
                case 2: SearchByDentistry(SearchWord); break;
            }
        }
    }
}

```

```

    catch(int error){
        MyException ex(error, interface);
    }
}

void MainWindow::SearchByName(QString SearchWord)
{
    try
    {
        QList<Dentist*> temp;
        temp.clear();
        hashTableByName.search(SearchWord, &temp);
        if (temp.isEmpty())
        {
            throw 1;
        }
        else
        {
            switch (ui->SortBy->currentIndex()) {
                case 0: sortByRating(&temp);
                    break;
                case 1 : sortByAscending(&temp);
                    break;
                case 2: sortByDescending(&temp);
                    break;
            }
            ShowDentist(&temp);
        }
    }
    catch(int error)
    {
        MyException ex(error, interface, SearchWord);
    }
}

void MainWindow::SearchBySpecialization(QString SearchWord)
{
    try
    {
        QList<Dentist*> temp;
        temp.clear();
        hashTableBySpecialization.search(SearchWord, &temp);
        if (temp.isEmpty())
        {

```

```

        throw 2;
    }
    else
    {
        switch (ui->SortBy->currentIndex()) {
            case 0: sortByRating(&temp);
                break;
            case 1 : sortByAscending(&temp);
                break;
            case 2: sortByDescending(&temp);
                break;
        }
        ShowDentist(&temp);
    }
}
catch(int error)
{
    MyException ex(error, interface, SearchWord);
}
}
void MainWindow::SearchByDentistry(QString SearchWord)
{
    try
    {
        QList<Dentist*> temp;
        temp.clear();
        hashTableByNameOfDentistry.search(SearchWord, &temp);
        if (temp.isEmpty())
        {
            throw 3;
        }
    }
    else
    {
        switch (ui->SortBy->currentIndex()) {
            case 0: sortByRating(&temp);
                break;
            case 1 : sortByAscending(&temp);
                break;
            case 2: sortByDescending(&temp);
                break;
        }
        ShowDentist(&temp);
    }
}
}

```

```

catch(int error)
{
    MyException ex(error, interface, SearchWord);
}
}

void MainWindow::sortByRating(QList<Dentist*> *list)
{
    for (int i = 0; i < list->size() - 1; i++)
    {
        for (int j = 0; j < list->size() - i - 1; j++)
        {
            if (list->at(j)->getRating() < list->at(j + 1)->getRating())
            {
                Dentist* tmp = new Dentist;
                *tmp = *list->at(j);
                *list->at(j) = *list->at(j + 1);
                *list->at(j + 1) = *tmp;
                delete tmp;
            }
        }
    }
}

void MainWindow::sortByAscending(QList<Dentist *> *list)
{
    for (int i = 0; i < list->size() - 1; i++)
    {
        for (int j = 0; j < list->size() - 1; j++)
        {
            for(int z = 0; z < list->at(j)->getLastName().size(); z++){
                if(list->at(j)->getLastName().at(z).unicode() > list->at(j+1)-
>getLastName().at(z).unicode()){
                    Dentist *temp = new Dentist;
                    *temp = *list->at(j);
                    list->removeAt(j);
                    list->insert(j+1,temp);
                    break;
                }
                else if(list->at(j)->getLastName().at(z).unicode() == list->at(j+1)-
>getLastName().at(z).unicode())
                    continue;
                else
                    break;
            }
        }
    }
}

```

```

    }
}
}
void MainWindow::sortByDescending(QList<Dentist*> *list)
{
    for (int i = 0; i < list->size() - 1; i++)
    {
        for (int j = 0; j < list->size() - 1; j++)
        {
            for(int z = 0; z < list->at(j)->getLastName().size(); z++){
                if(list->at(j)->getLastName().at(z).unicode() < list->at(j+1)-
>getLastName().at(z).unicode()){
                    Dentist *temp = new Dentist;
                    *temp = *list->at(j);
                    list->removeAt(j);
                    list->insert(j+1,temp);
                    break;
                }
                else if(list->at(j)->getLastName().at(z).unicode() == list->at(j+1)-
>getLastName().at(z).unicode())
                    continue;
                else
                    break;
            }
        }
    }
}

void MainWindow::ShowDentist(QList<Dentist*> *temp)
{
    QVBoxLayout *VLayout = new QVBoxLayout;
    QWidget* scrollAreaContent = new QWidget;
    ui->scrollArea->setWidget(scrollAreaContent);
    for(int i = 0; i < temp->size();i++){
        QVBoxLayout *DentistInfo = new QVBoxLayout;
        QHBoxLayout *HLayout = new QHBoxLayout;

        QPixmap pix(":/Images/ImageDentists/ImageDentists/" + temp->at(i)-
>getPhoto());

        QLabel *Icon = new QLabel;
        QPushButton *detailed = new
QPushButton(interface.object().value("Detailed").toString());

```



```

        QPushButton *comments = new
QPushButton(interface.object().value("Comments").toString());
        detailed->setStyleSheet("color: black; background-color: rgb(199, 199,
199); font: 14pt;");
        comments->setStyleSheet("color: black; background-color: rgb(199,
199, 199); font: 14pt;");
        try{
            if (pix.isNull())
            {
                throw QString (":/Images/ImageDentists/ImageDentists/no-
avatar.PNG");
            }
        }
        catch(QString path){
            MyException ex(pix, path);
        }

        Icon->setPixmap(pix.scaled(240,180));

        QLabel *Info = new QLabel;
        Icon->setAlignment(Qt::AlignRight);
        DentistInfo->setAlignment(Qt::AlignLeft);
        Info->setText(Info->text() +
interface.object().value("FirstName").toString() + ": " + temp->at(i)->getFirstName()
+ "\n");
        Info->setText(Info->text() +
interface.object().value("LastName").toString() + ": " + temp->at(i)->getLastName()
+ "\n");
        Info->setText(Info->text() +
interface.object().value("Patronymic").toString() + ": " + temp->at(i)-
>getPatronymic() + "\n");
        Info->setText(Info->text() +
interface.object().value("Specialization").toString() + ": " + temp->at(i)-
>getSpecialization() + "\n");
        Info->setText(Info->text() +
interface.object().value("WorkExperience").toString() + ": " +
QString::number(temp->at(i)->getWorkExperience()) + "\n");
        Info->setText(Info->text() +
interface.object().value("ContactsDentist").toString() + ": " + temp->at(i)-
>getContactsDentist() + "\n");
        Info->setText(Info->text() + interface.object().value("Rating").toString() +
": " + QString::number(temp->at(i)->getRating()) + "\n");

```

```

        Info->setText(Info->text() +
interface.object().value("NameOfDentistry").toString() + ": " + temp->at(i)-
>getDentistry().getNameOfDentistry());
        HLayout->addWidget(Icon);
        DentistInfo->addWidget(Info);
        QHBoxLayout *ButtonLayout = new QHBoxLayout();
        ButtonLayout->setAlignment(Qt::AlignLeft);
        ButtonLayout->addWidget(detailed);
        ButtonLayout->addWidget(comments);
        DentistInfo->addLayout(ButtonLayout);
        HLayout->addLayout(DentistInfo);
        Info->setMaximumHeight(220);
        VLayout->addLayout(HLayout);
        Dentist *tmp = temp->at(i);
        connect(detailed, &QPushButton::clicked, [this, tmp]() { this-
>SendDetailed(tmp); });
        connect(comments, &QPushButton::clicked, [this, tmp]() { this-
>SendComments(tmp); });
    }
    scrollAreaContent->setLayout(VLayout);
}

void MainWindow::SendDetailed(Dentist* temp){
    DetailedDentist *DD = new DetailedDentist();
    DD->sendDentist(temp, interface);
    DD->show();
}

void MainWindow::SendComments(Dentist* temp){
    CommentsDentist *cm = new CommentsDentist();
    cm->sendDentist(temp, interface);
    cm->show();
}

void MainWindow::on_actHelp_triggered()
{
    Help *t = new Help();
    t->SendSpecialization(&specialization, interface);
    t->show();
}

void MainWindow::on_actionEng_triggered()
{
    SaveDentist();
    QLabel *tmp = new QLabel();

```

```

    ui->scrollArea->setWidget(tmp);
    pathFileDentist = "../BotDetective-main/Date/dentist_eng.json";
    pathFileDentistry = "../BotDetective-main/Date/dentistry_eng.db";
    pathFileSpec = "../BotDetective-main/Date/spec_eng.json";
    pathInterface = "../BotDetective-main/Date/interface_eng.json";
    OpenInterface();
    OpenSpec();
    OpenDentistry();
    OpenDentist();
}

void MainWindow::on_actionUkr_triggered()
{
    SaveDentist();
    QLabel *tmp = new QLabel();
    ui->scrollArea->setWidget(tmp);
    pathFileDentist = "../BotDetective-main/Date/dentist_ukr.json";
    pathFileDentistry = "../BotDetective-main/Date/dentistry_ukr.db";
    pathFileSpec = "../BotDetective-main/Date/spec_ukr.json";
    pathInterface = "../BotDetective-main/Date/interface_ukr.json";
    OpenInterface();
    OpenSpec();
    OpenDentistry();
    OpenDentist();
}

void MainWindow::SaveDentist(){
    try{
        QFile fileDentist(pathFileDentist);
        if(!fileDentist.open(QIODevice::WriteOnly | QFile::Text)){
            throw 7;
        }
        else
        {
            QJsonDocument newDateDentist;
            for(int i = 0; i < listDentist.size(); i++){
                QVariantMap map;
                map.insert("firstName", listDentist.at(i)->getFirstName());
                map.insert("lastName", listDentist.at(i)->getLastName());
                map.insert("patronymic", listDentist.at(i)->getPatronymic());
                map.insert("years", listDentist.at(i)->getYears());
                map.insert("specialization", listDentist.at(i)->getSpecialization());
                map.insert("aboutDentist", listDentist.at(i)->getAboutDentist());
                map.insert("contactsDentist", listDentist.at(i)->getContactsDentist());
            }
        }
    }
}

```

```

        map.insert("workExperience", listDentist.at(i)->getWorkExperience());
        map.insert("photo", listDentist.at(i)->getPhoto());
        map.insert("idOfDentistry", listDentist.at(i)->getIdOfDentistry());
        QVariantMap insideMap;
        QJsonArray commentsArray;
        for(int j = 0; j < listDentist.at(i)->getListComments().size(); j++){
            insideMap.clear();
            insideMap.insert("commentatorName", listDentist.at(i)-
>getListComments()[j].getCommentatorName());
            insideMap.insert("comment", listDentist.at(i)-
>getListComments()[j].getComment());
            insideMap.insert("rating", listDentist.at(i)-
>getListComments()[j].getRating());
            commentsArray.append(QJsonObject::fromVariantMap(insideMap));
        }
        map.insert("comments", commentsArray);
        QJsonObject json = QJsonObject::fromVariantMap(map);
        QJsonArray toWrite = newDateDentist.array();
        toWrite.append(json);
        newDateDentist.setArray(toWrite);
    }
    fileDentist.write("{\"Doctors\":" + newDateDentist.toJson()+"}");
}
fileDentist.close();
}
catch(int error){
    MyException ex(error, interface);
}
}

```

```

void MainWindow::on_actDevelops_triggered()
{
    Developers *t = new Developers();
    t->SendInterface(interface);
    t->show();
}

```

```

void MainWindow::on_actionDentists_triggered()
{
    QList<Dentist*> temp;
    hashTableByName.searchAll(&temp);
    switch (ui->SortBy->currentIndex()) {
        case 0: sortByRating(&temp);
        break;
    }
}

```

```

        case 1 : sortByAscending(&temp);
        break;
        case 2: sortByDescending(&temp);
        break;
    }
    ShowDentist(&temp);
}

void MainWindow::on_actionDentistry_triggered()
{
    QVBoxLayout *VLayout = new QVBoxLayout;
    QWidget* scrollAreaContent = new QWidget;
    ui->scrollArea->setWidget(scrollAreaContent);
    for(int i = 0; i < listDentistry.size();i++){
        QVBoxLayout *DentistryInfo = new QVBoxLayout;
        QPushButton *ShowDentistFromDentistry = new
        QPushButton(interface.object().value("ShowDentistFromDentistry").toString());
        ShowDentistFromDentistry->setStyleSheet("color: black; background-
color: rgb(199, 199, 199); font: 14pt;");
        ShowDentistFromDentistry->setMaximumWidth(375);
        QTextBrowser *Info = new QTextBrowser;
        DentistryInfo->setAlignment(Qt::AlignCenter);
        Info->append(interface.object().value("NameOfDentistry").toString() + ": "
+ listDentistry.at(i)->getNameOfDentistry());
        Info->append( interface.object().value("WorkingHours").toString() + ": " +
listDentistry.at(i)->getWorkingHours());
        Info->append(interface.object().value("Address").toString() + ": " +
listDentistry.at(i)->getAddress());
        Info->append(interface.object().value("AboutUs").toString() + ": " +
listDentistry.at(i)->getAboutUs());
        Info->append(interface.object().value("ContactsDentistry").toString() + ": "
+ listDentistry.at(i)->getContactsDentistry());
        Info->setFixedHeight(250);
        DentistryInfo->setContentsMargins(0,10,0,10);
        DentistryInfo->addWidget(Info);
        DentistryInfo->addWidget(ShowDentistFromDentistry);
        QString tmp = listDentistry.at(i)->getNameOfDentistry();
        connect(ShowDentistFromDentistry, &QPushButton::clicked, [this,
tmp]() { this->SendShowDentistFromDentistry(tmp); });
        VLayout->addLayout(DentistryInfo);
    }
    scrollAreaContent->setLayout(VLayout);
}

```

```

void MainWindow::SendShowDentistFromDentistry(QString temp){
    SearchByDentistry(temp);
}

void MainWindow::on_actionSpecializations_triggered()
{
    QVBoxLayout *VLayout = new QVBoxLayout;
    QWidget* scrollAreaContent = new QWidget;
    ui->scrollArea->setWidget(scrollAreaContent);
    for(int i = 0; i < specialization.size();i++){
        QVBoxLayout *DentistryInfo = new QVBoxLayout;
        QPushButton *ShowDentistFromSpecialization = new
        QPushButton(interface.object().value("ShowDentistFromSpecialization").toString());
        ShowDentistFromSpecialization->setStyleSheet("color: black; background-
        color: rgb(199, 199, 199); font: 14pt;");
        ShowDentistFromSpecialization->setMaximumWidth(375);
        QTextBrowser *Info = new QTextBrowser;
        DentistryInfo->setAlignment(Qt::AlignCenter);
        Info->append(specialization.at(i).first);
        Info->setMaximumHeight(50);
        DentistryInfo->setContentsMargins(0,10,0,10);
        DentistryInfo->addWidget(Info);
        DentistryInfo->addWidget(ShowDentistFromSpecialization);
        QString tmp = specialization.at(i).first;
        connect(ShowDentistFromSpecialization, &QPushButton::clicked, [this,
        tmp](){ this->SendShowDentistFromSpecialization(tmp); });
        VLayout->addLayout(DentistryInfo);
    }
    scrollAreaContent->setLayout(VLayout);
}

void MainWindow::SendShowDentistFromSpecialization(QString temp){
    SearchBySpecialization(temp);
}

void MainWindow::on_actionClose_triggered()
{
    QApplication::quit();
}

```

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```