МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. В. ДАЛЯ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ

КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ПРОГРАМУВАННЯ

До захисту допускається.
Завідувач кафедри ІТП

Лифар В. О

«_____» _____2023 р.

ДИПЛОМНИЙ ПРОЕКТ (РОБОТА) БАКАЛАВРА

ПОЯСНЮВАЛЬНА ЗАПИСКА

НА ТЕМУ:

Медична інформаційна система

Освітній ступінь – «бакалавр»

Спеціальність 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

Виконав: студент групи ІПЗ-19д _____ К.Р. Бондаренко
(підпис) (ініціали і прізвище)

Керівник _____ В.Г. Іванов
(підпис) (ініціали і прізвище)

Завідувач кафедри ІТП _____ В.О.Лифар
(підпис) (ініціали і прізвище)

Київ- 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет Інформаційних технологій та електроніки

Кафедра Інформаційних технологій та програмування

Освітній ступінь бакалавр

Спеціальність 121 Інженерія програмного забезпечення

(шифр і назва)

ЗАТВЕРДЖУЮ:

завідувач кафедри ІТП

Лифар В.О

«_____» _____2023 р.

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) БАКАЛАВРА

Бондаренка Кирила Романовича

(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка медичної інформаційної системи

Керівник проекту (роботи) Іванов В. Г.,к.т.н.,доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від 26.04. 2023р. №240/15.15-ОД

3. Вихідні дані до роботи: матеріали переддипломної практики.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

а) аналіз предметної області;
б) вибір та обґрунтування програмних засобів розробки системи;
в) розробка системи;
г) охорона праці;
д) висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 25.03.2023

Керівник

(підпис)

Завдання прийняв до виконання

(підпис)

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Визначення літературних джерел для проекту	03.05.2023-04.05.2023	
2	Визначення потреб проекту	04.05.2023-08.05.2023	
3	Архітектурне проектування інформаційної системи	03.05.2023-03.06.2023	
4	Визначення технологічного стеку для інформаційної системи	08.05.2023-20.05.2023	
5	Ознайомлення з новими принципами технологічного стеку	20.05.2023-28.05.2023	
6	Створення графічних матеріалів для інформаційної системи	28.05.2023-30.05.2023	
7	Розробка інформаційної системи	28.05.2023-30.05.2023	
8	Тестування та впровадження інформаційної системи	31.05.2023-05.06.2023	

Здобувач вищої освіти _____ К.Р.Бондаренко
(підпис) (ініціали, прізвище)

Керівник _____ В.Г. Іванов
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Робота містить: 48 сторінок основного тексту, 11 сторінок додатків, 1 таблицю, рисунків 21, 4 використаних джерел.

Метою випускної кваліфікаційної роботи є вивчення особливостей розробки інформаційної системи для сфері медицини та охорони здоров'я . Дана інформаційна система спрямована на спрощення роботи медичних працівників.

В ході розробки медичної інформаційної системи були проаналізовані відмінності Qt та MFC при розробці десктопних додатків, визначені існуючі види інформаційних систем.

В результаті виконаної роботи, було реалізовано інформаційну систему, яка покращує швидкість та якість роботу як впевнених користувачів ПК, так і недосвідчених робітників медичного закладу.

Зроблено детальний опис процесу розробки інформаційної системи, а також, продемонстрована робота готового десктопного додатку

Система реалізована відповідно всім вимогам технічного завдання.

ЗМІСТ

ВСТУП 7
1. АНАЛІТИЧНИЙ ОГЛЯД.....9
1.1 Розробка десктопних програм та їх особливості.....9
1.2 Відмінності MFC, Qt Widgets та Qt Quick при розробці десктопних програм мовою C++.....10
1.3 Розробка додатку Qt Quick на мобільні пристрої.....15
1.4 Кросплатформове складання програмного забезпечення 17
1.5 C++.....18
1.6 Використання розподіленої систему управління версіями програмного забезпечення 20
2. МОДЕЛЬ ПРОЕКТУ ТА ПОСТАНОВКА ЗАДАЧІ 21
2.1 Модель 21
2.2 Збір вимог 24
2.3 Проектування 26
2.4 Етап даталоічного проектування.....28
3. ВИБІР МЕТОДІВ ТА РЕАЛІЗАЦІЯ ПОСТАВЛЕНИХ ЗАДАЧ 32
3.1 Розробка на Qt Quick 32
3.2 Використання патерну проектування Singleton.....33
3.3 Реалізація та створення проекту.....36
ВИСНОВКИ 47
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....48
ДОДАТКИ 49

ВСТУП

Актуальність. На даний момент впровадження та розробка інформаційних систем є ефективною практикою автоматизації будь-якого процесу. Їх використання покращує результативність виконання поставлених задач. Забезпечення надійності а саме, мінімізація збоїв в роботі цілої структури в якою було впроваджену систему - є головною задачею яку ця система вирішує.

Медична сфера є важливою частиною нашого життя і вона з кожним днем, потребує покращення інструментів. Прикладом цього може бути запровадження інформаційних технологій у методи обробки персональних даних та документообігу. Це допоможе надійно і зручно зберігати, обробляти та редагувати інформацію пацієнтів та персоналу медичного закладу. Більшість людей не мають бажання йти у медичні заклади для отримання медичних послуг через незручність їх надання яке пропонує медична сфера. Наслідком цього є те, що суспільство звертається до медичних закладів тільки у крайніх випадках, що ускладнює або унеможливлює вирішення проблем зі здоров'ям. Надання медичних послуг у зручному форматі та своєчасне отримання результатів медичного дослідження зможе врятувати життя пацієнту.

Об'єкт дослідження: медична інформаційна система.

Предмет дослідження: розробка медичної інформаційної системи для медичних закладів України.

Мета дослідження: метою даної дипломної роботи є створення медичної інформаційної системи

Задачі дослідження:

1. Ознайомитися з сучасними інформаційними системами і сформувати основні потреби для неї
2. Сформувати стек технологій на основі аналізу проблем та потреб які можуть виникнути під час розгортання цієї системи у закладах охорони здоров’я України
3. Провести проектування модулів системи з розрахунком розширення та кастомізації для можливості використання системи у будь-яких регіонів України.
4. Розробити медичну інформаційну систему
5. Виконати тестування інформаційної системи та її модулі

1. АНАЛІТИЧНИЙ ОГЛЯД

1.1 Розробка десктопних програм та їх особливості

Написання програм під ПК є складним процесом який потребує ретельного та детального планування архітектури додатку. В залежності від пріоритетних потреб інформаційної системи обирається певний стек технологій. Основні особливості проектування десктопних програм:

- Кросплатформність десктопних програм. Властивість яка дозволяє ПЗ працювати на різних програмних або апаратних платформах. При розробці кросплатформної інформаційної системи треба враховувати можливість підтримки та запуску програмного забезпечення на різних платформах.
- Сучасні фреймворки та інші додаткові інструменти для розробки системи. Використання передових технологій при розробці інформаційної системи гарантує безперебійність та зручну підтримку ПЗ протягом усього часу впровадження та його використання за рахунок постійних оновлень технологій які задіяні під час розробки. Це дає можливості для покращення самої системи відносно оновлень які були отриманні від розробників тієї чи іншої технології та гарантія того, що не доведеться у майбутньому шукати альтернативу іншим інструментам через її припинення підтримки зі сторони розробників.

1.2 Відмінності MFC, Qt Widgets та Qt Quick при розробці десктопних програм мовою C++

У сучасному світі, розробка додатків з GUI без використання додаткових бібліотек або фреймворків є складним завданням. Якщо враховувати використання складних та більш гнучких для користувачів інтерфейс, то побудувати та впровадити таке рішення без використання додаткових інструментів неможливо.

Прикладом потреби у використанні таких додаткових інструментів є елемент реагування графічного інтерфейсу додатка на дії користувача. У цьому випадку, частина графічного інтерфейсу повинна реагувати відносно дій користувачів за допомогою якоїсь концепції. Більш відома та неідеальна концепція це - функції зворотнього виклику. Вона заснована на механізмі використання звичайних функцій які повинні викликатися у результаті дії користувача. Використання такого методу взаємодії користувача з графічним інтерфейсом є поганим рішення у сучасній розробці ПЗ:

- Впровадження такої концепції значно ускладнює вихідний код програми, роблячи його менш зрозумілим і наслідком цього є уповільнення процесу розробки та підтримки ПЗ.
- Відсутня можливість проводити перевірку типів значень, що повертаються під час виклику функції. Під час використання такої концепції, в усіх випадках повертається вказівник на порожній тип void. При взаємодії користувача з графічним інтерфейсом функція буде викликана і бібліотека не може перевірити типи які були передані в функцію. Наслідком цього може стати критичний збій системи.

- Елементи графічного інтерфейсу користувача тісно пов’язані з функціональними частинами програми. Це ускладнює розширення функціоналу додатку за рахунок виникнення сильної залежностей між класами та окремими модулями.

Існують бібліотеки які значно спрощують процес розробки додатків з графічним інтерфейсом. Одна з найперших і найпопулярніших на сьогоднішній день є бібліотека Microsoft Foundation Class Library (MFC). Сама бібліотека MFC є надбудовою що надає доступ до функцій Windows реалізованих мовою C що змушує розробників час від часу використовувати застарілі структури що не вписуються в рамки концепції обєктно-орієнтованого підходу. Бібліотека не є об'єктно-орієнтованою бо вона створювалась розробниками які на той час не знали основних принципів об'єктно-орієнтованого програмування (ООП). Один із головних принципів ООП є інкапсуляція – доступ до стану об'єкта напряму заборонено, і ззовні з ним можна взаємодіяти виключно через заданий інтерфейс (відкриті поля та методи), що дозволяє знизити зв'язність[1]. Даний принцип ООП в MFC недотриманний.

Основна концепція побудови зв’язку в MFC між графічним інтерфейсом користувача і функціональною частиною програми є спеціальні макроси які мають назву – карти повідомлень.

```
class CPhotoStylerApp : public CWinApp {
public:
    CPhotoStylerApp();
public:
    virtual BOOL InitInstance();

    afx_msg void OnAppAbout();
    afx_msg void OnFileNew();

    DECLARE_MESSAGE_MAP()
};
BEGIN_MESSAGE_MAP(CPhotoStylerApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    ON_COMMAND(ID_FILE_NEW, OnFileNew)
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()
```

Рисунок 1.1 Синтаксис MFC

Подібні конструкції є незручними для програмістів через складність аналізу та сприйняття коду. Синтаксис MFC змушує розробників при внесенні незначних змін у додатку модифікувати код у декілька частина системи що є поганою ознакою для інформаційної системи метою якої є розширення та модифікація її під різний регіони.

Проблема розширення мови C++ для розробки кросплатформових додатків вирішенна за допомогою фреймворку Qt. Він має метаоб'єктний компілятор (MOC). Компілятор аналізує всі клас на наявність в них спеціального макросу Q_OBJECT і впроваджує в окремий файл усю необхідну інформацію. Розробник не повинен створювати цей файл і власноруч записувати усю інформацію. Це відбувається автоматично, що гарантує правильність формування цього файлу і застерігає від майбутніх сбоїв системи.

За допомогою макросу Q_OBJECT можна отримати підтримку різних служб системи метаоб’єктів Qt у тому числі – механізм слотів і сигналів. Цей механізм є ідеальною альтернативою старої концепції функцій зворотного виклику. Він виступає гнучким інструментом під час релізації архітектури додатку будь-якої складності через інтуїтивне використання. Прикладом цього є звичайний дзвінок на телефон: почувши дзвінок людина відповідає на вхідних виклик. Відносно сигналів і слотів це можна описати так: об’єкт “телефон” відправив сигнал “дзвінок”, на який об’єкт “людина” відреагувала слотом “відповісти на вхідний виклик”.

Використання сигналів і слотів надає розробнику наступні переваги:

- кожен клас, успадкований від QObject, може мати будь-яку кількість сигналів та слотів.
- повідомлення, що надсилаються за допомогою сигналів, можуть мати безліч аргументів будь-якого типу.
- слот може приймати повідомлення від багатьох сигналів, що належать різним об'єктам
- з'єднання та роз'єднання сигналів і слотів можна проводити в будь-якій точці програми;
- сигнали та слоти є механізмами, що забезпечують зв'язок між об'єктами. Більш того, цей зв'язок може виконуватися між об'єктами, що знаходяться в різних потоках.
- при знищенні об'єкта відбувається автоматичне роз'єднання всіх сигнально слотових зв'язків. Це гарантує, що сигнали не надсилатимуться до неіснуючих об'єктів.
- використання сигналів і слотів є повністю об’єкто-орієнтованим рішенням. Наслідком цього є притриманність стандартів ООП та

мови C++ що дає змогу правильно побудувати інформаційну систему та зручно підтримувати її.

- можливість з'єднувати між собою два сигнала для уникнення зайвого коду.

Як Qt Widgets так і Qt Quick підтримують механізм сигналів і слотів але ці два видів додатку зовсім різні. Не менш важливою вимогою для медичної інформаційної системи є здатність обробляти та зберігати велику кількість данних і одночасно мати можливість створювати або відтворювати анімацію в графічному інтерфейсі користувача.

Qt Widgets має безліч стандартних віджетів для реалізації нескладного дизайну. Також є можливість реалізувати власний віджет на основі стандартних але для більш гнучкого дизайну цього недостатньо. При розробці додатку, розробник і графічний дизайнер тісно взаємопрацюють. Нажаль, ця співпраця не є завжди комфортною та продуктивною. Найчастіше, дизайнери роблять занадто складний дизайн для додатку який іноді буває неможливо реалізувати навіть при використанні будь-яких існуючих інструментів. Наслідком цього є те, що доводиться швидко переробляти дизайн модульних частин додатку що негативно впливає на роботу команди вцілому.

Qt Quick спеціально створенний для втілення складних рішень з точки зору дизайну. Він використовує описову мову – QML. В ньому описується як виглядають і взаємодіють один з одним елементи інтерфейсу користувача. В силу свого описового характеру ця мова не передбачає використання конструкцій програмування - наприклад, циклів. Наслідком цього є те, що дизайнер може самостійно розробити та налаштувати дизайн додатку для клієнта не витрачаючи час та сили розробників. Крім того, є

засоби для конвертування дизайну, створеного в графічних редакторах Adobe Photoshop і GIMP прямо в QML. Вбудованість мови JavaScript в QML дає змогу використовувати можливості мови C++ для оптимізаціїї додатку за допомогою системи сигналів і слотів. Структура qml файлу виглядає наступним чином:

```
import QtQuick 2.8
Item {
    width: 200
    height: 200
    Rectangle {
       width: parent.width
       height: parent.height
       onWidthChanged: {
           console.log("width changed:" + width)
       }
       onHeightChanged: {
           console.log("height changed:" + height)
       }
    }
}
```

Рисунок 1.2 Структура qml файлу

1.3 Розробка додатку Qt Quick на мобільні пристрої

На сьогоднішній день, у кожної людини є мобільний телефон і цей прийстрій є невід'ємною частиною сучасного життя. Для зручного користування системою яка надає життєво важливі послуги суспільству треба розглядати можливість майбутньої реалізації інформаційної системи на мобільні пристрої так як це збільшить охоплення користувачів, які можуть користуватися додатком бо не усі мають персональні комп’ютери і користування мобільними додатками є більш зручним аніж ПК.

При розробці додатку на мобільні пристрої виникає складне питання на яку мобільну платформу треба обрати для початку впровадження інформаційної системи на ці пристрої. Адже всі вони такі різні, і інструменти для розробки теж дуже відрізняються один від одного. На їх вивчення необхідно витратити багато часу або розширити штат працівників. Наприклад, для створення програм для Android зазвичай використовується мова Java. Для iOS це може бути Objective або Swift. Використовувати одночасно дві мови, а також пов'язані з ними фреймворки та програмувати на них стає складним завданням. Це змушує проводити пошук нових розробників. Qt підтримує три популярні мобільні операційні системи. При впевненому володінні цим фреймворком, ніколи не буде поставати питання: яку з платформ вибрати, тому що рішення по відбору зводиться до знання деяких специфічних нюансів для цільової платформи та перекомпіляція програми.

Розробка графічного інтерфейсу для мобільного додатку за допомогою фреймворка Qt зводитись до використання Qt Widgets та Qt Quick. Використання Qt Quick є більш пріорітетним так як він створювався спеціально для цієї мети але це не означає що не можна використовувати Qt Widgets. Він також підтримує адаптивну розмітку графічного інтерфейсу і має безліч прикладів успішного його використання у комерційній розробці додатків. На відміну від цього, використання Qt Quick дає змогу легко та швидко розробити мобільний додаток для інформаційної системи так як, при використанні його у розробці, використовується зручна описова мова QML з мовою C++ що дає нам змогу легко перенести дизайн та деяку функціональну частину з десктопної версії на мобільну без великих витрат

ресурсів на проектування та розробку нової архітектури чи адаптацію дизайну.

1.4 Кросплатформове складання програмного забезпечення

Є багато утиліт та інструментів які призначені для складання проекту написанні мовою C/C++. Найбільш популярні це Autoconf та CMake. Autoconf у поєднанні з Automake дуже схожий на функціонал та принцип дії CMake, забезпечуючи деякі з тих самих функцій, що і CMake. Основний і вагомий недолік використання Autoconf є те, що для використання цієї утиліти та додаткових інструментів з якими вона працює, на платформі Windows потрібна встановлення багатьох додаткових інструментів, яких від самого початку немає у Windows. Він так само як і CMake, підтримує параметри користувача, але можливість підтримувати залежні параметри, коли один параметр залежить від іншої властивості – відсутня.

Фреймворк Qt підтримує складання проекту за допомогою утиліти CMake. Для використання цієї утіліти використовується текстовий файл формату txt який описує параметри, файли та ресурси проекту для складання готового додатку. У проекті цих файлів можуть бути декілька – для розбиття проекту на окремі модулі. Наприклад, якщо виникне потреба у написанні користувальницької розмітки для всього додатку то це рішення можна відокремити у модуль, який потім, при складанні программи буде частиної всієї системи.

Основна перевага цієї утілити у тому, що ми можемо побудувати зручну для кожного регіону архітектуру медичної інформаційної системи використвуючи зручні інструменти для розбиття проекту на декілька підпроектів. Тобто, налаштовувати файли із системно-залежною

інформацією, такою як розташування файлів даних та інше. CMake може створювати файли заголовків, що містять таку інформацію, як шляхи до файлів даних та іншу інформацію у вигляді #define макросів. Також є можливість легко створити кросплатформовий додаток. Він може контролювати які файли, модулі або бібліотеки будуть влючені під ту чи іншу специфічну платформу що дає розробникам єдину систему складання програми для більшої кількості існуючих мобільних та настільних платформ. Це є важливим елементом, особливо, коли над проектом працює велика команда розробників і планується розробляти та випускати додаток на кілька платформ одночасно.

1.5 C++

C++ - мова програмування, що підтримує об'єктно орієнтоване та узагальнене програмування. Розробленна в 1983 році датським програмістом Б'ярном Страуструпом.

Мова C++ відноситься до сім'ї мов з C-подібним синтаксисом. Основна і головна відмінність цієї мови від C це підтримка об'єктно-орієнтованого програмування. Наслідком цього, мова C++ є провідним у розробці десктопних, мобільних додатків і низькорівневих рішень.

Одним із головним критерієм успішного і ідеального додатку у сучасному світі розробки ПЗ є достатня оптимізація. Наразі є багато персональних комп’ютерів з різними технічними параметрами. Через це, розробники прагнуть якнайбільше охопити користувачів які зможуть використовувати їх додаток. Підтримка ООП з можливістю ефективно отпимізовувати програмне забезпечення за рахунок особливостей та інструментів мови таких як: власний контроль над життєвим циклом об’єкту

класса, перерозподіл ємності та використаної пам'яті контейнерів STL, механізм розумних вказівників, перевантаження операторів та семантика переміщення, дає розробникам більше можливостей для покращення системи за рахунок гнучкою оптимізації і ідеальної архітектури додатку. Наслідком цього, мова C++ є дуже популярною в області розробки ігор. Бо вони найбільш потребують оптимізацію.

Підтримка поліморфізму за допомогою динамічної типізації яка реалізована у механізмі віртуальних та шаблонних функцій, бібліотека STL та часткові модулі бібліотеки Boost які є частинами стандартами мови C++, підтримка старих функцій реалізовані з мови C, перевизначення конструкторів класу та велика кількість фрейморків і бібліотек які розширюють можливості мови C++ все це дає перевагу над іншими мовами під час вибору як головного інструменту для розробки інформаційної системи з графічним інтерфейсом і не тільки.

Головним недоліком цієї мови є відсутність гнучких інструментів для створення сучасного додатку з складною користувацькою анімацією. Є багато додаткових бібліотек та фреймворків таких як Qt, але вони не є стандартом мови C++ і виступають як розширення мови. Більшість інших мов так само не мають подібних інструментів і майже в усіх випадках при проектуванні та розробки великого проекту використовуються сторонні бібліотеки, фейворки та інші допоміжні інструменти. Під час останнього виходу стандарту C++20, ми можемо побачити, що більшість функціональних частин бібліотеки Boost вже є невід'ємною частиною самої мови і тому, у майбутньому, це може статись з подібними бібліотеками та фреймворки як Qt, які будуть поступово ставати стандартними елементами та функціоналом мови.

1.6 Використання розподіленої системи управління версіями програмного забезпечення

Під час розробки програмного забезпечення є потреба у збереженні прогресу та виділення стадій його розвитку. При зберіганні проекту локально є ризик втратити увесь прогрес створювання ПЗ. Щоб уникнути таких ризиків, розробники використовють систему управлінням версіями програмного забезпечення. Вона гарантує легкий доступ до актуальної версії проекту , надає можливості працювати над одним проектом одразу кільком програмістам не заважаючи один одному.

При розробці медичної інформаційної системи буде використовуватися система git. Основні переваги цієї системи:

- розподілена розробка
- можливість розгалуження
- зручне створення патчів програмного забезпечення
- простота та комфортне управління вихідним кодом

2. МОДЕЛЬ ПРОЕКТУ ТА ПОСТАНОВКА ЗАДАЧІ

2.1 Модель

Модель розробки програмного забезпечення - це структурована кількість процесів або методології, які використовуються для розробки проекту в залежності від завдань проекту. Основні моделі розробки програмного забезпечення:

- модель кодування та усунення помилок.
- каскадна модель
- розробка через тестування
- інкрементна модель
- ітераційна модель
- спіральна модель
- модель хаосу
- прототипна модель

Реалізація проекту “Медична інформаційна система” буде основана на каскадній моделі. Це модель процесу розробки програмного забезпечення, в якій процес розробки виглядає як потік, що послідовно проходить фази

аналізу вимог, проектування, реалізації, тестування, інтеграції та підтримки [2].

Основна відмінність каскадної моделі від інших є те, що всі етапи розробки йдуть один за одним. Під час розробки додатку, для переходу на наступний етап треба повністю виконати поточний. Тобто, перейшовши на наступний етап неможливо повернутися назад. Тому, перед тим як перейти на наступну фазу розробки треба пройти перевірку поточної і погодити її. Усі плани, вимоги та завдання проекту описують у документах. Усі учасники дотримуються формальних правил і не можуть їх змінювати під час роботи. Оскільки не можна повернутись до попереднього етапу, вимоги до проекту після затвердження не змінюються.

На рисунку 2.1. наведені етапи класичної каскадної моделі:

1. Збір вимог. Тут збирають вимоги до проекту, оформлюють їх у технічне завдання, в якому розписано план робіт, передбачувані ризики та ролі в команді;
2. Проектування. Тут визначають головні принципи продукту, наприклад логіку програмного забезпечення. Під ці принципи підбирають інструменти, наприклад мови програмування, бібліотеки, фреймоворки.
3. Розробка. Йде процес розробки програмного забезпечення відносно готового технічного завдання.
4. Тестування. Тут перевіряють продукт на відповідність технічним завданням. Проводиться пошук помилок і їх усунення.
5. Експлуатація та підтримка. Тут випускають та підтримують програмне забезпечення. Виправляють помилки на основі відгуків користувачів, модифікують модулі або функції ПЗ.

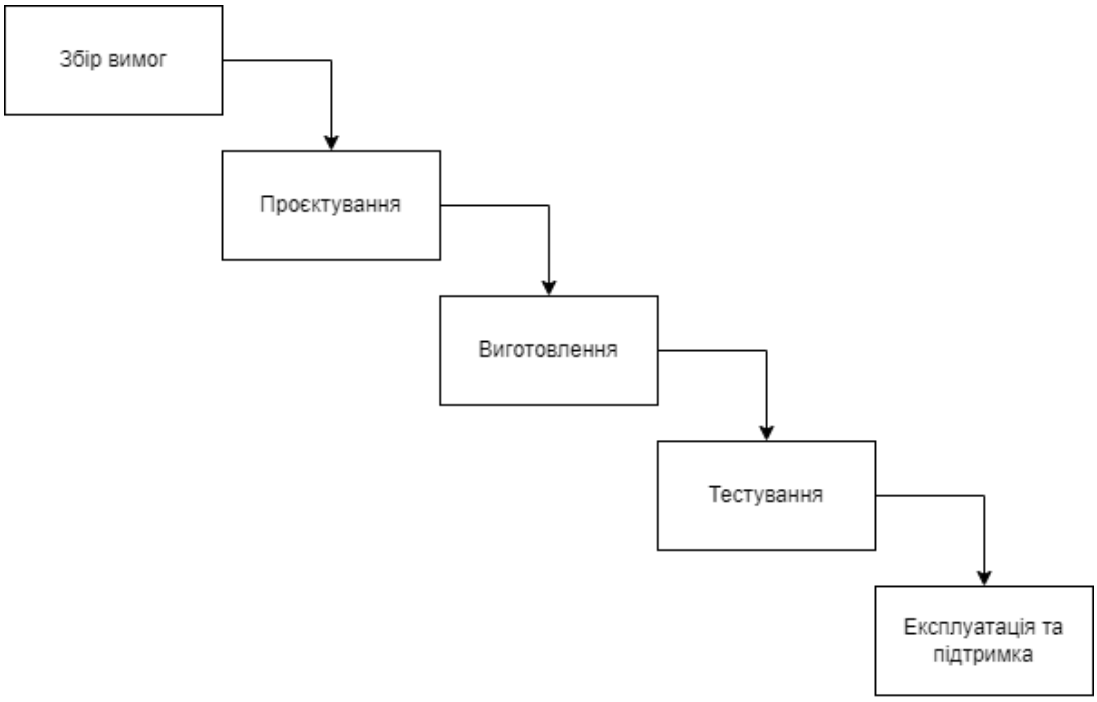

Рисунок 2.1 Каскадна модель

Основні переваги каскадної моделі:

- усі процеси зарегламентовані та описані що не завдасть шкоди для термінів та якості робіт
- Розробники працюють за чітким планом. Через це сам процес розробки додатку є прозорим та передбачуваним.
- Вимоги щодо проєкту не змінюються під час розробки. Так як під час роботи не можна перейти на попередній етап розробки то вимоги після затвердження не змінються що є наслідком уникнення зайвих затримок релізу готового рішення.
- Терміни і бюджет проєкту затвердженний на початку розробки додатку що дає змогу уникнути проблем з фінансуванням та

термінами розгортки системи через неочікуванні зміни планів замовника.

2.2 Збір вимог

Виходячи з того, що медична інформаційна система повина мати можливість розгортатися незалежно від клієнта та регіона, зберігати конфедеційність даних лікарів та пацієнтів та бути інтуїтивним під час використання маємо наступні вимогу :

- Дизайн. Він повинен бути одночасно сучасним та інтуїтивно зрозумілим. Не весь медичний персонал є впевненими користувачами персональних комп’ютерів тому потрібно досягти інтуїтивності у дизайні графічного інтерфейсу додатку для надання можливості використовувати систему для користувачів, які не мають добрих навичок володінням програм та інформаційними технологіями загалом.
- Зручна авторизація. Треба зробити зручний вхід до власного кабінету відносно типу користувача і у разі надання неправильних даних повідомити користувача які саме дані були веденні неправильно. Якщо користувач забув дані потрібно надати можливість для відновлення доступу до власного кабінету.
- Надійність. Одна з головних задач медичної інформаційної системи повинна бути забезпечення конфеденційності данних пацієнтів та результатів лабораторних досліджень. Методи захисту данних повині бути найдійні і ефективні.
- Можливість швидкого і легкого розгортання медичної інформаційної системи. Так як система планується розготатится

під будь-який регіон України, де можуть бути незначні але специфічні зміни, треба забезпечити таку можливість з використанням мінімальних витрат часу та невеликою переробкою модулів системи.

- Оптимізація. Інформаційна система повина бути якнайбільше оптимізована. У деяких медичих закладах використовуються застарілі персональні комп’ютери які не розраховані підтримувати сучасні додатки з використанням складної анімації. Система повинна використовувати мінімальні ресурси ПК визначивши його можливості і на підставі цього вимикати, приховувати деяку анімацію або елементи інтерфейсу які не є критично важливими і призначенні тільки для візуального покращення додатку.
- Швидкодія. Додаток повинен використовувати швидкі і маловитратні по пам'яті алгоритми обробки та сортування даних.
- Логічна і зрозуміла структура системи. Медична інформаційна системи повинна складатися з декілька модулів. Кожен з цих модулів при складанні системи буде часткою цілого. При такій структурі його буде легше підтримувати та додавати або змінювати функціонал.
- Адаптованість. Треба адаптувати розмітку та анімацію системи для різних дисплеїв ПК так як, через погану адаптацію буде некомфортно користуватися системою. Також, через це можуть зникати деякі частини користувацького графічного інтерфейсу що унеможливить подальшу роботу з програмним забезпеченням.

- Відокремлення бізнес-логіки системи від його графічного представлення через побудування грамотної об'єктної ієрархії за допомогою патернів проектування.
- Можливість продовжувати працювати при при критичних збоях програми. Так як медичні послуги в Україні і у будь-якому світі надаються у будь-який момент часу, то медична інформаційна система повинна мати можливість швидко виходити з критичного стану для відновлення роботи.

2.3 Проектування

Проектування — це процес створення прототипу, прообразу майбутнього об'єкта, стану та способів його виготовлення.[3] Це є важливим моментом під час всієї розробки програмного забезпечення. Від того, як від самого початку, було спроектовану інформаційну систему залежить подальший хід розробки. Якщо не врахувати важливі та критичні моменти або не прораховувати можливі ризики, помилки які можуть виникнути в процесі створення системи є великий шанс отримати серйозну проблему яка сповільнить подальші процеси і змусить повністю переглядати архітектуру та стратегію реалізації проекту. Також, потрібно завчасно визначитись з якими технологіями, бібліотеками, допоміжними інструментами та фреймворками доведеться працювати. Врахувати їх актуальність та доречність використання у проекті. Тому заздалегідь треба визначити критерії та потреби програмного забезпечення і тільки після цього приступати до побудови архітектури програмного забезпечення.

Інтерфейс:

1. Розробка унікального, сучасного та інтуїтивно зрозумілого інтерфейсу з використанням описової мови QML.
2. Впровадження адаптивності у графічний інтерфейс користувача.
3. Оптимізований код та нескладний дизайн графічної частини додатку для швидкодії роботи системи.
4. Використання платформо-незалежного механізму системи ресурсів фреймворку Qt.
5. Використання графічних редакторів таких як Adobe Photoshop для розробки дизайну та зручної конвертації його в QML.
6. Розробка оригінального та унікального логотипу інформаційної системи.

Програмна частина

1. Розробка медичної інформаційної системи на базі Qt Quick Application.
2. Використання популярного фреймворку Qt.
3. Використання утиліти CMake для складання та налаштування системи проекту.
4. Використання систем контролю версію програмного забезпечення git для збереження прогресу розробки і зручної модифікації системи.
5. Використання СКБД MySQL для збереження даних пацієнтів та лікарів.
6. Впровадження модульної структури у проект засобами Qt для створення власних модулів.
7. Використання патернів проектування для уникнення сильних залежностей між модулями програми.

8. Використання механізму сигналів-слотів для реалізації взаємодії користувача через графічний інтерфейс.

2.4 Етап даталоічного проектування

Проаналізувавши потреби медичної інформаційно системи в якості системою управлінням базами даних був обраний – MySQL.

MySQL – це безкоштовна реляційна система управлінням базами даних яка здійснює роботу з базами даних, заснованих на двовимірних таблицях. Ця СКБД є популярним інструментом для роботи з базами даних за рахунок появи у доступному просторі великою кількості документації, посібників та навчальних курсів. Створення користувацьких плагінів та розширень з метою зручної роботи з цією системою додало системі більшої популярності.

Основні перевари MySQL:

- Простота у використанні. MySQL досить легко встановлюється, а наявність безлічі плагінів, допоміжних програм та великою за об’ємом документації спрощує роботу з базами даних.
- Великий функціонал. Система MySQL має практично весь необхідний інструментарій, який може знадобитися в реалізації практично будь-якого проекту.
- Безпека. Система спочатку створена таким чином, що багато вбудованих функцій безпеки в ній працюють за умовчанням і при необхідності є можливість реалізувати та впровадити власний алгоритм шифрування даних.
- Масштабованість. Будучи дуже універсальною СУБД, MySQL однаково легко може бути використана для роботи і з малими, і з великими обсягами даних.

- Швидкість. Висока продуктивність системи забезпечується за рахунок спрощення деяких стандартів, що використовуються в ній.
- Підтримка кількох типів таблиць: MyISAM, InnoDB.
- Робота з довгим текстом. Функції COMPRESS() та UNCOMPRESS() дозволяють зберігати в БД довгий текст без втрати продуктивності.

Для даної медичної інформаційної системи можна виділити так сутності :

- сутність «Пацієнт».
- сутність «Лікар».
- сутність «Адміністратор».
- сутність «Амбулаторна картка».
- сутність « Лабораторні дослідження».
- сутність « Запис».

Існує безліч інструментів для керуванням MySQL. У сучасних процесах розробки програмного забезпечення, розробники використовують інструменти з графічною підтримкою так як, це прискорює процес роботи і дає змогу уникнути помилок проектування баз даних. Найбільш популярні інструменти керування з графічним інтерфейсом для MySQL:

- phpMyAdmin.
- MySQL Query.
- MySQL Workbench.
- dbForge Studio for MySQL.

Для більш зручної роботи з MySQL був обраний графічний інструмент dbForge Studio for MySQL. Основна особливість цього інструменту є зручне та легке використання навіть якщо нема досвіду зі створенням та адмініструванням баз даних. dbForge Studio підтримує два види створювання баз даних – візуальний спосіб та на основі скрипта. Перший спосіб ідеально підходить для розробників які не мають сильних навичок конструювання БД на основі скрипта. Це допоможе уникнути більшості помилок які можуть стати критичними особливо, якщо вони будуть поміченими до впровадження усієї датологічної моделі в інформаційну систему. Другий спосіб є класичним але використовуючи його ми так само можемо побачити результат виконання насписаного скрипта що є додатковою перевіркою на помилки та правильності досягнутої мети. Основні переваги dbForge Studio:

- великі можливості імпорту експорту у різні формати
- наявність вбудованої мови макрокоманд
- орієнтованість на користувача з різною професійною підготовкою

Після опису сутностей і зв’язків між ними створимо даталогічну модель для основної бізнес-логіки:

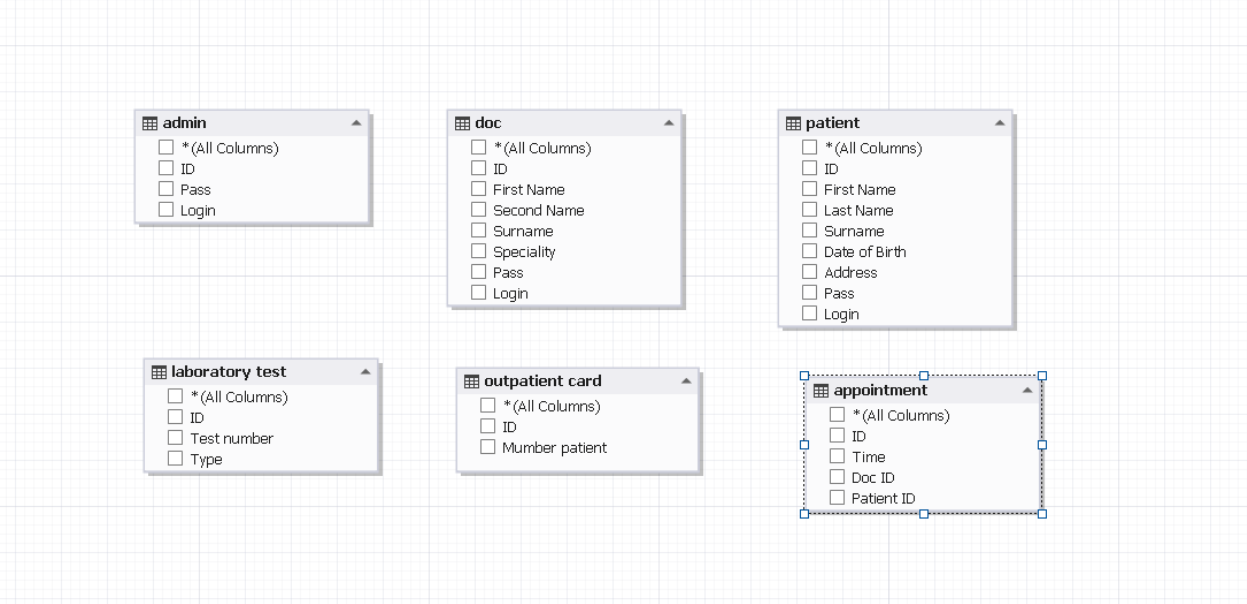

Рисунок 2.2 Даталогічна модель бази даних проекту

3. ВИБІР МЕТОДІВ ТА РЕАЛІЗАЦІЯ ПОСТАВЛЕНИХ ЗАДАЧ

3.1 Розробка на Qt Quick

Додаток Qt Quick можна розробляти на будь-якій операційній системі. Для нього можна використовувати безліч зовнішніх інтегрованих середовищ розробки (IDE) але Qt має свою власну IDE під назвою Qt creator. Також, при розробці на Qt для власного комфорту або для додаткового функціоналу які потребує інформаційна система можна легко встановити плагіни. Один з популярних додаткових інструментів який покращює програмування графічного інтерфейсу додатку є Qt Designer.

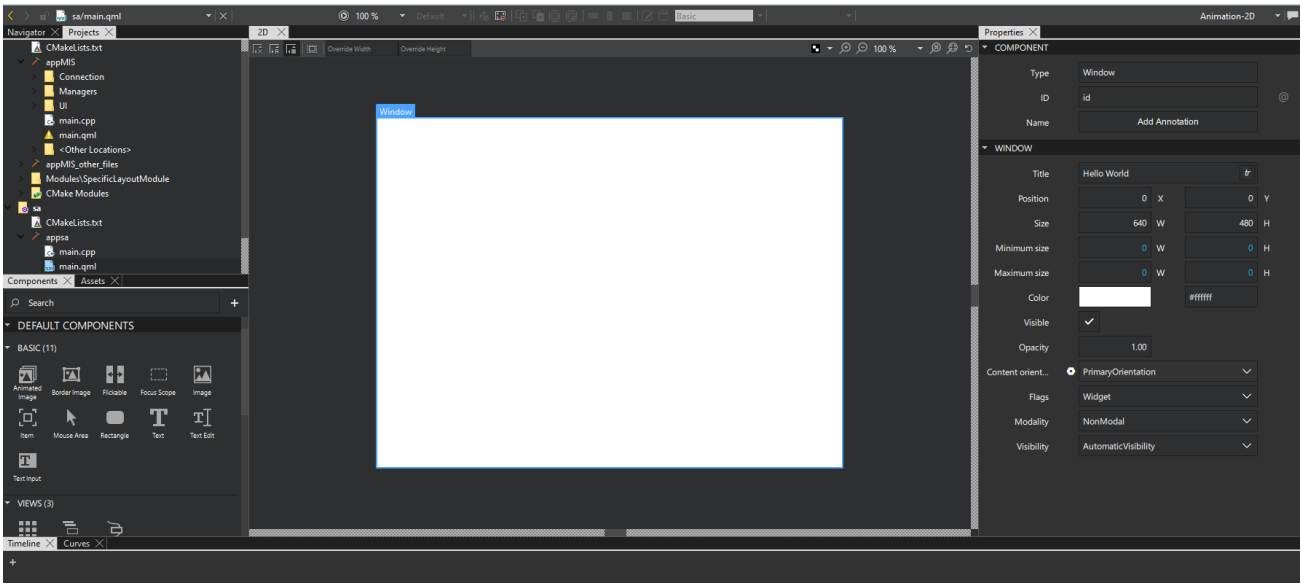

Рисунок 3.1 Середовище Qt Designer

Він надає можливість створювати адаптивну розмітку графічного інтерфейсу за допомогою інтерактивних інструментів які працюють з елементами. Основні можливості і переваги використання Qt Designer

- легке налаштування розмітки для додатку.

- зручний пошук елементів і автоматичне додовання модулів які потрібні для тих самих елементів.
- зручне створення нових та модифікація вже існуючих параметрів елементів.
- формування ієрархії елементів конкретного вікна програми.
- налаштування механізму сигналів-слотів.

Для більш гнучкого налаштування дизайну та функцій елементів графічного інтерфейсу використовується мова QML.

3.2 Використання патерну проектування Singleton

Призначення

Гарантує, що клас має лише один екземпляр, і надає до нього глобальну точку доступу.

Мета використання патерну

Для деяких класів важливо, щоб існував лише один екземпляр. Як гарантувати, що клас має єдиний екземпляр і що цей екземпляр легко доступний? Глобальна змінна дає доступ до об'єкта, але не забороняє інстанціювати клас у кількох примірниках. Більше вдале рішення - сам клас контролює те, що в нього є тільки один екземпляр може заборонити створення додаткових екземплярів, перехоплюючи запити на створення нових об'єктів, і він же здатний надати доступ до свого екземпляра. Це і є призначення патерна Singleton.

Застосовність

Патерн Singleton використовують коли:

- повинен бути рівно один екземпляр деякого класу, легко доступний всім клієнтам;
- єдиний екземпляр повинен розширюватися шляхом породження підкласів, і клієнтам потрібно мати можливість працювати з розширеним екземпляром без модифікації свого коду[4]

Структура патерну Singleton

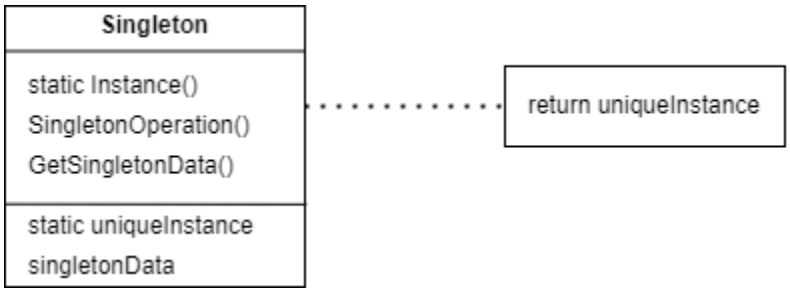

Рисунок 3.2 Структура Singleton

Учасники

Визначає операцію Instance, яка дозволяє клієнтам отримувати доступ до єдиного екземпляра. Instance - це операція класу, то є статична функція-член у C++. Може відповідати за створення власного унікального екземпляра.

Відношення

Клієнти мають можливість отримувати доступ до екземпляра класу Singleton тільки через його операцію Instance.

Основні переваги використання патерну Singleton:

1. зменшення числа імен. Паттерн Singleton - крок уперед порівняно з глобальними змінними. Він дозволяє уникнути засмічення

простору імен глобальними змінними, у яких зберігаються унікальні екземпляри;

2. від класу Singleton можна породжувати підкласи. Доступ до екземпляру класу Singleton можна використовувати для отримання даних або виклику функції без повторної ініціалізації та створення екземпляра класу
3. більша гнучкість, ніж в операцій класу. Ще один спосіб реалізувати функціональність Singleton - використовувати операції класу, тобто статичні функції-члени в C++. Але обидва ці прийоми перешкоджають зміні дизайну, якщо потрібно дозволити наявність кількох примірників класу. Крім того, статичні функції-члени в C++ не можуть бути віртуальними, так що їх не можна поліморфно замісити в підкласах.
4. Можливість використовувати більше одного екземпляра. Патерн може легко дозволити появу більше одного екземпляру Singleton

Реалізація

Гарантування єдиного екземпляра. Паттерн Singleton влаштований так, що той єдиний екземпляр, який є у класу, - звичайний, але більше одного екземпляра створити не вдасться. Найчастіше для цього ховають операцію, що створює екземпляри, за операцією класу (тобто статичною функцією-членом або методом класу), яка гарантує створення трохи більше одного екземпляра. Ця операція має доступ до перемінної, де зберігається унікальний екземпляр, і гарантує ініціалізацію змінної цим екземпляром перед поверненням її клієнту. При такому підході можна не сумніватися, що одинак буде створений та ініціалізований перед першим використанням.

3.3 Реалізація та створення проекту

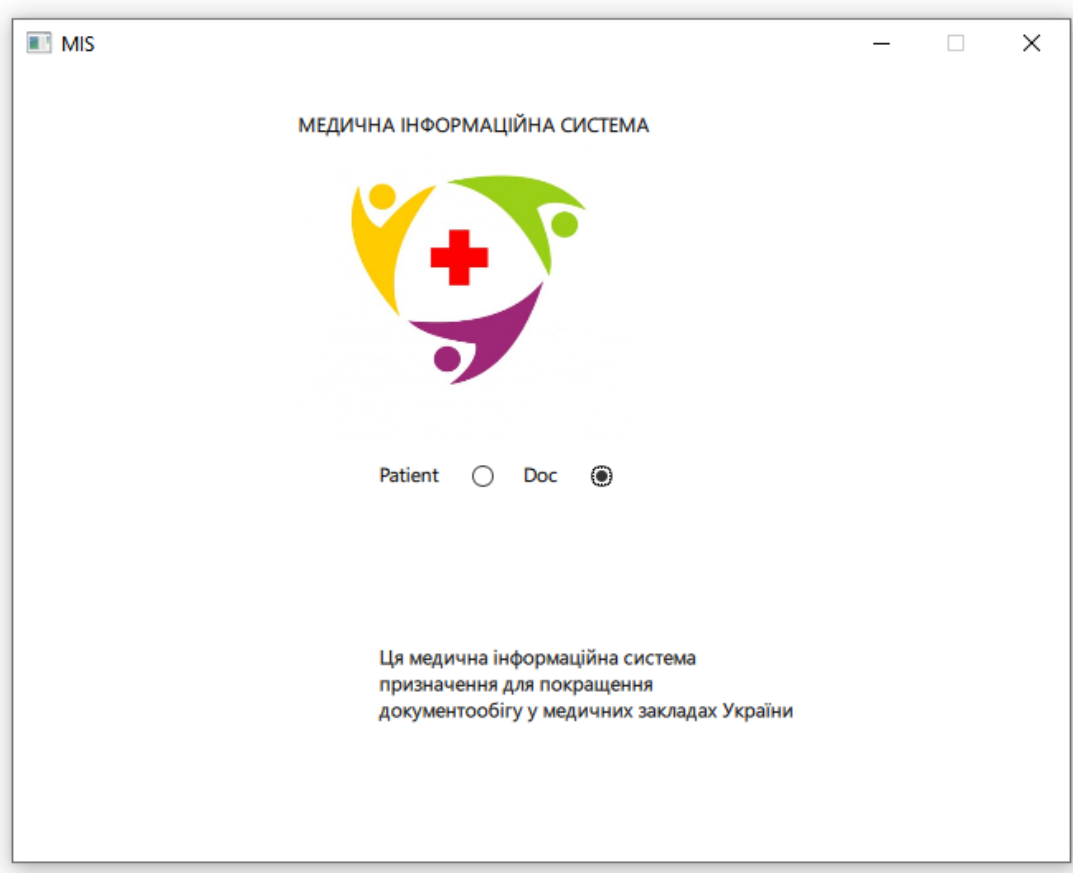

Рисунок 3.3. Головне меню інформаційної системи

На зображені 3.3 представлено головне меню медичної інформаційної системи.

Головне меню складається з логотипу додатку і вибори режиму для авторизації пацієнта або лікаря відповідно.

Логотип завантажується за допомогою системи ресурсів Qt. Для її використання треба визначити усі ресурси які ми будемо використовувати в додатку у файлі ресурсів. У даній системи цей файл має назву “MainResource”. Він має схожу структуру та синтаксис XML файлу. Для

визначення файлів ресурсів які ми хочемо використовувати у власному додатку треба її перерахувати в області <qresource>. На рисунку 3.4 зображенно файл ресурсів який додає ресурс логотипу у додаток

```
<!DOCTYPE RCC>
<RCC version="1.0"/>
<qresource>
    <file>Content/Images/logo.jpg</file>
</qresource>
```

Рисунок 3.4. Ресурс логотипу додатку

Тепер ми можемо використовувати цей ресурс для відображення логотипу у додатку відносно файлового розташування:

```
Image
{
    width: 200
    height: 200
    source: "Content/Images/logo.jpg"
}
Text
```

Рисунок 3.5. Встановлення логотипу

Вибір режиму авторизації реалізований за допомогою системи сигналів-слотів. Для з’єднання графічної і функціанальної частини за допомогою цього механізмом треба враховувати особливості інтеграції qml та C++. Модульна частина у системі яка має назву “ModuleConnection” реалізовує усі інтеграції і підключення qml файлів які у свою чергу, реалізують графічний інтерфейс додатку. Для з’єднання сигналів і слотів

треба мати вказівник на об’єкт який відсилає сигнал і клас який приймає цей сигнал. Для пошуку у конкретному вікні додатку об’єкта, який відсилає сигнал використовуються вбудована функція findChild(). Ця функція шукає вказаний об’єкт за властивістю objectName і при успішному знаходженні повертає на нього вказівник. Шукається об’єкт відносно вікна який повинен був бути створенний за допомогою функції create(). Приклад пошуку об’єкта за властивістю:

```
QObject* DocBtn = PObjectDoc->findChild<QObject*>("btn_doc");
```

Рисунок 3.6. Пошук об’єкта за властивістю

Отримавши посилання на об’єкт, можна використовувати механізм сигналів і слотів. Слоти для цих об’єктів розташовані у окремому модулі який має назву “WindowSwitcherManager”. Головна мета цього модулю контролювати і здійстнювати переключення вікон системи. Слот реалізується як звичайний метод класу але він не має можливості повертати будь-яке значення або мати параметри за змовчуванням:

```
void WindowSwitcherManager::SwitchPatientMode()
{
    const QUrl url("qrc:/MIS/UI/PatientWindow.qml");

    if(window_obj != nullptr)
        window_obj->close();

    patient_window = CreateCurrentObjWindow(url);

    ModuleConnection conn;

    if(!conn.CreateConnectionAuthorizationPatient(patient_window))
    {
    }
}
```

Рисунок 3.7. Реалізація слоту для режиму “Пацієнт”

З’єднання сигналу зі слотом представленно на рисунку 3.8

```
if(QObject::connect(DocBtn, SIGNAL(activateDocMode(QString,QString,QString)),
    WindowManager, SLOT(AutorizationDocMode(QString,QString,QString))))
```

Рисунок 3.8. Підключення сигналу до слоту

Вибираючи режим для авторизації з’являється відповідно окно авторизації для пацієнта або лікаря. Представленна класична авторизація потребує для входу логін та пароль користувача. У випадку авторизації для лікаря потрібно додатково вказати ID лікаря який відображає його відповідну спеціалізацію.

Рисунок 3.9. Вікно авторизації для лікаря

При наданні невірних даних для авторизації користувач побачить відповідну помилку. Механізм перевірки реалізований на перевірку даних які були отриманні від TextField елементу QML і порівняння їх з даними отриманих від запросу до баз даних.

```
if(login == loginQ && pass ==passQ)
{
    qDebug() << "Sur";
    const QUrl url("qrc:/MIS/UI/AccountPatient.qml");
    QQmlComponent current_comp_window(&switcher_engine, url);
    QObject* ObjectWindow =current_comp_window.create();

    account_patient = ObjectWindow;
    flag = true;
}
else
{
    qDebug() << "Error";
    conn.CloseMySQLConnection();
    const QUrl url("qrc:/MIS/UI/ErrorMessage.qml");
    QQmlComponent current_comp_window(&switcher_engine, url);
    QObject* ObjectWindow =current_comp_window.create();
    account_doc = ObjectWindow;
}
```

Рисунок 3.11. Перевірка даних для авторизації

Підключення до баз даних використовується за допомогою вбудованих функцій Qt. Для підлючення феймворк використовую класичний набір драйвері баз даних. Qt не має драйверів для MySQL тому його потрібно збирати власноруч. Для цього знадобиться інструмент CMake. Після успішної збірки драйверу для баз даних ми можемо підключатися до неї. Для підключення треба вказати так дані як : хост, ім’я бази даних, логін та пароль:

```
bool ModuleConnection::ConnectToMySQL()
{
    db = QSqlDatabase::addDatabase("QMYSQL");
    db.setHostName("localhost");
    db.setDatabaseName("mis");
    db.setUserName("root");
    db.setPassword("root");
    return db.open();
}
```

Рисунок 3.12. Підключення до бази даних

При успішній авторизації в режимі “Лікаря” ми можемо побачити основну інформацію про лікаря та назначені йому записи пацієнтів. Також, ми маємо можливість створити новий запис на прийом натиснувши кнопку “Створити запис”. Ми вказуємо дату зустрічі для пацієнта і після цього до баз даних записується дата, спеціальність та П.І.Б лікаря і пацієнта.

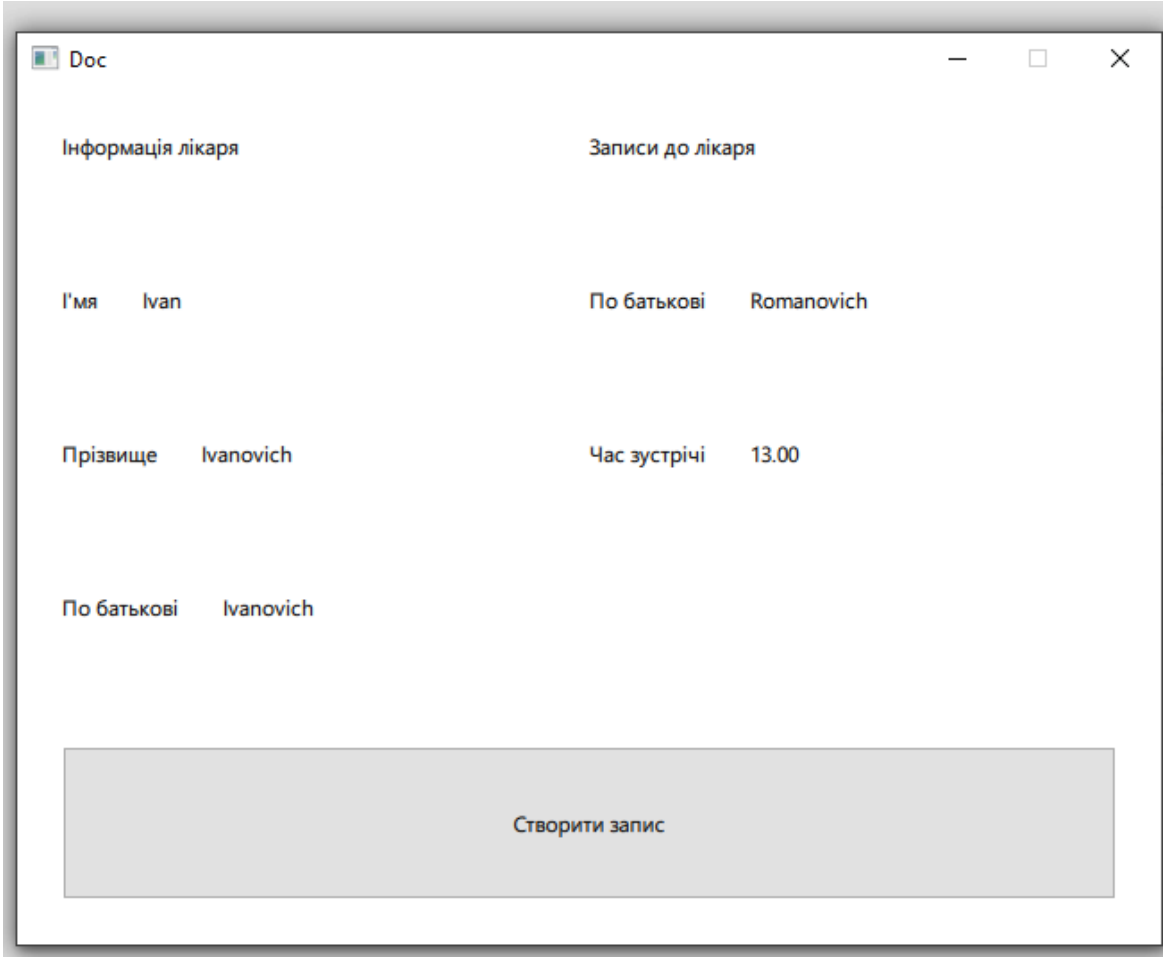

Рисунок 3.13 Кабінет лікаря

У режимі пацієнт ми можемо побачити основну інформацію яка стосується пацієнта та редагувати її. Також є можливість переглядати усі записи до лікаря у відповідній категорії

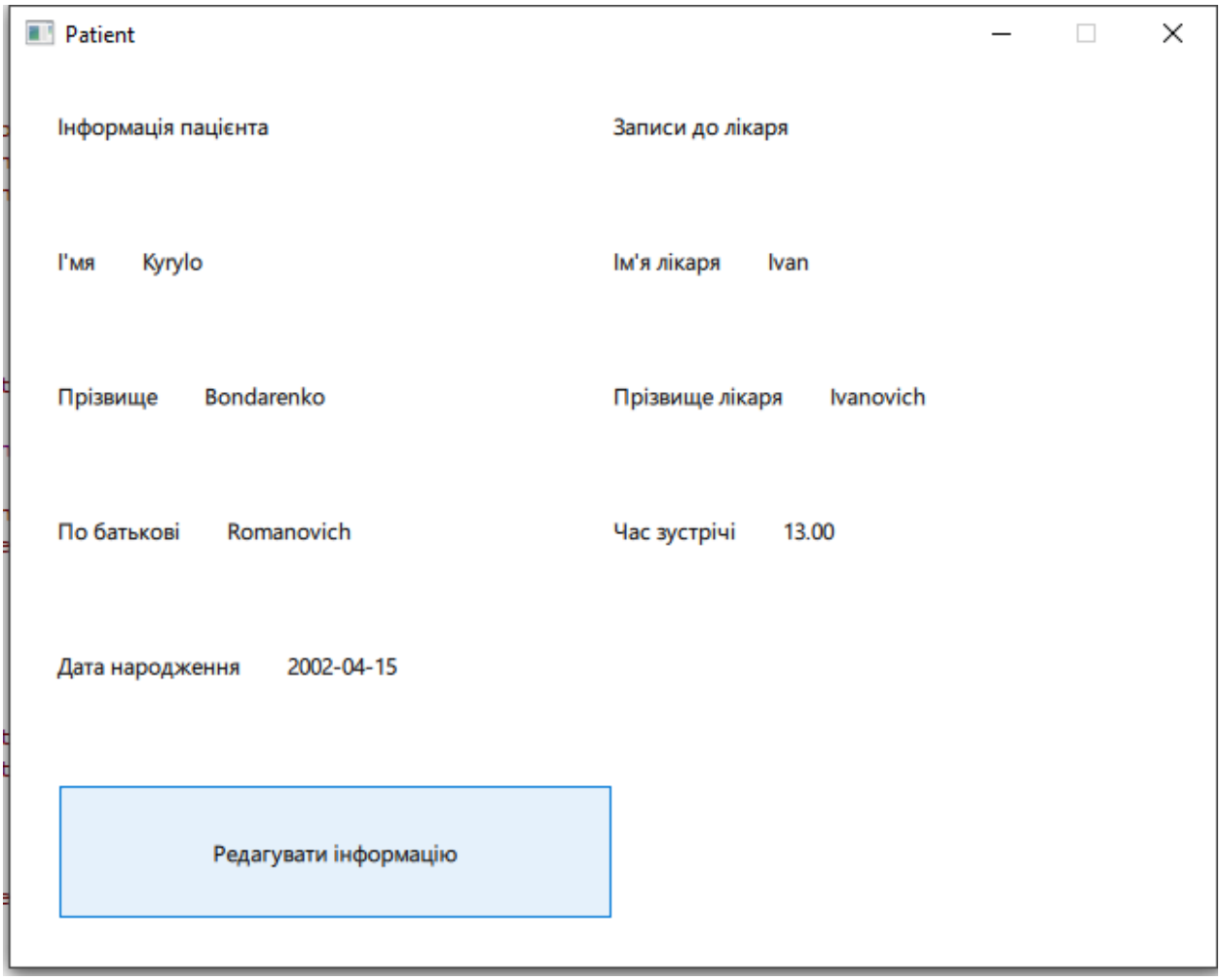

Рисунок 3.14. Кабінет пацієнта

Для зручної розмітки в медичній інформаційній системі використовуються написаний власноруч модуль під назвою “SpecificLayoutModule”. Він схожий за принципом дії на класичний GridLayout але в ньому реалізована підтримка columnSpan та rowSpan. Що дає змогу робити “порожні” місця або розтягувати місце у розмітки без зайвих відступів які є під час використання GridLayout. Також, є можливість

створювати різні версії модулів які будуть мати деякі особливості або нові функції. Можна використовувати будь-яку реалізацію модулю вказавши її версію під час підключення. Часткова реалізації модулю представлена на рисунку 3.15

```
{
    id: layoutGrid

    property int column: 1
    property int row: 1


        onChildrenChanged: updatePreferredSizes()
        onWidthChanged: updatePreferredSizes()
        onHeightChanged: updatePreferredSizes()
        onColumnChanged: updatePreferredSizes()
        onRowChanged: updatePreferredSizes()

    function updatePreferredSizes()
    {
        var cellWidth = layoutGrid.width / column
        var cellHeight = layoutGrid.height / row
        var lenghtGhildrenLayout = Object.keys(layoutGrid.children).length


        if( lenghtGhildrenLayout === 0 || lenghtGhildrenLayout === undefined )
        {
          return;
        }

        for(var keyWhile = 0; keyWhile !== lenghtGhildrenLayout; keyWhile++)
        {
          var obj = layoutGrid.children[keyWhile]
          if(obj !== undefined)
          {
              var tempColumnValue = obj.Layout.column
              var tempRowValue = obj.Layout.row
              var tempColumnSpan = obj.Layout.columnSpan
              var tempRowSpan = obj.Layout.rowSpan

              obj.x = tempColumnValue * cellWidth
              obj.y = tempRowValue * cellHeight
              obj.height = tempRowSpan * cellHeight
              obj.width = tempColumnSpan * cellWidth
          }

        }
    }
```

Рисунок 3.15. Модуль розмітки

Для підключення модулю треба дотримуватися певноъ структури: у кожному модулі повинен бути файли qml та файл складання модулю CMakeLists у якому вказується основні параметри цього модулю:

```
qt_add_library(specific_layout_module STATIC)

qt_add_qml_module(specific_layout_module
   URI SpecificLayoutModule
   VERSION 1.0
   QML_FILES
       GridLayoutUtil.qml

    RESOURCE_PREFIX /
   )
```

Рисунок 3.16 CMakeLists файл модуля

При цьому, автоматично генерується плагін для цього модулю який треба підключити у файлі головної програми main.cpp. Після цього, у файлі кореневого CMakeLists треба підлючити модуль до додатку. Підключення модуля реалізовується за допомогою комбінації add_subdirectory у якому вказується розміщення директорії файлів модуля та target_link_libraries який вказує посилання на плагін модуля. Для реалізації множиної версії модуля треба вказати їх перелік у файлі опису модуля CMakeLists вказавши версію яка буде використовуватися як індетифікатор модуля і файли які відносяться до тієї чи іншої версії. Якщо є загальні файли для модуля то при визначенні кожної версії модуля треба їх вказувати кожного разу. Якщо їх не вказати можна отримати критичну помилку під час підключення або використання модуля.

```
add_subdirectory(Modules/SpecificLayoutModule/GridLayout)
target_link_libraries(appMIS PRIVATE specific_layout_moduleplugin)
```

Рисунок 3.17. Підключення модуля до програми

Для підключення модуля використовується ключове слово import у якому вказується назва самого модулю і його версія.

ВИСНОВКИ

В процесі виконання дипломної роботи було розглянуто питання архітектури медичної інформаційної системи, вибору правильних інструментів, бібліотек і фреймворків, які найбільше підходять під завдання реалізації інформаційної системи.

На основі аналізу, необхідних до зберігання даних, була спроектована даталогічна модель БД. Виходячи з даних даталогічної моделі БД, була спроектована фізична БД.

Виходячи з вимог інформаційнох системи і списку завдань, був обраний і вивчений фреймвок Qt, його особливості, такі як створення додатку на основі Qt Quick та механізм сигналів і слотів.

За результатами вивчених матеріалів і досліджень, була створенна медичная інформаційна систем, яка може бути розгорнута для будь-якого регіону України що дозволяє зберігати і оброблювати конфедеційну і не тільки інформацію пацієнтів та лікарів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке ООП? - schoolboyprog10.blogspot.com [Електронний ресурс] - Режим доступу: https://schoolboyprog10.blogspot.com/p/blog-page_21.html
2. Каскадна модель - um.co.ua [Електронний ресурс] - Режим доступу: http://um.co.ua/8/8-9/8-94331.html
3. Що таке проектування? - https://dl.kpt.sumdu.edu.u [Електронний ресурс] - Режим доступу: https://dl.kpt.sumdu.edu.ua/mod/book/tool/print/index.php?id=12796
4. Каскадна модель - um.co.ua [Електронний ресурс] - Режим доступу: http://um.co.ua/8/8-9/8-94331.html

ДОДАТКИ

ДОДАТОК А. СТРУКТУРА ТАБЛИЦЬ В БАЗІ ДАНИХ

Таблиця А.1

Схема	**Таблиця**	Атрибут	Тип
mis	patient	ID	INT
mis	patient	First Name	DATE
mis	patient	Last Name	VARCHAR(255)
mis	patient	Date of Birth	VARCHAR(255)
mis	patient	Address	VARCHAR(255)
mis	patient	Pass	VARCHAR(255)
mis	patient	Login	VARCHAR(255)
mis	doc	ID	INT
mis	doc	First Name	VARCHAR(255)
mis	doc	Second Name	VARCHAR(255)
mis	doc	Surname	VARCHAR(255)
mis	doc	Speciality	VARCHAR(255)
mis	doc	Pass	VARCHAR(255)
mis	doc	Login	VARCHAR(255)
mis	appointment	ID	INT
mis	appointment	Time	VARCHAR(255)
mis	appointment	Doc ID	INT
mis	appointment	Patient ID	INT

ДОДАТОК Б1. МОДУЛЬ ПІДКЛЮЧЕННЯ СИГНАЛІВ-СЛОТІВ ТА БАЗ ДАНИХ

```
bool ModuleConnection::CreateConnectionModeSwitcher(QObject
*PObject)
{
   QObject* Rbnt = PObject->findChild<QObject*>("rb_patient_mode");
   QObject* Lbnt = PObject->findChild<QObject*>("rb_doc_mode");

   WindowSwitcherManager* WindowManager =
WindowSwitcherManager::CreateInstancePtr();

   if(connect(Rbnt, SIGNAL(changeAuthorizationMode()),
WindowManager, SLOT(SwitchPatientMode())) &&
      connect(Lbnt, SIGNAL(changeAuthorizationMode()),
WindowManager, SLOT(SwitchDocMode())))
   {
      return true;
   }
   else
   {
      return false;
   }

}

bool ModuleConnection::CreateConnectionAuthorizationPatient(QObject*
PObjectPatient)
{
   QObject* PatientBtn = PObjectPatient-
>findChild<QObject*>("btn_patient");


   WindowSwitcherManager* WindowManager =
WindowSwitcherManager::CreateInstancePtr();
```

```
   if(QObject::connect(PatientBtn,
SIGNAL(activatePatientMode(QString,QString)),
      WindowManager, SLOT(AutorizationPatientMode(QString,QString))))
   {
      return true;
   }
   else
   {
      return false;
   }
}

bool ModuleConnection::CreateConnectionAuthorizationDoc(QObject
*PObjectDoc)
{
   QObject* DocBtn = PObjectDoc->findChild<QObject*>("btn_doc");

   WindowSwitcherManager* WindowManager =
WindowSwitcherManager::CreateInstancePtr();

   if(QObject::connect(DocBtn,
SIGNAL(activateDocMode(QString,QString,QString)),
      WindowManager,
SLOT(AutorizationDocMode(QString,QString,QString))))
   {
      return true;
   }
   else
   {
      return false;
   }
}

bool ModuleConnection::ConnectToMySQL()
```

```
{
   db = QSqlDatabase::addDatabase("QMYSQL");
   db.setHostName("localhost");
   db.setDatabaseName("mis");
   db.setUserName("root");
   db.setPassword("root");
   return db.open();
}

void ModuleConnection::CloseMySQLConnection()
{
   db.close();
}
```

ДОДАТОК Б2. ПЕРЕКЛЮЧЕННЯ ВІКНОН ДОДАТКУ

```
void WindowSwitcherManager::SwitchPatientMode()
{
   const QUrl url("qrc:/MIS/UI/PatientWindow.qml");

    if(window_obj != nullptr)
       window_obj->close();

   patient_window = CreateCurrentObjWindow(url);

   ModuleConnection conn;

   if(!conn.CreateConnectionAuthorizationPatient(patient_window))
   {
   }
}

void WindowSwitcherManager::SwitchDocMode()
{
   const QUrl url("qrc:/MIS/UI/DocWindow.qml");
```

```
    if(window_obj != nullptr)
        window_obj->close();

    doc_window =  CreateCurrentObjWindow(url);

    ModuleConnection conn;

    if(!conn.CreateConnectionAuthorizationDoc(doc_window))
    {

    }
}

void WindowSwitcherManager::AutorizationPatientMode(QString login,
QString pass)
{

    ModuleConnection conn;
    bool flag = false;
    if(conn.ConnectToMySQL())
    {
        QSqlQuery query("SELECT Login, Pass FROM patient");
        while (query.next())
        {
            QString loginQ = query.value(0).toString();
            QString passQ =  query.value(1).toString();

            if(login == loginQ && pass ==passQ)
            {
                qDebug() << "Sur";
                const QUrl url("qrc:/MIS/UI/AccountPatient.qml");
                QQmlComponent current_comp_window(&switcher_engine, url);
                QObject* ObjectWindow =current_comp_window.create();
```

```
            account_patient = ObjectWindow;
            flag = true;
        }
        else
        {
            qDebug() << "Error";
            conn.CloseMySQLConnection();
            const QUrl url("qrc:/MIS/UI/ErrorMessage.qml");
            QQmlComponent current_comp_window(&switcher_engine, url);
            QObject* ObjectWindow =current_comp_window.create();
            account_doc = ObjectWindow;
        }
    }

    if(flag)
    {
        QSqlQuery queryFN("SELECT * FROM patient");
        while (queryFN.next())
        {
            QObject *value_first_name_patient = account_patient-
>findChild<QObject*>("value_first_name_patient");
            value_first_name_patient->setProperty("text",
queryFN.value(1).toString());

            QObject *value_second_name_patient = account_patient-
>findChild<QObject*>("value_second_name_patient");
            value_second_name_patient->setProperty("text",
queryFN.value(2).toString());

            QObject *value_middle_name_patient = account_patient-
>findChild<QObject*>("value_middle_name_patient");
            value_middle_name_patient->setProperty("text",
queryFN.value(3).toString());
```

```
            QObject *value_db_patient = account_patient-
>findChild<QObject*>("value_db_patient");
            value_db_patient->setProperty("text",
queryFN.value(4).toString());
        }

        QSqlQuery queryList("SELECT appointment.Time, doc.`First
Name`,doc.`Second Name` FROM patient INNER JOIN appointment
INNER JOIN doc WHERE patient.ID = appointment.`Patient ID`AND
doc.ID = appointment.`Doc ID`");
        while (queryList.next())
        {
            QObject *value_first_name_da = account_patient-
>findChild<QObject*>("value_name_doc");
            value_first_name_da->setProperty("text",
queryList.value(1).toString());
            qDebug() << value_first_name_da;

            QObject *value_second_name_da = account_patient-
>findChild<QObject*>("value_sname_doc");
            value_second_name_da->setProperty("text",
queryList.value(2).toString());

            QObject *value_time_app = account_patient-
>findChild<QObject*>("value_time");
            value_time_app->setProperty("text",
queryList.value(0).toString());
        }
      }
   }
}

void WindowSwitcherManager::AutorizationDocMode(QString loginQ,
QString passwordQ, QString id)
{
```

```
ModuleConnection conn;
bool flag = false;
if(conn.ConnectToMySQL())
{
   QSqlQuery query("SELECT Login, Pass, ID FROM doc");
   while (query.next())
   {
      QString login = query.value(0).toString();
      QString password =  query.value(1).toString();
      QString ID =  query.value(2).toString();

      if(login == loginQ && password ==passwordQ && ID == id)
      {
         qDebug() << "Sur";
         const QUrl url("qrc:/MIS/UI/AccountDoc.qml");
         QQmlComponent current_comp_window(&switcher_engine, url);
         QObject* ObjectWindow =current_comp_window.create();

         account_doc = ObjectWindow;
         flag = true;
      }
      else
      {
         qDebug() << "Error";
         conn.CloseMySQLConnection();
         const QUrl url("qrc:/MIS/UI/ErrorMessage.qml");
         QQmlComponent current_comp_window(&switcher_engine, url);
         QObject* ObjectWindow =current_comp_window.create();
         account_doc = ObjectWindow;
      }
   }

   if(flag)
   {
      QSqlQuery queryFN("SELECT * FROM doc");
```

```
        while (queryFN.next())
        {
            QObject *value_first_name_patient = account_doc-
>findChild<QObject*>("value_first_name_doc");
            value_first_name_patient->setProperty("text",
queryFN.value(1).toString());

            QObject *value_second_name_patient = account_doc-
>findChild<QObject*>("value_second_name_doc");
            value_second_name_patient->setProperty("text",
queryFN.value(2).toString());

            QObject *value_middle_name_patient = account_doc-
>findChild<QObject*>("value_middle_name_doc");
            value_middle_name_patient->setProperty("text",
queryFN.value(3).toString());

        }

        QSqlQuery queryDoc("SELECT appointment.Time, patient.Surname
FROM doc INNER JOIN appointment INNER JOIN patient WHERE
patient.ID = appointment.`Patient ID`");
        while (queryDoc.next())
        {
            QObject *value_time = account_doc-
>findChild<QObject*>("value_time_patient");
            value_time->setProperty("text", queryDoc.value(0).toString());

            QObject *value_second_name_patient = account_doc-
>findChild<QObject*>("value_middle_name_patient");
            value_second_name_patient->setProperty("text",
queryDoc.value(1).toString());
        }
    }
  }
```

```
}

WindowSwitcherManager *WindowSwitcherManager::CreateInstancePtr()
{
   if(Inst_ptr == nullptr)
   {
      Inst_ptr = new WindowSwitcherManager;

   }

   return Inst_ptr;
}

bool WindowSwitcherManager::IsValid()
{
   return (Inst_ptr != nullptr) ? true : false;
}

QObject* WindowSwitcherManager::CreateCurrentObjWindow(const
QUrl& urlWindow)
{
   QQmlComponent current_comp_window(&switcher_engine,
urlWindow);
   QObject* ObjectWindow =current_comp_window.create();
   window_obj =
QSharedPointer<QWindow>(qobject_cast<QWindow*>(ObjectWindow));
   return ObjectWindow;
}
```